Machine Learning

Andrea Asperti

Department of Computer Science, University of Bologna
Mura Anteo Zamboni 7, 40127, Bologna, ITALY
asperti@cs.unibo.it

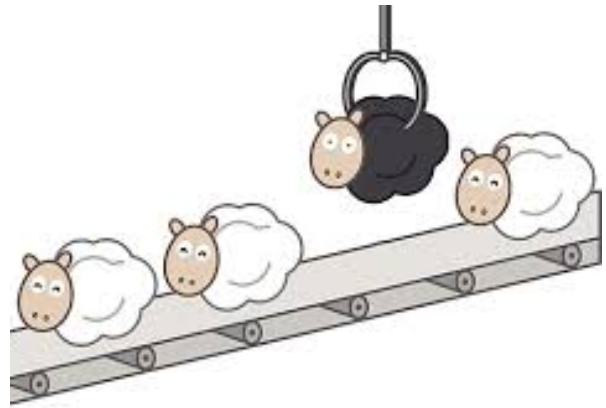
What Machine Learning is about

Target

Problems difficult to address by algorithmic means:

- classification problems

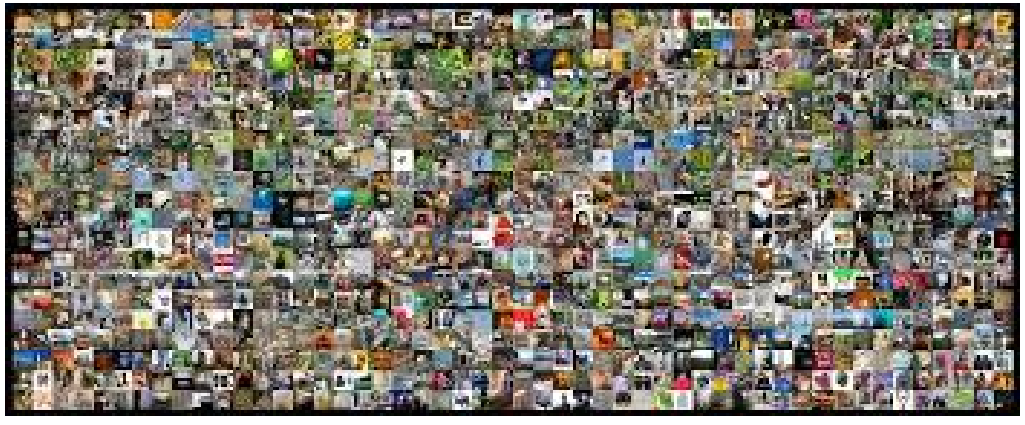
  spam detection, sentiment analysis, fraudulent transactions, ...
- image recognition
- speech/music recognition
- autoencoding
- data mining (e.g. clustering)
- ...



More and more problems addressed by Machine Learning techniques

Characteristics

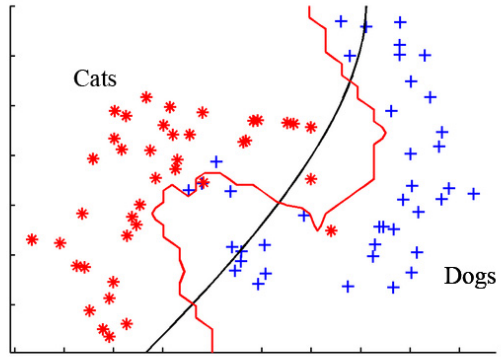
- low (metatheoretical) knowledge
- large number of input features
- big volume of training data
- adaptability



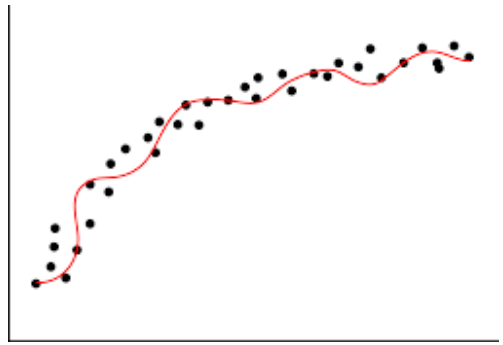
Imagenet database: $\approx$ 10 million images

The Machine Learning approach

- define a **model** for the task to be solved
  the model depends on a set of **parameters** Θ
- define a **performance metric**:
  some **error** measure to evaluate the model
- tune the parameters Θ to **minimize** the error on the **training set**



classification



regression

Machine learning is an optimization process

Why Learning?

Machine Learning problems are in fact **optimization problems**!
So, why talking about learning?

- The tuning of parameters is based on observations (**training set**). We learn from our past experience.

Why Learning?

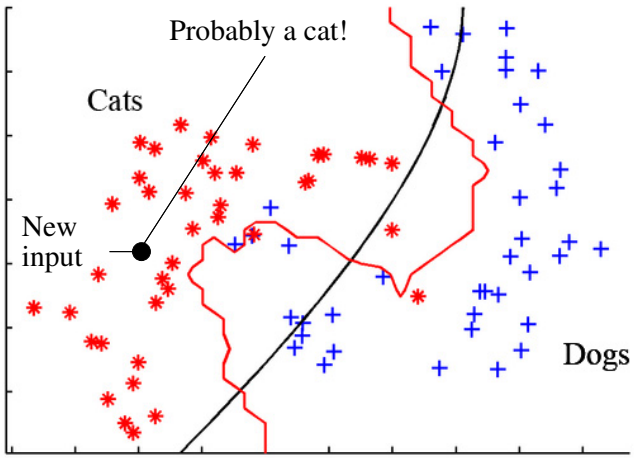
Machine Learning problems are in fact **optimization problems**!
So, why talking about learning?

- The tuning of parameters is based on observations (**training set**). We learn from our past experience.

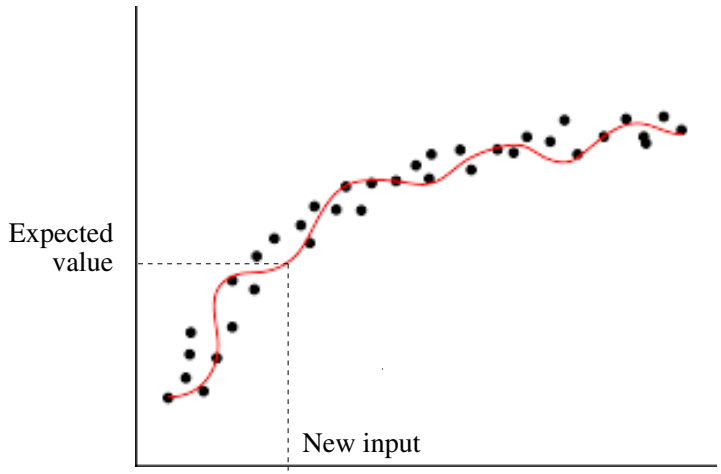
- the solution to the optimization problem is not given in an analytical form (often there is no closed form solution).

  We use **iterative** techniques (e.g. gradient descent) to progressively approximate the result, and this can be be understood as a form of learning process.

Goal: making predictions



classification

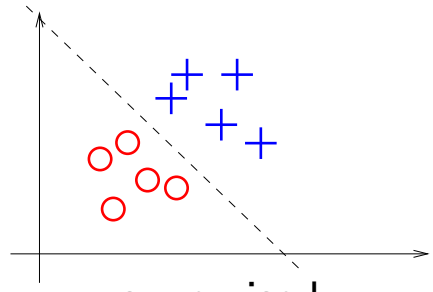


regression

Models can also be used to acquire knowledge on input data: finding clusters, making correlations, etc.

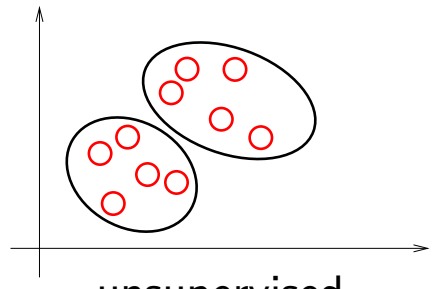
Different types of Learning Tasks

- **supervised learning**:
  inputs + outputs (labels)
  - classification
  - regression



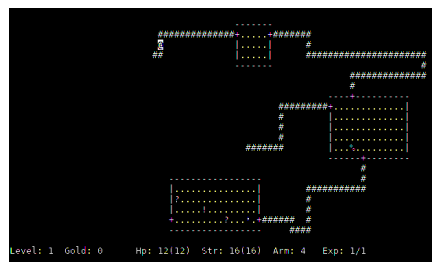
supervised

- **unsupervised learning**:
  just inputs
  - clustering
  - component analysis
  - autoencoding



unsupervised

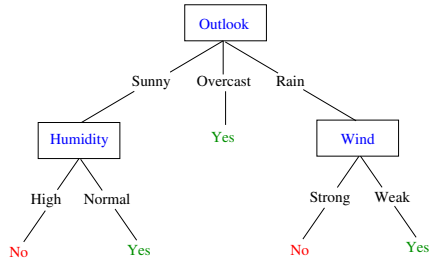
- **reinforcement learning**
  actions and rewards
  - learning long-term gains
  - "model-free" planning



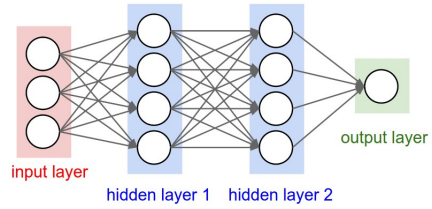
reinforcement

Many different techniques

- **Different ways to define the models**:
  - decision trees
  - mathematical expressions
  - neural networks
  - ...

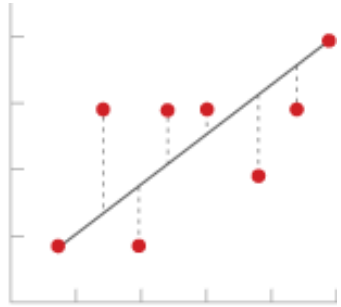


decision tree

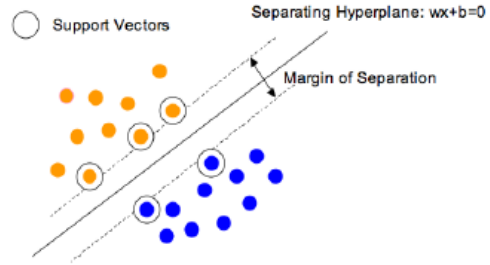


neural net

- **Different error (loss) functions**:
  - mean squared errors
  - logistic loss
  - cross entropy
  - cosine distance
  - maximum margin
  - ...



mean squared errors



maximum margin

Features

Features

Any information relative to a datum that describes some of its relevant properties is called a feature.

Features are the input of the learning process.

Learning is highly sensible to the choice of features.

Choosing good features is difficult (requires good domain knowledge).

Example of features

medical diagnosis	user profiling	weather forecasting
symptoms	demographic data	temperature
patient condition	personal interests	humidity
medical record	social communities	pressure
result of exams	life style	rain, wind, ...
...	...	...

Features in image processing

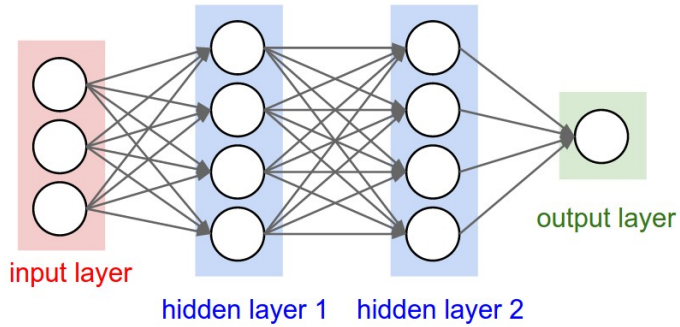
Picture taken from the OpenCv python-tutorial

Deep learning

Old approach: compute by hand good features, and apply a simple and well understood (e.g. linear) learning algorithm.

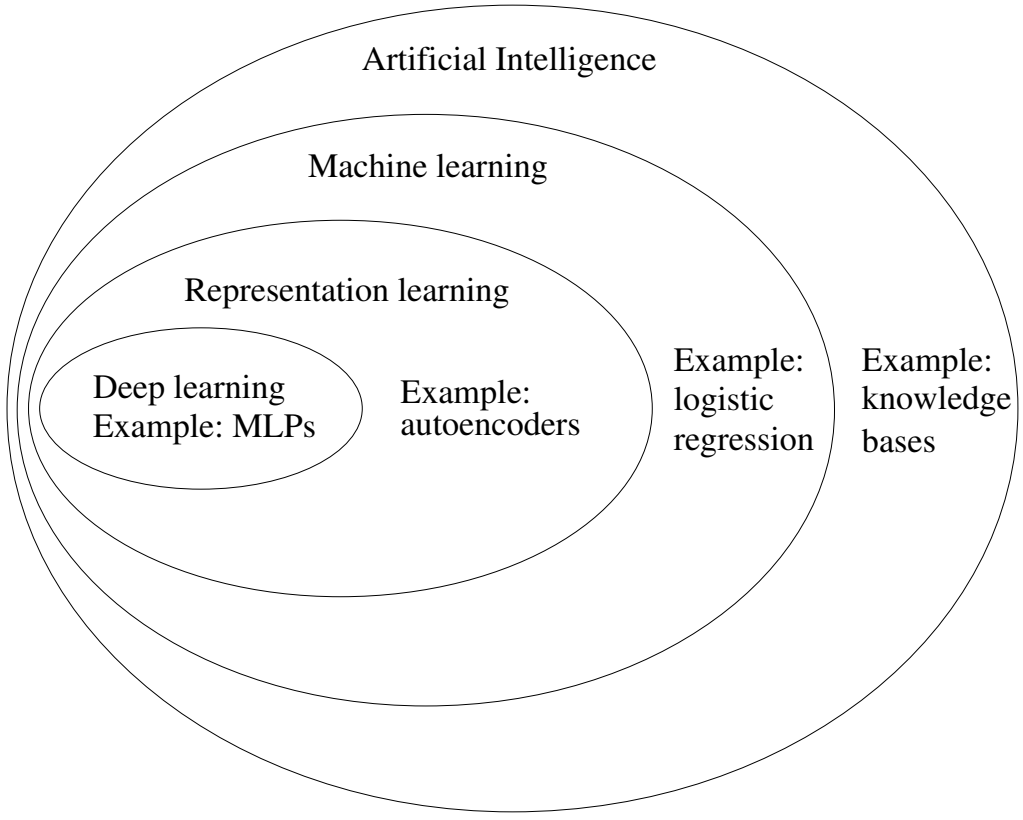
Modern (deep) approach: supply raw data and let to the machine the burden to synthesize good features (internal representations).

Deep learning is implemented through neural networks (NN)
A NN is deep when we have multiple hidden layers:



each hidden layer computes new features from previous ones.

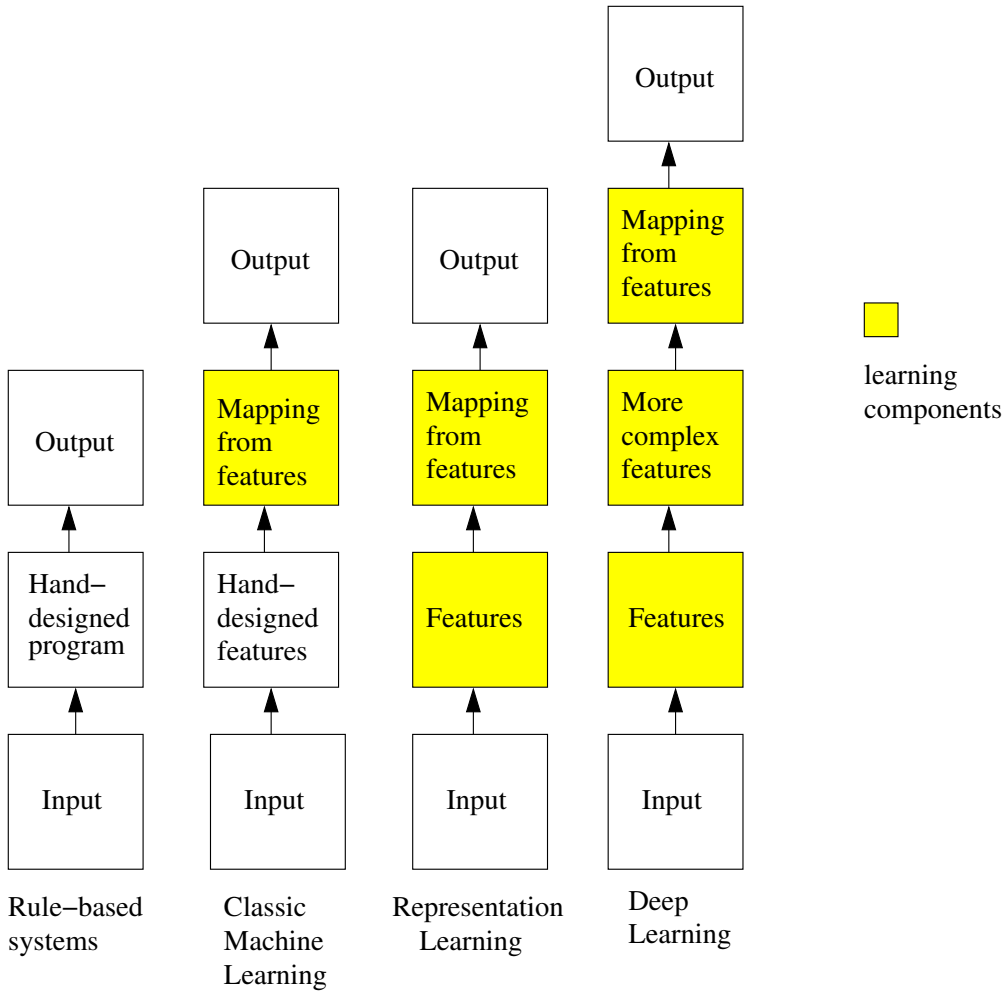
Relations between research areas



From "Deep Learning" by Y.Bengio, I.Goodfellow, A.Courville, MIT Press.

For a more historical perspective on the different fields consult my blog.

Components trained to learn



From "Deep Learning" by Y.Bengio, I.Goodfellow, A.Courville, MIT Press.

AI, machine learning, deep learning

- **Knowledge-based systems**: take an expert, ask him how he solves a problem and try to mimic his approach by means of logical rules

AI, machine learning, deep learning

- **Knowledge-based systems**: take an expert, ask him how he solves a problem and try to mimic his approach by means of logical rules
- **Traditional Machine-Learning**: take an expert, ask him what are the features of data relevant to solve a given problem, and let the machine learn the mapping

AI, machine learning, deep learning

- **Knowledge-based systems**: take an expert, ask him how he solves a problem and try to mimic his approach by means of logical rules
- **Traditional Machine-Learning**: take an expert, ask him what are the features of data relevant to solve a given problem, and let the machine learn the mapping
- **Deep-Learning**: get rid of the expert

Didactic Information

- structure of the course
- books, tutorials and blogs
- software
- examination
- office hours

Structure of the course

Part I: Machine Learning

- Decision Trees
- Probability basics
- Naif Bayes
- Maximum Likelihood estimation
- Logistic regression
- The gradient technique
- Generative vs Discriminative
- Multinomial regression
- Linear regression

Part II: Deep Learning

- Neural networks
- Backpropagation and Training
- Convolutions
- Convolutional Networks
- Autoencoders
- Generative Adversarial Networks
- Object detection and segmentation
- Recurrent Networks
- Attention & Transformers

Books

(Shallow) Machine Learning

- C.M.Bishop. Pattern Recognition and Machine Learning. Springer 2006.
- E. Alpaydin, Introduction to Machine Learning, Cambridge University Press, 2010.

Deep Learning

- Y.Bengio, I.Goodfellow and A.Courville. Deep Learning, MIT Press to appear.
- Dive into deep learning (D2L)

Online Tutorials and Blogs

Pssible to study on online material (fast updating):

- Tensorflow tutorials
- Deep Learning Tutorial. LISA lab. University of Montreal.
- Deep Mind blog
- Open AI blog
- Keras blog
- towardsdatascience
- Machine learning tutorial with Python
- a lot of interesting lessons and seminars on youtube
- a lot of material on github
- ...

Suggested software

Machine learning "legacy" techniques:
- Scikit-learn

Neural Networks and Deep Learning:
- TensorFlow, Google Brain
- Keras, F.Chollet
- PyTorch, Facebook
- MXNET, Apache
- ...

Image and signal processing:
- OpenCV
- Scipy

Assessment

- individual test (40%)
- individual project (60%)

The topic of the project will be given to you (on virtuale) 7-8 days before the scheluded date of the exam, and you need to submit your solution (a single python notebook) within the specified deadline. Typically, you have a week time to complete the task.

The topic is different for each session.

The individual test, once passed, does not need to be repeated.

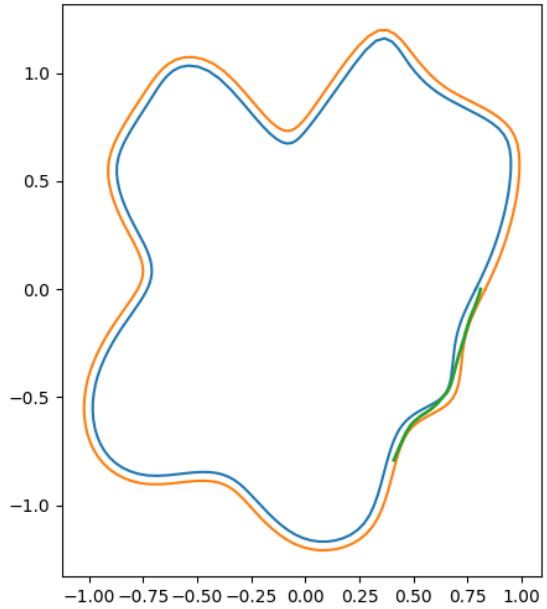
Extra credits: MicroRacer

Organize and participate in the MicroRacer annual Championship.

MicroRacer is a small didactic environment for testing DRL techniques in a racing setting, presented at the 8th International Conference on Machine Learning, Optimization, and Data Science. Here is the article.

- random track generation
- fast training (15-30 mn)
- baselines provided

- 1 point for participating
- 1 extra point for second place
- 2 extra points for first place



Office hours and links

Wednesday, 11-13

Prof.Andrea Asperti
andrea.asperti@unibo.it
Via Malaguti 1D

- homepage:https://www.unibo.it/sitoweb/andrea.asperti
- my own deeplearningblog
- deepfridays seminars

Decision Trees

Function Approximation

Train set: a set of training examples

$$\langle x^{(i)}, y^{(i)} \rangle$$

where

- $x^{(i)} \in X$ (set of inputs)
- $y^{(i)} \in Y$ (set of outputs)
- i is the instance of the training sample

Problem: "learn" the function mapping $x^{(i)}$ to $y^{(i)}$

Y discrete: classification problem (class prediction)
Y continuous: regression problem (value prediction).

Hypothesis Space

Machine learning techniques require a commitment to a given function space H, inside which we look for the function that provides the best approximation for the training set.
H reflects the way we are modeling data.

Example: Training set $= \{\langle 2, 3\rangle, \langle 3, 4\rangle\}$

Hypothesis Space

Machine learning techniques require a commitment to a given function space H, inside which we look for the function that provides the best approximation for the training set.
H reflects the way we are modeling data.

Example: Training set $= \{\langle 2, 3\rangle, \langle 3, 4\rangle\}$

$H =$ linear functions. We have an exact solution: $y = x + 1$

Hypothesis Space

Machine learning techniques require a commitment to a given function space H, inside which we look for the function that provides the best approximation for the training set.
H reflects the way we are modeling data.

Example: Training set $= \{\langle 2, 3\rangle, \langle 3, 4\rangle\}$

$H =$ linear functions. We have an exact solution: $y = x + 1$

Adding the instance $\langle 1, 0\rangle$, the model is not exact any more.

- we keep the model and content ourselves with approximate answers
- we change the model, for instance taking quadratic functions $y = -x^2 + 6x - 5$

Overfitting and underfitting

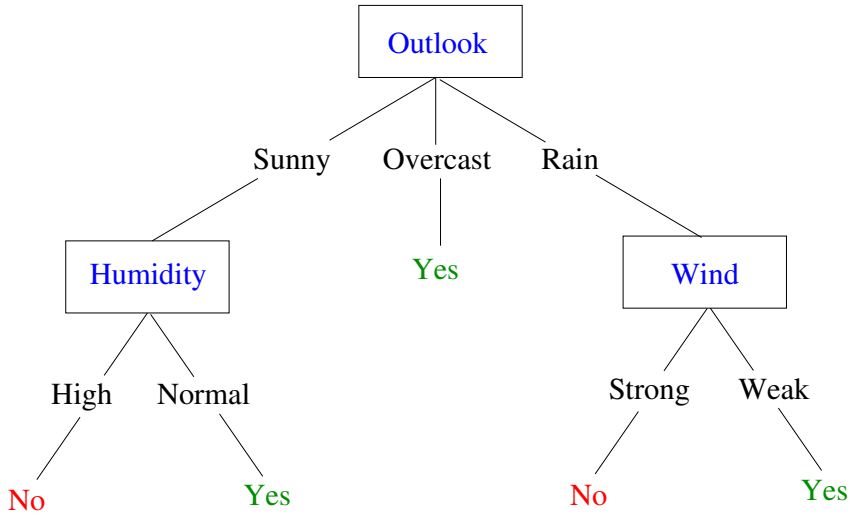
overfitting the model is too complex and specialized over the peculiarities of the instances of the training set

underfitting the model is too simple and does not allow to express the complexity of the observations.

Decision trees: an example

A good day to play tennis?

F : Outlook $\times$ Humidity $\times$ Wind $\times$ Temp $\rightarrow$ Play Tennis?



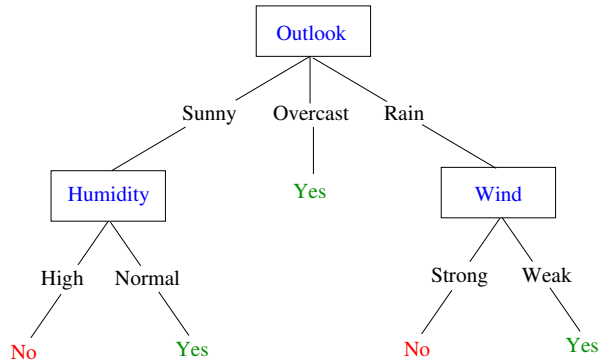
Every node tests an attribute (feature) X

Each branch corresponds to one of the possible discrete values of X

Every leaf predicts the answer Y (or a probability $P(Y|X)$)

Learn to build a decision tree

Problem configuration



- Input set X
  every instance $x \in X$ is a vector of features of the following kind:
  $\langle$ Humidity=high, Wind=weak, Outlook=rain, Temp=hot $\rangle$
- Target function $f : X \to Y$
  Y takes discrete values (booleans)
- Hypothesis Space $H = \{h \mid h : X \to Y\}$ (no restriction)
  - we try to model each $h \in H$ with a decision tree
  - each instance x defines a path in the tree leading to a leaf labelled with y

Play-tennis training set

Outlook	Temp	Humidity	Wind	Play
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Input and output of the learning task

Input A set of training samples $\{\langle x^{(i)}, y^{(i)} \rangle\}$ relative to the target function f

Output The hypothesis $h \in H$ that best approximates f

But f is unknown ... so?

Input and output of the learning task

Input A set of training samples $\{\langle x^{(i)}, y^{(i)} \rangle\}$ relative to the target function f

Output The hypothesis $h \in H$ that best approximates f

But f is unknown ... so?

We try to approximate the training set

Expressiveness of the model

Let $X = X_1 \times X_2 \cdots \times X_n$ where $X_i = \{\text{True,False}\}$

Can we represent $Y = X_2 \wedge X_5$? and $Y = X_4 \vee X_1$?

Can we represent $Y = X_2 \wedge X_5 \vee (\neg X_3) \wedge X_4 \wedge X_1$?

Expressiveness of the model (2)

Important questions

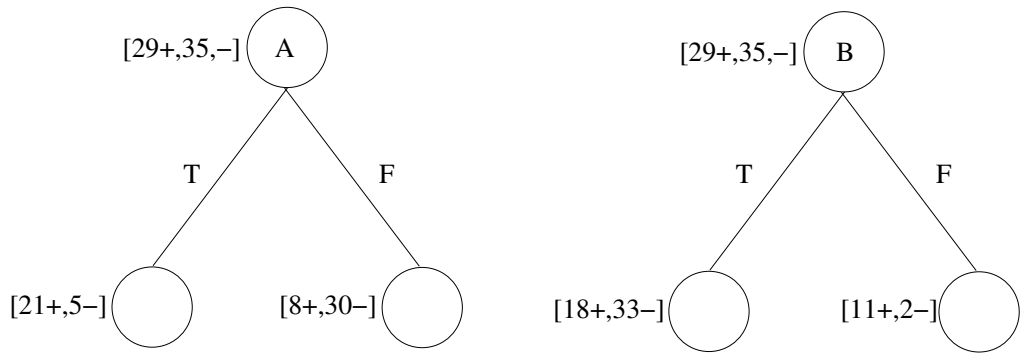
- do we have a decision tree for each h in the space hypothesis?
- if the tree exists, is it unique?
- if it is not unique, do we have a preference?

Top-down inductive construction

Main loop:

1. assign to the current node the "best" attribute X_i
2. create a child node for every possible value of X_i
3. for every child node, if all the examples in the training set associated with the node have a same label y, mark the node (leaf) with label y, otherwise iterate from point 1

What is the "best" attribute?



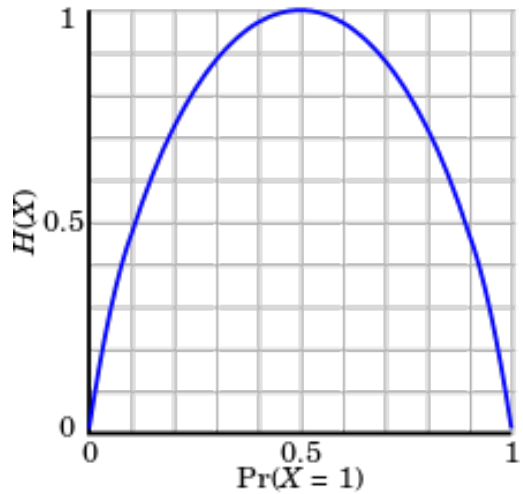
Entropy

The entropy $H(X)$ of a random variable X is

$$H(X) = -\sum_{i=1}^{n} P(X = i) log_2 P(X = i)$$

where n is the number of possible values of X.

Entropy measures the degree of impurity of the information. It is maximal when X is uniformly distributed over all values, and minimal (0) when it is concentrated on a single value.



Information Theory (Shannon)

Entropy is the average amount of information produced by a stochastic source of data.

Information is associated with the *probability* of each data (the "surprise" carried by the event):

- an event with probability 1 carries no information: $I(1) = 0$
- given two independent events with probabilities p_1 and p_2 their joint probability is $p_1 p_2$ but the information acquired is the sum of the informations of the two independent events, so

$$I(p_1 p_2) = I(p_1) + I(p_2)$$

It is hence natural to define

$$I(p) = -log(p)$$

Code Theory (Shannon)

Entropy also measures the average number of bits required to transmit outcomes produced by stochastic process X.

Suppose to have n events with the same probability. How many bits do you need to encode each possible outcome?

Code Theory (Shannon)

Entropy also measures the average number of bits required to transmit outcomes produced by stochastic process X.

Suppose to have n events with the same probability. How many bits do you need to encode each possible outcome?

$$log(n)$$

Code Theory (Shannon)

Entropy also measures the average number of bits required to transmit outcomes produced by stochastic process X.

Suppose to have n events with the same probability. How many bits do you need to encode each possible outcome?

$$log(n)$$

In this case,

$$\begin{aligned} H(X) &= -\sum_{i=1}^{n} P(X=i) log_2 P(X=i) \\ &= -\sum_{i=1}^{n} 1/n \, log_2(1/n) \\ &= log(n) \end{aligned}$$

Code Theory (Shannon)

Entropy also measures the average number of bits required to transmit outcomes produced by stochastic process X.

Suppose to have n events with the same probability. How many bits do you need to encode each possible outcome?

$$log(n)$$

In this case,

$$\begin{aligned} H(X) &= -\sum_{i=1}^{n} P(X=i) log_2 P(X=i) \\ &= -\sum_{i=1}^{n} 1/n \, log_2(1/n) \\ &= log(n) \end{aligned}$$

If events are not equiprobable we can do better!!!

Information gain

Entropy of X

$$H(X) = -\sum_{i=1}^{n} P(X = i) log_2 P(X = i)$$

Conditional Entropy of X given a specific $Y = v$

$$H(X|Y = v) = -\sum_{i=1}^{n} P(X = i|Y = v) log_2 P(X = i|Y = v)$$

Conditional Entropy of X given Y
(weighted average over all m possible values of Y)

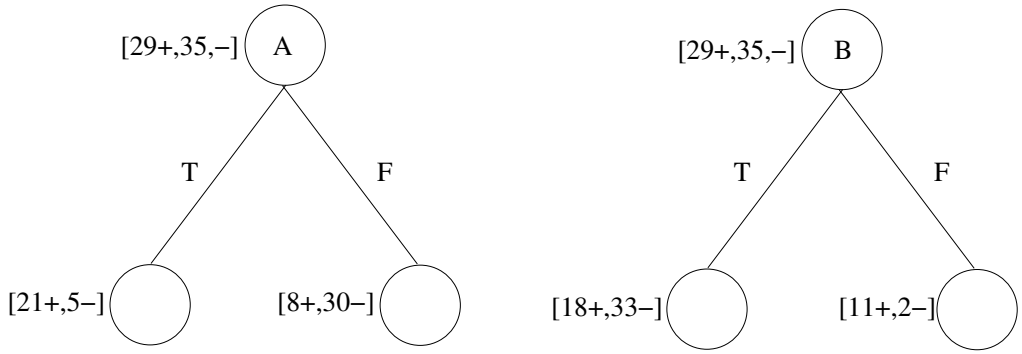
$$H(X|Y) = \sum_{v=1}^{m} P(Y = v) H(X|Y = v)$$

Information Gain between X and Y:

$$I(X, Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

Better A or B?

Let us measure the entropy reduction of the target variable Y due to some attribute X, that is the information gain $I(Y, X)$ between Y and X.



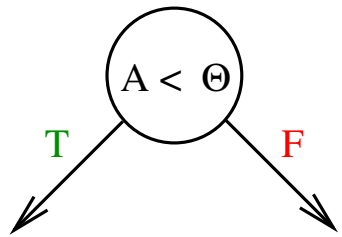
$H(Y) = -(29/64) \cdot log_2(29/64) - (35/64) \cdot log_2(35/64) = .994$
$H(Y|A = T) = -(21/26) \cdot log_2(21/26) - (5/26) \cdot log_2(5/26) = .706$
$H(Y|A = F) = -(8/38) \cdot log_2(8/38) - (30/38) \cdot log_2(30/38) = .742$
$H(Y|A) = .706 \cdot 26/64 + .742 \cdot 38/64 = .726$
$I(Y, A) = H(Y) - H(Y|A) = .994 - .726 = .288$

For B we get $H(Y|B) = .872$ and $I(Y, B) = .122$

So A is better!

The continuous case

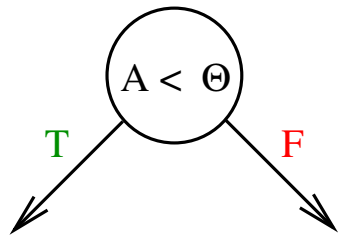
When attributes are continuous we take decisions based on thresholds:



- we compare thresholds with information gain
- how to choose candidate thresholds?

The continuous case

When attributes are continuous we take decisions based on thresholds:

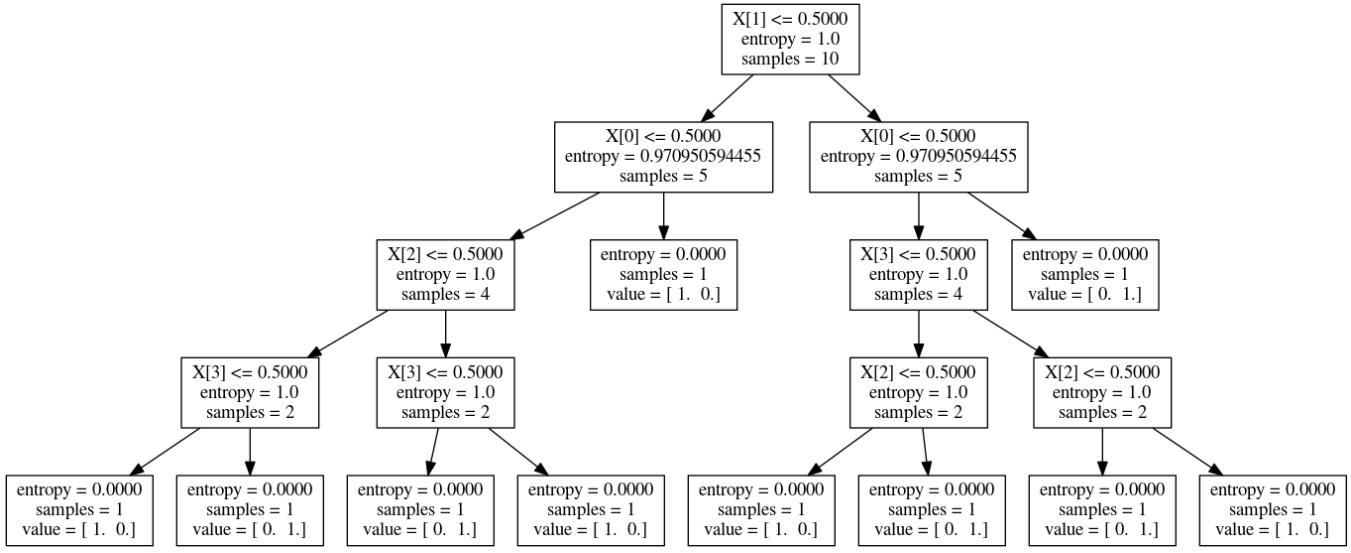


- we compare thresholds with information gain
- how to choose candidate thresholds?
  - sample at discrete intervals
  - order the test set w.r.t. the given attribute and choose thresholds at the average of two consecutive data

An example

```
from sklearn import tree
X = [[0,0,0,0], [1,0,0,0], [0,0,0,1], [0,0,1,0], [0,0,1,1],
     [0,1,0,0], [0,1,0,1], [1,1,0,1], [0,1,1,0], [0,1,1,1]]
Y = [0,0,1,1,0,0,1,1,1,0]
#label is 1 if X[2]==X[3], 0 otherwise

clf = tree.DecisionTreeClassifier(criterion='entropy',
                                  random_state=0)
clf = clf.fit(X, Y)

print(clf.predict([1,1,1,1]))   #> [1]
print(clf.feature_importances_) #> [0.17, 0.029, 0.4, 0.4]

tree.export_graphviz(clf, out_file='tree.dot')
#dot -T png tree.dot -o tree.png
```

Example: the decision tree



In this case, the information gain of individual features is not a good selection policy!

Still, importance is fair, since it is calculated ex post for a single decision tree by the amount that each attribute split point improves the performance measure, weighted by the number of observations the node is responsible for.

Next arguments

- No free lunch
- Inductive Bias

Function approximation problem

Problem Configuration:

- a set of instances X
  every instance $x \in X$ is a vector of features (attributes)
  $$x = \langle x_1, x_2, \ldots x_n \rangle$$
- Target function $f : X \to Y$, where Y takes discrete values
- Hypothesis Space: $H = \{h \mid h : X \to Y\}$

Input (of the training): A training set of $\{\langle x^{(i)}, y^{(i)} \rangle\}$ relative to the target function f

Output (of the training): The hypothesis $h \in H$ that better approximate (?!?) the target function f (not known)

Example: guessing the next element

$$2, 3, 5,$$

Example: guessing the next element

$$2, 3, 5, 9,$$

Example: guessing the next element

$$2, 3, 5, 9, 17, \ldots$$

Example: guessing the next element

$$2, 3, 5, 9, 17, \ldots$$

Not a single solution.
Why one should be better than others?

Example: guessing the next element

$$2, 3, 5, 9, 17, \ldots$$

Not a single solution.
Why one should be better than others?

To learn, we need an inductive bias!

A more complex example:
(just for fun)

1. 1
2. 11
3. 21
4. 1211
5. 111221
6. 312211
7. 13112221
8. ???

Example: guessing the next element

$$2, 3, 5, 9, 17, \ldots$$

Not a single solution.
Why one should be better than others?

To learn, we need an inductive bias!

A more complex example:
(just for fun)

1. 1
2. 11
3. 21
4. 1211
5. 111221
6. 312211
7. 13112221
8. ???

Exercise: what would a decision tree predict?

Learning: search for the best hypothesis

Learning can be understood as a search in the Space Hypothesis for the function that better fits the training set (according to some error measure).

Example: input described by 20 boolean features; boolean output.

- What is the dimension of the Space Hypothesis?
- How many training instances are need to uniquely determine the target function f?
- If we have m training instances, how many distinct functions satisfy them?
- Given a new instance, how many of the previous ones will satisfy it?

No Free Lunch (NFL)

Supposing that all hypothesis have the same probability, there is no reason to prefer one to the other.

Hume

Even after the observation of the frequent or constant conjunction of objects, we have no reason to draw any inference concerning any object beyond those of which we have had experience.

The choice of the hypothesis (or model) must be driven by inductive biases, otherwise no learning is possible.

The learning biases are the set of assumptions that the learner uses to predict outputs for new inputs.

An example: Occam's razor

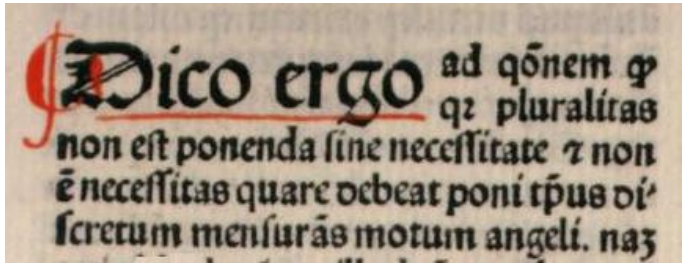
A well known example of inductive bias is the so called Occam's razor:

"*The simplest solution consistent with observation is to be preferred*" (lex parsimoniae).

Never explicitly stated in Occam's works

Similar concepts expressed by Duns Scoto and Thomas Aquinas:

"*Plurality is not to be posited without necessity*"



Occam's razor is e.g. used by decision trees.

Other examples of inductive biases

- **conditional independence** if hypothesis are expressed in a Bayesian framework (see later), we assume the conditional independence of the input random variables. Bias used by Naïve Bayes classifiers.
- **Maximum margin** when drawing a discrimination line between two classes, prefer the one that maximize the width of the margin between elements of the classes. Bias used by Support Vector Machines.
- **Features importance** compare features (e.g. via information gain) and give priority to the most useful ones. Bias used in Decision Tree learning and several algorithms based on feature selection.
- **Nearest neighbours** we assume that all data in a small neighbourhood of a given data belongs to a same class. Bias used in k-nearest neighbour algorithm.

No Free Lunch - discussion

The NFL theorem is based on the assumption that any possible extension of the target function beyond the training samples remains equiprobable.

Learning can only be done on a known training dataset.
We presume the training set is a significant sample of some target distribution we are trying to learn.

- discriminative approaches try to learn the probability of an output given the input, that is the conditional distribution $P(Y|X)$
- generative approaches try to learn the joint input-output distribution $P(X, Y)$

Overfitting

Overfitting

Let us consider the error of the hypothesis h

- on the training set, $error_{train}(h)$
- on the full data set $\mathcal{D}$, $error_{\mathcal{D}}(h)$

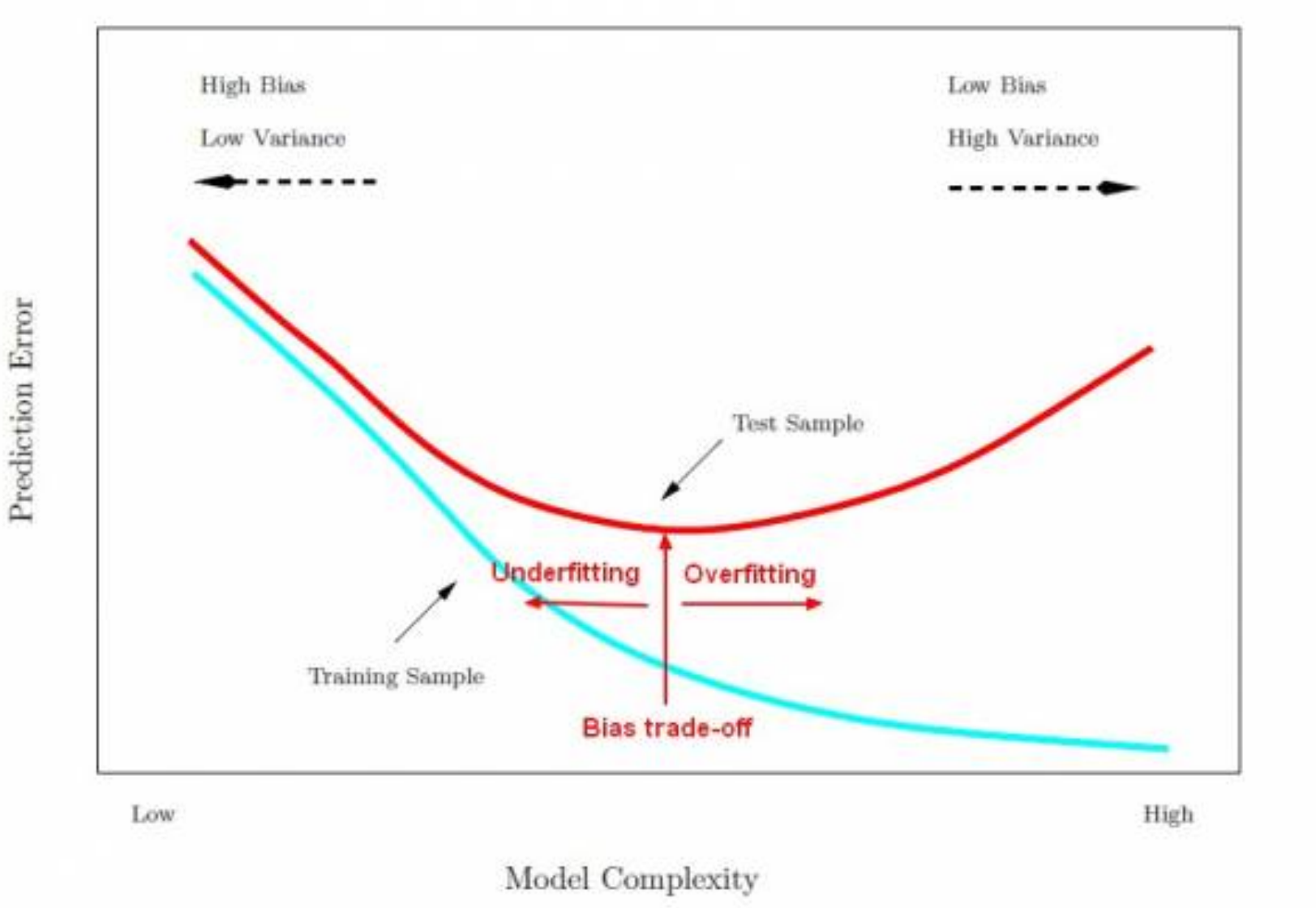
We say that h overfits the training set if there exists another hypothesis h' such that

$$error_{train}(h) < error_{train}(h')$$

but

$$error_{\mathcal{D}}(h) > error_{\mathcal{D}}(h')$$

Overfitting and model complexity



How to check and avoid overfitting

Problem: we do not know $\mathcal{D}$!

Divide the available data in two disjoint sets:

training set to be used to choose a candidate h

validation set to be used to asses the accuracy of h

Avoid overfitting with decision trees

- early stopping: terminate the construction of the tree as soon as the classification improvement is not statistically significative (e.g. the information gain is below some threshold, and/or we do not have sufficient data)
- post-pruning: develop the full tree and then proceed to backward prune it

reduce-error pruning

Build a full and exact decision tree for the training set.

Repeat the following operation until further pruning do not improve accuracy:

1. for any subtree, compute the impact of its removal on the classification accuracy on the validation set
2. greedily perform pruning of the subtree that optimizes accuracy

Variants: Gini's impurity

Gini's impurity measures the probability that a generic element get misclassified according to the current classification (an alternative to entropy).

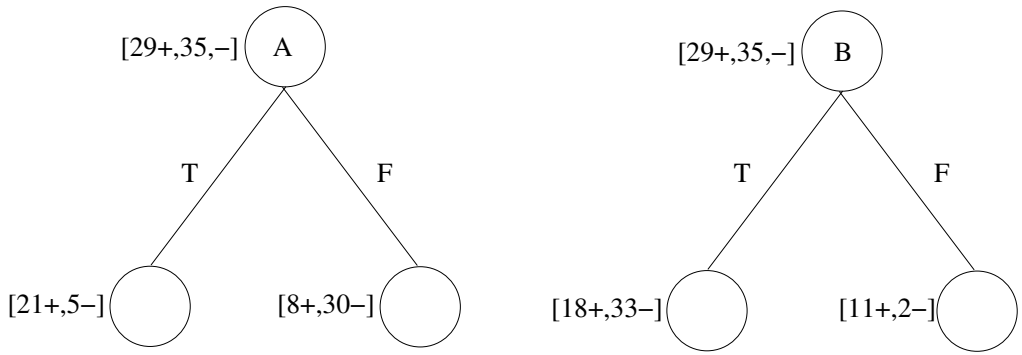
Given m categories, let f_i be the fraction of data with label i. This is equal to the probability that an input belongs to the category i. The probability of misclassifying it is hence $1 - f_i$, and its weighted average on all categories is just Gini's impurity, that is

$$I_G(f) = \sum_{i=1}^{m} f_i(1 - f_i) = \sum_{i=1}^{m} (f_i - f_i^2) = \sum_{i=1}^{m} f_i - \sum_{i=1}^{m} f_i^2 = 1 - \sum_{i=1}^{m} f_i^2$$

This metric is applied to every child node, and values are summed in a weighted way (similarly to the definition of Information-gain) to get a measure of the quality of an attribute.

Better A or B?

Let us evaluate the split using Gini's impurity.



For the attribute A:
$I_G(A = T) = 1 - (21/26)^2 - (5/26)^2 = .310$
$I_G(A = F) = 1 - (8/38)^2 - (30/38)^2 = .332$
$I_G(A) = .310 \cdot 26/64 + .332 \cdot 38/64 = .323$

For the attribute B:
$I_G(B = T) = 1 - (18/51)^2 - (33/51)^2 = .456$
$I_G(B = F) = 1 - (11/13)^2 - (2/13)^2 = .260$
$I_G(B) = .456 \cdot 51/64 + .260 \cdot 13/64 = .416$

Hence, A is better (lower impurity)

Positive aspects of decision trees

- easy to understand: simple logical rules, trees can be visualized
- little or no data preprocessing is required
- very low prediction cost
- can be used with both discrete and continuous features

Negative aspects of decision trees

- high risk of overfitting
- selection of attributes quite unstable
- easy to build strongly unbalanced trees, especially if a class is dominant It can be useful to pre-balance the dataset.

Decision Trees on Scikit-learn

Look at the following page

```
>>> from skelearn import tree
>>> X = [[0,0],[1,1]]
>>> Y [[0,1]]
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X,Y)
```

- any method in the first part of course can be tested that easily
- you are **warmly** invited to play with them
- do not be too naif in your project

Random Forests

Decision Trees are normally used as components in Random Forests, operating as an ensemble.

Ensemble techniques exploits the principle that a large number of relatively uncorrelated models (e.g. trees) operating as a committee will typically outperform any of the individual constituent models.

Ensure differentiation by:

- Bagging: feeding different, randomly choosen sets of input data
- Feature Randomness: building trees from random subsets of features

Summary of main notions learned

- **Approximation problem**:
  - Features X, labels (categories) Y
  - Training set $\{\langle x^{(i)}, y^{(i)} \rangle\}$
  - Hypothesis Space $H = \{h \mid h : X \to Y\}$

- **Learning = search/optimization in H**
  - Different objectives are possible
    - minimize the error on the training set (0-1 loss)
    - among candidate solutions look for the "simpler" one (occam razor's)
    - ...
  - No learning is possible without an inductive bias (NFL theorem)

Probability basics

Probability basics

- **Events**
  - discrete and continuous random variable
- **Axioms of Probability Theory**
  - a reasonable theory of uncertainty
- **Independent events**
- **Conditional Probability**
- **Bayes Rule**
- **Joint Probability Distribution**
- **Expectation**
- **Independence, conditional Independence**

Random variable

A random variable X denotes an outcome about which we are uncertain, for instance the result of a randomized experiment

Examples

- $X = true$ if a (randomly drawn) student in this room is male
- $X =$ first name of the student
- $X = true$ if two randomly drawn students in this room have the same birthday

We define $P(X)$ (probability of X) as the fraction of times X is true, in repeated runs of the same experiment.

More formally

- the set Ω of possible outcomes of a random experiment is called sample space
  - e.g. the set of students in this room
- a random variable is a measurable function over Ω:
  - discrete: e.g. gender: $\Omega \rightarrow \{m, f\}$
  - continuous: e.g. height: $\Omega \rightarrow \mathcal{R}$
- an event is an arbitrary subset of Ω
  - $\{x \in \Omega | gender(x) = m\}$
  - $\{x \in \Omega | height(x) \leq 175cm\}$
- we are interested in probabilities of specific events
  - $P(gender = m)$
- and in their probabilities conditioned on other events
  - $P(gender = m | height \leq 175)$

Sample space

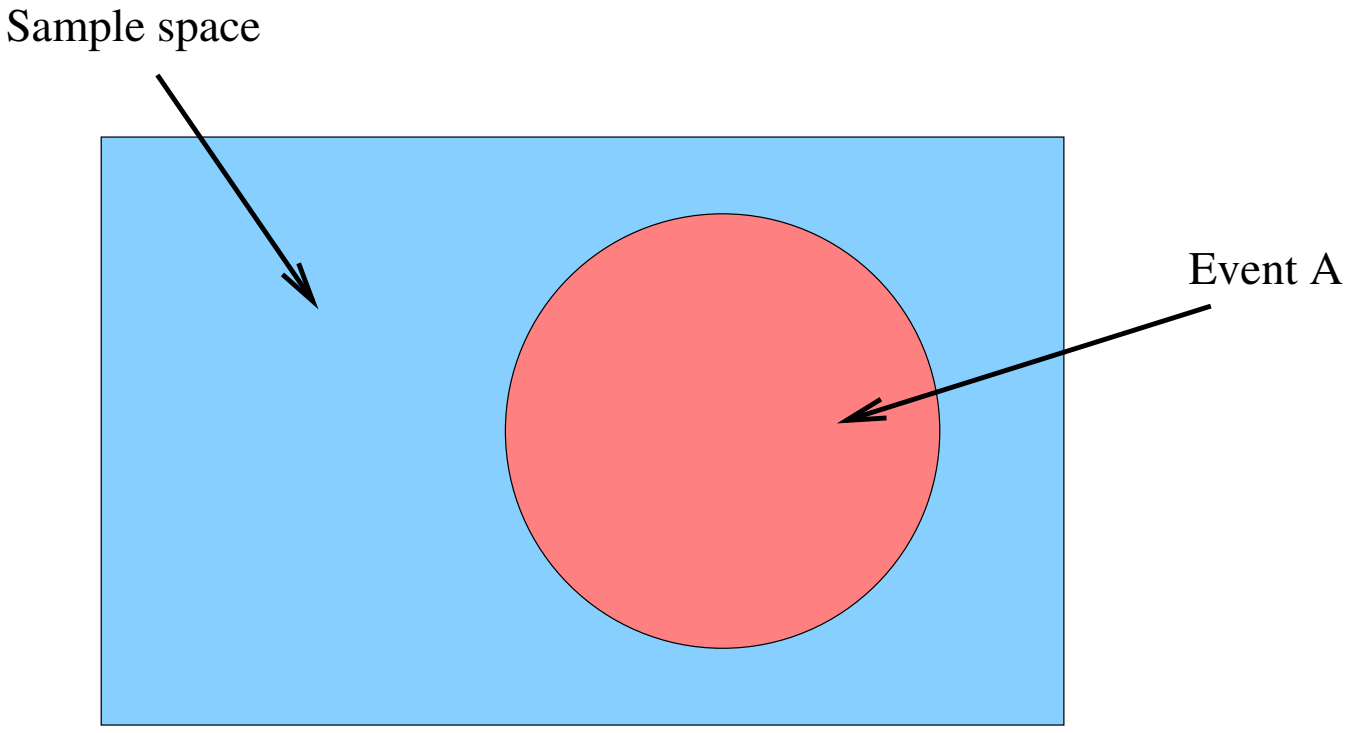
The definition of the sample space requires some caution.

Our probability intuition relies on the assumption that all samples in the space have the same probability.

Example: toss of a pair of dices.
The outcome is an integer number between 2 and 12.
However, not all of them have the same probability.

Better to consider the pairs of outcomes of each dice as elements of the sample space, and a random variable X expressing their sum.

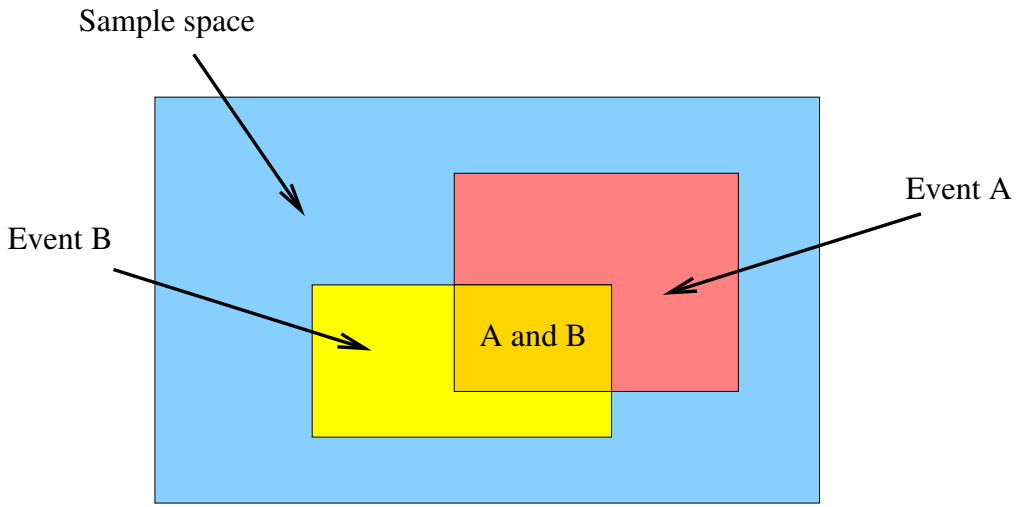
Graphical Visualization



The probability $P(A)$ of the event A is the ratio between the area of A and the area of the full sample space.

Probability Axioms

- $0 <= P(A) <= 1$
- $P(True) = 1$
- $P(False) = 0$
- $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$



Derived theorems

$$P(\neg A) = 1 - P(A)$$

Proof: We know that

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$

and in particular

$$P(A \vee \neg A) = P(A) + P(\neg A) - P(A \wedge \neg A)$$

but

$$P(A \vee \neg A) = P(\mathit{True}) = 1 \quad \mathit{and} \quad P(A \wedge \neg A) = P(\mathit{False}) = 0$$

so

$$1 = P(A) + P(\neg A) - 0$$

q.e.d.

Another interesting theorem

$$P(A) = P(A \wedge B) + P(A \wedge \neg B)$$

Proof: We know that

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$

Since

$$A = A \wedge (B \vee \neg B) = (A \wedge B) \vee (A \wedge \neg B)$$

we have:

$$\begin{aligned} P(A) = & \quad P(A \wedge B) + P(A \wedge \neg B) - P((A \wedge B) \wedge (A \wedge \neg B)) \\ = & \quad P(A \wedge B) + P(A \wedge \neg B) - P(\mathit{False}) \\ = & \quad P(A \wedge B) + P(A \wedge \neg B) \end{aligned}$$

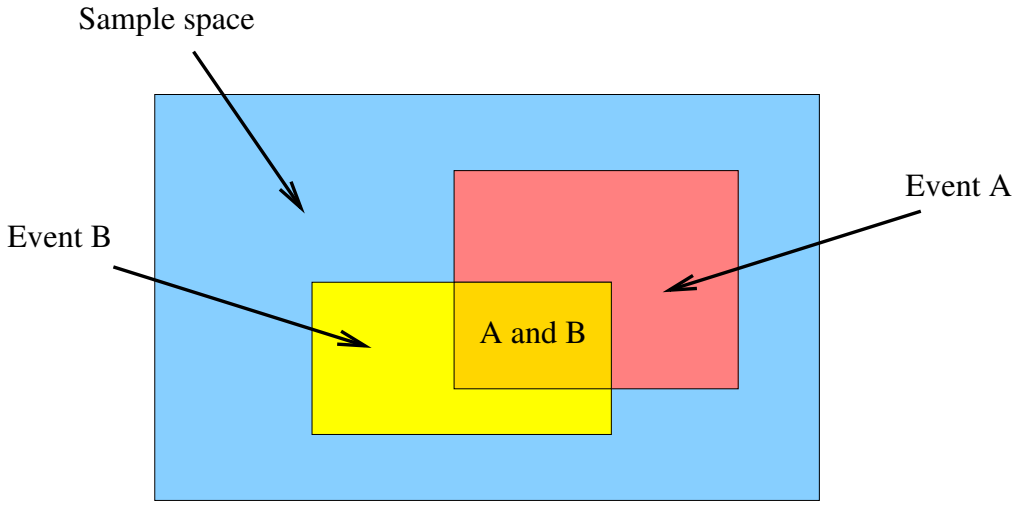
Multivalued discrete random variables

A is a k-valued discrete random variable if it may take exactly one of the values in the set $\{\nu_1, \nu_2, \ldots \nu_k\}$.

$P(A = \nu_i)$ is the event of all samples where A takes value ν_i.

$P(A = \nu_i \wedge A = \nu_j) = 0$ if $i \neq j$
$P(A = \nu_1 \vee A = \nu_2 \vee \ldots A = \nu_k) = 1$

Conditional Probability



Definition The conditional probability of the event A given the event B is defined as the quantity

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

Corollary: Chain rule

$$P(A \wedge B) = P(B) \cdot P(A|B) = P(A) \cdot P(B|A)$$

Independent events

Two events A and B are independent when

$$P(A|B) = P(A)$$

That is, the event B has no influence over A.
As a corollary,

$$P(A \wedge B) = P(A) \cdot P(B)$$

Moreover, since

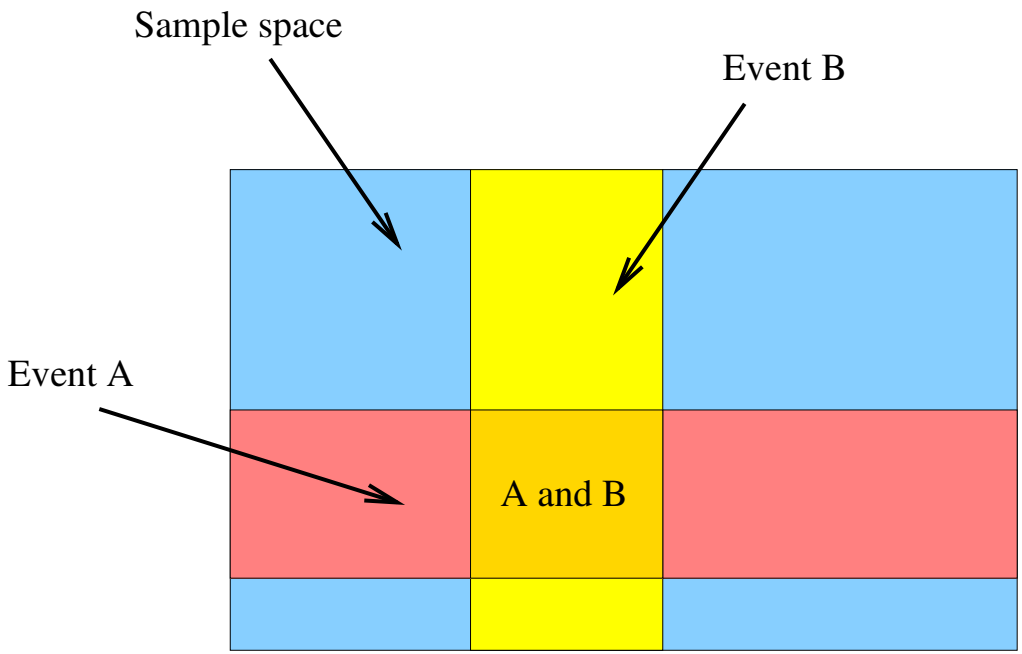
$$P(A \wedge B) = P(B|A) \cdot P(A)$$

we also have

$$P(B|A) = P(B)$$

Graphical intuition

Two “orthogonal” events:



Bayes' rule

$$P(A|B) = \frac{P(A) \cdot P(B|A)}{P(B)}$$

Example

- in a school 60% of students are boys and 40% are girls.
- girls wear in the same number skirts and trousers
- boys only wear trousers

If we see a student wearing trousers, what is the probability that is a girl?

Solution

The probability a priori that a student is a girl is

$$P(G) = 2/5$$

the probability that a student wears trousers is

$$P(T) = 1/5 + 3/5 = 4/5$$

the probability that a student wear trousers, given that the student is a girl, is

$$P(T|G) = 1/2$$

Hence, the probability that a student wearing trousers is a girl is

$$P(G|T) = \frac{P(G) \cdot P(T|G)}{P(T)} = \frac{2/5 \cdot 1/2}{4/5} = 1/4$$

Another formulation of Bayes rule

$$P(Y|X) = \frac{P(Y) \cdot P(X|Y)}{P(X)}$$

or more precisely, for every i, j,

$$P(Y = y_i|X = x_j) = \frac{P(Y = y_i) \cdot P(X = x_j|Y = y_i)}{P(X = x_j)}$$

But we know that

$$P(X = x_j) = \sum_i P(X = x_j, Y = y_i) = \sum_i P(Y = y_i) \cdot P(X = x_j|Y = y_i)$$

so

$$P(Y = y_i|X = x_j) = \frac{P(Y = y_i) \cdot P(X = x_j|Y = y_i)}{\sum_i P(Y = y_i) \cdot P(X = x_j|Y = y_i)}$$

Posterior, likelihood, prior and marginal

$$\underbrace{P(Y|X)}_{posterior} = \frac{\overbrace{P(X|Y)}^{likelihood} \cdot \overbrace{P(Y)}^{prior}}{\underbrace{P(X)}_{marginal\ likelihood}} = \frac{\overbrace{P(X|Y)}^{likelihood} \cdot \overbrace{P(Y)}^{prior}}{\underbrace{\sum_{Y} P(X|Y) \cdot P(Y)}_{marginal\ likelihood}}$$

"Marginal" because we marginalized (i.e. integrated) over Y

Next arguments

- The Joint Distribution
- Bayes classifiers
- Naïve Bayes

The Joint distribution

1. build a table with all possible combinations of values of random variables (features)

2. compute the probability for any different combination of values

gender	working hours	health	prob (params)
F	≤ 40	poor	0.25
F	≤ 40	rich	0.03
F	> 40	poor	0.04
F	> 40	rich	0.01
M	≤ 40	poor	0.33
M	≤ 40	rich	0.10
M	> 40	poor	0.13
M	> 40	rich	0.11

Given n (boolean) features, we need to compute $2^n - 1$ parameters.

Use of the joint distribution

Having the joint distribution we may compute the probability of any event expressible as a logical combination of the features

$$P(E) = \sum_{row \in E} P(row)$$

Example

Let us compute the probability $P(M, poor)$

gender	w. hours	wealth	prob.
F	≤ 40	poor	0.25
F	≤ 40	rich	0.03
F	> 40	poor	0.04
F	> 40	rich	0.01
M	≤ 40	poor	0.33
M	≤ 40	rich	0.10
M	> 40	poor	0.13
M	> 40	rich	0.11

$P(M, poor) = 0.33 + 0.13 = 0.46$

Inference with the Joint distribution

It is also easy to compute the probability of an event E_1 given another event E_2

$$P(E_1|E_2) = \frac{P(E_1 \wedge E_2)}{P(E_2)} = \frac{\sum_{row \in E_1 \wedge E_2} P(row)}{\sum_{row \in E_2} P(row)}$$

Example

$$P(M|poor) = \frac{P(M, poor)}{P(poor)}$$

We know that $P(M, poor) = 0.46$. Let us compute $P(poor)$:

gender	w. hours.	wealth	prob.
F	≤ 40	poor	0.25
F	≤ 40	rich	0.03
F	> 40	poor	0.04
F	> 40	rich	0.01
M	≤ 40	poor	0.33
M	≤ 40	rich	0.10
M	> 40	poor	0.13
M	> 40	rich	0.11

$P(poor) = .75$ and $P(M|poor) = 0.46/0.75 = 0.61$

Conditional Probability vs. learning

Instead of computing

$$f : X \rightarrow Y$$

we may compute the probability

$$p : P(Y|X)$$

We know that we can use the joint distribution, so we just need to compute it.

End of the story?

Complexity issues

Let us build the joint table relative to

$$P(Y = wealth | X_1 = gender, X_2 = orelav.)$$

X_1=gender	X_2=ore lav.	$P(rich \mid X_1, X_2)$	$P(poor \mid X_1, X_2)$
F	≤ 40	.09	.91
F	> 40	.21	.79
M	≤ 40	.23	.77
M	> 40	.38	.62

To fill the table we need to compute $4 = 2^2$ parameters.

If we have n random variable $X = X_1 \times X_2, \ldots X_n$ where each X_i is boolean, we need to compute 2^n parameters.

These parameters are probabilities: to get reasonable value we would need a huge amount of data.

Can we use Bayes rule?

We know that

$$P(Y = y_i|X = x_j) = \frac{P(Y = y_i) \cdot P(X = x_j|Y = y_i)}{\sum_i P(Y = y_i) \cdot P(X = x_j|Y = y_i)}$$

So, to compute $P(Y = y_i|X = x_j)$ it is enough to compute

$$P(Y) \quad \text{and} \quad P(X_1, X_2, \ldots X_n|Y)$$

- how many parameters for $P(Y)$?
- how many parameters for $P(X_1, X_2, \ldots X_n|Y)$?

Naïve Bayes

Naïve Bayes assumes that

$$P(X_1, X_2, \ldots X_n|Y) = \prod_i P(X_i|Y)$$

that is, given Y, X_i and X_j are independent from each other.

Conditional independence

Two events X_i and X_j are independent given Y if

$$P(X_i|X_j, Y) = P(X_i|Y)$$

Example 1 A box contains two coins: a regular coin and a fake two-headed coin (P(H)=1). Choose a coin at random, toss it twice and consider the following events:

A = First coin toss is H
B = Second coin toss is H
C = First coin is regular

A and B are NOT independent (prove it), but they are conditionally independent given C.

Example 2 For individuals, height and vocabulary are not independent, but they are if age is given.

Conditional independence in Naïve Bayes

$$\begin{aligned} P(X_1, X_2|Y) &= P(X_1|X_2, Y) \cdot P(X_2|Y) && \text{by the chain rule} \\ &= P(X_1|Y) \cdot P(X_2|Y) && \text{by cond. ind.} \end{aligned}$$

In general,

$$P(X_1, X_2, \ldots X_n|Y) = \prod_i P(X_i|Y)$$

How many parameters to describe $P(X_1, X_2, \ldots X_n|Y)$ (in the boolean case)?

- without conditional Independence
- in Naïve Bayes

Naïve Bayes in a nutshell

Bayes rule

$$P(Y = y_i | X_1, \ldots X_n) = \frac{P(Y = y_i) \cdot P(X_1, \ldots X_n | Y = y_i)}{P(X_1, \ldots X_n)}$$

Naïve Bayes

$$P(Y = y_i | X_1, \ldots X_n) = \frac{P(Y = y_i) \cdot \prod_j P(X_j | Y = y_i)}{P(X_1, \ldots X_n)}$$

Classification of a new sample $x^{new} = \langle x_1, \ldots x_n \rangle$

$$Y^{new} = arg\ max_{y_i} P(Y = y_i) \cdot \prod_j P(X_j = x_j | Y = y_i)$$

Learning algorithm

Discrete Random variables X_i, Y

- **Training**
  - for any possible value y_k of Y, estimate

$$\pi_k = P(Y = y_k)$$

  - for any possible value x_{ij} of X_i estimate

$$\theta_{ijk} = P(X_i = x_{ij} | Y = y_k)$$

- **Classification of** $a^{new} = \langle a_1, \ldots a_n \rangle$

$$\begin{aligned} Y^{new} &= arg\ max_{y_k}\ P(Y = y_k) \cdot \prod_i P(X_i = a_i | Y = y_k) \\ &= arg\ max_k\ \pi_k \cdot \prod_i \theta_{ijk} \end{aligned}$$

supposing that $a_i = x_{ij}$ (i.e. a_i is the j-th among the discrete values of the attribute X_i)

Estimating parameters

Maximum likelihood estimates (MLE's):

$$\pi_k = P(Y = y_k) = \frac{\#\mathcal{D}\{Y = y_k\}}{|\mathcal{D}|}$$

$$\theta_{ijk} = P(X = x_{ij}|Y = y_k) = \frac{\#\mathcal{D}\{X_i = x_{i,j} \wedge Y = y_k\}}{\#\mathcal{D}\{Y = y_k\}}$$

Example: a good day to play tennis?

Outlook	Temp	Humidity	Wind	Play
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Priors

$\pi_{Yes} = 9/14 = .64$

$\pi_{No} = 5/14 = .36$

Computing θ (Yes)

Outlook	Temp	Humidity	Wind	Play
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Overcast	Cool	Normal	Strong	Yes
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes

Outlook

$\theta_{Sunny,Yes} = 2/9$

$\theta_{Overc.,Yes} = 4/9$

$\theta_{Rain,Yes} = 3/9$

Temp

$\theta_{Hot,Yes} = 2/9$

$\theta_{Mild,Yes} = 4/9$

$\theta_{Cool,Yes} = 3/9$

Humidity

$\theta_{High,Yes} = 3/9$

$\theta_{Normal,Yes} = 6/9$

Wind

$\theta_{Weak,Yes} = 6/9$

$\theta_{Strong,Yes} = 3/9$

Computing θ (No)

Outlook	Temp	Humidity	Wind	Play
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Rain	Cool	Normal	Strong	No
Sunny	Mild	High	Weak	No
Rain	Mild	High	Strong	No

Outlook

$\theta_{Sunny,No} = 3/5$

$\theta_{Overc.,No} = 0$

$\theta_{Rain,No} = 2/5$

Temp

$\theta_{Hot,No} = 2/5$

$\theta_{Mild,No} = 2/5$

$\theta_{Cool,No} = 1/5$

Humidity

$\theta_{High,No} = 4/5$

$\theta_{Normal,No} = 1/5$

Wind

$\theta_{Weak,No} = 2/5$

$\theta_{Strong,No} = 3/5$

Prediction

New instance:

Outlook=Sunny,Temp.=Cool,Humidity=High,Wind=Strong

We need to compute

$$arg\ max_{y \in yes,no} p(y) \cdot p(sunny|y) \cdot p(cool|y) \cdot p(high|y) \cdot (strong|y)$$

that is, we need to compare

$$\pi_{yes} \cdot \theta_{sunny,yes} \cdot \theta_{cool,yes} \cdot \theta_{high,yes} \cdot \theta_{strong,yes}$$
$$= 9/14 \cdot 2/9 \cdot 3/9 \cdot 3/9 \cdot 3/9 = .0053$$

$$\pi_{no} \cdot \theta_{sunny,no} \cdot \theta_{cool,no} \cdot \theta_{high,no} \cdot \theta_{strong,no}$$
$$= 5/14 \cdot 3/5 \cdot 1/5 \cdot 4/5 \cdot 3/5 = .0205$$

NO!

Next arguments

- Generative techniques
- Limitations and cautions

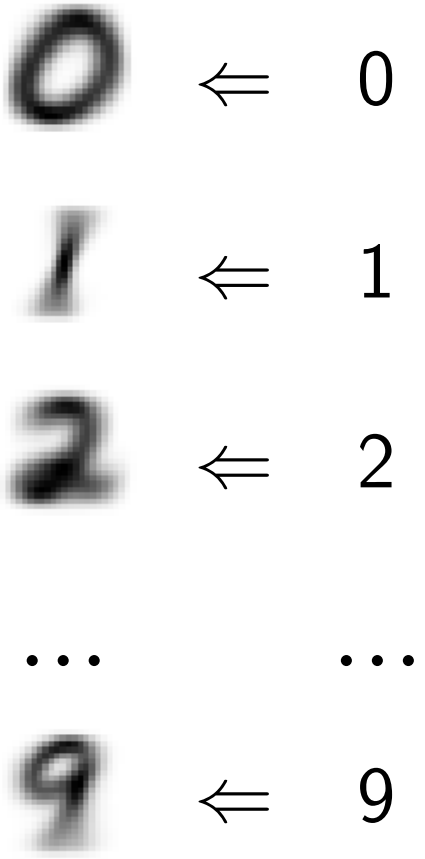
The generative nature of Naive Bayes

We are interested in computing

$$P(Y = y_i | X_1, \ldots X_n)$$

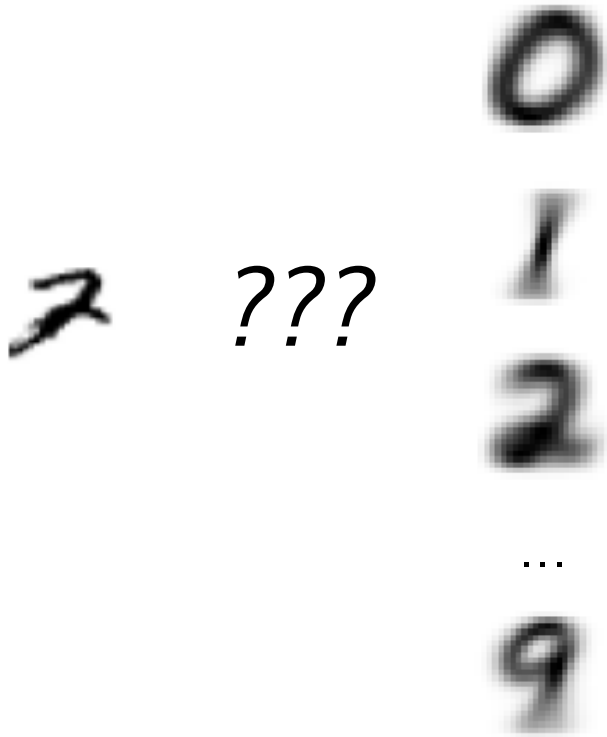
Bayes' rule allow to reverse the problem, asking instead to understand the **distribution of data, given the category**

$$P(X_1, \ldots X_n | Y = y_i)$$



Classification of new data

We classify a new data point by asking to which one of the input distributions it is the most likely to belong to:



Joint vs. Naïve

The distributions we would be interested in are the joint distributions

$$P(X_1, \ldots X_n | Y = y_i)$$

In the case of mnist, the distributions of images, reparted by categories, in a space with 784 dimensions.
Since this is unfeasible, we use the naive approach to assume that all features are independent, reducing the problem to the estimation of

$$P(X_j | Y = y_i)$$

for all possible feautures $j = 1, \ldots, n$.

Cautions (1)

In some cases, MLE for $P(X_i|Y)$ may be 0

e.g. $P(Play = No|Outlook = Overcast)$

- Why should be worry?
- How can we avoid the problem?

Cautions (2)

Naïve Bayes assumes events are independent from each other (given Y).

What if this is not the case?

Limitations

Suppose we have a set of random images, with pixels either 0 or 1.

Choose two pixels p_1 and p_2. We want to classify the image in category A if $p_1 == p_2$ and in category B otherwise.

It looks like a pretty simple classification. Let us try to use Naive Bayes.

What is $P(p_1 = 1|A)$?

What is $P(p_1 = 1|B)$?

What is $P(p_2 = 1|A)$?

What is $P(p_2 = 1|A)$?

So? ...

About
Maximum Likelihood Estimation

Back to parameters estimation

Maximum likelihood estimates (MLE's):

$$\pi_k = P(Y = y_k) = \frac{\#\mathcal{D}\{Y = y_k\}}{|\mathcal{D}|}$$

$$\theta_{ijk} = P(X_i = x_{ij}|Y = y_k) = \frac{\#\mathcal{D}\{X_i = x_{i,j} \wedge Y = y_k\}}{\#\mathcal{D}\{Y = y_k\}}$$

Back to parameters estimation

Maximum likelihood estimates (MLE's):

$$\pi_k = P(Y = y_k) = \frac{\#\mathcal{D}\{Y = y_k\}}{|\mathcal{D}|}$$

$$\theta_{ijk} = P(X_i = x_{ij}|Y = y_k) = \frac{\#\mathcal{D}\{X_i = x_{i,j} \wedge Y = y_k\}}{\#\mathcal{D}\{Y = y_k\}}$$

Why ?

Example

Suppose to toss a (possibly fake) coin 10 times.
We get 6 Heads (H) and 4 Tails (T).

One could "naturally" conclude that $P(H) = .6$ and $P(T) = .4$.

Example

Suppose to toss a (possibly fake) coin 10 times.
We get 6 Heads (H) and 4 Tails (T).

One could "naturally" conclude that $P(H) = .6$ and $P(T) = .4$.

Suppose that there are just two possibilities:

- the coin is fair, that is $P(H) = .5$ and $P(T) = .5$
- the coin is fake, with probabilities $P(H) = .7$ and $P(T) = .3$

Which is the most likely according to our observations, and why?

Independent ed equiprobable events

Bernoulli's Distribution (ex. flipping a coin)

two possible outcomes 0 and 1 with probabilities θ and $1 - \theta$.

Let X^n be the number of 0 in a sequence of n flips
X^n follows a binomial distribution

$$P(X^n = \alpha_0|\theta) = \binom{n}{\alpha_0} \cdot \theta^{\alpha_0} \cdot (1-\theta)^{\alpha_1}$$

where $\alpha_1 = n - \alpha_0$ is the number of 1 in the sequence (id est, $\alpha_0 + \alpha_1 = n$)

Back to our example

$n = 10$, $\alpha_0 = 6$ and $\alpha_1 = 4$.

If $\Theta = .5$ and $1 - \Theta = .5$

$$P(X^n = 6|\theta) = \binom{n}{\alpha_0} \cdot \theta^{\alpha_0} \cdot (1-\theta)^{\alpha_1} = \binom{10}{6} \cdot \left(\frac{1}{2}\right)^6 \cdot \left(\frac{1}{2}\right)^4 = .205$$

If $\Theta = .7$ and $1 - \Theta = .3$

$$P(X^n = 6|\theta) = \binom{n}{\alpha_0} \cdot \theta^{\alpha_0} \cdot (1-\theta)^{\alpha_1} = \binom{10}{6} \cdot \left(\frac{7}{10}\right)^6 \cdot \left(\frac{3}{10}\right)^4 = .200$$

So, $\Theta = .5$ is slightly more likely than $\Theta = .7$.

What is the most likely value for Θ?

Computing MLE

$$\hat{\theta} = arg\ max_\theta\ P(X^n = \alpha_0|\theta) =$$
$$= arg\ max_\theta\ \theta^{\alpha_0} \cdot (1-\theta)^{\alpha_1}$$

Equivalently we may look for θ maximizing the logarithm of the previous expression

$$ln(\theta^{\alpha_0} \cdot (1-\theta)^{\alpha_1}) = \alpha_0 ln(\theta) + \alpha_1 ln(1-\theta)$$

Deriving with respect to θ we get

$$\frac{\alpha_0}{\theta} - \frac{\alpha_1}{1-\theta} = \frac{\alpha_0 - \alpha_0\theta - \alpha_1\theta}{\theta \cdot (1-\theta)}$$

That is zero for

$$\theta = \frac{\alpha_0}{\alpha_0 + \alpha_1} = \frac{\alpha_0}{n}$$

Multivalued case

Discrete Distribution (ex. tossing a dice)
k possible outcomes $\{1, 2, \ldots, k\}$ with probabilities θ_i where $\sum_i \theta_i = 1$.

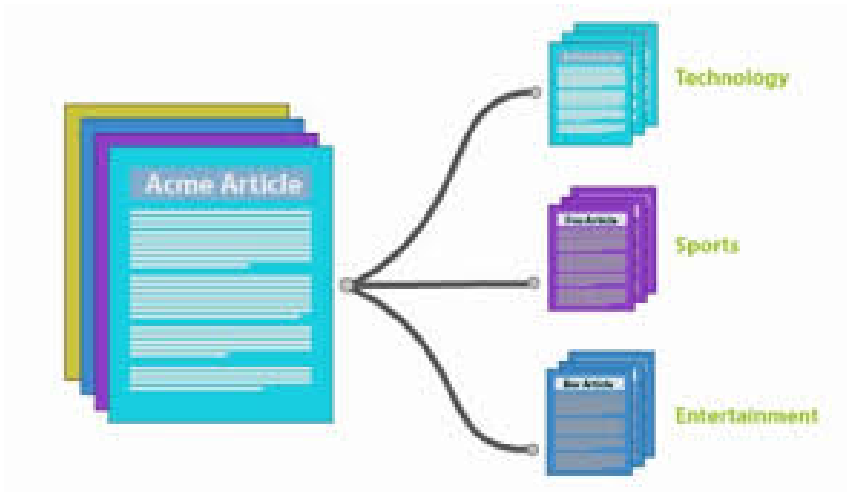
Sequence of n tosses: multinomial distribution

$$P(X^n = \overline{\alpha_i}|\theta) = c_{\overline{\alpha_i}} \prod_i \theta_i^{\alpha_i}$$

where α_i is the number of i in the sequence and $c_{\overline{\alpha_i}}$ is a combinatorial constant not depending on θ

$$\textbf{MLE}: \quad \theta_i = \frac{\alpha_i}{\sum_i \alpha_i} = \frac{\alpha_i}{n}$$

Document classification (bag of words approach)



Document classification

event X_i = i-th word in the document: a discrete random variable assuming as many values as words in the language

$$\theta_{i,word,\ell} = P(X_i = word | Y = \ell)$$

probability that in a document of the category "ℓ" the word "word" appears at position "i"

we assume that all event are independents (?) and have a distribution independent from the position (?)

$$\theta_{i,word,\ell} = \theta_{j,word,\ell} = \theta_{word,\ell}$$

Training and classification

Discrete random variables X_i, Y

- **Training**
  - for any possible value y_k of Y, estimate

$$\pi_k = P(Y = y_k)$$

  - for any possible value x_{ij} of the attribute X_i estimate

$$\theta_{ijk} = P(X_i = x_{ij} | Y = y_k)$$

- **Classification of** $a^{new} = \langle a_1, \ldots a_n \rangle$ (sequence of words)

$$\begin{aligned} Y^{new} &= arg\ max_{y_k}\ P(Y = y_k) \cdot \prod_i P(X_i = a_i | Y = y_k) \\ &= arg\ max_k\ \pi_k \cdot \prod_i \theta_{ijk} \end{aligned}$$

where $x_{ij} = a_i$

Estimating parameters

Maximum likelihood estimates (MLE's):

- $\pi_k = P(Y = y_k)$
  fraction of the documents in category y_k
- $\theta_{i,word,k} = \theta_{word,k} = P(X = word | Y = y_k)$
  frequency of the word "word" in documents of the category y_k

Log likelihood

Instead of

$$\begin{aligned} Y^{new} &= arg\ max_{y_k}\ P(Y = y_k) \cdot \prod_i P(X_i = w_j | Y = y_k) \\ &= arg\ max_k\ \pi_i \cdot \prod_i \theta_{ijk} \end{aligned}$$

We may compute

$$\begin{aligned} Y^{new} &= arg\ max_{y_k}\ log(P(Y = y_k) \cdot \prod_i P(X_i = w_j | Y = y_k)) \\ &= arg\ max_k\ log(\pi_k) + \sum_i log(\theta_{ijk}) \end{aligned}$$

Moreover, if $\theta_{ijk} = \theta_{i'jk} = \theta_{jk}$

$$\sum_i log(\theta_{ijk}) = \sum_j n_j \cdot log(\theta_{jk})$$

where n_j is the number of occurrences of the word w_j in the document to be classified.

Cosine similarity

Consider vectors where each word is a different dimension.

Training

For any category k build a "spectral" vector

$$s_k = \langle log(\theta_{jk}) \rangle_{j \in words}$$

θ_{jk} = frequency of the word j in documents of the category k.

Given a new document, compute a vector

$$d = \langle n_j \rangle_{j \in words}$$

and classify the document as

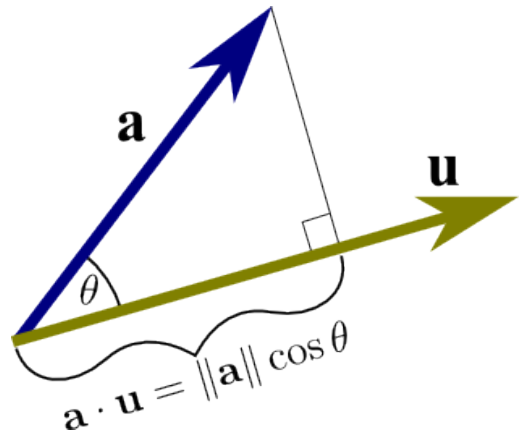
$$arg\ max_k \quad \underbrace{d \cdot s_k}_{\substack{cosine \\ similarity}} = \sum_j d_j \cdot s_{jk}$$

Dot product, geometrically and analytically

Geometric definition

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}|cos(\theta)$$

where θ is the angle between the two vectors



Analytic definition

given $\mathbf{a} = (a_1, a_2, \ldots, a_n)$
and $\mathbf{b} = (b_1, b_2, \ldots, b_n)$

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$$

Equivalence

By the cosine rule (a generalization of Pythagoras' theorem),

$$|\mathbf{a} - \mathbf{b}|^2 = |\mathbf{a}|^2 + |\mathbf{b}|^2 - 2|\mathbf{a}||\mathbf{b}|cos(\theta)$$

so

$$\mathbf{a} \cdot \mathbf{b} = \frac{|\mathbf{a}|^2 + |\mathbf{b}|^2 - |\mathbf{a} - \mathbf{b}|^2}{2}$$

Analytically (planar case):

let $\mathbf{a} = [a_1, a_2]$ and $\mathbf{b} = [b_1, b_2]$; we get

$$\mathbf{a} \cdot \mathbf{b} = \frac{a_1^2 + a_2^2 + b_1^2 + b_2^2 - (a_1 - b_1)^2 - (a_2 - b_2)^2}{2} = a_1 b_1 + a_2 b_2$$

Next Arguments

- The linear nature of Naïve Bayes

  (boolean case)

The linear nature of Naïve Bayes

X_i, Y booleans

Classification of $\vec{x} = \langle x_1, \ldots, x_n \rangle$:

$$\frac{P(Y=1|X_1 \ldots X_n = \vec{x})}{P(Y=0|X_1 \ldots X_n = \vec{x})} = \frac{P(Y=1)\prod_i P(X_i = x_i|Y=1)}{P(Y=0)\prod_i P(X_i = x_i|Y=0)} \geq 1$$

Passing to logarithms

$$log\frac{P(Y=1)}{P(Y=0)} + \sum_i log\frac{P(X_i = x_i|Y=1)}{P(X_i = x_i|Y=0)} \geq 0$$

Let $\theta_{ik} = P(X_i = 1|y = k)$ (hence $P(X_i = 0|y = k) = 1 - \theta_{ik}$), then we have the following linear discrimination test:

$$log\frac{P(Y=1)}{P(Y=0)} + \sum_i x_i \cdot log\frac{\theta_{i1}}{\theta_{i0}} + \sum_i (1-x_i) \cdot log\frac{1-\theta_{i1}}{1-\theta_{i0}} \geq 0$$

The case unfolding trick

Consider a function $f(x)$ where x is a boolean variable, then:

$$f(x) = x * f(1) + (1 - x) * f(0)$$

In general, given a function $f(x)$ with x ranging over a discrete domain $\{a_1, \ldots, a_n\}$ we can unfold f in the form

$$\begin{aligned} f(x) = \text{ case } x \text{ of:} \\ & [a_1 \Rightarrow f(a_1) \\ & |a_2 \Rightarrow f(a_2) \\ & \ldots \\ & |a_n \Rightarrow f(a_n) \\ &] \end{aligned}$$

A form of (finite) memoization.

Linear Classification

Classification algorithms based on a linear combination of the features values.

Every feature is evaluated independently from the others and contributes to the result in a linear way, with a suitable weight (that is a parameter of the model, to be estimated).

For instance, if the features are pixels of some image, we may use linear methods only up some normalization (in position and dimension) of the object to be recognized.

Gaussian Naïve Bayes

What if the features X_i are continuous?

For instance, X_i could be the height or age or annual income of some individual, or the color of a pixel in an image, etc.

To use Naïve Bayes we need to compute $P(X_i|Y)$, but if X_i is continuous, pointwise probabilities are null.

A common approach is to suppose $P(X_i|Y)$ has a Gaussian (or Normal) distribution.

Gaussian Distribution

Probability density (with integral = 1)

$$p(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

mean value

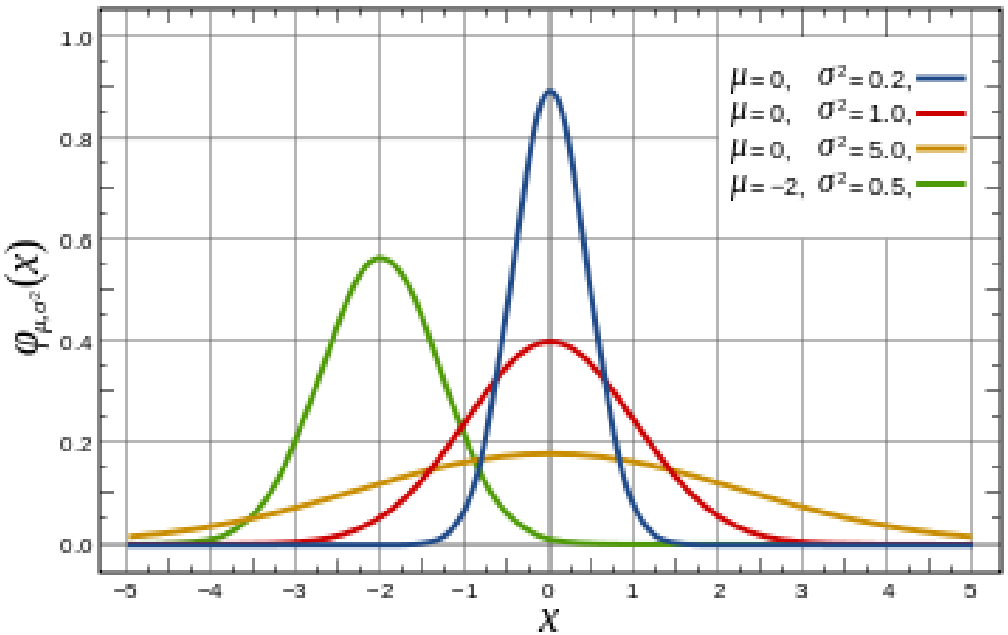
$$E[X] = \mu$$

variance

$$Var[X] = \sigma^2$$

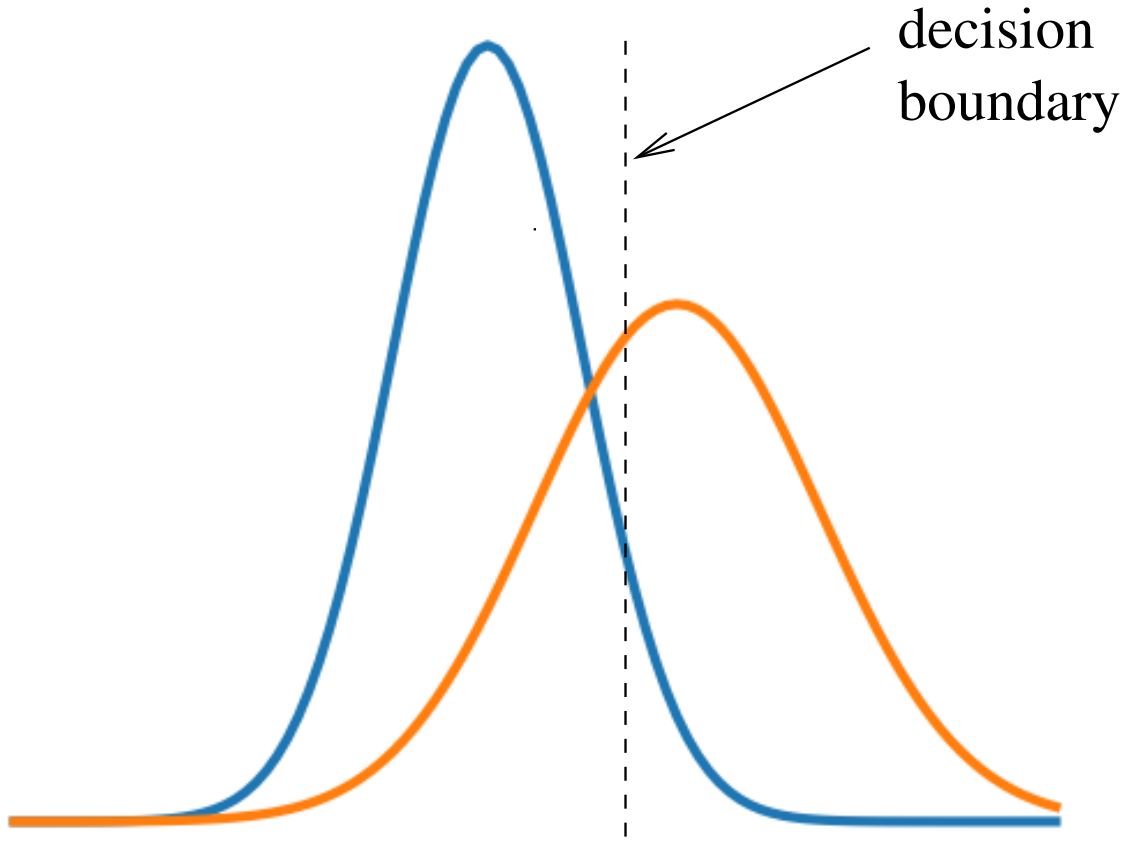
standard deviation

$$\sigma_X = \sigma$$

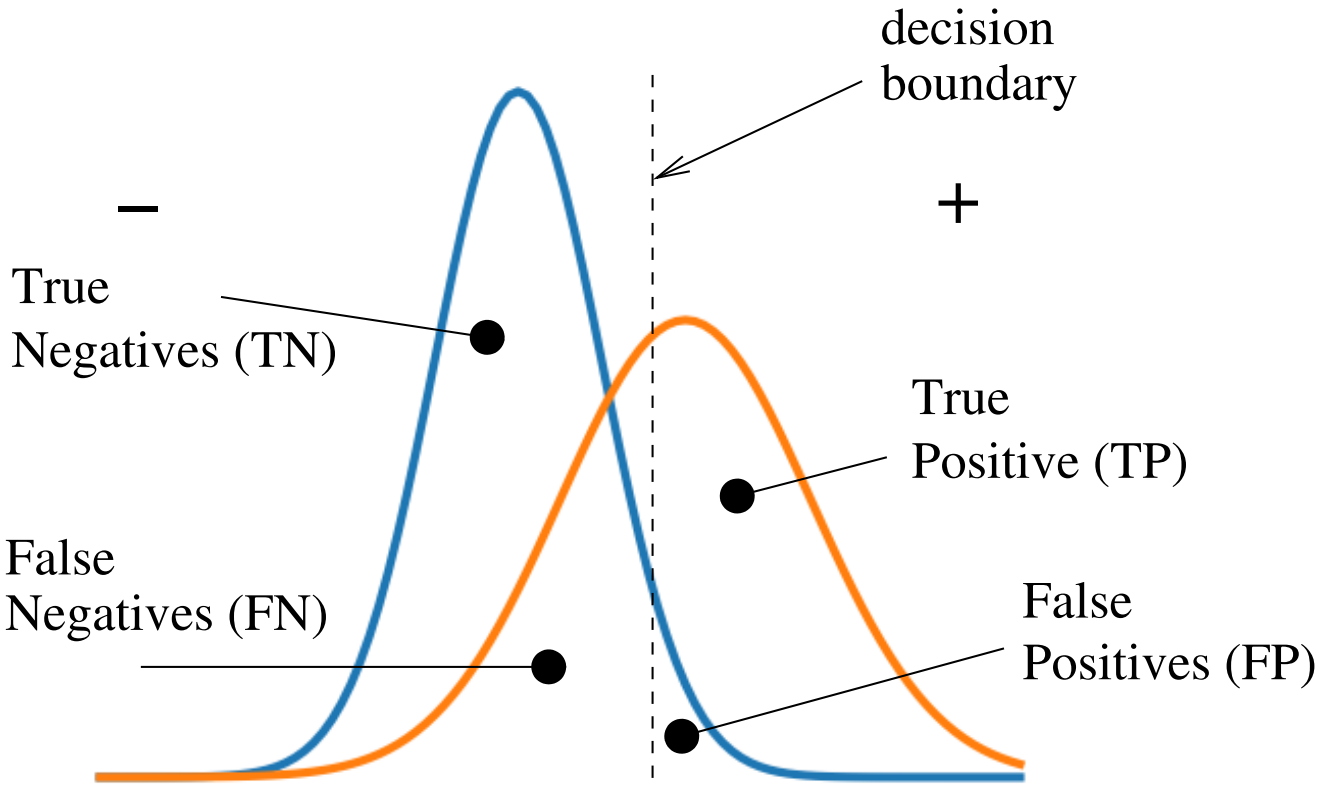


Example

Classify gender given height.



TP, FP, TN, FN


decision
boundary
–
+
True
Negatives (TN)
True
Positive (TP)
False
Negatives (FN)
False
Positives (FP)

Accuracy, Precision and Recall

$$Accuracy = \frac{TP + TN}{All}$$

How many samples have been correctly classified?

$$Precision = \frac{TP}{TP + FP}$$

how many of the selected samples are relevant?

$$Recall = \frac{TP}{TP + FN}$$

how many of relevant samples have been selected?

$$F1 = 2\frac{Precision \cdot Recall}{Precision + Recall}$$

armonic mean of precision and recall

Descriptive parameters of the model

We assume (inductive bias) that for every value y_k of Y the random variable $P(X_i|Y = y_k)$ has a Gaussian distribution

$$N(x|\mu_{ik}, \sigma_{ik}) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} e^{-\frac{(x-\mu_{ik})^2}{2\sigma_{ik}^2}}$$

Learning: estimate the values of the parameters μ_{ik}, σ_{ik} and $\pi_k = P(Y = y_k)$.

Classification of $x^{new} = \langle a_1, \ldots a_n \rangle$

$$\begin{aligned} Y^{new} &= arg\ max_{y_k}\ P(Y = y_k) \cdot \prod_i P(X_i = a_i|Y = y_k) \\ &= arg\ max_k\ \pi_k \cdot \prod_i N(a_i|\mu_{ik}, \sigma_{ik}) \end{aligned}$$

Maximum Likelihood Estimates

μ_{ik} = **mean value of** X_i **for samples with label** $Y = y_k$

Formally:

$$\mu_{ik} = \frac{\sum_j X_i^j \delta(Y^j = y_k)}{\sum_j \delta(Y^j = y_k)}$$

where j ranges over samples in the training set

$$\delta(Y^j = y_k) = \begin{cases} 1 & \textit{se } Y^j = y_k \\ 0 & \textit{altrimenti} \end{cases}$$

σ_{ik}^2 = variance of X_i for samples with label $Y = y_k$

$$\sigma_{ik}^2 = \frac{\sum_j (X_i^j - \mu_{ij})^2 \delta(Y^j = y_k)}{\sum_j \delta(Y^j = y_k)}$$

Next arguments

- Logistic Regression
- The logistic function

Logistic Regression

Idea:

- Naïve Bayes allows us to compute $P(Y|X)$ after having learned $P(Y)$ and $P(X|Y)$
- Why not directly learn $P(Y|X)$?

The shape of $P(Y|X)$

Hypothesis:

- Y boolean random variable
- X_i continuous random variable
- X_i independent from each other Y
- $P(X_i|Y = k)$ have Gaussian distributions $N(\mu_{ik}, \sigma_i)$ (**Warning** not σ_{ik}!)
- Y has a Bernoulli distribution (π)

Then

$$P(Y = 1|X = \langle x_1 \dots x_n \rangle) = \frac{1}{1 + exp(w_0 + \sum_i w_i x_i)}$$

Proof

By hypothesis

$$P(X_i|Y=k) = N(X_i, \mu_{ik}, \sigma_i) = \frac{1}{\sigma_i\sqrt{2\pi}} e^{-\frac{(x-\mu_{ik})^2}{2\sigma_i^2}}$$

Hence:

$$\begin{aligned}
P(Y=1|X) &= \frac{P(Y=1)\cdot P(X|Y=1)}{P(Y=1)\cdot P(X|Y=1) + P(Y=0)\cdot P(X|Y=0)} \\
&= \frac{1}{1+\frac{P(Y=0)\cdot P(X|Y=0)}{P(Y=1)\cdot P(X|Y=1)}} \\
&= \frac{1}{1+exp(ln(\frac{P(Y=0)\cdot P(X|Y=0)}{P(Y=1)\cdot P(X|Y=1)}))} \\
&= \frac{1}{1+exp(ln\frac{1-\pi}{\pi} + \sum_i ln\frac{P(X_i|Y=0)}{P(X_i|Y=1)})} \\
&= \frac{1}{1+exp(ln\frac{1-\pi}{\pi} + \sum_i (\frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2}X_i + \frac{\mu_{i1}^2-\mu_{i0}^2}{2\sigma_i^2})}
\end{aligned}$$

Very useful!

If

$$P(Y = 1|X = \langle x_1 \dots x_n \rangle) = \frac{1}{1 + exp(w_0 + \sum_i w_i x_i)}$$

then

$$P(Y = 0|X = \langle x_1 \dots x_n \rangle) = \frac{exp(w_0 + \sum_i w_i x_i)}{1 + exp(w_0 + \sum_i w_i x_i)}$$

Hence

$$\frac{P(Y = 0|X = \langle x_1 \dots x_n \rangle)}{P(Y = 1|X = \langle x_1 \dots x_n \rangle)} = exp(w_0 + \sum_i w_i x_i)$$

and in particular

$$ln \frac{P(Y = 0|X = \langle x_1 \dots x_n \rangle)}{P(Y = 1|X = \langle x_1 \dots x_n \rangle)} = w_0 + \sum_i w_i x_i$$

To classify $X = \langle x_1 \dots x_n \rangle$ it suffices to check if $w_0 + \sum_i w_i x_i > 0$

Logistic regression

Logistic regression *assumes*

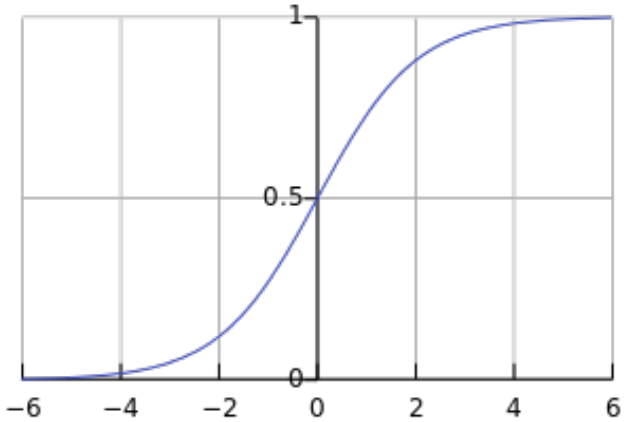
$$P(Y = 1|X = \langle x_1 \dots x_n \rangle) = \frac{1}{1 + exp(-w_0 - \sum_i w_i x_i)}$$

and directly tries to estimate the parameters w_i (the change in sign is ininfluent).

The function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

is an important function called logistic function



Training for logistic regression (binary case)

We know that

$$P(Y = 1|x_i w_i) = \sigma(w_0 + \sum_i w_i x_i)$$

Given the (independent) samples $\langle x_i^\ell, y^\ell \rangle$, their probability is

$$\prod_\ell P(y^\ell|x_i^\ell w_i) = \prod_\ell P(y^\ell = 1|x_i^\ell w_i)^{y^\ell} \cdot P(y^\ell = 0|x_i^\ell w_i)^{(1-y^\ell)}$$

We want to find the values for the parameters w_i that maximize this probability (MLE).

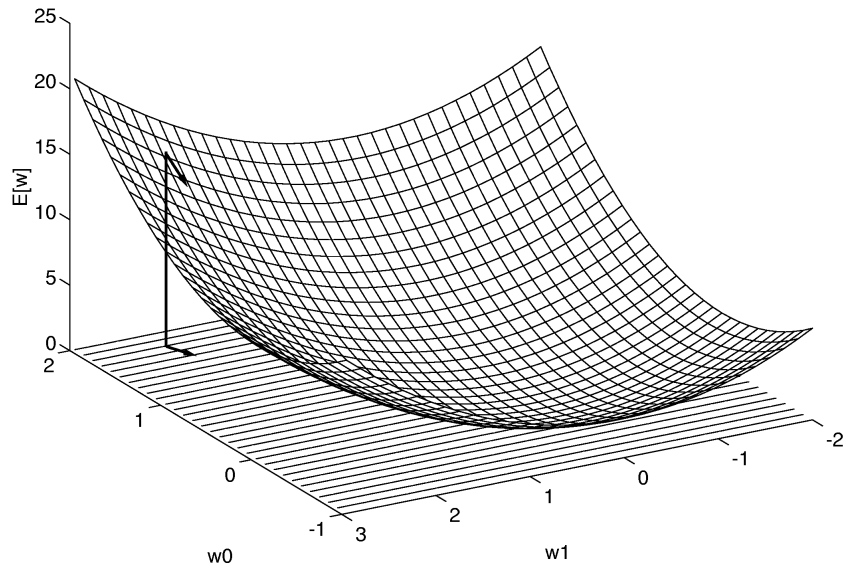
Analogously, we can work on the logarithm and maximize

$$\sum_\ell log\ P(y^\ell|x_i^\ell w_i) =$$
$$\sum_\ell (y^\ell \cdot log\ P(Y = 1|x_i^\ell w_i) + (1 - y^\ell) \cdot log\ P(Y = 0|x_i^\ell w_i))$$

Gradient

Unfortunately, there is no analytic solution for the previous optimization problem.

so, we use iterative optimization methods, like the gradient technique:



$$\nabla_w[E] = [\frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}]$$

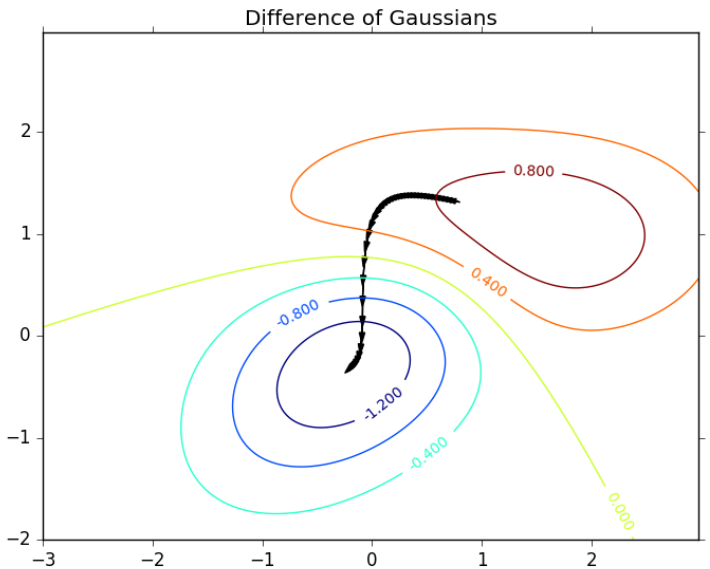
training: $\Delta w_i = \mu \cdot \frac{\partial E}{\partial w_i}$

$$w_i = w_i + \Delta w_i$$

The gradient technique

Iterative approach

The objective is to minimize some error function $\Theta(w)$ on all training examples, suitably adjusting the parameters.

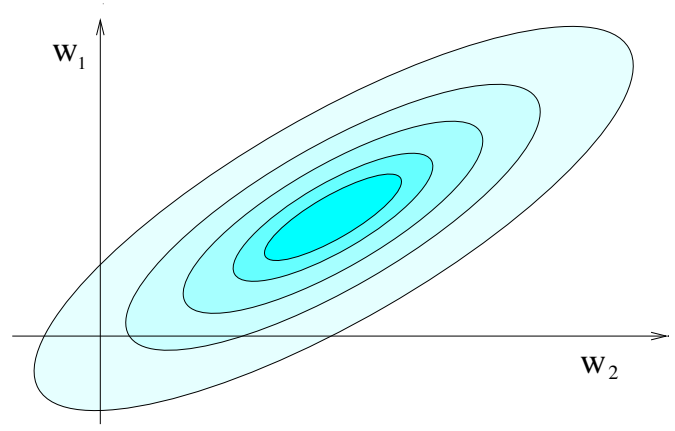


We can reach a minimal configuartion for $\Theta(w)$ by iteratively taking small steps in the direction opposite to the gradient (gradient descent).

This is a **general technique**.

The error surface

Image one "horizontal" axis for each parameter and one "vertical" axis for the error.

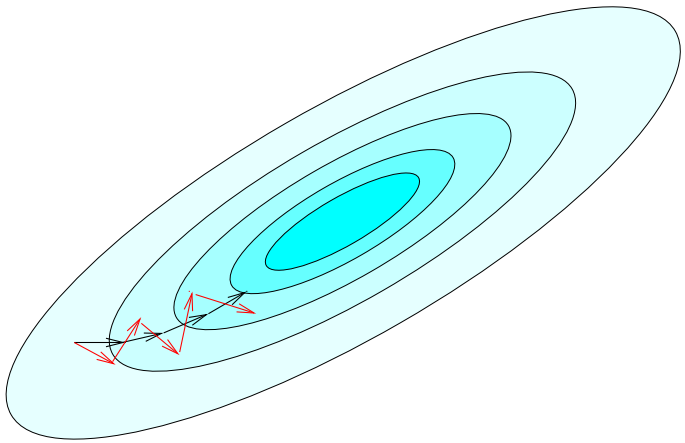


For a linear net with a squared error, the surface is a quadratic bowl:
- Vertical cross-sections are parabolas
- Horizontal cross-sections are ellipses

The error surface for logistic regression is convex too.

For multi-layer, non-linear nets the error surface can be much more complicated (with many local minima).

Online vs Batch learning

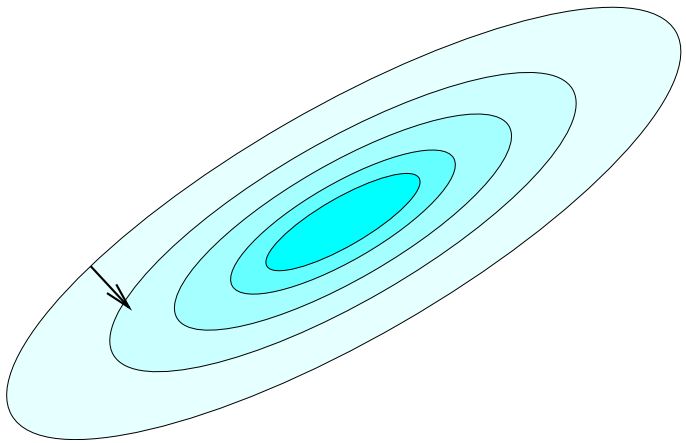


Full batch on all training samples: gradient points to the direction of steepest descent on the error surface (perpendicular to contour lines of the error surface)

Online (one sample at a time) gradient zig-zags around the direction of the steepest descent.

Mini-batch (random subset of training samples): a good compromise.

Backpropagation can be slow



The gradient does not necessarily points to the direction of the local minimum

Gradient for logistic regression

- probability that sample ℓ is in category $Y = 1$

$$P(Y = 1|x_i^\ell w_i) = \sigma(w_0 + \sum_i w_i x_i^\ell) = \alpha^\ell$$

- Log-likelihood $l(w)$

$$\sum_\ell log\, P(Y = y^\ell|x_i^\ell w_i) = \sum_\ell y^\ell log(\alpha^\ell) + (1 - y^\ell) log(1 - \alpha^\ell)$$

- gradient (proof on next slide)

$$\frac{\partial l(w)}{\partial w_i} = \sum_\ell x_i^\ell \cdot (y^\ell - \alpha^\ell)$$

Computing the gradient

$$\alpha^\ell = \sigma(z) = \frac{1}{1 + exp(-z))} \text{ where } z = w_0 + \sum_i w_i x_i^\ell$$

$$\frac{\partial log(\alpha^\ell)}{\partial z} = \frac{1}{\alpha^\ell}\frac{\partial \alpha^\ell}{\partial z} = \frac{exp(-z)}{1 + exp(-z)} = 1 - \alpha^\ell$$

$$log(1 - \alpha^\ell) = log\frac{exp(-z)}{1 + exp(-z)} = -z + log(\alpha^\ell)$$

$$\frac{\partial log(1 - \alpha^\ell)}{\partial z} = -1 + 1 - \alpha^\ell = -\alpha^\ell$$

remember that $l(w) = \sum_\ell y^\ell log(\alpha^\ell) + (1 - y^\ell) log(1 - \alpha^\ell)$

$$\frac{\partial l(w)}{\partial w_i} = \frac{\partial l(w)}{\partial z}\frac{\partial z}{\partial w_i} \quad = (\sum_\ell y^\ell(1 - \alpha^\ell) - (1 - y^\ell)\alpha^\ell)x_i^\ell$$
$$= (\sum_\ell (y^\ell - \alpha^\ell))x_i^\ell$$

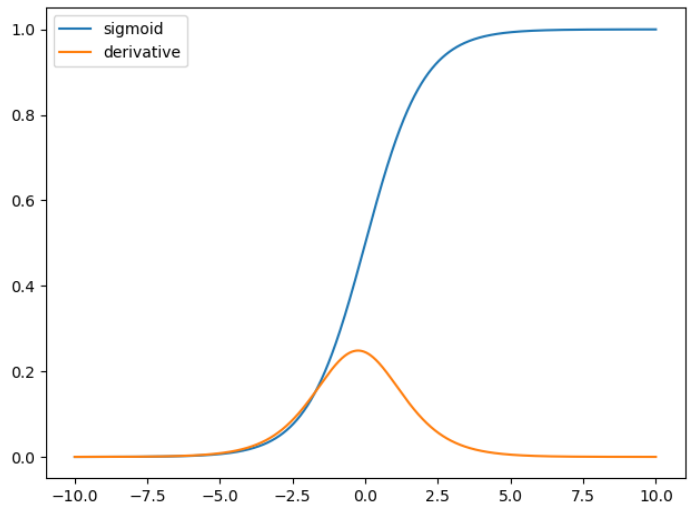
Observation: the derivative of the sigmoid

In the previous slide we proved that

$$\frac{\partial log(\alpha^{\ell})}{\partial z} = \frac{\partial \sigma(z)}{\partial z} = \frac{1}{\sigma(z)} \frac{\partial \sigma(z)}{\partial z} = 1 - \sigma(z)$$

Hence:

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1 - \sigma(z))$$



The learning process

Iterate the following update operation until reaching the desired approximation (e.g. until accuracy on testing data is satisfactory, or increment is below a given threshold ϵ)

$$w_i \leftarrow w_i + \mu \sum_{\ell} x_i^{\ell} \cdot (y^{\ell} - P(Y = y^{\ell} | x_i w_i))$$

Frequently one add a regularizer (prior):

$$w_i \leftarrow w_i - \mu\lambda |w_i| + \mu \sum_{\ell} x_i^{\ell} \cdot (P(Y = y^{\ell} | x_i w_i) - y^{\ell})$$

- helps to keep parameters w_i close to 0
- tends to reduce overfitting

Log likelyhood vs square error

Let us compare a couple of classification models, according to their predictions:

correct label	1	0	1	0
prediction model 1	.6	.6	.6	.4
prediction model 2	.8	.8	.7	.25

Log likelyhood vs square error

Let us compare a couple of classification models, according to their predictions:

correct label	1	0	1	0
prediction model 1	.6	.6	.6	.4
prediction model 2	.8	.8	.7	.25

negative log likelihood: $-\sum_\ell logP(Y = y^\ell | x^\ell)$

model 1 : $-log(.6) - log(.4) - log(.6) - log(.6) \approx 3.53$

model 2 : $-log(.8) - log(.2) - log(.7) - log(.75) \approx 3.57$

Log likelyhood vs square error

Let us compare a couple of classification models, according to their predictions:

correct label	1	0	1	0
prediction model 1	.6	.6	.6	.4
prediction model 2	.8	.8	.7	.25

negative log likelihood: $-\sum_\ell logP(Y = y^\ell | x^\ell)$

model 1 : $-log(.6) - log(.4) - log(.6) - log(.6) \approx 3.53$

model 2 : $-log(.8) - log(.2) - log(.7) - log(.75) \approx 3.57$

square error: $\sum_\ell (y^\ell - pred^\ell)^2$

model 1 : $(1 - .6)^2 + (0 - .6)^2 + (1 - .6)^2 + (0 - .4)^2 = .84$

model 2 : $(1 - .8)^2 + (0 - .8)^2 + (1 - .7)^2 + (0 - .25)^2 = .8325$

Log likelyhood vs square error: the gradients

negative loglikelihood

$$\frac{\partial l(z)}{\partial z} = \sigma(z) - y^{\ell}$$

square error

$$\frac{\partial(\sigma(z) - y^{\ell})^2}{\partial z} = 2(\sigma(z) - y^{\ell})\frac{\sigma(z) - y^{\ell}}{\partial z}$$

$$= 2(\sigma(z) - y^{\ell})\sigma(z)(1 - \sigma(z))$$

The gradient for s.e. is nearly 0 if the prediction is entirely wrong!!

Generative vs discriminative classifiers

Generative vs discriminative methods

Classification: estimate $f : X \rightarrow Y$ or $P(Y|X)$.

Generative Classifiers (e.g., Naïve Bayes)

- assume a given distribution for $P(X|Y), P(X)$
- estimate parameters for $P(X|Y), P(X)$ from training data
- use Bayes' rule to infer $P(Y|X)$

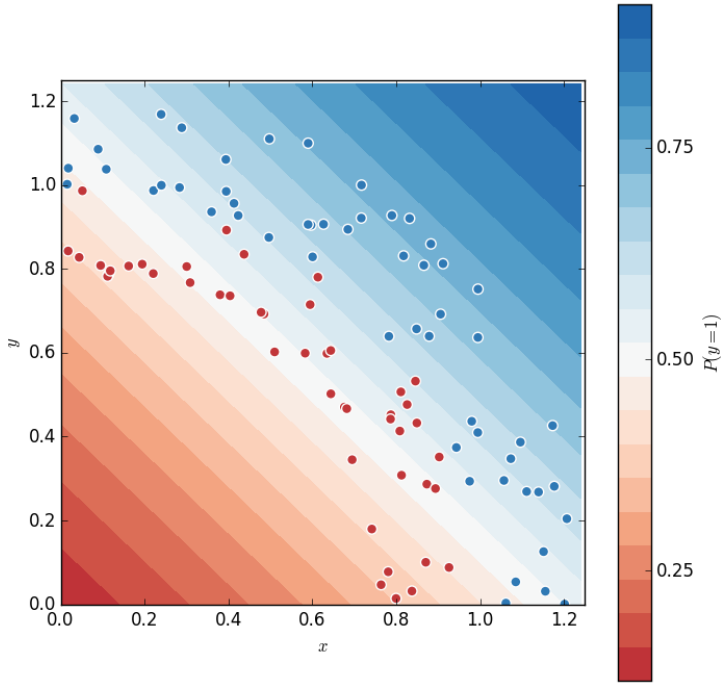
Discriminative Classifiers (e.g., Logistic regression)

- assume a given distribution for $P(Y|X)$ (or a given shape for the discrimination function)
- estimate the parameters for $P(Y|X)$ (or the discrimination function) from training data

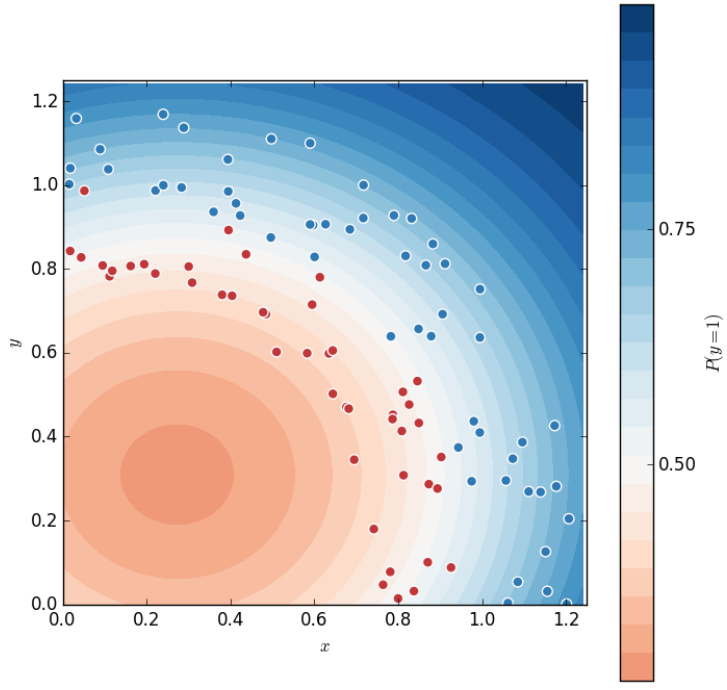
Naïve Bayes vs. Logistic regression

Example: points with random angle $0 \leq \theta \leq \pi$ and random distance $.75 \leq r \leq 1.25$ from origin.
Input features are the cartesian coordinates of the points (not mutually independent), and we try to discriminate points internal to the unary circle.

logistic regression



gaussian naïve bayes



Lo script in python (1)

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from matplotlib import pyplot as plt
from matplotlib.colors import ListedColormap

size = 100
# random training data
X = np.random.rand(size, 2)

def b(p): return p[1]>.5
y = map(b,X)

def toXY(p):
  theta = p[0]*np.pi/2
  len = (p[1]-.5)/2+1
  x,y=np.cos(theta)*len,np.sin(theta)*len
  return np.array([x,y],float)

for i in range(len(X)): X[i] = toXY(X[i])

xx, yy = np.mgrid[0:1.25:.01, 0:1.25:.01]
grid = np.c_[xx.ravel(), yy.ravel()]
```

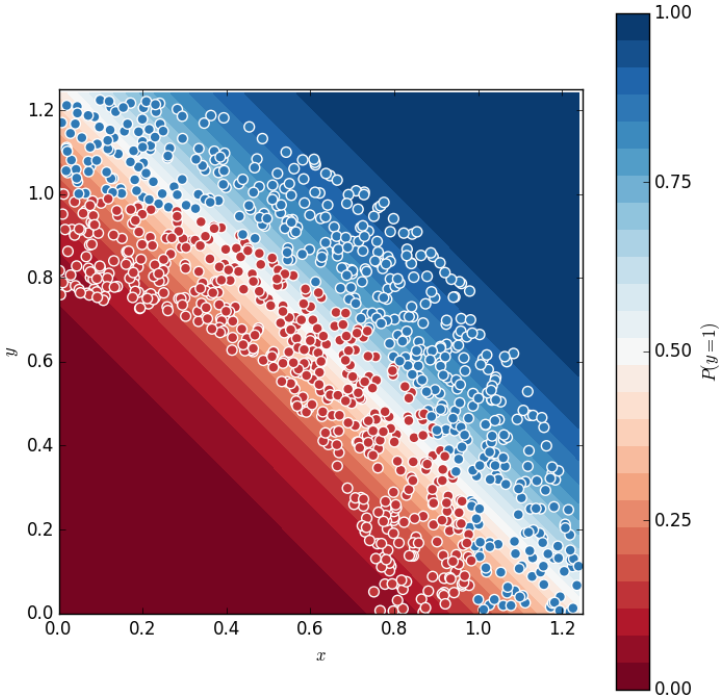
Lo script in python (2)

```
names = ["Logistic Regression", "Gaussian Naive Bayes"]
classifiers = [LogisticRegression(fit_intercept=True),GaussianNB()]

for name, clf in zip(names, classifiers):
  clf.fit(X,y)
  probs = clf.predict_proba(grid)[:, 1].reshape(xx.shape)
  #Now, plot the probability grid as a contour map
  f, ax = plt.subplots(figsize=(8, 8))
  contour = ax.contourf(xx, yy, probs, 25, cmap="RdBu",vmin=0, vmax=1)
  ax_c = f.colorbar(contour)
  ax_c.set_label("$P(y = 1)$")
  ax_c.set_ticks([0, .25, .5, .75, 1])
  #show the test set samples on top of it
  ax.scatter(X[:,0], X[:, 1], c=y, s=50,
             cmap="RdBu", vmin=-.2, vmax=1.2,
             edgecolor="white", linewidth=1)
  ax.set(aspect="equal",
         xlim=(0, 1.25), ylim=(0, 1.25),
         xlabel="$x$", ylabel="$y$")

plt.show()
```

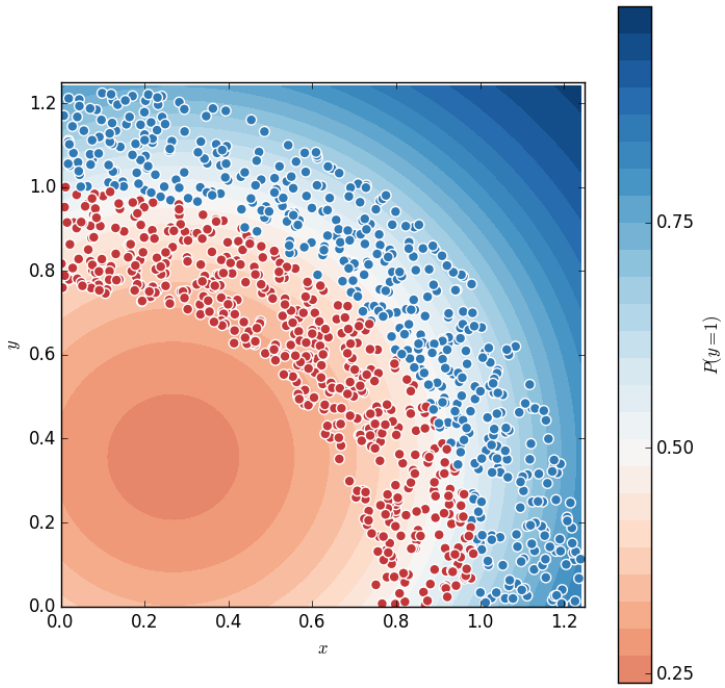
Naïve Bayes vs.Logistic regression

Same example with 1000 points instead of 100.

logistic regression



gaussian naïve bayes



Naïve Bayes vs.Logistic regression

Similar techniques.

Naïve Bayes is slightly more expressive.

Logistic regression is simpler and makes fewer assumptions on the model.

Cross entropy

Kullback-Leibler divergence

The Kullback-Leibler divergence $DKL(P\|Q)$ between two distributions Q and P, is a measure of the information loss due to approximating P with Q:

$$\begin{aligned} DKL(P\|Q) &= \sum_i P(i) log \frac{P(i)}{Q(i)} \\ &= \sum_i P(i)(logP(i) - logQ(i)) \\ &= -\underbrace{\mathcal{H}(P)}_{entropy} - \sum_i P(i) logQ(i) \end{aligned}$$

We call Cross-Entropy between P and Q the quantity

$$\mathcal{H}(P,Q) = -\sum_i P(i) logQ(i) = \mathcal{H}(P) + DKL(P\|Q)$$

Minimizing the cross entropy

Let P be the distribution of training data, and Q the distribution induced by the model.

We can take as our learning objective the minimization of the Kullback-Leibler divergence $DKL(P\|Q)$.

Since, given the training data, their entropy $\mathcal{H}(P)$ is constant, minimizing $DKL(P\|Q)$ is equivalent to minimizing the cross-entropy $\mathcal{H}(P,Q)$ between P and Q.

Cross entropy and log-likelihood

Let us consider the case of a binary classification.

Let $Q(y = 1|\mathbf{x})$ the probability that $\mathbf{x}$ is classified 1.
Hence, $Q(y = 0|\mathbf{x}) = 1 - Q(y = 1|\mathbf{x})$.

The real (observed) classification is $P(y = 1|\mathbf{x}) = y$ and similarly $P(y = 0|\mathbf{x}) = 1 - y$.

So we have

$$\begin{aligned}\mathcal{H}(P, Q) &= -\sum_i P(i) log Q(i) \\ &= -y\, log(Q(y = 1|\mathbf{x})) - (1 - y) log(1 - Q(y = 1|\mathbf{x})\end{aligned}$$

That is just the (negative) log-likelihood!

Multinomial Logistic Regression

Use of the multinomial model

Classification problems with multiple categories

- means of transport (bus, car, train, bicycle, etc.)
- favourite ice flavor (strawberry, lemon, cream, etc.)
- different characters in writing recognition
- ...

Multinomial regression allows us to associate a probability with the different categories, that is particularly useful when combining in cascade different learning techniques.

Binary case

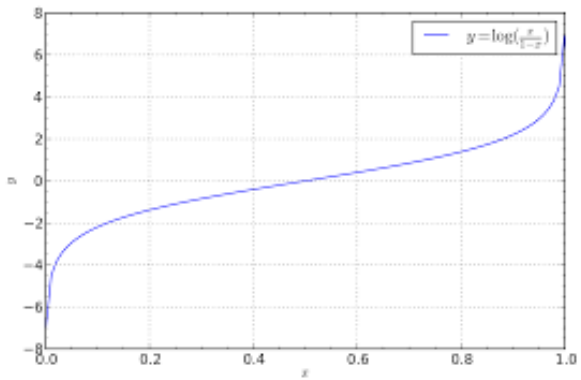
In the case of binary regression, we assume that the discrimination function between the two cases $Y = 0$ and $Y = 1$ is linear:

$$\underbrace{\underbrace{log \frac{P(Y = 0|\langle x_1 \dots x_n \rangle)}{P(Y = 1|\langle x_1 \dots x_n \rangle)}}_{odds}}_{log\ odds} = w_0 + \sum_i w_i x_i$$

This is sometimes expressed in terms of the logit function:

$$logit(P(Y = 0|\mathbf{x})) = w_0 + \sum_i w_i x_i$$

$$logit(p) = log \frac{p}{1 - p}$$



Logistic regression is also called binary logit.

n-ary case

In the case of multinomial regression with k categories $\{1, \ldots, k\}$ we have $k-1$ equations describing the log-odds of each category with respect to a reference category (e.g. category k):

$$log \frac{P(Y=1|\langle x_1 \ldots x_n \rangle)}{P(Y=k|\langle x_1 \ldots x_n \rangle)} = w_{1,0} + \sum_i w_{1,i} x_i$$

...

$$log \frac{P(Y=k-1|\langle x_1 \ldots x_n \rangle)}{P(Y=k|\langle x_1 \ldots x_n \rangle)} = w_{k-1,0} + \sum_i w_{k-1,i} x_i$$

This model relies on the **Independence of Irrelevant Alternatives** (IIA) assumption:
the odds between two categories A and B are not influenced by a third category C

Probability distributions

Let us use a vectorial notation (comprising the constant w_0)

$$log \frac{P(Y=j|\mathbf{x})}{P(Y=k|\mathbf{x})} = \mathbf{w}_j \mathbf{x}$$

Then, for every j

$$P(Y=j|\mathbf{x}) = P(Y=k|\mathbf{x}) \cdot e^{\mathbf{w}_j \mathbf{x}}$$

The sum of probabilities must be 1, that is

$$\sum_{j<k} P(Y=k|\mathbf{x}) \cdot e^{\mathbf{w}_j \mathbf{x}} + P(Y=k|\mathbf{x}) = 1$$

and hence

$$P(Y=k|\mathbf{x}) = \frac{1}{1+\sum_{j<k} e^{\mathbf{w}_j \mathbf{x}}} \quad \textit{and} \quad P(Y=j|\mathbf{x}) = \frac{e^{\mathbf{w}_j \mathbf{x}}}{1+\sum_{j<k} e^{\mathbf{w}_j \mathbf{x}}}$$

Comparison with the binary case

Binary case

$$P(Y = 1|\mathbf{x}) = \frac{1}{1 + e^{\mathbf{wx}}} \quad \textit{and} \quad P(Y = 0|\mathbf{x}) = \frac{e^{\mathbf{wx}}}{1 + e^{\mathbf{wx}}}$$

n-ary case

$$P(Y = k|\mathbf{x}) = \frac{1}{1 + \sum_{j<k} e^{\mathbf{w}_j \mathbf{x}}} \quad \textit{and} \quad P(Y = j|\mathbf{x}) = \frac{e^{\mathbf{w}_j \mathbf{x}}}{1 + \sum_{j<k} e^{\mathbf{w}_j \mathbf{x}}}$$

There exists a generalization of the logistic function?

softmax

The function

$$\text{softmax}(j, x_1, \ldots, x_k) = \frac{e^{x_j}}{\sum_{j=1}^{k} e^{x_j}}$$

generalizes the logistic function to the multinomial case.

Using softmax, we can express probabilities in the following way

$$P(Y = j|\mathbf{x}) = \text{softmax}(j, \mathbf{w}_1\mathbf{x}, \ldots, \mathbf{w}_{k-1}\mathbf{x}, 0)$$

It is easy to prove that for any c,

$$\text{softmax}(j, x_1 + c, \ldots, x_k + c) = \text{softmax}(j, x_1, \ldots, x_k)$$

hence, it is natural to assume that one of the argument (corresponding to the "reference category") is null, taking e.g. $c = -x_k$.

The same reasoning shows the irrelevance of the choice of the reference category for computing odds.

Training

The matrix of parameters $\mathbf{w_j}$ is computed via maximum likelihood estimation. The probability that a sample $\langle \mathbf{x}^\ell, y^\ell \rangle$ is correctly classified in the category y^ℓ is

$$P(Y = y^\ell | \mathbf{x}) = \prod_j P(Y = j | \mathbf{x})^{\delta_{jy^\ell}}$$

where δ_{ij} ìs the Kronecker δ: $\delta_{ij} := \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$

Supposing that data are independent, their likelihood is the product of individual probabilities. Passing to logarithms:

$$\sum_l \sum_j \delta_{jy^\ell} \cdot log(P(Y = j | \mathbf{x}))$$

that is usually maximized using a gradient technique.

About the loss function

$$\sum_{l} \underbrace{\sum_{j} \delta_{jy^{\ell}} \cdot log(P(Y = j|\mathbf{x}))}_{means\ \ that...}$$

for sample ℓ you only need to consider the category j corresponding to the true label y^{ℓ}, and the log of the probability $P(Y = j|\mathbf{x}))$ assigned by the model to the sample.

For multinomial regression, the loss function deriving from maximum loglikelihood is the categorical crossentropy between the observed categorical probability $\delta_{jy^{\ell}}$ of data (1 for the right category, 0 for others) and that syntesized by the model.

Gradient for multinomial logistic regression

- probability that sample ℓ is in category $Y = j$

$$P(Y = j|\mathbf{x}) = \text{softmax}(j, \mathbf{w}_1\mathbf{x}, \ldots, \mathbf{w}_{k-1}\mathbf{x}, 0) = \alpha_j^l$$

- Log-likelihood $l(w)$

$$\sum_l \sum_j \delta_{jy^\ell} \cdot log(P(Y = j|\mathbf{x}))$$

- gradient (feature i and category j)

$$\frac{\partial l(w)}{\partial w_{ij}} = \sum_\ell x_i^\ell \cdot (y^\ell - P(Y = j|\mathbf{x}))$$

Example: classification of mnist digits with MLR

Linear methods are powerful

One may have the erroneous feeling that linear methods are of no real interest, since too weak.

Usually, they becomes interesting when you have a large number of (independent) features at your disposal.

In a high dimensional space training points presumably have a very sparse distribution, hence it may becomes plausible to discriminate classes via hyperplanes.

Thinking in higher dimensions

The n-ball volume example.

The volume of a d-dimensional sphere of radius r grows as

$$Kr^d$$

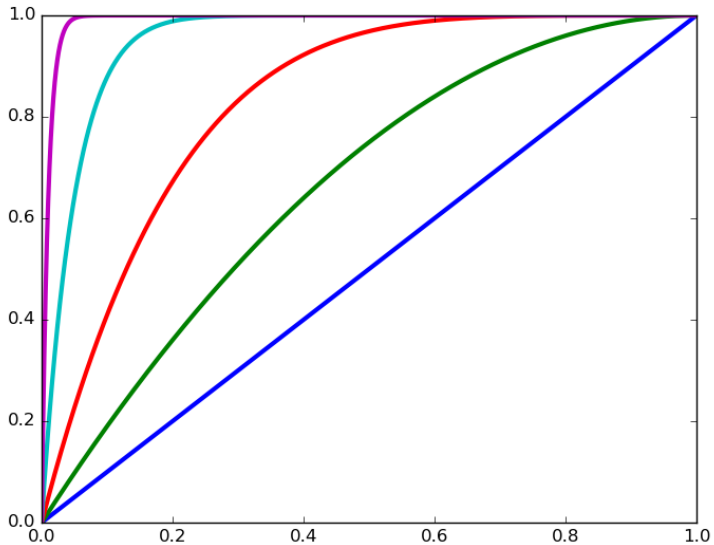
for some constant K.

The fraction of the volume of the sphere that lies between radius $r = 1$ and $r = 1 - \epsilon$ is

$$\frac{K - K(1-\epsilon)^d}{K} = 1 - (1-\epsilon)^d$$

The volume of a n-ball

The function $1 - (1 - \epsilon)^d$ for d=1,2,5,20,100



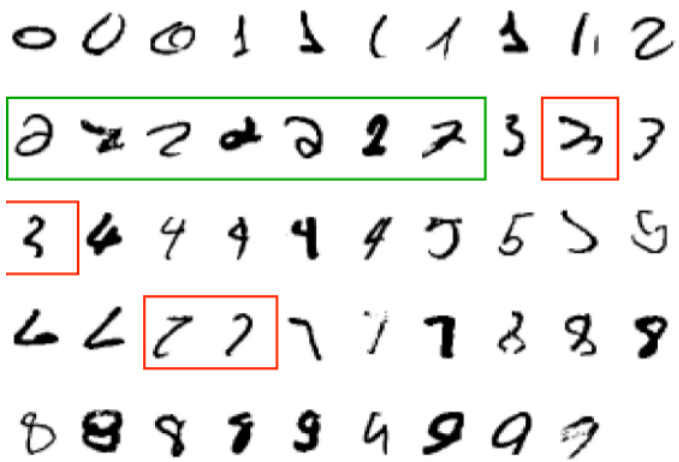
For a really high dimensional sphere, its volume is concentrated in a thin area close to the surface!!

The mnist case

In the case of images, each input digit is a possible feature.

Modified National Institute of Standards and Technology database

- grayscale images of handwritten digits, 28×28 pixels each
- 60,000 training images and 10,000 testing images



A tiny 28×28 image, already has 784 features.

Classification via multinomial regression

DEMO!

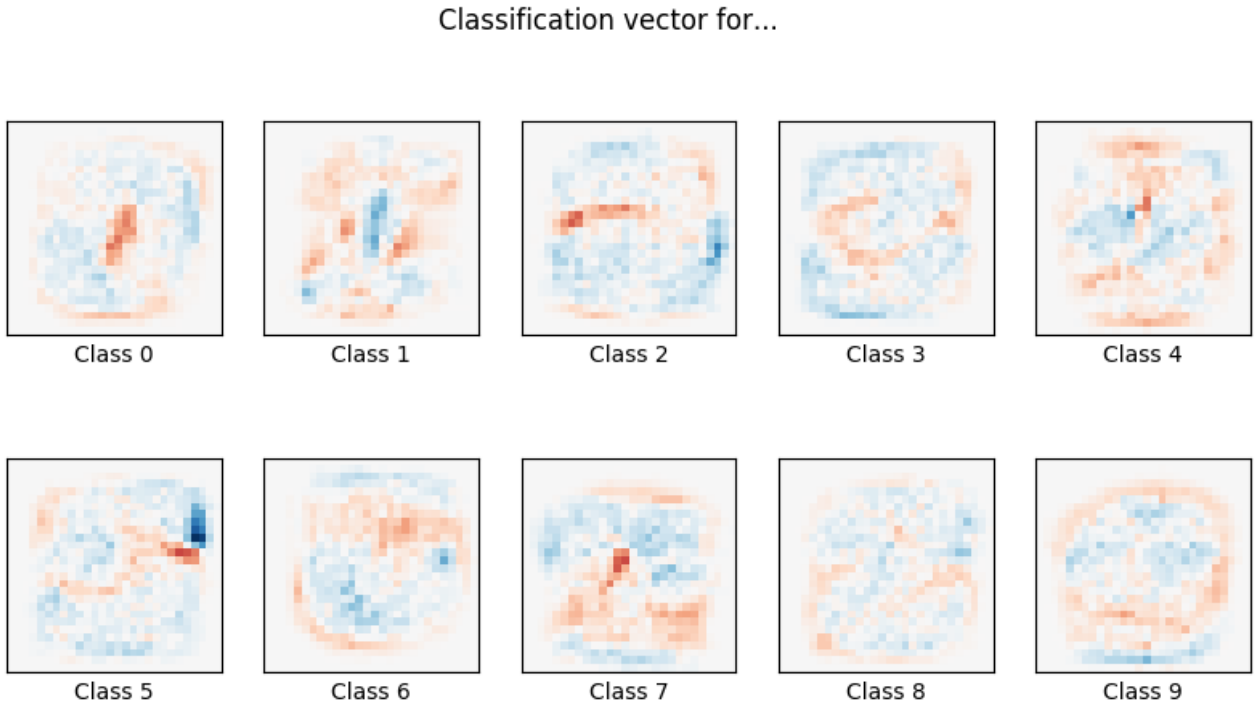
Classification via multinomial regression

DEMO!

```
Sparsity with L1 penalty: 25.18%
Test score with L1 penalty: 0.9257
Example run in 22.404 s
```

Weights as images

For each class, the coresponding weights have the same dimensionality of the input image (we have one weight for each feature).
We can draw them!



Linear Regression

Classification and regression

classification predicting a class:

Y discrete

regression predicting a numerical value:

Y continuous

Warning: the name logistic regression is misleading. Logistic regression is a classification technique.

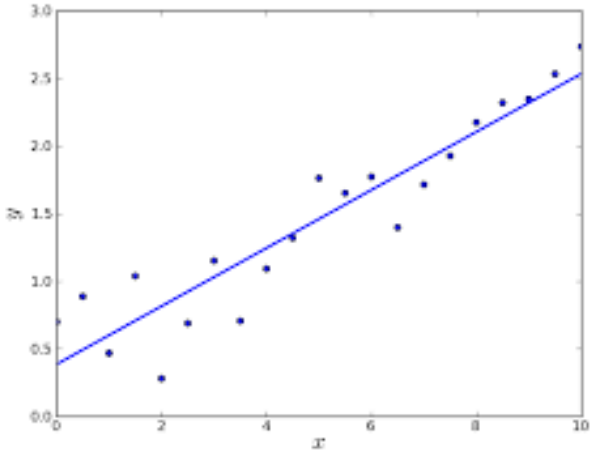
Regression

Learning $f : X \rightarrow Y$ where Y is a real number.

Probabilistic approach:

- choose a distribution $P(Y|X)$, characterized by some parameters θ
- optimize the parameters θ according to training data (MLE or MAP)

Parametric shape of $P(Y|X)$



Let us assume $Y = f(X)$ up to random, gaussian distributed, noise

$$Y = f(X) + \epsilon \text{ where } \epsilon \approx N(0, \sigma)$$

Hence, Y is a random variable with a distribution

$$p(Y|X) = N(f(X), \sigma)$$

with expected value $f(x)$ and standard deviation σ.

Estimating parameters

$$P(Y|X,W) = N[f(x),\sigma] = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-f(x))^2}{2\sigma^2}}$$

Log-likelihood:

$$l(W) = \sum_{\ell} ln \frac{1}{\sigma\sqrt{2\pi}} - \frac{(y^{\ell} - f(x^{\ell}))^2}{2\sigma^2}$$

Maximizing this quantity is the same as minimizing

$$err(W) = \sum_{\ell} (y^{\ell} - f(x^{\ell}))^2$$

But this is just the sum of squared errors.

The linear case

If $f(x) = w_0 + w_1(x)$, then

$$err(W) = \sum_{\ell} (y^{\ell} - w_0 - w_1 x^{\ell})^2$$

so

$$\frac{\partial err(W)}{\partial w_0} = -2 \sum_{\ell} (y^{\ell} - w_0 - w_1(x^{\ell}))$$

$$\frac{\partial err(W)}{\partial w_1} = -2 \sum_{\ell} (y^{\ell} - w_0 - w_1(x^{\ell}))x^{\ell}$$

We can use values for the gradient technique...

or solve the optimization problem analytically.

Analytic solution

Equating with 0 the partial derivatives, we get two equations in in two variables:

$$\begin{aligned} \textstyle\sum_\ell y^\ell &= N w_0 + w_1 \textstyle\sum_\ell x^\ell \\ \textstyle\sum_\ell y^\ell x^\ell &= w_0 \textstyle\sum_\ell x^\ell + w_1 \textstyle\sum_\ell (x^\ell)^2 \end{aligned}$$

Let

$$A = \begin{bmatrix} N & \sum_\ell x^\ell \\ \sum_\ell x^\ell & \sum_\ell (x^\ell)^2 \end{bmatrix} \quad W = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \quad Y = \begin{bmatrix} \sum_\ell y^\ell \\ \sum_\ell y^\ell x^\ell \end{bmatrix}$$

the previous equations are expressed by the system $AW = Y$, and hence

$$W = A^{-1} Y$$

Example

Let us look for a linear fitting for the points $(1, 2), (2, 3), (6, 4)$.

$$\sum_\ell y^\ell = N w_0 + w_1 \sum_\ell x^\ell$$
$$\sum_\ell y^\ell x^\ell = w_0 \sum_\ell x^\ell + w_1 \sum_\ell (x^\ell)^2$$

In our case,

$$\sum_\ell y^\ell = 9, N = 3, \sum_\ell x^\ell = 9$$
$$\sum_\ell y^\ell x^\ell = 32, \sum_\ell (x^\ell)^2 = 41$$

From which we get the equations

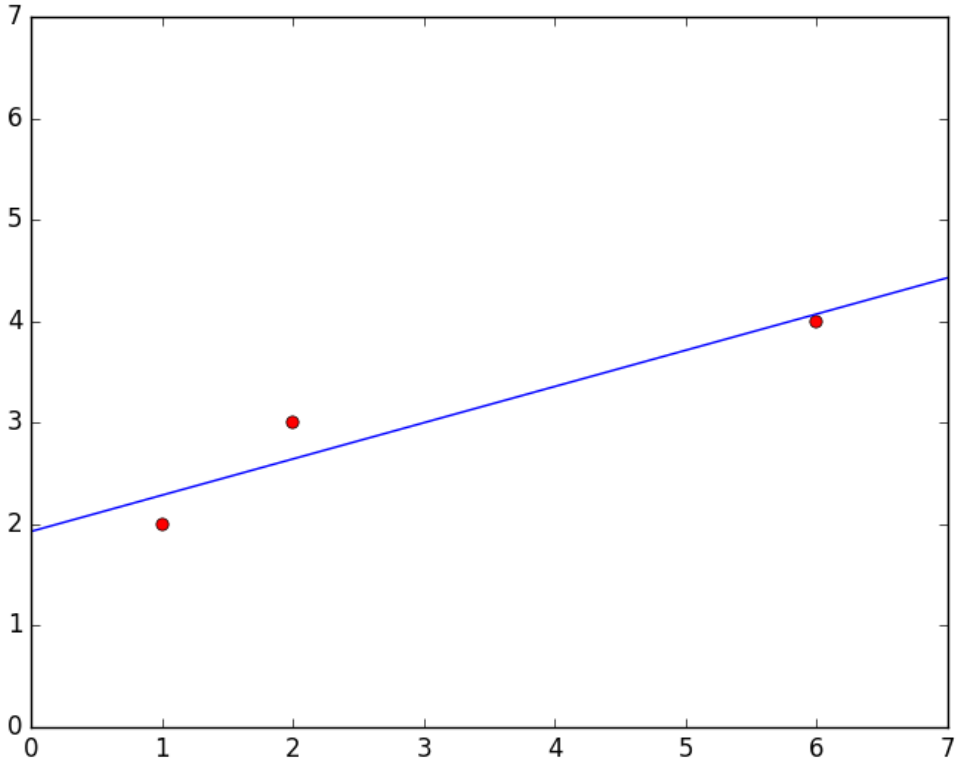
$$9 = 3w_0 + 9w_1$$
$$32 = 9w_0 + 41w_1$$

that has solutions $w_0 = 27/14$ and $w_1 = 5/14$

Example

Linear fitting for the points $(1, 2), (2, 3), (6, 4)$

$$y = \frac{27}{14} + \frac{5}{14}x$$



Regularization

Instead of minimizing

$$err(W) = \sum_{\ell} (y^{\ell} - f(x^{\ell}))^2$$

add a regularizer

$$err(W) = \lambda \sum_{i} w_i^2 + \sum_{\ell} (y^{\ell} - f(x^{\ell}))^2$$

(not solvable any longer by analytic means, even in the linear case)

Neural Networks and Deep Learning

Andrea Asperti

DISI - Department of Informatics: Science and Engineering
University of Bologna
Mura Anteo Zamboni 7, 40127, Bologna, ITALY
andrea.asperti@unibo.it

Deep Learning in a nutshell

Q: **What is Deep Learning?**

Deep Learning in a nutshell

Q: **What is Deep Learning?**

A: A branch of Machine Learning

Deep Learning in a nutshell

Q: **What is Deep Learning?**

A: A branch of Machine Learning

Q: **To what kind of problems it can be applied?**

Deep Learning in a nutshell

Q: **What is Deep Learning?**

A: A branch of Machine Learning

Q: **To what kind of problems it can be applied?**

A: To all problems suitable for Machine Learning

Deep Learning in a nutshell

Q: **What is Deep Learning?**

A: A branch of Machine Learning

Q: **To what kind of problems it can be applied?**

A: To all problems suitable for Machine Learning

Q: **Are there problems where DL particularly excels?**

Deep Learning in a nutshell

Q: **What is Deep Learning?**

A: A branch of Machine Learning

Q: **To what kind of problems it can be applied?**

A: To all problems suitable for Machine Learning

Q: **Are there problems where DL particularly excels?**

A: All problems involving myriads of features, e.g. images/speech/text processing, recommendation systems, ...

Deep Learning in a nutshell

Q: **What is Deep Learning?**

A: A branch of Machine Learning

Q: **To what kind of problems it can be applied?**

A: To all problems suitable for Machine Learning

Q: **Are there problems where DL particularly excels?**

A: All problems involving myriads of features, e.g. images/speech/text processing, recommendation systems, ...

Deep Learning in a nutshell (2)

Q: **What is the basic technique for DL?**

Deep Learning in a nutshell (2)

Q: **What is the basic technique for DL?**

A: Neural Networks

Deep Learning in a nutshell (2)

Q: **What is the basic technique for DL?**

A: Neural Networks

Q: **Why "Deep"?**

Deep Learning in a nutshell (2)

Q: **What is the basic technique for DL?**

A: Neural Networks

Q: **Why "Deep"?**

A:
- because it exploits Deep Neural Networks, composed by many layers of neurons

Deep Learning in a nutshell (2)

Q: **What is the basic technique for DL?**

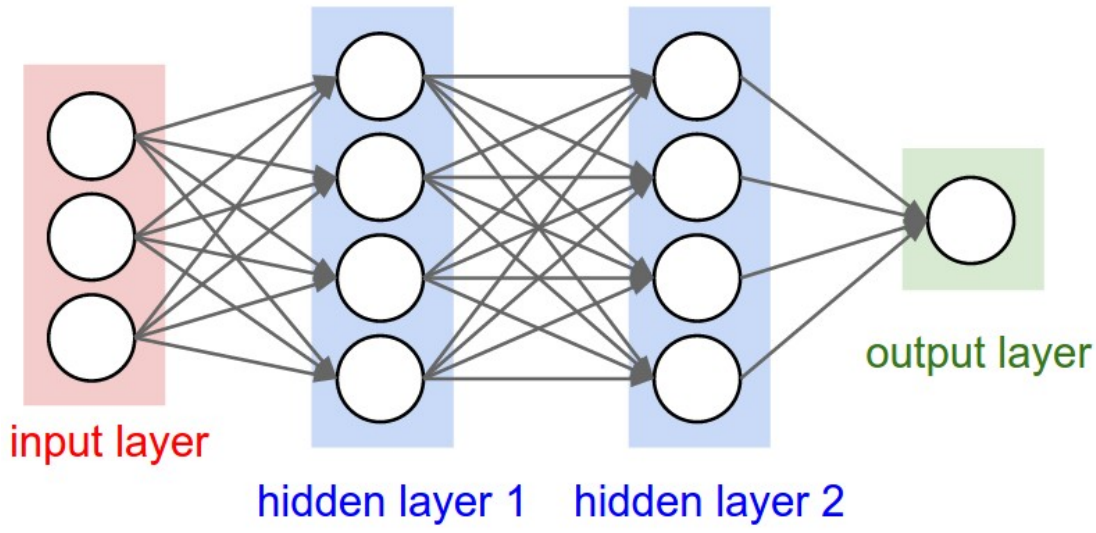
A: Neural Networks

Q: **Why "Deep"?**

A:
- because it exploits Deep Neural Networks, composed by many layers of neurons
- because it exploits deep features of data, that is features extracted from other features

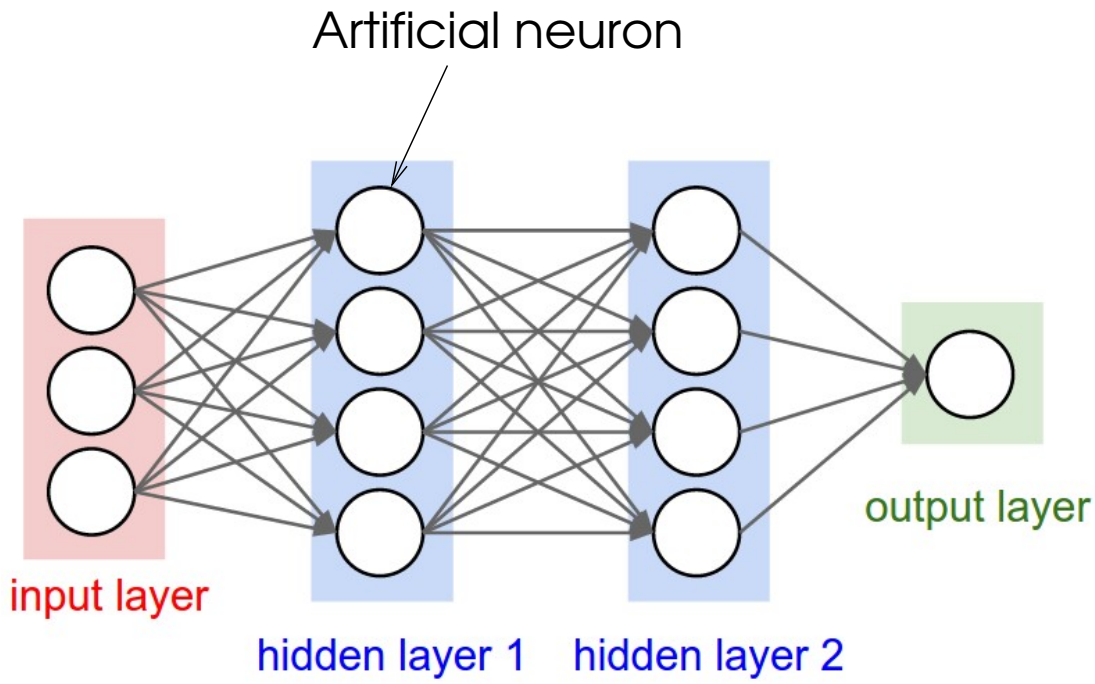
Next argument

Neural Networks



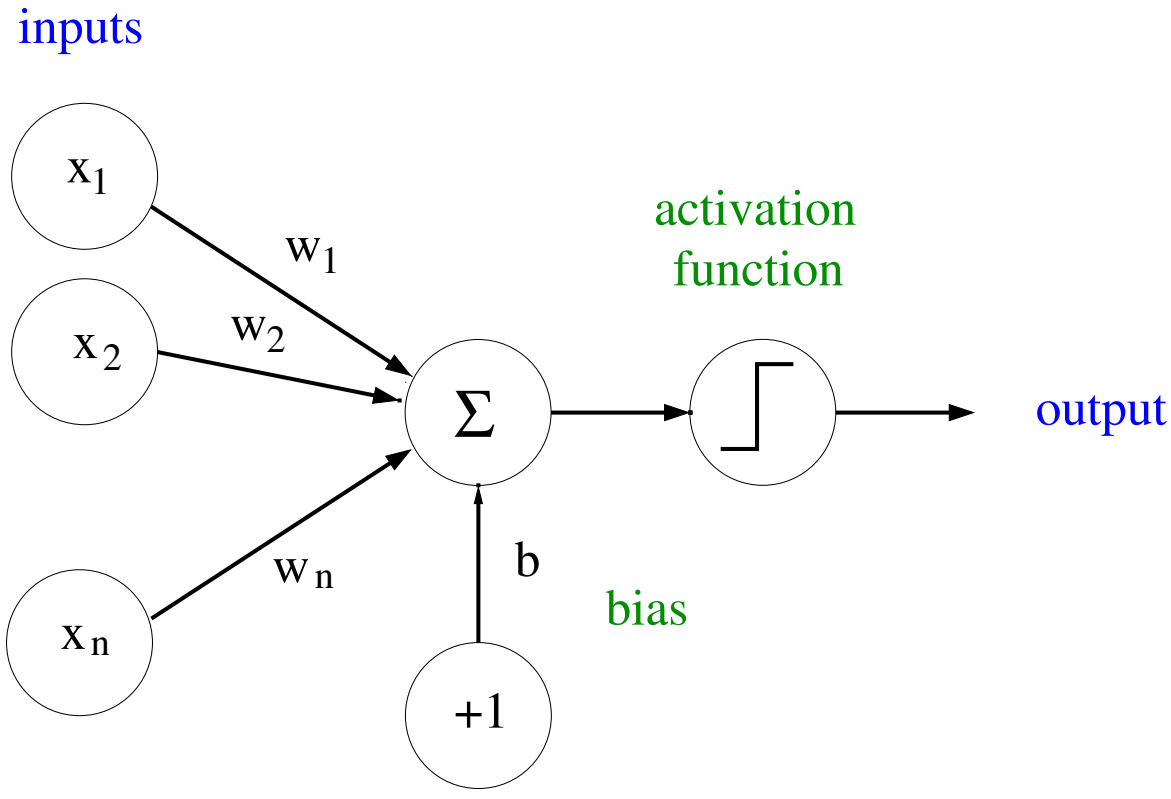
Neural Network

A network of (artificial) neurons



Each neuron takes multiple inputs and produces a single output (that can be passed as input to many other neurons).

The artificial neuron

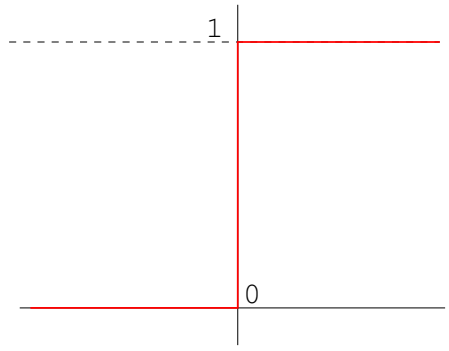


Each neuron (!) implements a logistic regressor

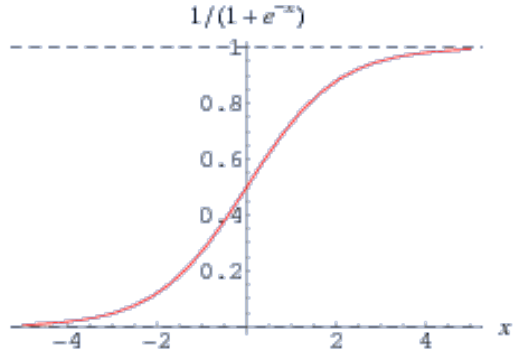
$$\sigma(wx + b)$$

Different activation functions

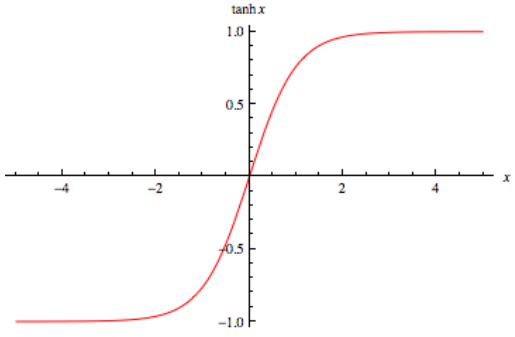
The activation function is responsible for threshold triggering.



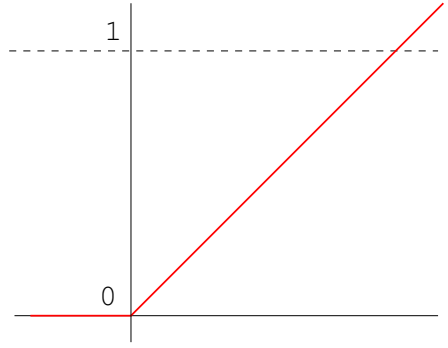
threshold: if $x > 0$ then 1 else 0



logistic function: $\frac{1}{1+e^{-x}}$

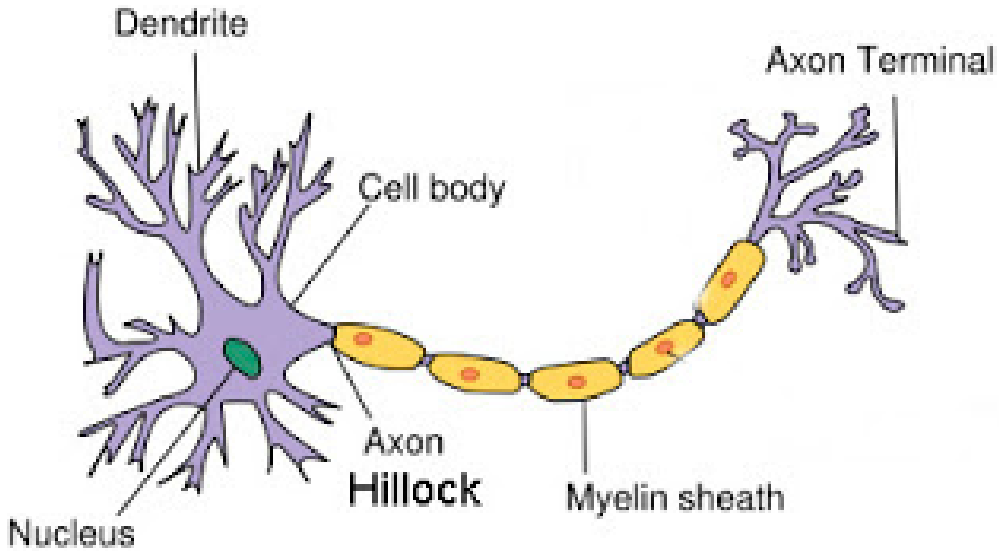


hyperbolic tangent: $\frac{e^x - e^{-x}}{e^x + e^{-x}}$



rectified linear (RELU): if $x > 0$ then x else 0

The cortical neuron



- the dendritic tree of the cell collects inputs from other neurons, that get summed together
- when a triggering threshold is exceeded, the Axon Hillock generate an impulse that get transmitted through the axon to other neurons.

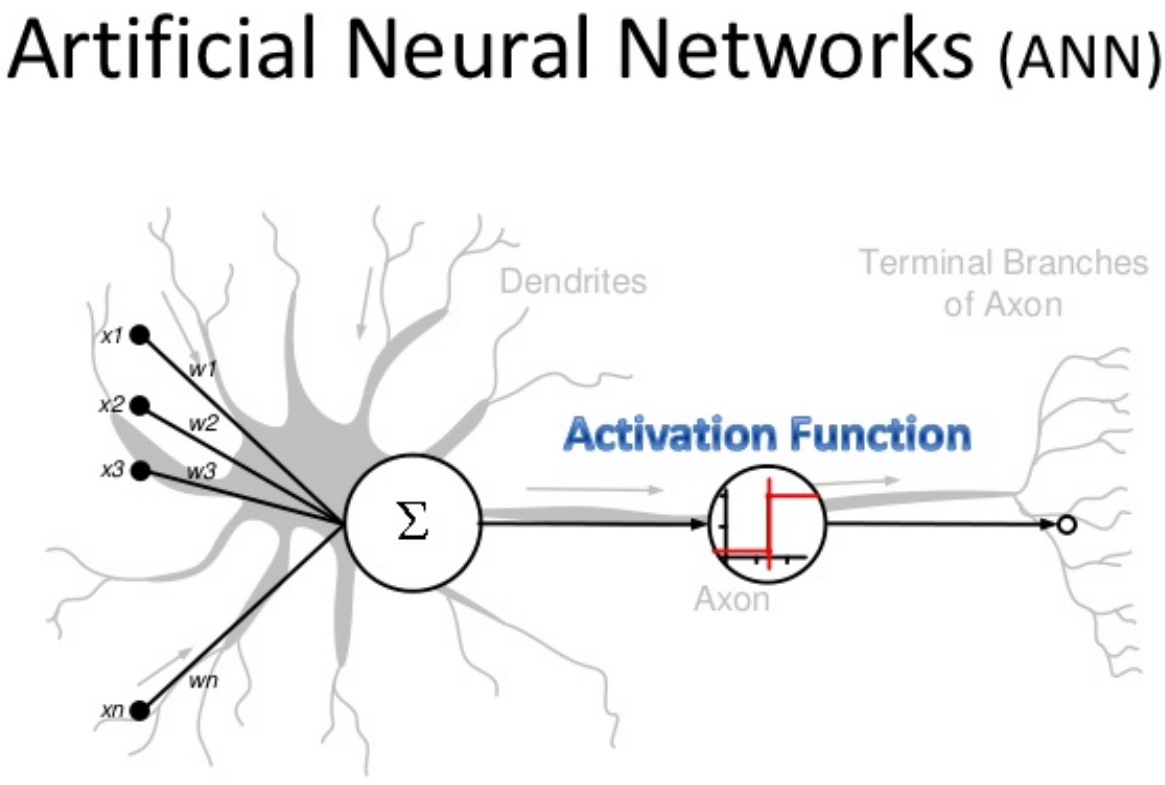
Some figures for human brains

- number of neurons: $\sim 2 \cdot 10^{10}$
- switching time for neuron: $\sim .001$ s. (slow!)
- synapses (connections) per neuron: $\sim 10^{4-5}$
- time to recognize an image: $\sim .1$ s.

not too deep (< 100)

very high parallelism

A comparison with the cortical neuron



Slide credit : Andrew L. Nelson

Motivations behind neural computation

- to understand, via simulation, how the brain works
- to investigate a different paradigm of computation

  very far from traditional programming languages
- to solve practical problems difficult to address with algorithmic techniques

  useful even if the brain works in a different way

Network topologies

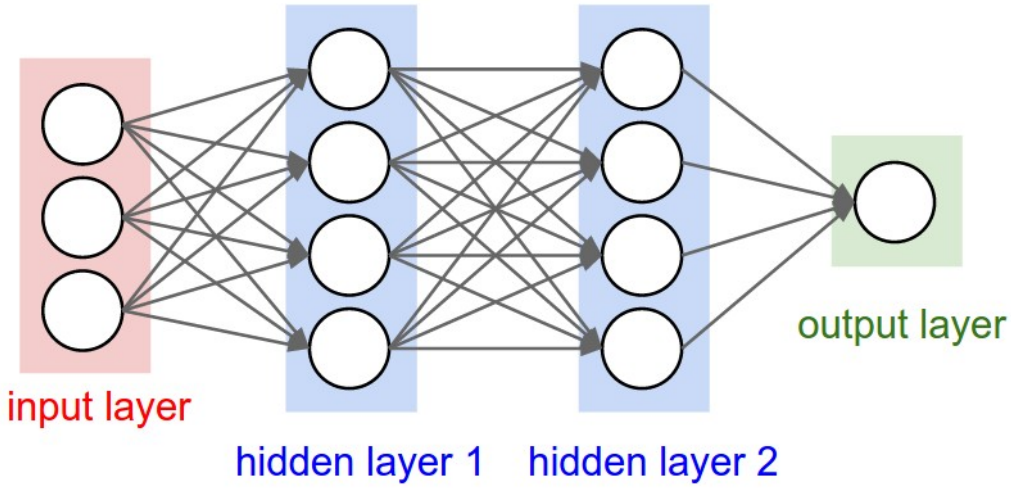
Feed-forward and recurrent networks

If the network is acyclic, it is called a feed-forward network.

If it has cycles it is called recurrent.
(no time to talk about them)

Layers

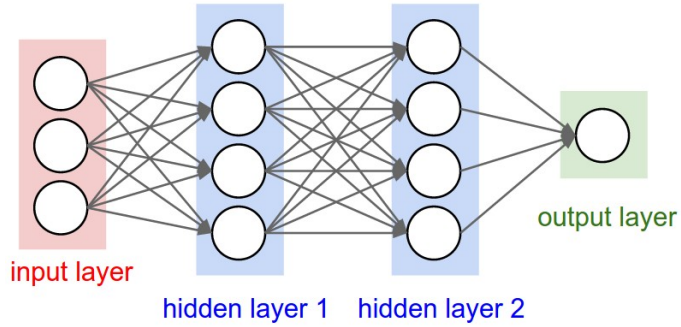
In a feed-forward network, neurons are usually organized in layers.



If there is more than one hidden layer the network is deep, otherwise it is called a shallow network.

Main layers in feed-forward networks: dense layer

Dense layer: each neuron at layer $k-1$ is connected to **each each** neuron at layer k.



A single neuron:

$$I^n \cdot W^n + B^1 = O^1$$

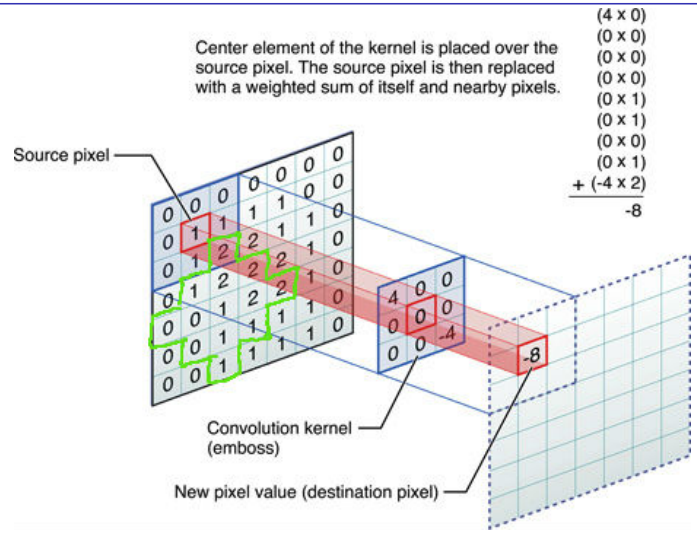
the operation can be vectorized to pruduce m outputs in parallel:

$$I^n \cdot W^{n \times m} + B^m = O^m$$

- dense layers usually work on flat (unstructured) inputs
- the order of elements in input is irrelevant

Main layers in feed-forward networks: convolutional layer

Convolutional layer: each neuron at layer $k-1$ is connected via a parametric **kernel** to a fixed subset of neurons at layer k. The kernel is convolved over the whole previous layer.



1. move the kernel K over a portion M of the input of equal size
2. compute the dot product $M \cdot K$ and possibly add a bias
3. shift the kernel and repeat

The dimension of the output only depends from the number of times the kernel is applied.

Input is structured, and the structure is reflected in the output.

Diving into DL

A first example

[demo]

Understanding DL

- understand the **different layers**, and their purpose
- understand how layers can be organized in **relevant architectures**
- understand the different possible **applications** of DL, and their specific solutions
- understand the main **issues, problems** and **costs**

Frameworks for DL

- TensorFlow/Keras, Google Brain
- PyTorch, Facebook
- MXNET, Apache

We shall mostly use Keras.

Historical remarks - Legacy

Legacy

1958	perceptron
1975	backpropagation
1980	convolutional layers
1992	Max-pooling
1997	LSTM
...	...

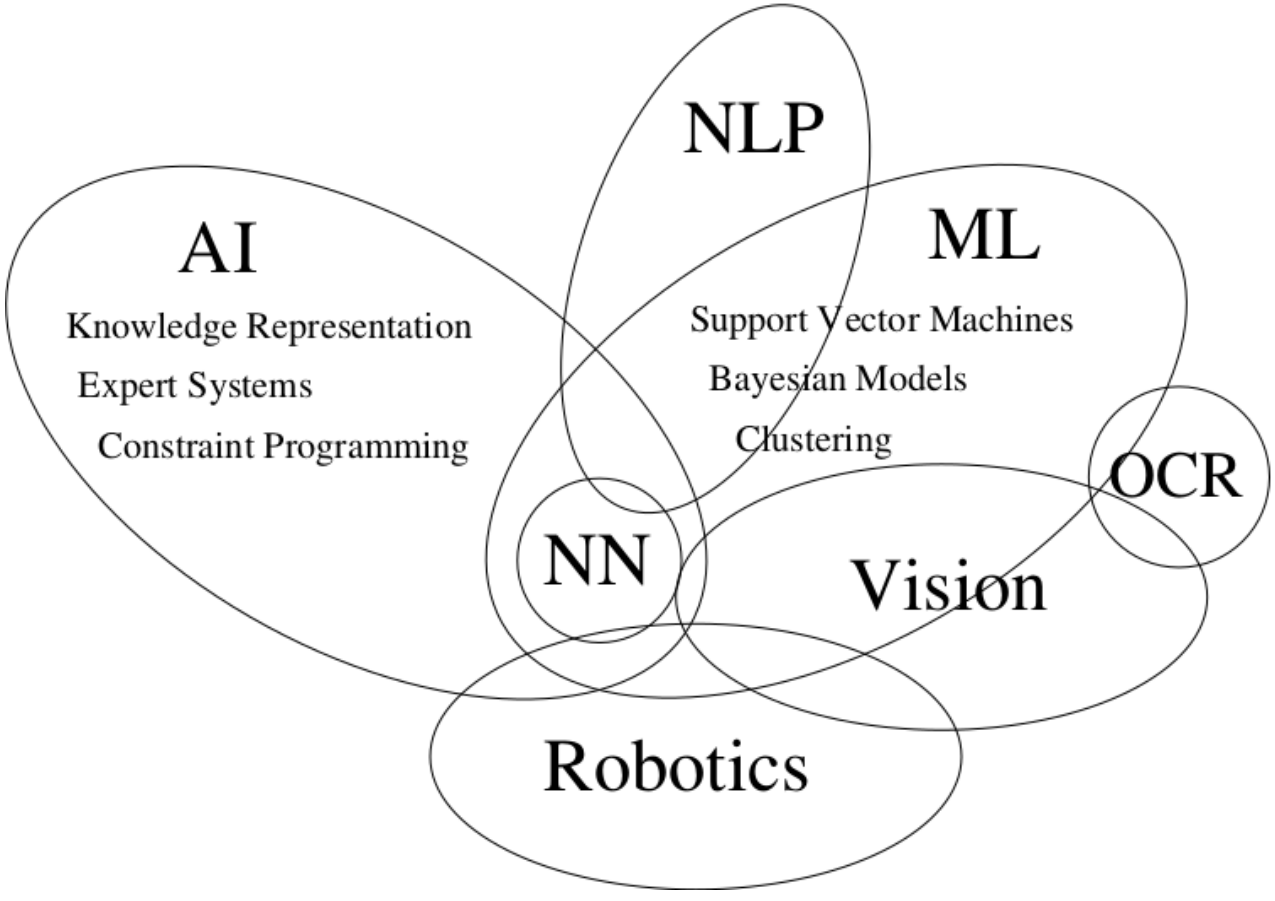
Extremely slow progress

The Deep Learning revolution

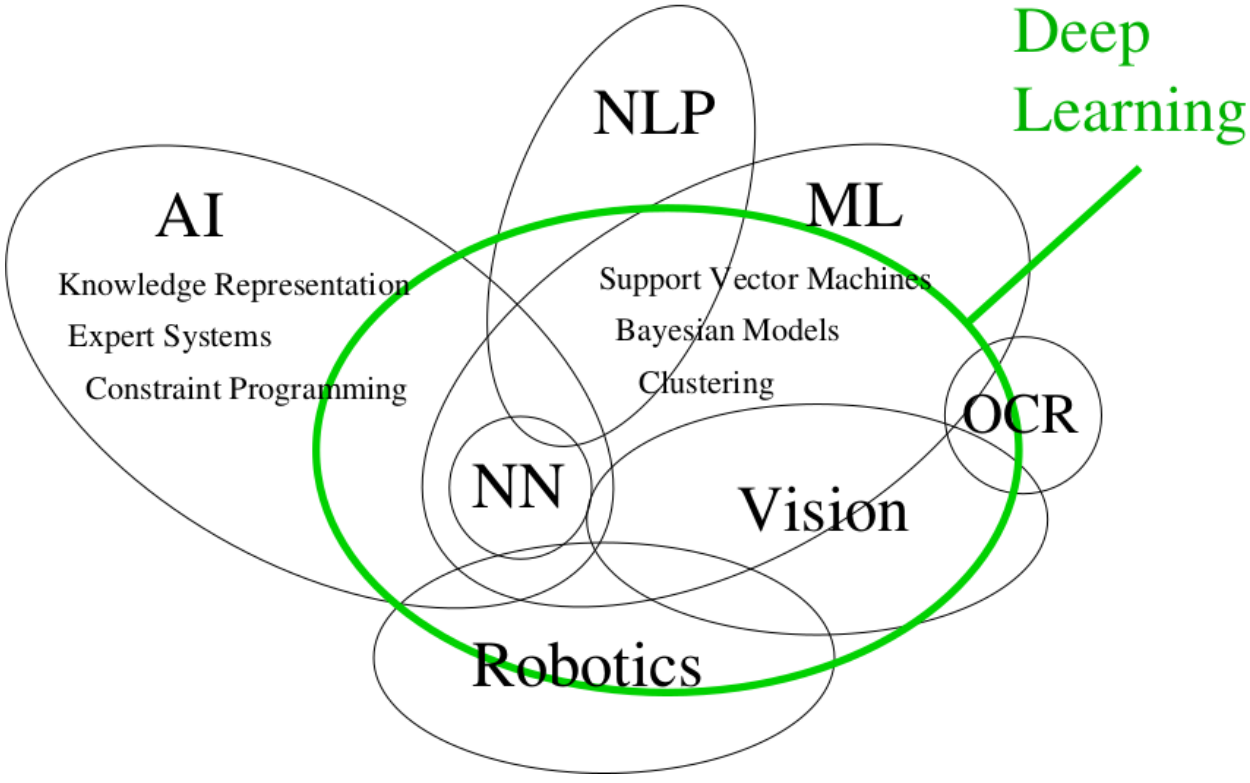
2011	**Google Brain foundation**
2012	ReLU and Dropout
2012	ImageNet Competition
2013	DQN
2014	Inception v1
2015	**Tensorflow release**
2015	Keras release
2015	Batchnormalization
2015	YOLO v1
2015	**OpenAI foundation**
2016	Residual connections
2017	**Pytorch release**
2017	Mask-RCNN
2017	PPO
2018	Transformers
2018	BERT
2018	GPT
2018	Soft Actor Critic
2020	OpenAI Jukebox
2020	Vision Transformer
2021	**MXNet release**
2022	DallE

Just to mention a few milestones ...

The situation at the beginning of the century



The deep learnig era



See my blog for a short historical perspective.

Lesson 2

- Interesting Links
- Successful applications

Books and Tutorials

- Deep Learning, MIT Press. By Y.Bengio, I.Goodfellow and A.Courville.
- Dive into Deep Learning. An interactive book with code, theory and discussions
- Tensorflow Tutorials By Tensorflow org.
- Keras Blog. By F.Chollet.
- Deep Learning Tutorial. LISA lab. University of Montreal.
- Towards Data Science. A Medium publication sharing concepts, ideas, and codes.
- ... so many others

Code and Datasets

The State of the Art site! (papers with code)

- labeled natural images: ImageNet (@Stanford Vision Lab)
  $\approx$ 15M high res color images covering 22K object classes
  ground truth for discrimination, segmentation, borders
- faces
  - CelebA (many facial attributes: hear color, beard, mustaches, age, glasses, ...)
  - Labeled Faces in the Wild (detection/recognition)

Some Dataset repositories

Tensor flow dataset Kaggle Datasets
Amazon Datasets Biomedical challenges . . .

Computational facilities

Training may be expensive.

Some example:

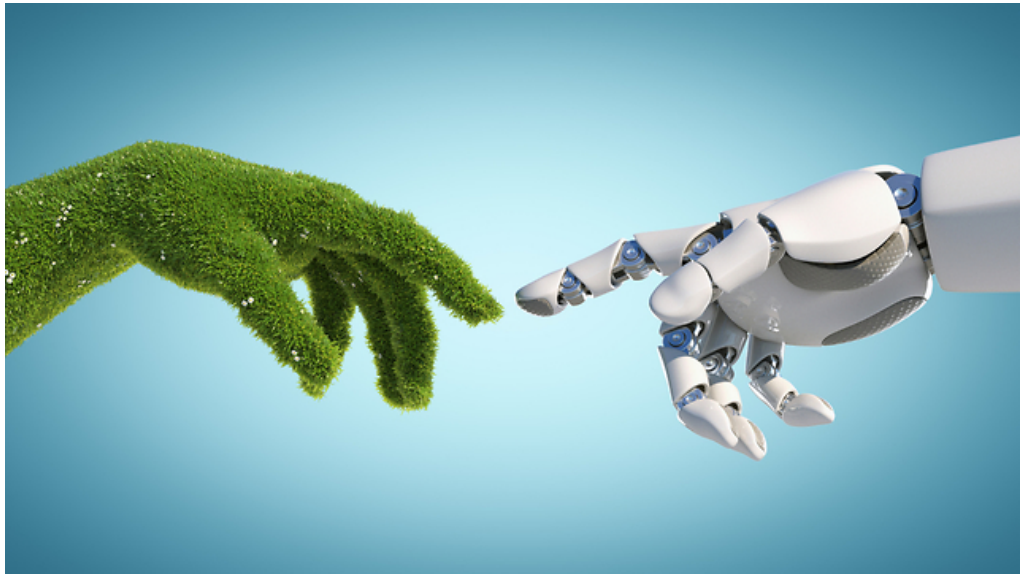
- the hyper-realistic Generative Adversarial Network for face generation by Tesla takes 4 days 8 Tesla V100 GPUs
- training of BERT, a well known generative model for NLP, takes about 96 hours on 64 TPU2 chips.

Major companies offer free computational resources on their clouds:

- Colab, by Google.
- Kaggle
- Amazon Web Services (AWS)
- ...

Green AI

The growing consumption of computational resources is raising social concerns. People is aiming to a more Green AI



Emphasis on efficiency as well as performance.
See this article for a discussion of evaluation metrics.

Examples of successful applications

Examples of successful applications

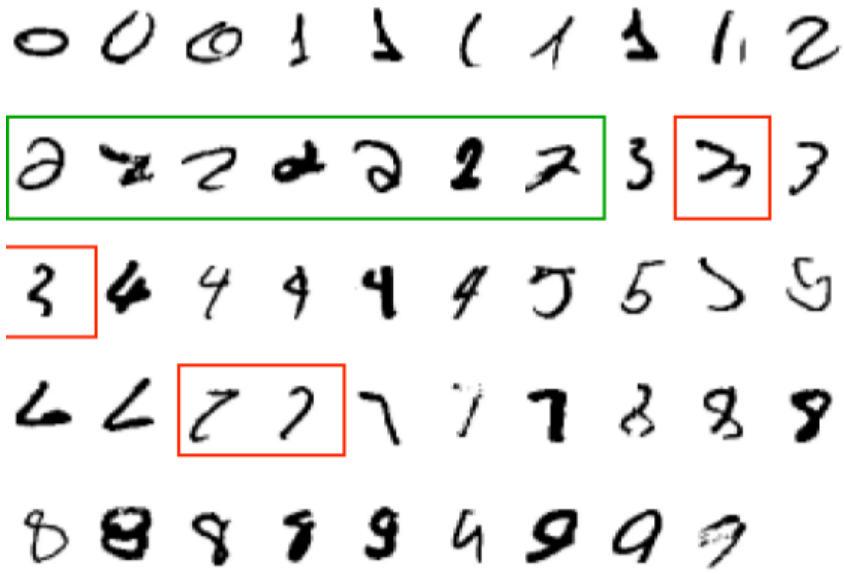
- **Image Processing**
  - Image Classification and Detection
  - Image Segmentation, Scene understanding
  - Style transfer
  - Deep dreams and Inceptionism
- **Natural Language Processing**
  - Speech Recognition
  - Text processing (translation, summarization, generation, . . .)
- **Generative modeling** (GANs, VAEs, Cycle Gans)
- **Deep Reinforcement Learning**
  - Robot navigation and autonomous driving
  - Model-free learning

Image Processing

MNIST

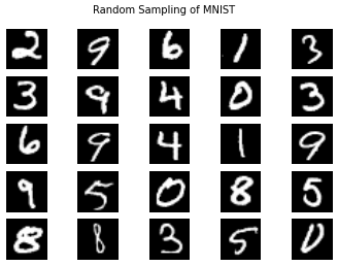
Modified National Institute of Standards and Technology database

- grayscale images of handwritten digits, 28×28 pixels each
- 60,000 training images and 10,000 testing images



MNIST

A comparison of different techniques



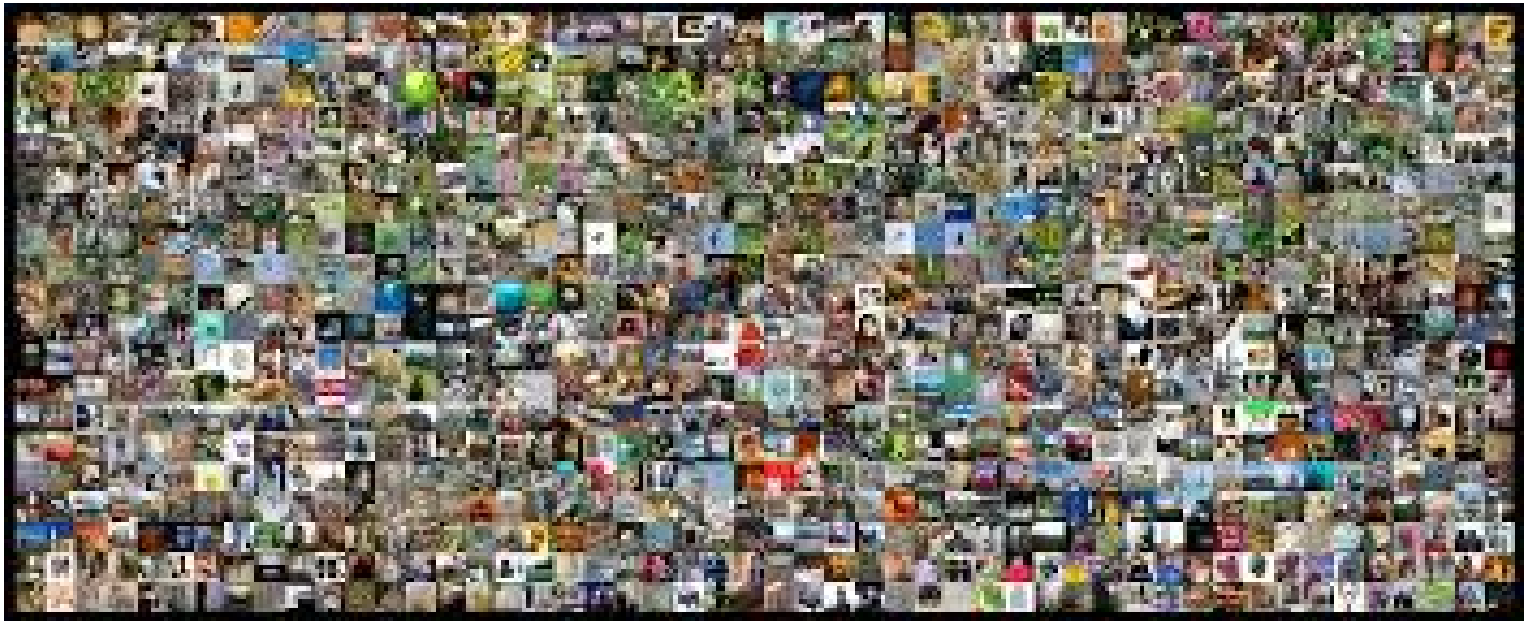
Classifier	**Error rate**
Linear classifier	7.6
K-Nearest Neighbors	0.52
SVM	0.56
Shallow neural network	1.6
Deep neural network	0.35
Convolutional neural network	0.21

See LeCun's page the mnist database for more data.

ImageNet

ImageNet (@Stanford Vision Lab)

- high resolution color images covering 22K object classes
- over 15 million labeled images from the web



ImageNet competition

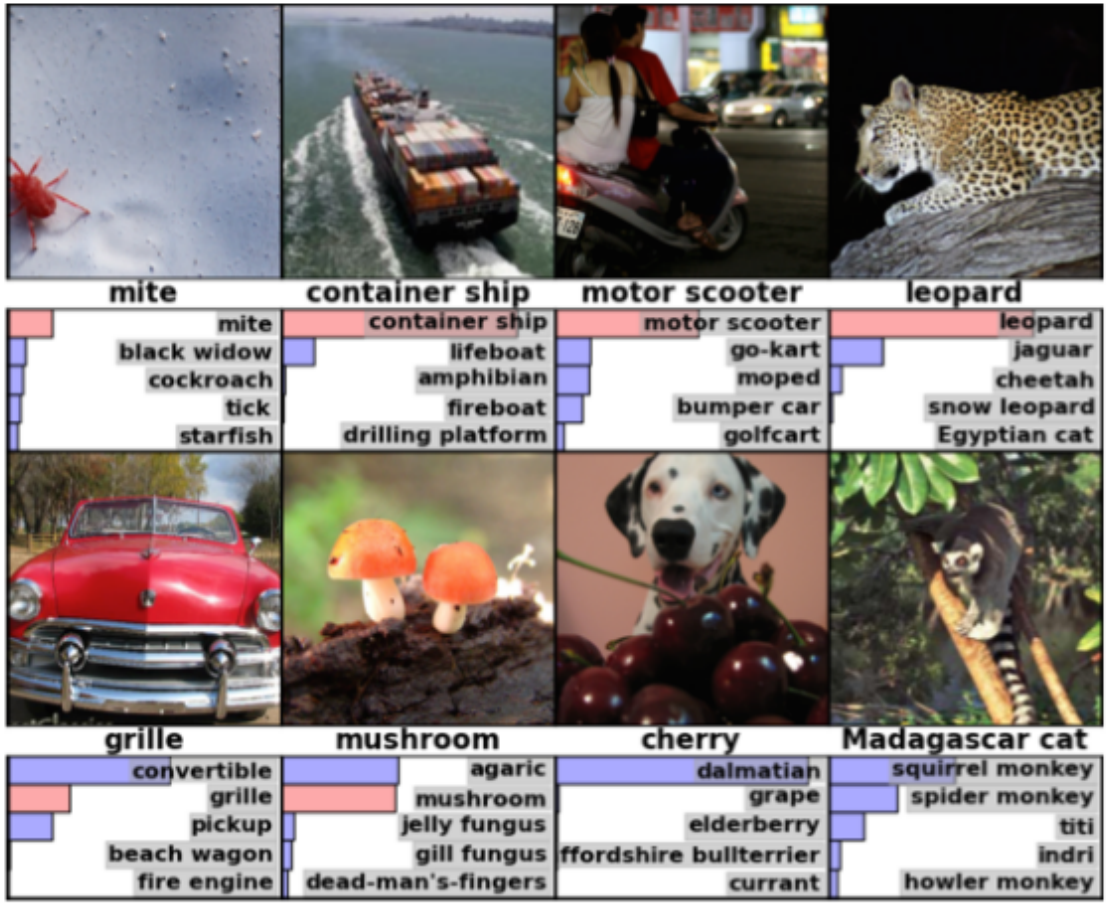
Annual competition of image classification: 2010-2017.

- 1.2 Million images, covering 1K different categories
- make five guesses about image label, ordered by confidence

EntleBucher

Appenzeller

ImageNet samples



ImageNet results

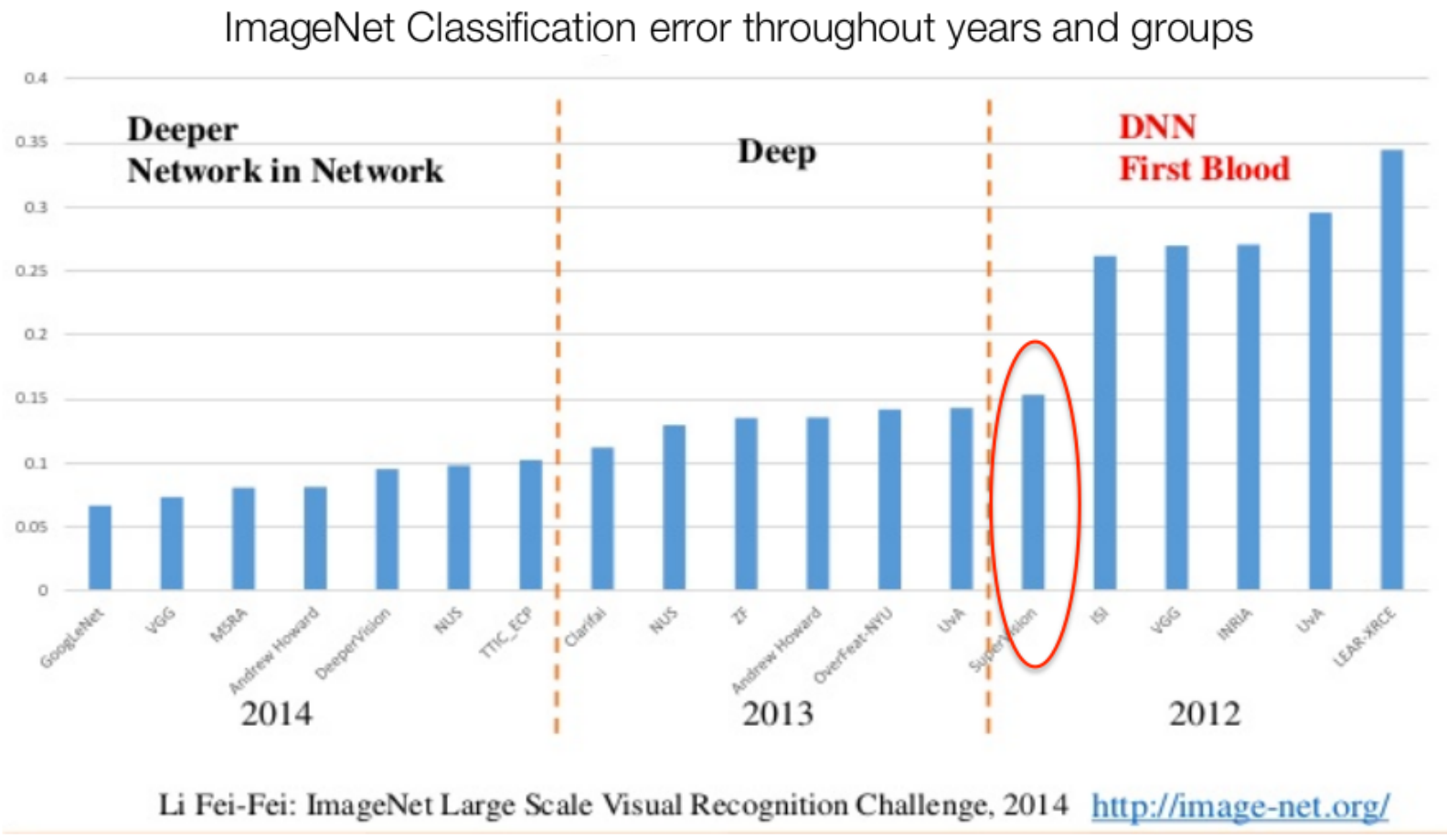
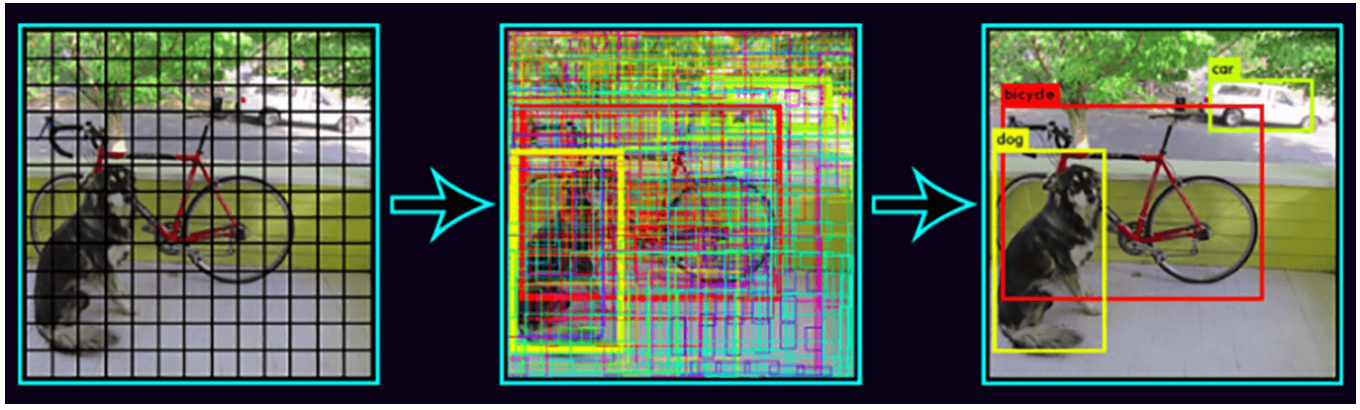


Image Detection

YOLO: Real-Time Object Detection



You only look once (YOLO) is a state-of-the-art, real-time object detection system. On a Pascal Titan X it processes images at 30 FPS and has a mAP of 57.9% on COCO test-dev.

First release in 2016, now at version 7.

Image Segmentation - Scene understanding

Video-to-Video Synthesis

Mimicking style

A neural algorithm of artistic style

L.A. Gatys, A.S. Ecker, M. Bethge

Change the style of an image, preserving the content.

Deep dreams

Source: Google Inceptionism

Visit Deep dreams generator

Many videos on youtube (e.g. this)

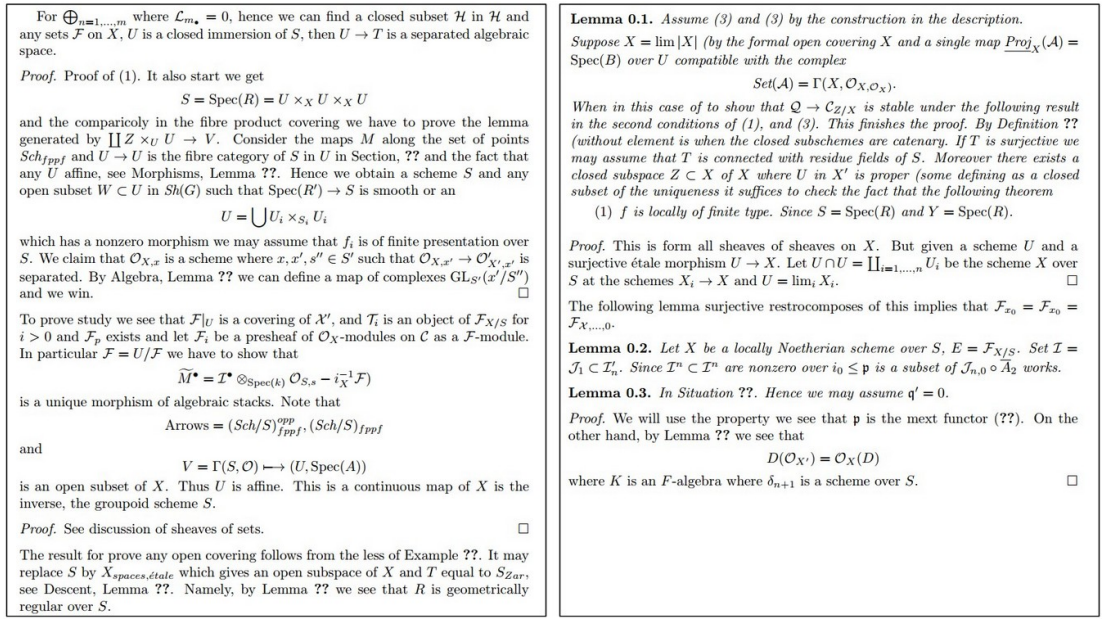
Natural Language Processing

Language Modeling

Predict the next character in a document (self-supervised)

First attempts with RNN (LSTM).

See Andrej Karpathy's blog The Unreasonable Effectiveness of Recurrent Neural Networks (old but still inspiring)



For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the compariconly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, **??** and the fact that any U affine, see Morphisms, Lemma **??**. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma **??** we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example **??**. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma **??**. Namely, by Lemma **??** we see that R is geometrically regular over S.

Lemma 0.1. *Assume (3) and (3) by the construction in the description.*

Suppose $X = \lim |X|$ *(by the formal open covering* X *and a single map* $\underline{Proj}_X(\mathcal{A}) =$ $\mathrm{Spec}(B)$ *over* U *compatible with the complex*

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ *is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition* **??** *(without element is when the closed subschemes are catenary. If* T *is surjective we may assume that* T *is connected with residue fields of* S*. Moreover there exists a closed subspace* $Z \subset X$ *of* X *where* U *in* X' *is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem*

(1) f *is locally of finite type. Since* $S = \mathrm{Spec}(R)$ *and* $Y = \mathrm{Spec}(R)$.

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\ldots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\ldots,0}$.

Lemma 0.2. *Let* X *be a locally Noetherian scheme over* S, $E = \mathcal{F}_{X/S}$. *Set* $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. *Since* $\mathcal{I}^n \subset \mathcal{I}^n$ *are nonzero over* $i_0 \leq \mathfrak{p}$ *is a subset of* $\mathcal{J}_{n,0} \circ \bar{A}_2$ *works.*

Lemma 0.3. *In Situation* **??**. *Hence we may assume* $\mathfrak{q}' = 0$.

Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (**??**). On the other hand, by Lemma **??** we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

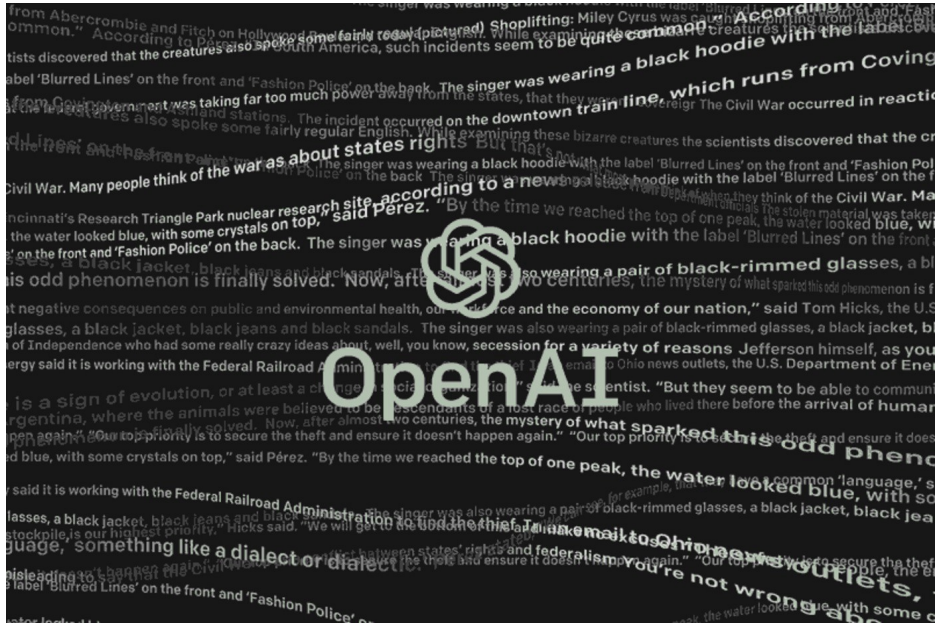
where K is an F-algebra where δ_{n+1} is a scheme over S. □

Examples of fake algebraic documents generated by a RNN.

Transformers

RNNs have been replaced by **Transformers**, based on a mechanism called **attention**

See Bert, Albert, GPT, ...



GPT2 is a huge model, with 1.2 billion parameters, trained over 8 million web pages.

Other applications in NLP

- Sentiment analysis. Classify a document according to its "polarity"
- Machine Translation
- Text summarization/completion
- Text Generation: a truly generative task
- Speech recognition
- Dialog Systems - Chatboxes

Generative Modeling

Generative Modeling

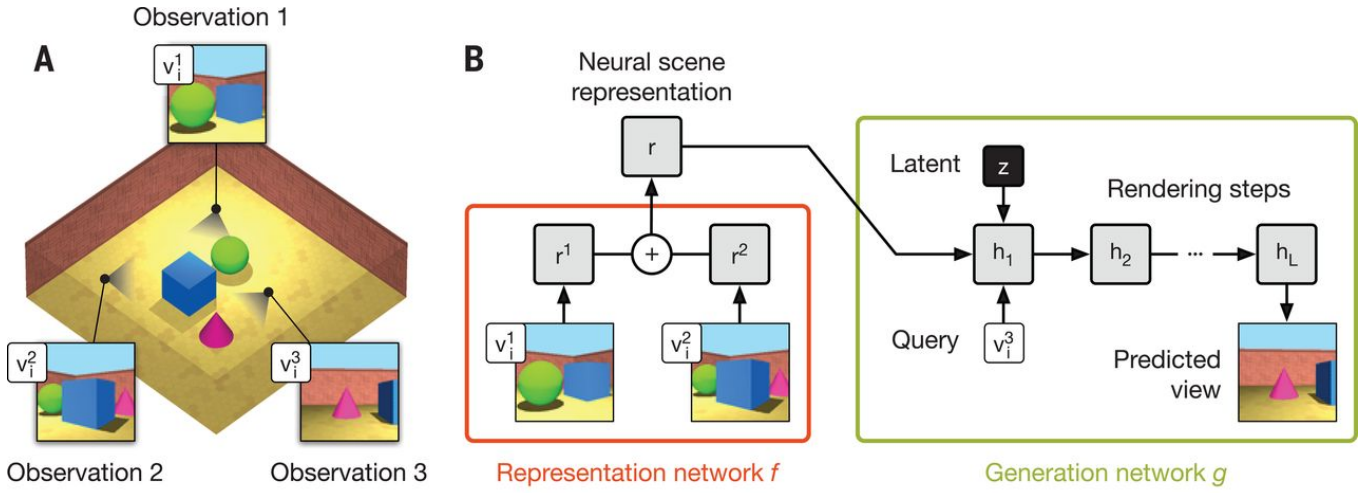
Goal: Generate new samples similar to training data.

Face generation video by Nvidia

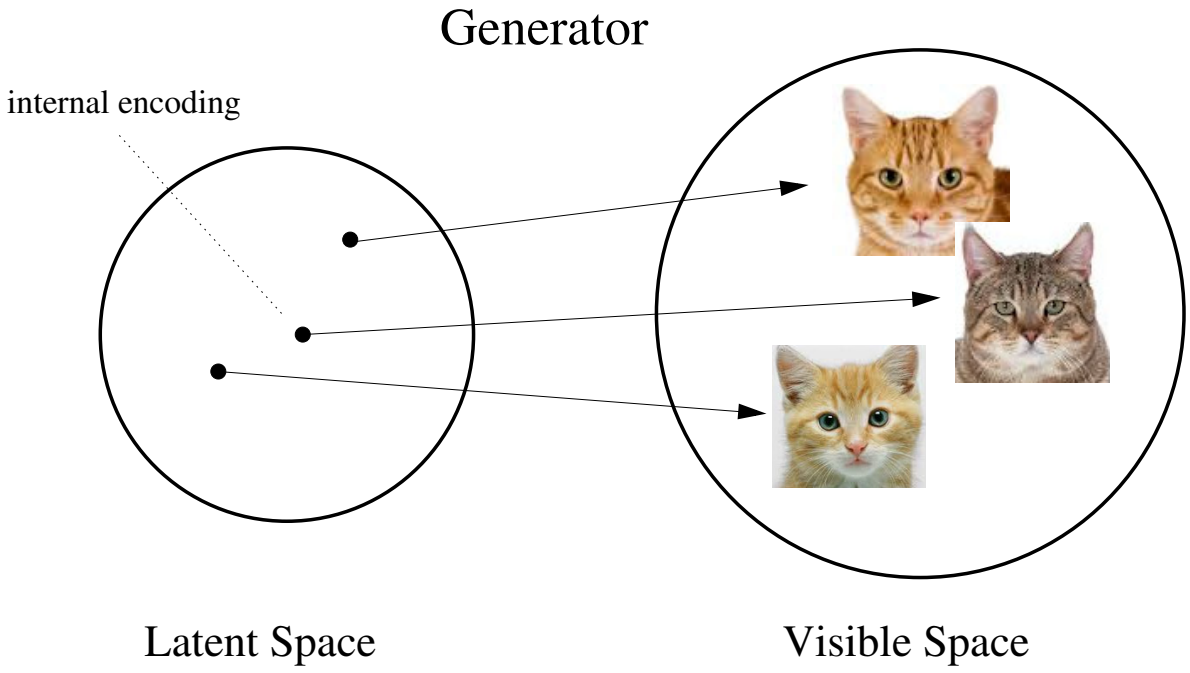
Scene representation and rendering

Neural scene representation and rendering (VAE)

Work published on Science (June 2018)



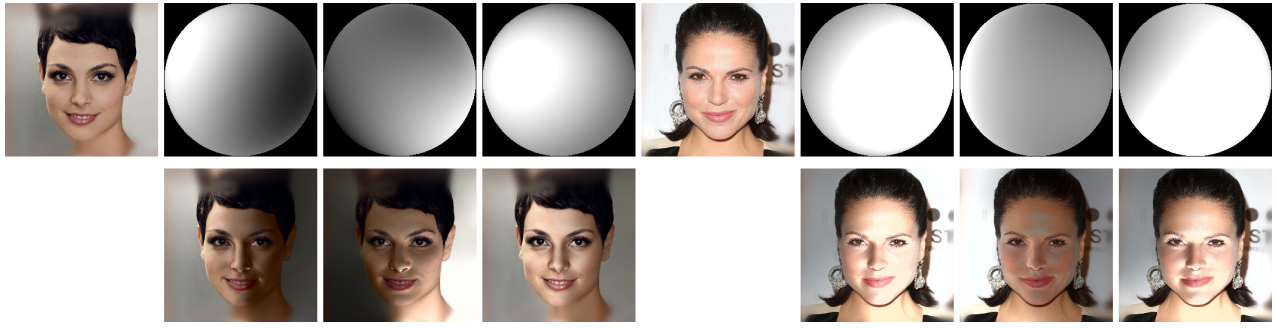
Latent space



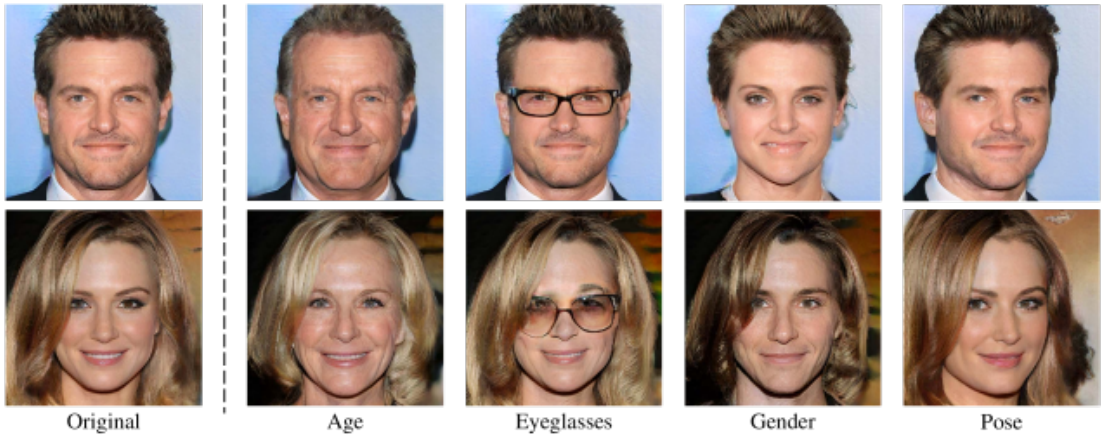
Suggested reading:

Comparing the latent space of generative models

Conditional generation

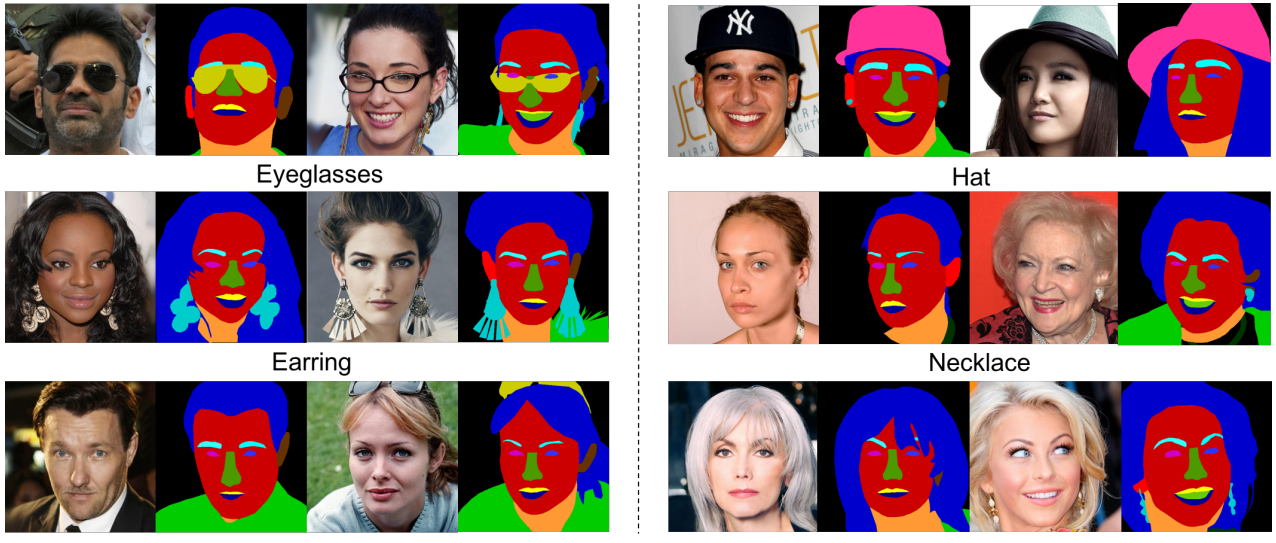


Deep Single Image Portrait Relighting



Interpreting the Latent Space of GANs for Semantic Face Editing

Conditional generation



MaskGAN: Towards Diverse and Interactive Facial Image Manipulation

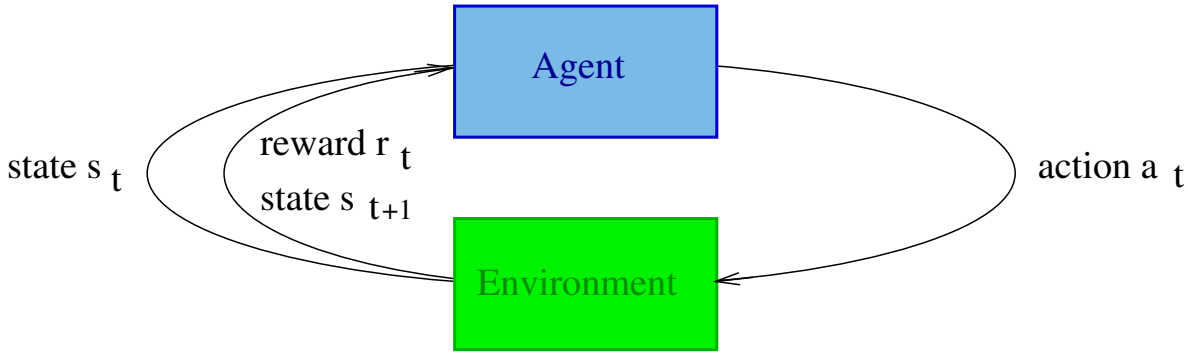
Dall·E - OpenAI

Dall·E is a new AI system that can create realistic images and art from a description in natural language.

Reinforcement Learning

Reinforcement Learning

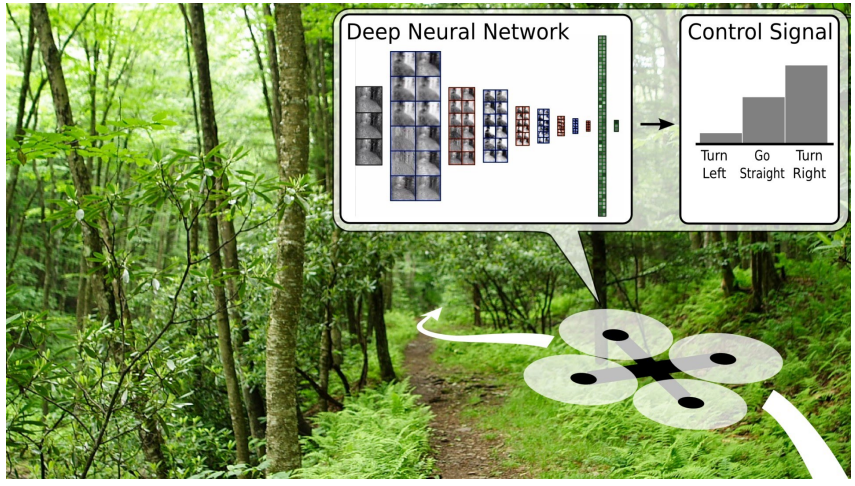
Problems involving an agent interacting with an environment, which provides numeric rewards



Goal: learn how to take actions in order to maximize the future cumulative reward.

Robot navigation

Quadcopter Navigation in the Forest using Deep Neural Networks



Robotics and Perception Group, University of Zurich, Switzerland &
Institute for Artificial Intelligence (IDSIA), Lugano Switzerland

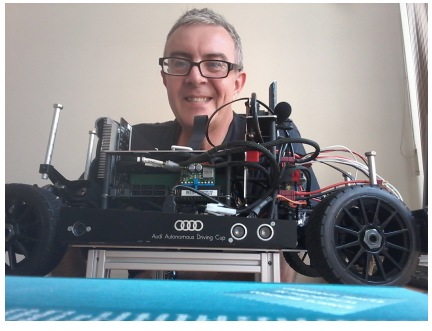
Based on Imitation Learning

Autonomous driving

Develop intelligent, fully automatic driving functions for vehicles.

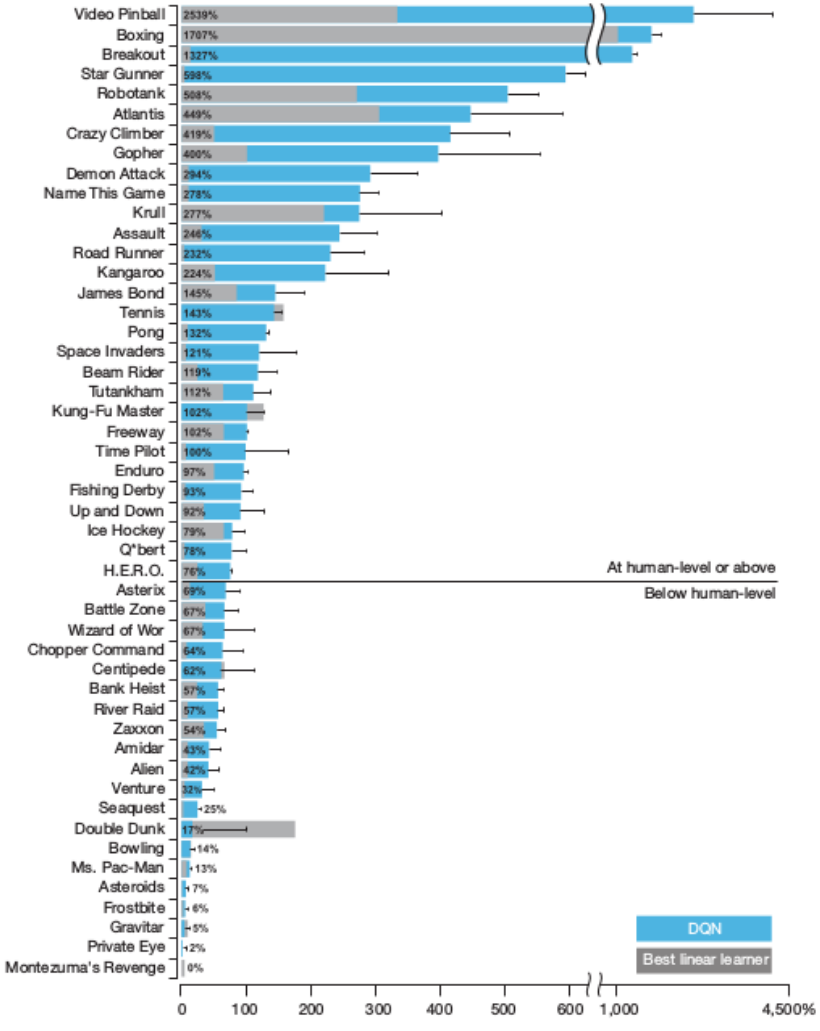
Merging of signals collected by different sensors (camera, lidar, sonar, dots). Needs to accurately evaluate distances and speeds.

Turn observations into actions.



Several competitions around. We took part to the 2018 Audi Autonomous Driving Cup

Game Simulation



Google DeepMind's system playing Atari games (2013)

The same network architecture was applied to all games

End-to-end training starting from screen frames

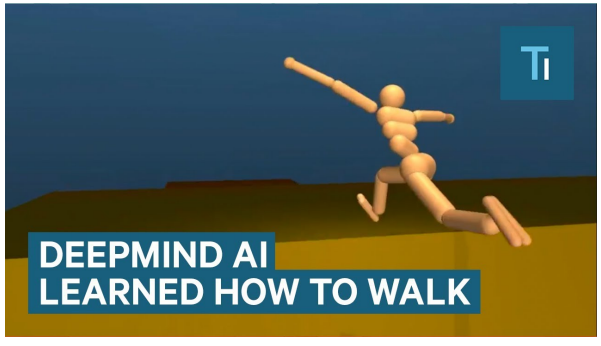
Works well for reactive games; problems with planning...

but see An investigation of Model-Free planning (ICML 2019)

Open AI-gym

OpenAI Gym is a Gym is a toolkit for developing and comparing reinforcement learning algorithms (DQN, A3C, A2C, Acer, PPO, . . .).

It offers many learning scenarios, from walking to playing games like Pong or Pinball, as well as other classical physical "equilibrium" problems.

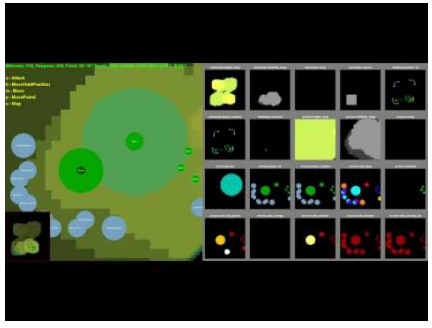


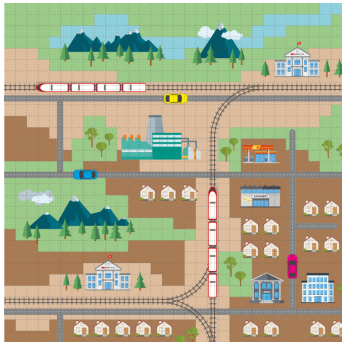
Multi agent DRL

It requires interaction and cooperation of multiple agents.

Examples:

StarCraft II: a RL environment based on the game StarCraft II. The environment consists of three sub-components: a Linux StarCraft II binary, the StarCraft II API providing programmatic control over the game, and a python wrapper over the API called PyC2.





Flatland: a train rescheduling problem on a complex grid world environment.

Flatland is organized every year by Alcrowd in collaboration with the Swiss Federal Railways, SBB

Expressiveness & Training

Next argument

Expressiveness

Expressiveness

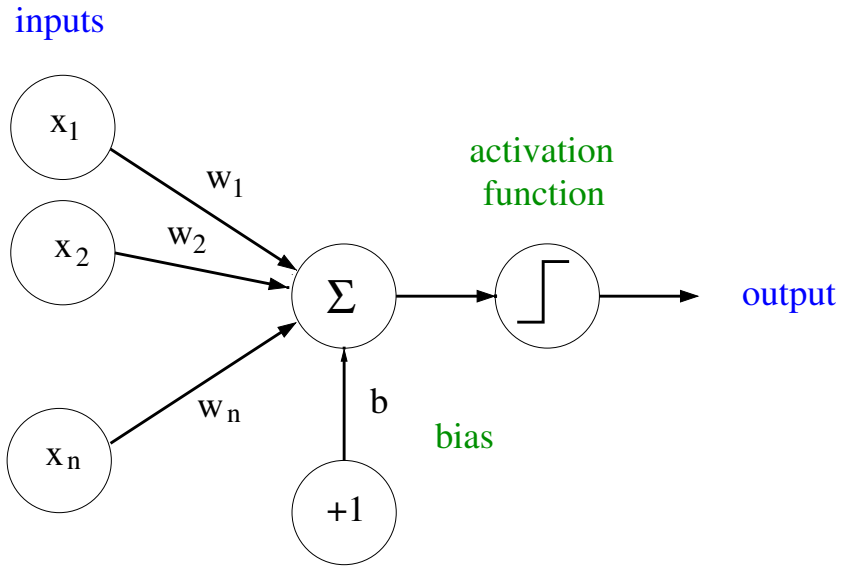
Can we compute any function by means of a Neural Network?

Do we really need **deep** networks?

Can we compute any function with a single neuron?

Single layer case: the perceptron

Binary threshold:



$$output = \begin{cases} 1 & if \sum_i w_i x_i + b \geq 0 \\ 0 & otherwise \end{cases} \qquad output = \begin{cases} 1 & if \sum_i w_i x_i \geq -b \\ 0 & otherwise \end{cases}$$

Remark: the bias set the position of threshold.

Hyperplanes

The set of points

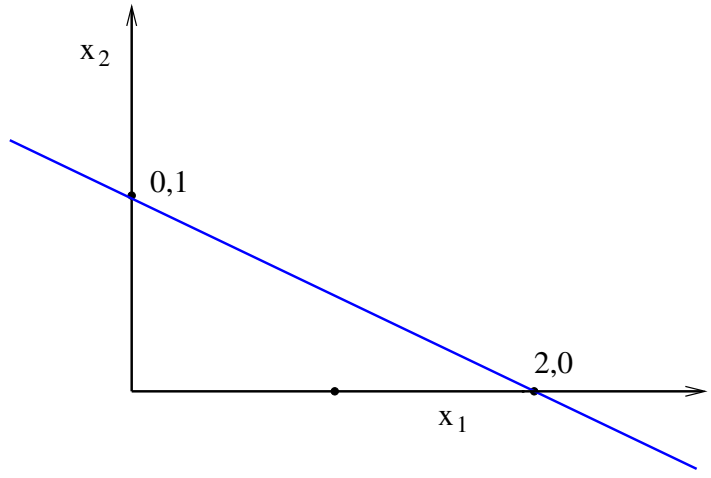
$$\sum_i w_i x_i + b = 0$$

defines a hyperplane in the space of the variables x_i

Example:

$$-\frac{1}{2}x_1 + x_2 + 1 = 0$$

is a line in the bidimensional space



Hyperplanes

The hyperplane

$$\sum_i w_i x_i + b = 0$$

divides the space in two parts: to one of them (above the line) the perceptron gives value 1, to the other (below the line) value 0.

"above" and "below" can be inverted by just inverting parameters:

$$\sum_i w_i x_i + b \leq 0 \iff \sum_i -w_i x_i - b \geq 0$$

Computing logical connectives: NAND

Can we implement this function (NAND) with a perceptron?

x_1	x_2	*output*
0	0	1
0	1	1
1	0	1
1	1	0

Can we find two weights w_1 and w_2 and a bias b such that

$$nand(x_1, x_2) = \begin{cases} 1 & if \sum_i w_i x_i + b' \geq 0 \\ 0 & otherwise \end{cases}$$

Computing logical connectives: NAND

Can we implement this function (NAND) with a perceptron?

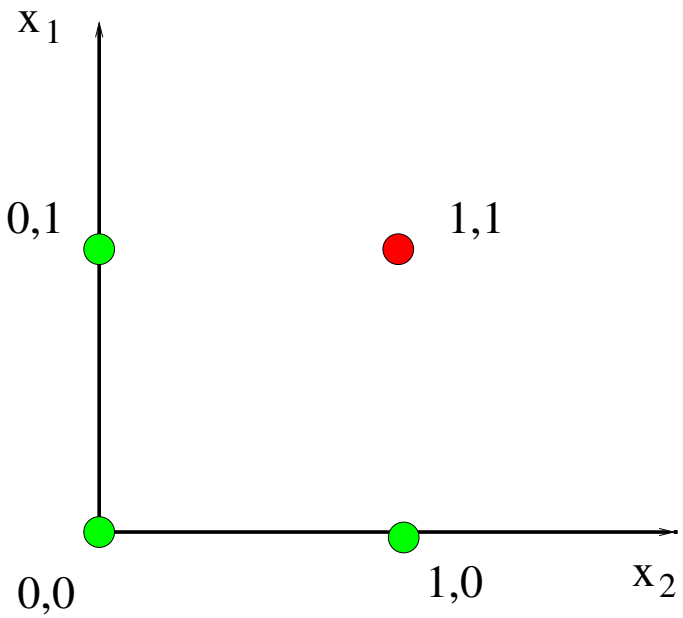
x_1	x_2	*output*
0	0	1
0	1	1
1	0	1
1	1	0

Can we find two weights w_1 and w_2 and a bias b such that

$$nand(x_1, x_2) = \begin{cases} 1 & if \sum_i w_i x_i + b' \geq 0 \\ 0 & otherwise \end{cases}$$

Graphical representation

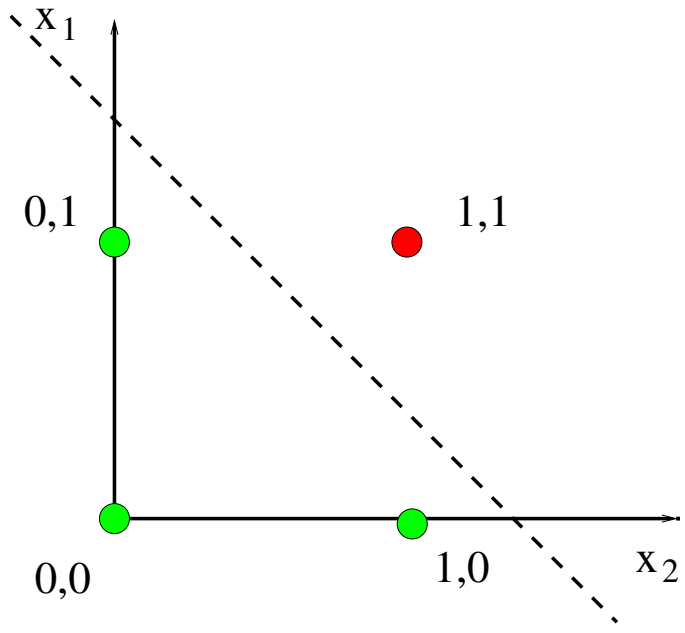
Same as asking:
can we draw a straight line to separate green and red points?



NAND

Lines, planes, hyperplanes

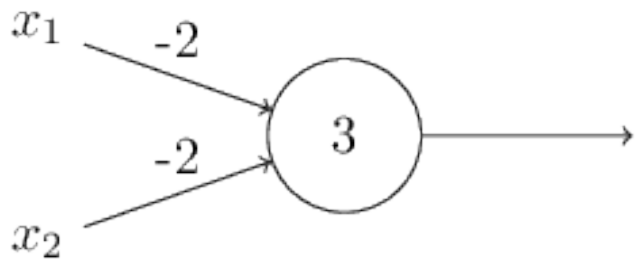
Yes!



NAND

line equation: $1.5 - x_1 - x_2 = 0$ or $3 - 2x_1 - 2x_2 = 0$

The NAND-perpceptron

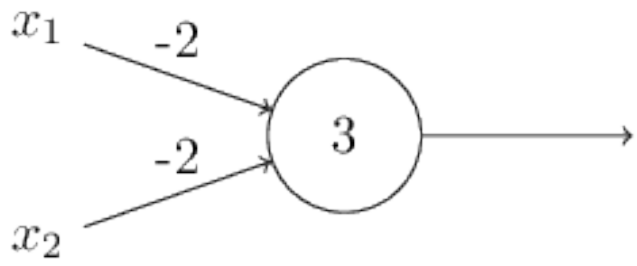


$$output = \begin{cases} 1 & if\ -2x_1 - 2x_2 + 3 \geq 0 \\ 0 & otherwise \end{cases}$$

x_1	x_2	*output*
0	0	1
0	1	1
1	0	1
1	1	0

Can we compute any logical circuit with a perceptron?

The NAND-perpceptron



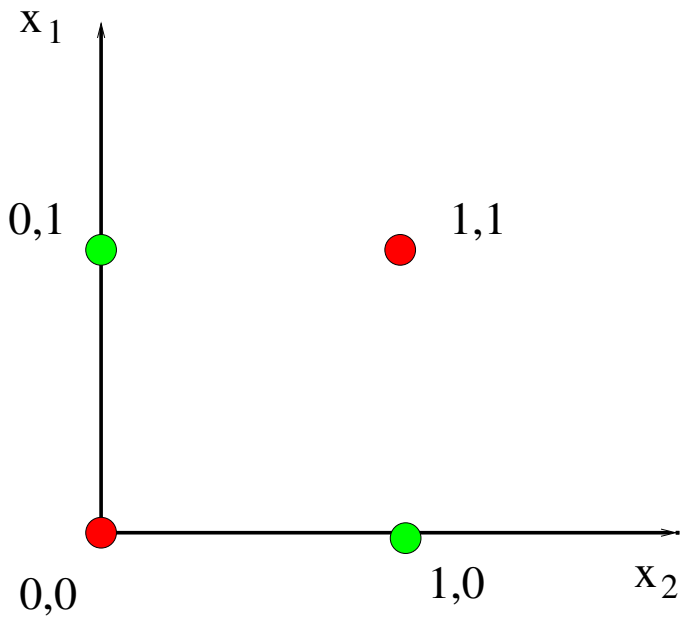
$$output = \begin{cases} 1 & if\ -2x_1 - 2x_2 + 3 \geq 0 \\ 0 & otherwise \end{cases}$$

x_1	x_2	*output*
0	0	1
0	1	1
1	0	1
1	1	0

Can we compute any logical circuit with a perceptron?

The XOR case

Can we draw a straight line separating red and green points?

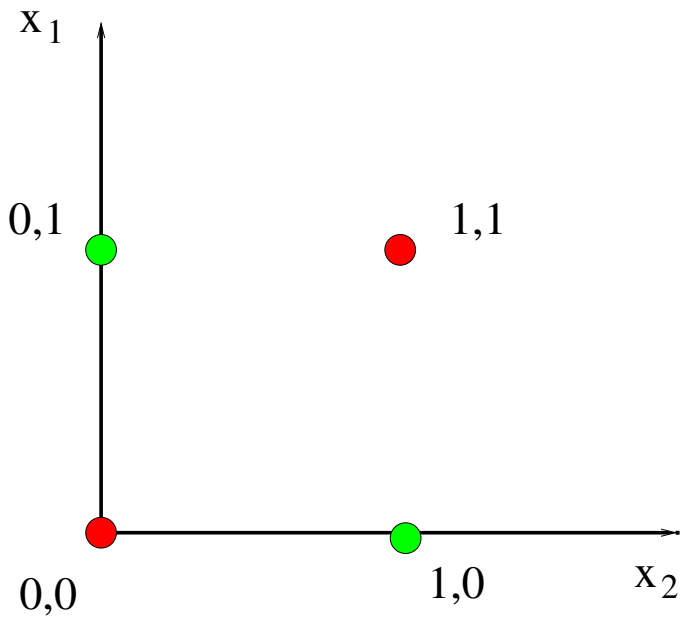


No way!

Single layer perceptrons are not complete!

The XOR case

Can we draw a straight line separating red and green points?



No way!

Single layer perceptrons are not complete!

Multi-layer perceptrons

Question:

- we know we can compute nand with a perceptron
- we know that nand is logically complete
  (i.e. we can compute any connective with nands)

so:

why perceptrons are not complete?

answer:

because we need to compose them and consider
Multi-layer perceptrons

Multi-layer perceptrons

Question:

- we know we can compute nand with a perceptron
- we know that nand is logically complete
  (i.e. we can compute any connective with nands)

so:

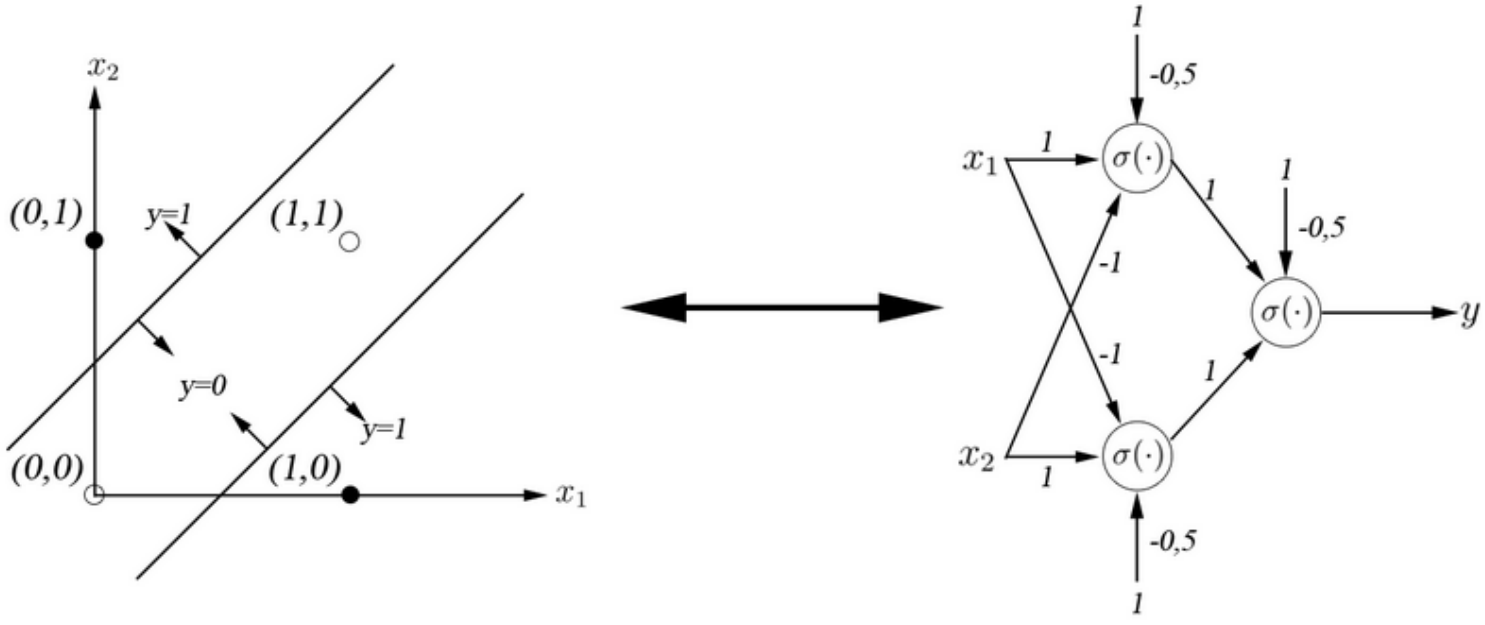
why perceptrons are not complete?

answer:

because we need to compose them and consider
Multi-layer perceptrons

Example: Multi-layer perceptron for XOR

Can we compute XOR by **stacking** perceptrons?



Multilayer perceptrons are logically complete!

Important Points

- shallow nets are already **complete**

  Why going for deep networks?

  With deep nets, the same function may be computed with **less neural units** (Cohen, et al.)

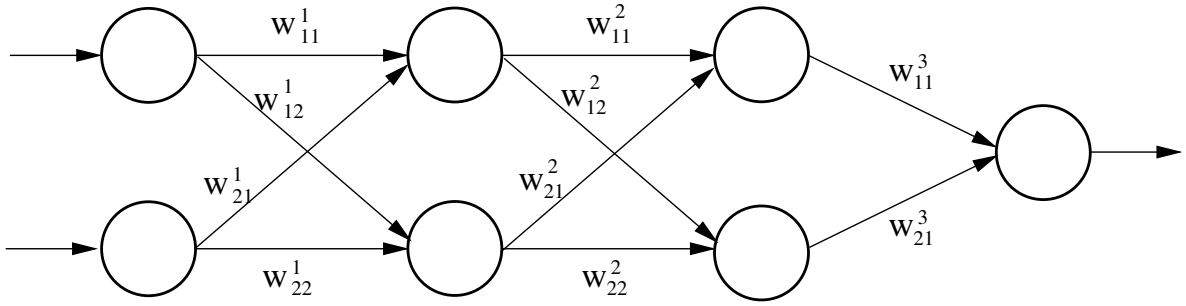
- Activation functions play an **essential role**, since they are the only source of nonlinearity, and hence of the expressiveness of NNs.

  Composing linear layers not separated by nonlinear activations **makes no sense**!

Training

Current loss

Suppose to have a neural network with some configurations of the parameters θ.

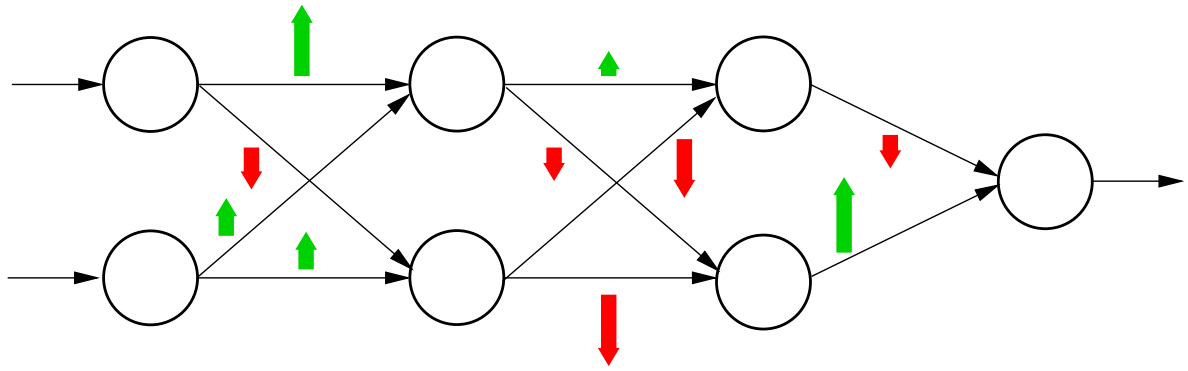


We can take a **batch** of data, pass them (in parallel) through the network, compute the output, and evaluate the current loss relative to θ.

This is a **forward pass** through the network.

Parameter updating

Next, we would like to adjust the parameters in such a way to decrease the current loss.

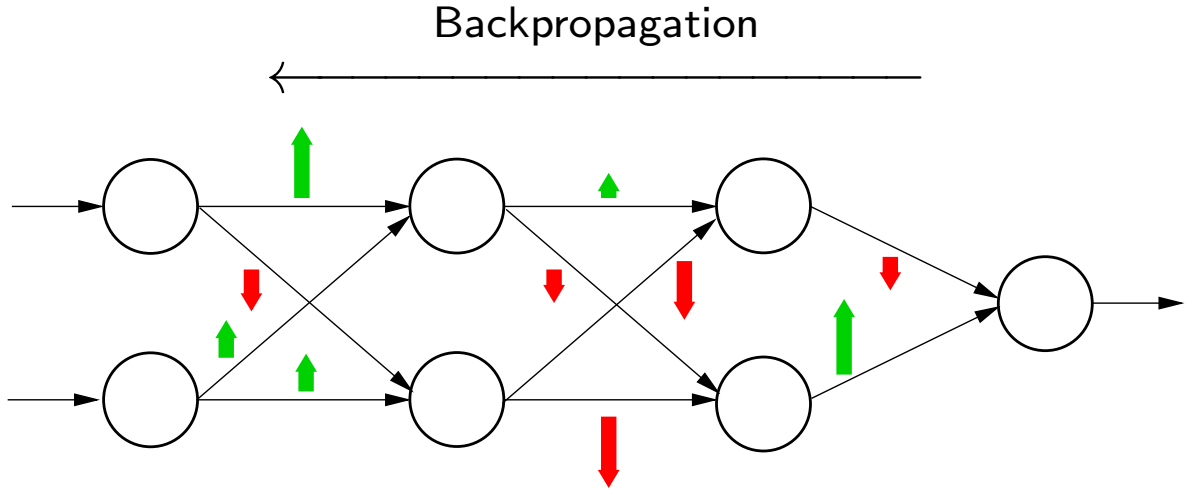


Each parameter should deserve a different adjustment, some of them positive, other negative.

The mathematical tool that allows us to establish in which way parameters should be updated is the **gradient**: a vector of **partial derivatives**.

Backpropagation

The gradient is computed **backward**, **backpropagating** the loss to all neurons inside a networks, and their connections.



This is a **backward pass** through the network.

The algorithm for computing parameters updates is known as **backpropagation algorithm**.

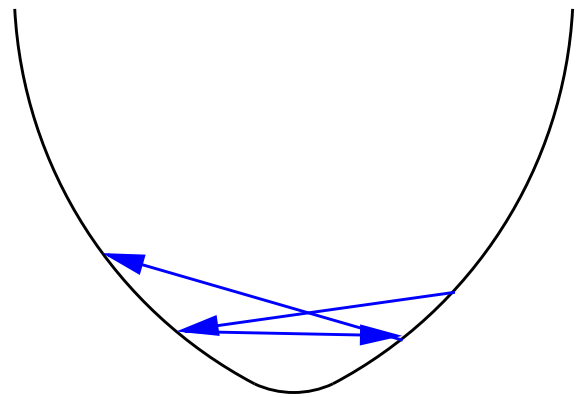
Learning rate

The backpropagation algorithms only gives a **direction** in which gradients should be updated.

The actual amount of the update is obtained by multiplication with a scalar hyperparameter (fixed externally, not learned) called **learning rate**.

Increasing the learning rate can make training faster, but it reduces the accuracy of the result.

If we make large steps nearby the optimum, we can miss it.



Optimizer

Many techniques can be used to **tune** the learning rate during training.

The tool in charge of governing the gradient descent technique - possibly dynamically adapting the learning rate - is the so called **optimizer** (e.g. Adam, in our example).

Many possibilites:
- Use a fixed learning rate
- adapt the global learning rate during time
- adapt the learning rate on each connection separately
- use the so called *momentum*
- ...

suggested lecture: Geoffrey Hinton's lecture

Varying the batchsize

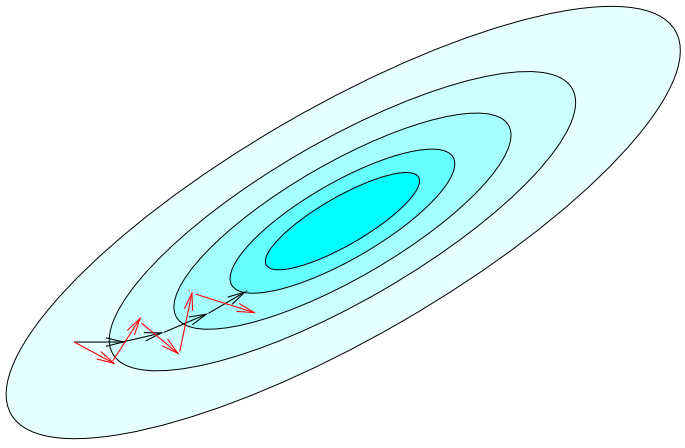
Parameters are updated **every time** a batch is processed.

During an epoch, we make a total number of updates equal to the size of the training set divided by the batchsize.

If we decrease the batchsize we update more frequently (that is good), but updates are less accurate, since we are backpropagating form a loss relative to very specific data.

Conversely, if we increase the batchsize, updates grow in accuracy (the ideal would be to compute them on the whole training set - **fullbatch**) but training can be slow, since parameters are too rarely updated.

Fullbatch, Online and MiniBatch



Fullbatch (all training samples): the gradient points to the direction of steepest descent on the error surface (perpendicular to contour lines of the error surface)

Online (one sample at a time) gradient zig-zags around the direction of the steepest descent.

Minibatch (random subset of training samples): a good compromise.

The Backpropagation algorithm (and its problems)

Computing the gradient

A neural network computes a complex function resulting from the composition of many neural layers. How can we compute the gradient w.r.t. a specific parameter (weight) of the net?

We need a mathematical rule know as the chain rule (for derivatives).

The chain rule

Given two derivable functions f, g with derivatives f' and g', the derivative of the composite function $h(x) = f(g(x))$ is

$$h'(x) = f'(g(x)) * g'(x))$$

Equivalently, letting $y = g(x)$,

$$h'(x) = f'(g(x)) * g'(x)) = f'(y) * g'(x)$$

The derivative of a **composition** of a sequence of functions is the **product** of the derivatives of the individual functions.

QUESTION: why binary thresholding is not a good activation function for backpropagation?

Backpropagation rules in vectorial notation

Given some error function E (e.g. euclidean distance) let us define the error derivative at l as the following vector of partial derivatives:

$$\delta^l = \frac{\partial E}{\partial z^l}$$

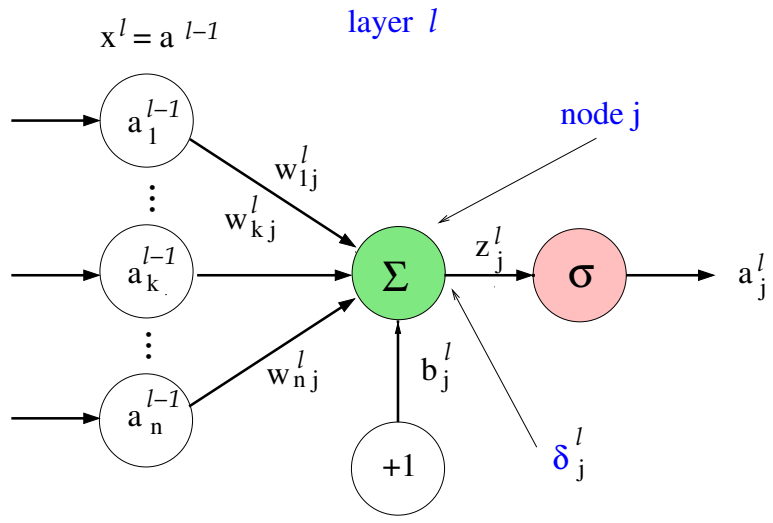
We have the following equations

$$(BP1) \quad \delta^L = \nabla_{a^L} E \odot \sigma'(z^L)$$

$$(BP2) \quad \delta^l = (W^{l+1})^T \delta^{l+1} \odot \sigma'(z^l)$$

$$(BP3) \quad \frac{\partial E}{\partial b^l_j} = \delta^l_j$$

$$(BP4) \quad \frac{\partial E}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$



where $\odot$ is the Hadamard product (component-wise)

The vanishing gradient problem

$$(BP2) \quad \delta^l = (w^{l+1})^T \delta^{l+1} \odot \sigma'(z^l)$$

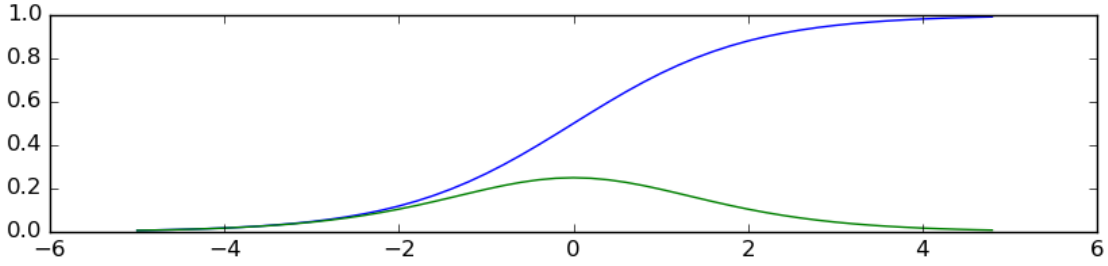
By the chain rule, the derivative is a long sequence of factors, where these factors are, alternately

- derivatives of activation functions
- derivative of linear functions, that are constants (in fact, the transposed matrix of the linear coefficients)

Let's have a look at the derivatives of a couple of activation functions.

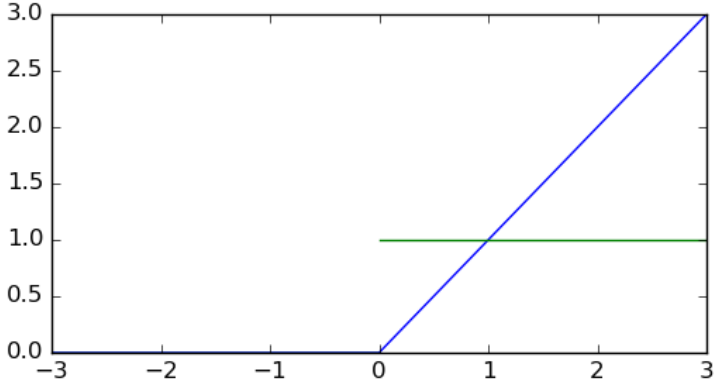
Derivatives of common activation functions

Sigmoid



Observe the flat shape of $\sigma'(x)$, always below 0.25

Relu



The vanishing gradient problem

If you systematically use the sigmoid as activation function in all layers of a deep network, the gradient will contain a lot of factors below 0.25, resulting in a very small value.

If the gradient is close to zero, learning is impossible.

This is known as the **vanishing gradient problem**.

A bit of history

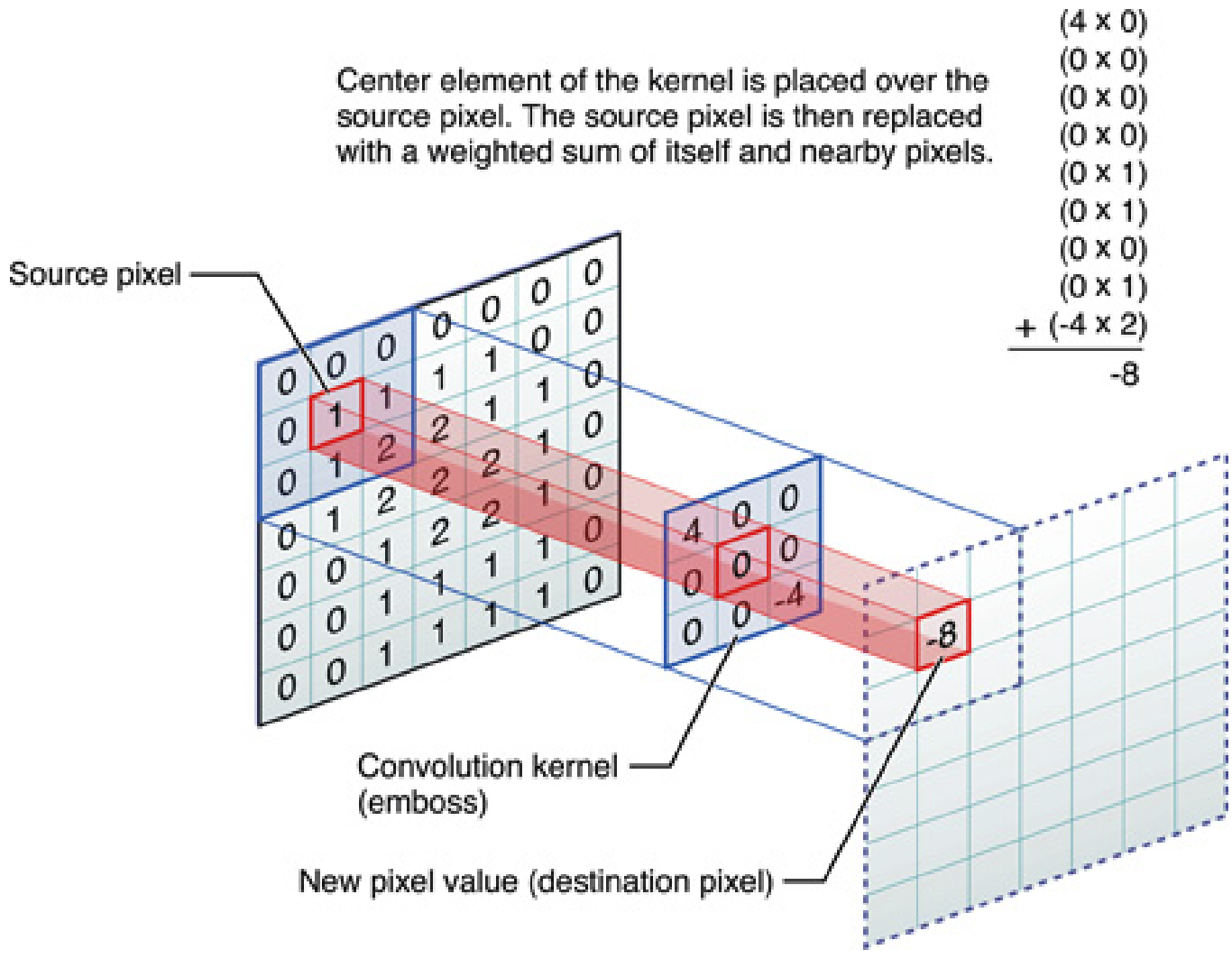
The vanishing gradient problem blocked the progress on neural netwoks for almost 15 years (1990-2005).

It was first bypassed by network pre-training (e.g. with Boltzmann Machines), and later by the introduction on new activation functions, such as Rectified Linear Units (RELU), making pre-training obsolete.

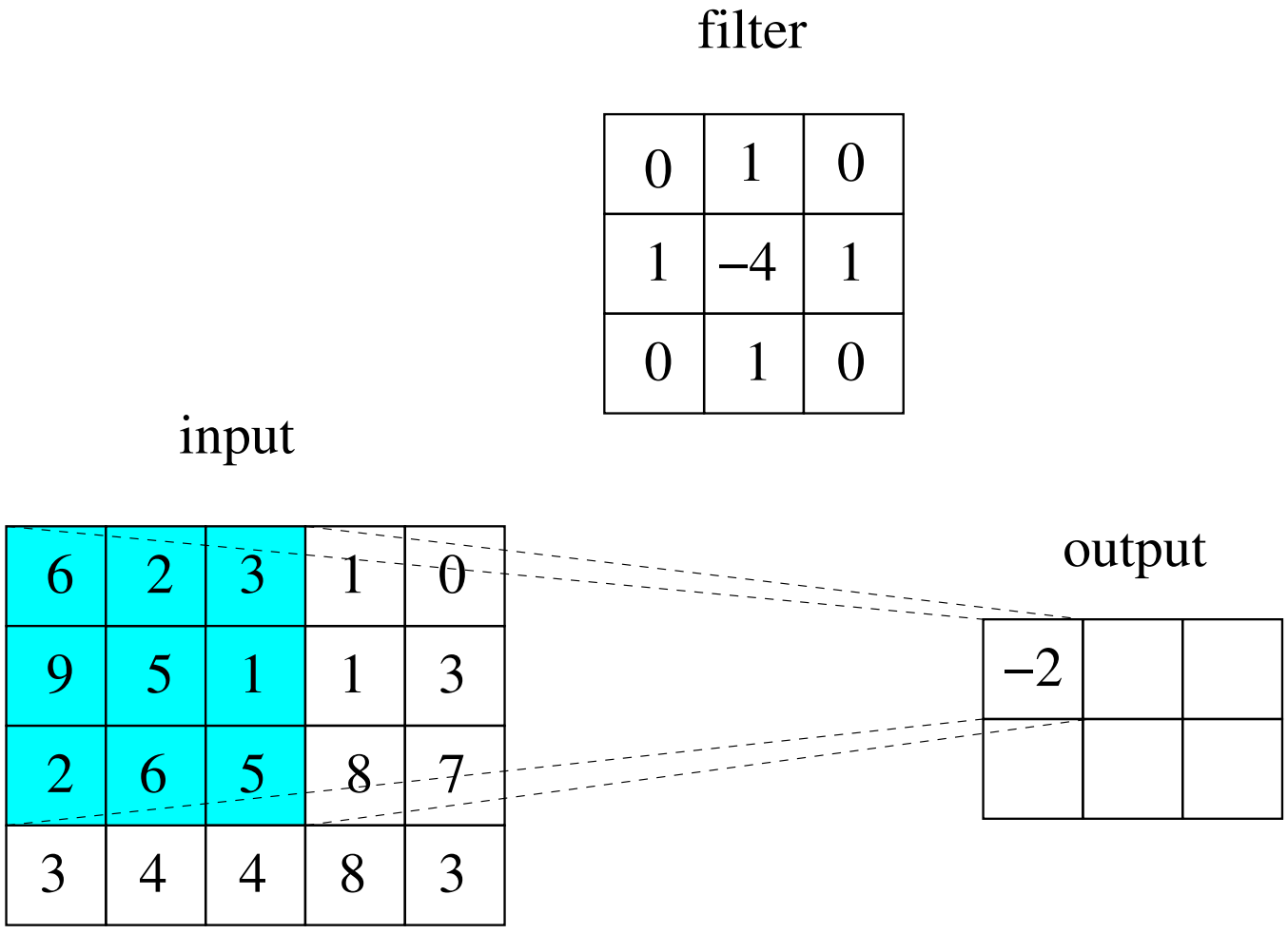
Still, fine-tuning starting from good network weights (e.g. VGG) is a viable approach for many problems (transfer learning).

Convolutional Neural Networks

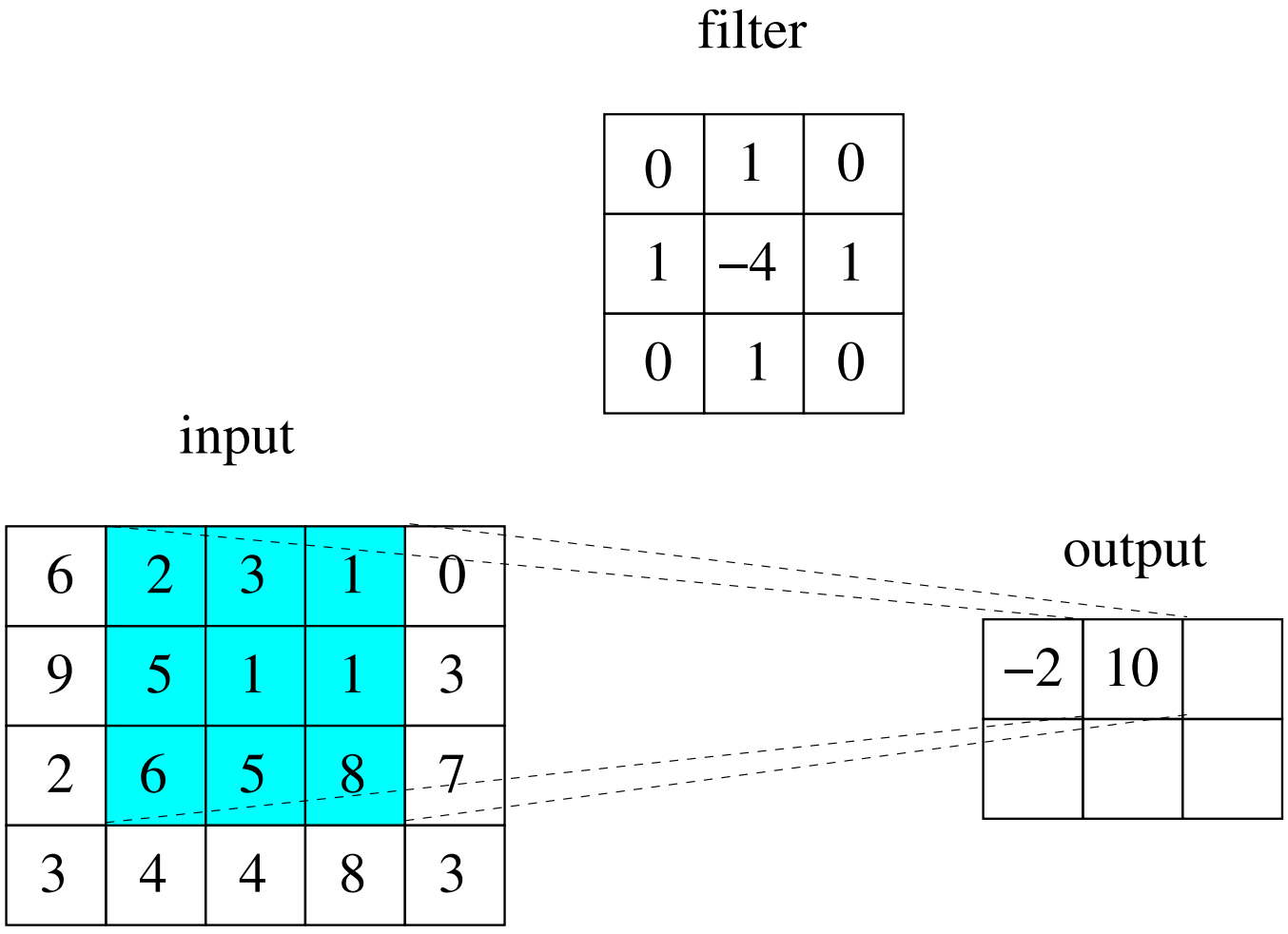
Filters and convolutions



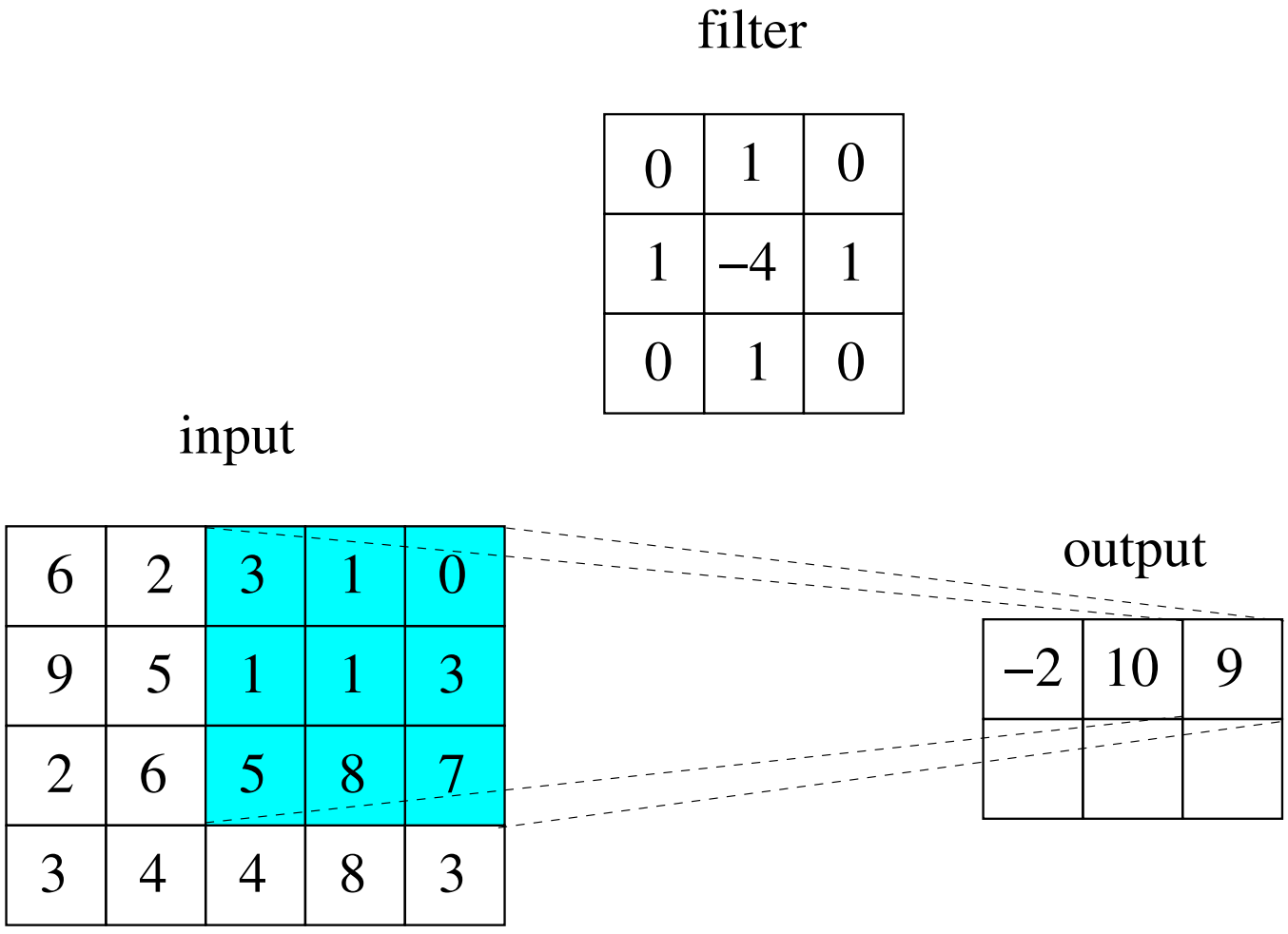
Filters and convolutions



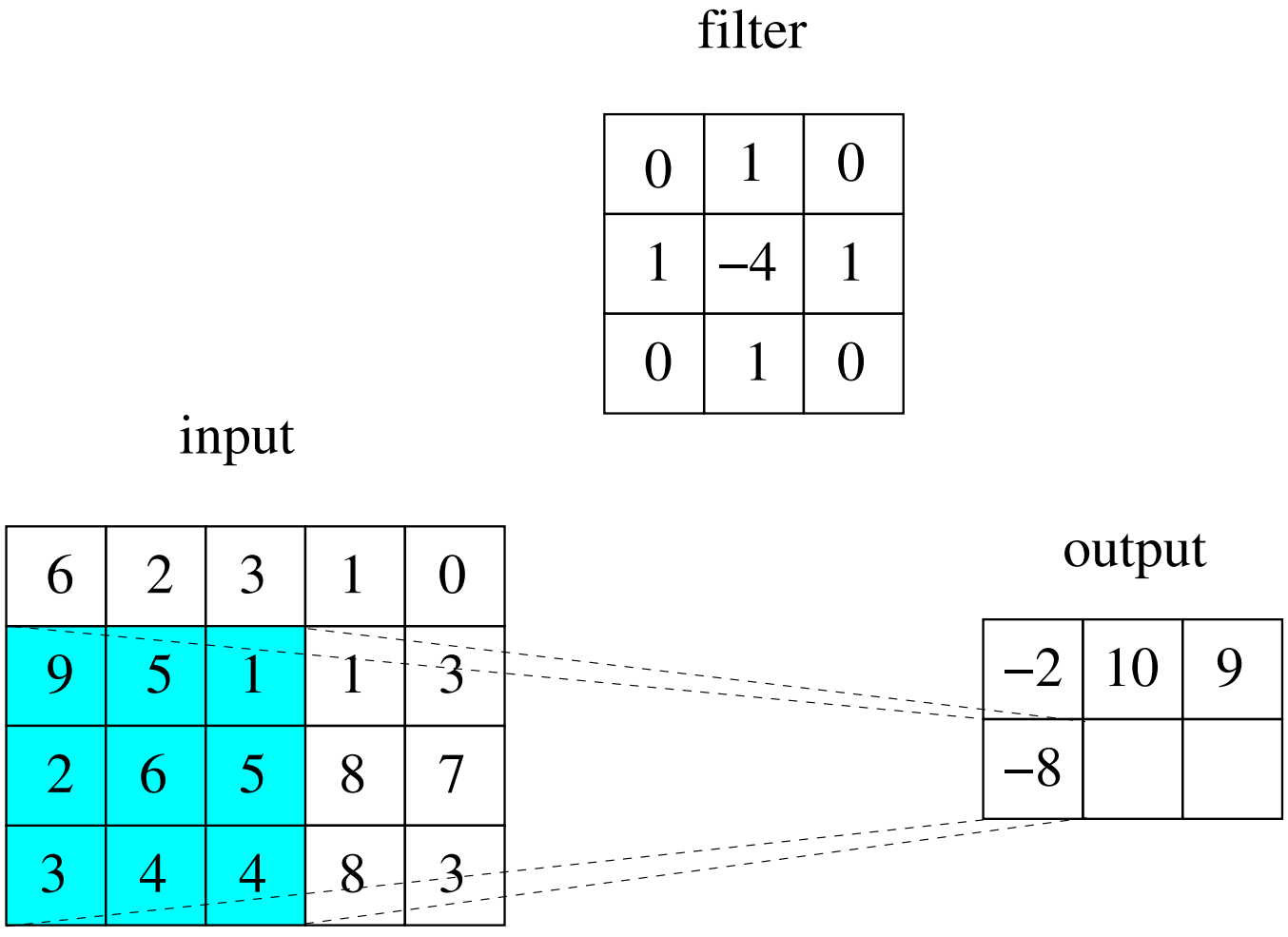
Filters and convolutions



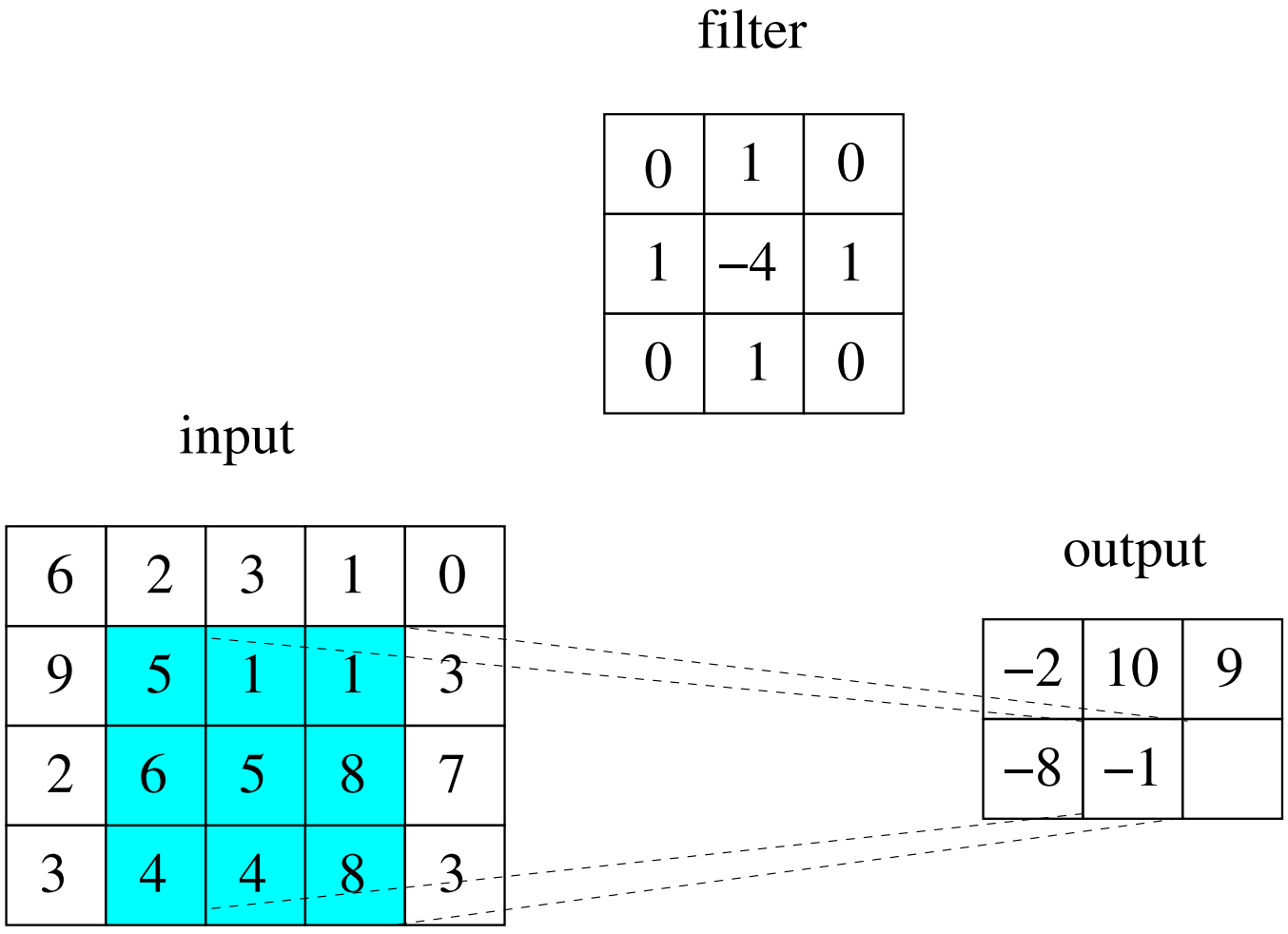
Filters and convolutions



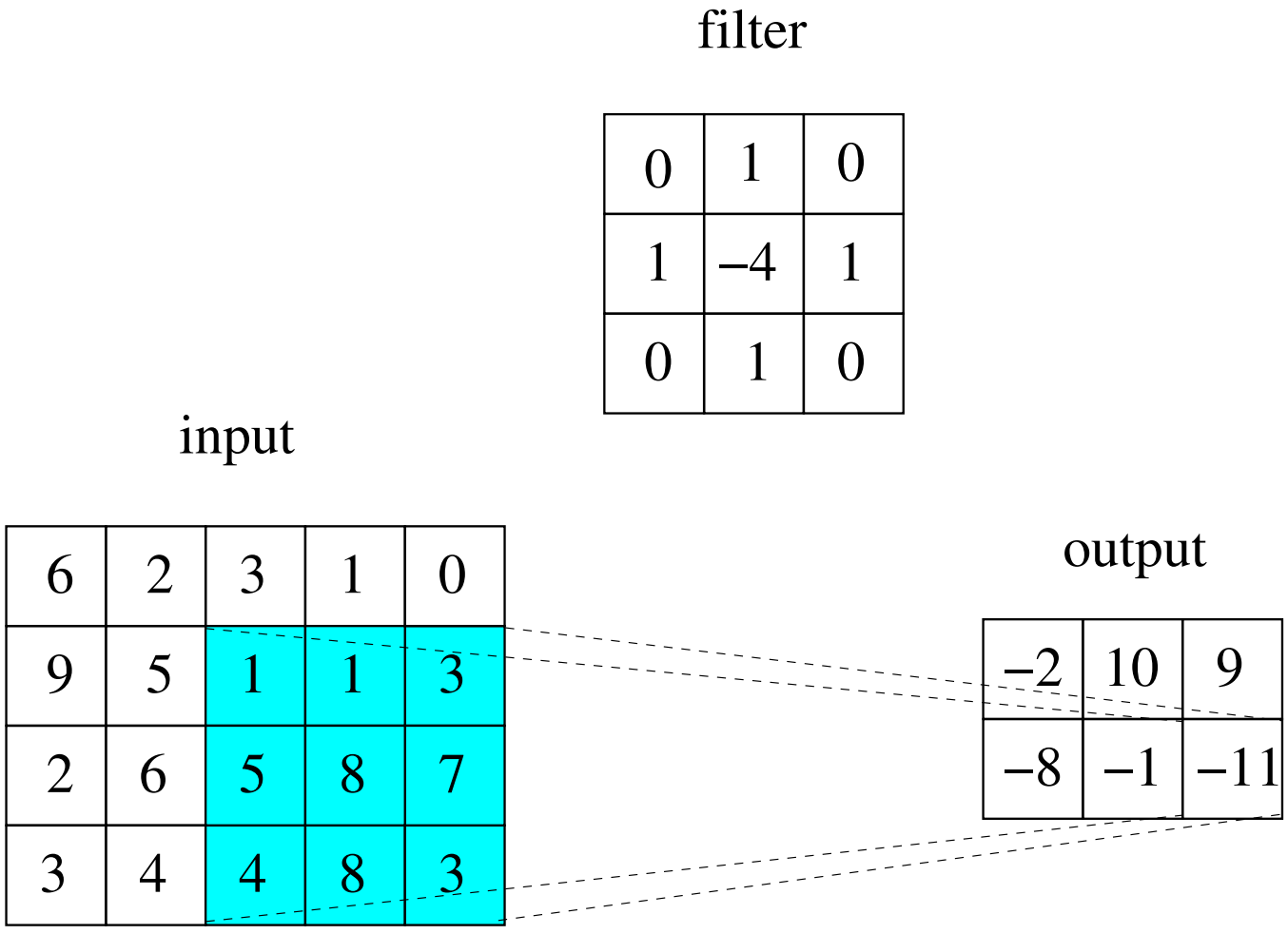
Filters and convolutions



Filters and convolutions



Filters and convolutions



Loose connectivity and shared weights

- the activation of a neuron is not influenced from all neurons of the previous layer, but only from a small subset of adjacent neurons: his receptive field
- every neuron works as a convolutional filter. Weights are shared: every neuron perform the **same trasformation** on **different areas** of its input
- with a cascade of convolutional filters intermixed with activation functions we get complex non-linear filters assembing local features of the image into a global structure.

A parenthesis

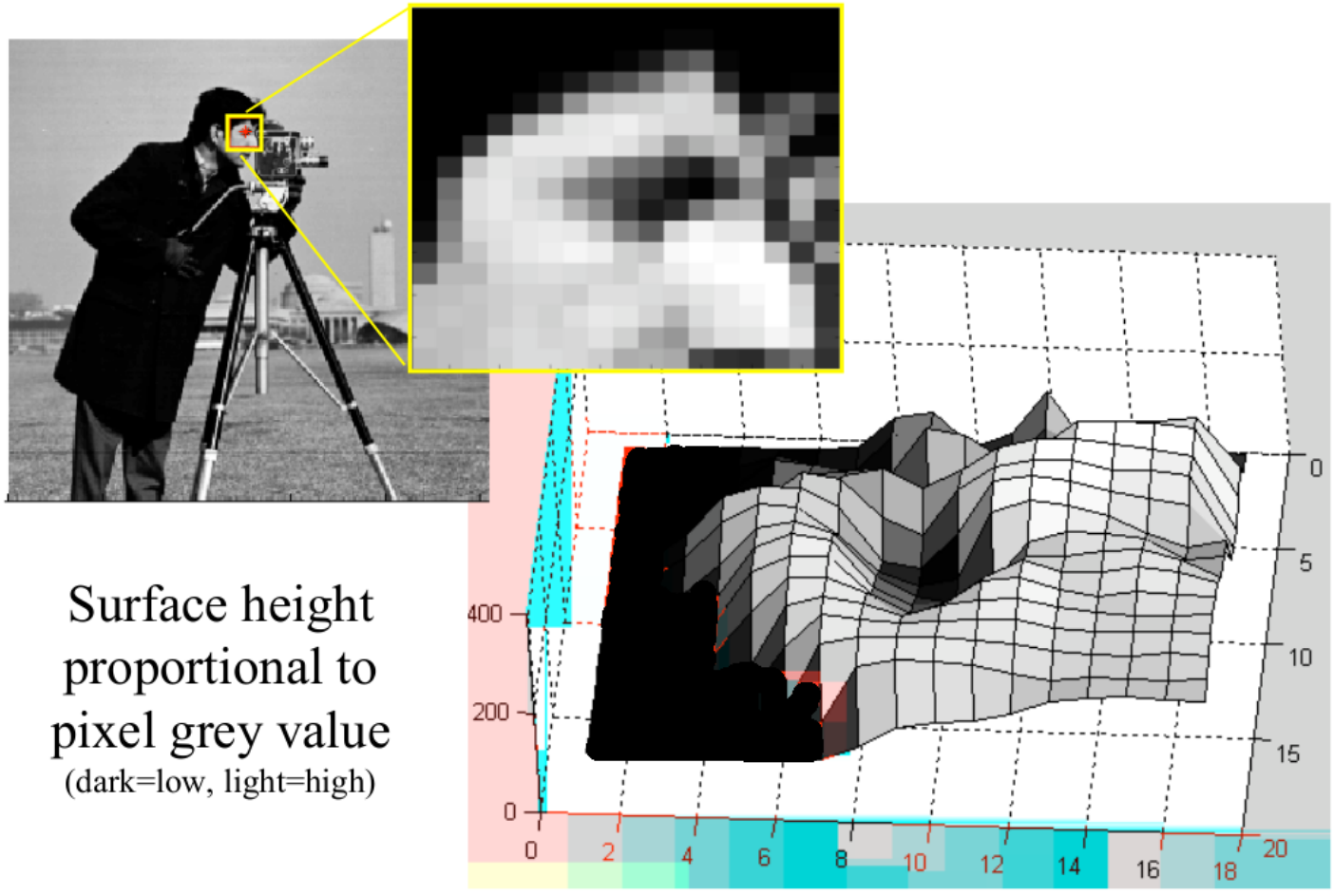
About the relevance of convolutions for image processing

Images are numerical arrays

An image is coded as a numerical matrix (array)
grayscale (0-255) or rgb (triple 0-255)

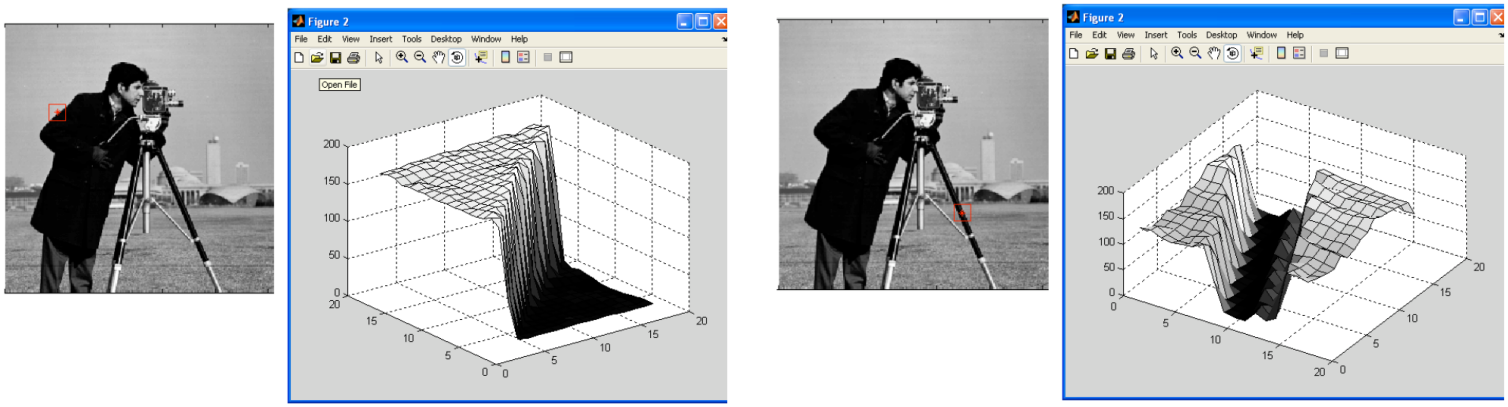
$$\begin{bmatrix} 207 & 190 & 176 & 204 & 204 & 208 \\ 110 & 108 & 114 & 112 & 123 & 142 \\ 94 & 100 & 96 & 121 & 125 & 108 \\ 95 & 86 & 81 & 84 & 88 & 88 \\ 69 & 51 & 36 & 72 & 78 & 81 \\ 74 & 97 & 107 & 116 & 128 & 133 \end{bmatrix}$$

Images as surfaces



Interesting points

Edges, angles, ...: points where there is a discontinuity, i.e. a fast variation of the intensity



More generally, are interested to identify patterns inside the image.

The key idea is that the kernel of the convolution expresses the pattern we are looking for.

Example: finite derivative

Suppose we want to find the positions inside the image where there is a sudden horizontal passage from a dark region to a bright one. The pattern we are looking for is

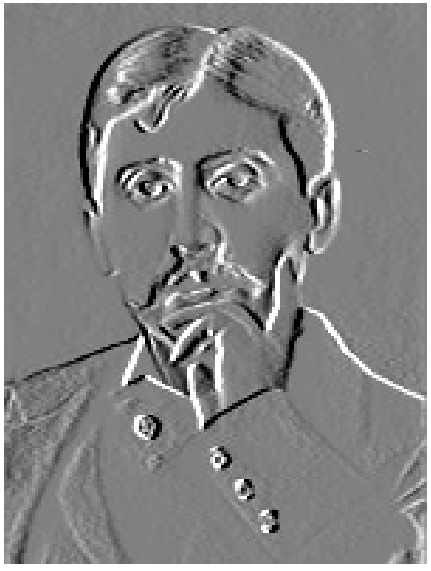
$$\begin{bmatrix} -1 & 1 \end{bmatrix}$$

or, varying the distance between pixels:

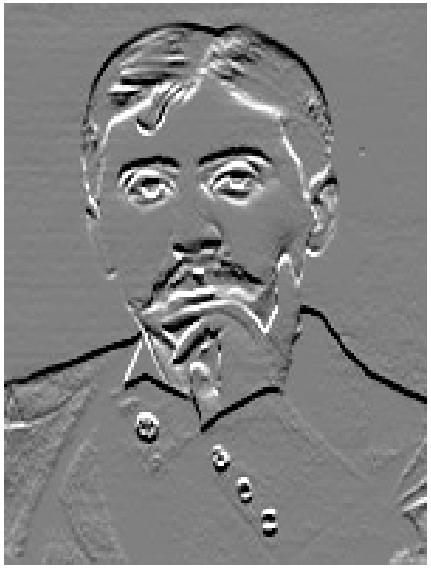
$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

The finite derivative at work

$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \longrightarrow$$



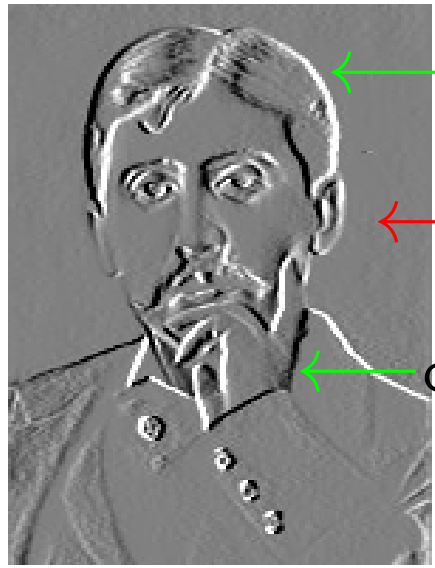
$$\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \longrightarrow$$



Recognizing Patterns

Each neuron in a convolutional layer gets activated by specific patterns in the input image.

$$\text{pattern} = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$



Another example: the finite laplacian

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Discovering patterns

But how to find good patterns?

Usual idea:

instead of using human designed pre-defined patterns, let the net learn them.

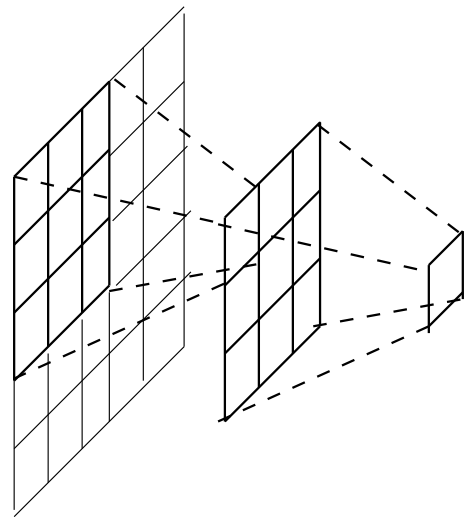
Particularly important in deep architectures, because:

- stacking kernels we **enlarge their receptive fields** (see next slide)
- adding non-linear activations we synthesize complex, **non-linear kernels**

Receptive field

The receptive field of a (deep, hidden) neuron is the dimension of the input region influencing it.

It is equal to the dimension of an input image producing (without padding) an output with dimension 1.



A neuron cannot see anything outside its receptive field!

We may also rapidly enlarge the receptive fields by means of **downsampling** layers, e.g. pooling layers or convolutional layers with non-unitarian stride

Complex (deep) patterns

The intuition is that neurons at higher layers should recognize increasingly complex patterns, obtained as a combination of previous patterns, over a larger receptive field.

In the highest layers, neurons may start recognizing patterns similar to features of objects in the dataset, such as feathers, eyes, etc.

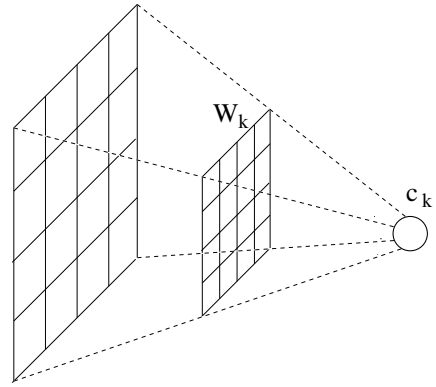
In the final layers, neurons gets activated by "patterns" identifying objects in the category.

can we confirm such a claim?

How CNNs see the world

Visualization of hidden layers

Goal: find a way to visualize the kind of patterns a specific neuron gets activated by.



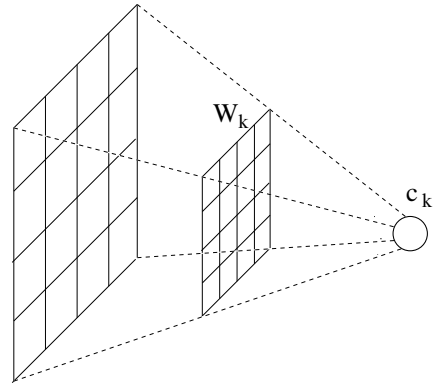
The loss function $\mathcal{L}(\theta, x)$ of a NN depends on the parameters θ and the input x.

During training, we fix x and compute the partial derivative of $\mathcal{L}(\theta, x)$ w.r.t the parameters θ to adjust them in order to decrease the loss.

In the same way, we can fix θ and use partial derivatives w.r.t. input pixels in order to syntehsize images minimizing the loss.

Visualization of hidden layers

Goal: find a way to visualize the kind of patterns a specific neuron gets activated by.



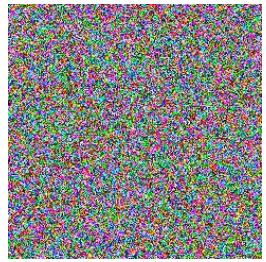
The loss function $\mathcal{L}(\theta, x)$ of a NN depends on the parameters θ and the input x.

During training, we fix x and compute the partial derivative of $\mathcal{L}(\theta, x)$ w.r.t the parameters θ to adjust them in order to decrease the loss.

In the same way, we can fix θ and use partial derivatives w.r.t. input pixels in order to syntehsize images minimizing the loss.

The "gradient ascent" technique

Start with a random image, e.g.

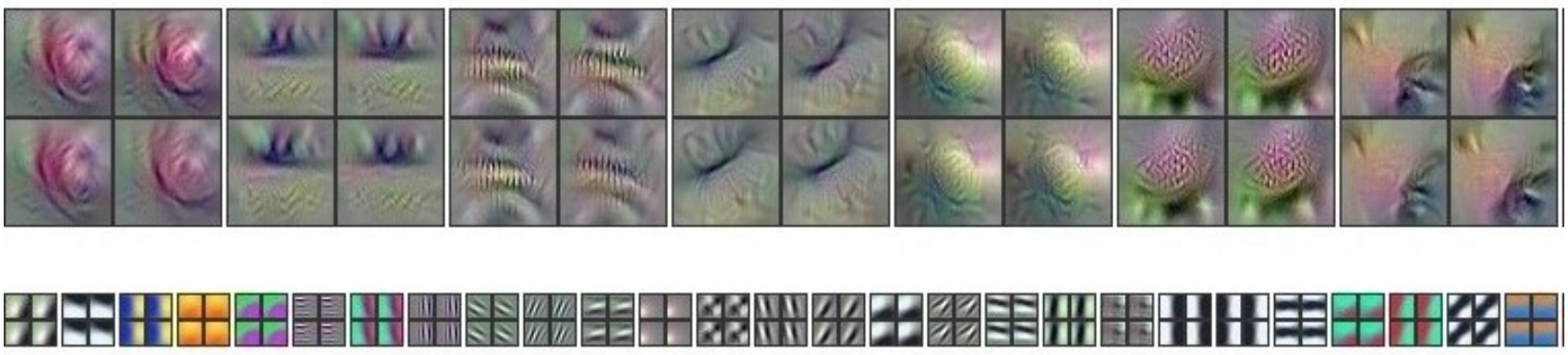


- do a forward pass using this image x as input to the network to compute the activation $a_i(x)$ caused by x at some neuron (or at a whole layer)
- do a backward pass to compute the gradient of $\partial a_i(x)/\partial x$ of $a_i(x)$ with respect to each pixel of the input image
- modify the image adding a small percentage of the gradient $\partial a_i(x)/\partial x$ and repeat the process until we get a sufficiently high activation of the neuron

First layers

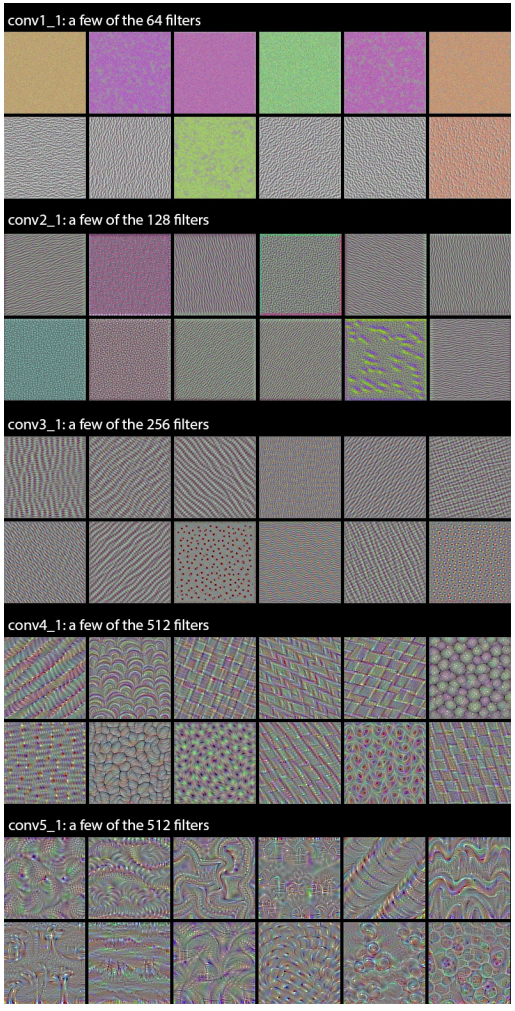
Some neurons from the first two layers of AlexNet

(Understanding Neural Networks Through Deep Visualization by A.Nguyen et al., 2015)



First features (lower picture) are very simple, and get via via more complex at higher levels, as their receptive field get larger due to nested convolutions.

First layers



For a visualization of the first layers of VGG see:

An exploration of convnet filter with keras

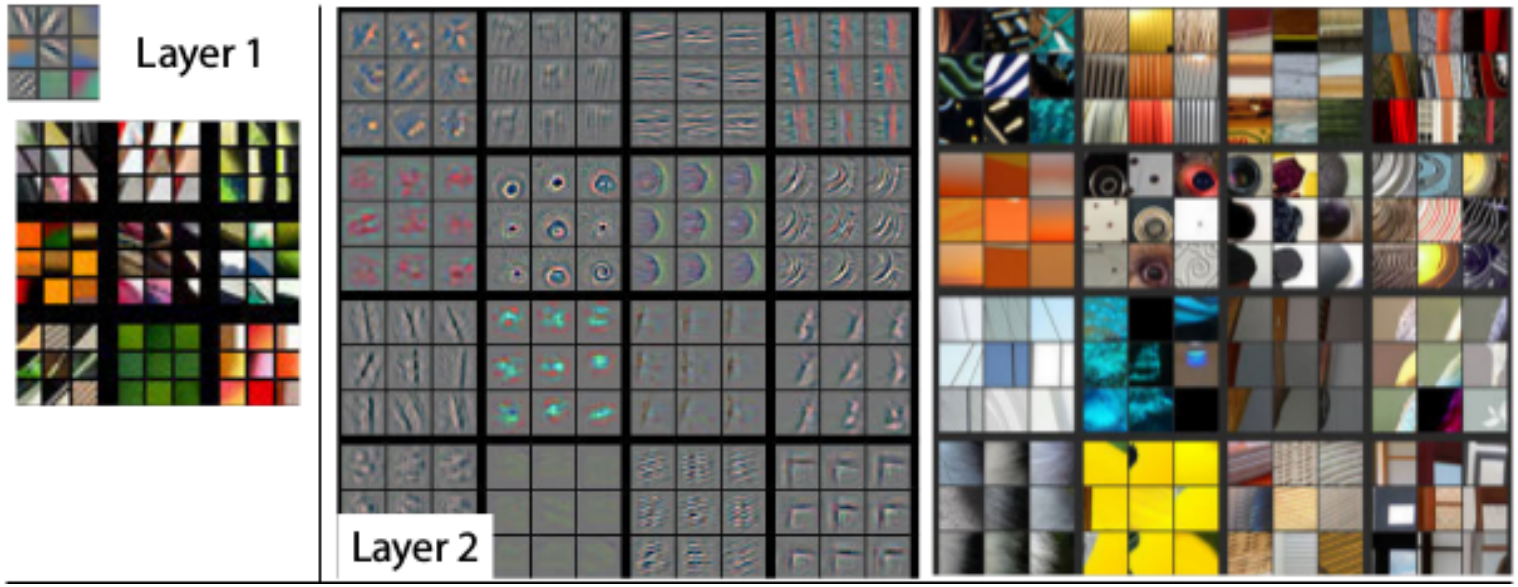
What caused the activation of this neuron **in this image**?

Instead of trying to syntesize the pattern recognized by a given neuron, we can use the gradient ascent technique to emphasize in real images what is causing its activation.

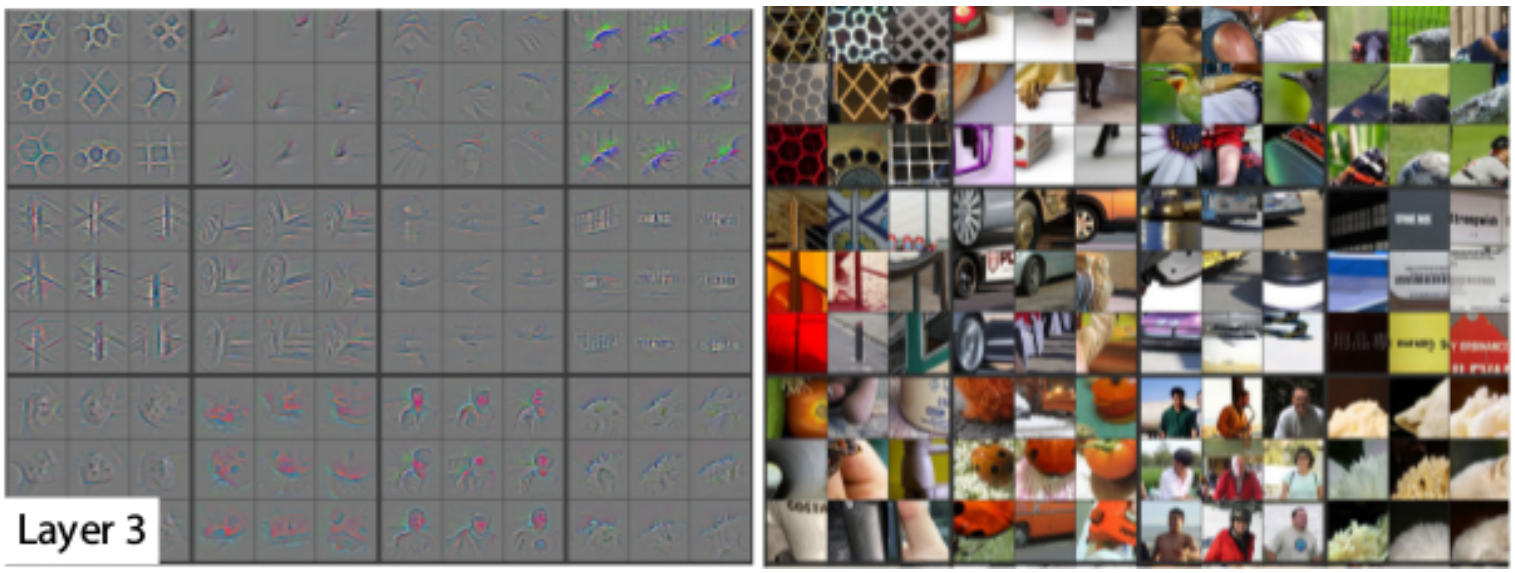
Visualizing and Understanding Convolutional Networks Matthew D Zeiler, Rob Fergus (2013)

results - layers 1 and 2

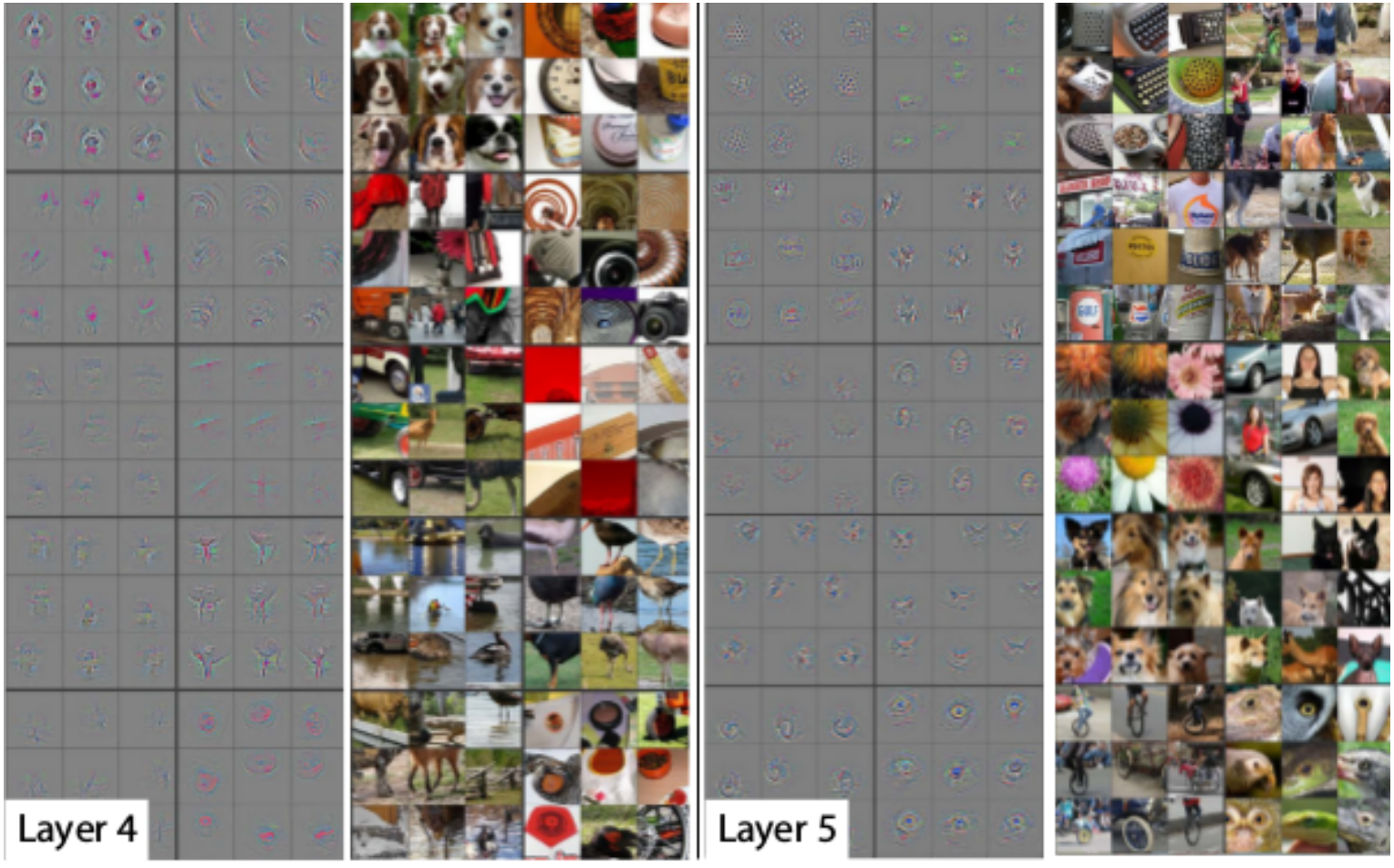
(better viewed in the original article)



results - layer 3



results - layers 4 and 5



Summary

Moving towards higher levels we observe

- growing structural complexity:
  oriented lines, colors $\rightarrow$ angles, arcs $\rightarrow$ textures
- more semantical grouping
- greater invariance to scale and rotation

See also Understanding Deep Image Representations by Inverting Them A. Mahendran, A. Vedaldi (2014)

Lesson 5

- Size, stride, padding, depth
- Examples of real CNNs
- Transfer Learning

Tensors for 2D processing

Tensors

Convolutional Networks process Tensors. A Tensor is just an multidimensional array of floating numbers.

The typical tensor for 2D images has **four** dimensions:

$$batchsize \times width \times height \times channels$$

Features maps are **stacked** along the channel dimension.

At start, for a color image, we just have 3 channels: r,g,b.

How do kernel operate along the channel dimension?

Tensors

Convolutional Networks process Tensors. A Tensor is just an multidimensional array of floating numbers.

The typical tensor for 2D images has **four** dimensions:

$$batchsize \times width \times height \times channels$$

Features maps are **stacked** along the channel dimension.

At start, for a color image, we just have 3 channels: r,g,b.

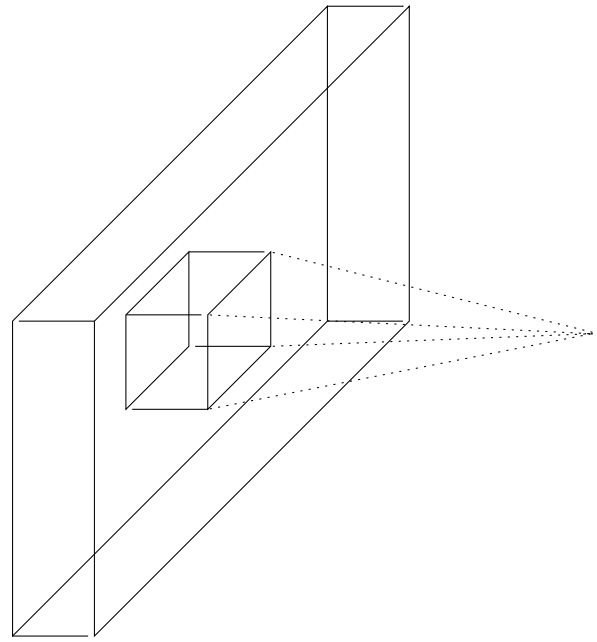
How do kernel operate along the channel dimension?

Dense processing along channel axis

Unless stated differently (e.g. in separable convolutions), a filter operates on all input channels in parallel.

So, if the input layer has depth D, and the kernel spatial size is NxM, the actual dimension of the kernel will be

NxMxD



The convolution kernel is tasked with simultaneously mapping cross-channel correlations and spatial correlations

Spatial dimension of the resulting feature map

Each kernel produces a single feature map.
Feature maps produced by different kernels are stacked along the channel dimension: the number of kernels is equal to the channel-**depth** of the next layer.

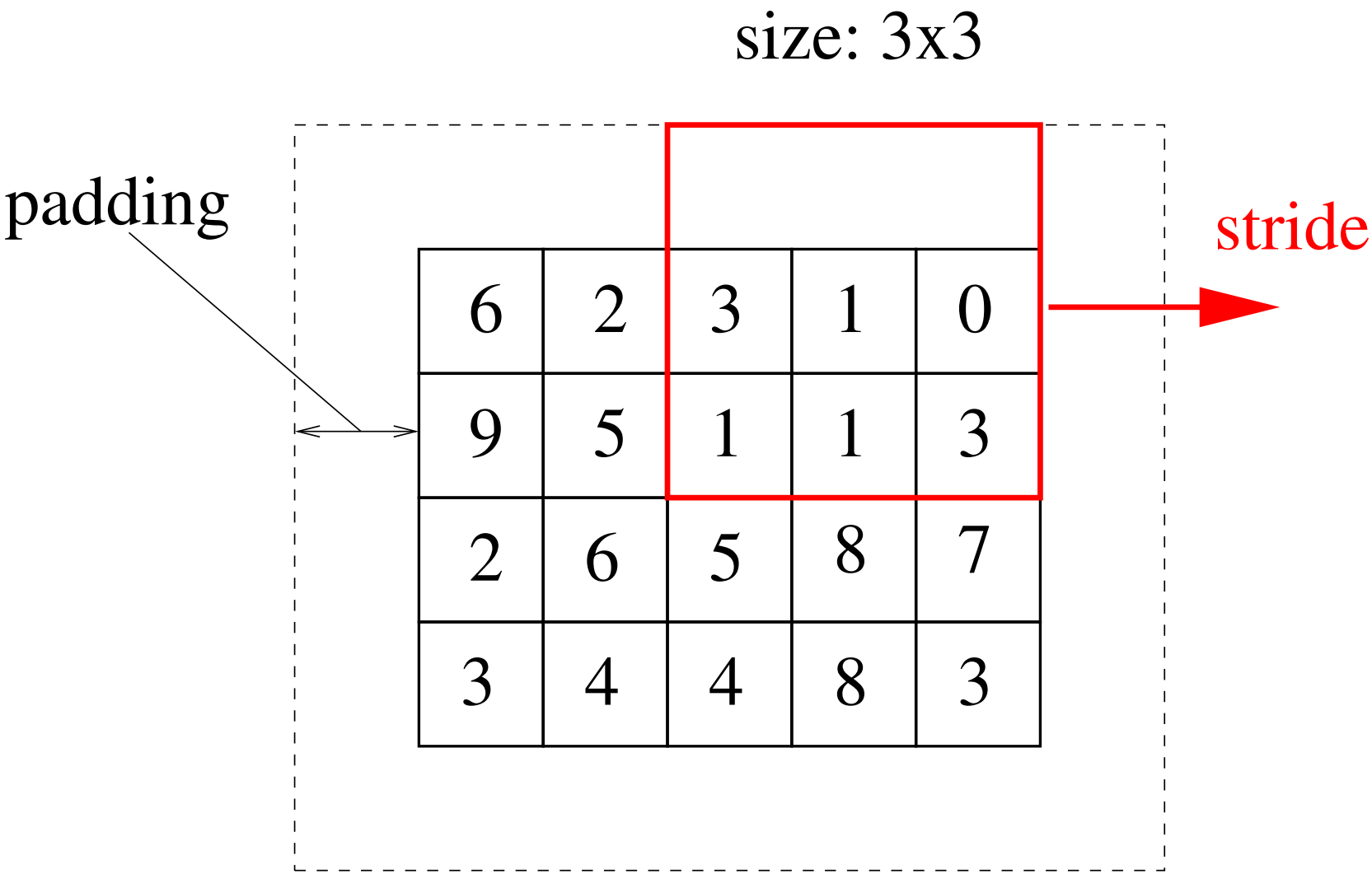
The **spatial** dimension of the feature map depends from two configurable factors:

- **paddding**: extra space added around the input
- **stride**: kernel deplacement over the input during convolution

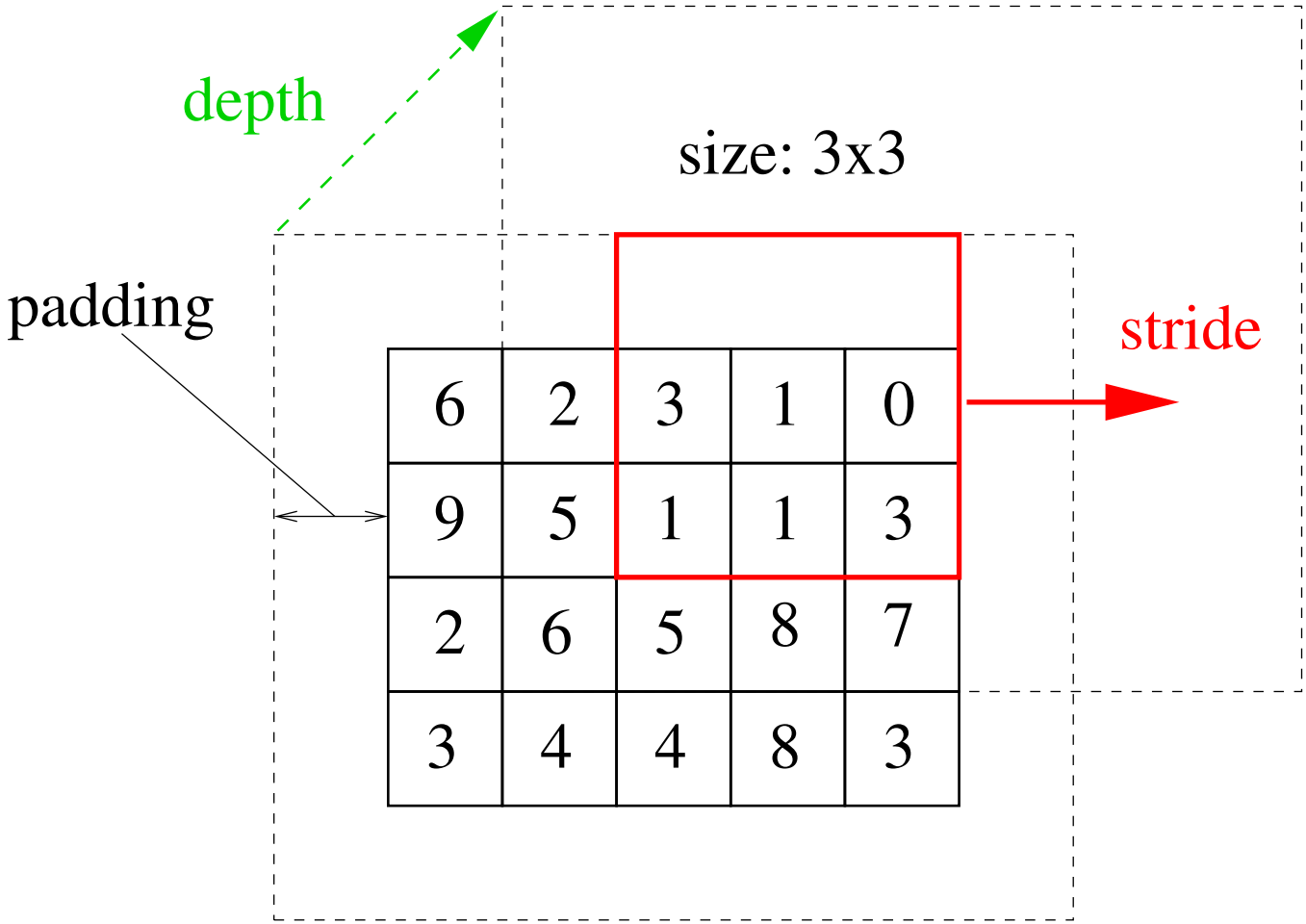
Relevant parameters for convolutional layers

- **kernel size**: the dimension of the linear filter.
- **stride**: movement of the linear filter. With a low stride (e.g. unitary) receptive fields largely overlap. With a higher stride, we have less overlap and the dimension of the output get smaller (lower sampling rate).
- **padding** Artificial enlargement of the input to allow the application of filters on borders.
- **depth**: number of features maps (stacked along the so called channel axis) that are processed in parallel.
  The depth of the output layer depends from the number of different kernels that we want to synthesize (each producing a different feature map).

Configuration params for conv2D layers



Configuration params for conv2D layers



Input-output spatial relation

Along each axes the dimension of the output is given by the following formula

$$\frac{W + P - K}{S} + 1$$

where:

W = dimension of the input
P = padding
K = Kernel size
S = Stride

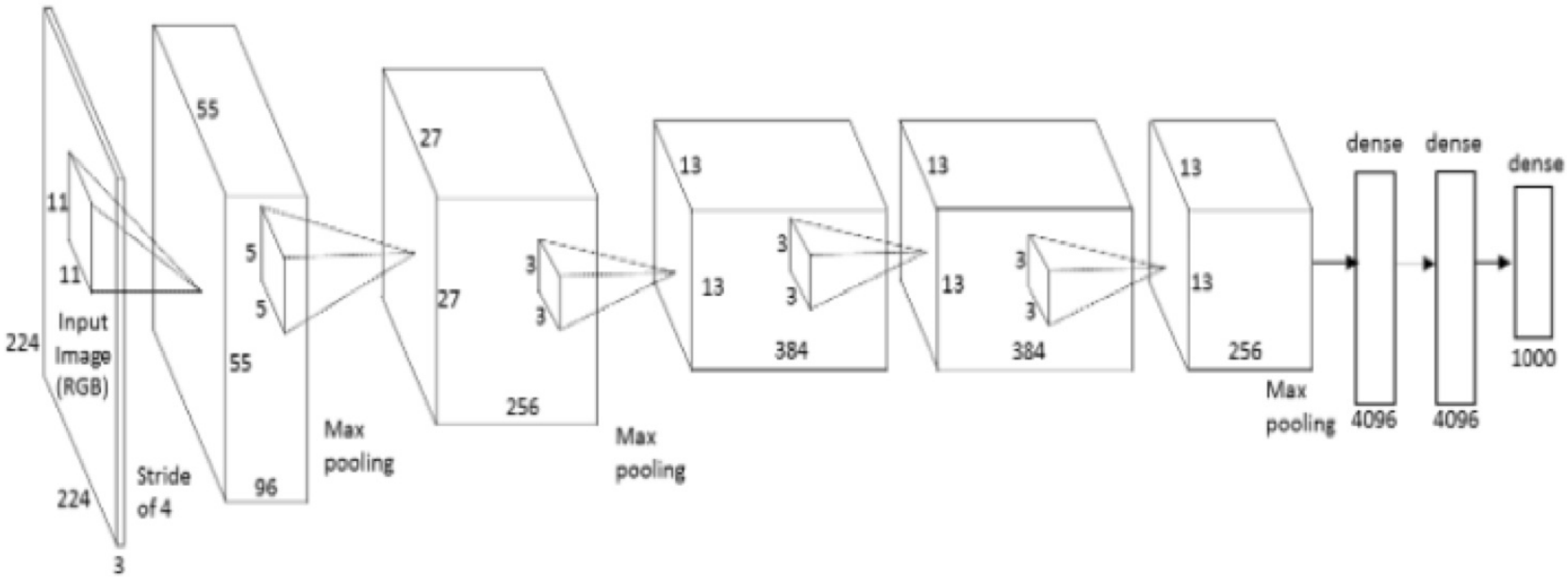
An example

[DEMO]

Important networks

AlexNet

AlexNet Architecture (Krizhevsky, Sutskever e Hinton), winner of a NIPS contest in 2012.



Pooling

In deep convolutional networks, it is common practice to alternate convolutional layers with pooling layers, where each neuron simply takes the mean or maximal value in its receptive field.

This has a double advantage:

- it reduces the dimension of the output
- it gives some tolerance to translations

Max Pooling example

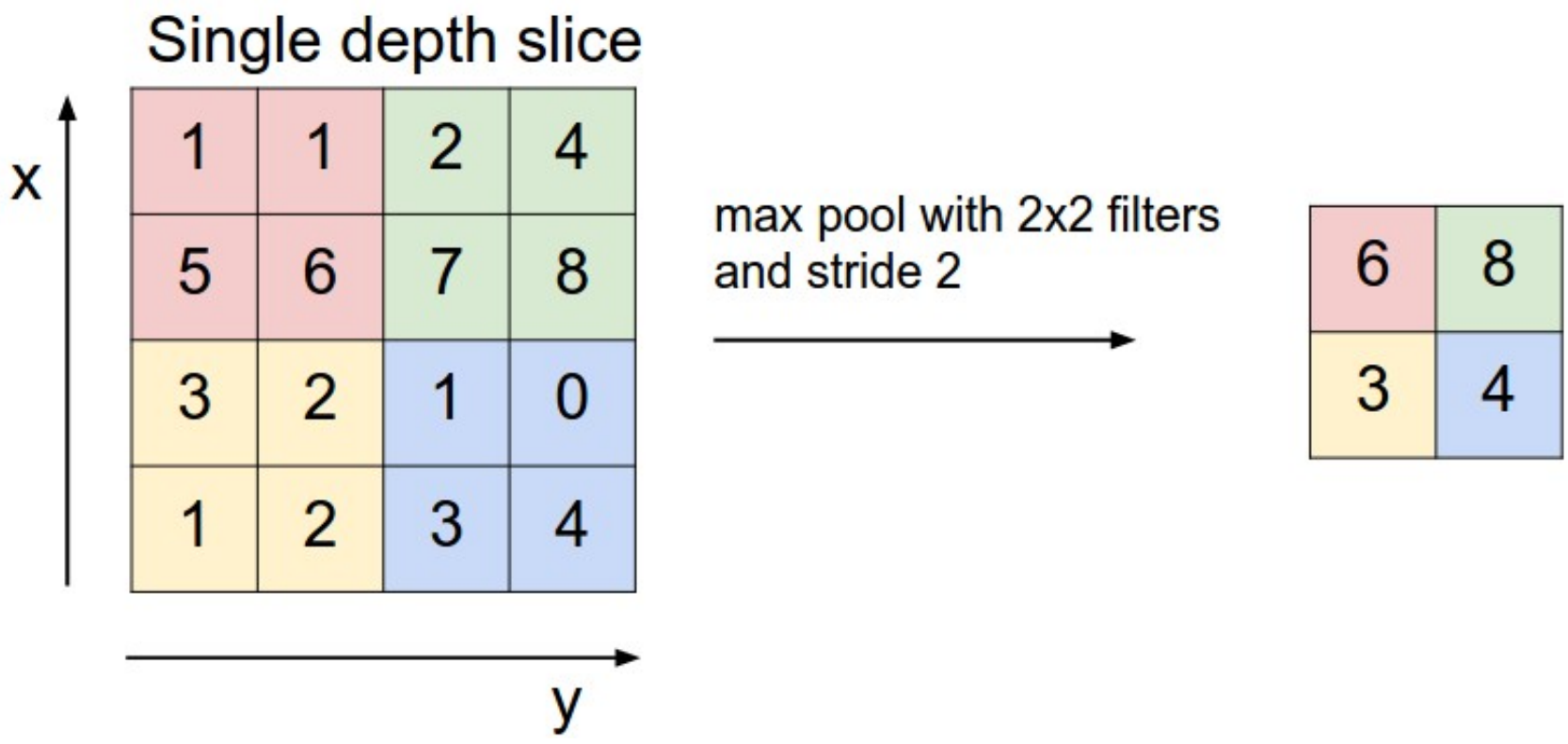
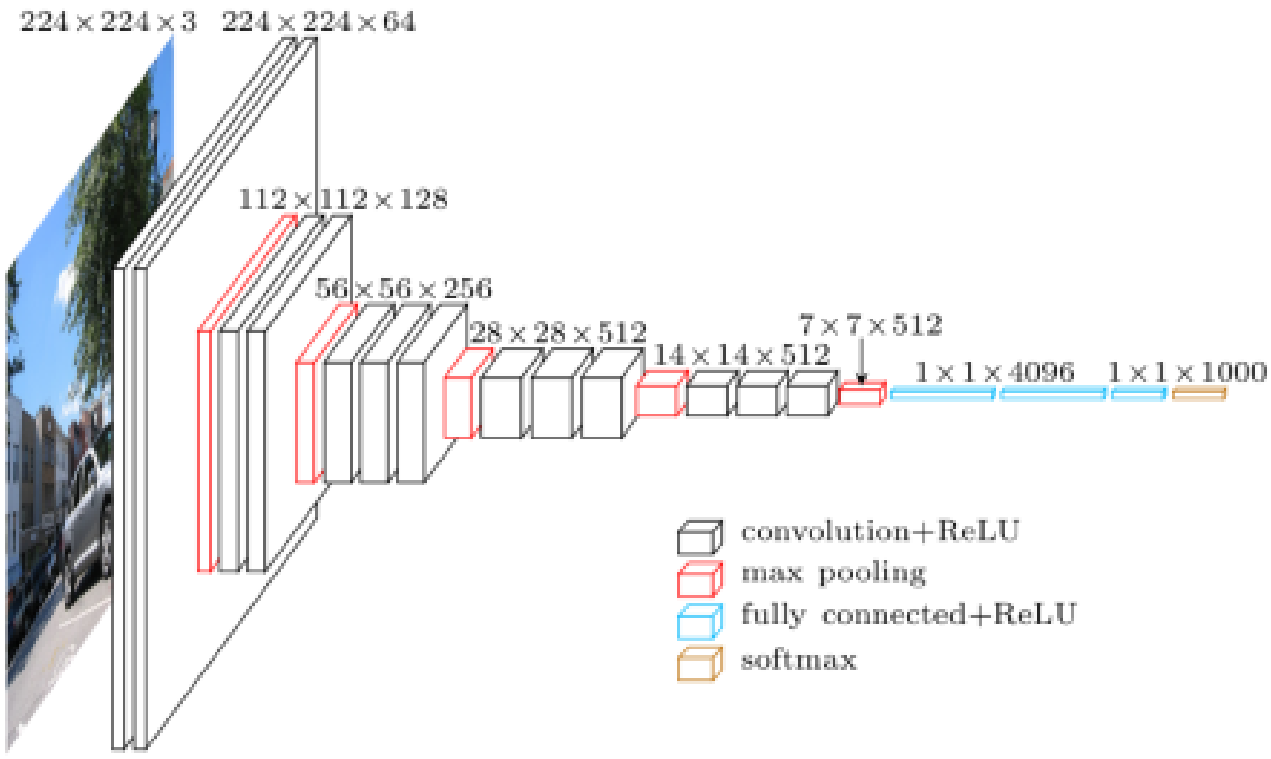


Immagine tratta da
`http://cs231n.github.io/convolutional-networks/`

VGG

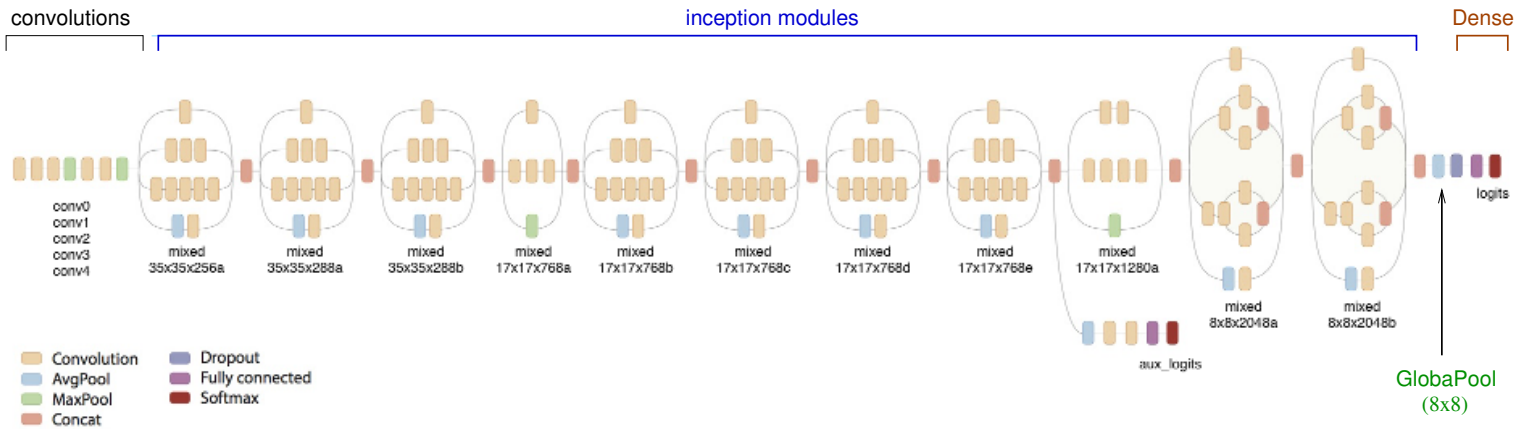
VGG 16 (Simonyan e Zisserman). 92.7 accuracy (top-5) in ImageNet (14 millions images, 1000 categories).



Picture by Davi Frossard: VGG in TensorFlow

Inception V3

Inception V3

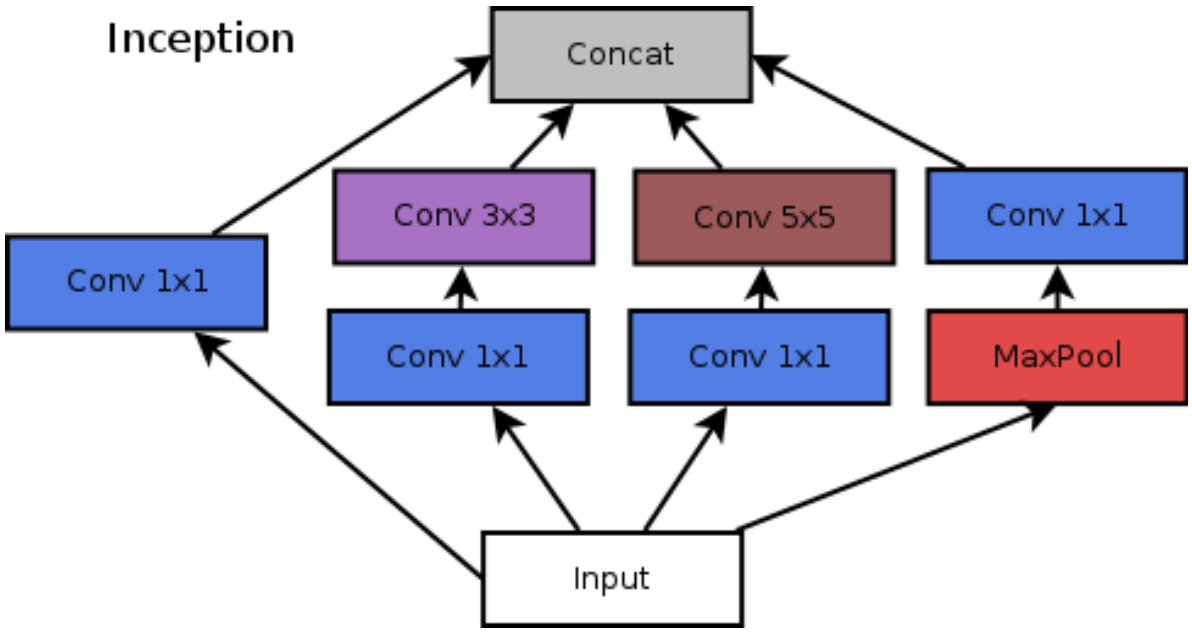


The convolutional part is a long composition of

inception modules

Inception modules

The networks is composed of inception modules (towers of nets):

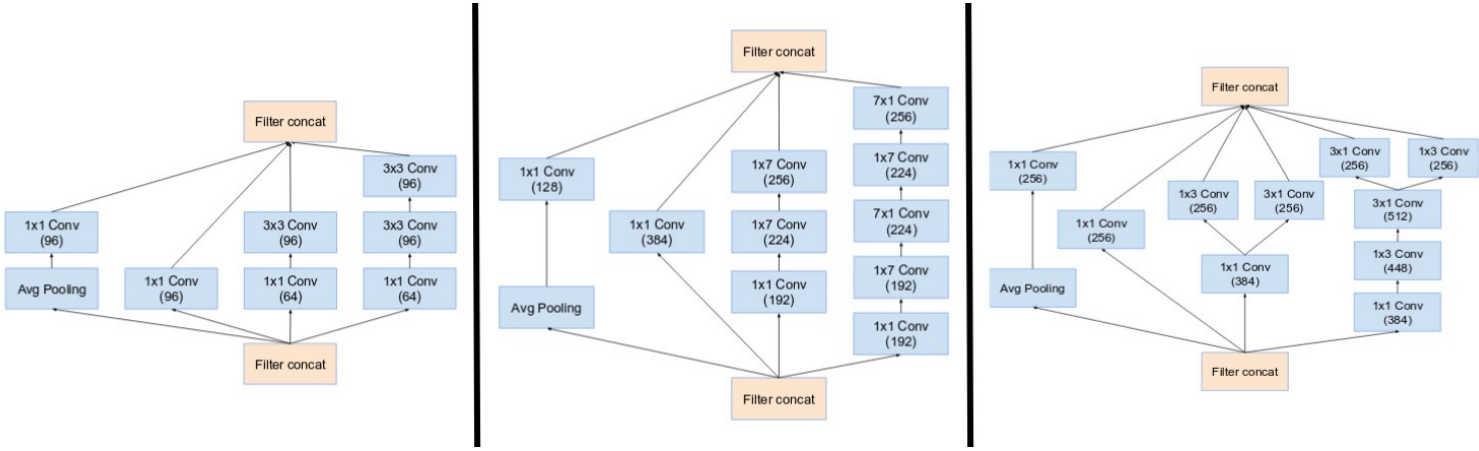


Video from the Udacity course "Deep Learning"

Variants

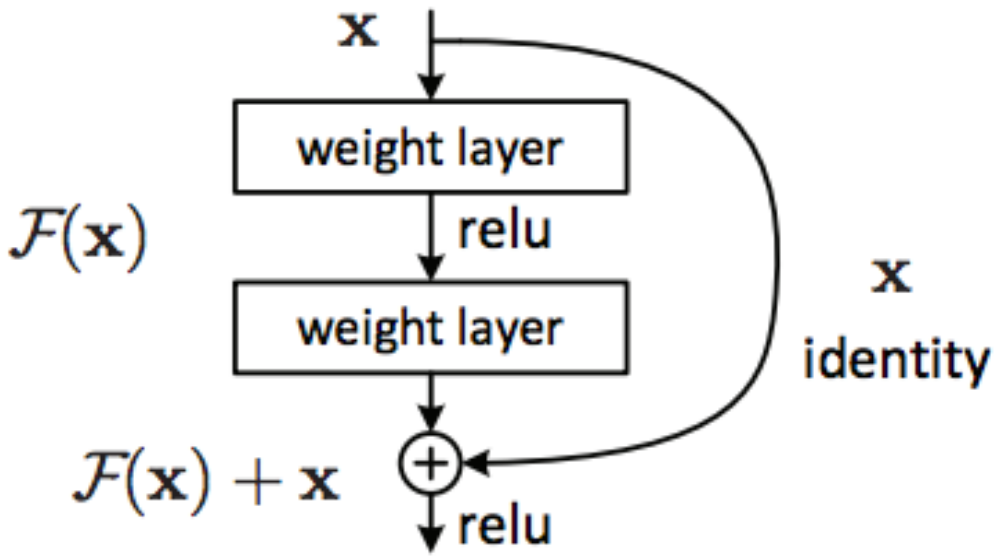
The point is to induce the net to learn different filters.

Many variants proposed and used over years:



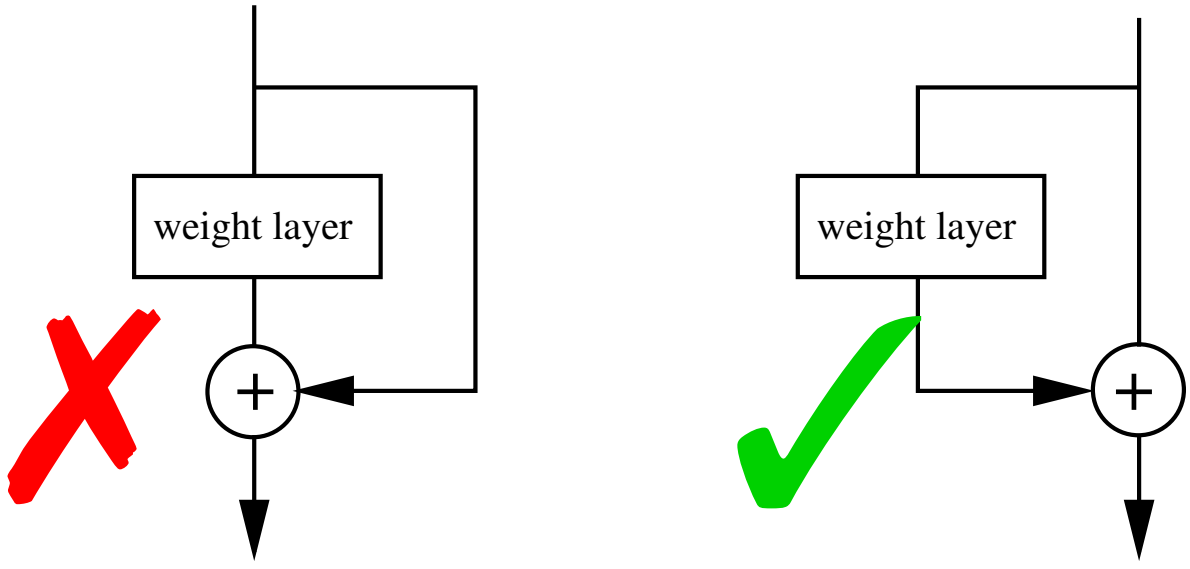
Residual Learning

Another recent topic is residual learning.

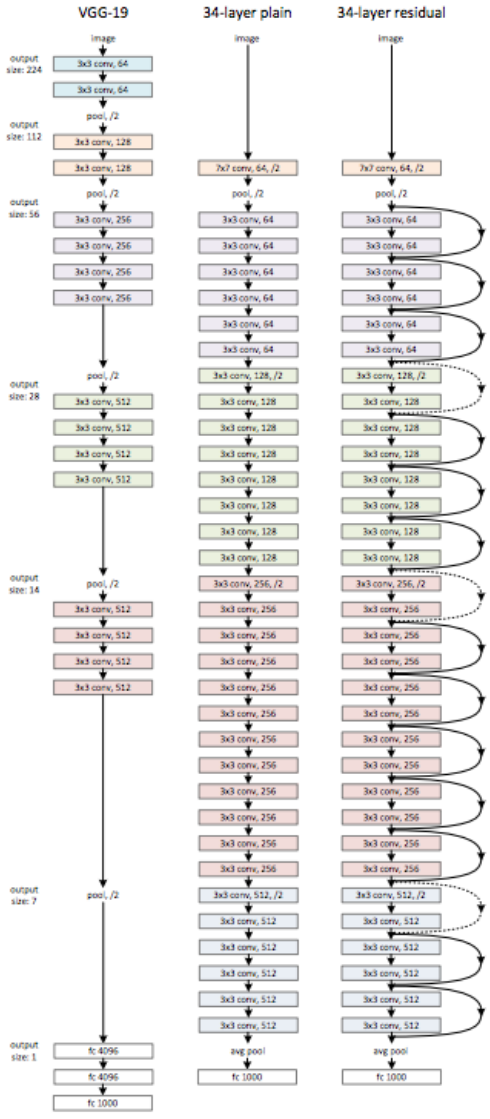


Instead of lerning a function $\mathcal{F}(x)$ you try to learn $\mathcal{F}(x) + x$.

The right intuition



Residual networks



you add a residual shortcut connection every 2-3 layers

Inception Resnet is an example of a such an architecture

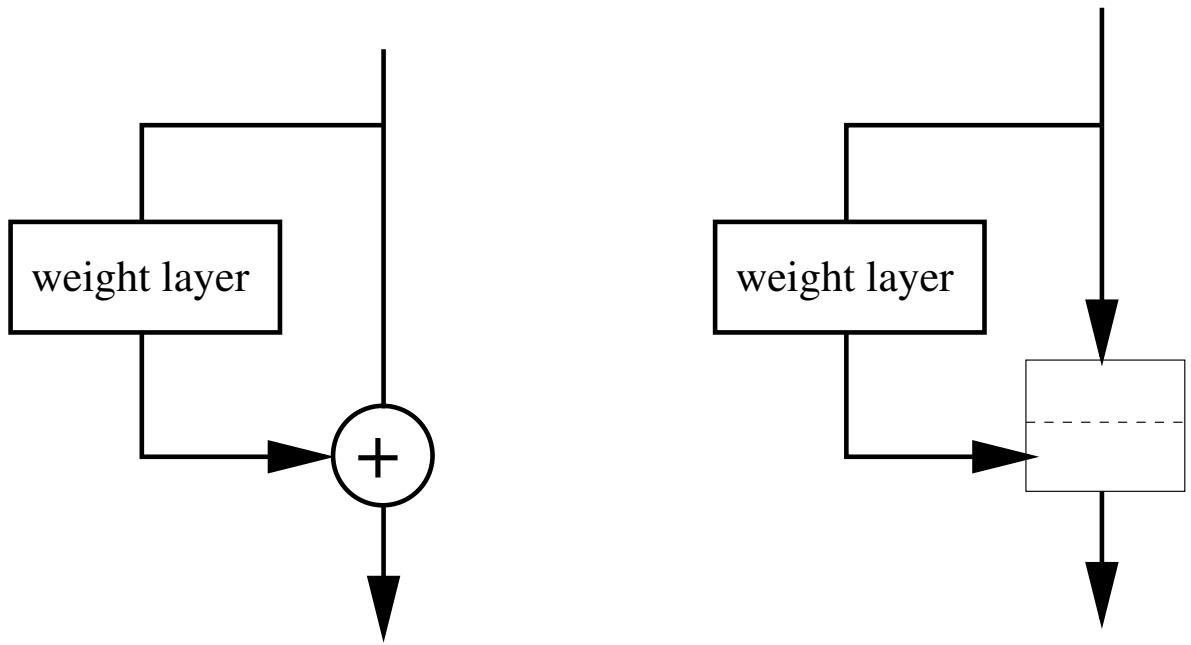
Why Residual Learning works?

Not well understood yet.

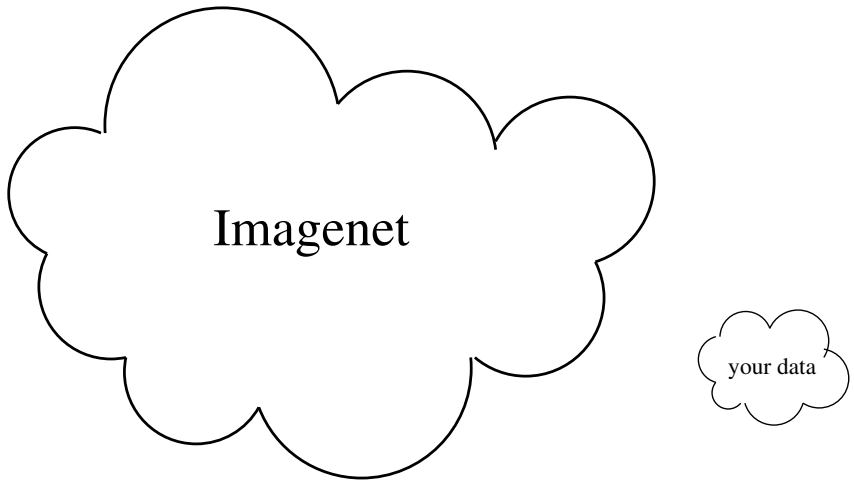
The usual explanation is that during back propagation, **the gradient at higher layers can easily pass to lower layers**, withouth being mediated by the weight layers, which may cause vanishing gradient or exploding gradient problem.

Sum or concatenation?

The "sum" operation can be interpreted in a liberal way.
A common variant consists in concatenating instead of adding (usually along the channel axis):



Transfer Learning



Reusing Knowledge

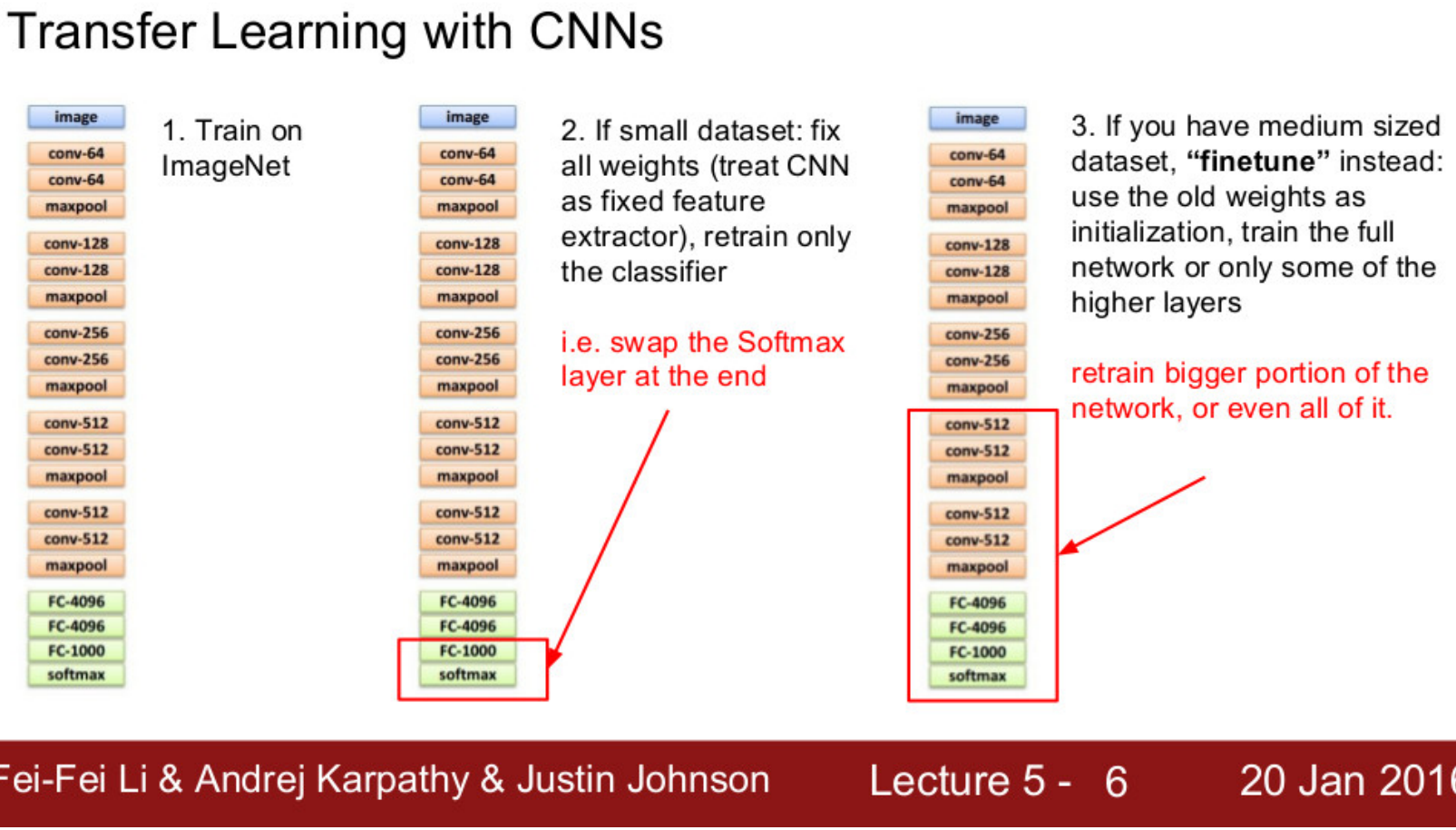
We learned that the first layers of convolutional networks for computer vision compute feature maps of the original image of growing complexity.

The filters that have been learned (in particular, the most primitive ones) are likely to be independent from the particular kind of images they have been trained on.

They have been trained on a huge amount of data and are probably very good.

It is a good idea to try to *reuse them* for other classification tasks.

Transfer Learning with CNNs

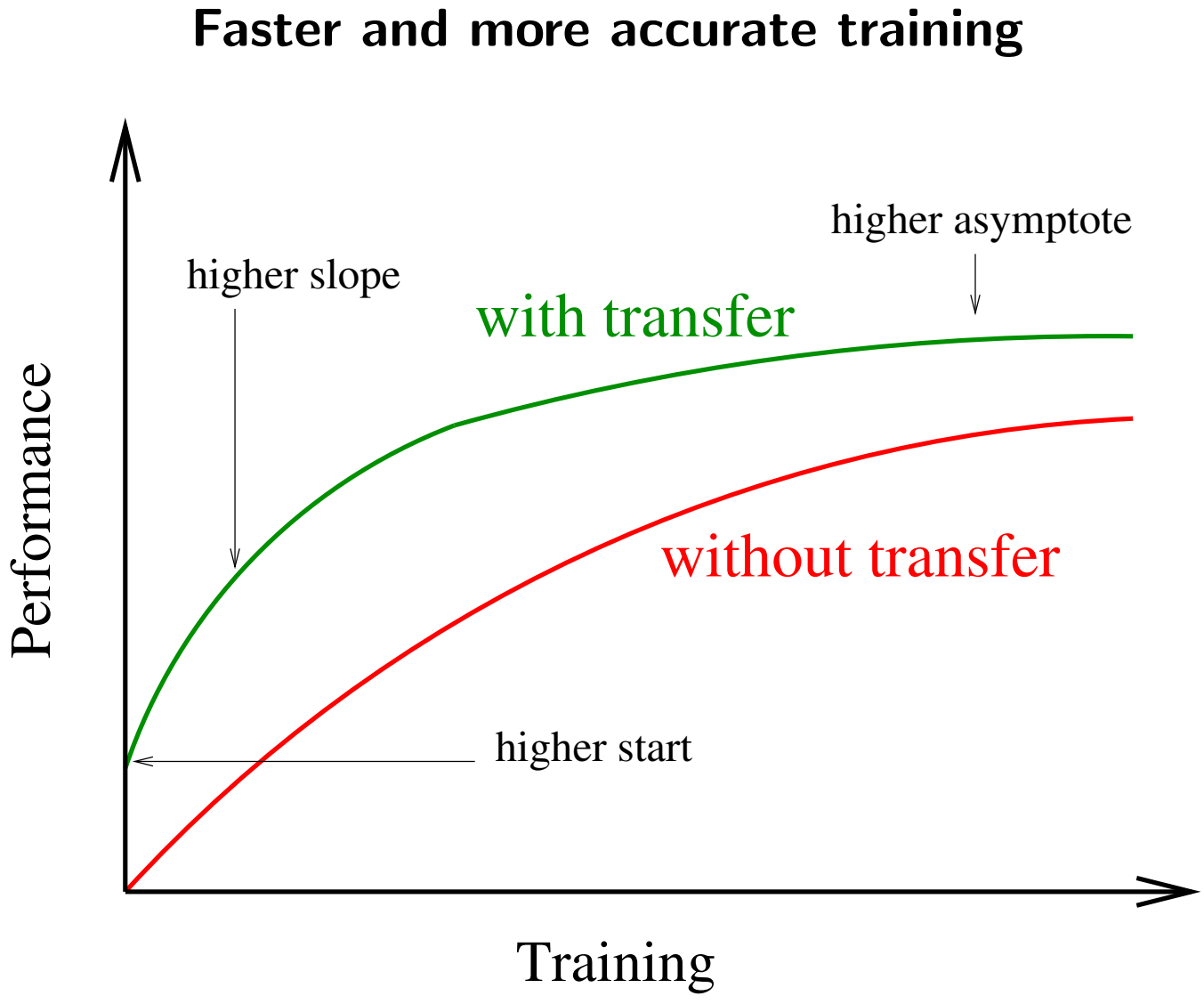


When Transfer Learning makes sense

transferring knowledge from problem A to problem B makes sense if

- the two problems have “similar” inputs
- we have much more training data for A than for B

What we may expect



Lesson 6

- adversarial attacks
- the data manifold
- autoencoders

How to fool a Neural Networks

Reducing distance from a target category

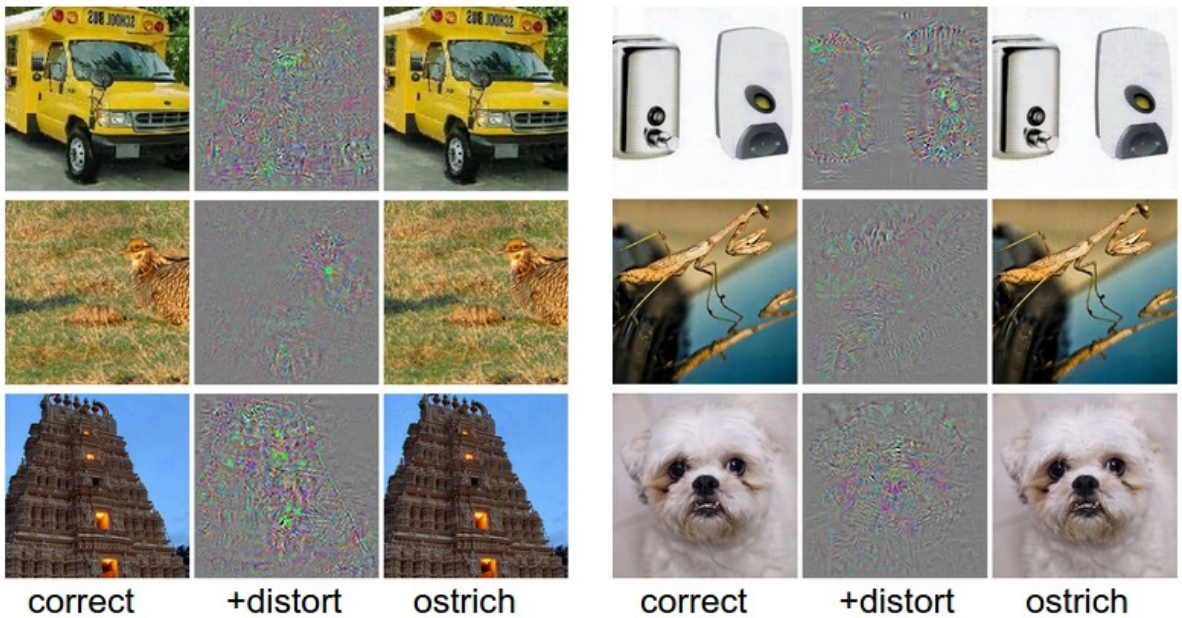
In an image classification framework, we can use the **gradient ascent technique** to increase, starting from noise or any given picture, the score of whatever class we want.

Demo!

We shall try to transform an elephant into a tigershark.

Fooling Neural Networks

Since we have many pixels, a tiny (imperceptible to humans!), consistent perturbation of all of them is able to fool the classifier.



Intriguing Properties of Neural Networks, C. Szegedy et al., 2013

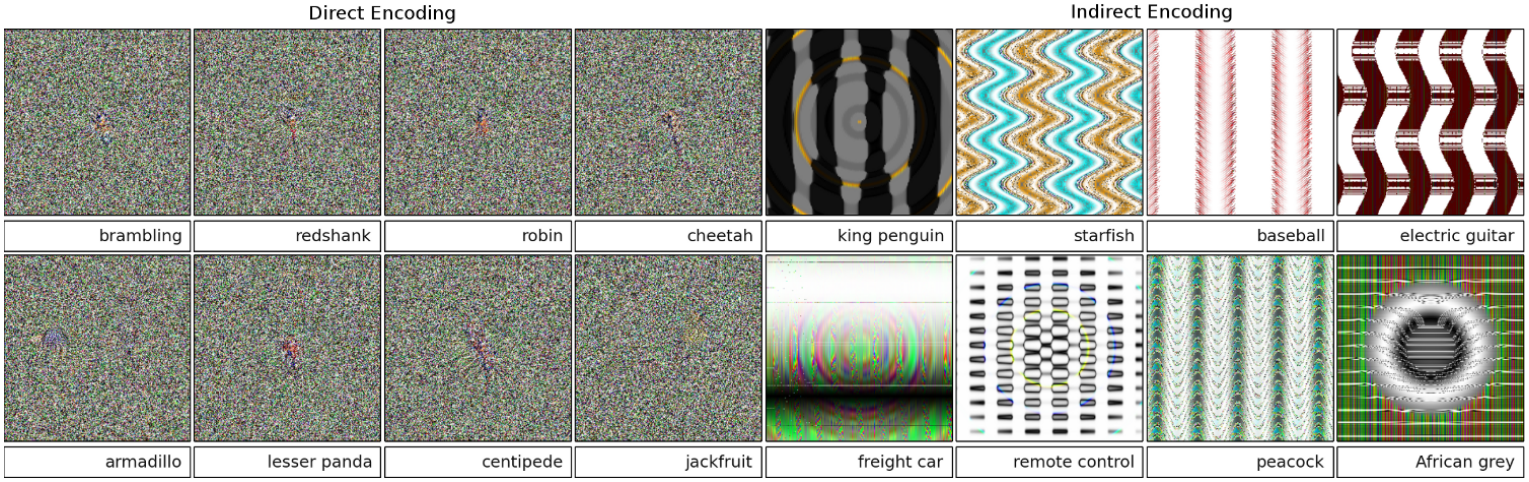
A different technique

The previous technique, being based on gradient ascent, requires the knowledge of the neural net in order to fool it.

We can do something similar using the network as a **black box**, for instance by means of evolutionary techniques

- Nguyen A, Yosinski J, Clune J. Deep Neural Networks are Easily Fooled. In Computer Vision and Pattern Recognition (CVPR '15), IEEE, 2015.

They were able to produce not only "noisy" adversarial images, but also geometrical examples with high regularities (meaningful for humans).



Evolutionary approach

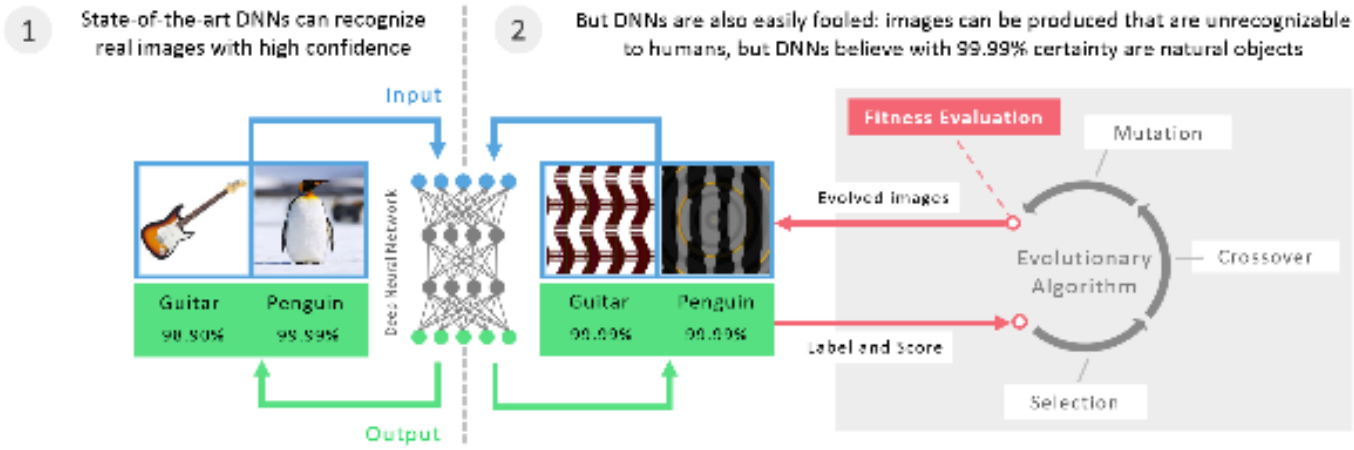
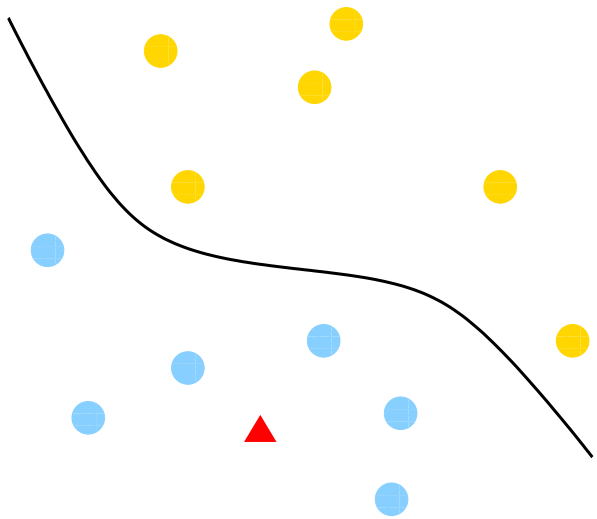


Figure 2. Although state-of-the-art deep neural networks can increasingly recognize natural images (*left panel*), they also are easily fooled into declaring with near-certainty that unrecognizable images are familiar objects (*center*). Images that fool DNNs are produced by evolutionary algorithms (*right panel*) that optimize images to generate high-confidence DNN predictions for each class in the dataset the DNN is trained on (here, ImageNet).

- start with a random population of images
- alternately apply selection (keep best) and mutation (random perturbation/crossover)

Why classification techniques are vulnerable



Why classification techniques are vulnerable

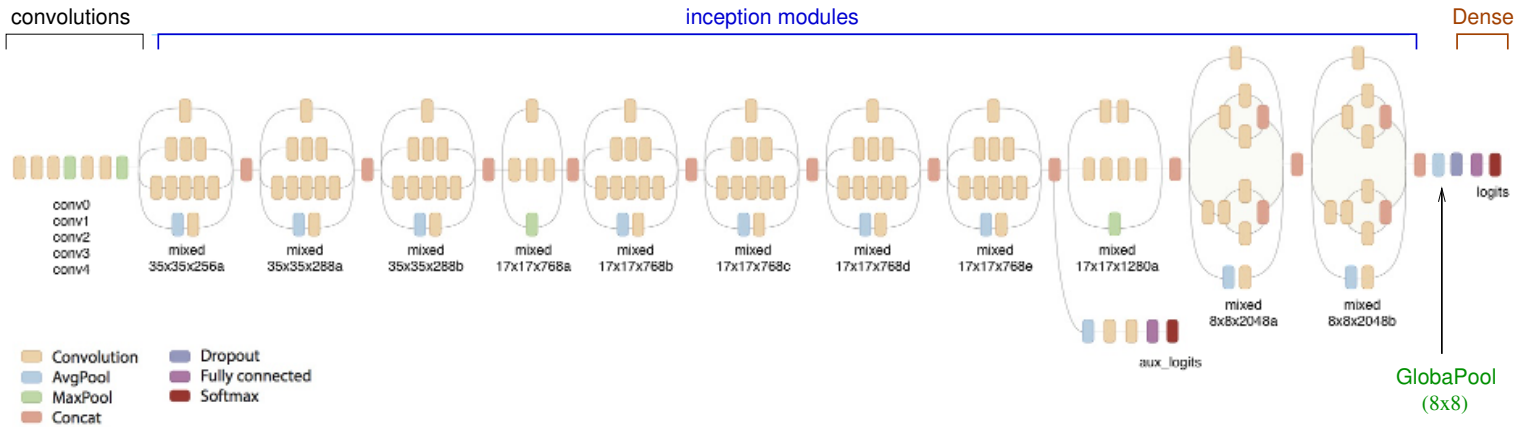
- **discriminating** between domains does not give us much knowledge about those domains.

  Difference between **generative** and **discriminative** approaches in machine learning.
- if data occupy a **low-dimensional** portion in the feature space, it is easy to modify features, to pass the borders of our discrimination boundaries.

In the next slides we try to better understand these concepts.

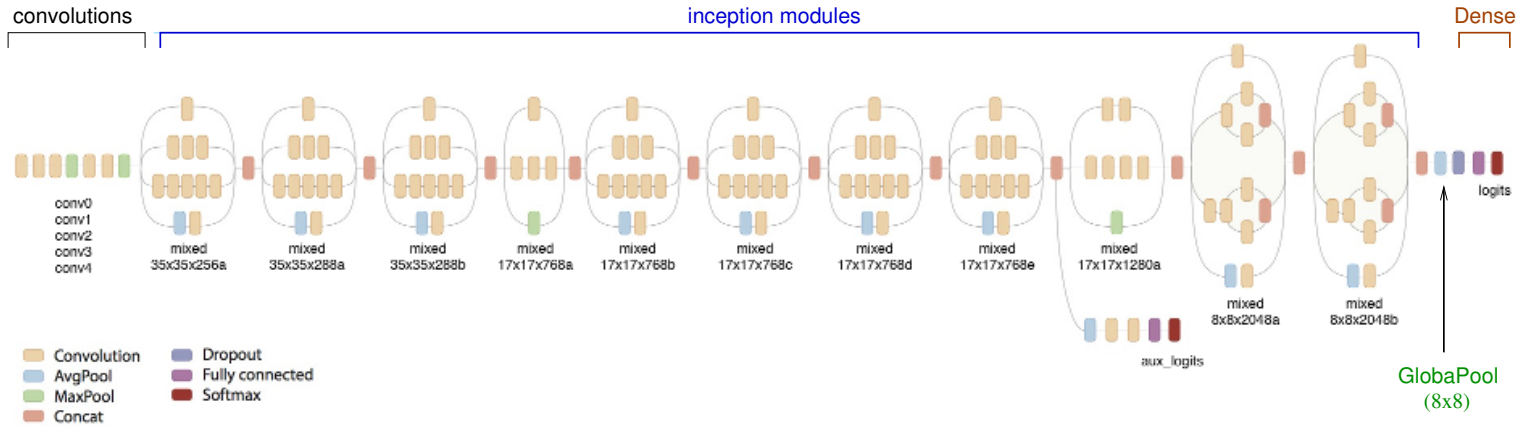
NN for image processing

A Neural Network for image processing has a structure of the following kind (this is Inception V3):



- a **long** sequence of convolutional layers, possibly organized in suitable modules (e.g. inception modules)
- a **short** (2 or 3) final sequence of dense layers

Feature extraction and exploitation



- with convolutional layers we **extract** interesting features from the input image, generating a different internal representation of data in terms of these features
- with the dense layers, we **exploit** these features in view of the particular problem we are aiming to solve (e.g. classification).

Reversing the representation

Reversing the representation of data makes sense as far as we are in extraction phase

- **we can syntesize interesting patterns** recognized by neurons of convolutional layers

We cannot expect to derive interesting information about categories from the information that the network uses to discriminate among them.

- **we cannot automatically synthesize a "cat"** (starting from a classifier; we shall see specific generative techniques in the next lesson)

Fooling networks

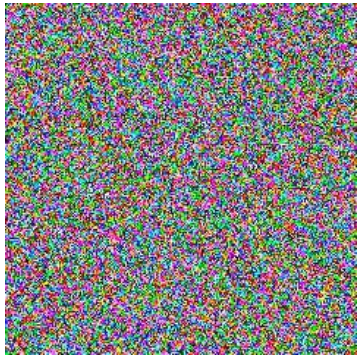
The previous point explains why we should not expect to be able to syntesize images of high level categories starting from a classifier.

But it does not explain why an almost imperceptible modification of an image is enough to fool the classifier.

To answer to this question we must discuss the dimensionality of the data manifold.

Dimensionality of data and feature space

If we generate an image at random, it looks like noise.
The probability of randomly generating an image having some sense for us **is null**.



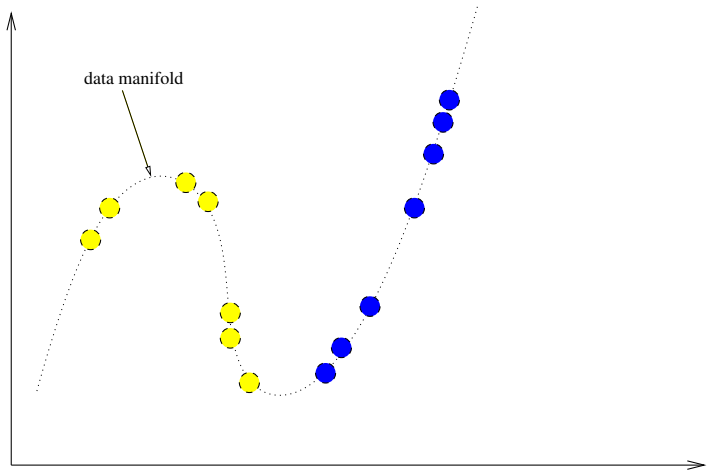
This means that "natural images" occupy a portion of the features space of almost **no dimension**.

Moreover, due their regularities, we expect them to be organized along some **disciplined, smooth-runnning** surfaces.

This is the so called **manifold** of data (in the features space).

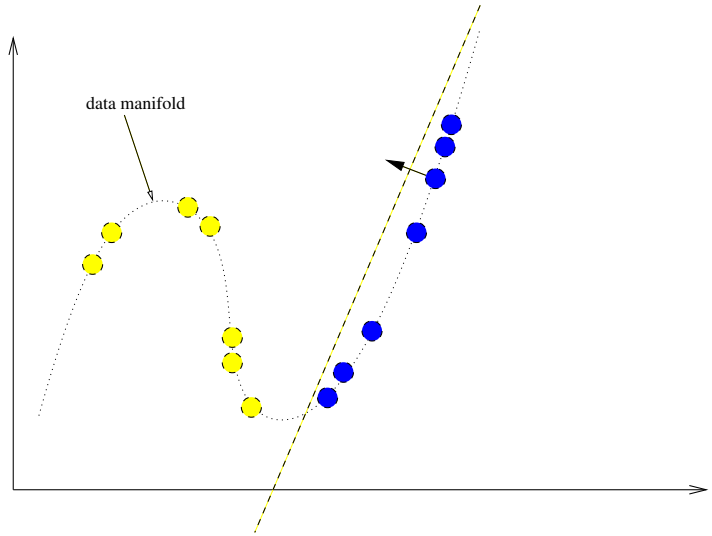
The Manifold issue

Suppose we have a space with two features, and our data occupy the manyfold of dimension 1, along the dotted line described in the following picture.



Suppose moreover that our data are splitted in two categories (yellow and blue) and we want to perform their classification

The Manifold issue



We have little knowledge of where the classifier will draw the boundary.
A tiny change in the data features may easily result in a miss-classification.

Now imagine the possibilities in a space with hundreads or thousands of dimensions.

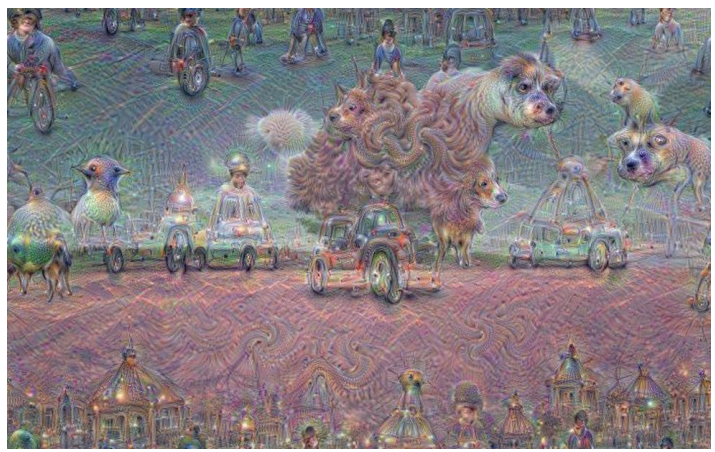
Observe that we are **escaping** from the actual data manifold.

A remark on inceptionism

The complexity of inceptionims consists in modifying an image **remaining inside the expected data manifold**.

This is difficult, since we have little knowldge about the actual data distribution in the feature space.

To this aim, deepdream generator exploits **regularization** techniques (smoothing, texture similarities, etc.) trying to obtain images similar (in statistical terms) to those in the training set.



Autoencoders

Introduction

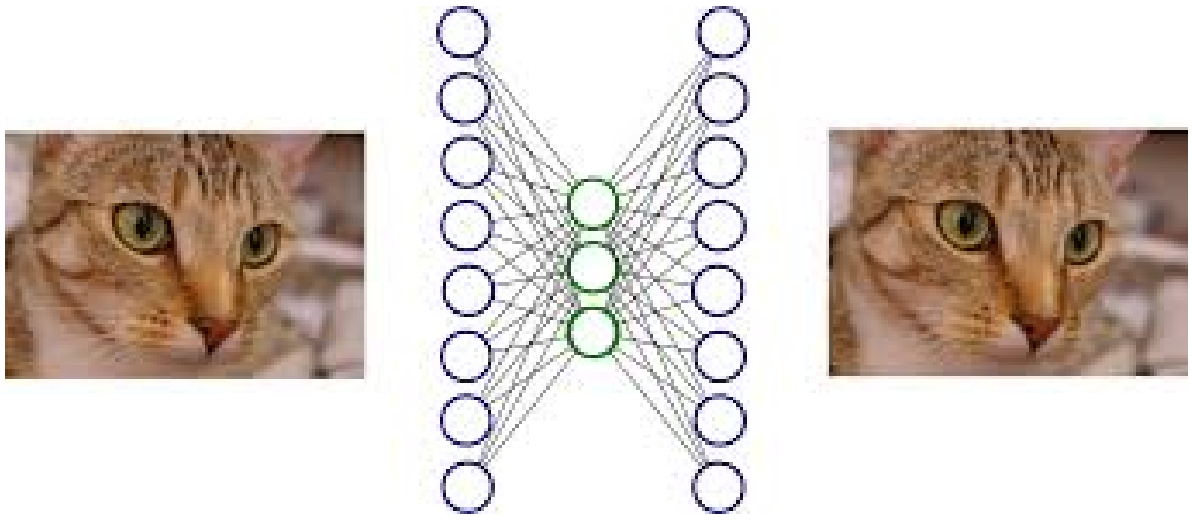
Two natural questions about the data manifold:

1. We said that (in frequent cases) the actual dimensionality of the data manifold is low in comparison with the latent space.
   Can we experimentally confirm this claim?
   Can we **compress** data?
2. For fooling networks, we synthesized new samples **outside** the actual data manifold.
   Is it possible to **automatically detect** this kind of **anomalies**?

To answer to these kind of questions it is worth to have a look at particular neural network models called **autoencoders**.

Autoencoders

An autoencoder is a net trained to reconstruct input data out of a learned internal representation



Usually, the internal representation has lower dimensionallity w.r.t. the input

Compression

Why is data compression possible, in general?

Because we exploit **regularities** (correlations) in the features describing input data.

If the input has a random structure (high **entropy**) no compression is possible

- random, lawless, uncompressible, high entropy
- ordered, lawfull, compressible, low entropy

Compression

Why is data compression possible, in general?

Because we exploit **regularities** (correlations) in the features describing input data.

If the input has a random structure (high **entropy**) no compression is possible

- random, lawless, uncompressible, high entropy
- ordered, lawfull, compressible, low entropy

A form of data compression

If (as usual) the internal layer has fewer units of the input, autoencoders can be seen as a form data compression. The compression is

- **data-specific**: it only works well on data with strong correlations (e.g. digits, faces, etc.) This is different from traditional data compression algorithms
- **lossy**: the output is degraded with respect to the input. This is different from textual compression algorithms, such as gzip
- **directly trained** on unlabeled data samples. We usually talk of self-supervised training

What are they good for?

Non so good for data compression, due to the lossy nature.

Applications to

- data denoising
- anomaly detection
- feature extraction (generalization of PCA)
- generative models (VAE)

Especially, an amusing and simple to understand topic.

Demo: autoencoders in Keras

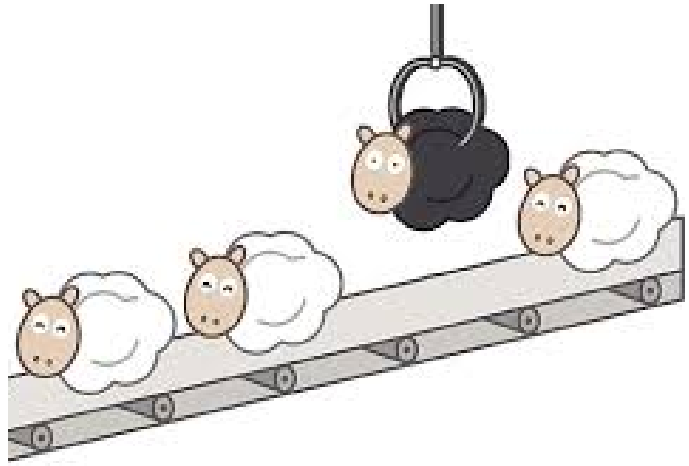
See Building autoencoders in Keras, on the Keras blog

Image denoising

Artificially add noise and train to reconstruct the original image

Denoising autoencoders inspired dropout

Anomaly detection



The latent encoding

The latent encoding of data is meant to capture the **main components** of the data features.

We can hence expect to **easily detect anomalies** by looking at points with **abnormal latent values**.

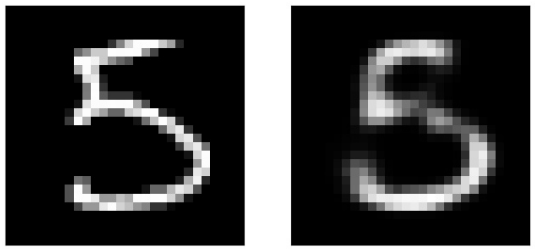
Equivalently, we may look at points with a **reconstruction below the expected quality**.

Anomaly detection

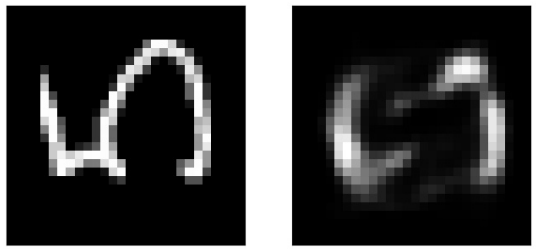
Autoencoding is **data specific**: the autoencoder works well on data similar to those it was trained on.

If applied on different data (anomaly), it will perform poorly.

Example on mnist data with the autoencoder of the previous demo.
mean loss = 0.105, std: 0.036



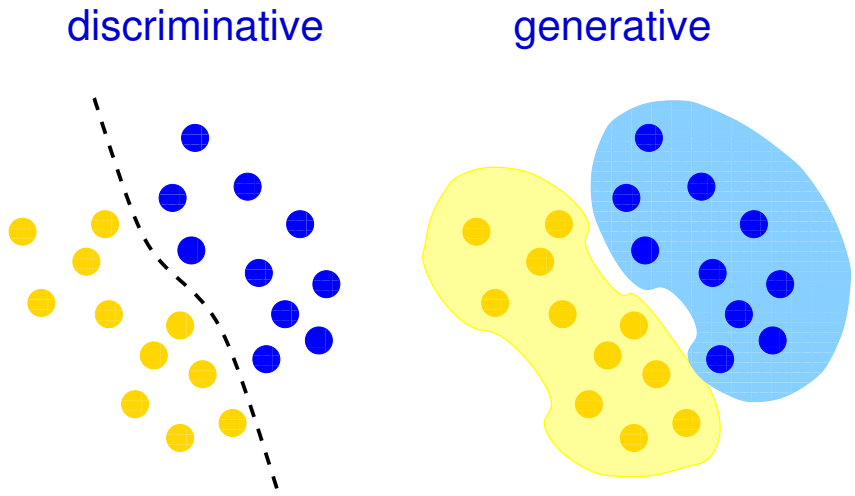
loss = 0.141 OK!



loss = 0.269 LIAR!

Suggested reading: Anomaly Detection with Autoencoders Made Easy

Generative Models



Generative Models

Generative Model: a model that tries to learn the actual distribution p_{data} of real data from available samples (training set).

Goal: build probability distribution p_{model} close to p_{data}.

We can either try to

- explicitly estimate the distribution
- build a generator able to sample according to p_{model}, possibly providing estimations of the likelihood

Why studying Generative Models?

- improve our knowledge on data and their distribution in the visible feature space
- improve our knowledge on the latent representation of data and the encoding of complex high-dimensional distributions
- tipical approach in many problems involving multi-modal outputs
- find a way to produce realistic samples from a given probability distribution
- generative models can be incorporated into reinforcement learning, e.g. to predict possible futures
- ...

Multi-modal output

In many interesting cases there is **no unique intended solution** to a given problem:

- add colors to a gray-scale image
- guess the next word in a sentence
- fill a missing information
- predict the next state/position in a game
- ...

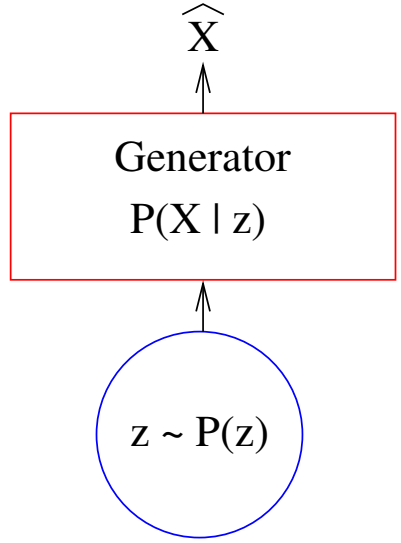
When the output is intrinsically multi-modal (and we do not want to give up to the possibility to produce multiple outputs) we need to rely on generative modeling.

Latent variables models

In latent variable models we express the probability of a data point X through **marginalization** over a vector of latent variables:

$$P(X) = \int P(X|z)P(z)dz \approx \mathop{\mathbb{E}}_{z \sim P(z)} P(X|z) \tag{1}$$

This simply means that we try to learn a way to sample X starting from a vector of values z (this is $P(X|z)$), where z is distributed with a **know prior** distribution $P(z)$. z is the **latent encoding** of X.

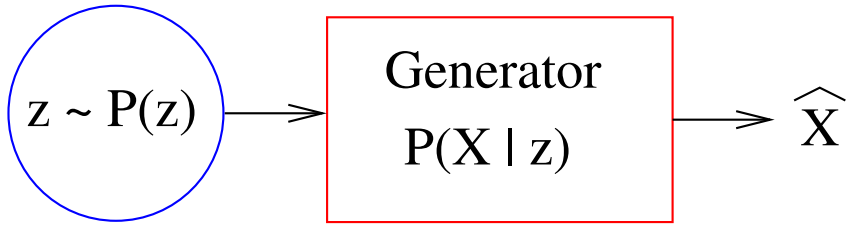


Generative models

There are four main classes of generative models:

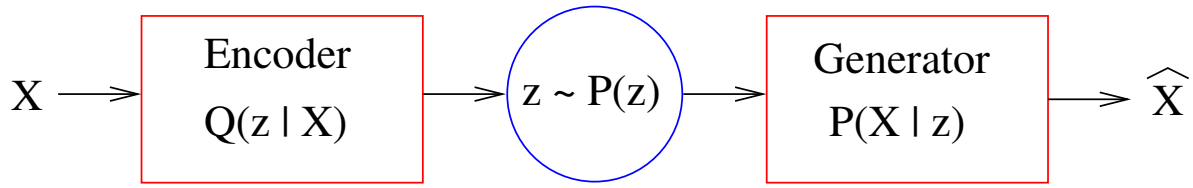
- compressive models
  - Variational Autoencoders (VAEs)
  - Generative Adversarial Networks (GANs)
- dimension preserving models
  - Normalizing Flows
  - Denoising Diffusion Models

The models differ in the way the generator is trained



VAE

In a Variational Autoencoder the generator is coupled with an **encoder** producing a latent encoding z given X. This will be distributed according to an inference distribution $Q(z|X)$.

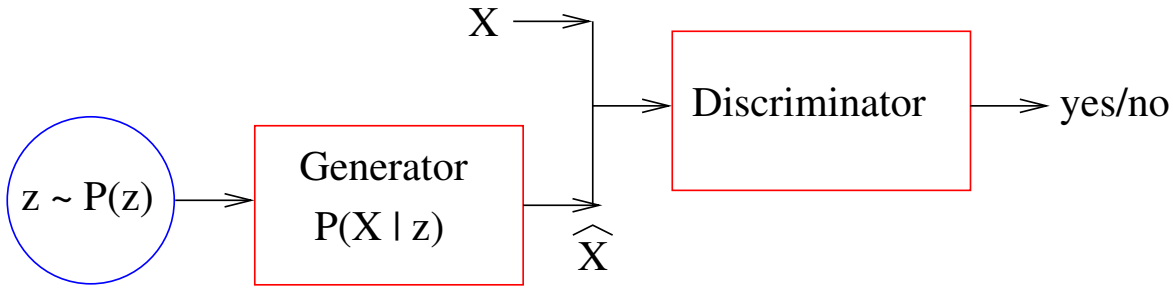


The loss function aims to:

- minimize the reconstruction error between X and $\hat{X}$
- bring the marginal inference distribution $Q(z)$ close to the prior $P(z)$

GAN

In a Generative Adversarial Network, the generator is coupled with a **discriminator** trying to tell apart **real** data from **fake** data produced by the generator.



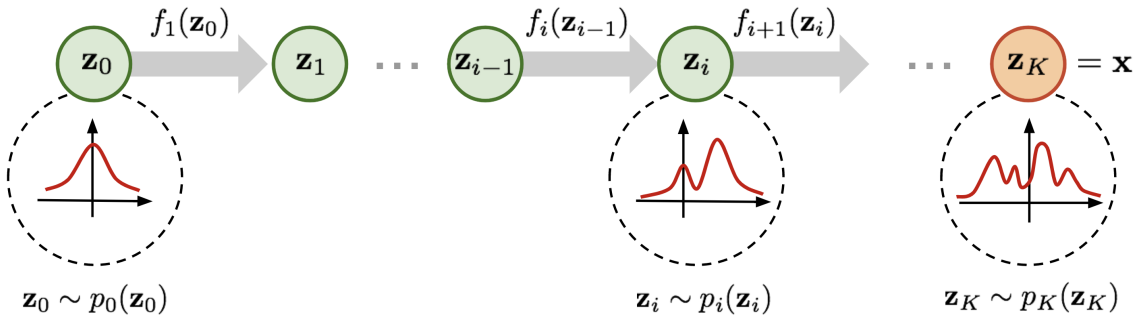
Detector and Generator are trained together.

The loss function aims to:

- instruct the detector to spot the generator
- instruct the generator to fool the detector

Normalizing Flows

In Normalizing Flows the generator is split into a long chain of *invertible* transformations.



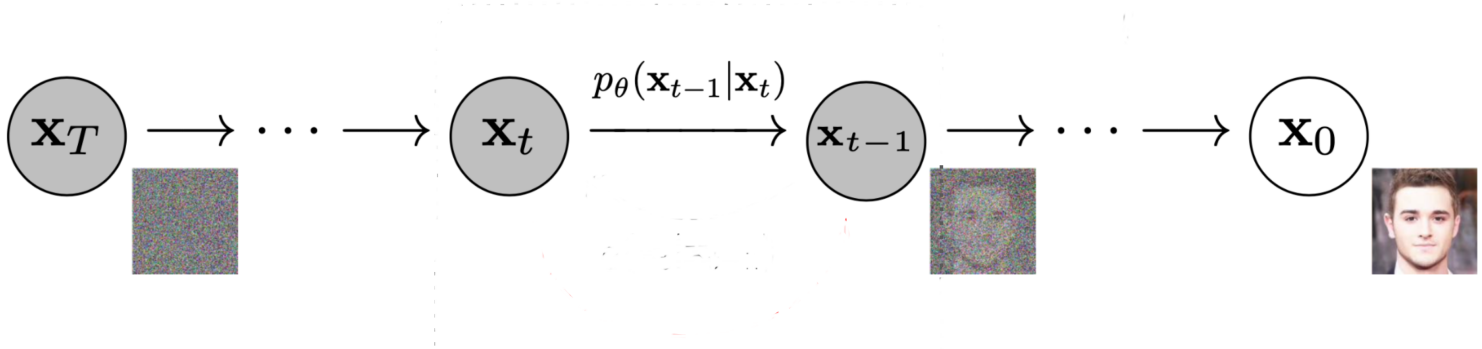
The network is trained by maximizing loglikelihood.

- **Pros**: it allows a precise computation of the resulting loglikelihood
- **Cons**: the fact of restricting to invertible transformation limit the expressiveness of the model

Diffusion Models

In Diffusion Models the latent space is understood as a strongly noised version of the image to be generated.

The generator is split into a long chain of *denoising steps*, where each step t attempts to remove gaussian noise with a given variance σ_t.



We train a single network implementing the denoising operation, parametric in σ_t.

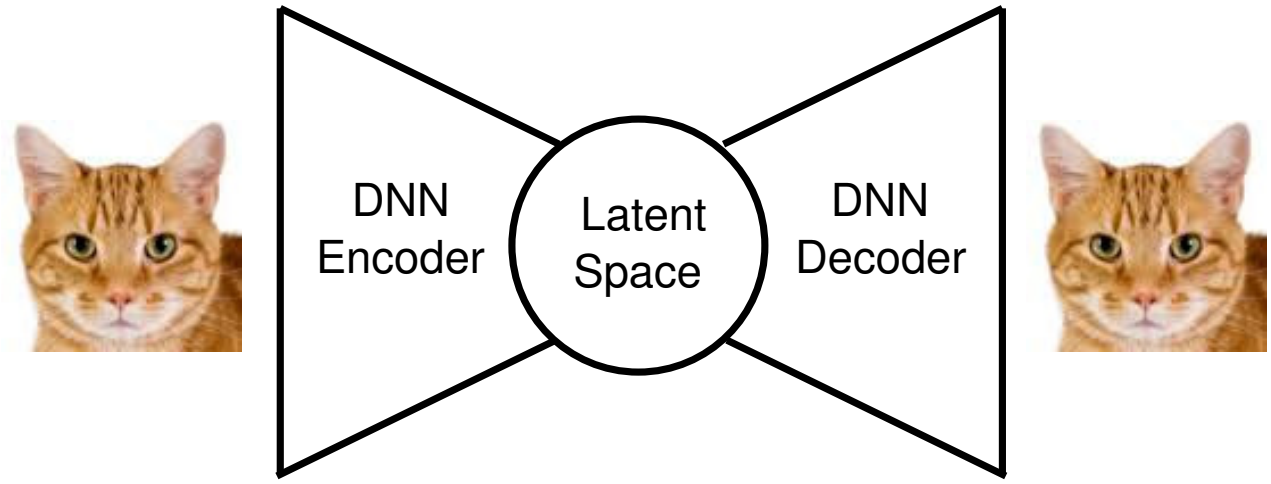
Variational Autoencoders

Suggested reading:

A survey on Variational Autoencoders from a GreenAI Perspective

The problem with the deterministic autoencoder

An autoencoder is a net trained to reconstruct input data out of a learned internal representation (e.g. minimizing quadratic distance)

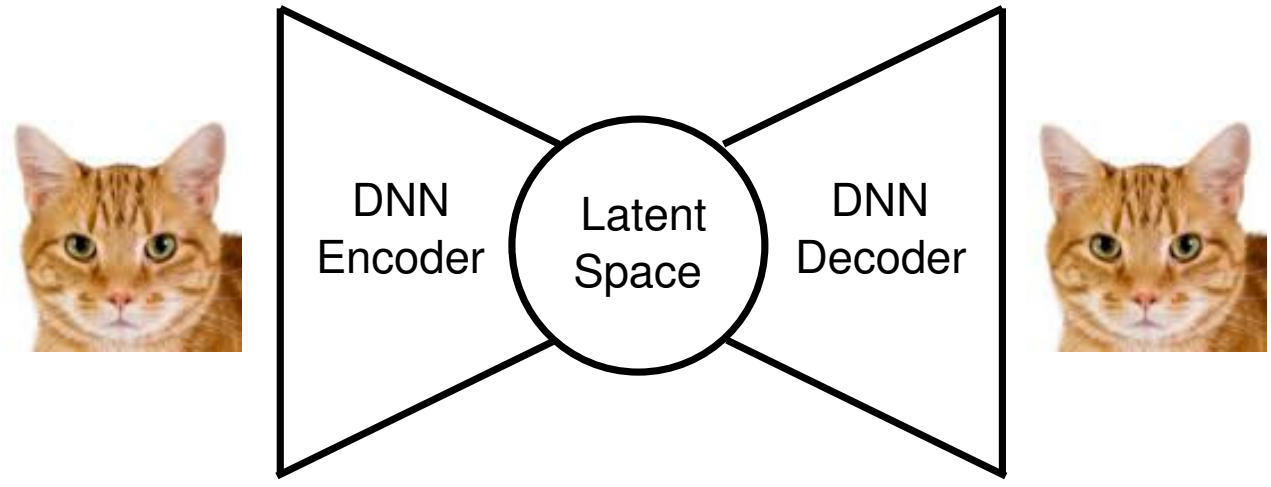


Can we use the decoder to **generate** data by **sampling** in the latent space?

No, since we do not know the distribution of latent variables.

The problem with the deterministic autoencoder

An autoencoder is a net trained to reconstruct input data out of a learned internal representation (e.g. minimizing quadratic distance)

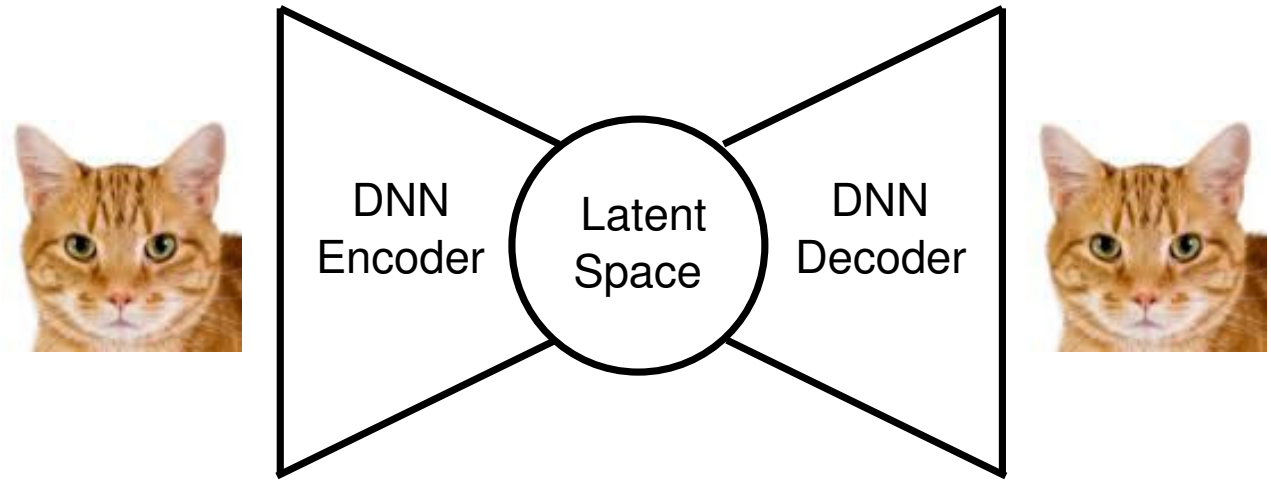


Can we use the decoder to **generate** data by **sampling** in the latent space?

No, since we do not know the distribution of latent variables.

The problem with the deterministic autoencoder

An autoencoder is a net trained to reconstruct input data out of a learned internal representation (e.g. minimizing quadratic distance)

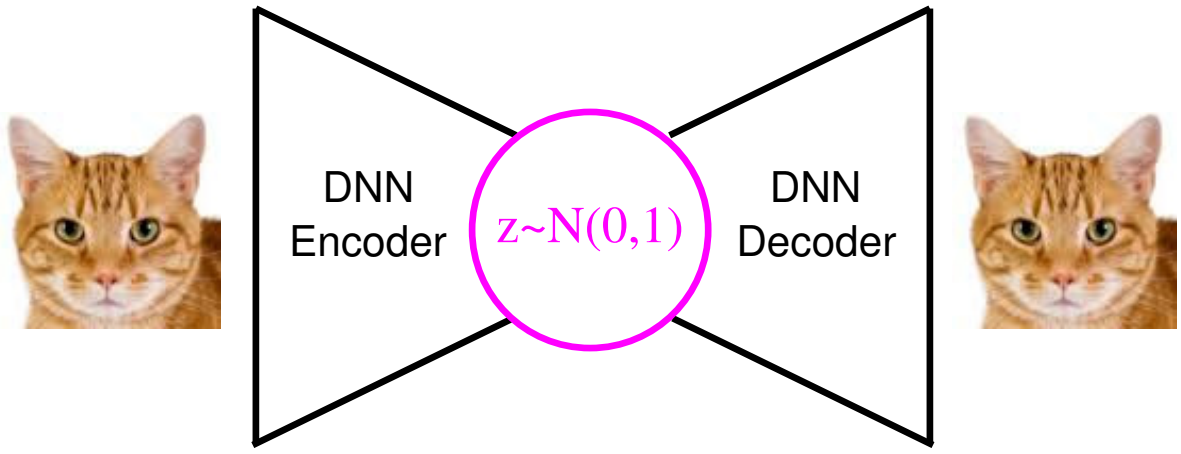


Can we use the decoder to **generate** data by **sampling** in the latent space?

No, since we do not know the distribution of latent variables.

Variational autoencoder

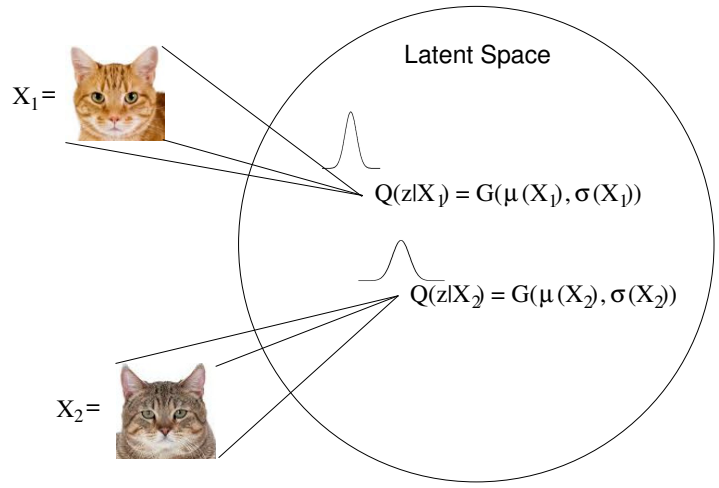
In a Variational Autoencoder (VAE) we try **to force** latent variables to have a known prior distribution $P(z)$ (e.g. a Normal distribution)



If the distribution computed by the generator is $Q(z|X)$ we try to force the marginal distribution $Q(z) = \mathbb{E}_{X \sim P_{data}} Q(z|X)$ to look like a normal distribution.

Different moments for each point

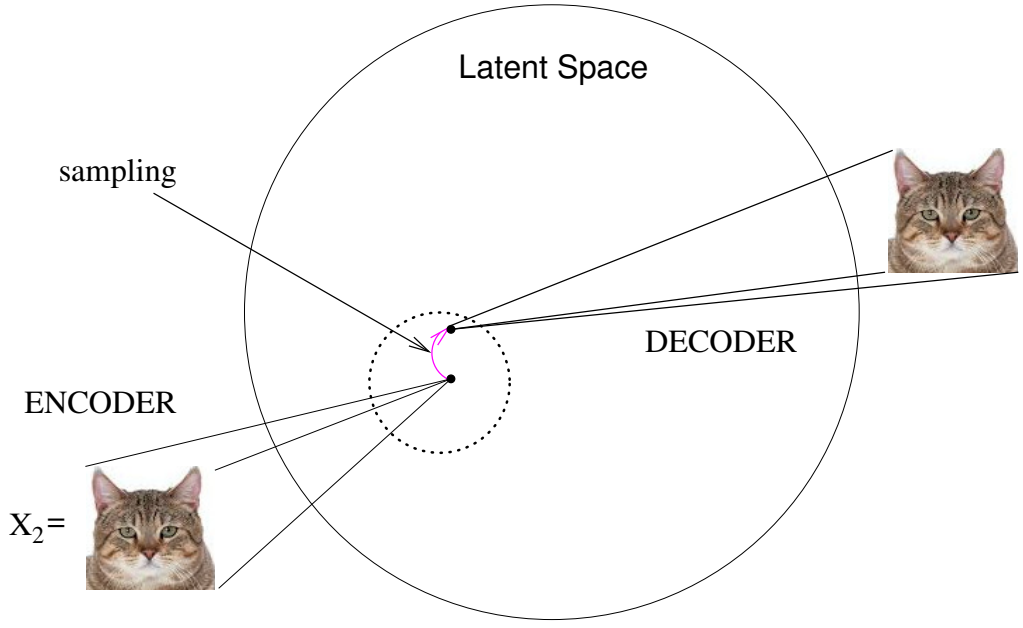
We assume $Q(z|X)$ has a Gaussian distribution $G(\mu(X), \sigma(X))$ with different moments for each different input X.



The values $\mu(X), \sigma(X)$ are both computed by the generator, that is hence returning an **encoding** $z = \mu(X)$ and a variance $\sigma(X)$ around it, expressing the portion of the latent space essentially encoding an information similar to X.

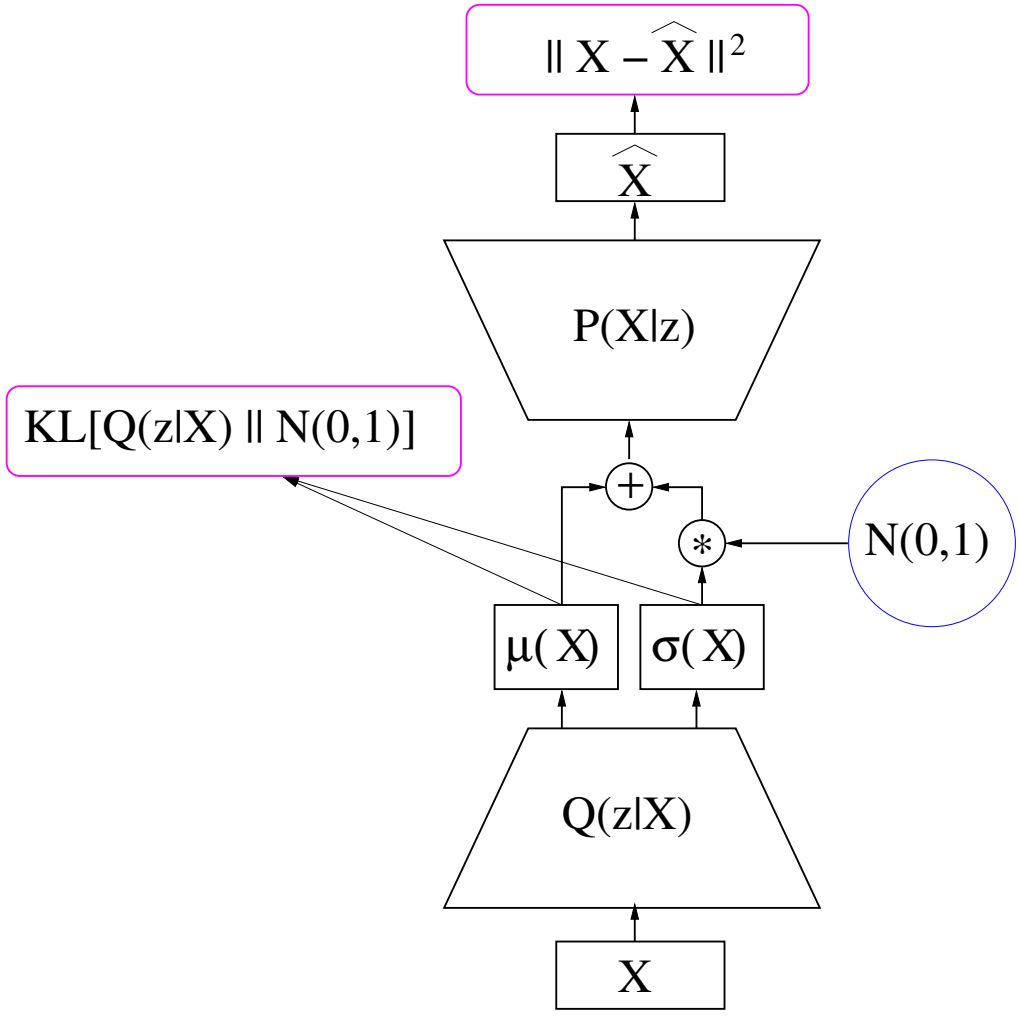
Sampling in the latent space

During training, we sample around $\mu(X)$ with the computed $\sigma(X)$ before passing the value to the decoder



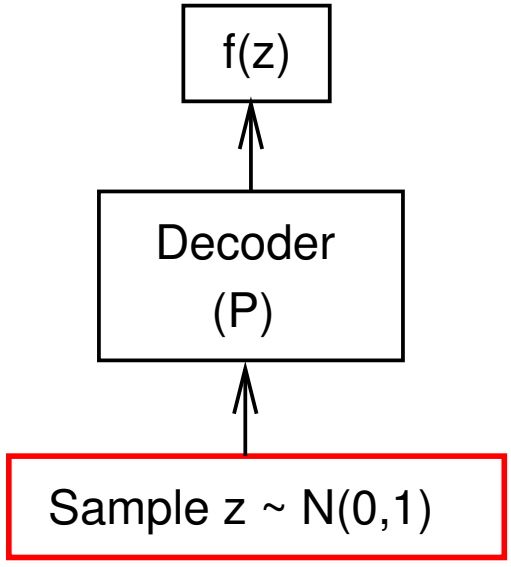
Among other things, sampling add noise to the encoding, improving its robustness.

The full picture

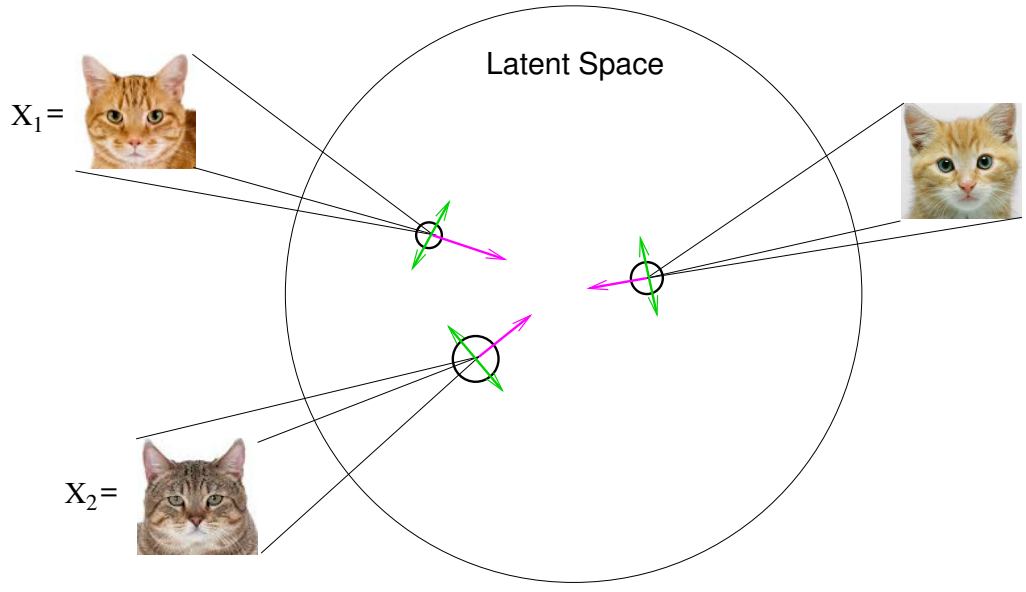


Generation of new samples

$\mu(X)$ and $\Sigma(X)$ are not used to generate new samples from the input domain (we have no X)



The effect of KL-divergence



The effect of the KL-divergence on latent variables consist in

- pushing $\mu_z(X)$ towards 0, so as to center the latent space around the origin
- push $\sigma_z(X)$ towards 1, augmenting the "coverage" of the latent space, essential for generative purposes.

Problems with VAE

- balancing loglikelihood and KL regularizer in the loss function
- variable collapse phenomenon
- marginal inference vs prior mismatch
- blurriness (aka variance loss)

VAE Demo

DEMO!

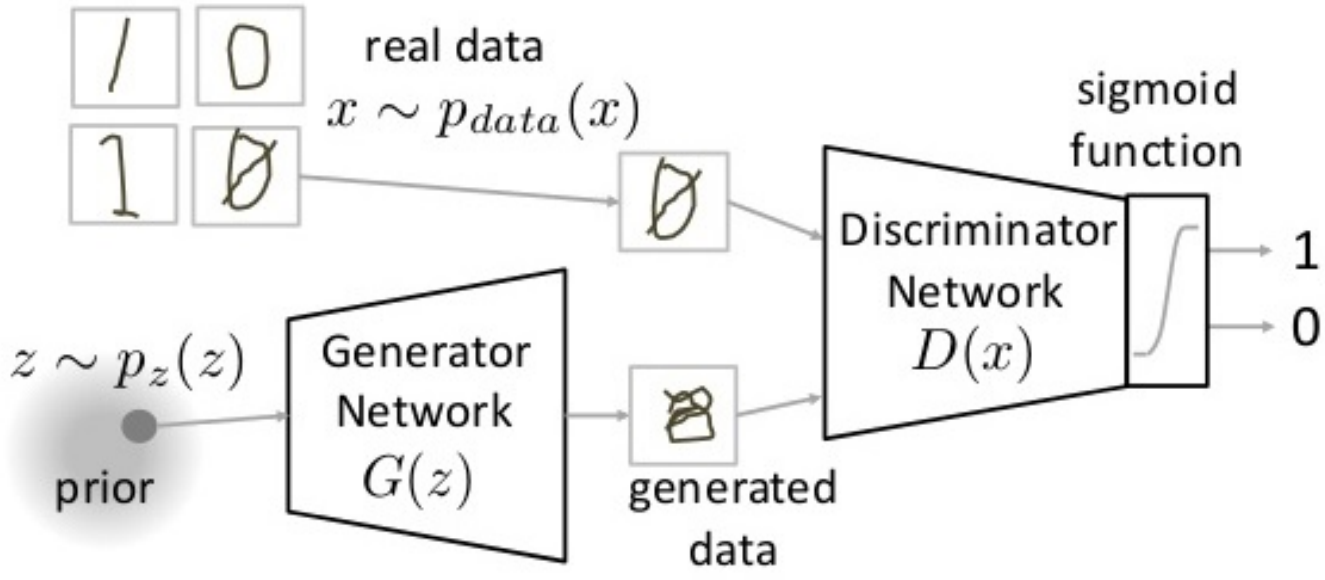
Generative Adversarial Networks

Suggested reading:

NIPS 2016 Tutorial: Generative Adversarial Networks

The GAN approach: a two player game

A game between the generator and the discriminator



Generative Adversarial Networks I.J.Goodfellow et al., 2014

A Min Max game

$$Min_G Max_D V(D, G)$$

$$V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[log\, D(x)] + \mathbb{E}_{z \sim p_z(z)}[log\,(1 - D(G(z)))]$$

- $\mathbb{E}_{x \sim p_{data}(x)}[log\, D(x)]$ = negative cross entropy of the discriminator w.r.t the true data distribution
- $\mathbb{E}_{z \sim p_z(z)}[log\,(1 - D(G(z)))]$ = negative cross entropy of the "false" discriminator w.r.t the fake generator

Training

Alternately train the discriminator, freezing the generator, and the generator freezing the discriminator:

---

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

An example

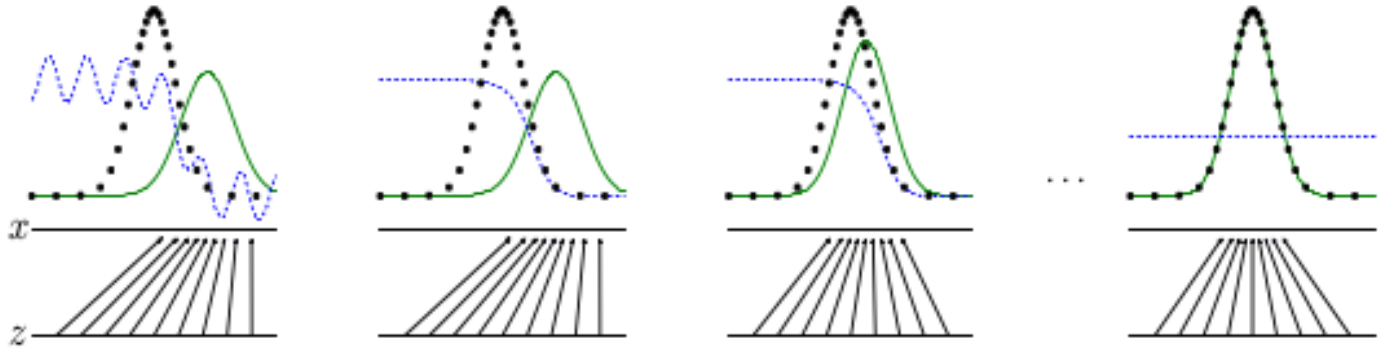


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

A simple demo

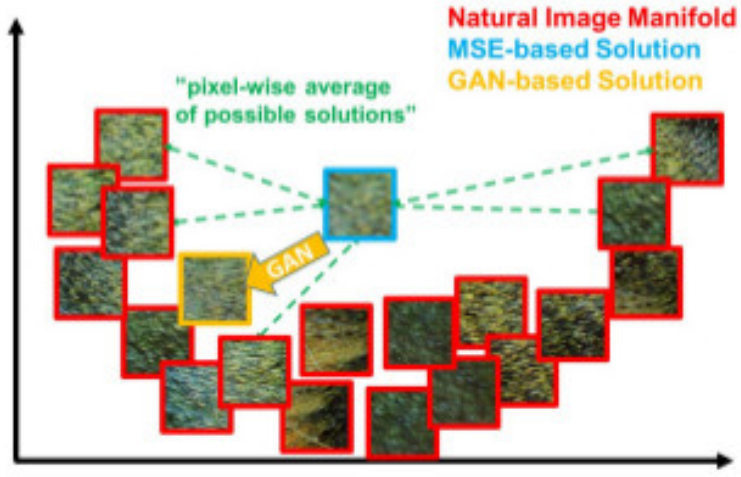
Demo!

Stay inside the data manifold

Patches from the natural image manifold (red) and super-resolved patches obtained with MSE (blue) and GAN (orange).

Pixel-wise average of possible solutions could produce images outside the actual data manifold.

GAN drives the reconstruction towards the natural image manifold producing perceptually more convincing solutions.



picture from Photo-Realistic Single Image Super-Resolution. C.Ledig et al., 2016.

An application: super-resolution

Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]

Forcing to operate a choice - instead of mediating - could result in sharper images

Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. C.Ledig et al., 2016.

An application: face generation

Goal: Generation of plausible realistic photographs of human faces.

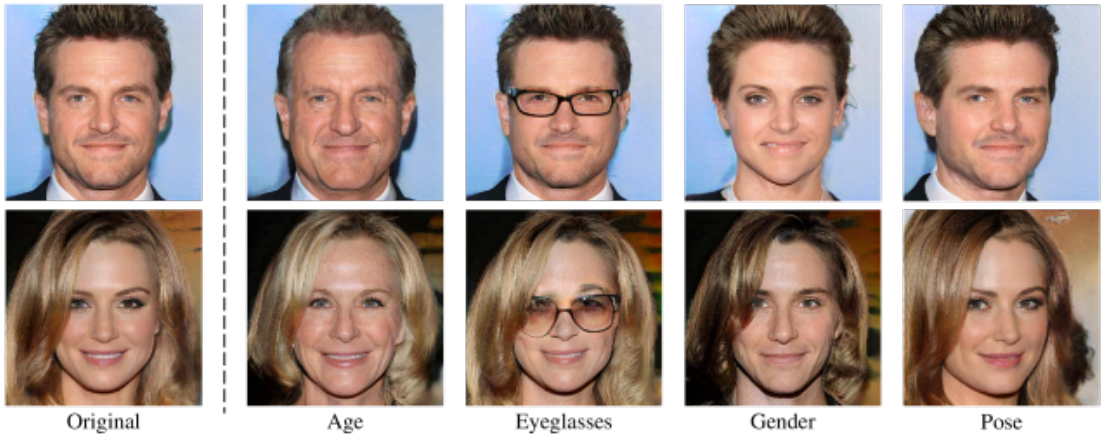
Face generation video by Nvidia

Problems with Gans

- the fact that the discriminator get fooled does not mean the fake is good (neural networks are easily fooled)
- problems with counting, perspective, global structure, ...
- **mode collapse**: generative specialization on a good, fixed sample

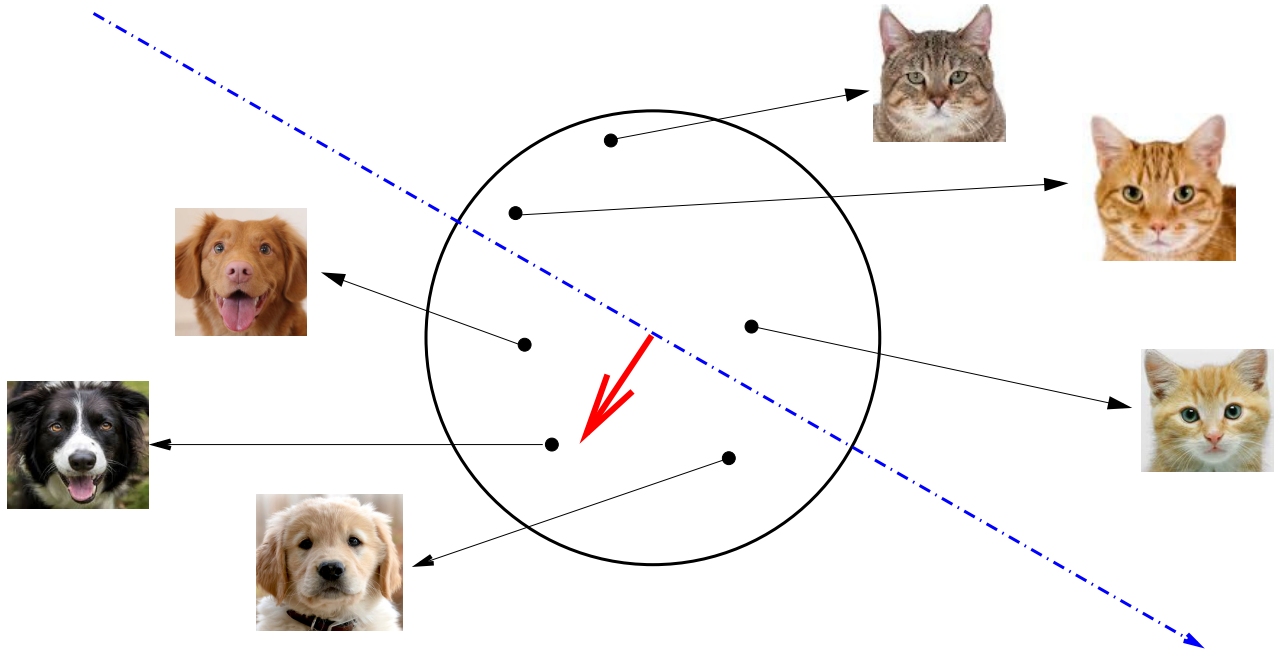
See Ian Goodfellows' slides

Latent space exploration



Interpreting the Latent Space of GANs for Semantic Face Editing

Attribute editing



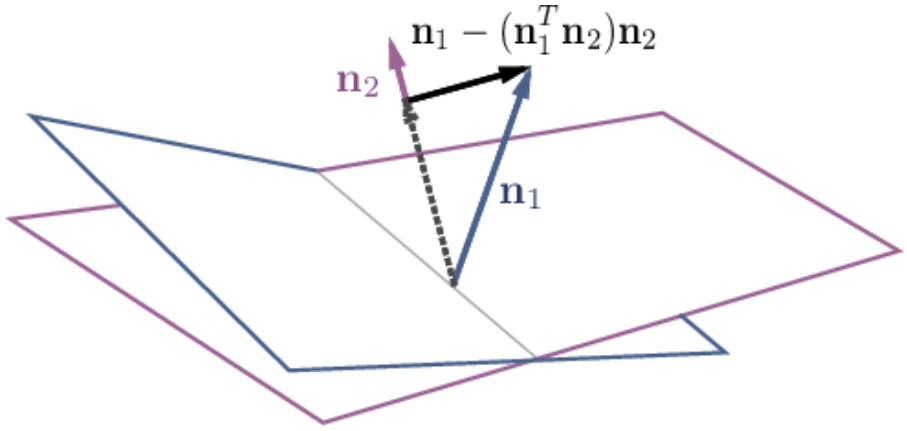
Key ideas behind Representation Learning

The generative process is **continuous**: a small deplacement in the latent space produces a small modification in the visible space.

Real-world data depends on a relatively **small number of explanatory factors of variation** (latent features) providing compressed internal representations.

Understanding these features we may define **trajectories** producing desired alterations of data in the visible space.

Entanglement and disentanglement

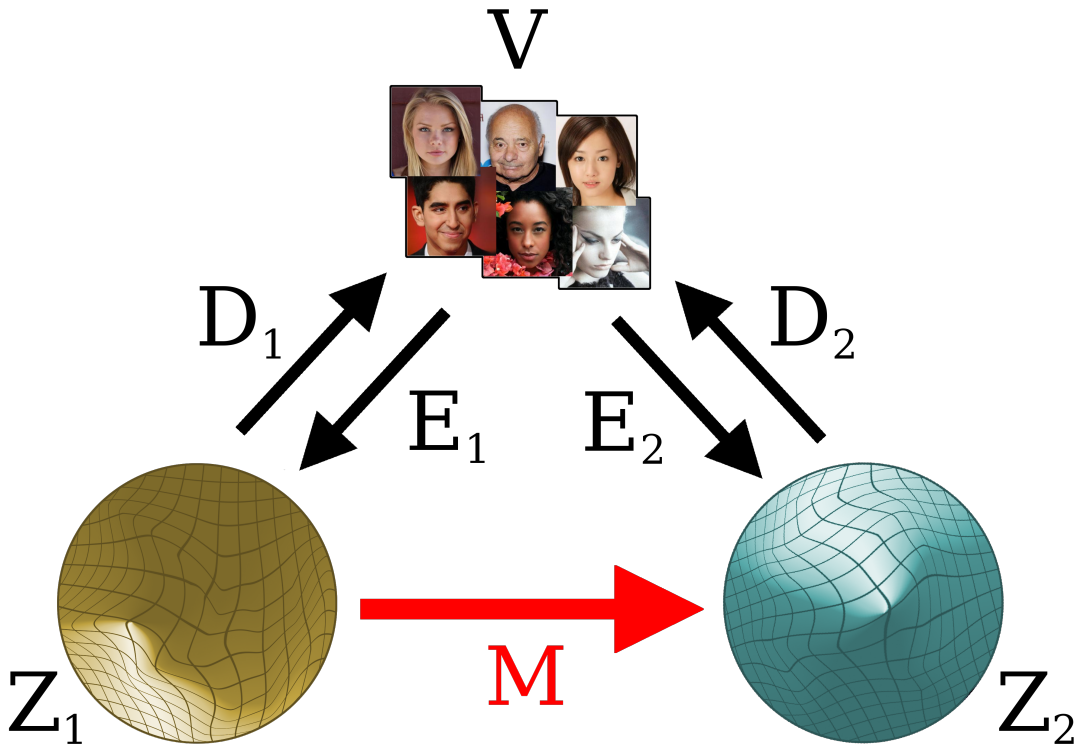


When there is more than one attribute, editing one may affect another since some semantics can be coupled with each other (entanglement).

To achieve more precise control (disentanglement), we can use projections to force the different directions of variation to be orthogonal to each other.

Comparing spaces

Learn a direct map between spaces



Comparing the latent space of generative models

Achievements

We can pass from a latent space to another by means of a simple **linear map** preserving most of the content.

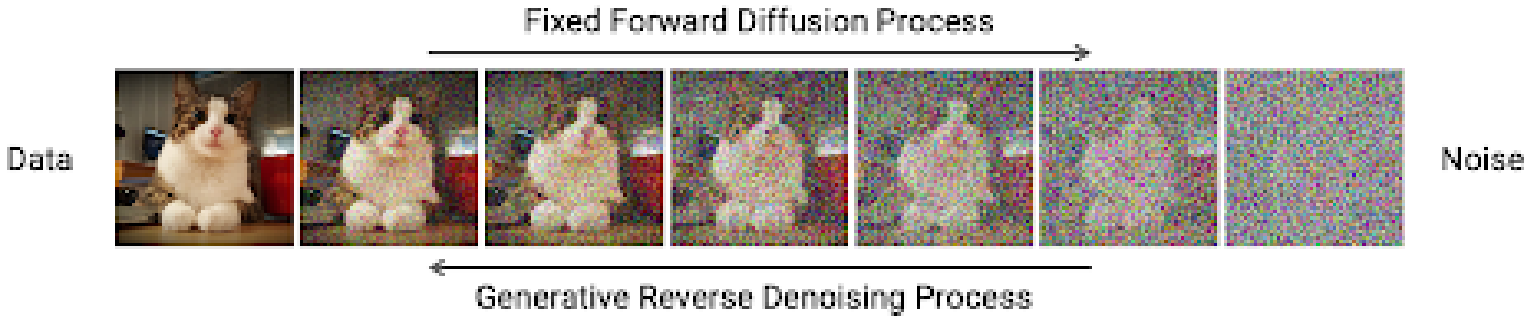
The organization of the latent space seems to be independent from

- the training process
- the network architecture
- the learning objective: GAN and VAE share the same space!

The map can be defined by a small set of points common to the two spaces: the **support set**. Locating these points in the two spaces is enough to define the map.

See also my blog for a discussion.

Diffusion models



Suggested reading: What are diffusion models

The denoising network

The denoising network implements the inverse of the operation of adding a given amount of noise to an image (direct diffusion).

The denoising network takes in input:

1. a noisy image x_t
2. a signal rate α_t expressing the amount of the original signal remaining in the noisy image

and try to predict the noise in it:

$$\epsilon_\theta(x_t, \alpha_t)$$

The predicted image would be:

$$\hat{x_0} = (x_t - \sqrt{1-\alpha_t} \cdot \epsilon_\theta(x_t, \alpha_t))/\sqrt{\alpha_t}$$

Training step

- take an input image x_0 in the training set and normalize it
- consider a **signal ratio** α_t
- generate a random noise $\epsilon \sim N(0, 1)$
- generate a noisy version x_t of x_0 defined as

$$x_t = \sqrt{\alpha_t} \cdot x_0 + \sqrt{1 - \alpha_t} \cdot \epsilon$$

- let the network predict the noise $\epsilon_\theta(x_t, \alpha_t)$ from the noisy image x_t and the signal ratio α_t
- train the network to minimize the prediction error, namely

$$\|\epsilon - \epsilon_\theta(x_t, \alpha_t)\|$$

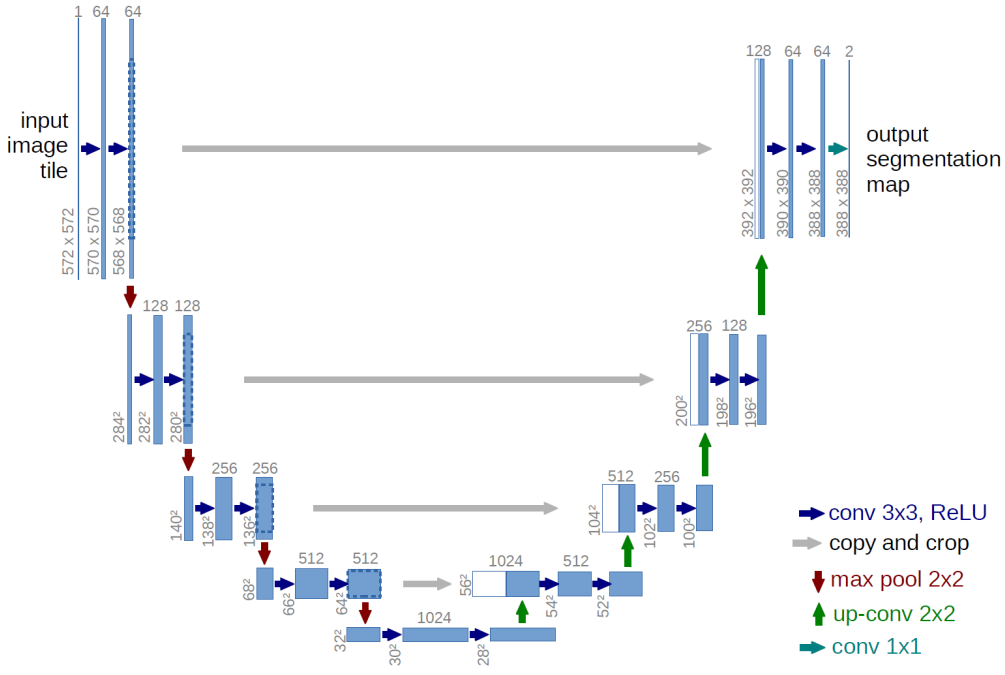
Sampling procedure

- fix a sheduling $\alpha_T > \alpha_{T-1} > \ldots\ldots > \alpha_1$
- start with a random noisy image $x_T \sim N(0,1)$
- for t in T...1 do:
- compute the predicted error $\epsilon_\theta(x_t, \alpha_t)$
- compute $\hat{x_0} = (x_t - \sqrt{1-\alpha_t} \cdot \epsilon_\theta(x_t, \alpha_t))/\sqrt{\alpha_t}$
- if $t \neq 0$, obtain x_{t-1} reinjecting noise at rate α_{t-1}, namely

$$x_{t-1} = \sqrt{\alpha_{t-1}} \cdot \hat{x_0} + \sqrt{1-\alpha_{t-1}} \cdot \epsilon$$

Network architecture

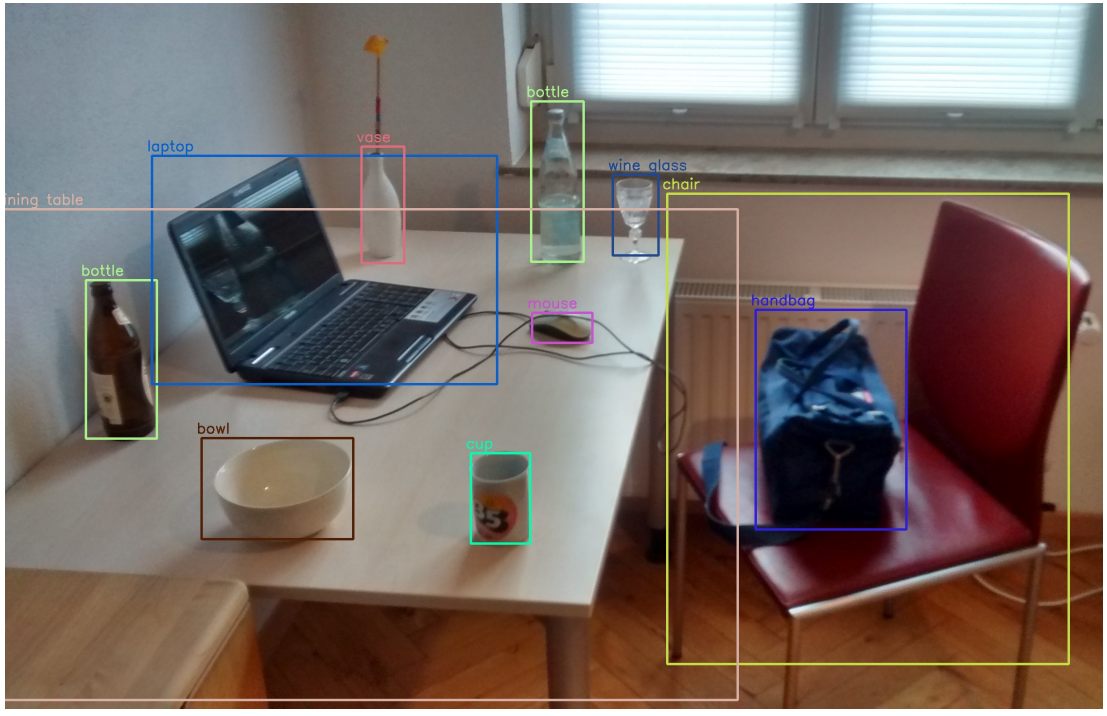
Use a (conditional) Unet!



Object Detection & Segmentation

Object detection

Identify the class and **position** of objects in an image, typically returning a boundary box for each prediction.



Suggested reading: Object detection algorithms

Semantic Segmentation

Classify **each pixel** in an image according to the object category it belongs to.

Suggested reading: A short guide to semantic segmentation

Both object detection and semantics segmentation are basic techniques to *understand the content* on an image.

Datasets

Building supervised training sets is expensive, since it requires a complex human operation to create ground truth annotations.

Many datasets provide annotations both for detection and segmentation tasks.

Pascal VOC 2012

PASCAL Visual Object Classes

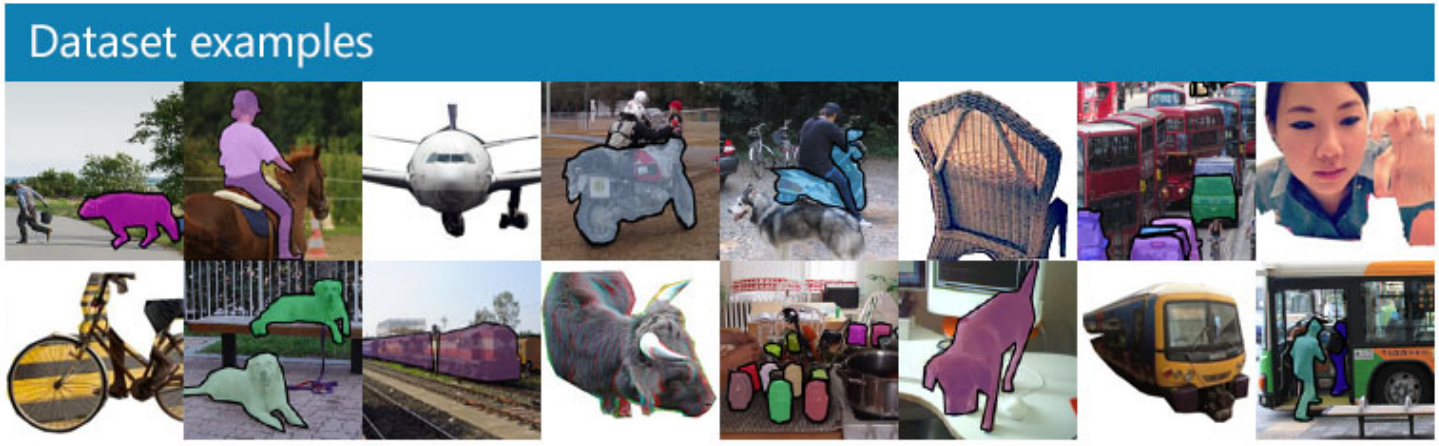
20 classes:

- Person: person
- Animal: bird, cat, cow, dog, horse, sheep
- Vehicle: aeroplane, bicycle, boat, bus, car, motorbike, train
- Indoor: bottle, chair, dining table, potted plant, sofa, tv/monitor

The train/val data has 11,530 images containing 27,450 ROI annotated objects and 6,929 segmentations.

Coco dataset

Coco: a large-scale object detection, segmentation, and captioning dataset composed of over 200K labeled images, spanning 80 categories.
There is an annual competiton



Cityscapes dataset

Semantic understanding of urban street scenarios

A large-scale dataset containing stereo video sequences recorded in street scenes from 50 different cities, with high quality pixel-level annotations of 5K frames, and a larger set of 20K weakly annotated frames.

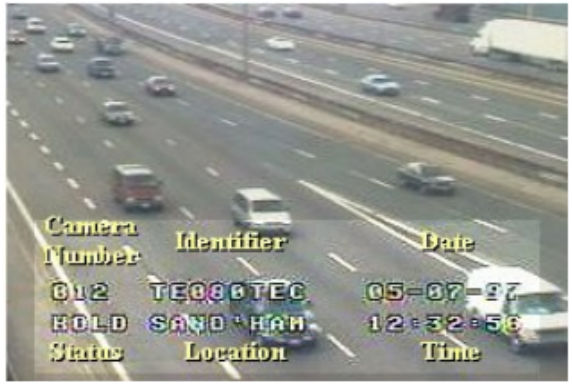
More datasets

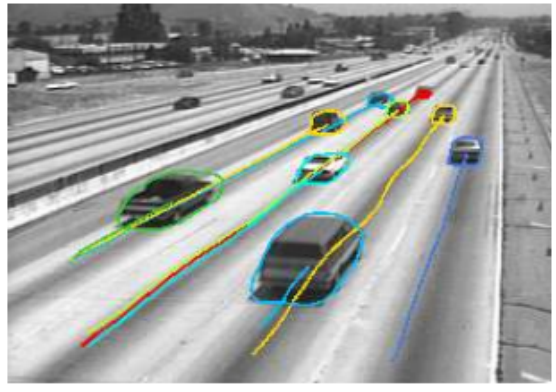
- Syntia a collection of photo-realistic frames rendered from a **virtual** city and **precise** pixel-level semantic annotations for 13 classes: misc, sky, building, road, sidewalk, fence, vegetation, pole, car, sign, pedestrian, cyclist, lane-marking.
- ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes. ScanNet is an RGB-D video dataset containing 2.5 million views in more than 1500 scans, annotated with 3D camera poses, surface reconstructions, and instance-level semantic segmentations.
- Sun RGB-D: 10,000 RGB-D images of densely annotated indoor scenes, includes 146617 2D polygons and 58657 3D bounding boxes with accurate object orientations, as well as a 3D room layout and category for scenes.

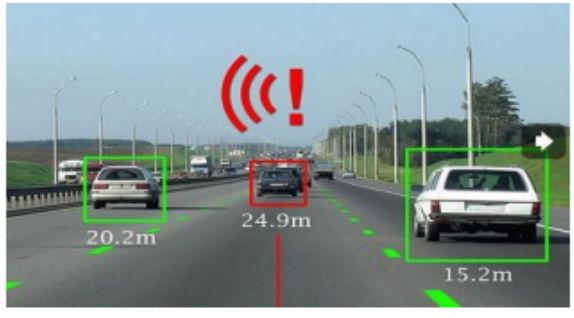
Some applications

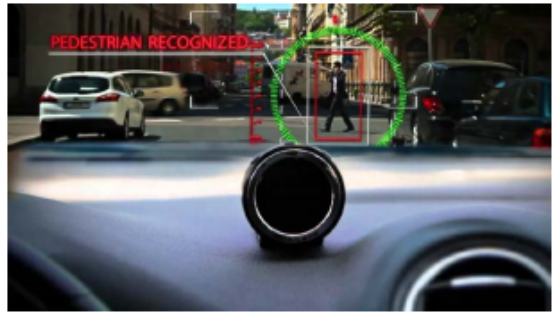
Self driving cars

Tracking cars and pedestrians.

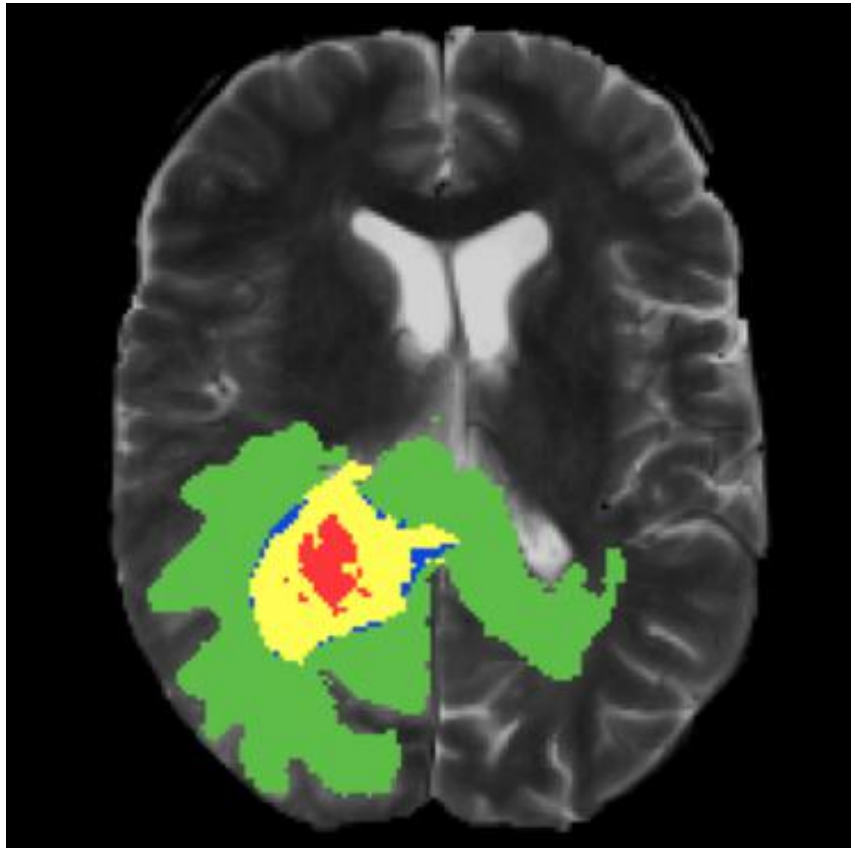








Medical imaging



Video surveillance

Activity recognition and pose estimation

Suggested reading: Keypoint Detection with Transfer Learning

State of the art technologies

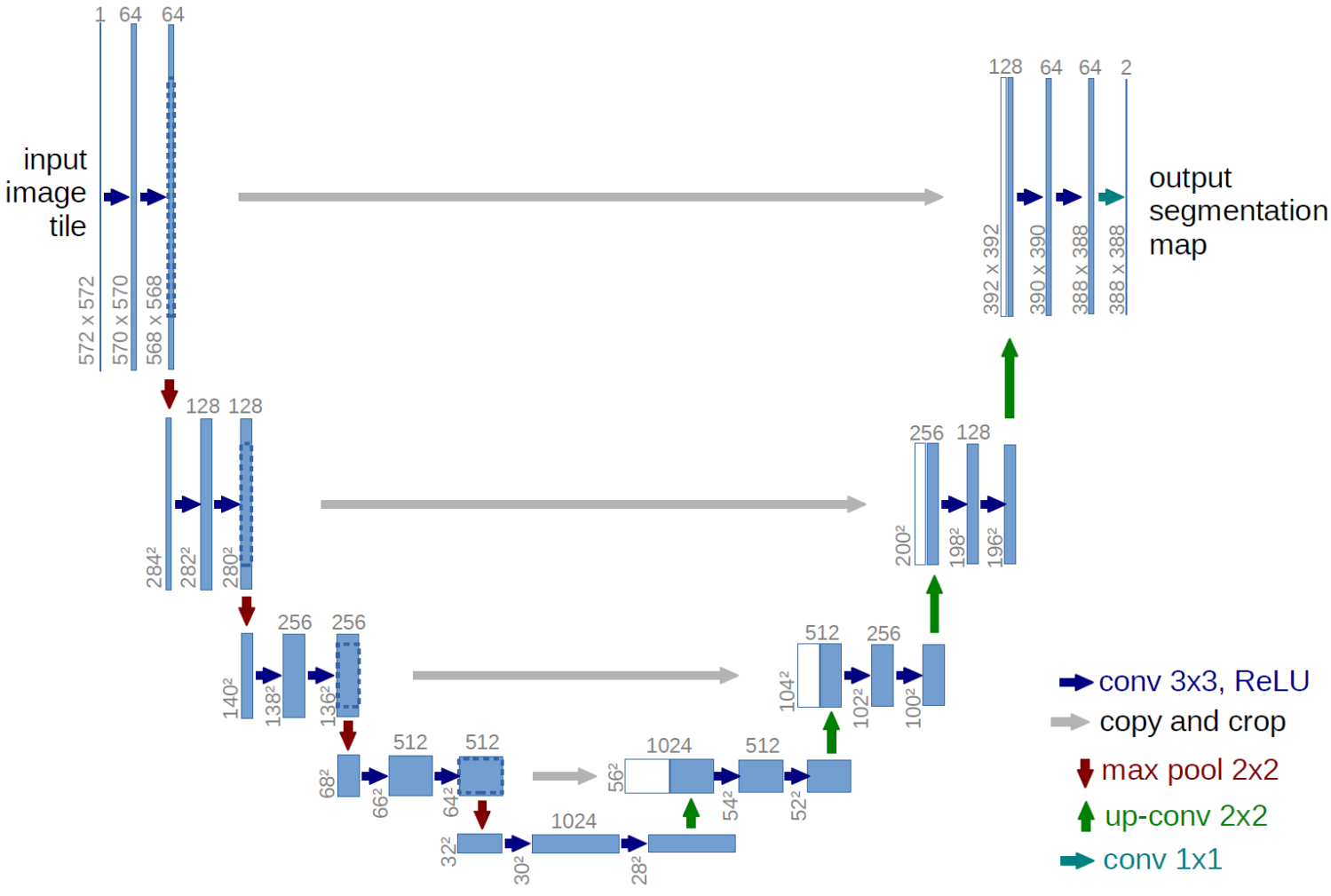
- U-net
- Detectron 2
- Yolo

U-net

Suggested reading:

U-Net: Convolutional Networks for Biomedical Image Segmentation. By O.Ronneberger, P.Fischer, T.Brox

U-net

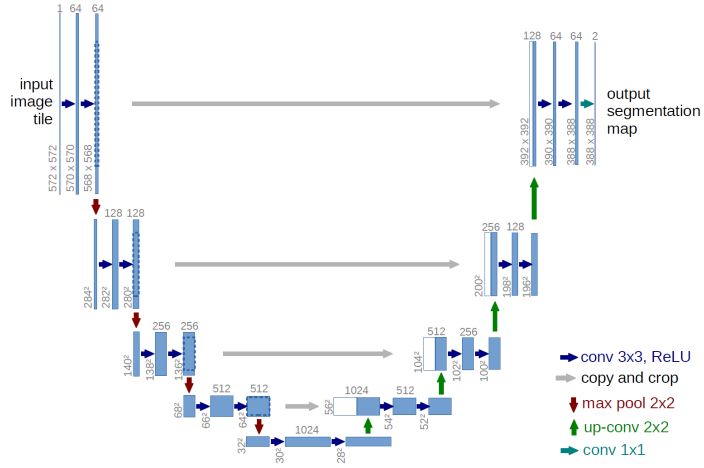

input image tile
572 x 572
570 x 570
568 x 568
1 64 64
128 128
284²
282²
280²
256 256
140²
138²
136²
512 512
68²
66²
64²
1024
32²
30²
28²
1024
512
56²
54²
52²
512 256
104²
102²
100²
256 128
200²
198²
196²
128 64 64 2
392 x 392
390 x 390
388 x 388
388 x 388
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

U-net key properties

It does not rely on a classification network.

U-net learns segmentation in an end-to-end setting.

It can work with relatively few training images (data augmentation was largely exploited) and yields accurate precise segmentations.



Deep Object Detection

Two main approaches:

- **Region Proposals** methods (R-CNN, Fast R-CNN, Faster R-CNN). Region Proposals are usually extracted via Selective Search algorithms, aimed to identify possible locations of interest. These algorithms typically exploit the texture and structure of the image, and are object independent.

- **Single shots** methods (Yolo, SSD, Retina-net, FPN). We shall mostly focus on these **really fast** techniques.

Detectron 2

Detectron2 is a pytorch library developed by Facebook AI Research (FAIR) to support rapid implementation and evaluation of novel computer vision research.

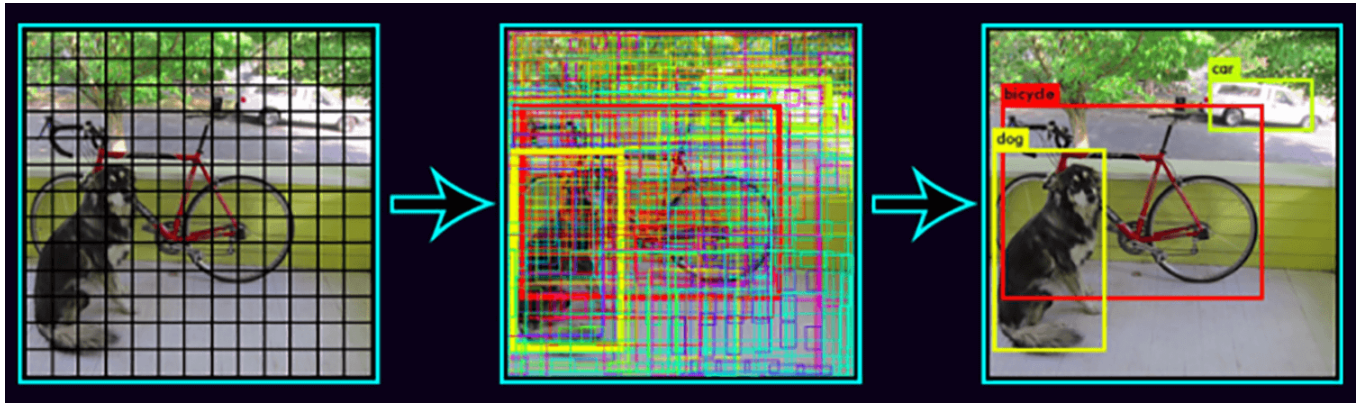
It includes implementations of the following object detection algorithms:

- Mask R-CNN
- RetinaNet
- Faster R-CNN
- RPN
- Fast R-CNN
- TensorMask
- PointRend
- DensePose

and more ...

YOLO

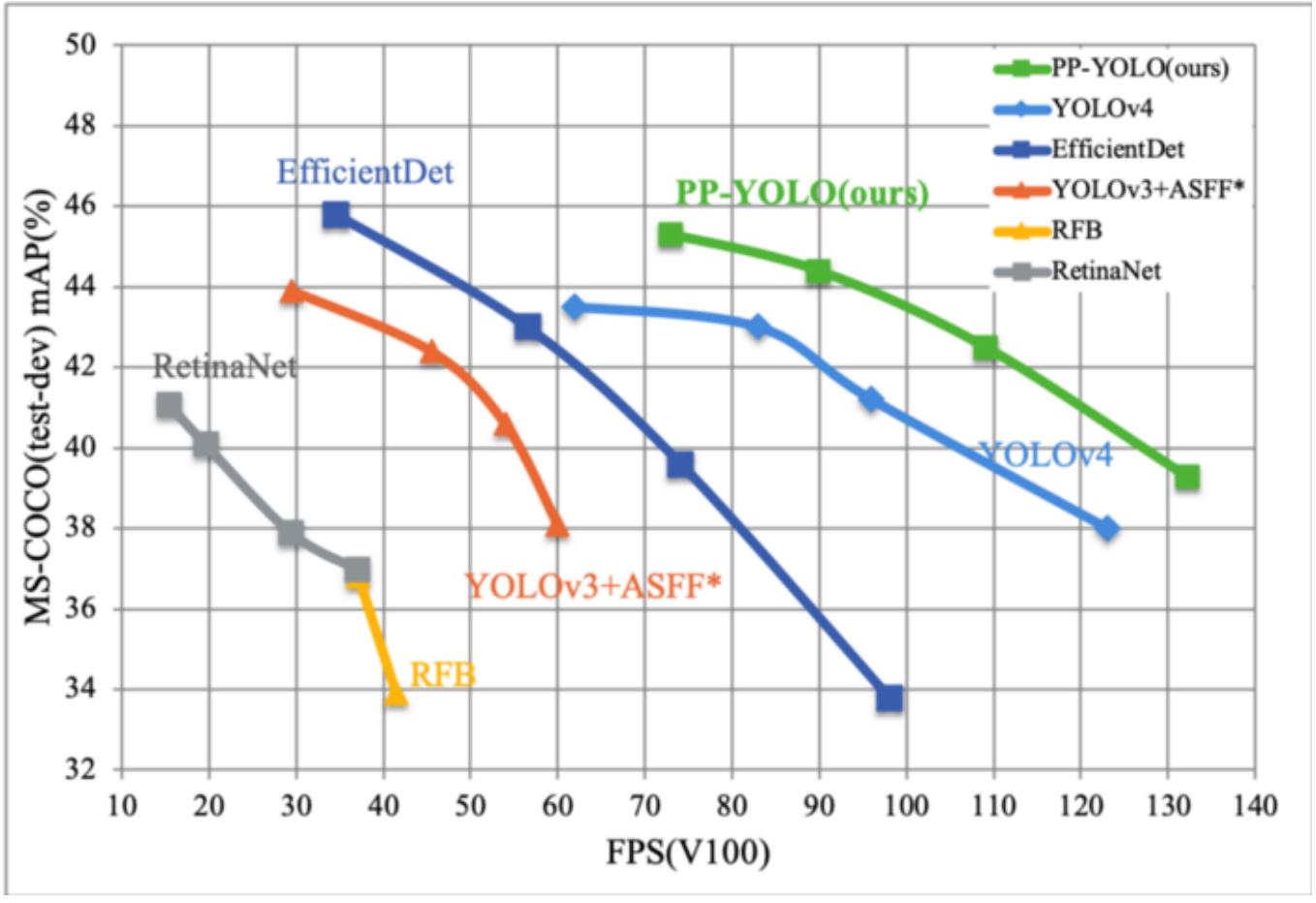
YOLO: Real-Time Object Detection



First release in 2016. Now at version 5.

Suggested reading: YOLO v4 or YOLO v5 or PP-YOLO?

Speed and accuracy



Source PP YOLO repo

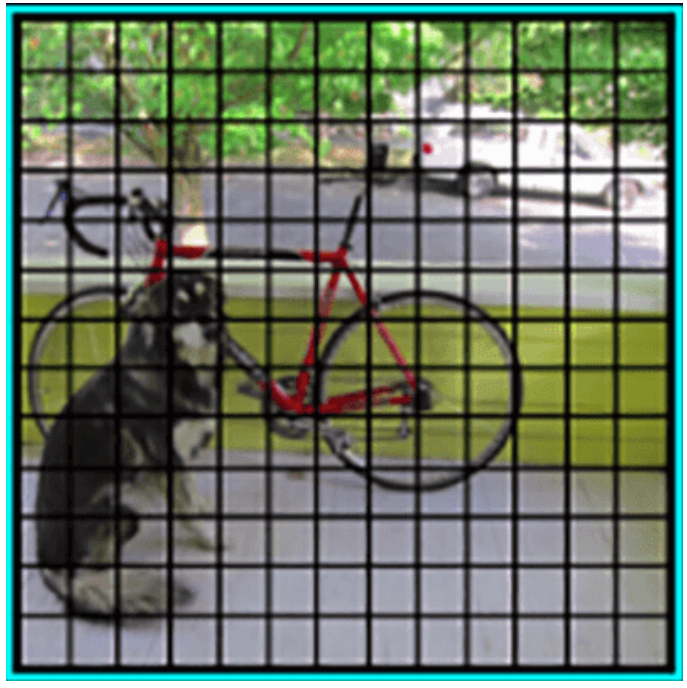
YOLO's architecture

Suggested reading: Yolo-tutorial in pytorch

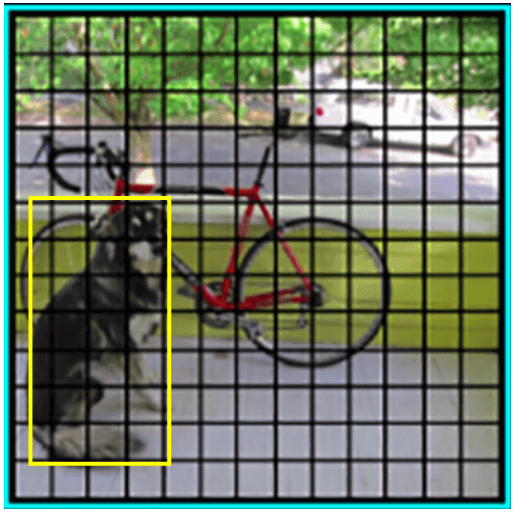
Yolo network

Yolo is a Fully Convolutional Networks. The input is progressively downsampled by a factor $2^5 = 32$.

For instance, an input image of dimension 416x416 would be reduced to a grid of neurons of dimension 13x13: the **feature map**



Which neuron is in charge of detection?



Detection of an object may concern all neurons inside the bounding box.

So, **who's in charge for detection?**

The answer crucially influences the way the network should be trained.

In YOLO, a single neuron is responsible for detection: the one whose grid-cell contains the center of the bounding box.

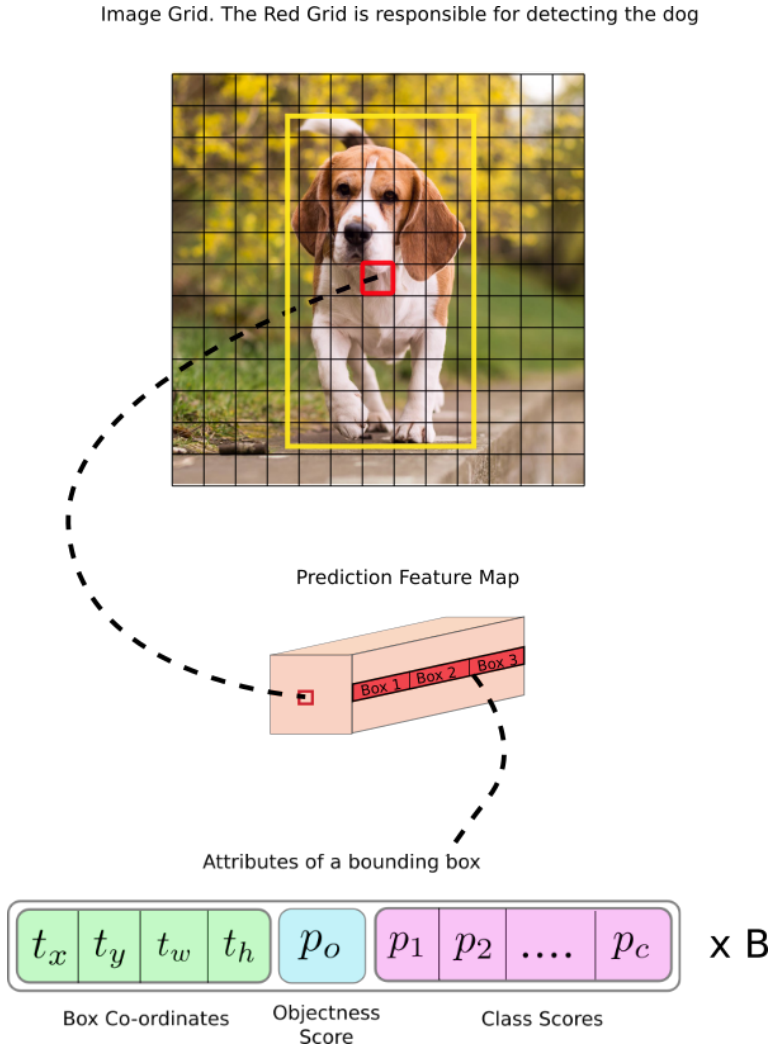
This neuron makes a finite number of **predictions** (e.g. 3).

Predictions

We have 13x13 neurons in the feature map.

Depth-wise, we have (B x (5 + C)) entries, where B represents the number of bounding boxes each cell can predict (say, 3), and C is the number of different object categories.

Each bounding box has 5 + C attributes, which describe the center co-ordinates (2), the dimensions (2), the objectness score (1) and C class confidences.



Anchor Boxes

Trying to directly predict width and the height of the bounding box leads to **unstable gradients** during training.

Most of the modern object detectors predict log-space affine transforms for **pre-defined** default bounding boxes called **anchors**.

Then, these transforms are applied to the anchor boxes to obtain the prediction. YOLO v3 has three anchors, which result in prediction of three bounding boxes per cell.

The bounding box responsible for detecting the object is one whose anchor has the highest IoU with the ground truth box.

Objectness Score

Objectness score represents the probability that an object is contained inside a bounding box.

The objectness score is also passed through a sigmoid, as it is to be interpreted as a probability.

Class Confidences

Class confidences represent the probabilities of the detected object belonging to a particular class. Before v3, YOLO class scores were computed via a softmax.

Since YOLOv3, multiple sigmoid functions are used instead, considering that objects may belong to multiple (hierachical) categories, and hence labels are not guaranteed to be mutually exclusive.

YOLO's loss function

YOLO's loss function

The loss consists of two parts, the **localization loss** for bounding box offset prediction and the **classification loss** for conditional class probabilities.

As usual, we shall use v and $\hat{v}$ to denote a *true* value, and the corresponding *predicted* one.

The lolization loss is

$$\mathcal{L}_{loc} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

where i ranges over cells, and j over bounding boxes.

1_{ij}^{obj} is a delta function indicating whether the j-th bounding box of the cell i is responsible for the object prediction.

YOLO's loss function (2)

The classification loss is the sum of two components, relative to the objectness confidence and the actual classification:

$$\mathcal{L}_{cls} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} (1_{ij}^{obj} + \lambda_{noobj}(1 - 1_{ij}^{obj}))(C_{ij} - \hat{C}_{ij})^2$$

$$+ \sum_{i=0}^{S^2} \sum_{c \in C} 1_i^{obj} (p_i(c) - \hat{p}_i(c))^2$$

λ_{noobj} is a configurable parameter meant to down-weigth the loss contributed by "background" cells containing no objects.
This is important because they are a large majority.

YOLO's loss function (3)

The whole loss is:

$$\mathcal{L} = \lambda_{coord}\mathcal{L}_{loc} + \mathcal{L}_{cls}$$

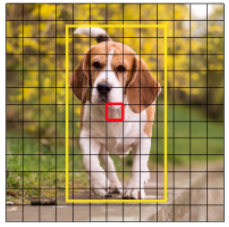
λ_{coord} is an additional parameter, balancing the contribution between $\mathcal{L}_{loc}$ and $\mathcal{L}_{cls}$.

In YOLO, $\lambda_{coord} = 5$ and $\lambda_{noobj} = 0.5$.

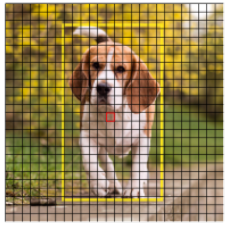
Non Maximum Suppression

Non Maximum Suppression

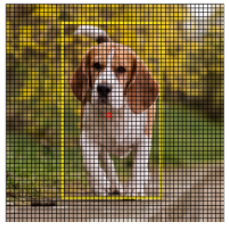
Prediction Feature Maps at different Scales



13 x 13



26 x 26



52 x 52

YOLOv3 predicts feature maps at scales 13, 26 and 52.

At the end, we have

$$((13\times13)+(26\times26)+(52\times52))x3 = 10647$$

bounding boxes, each one of dimension 85 (4 coordinates, 1 confidence, 80 class probabilities).

How can we reduce this number to the few bounding boxes we expect?

Intersection over Union

Typically, the quality of each individual bounding box is evaluated vs. the corresponding ground truth using **Intersection over Union**

$$IoU(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

IoU is also used to evalute the classifier, summing the accuracy of all detections, suitably combined with classification errors, the so called **Mean Average Precision** (MAP).

Non Maximum Suppression

The selection of the "best" bounding boxes is done algorithmically, and it usually composed of two phases:

- **Thresholding by Object Confidence:** first, we filter boxes based on their objectness score. Generally, boxes having scores below a threshold are ignored.
- **Non Maximum Suppression:** NMS addresses the problem of multiple detections of the same image, corresponding to different anchors, adjacent cells in maps.

NMS outline

Divide the bounding boxes BB according to the predicted class c.
Each list BB_c is processed separately

Order BB_c according to the object confidence.

Initialize TruePredictions to an empty list.

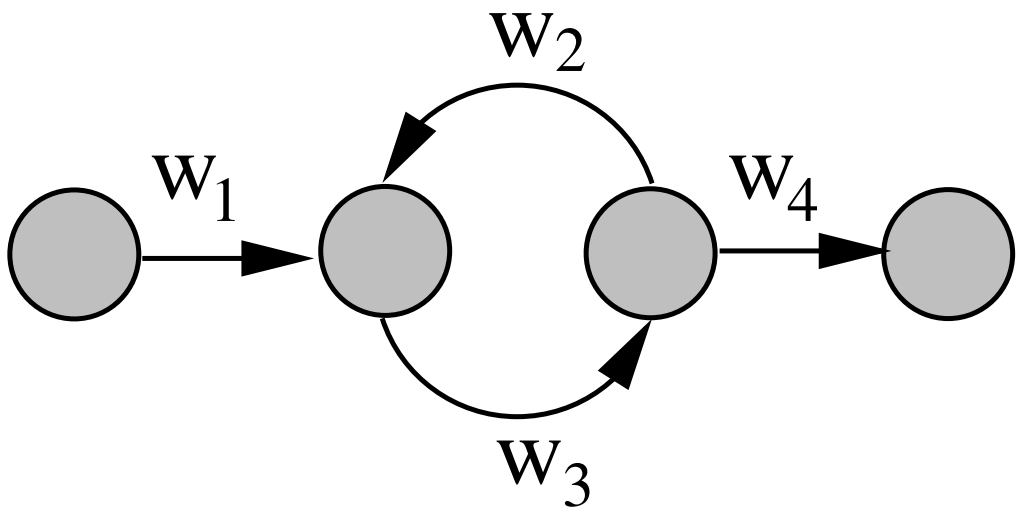
while BB_c is not empy:
    pop the first element p from BB_c
    add p to TruePredictions
    remove from BB_c all elements with an IoU with $p > th$

return TruePredictions

Recurrent Neural Networks

Recurrent Neural Networks

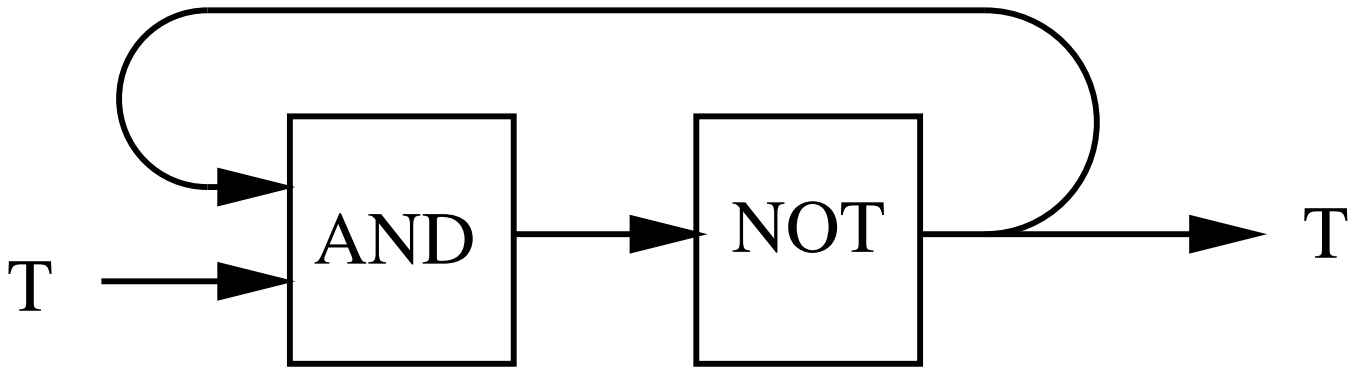
A recurrent network is simply a network with cycles.



In presence of backward connections, hidden states depend on the past history of the net.

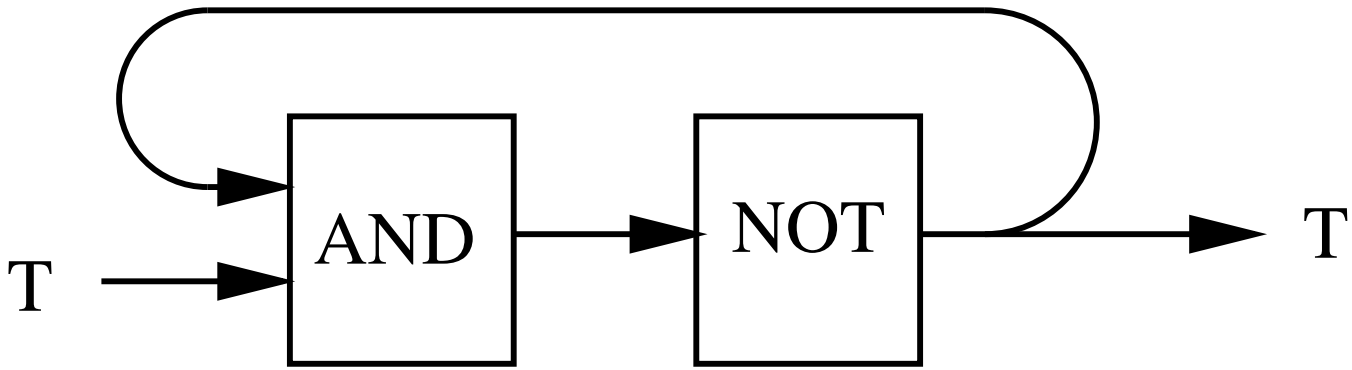
Instability

Cycles in logical circuits are a potential source of instability:



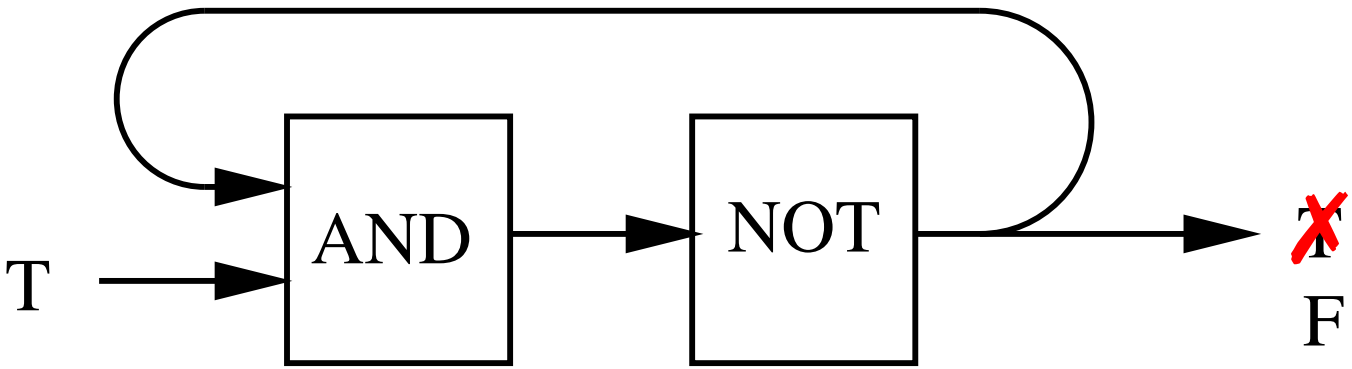
Instability

Cycles in logical circuits are a potential source of instability:



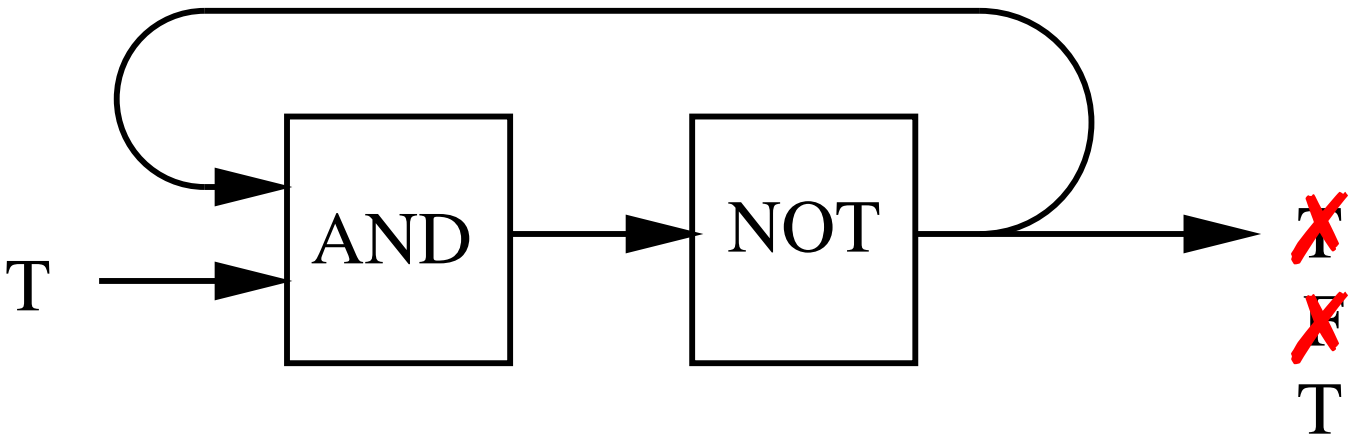
Instability

Cycles in logical circuits are a potential source of instability:

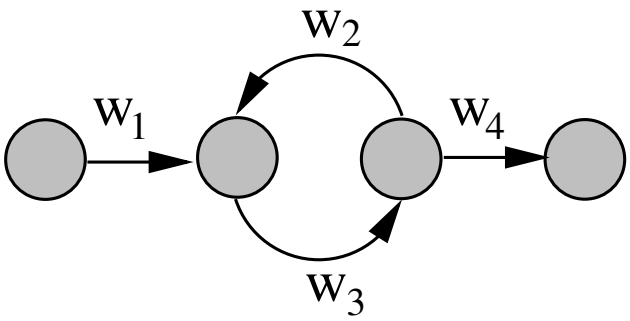


Instability

Cycles in logical circuits are a potential source of instability:

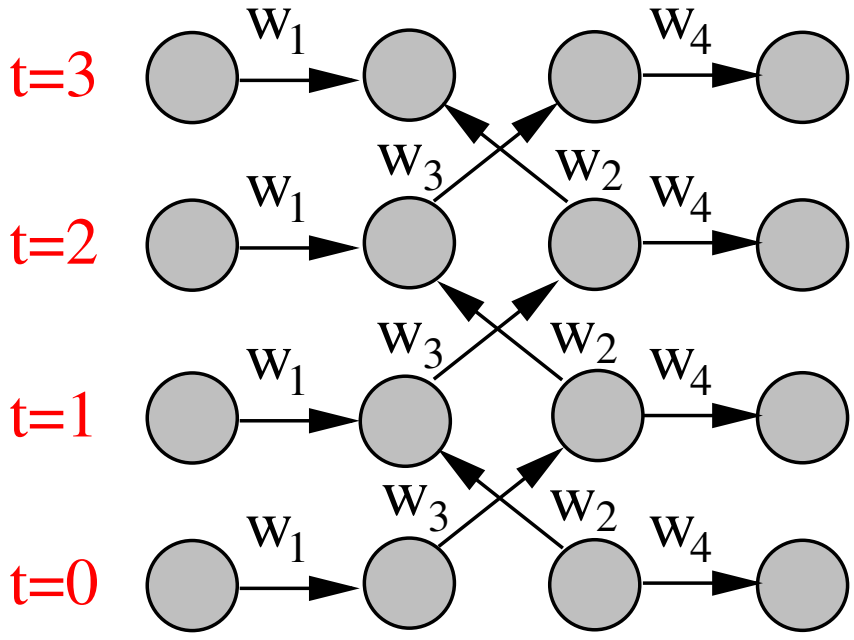


Temporal unfolding



Activations are updated at precise times steps

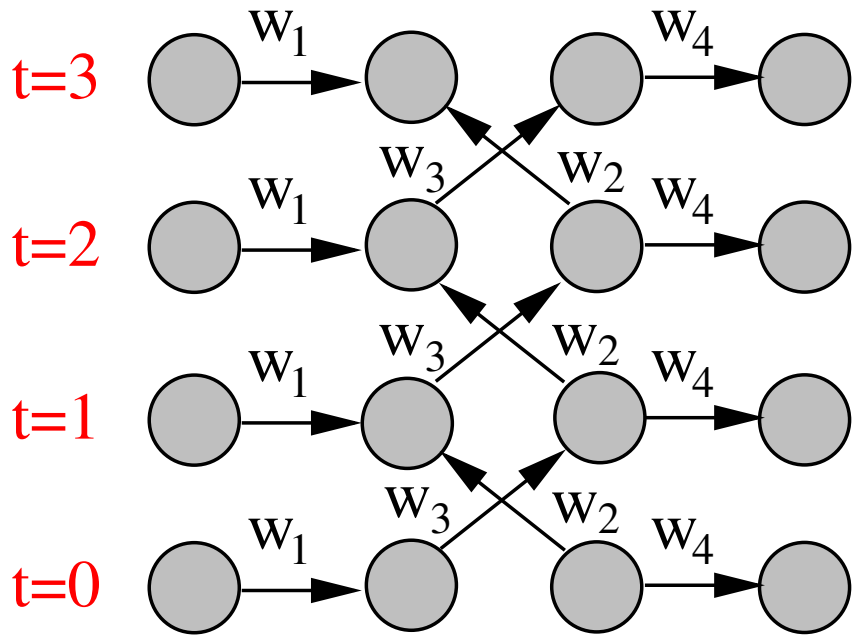
The recurrent net is just a layered net that keeps reusing the same weights



Input/output sequences

Due to the temporal unfolding, you expect an input and produce an output at **each timestep**

This is why recurrent networks are naturally suited to **process sequences**.



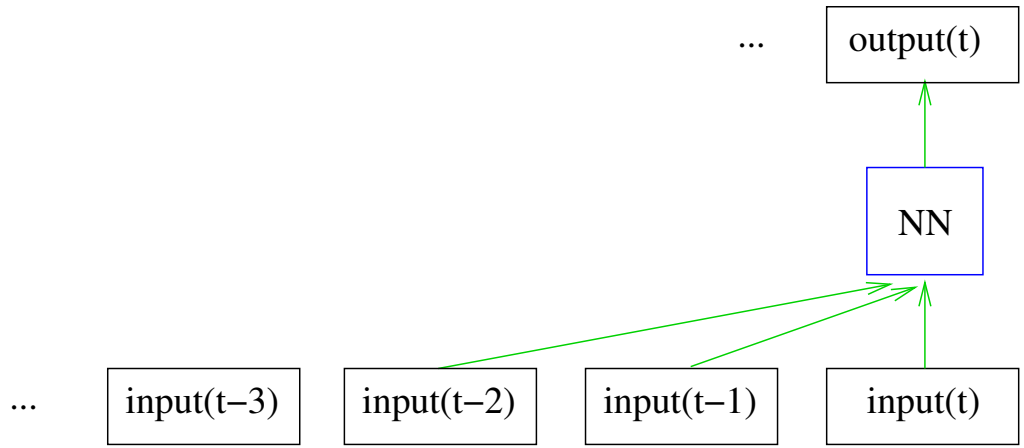
Modelling sequences

Typical problems:

- **turn an input sequence into an output sequence** (possibly in a different domain):
  - - translation between different languages
  - - speech/sound recognition
  - - ...
- **predict the next term in a sequence**
  The target output sequence is the input sequence with an advance of 1 step. Blurs the distinction between supervised and unsupervised learning.
- **predict a result from a temporal sequence of states**
  Typical of Reinforcement learning, and robotics.

Memoryless approach

Compute the output as a result of a fixed number of elements in the input sequence



Used e.g. in

- - Bengio's (first) predictive natural language model
- - Qlearning for Atari Games

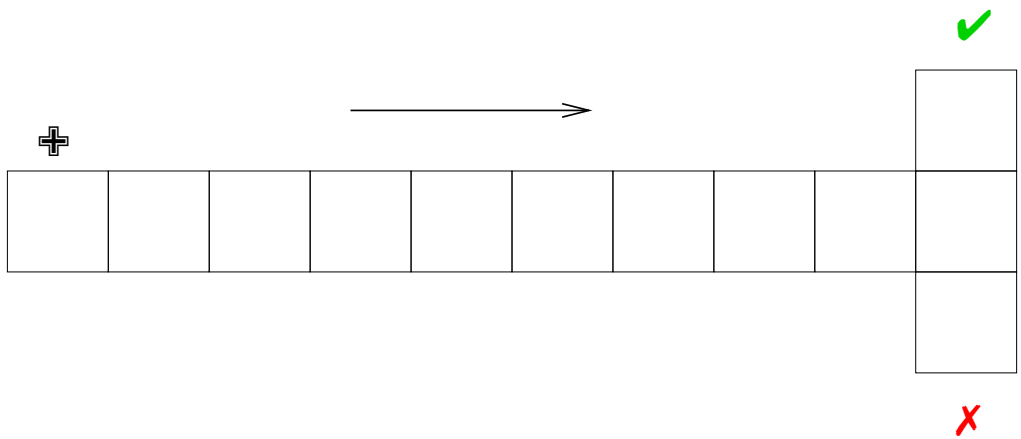
Difficult to deal with very long-term dependencies.

Simple problems requiring memory

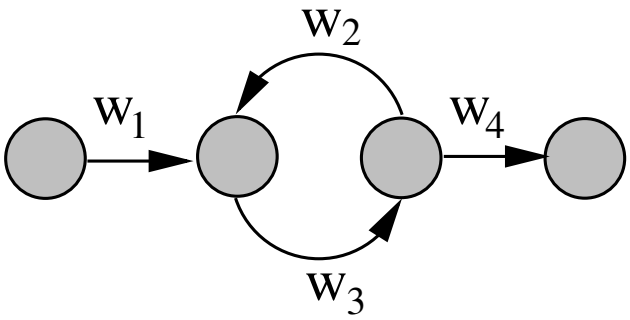
arithmetical sum

$$
\begin{array}{cccccccc}
 & & & \leftarrow & & & & \\
\ldots & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
\ldots & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\
\hline
\ldots & 0 & 1 & 0 & 1 & 0 & 0 & 1
\end{array}
$$

the T-maze

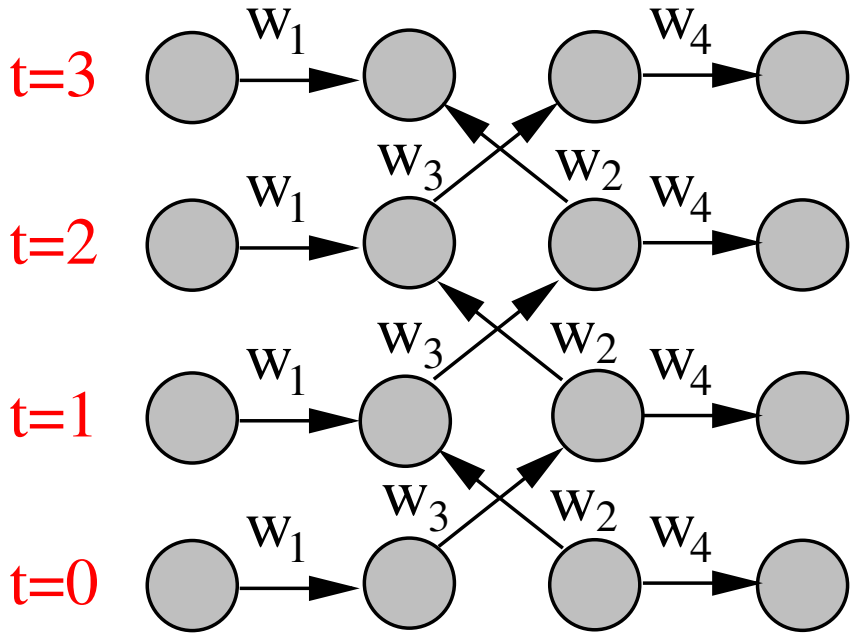


Back to Recurrent Networks



Activations are updated at precise times steps

The recurrent net is just a layered net that keeps reusing the same weights



Sharing weights through time

It is easy to modify the backprop algorithm to incorporate equality constraints between weights.

We compute the gradients as usual, and then average gradients so that they induce a same update.

If the initial weights started satisfied the constraints, they will continue to do.

To constrain $w_1 = w_2$
we need $\Delta w_1 = \Delta w_2$

compute $\frac{\partial E}{\partial w_1}$ and $\frac{\partial E}{\partial w_2}$

and use $\frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2}$

to update both w_1 and w_2

Hidden state initialization

We need to specify the initial activity state of all the hidden and output units.

The best approach is to treat them as parameters, learning them in the same way as we learn the weights:

- start off with an initial random guess for the initial states
- at the end of each training sequence, backpropagate through time all the way to the initial states to get the gradient of the error function with respect to each initial state
- adjust the initial states by following the negative gradient

Long-Short Term Memory (LSTM)

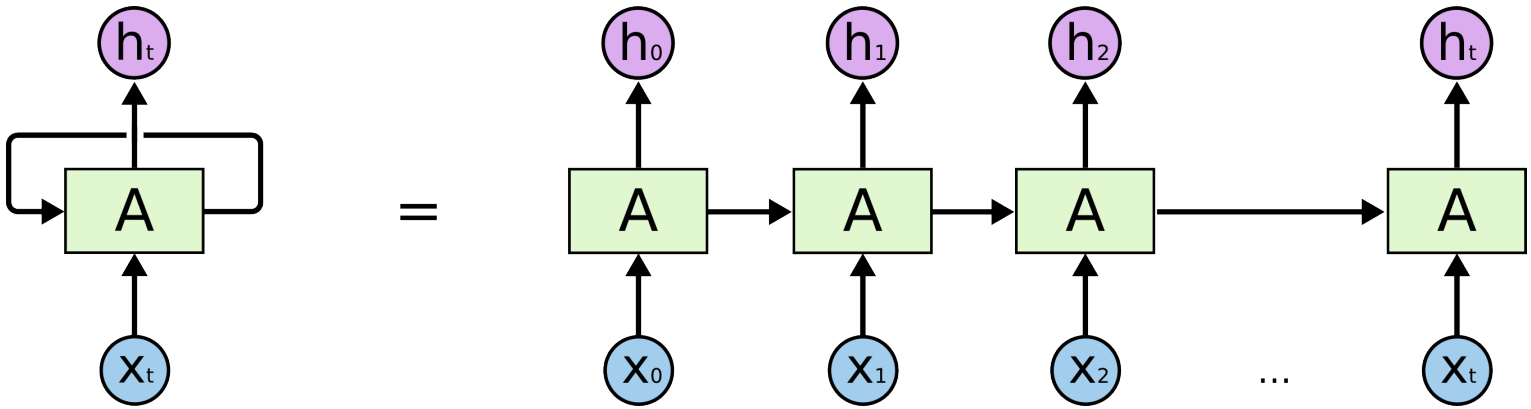
Largely based on Colah's blog

The goal

Find a **basic** component (NN-layer):

- simple
- flexible
- effective
- modular

Unrolling recurrent nets

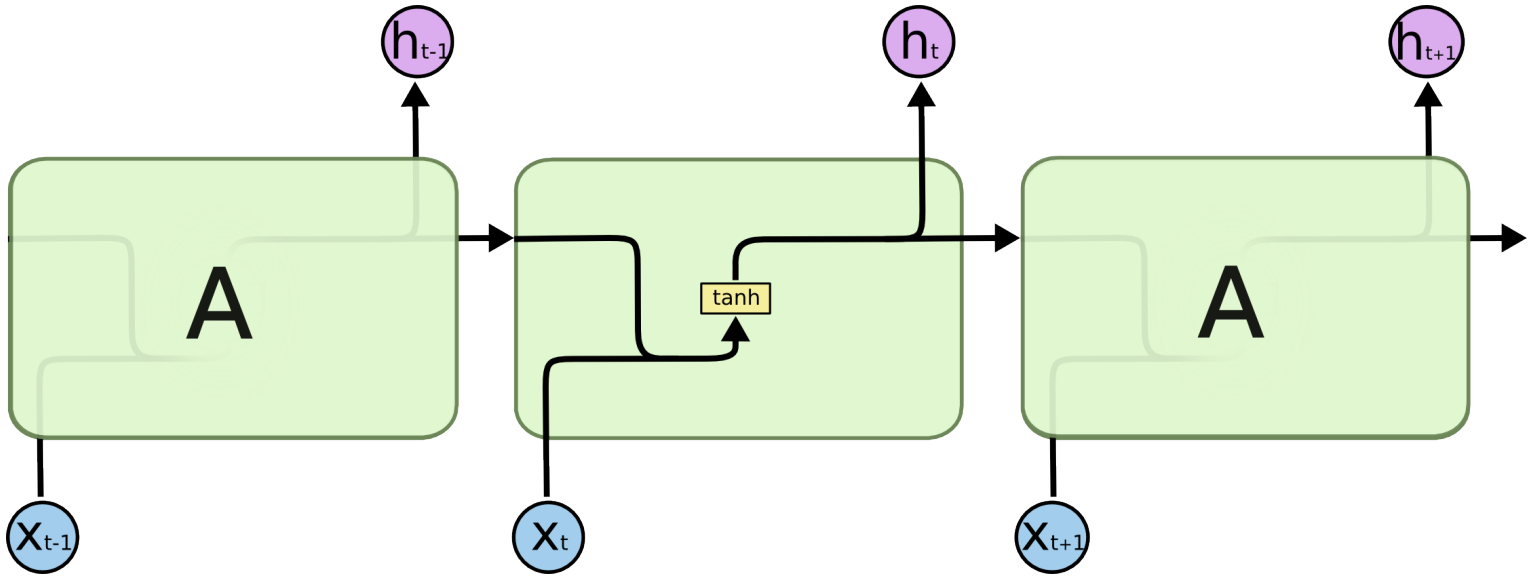


In the following, we shall mostly depict RNN in unrolled form.

A forward link between two units muts be understood as a looping connection.

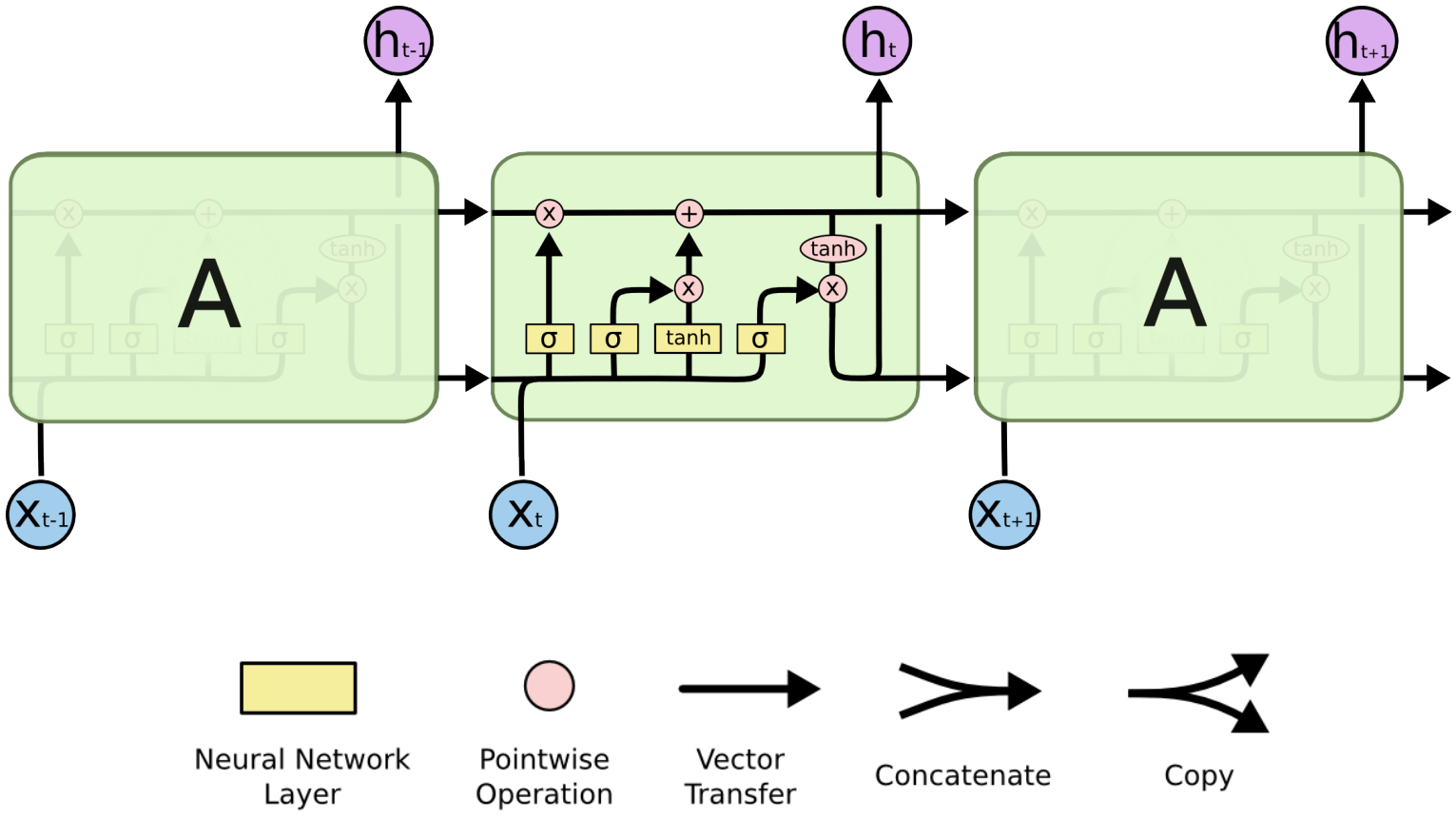
A simple, basic RNN

The content of the memory cell C_t, and the input x_t are combined through a simple neural net to produce the output h_t that coincides with the new content of the cell C_{t+1}.



Why $C_{t+1} = h_t$? Better trying to preserve the memory cell, letting the neural net learn how and when to update it.

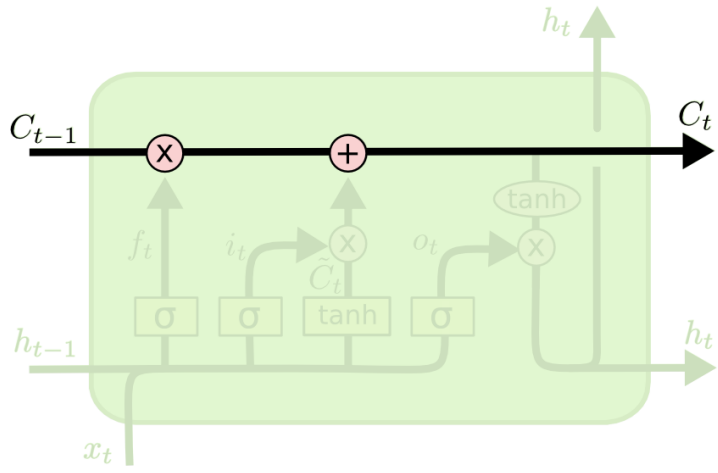
The overall structure of a LSTM



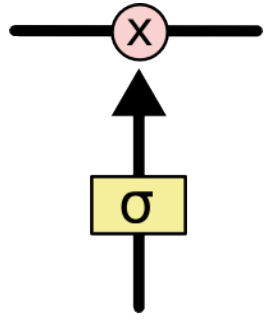
C-line and gates

The LSTM has the ability to remove or add information to the cell state, in a way regulated by suitable gates.

Gates are a way to optionally let information through: the product with a sigmoid neural net layer simulates a boolean mask.

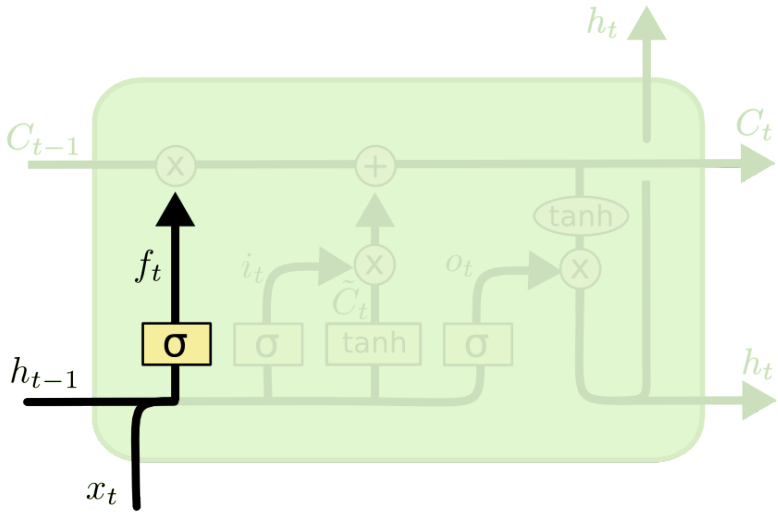


the C-line



a gate

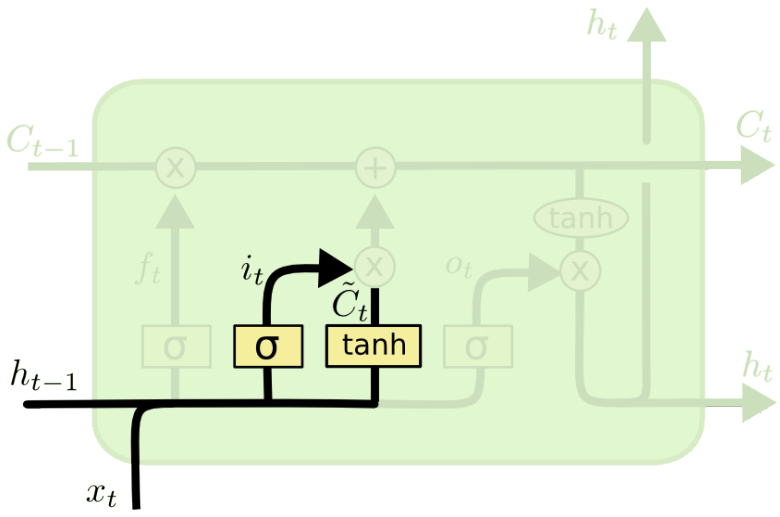
The forget gate



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

The forget gate decides what part of the memory cell to preserve

The update gate



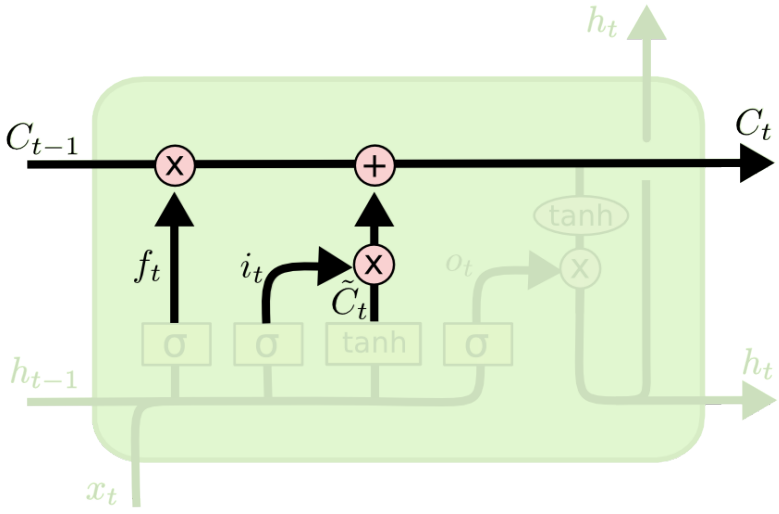
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

The input gate decides what part of the input to preserve.

The tanh layer creates a vector of new candidate values $\tilde{C}_t$ to be added to the state.

Cell updating



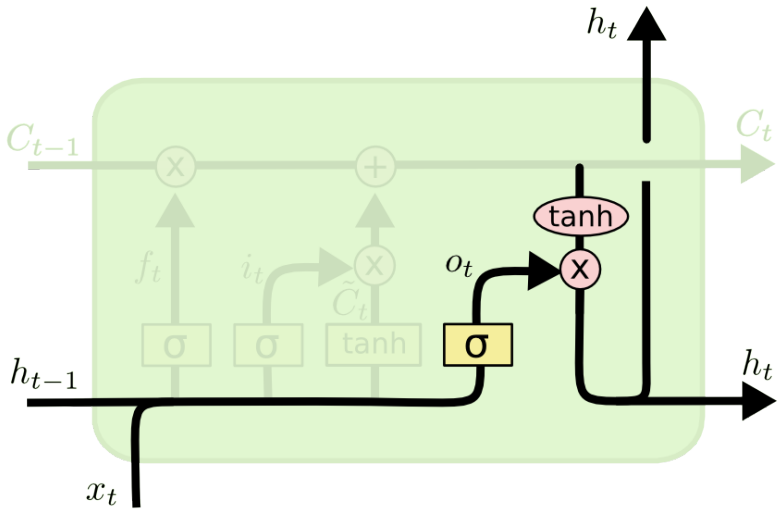
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

We multiply the old state by the boolean mask f_t.

Then we add $i_t * \tilde{C}_t$.

output gate

The output h_t is a filtered version of the content of the cell.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

The output gate decides what parts of the cell state to output.

The tanh function is used to renormalize values in the interval $[-1, 1]$.

Many variants

Essential bibliography

- S.Hochreiter, J. Schmidhuber. "Long short-term memory". Neural Computation. 9 (8): pp.1735-1780. 1997
- F.A.Gers, Jürgen Schmidhuber, F.Cummins. "Learning to Forget: Continual Prediction with LSTM". Neural Computation. 12 (10), pp.2451-2471. 2000.
- F.A.Gers, E.Schmidhuber. "LSTM recurrent networks learn simple context-free and context-sensitive languages". IEEE Transactions on Neural Networks. 12 (6): pp. 1333-1340. 2001.
- Y.Chung, C.Gulcehre, K.Cho, Y.Bengio. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". arXiv:1412.3555. 2014

The lstm layer in Keras

From a practical point of view, the LSTM layer is very similar to a traditional layer.

When you **define** the layer, you specify the number of **units**, that is the dimension of the memory cell, equal to the dimension of the hidden state and the output.

When you **apply** the layer, you pass as **input** an array of dimension

$$[batch, timesteps, features]$$

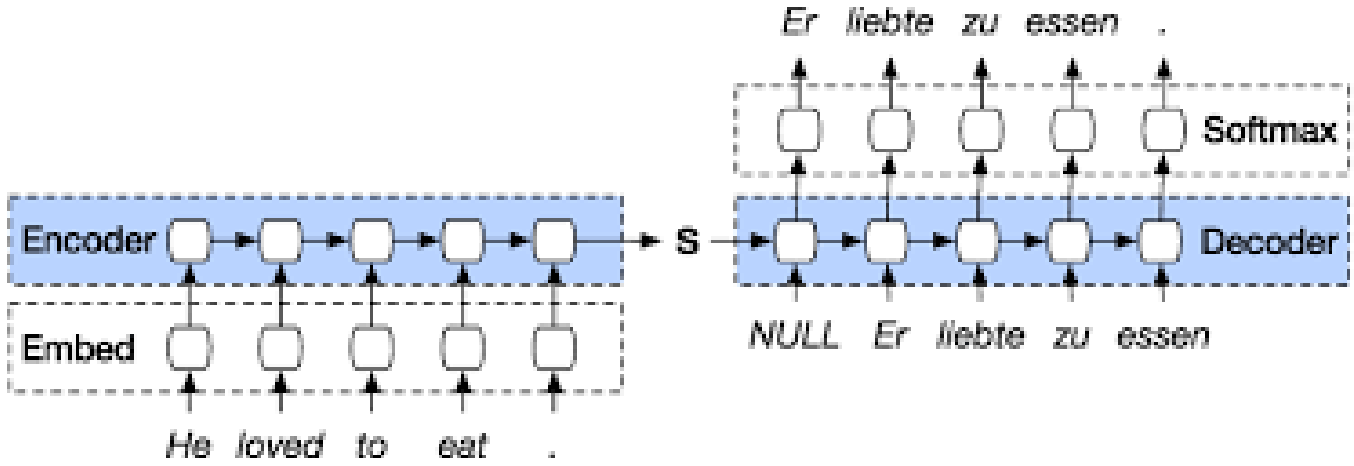
You get as output an array of dimension

$$[batch, units]$$

(unless you ask to return sequences)

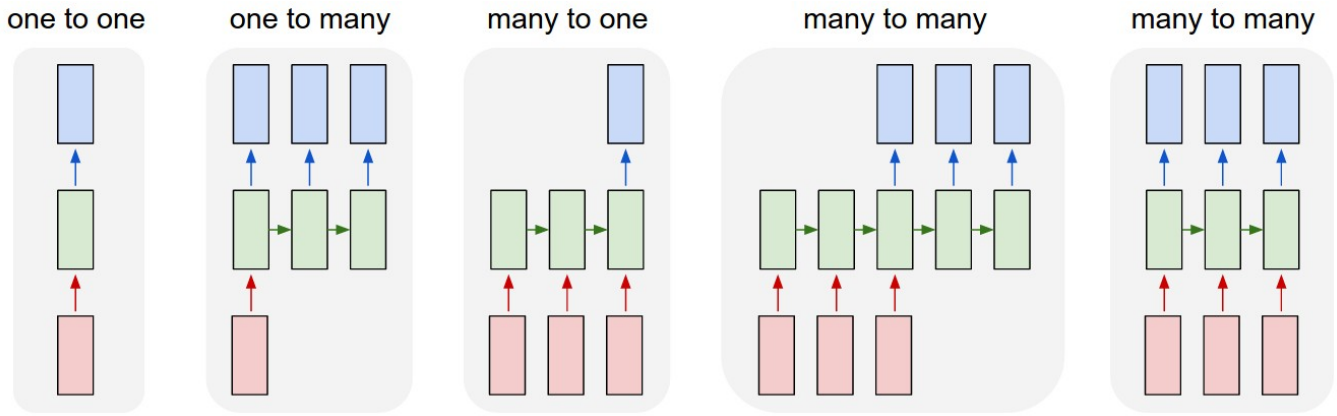
A simple application

A ten-minute introduction to sequence-to-sequence learning in Keras



An old but wondrous blog

The Unreasonable Effectiveness of Recurrent Neural Networks



- one to one: no recurrrence
- one to many: e.g. caption generation
- many to one: e.g. sentiment ananlysis
- many to many (async): e.g. language translation
- many to many (sync): per frame video processing

Lesson 10

- Attention
- Transformers

Attention

Attention

Attention is the ability to focus on different parts of the input, according to the requirements of the problem being solved.

It is an **essential** component of any intelligent behaviour, with potential application to a wide range of domains.

A differential mechanism

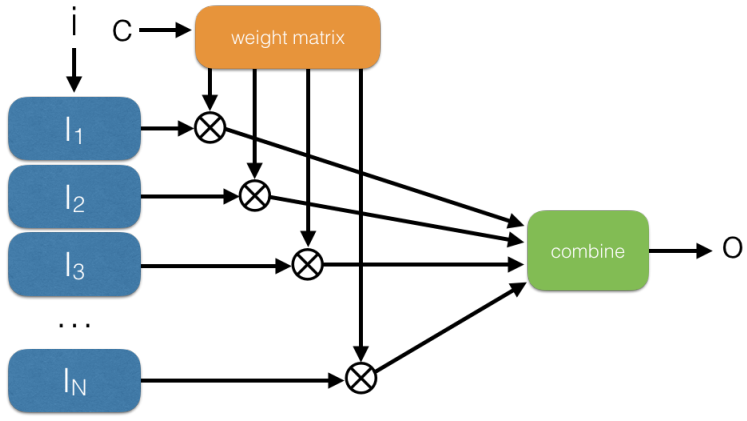
From the point of view of Neural Networks, we would expect the attention mechanism to be **differentiable**, so that we can learn where to focus by standard **backpropagation techniques**.

The current approach (not necessarily the best one) is to **focus everywhere**, just to **different extents**.

Attention as gating maps

Attention mechanisms can be implemented as gating functions.

The gating maps are dynamically generated by some neural net, allowing to focus on different part on the input at (e.g.) different times.



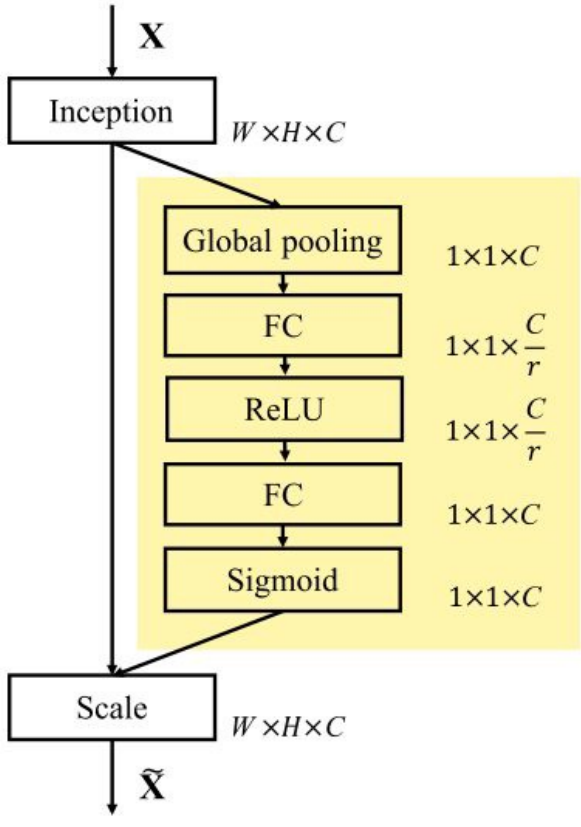
Picture from The fall of RNN / LSTM by E.Colurciello

The forget map, input map and output map in LSTMs are examples of attention mechanisms.

Another example: squeeze and excitation

SE layers are a building component of Squeeze and Excitation Networks

SE layers implement a form of self attention, allowing to focus on particular channels in a dinamical way, according to the input under consideration.



SE-Inception Module

A modular multi purpose layer

The most typical attention layer is based on the **key-value** paradigm, implementing a sort of associative memory.

We access this memory with **queries** to be matched with keys.

The resulting **scores** generate a boolean map that is used to weight values.

Attention: the key-value approach

For each key k_i compute the **scores** a_i as

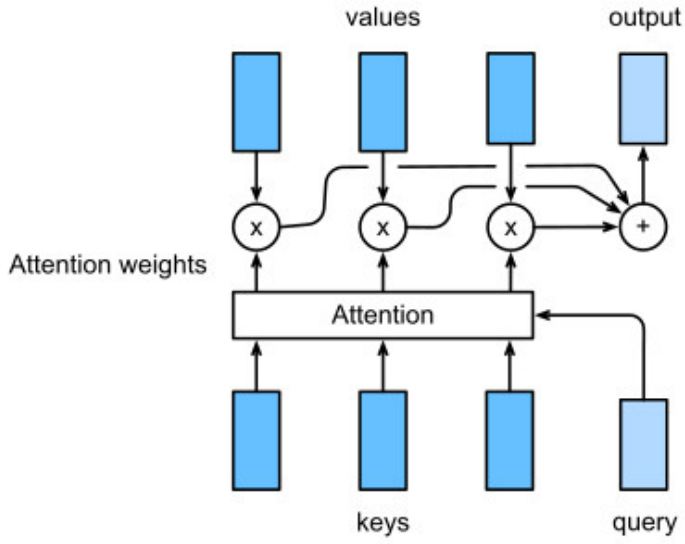
$$a_i = \alpha(q, k_i)$$

obtain **attention weights** via softmax:

$$\vec{b} = softmax(\vec{a})$$

retrun a weighted sum of the values:

$$o = \sum_{i=1}^{n} b_i v_i$$



In many applications, values are also used as keys (self-attention).

Typical score functions

Different score functions lead to different attention layers.

Two commonly used approaches are:

- Dot product.

$$\alpha(q, k) = q \cdot k / \sqrt{(d)}$$

  The query and the key must have the same dimension d

- MLP: α is computed by a neural network (usually composed by a single layer):

$$\alpha(k, q) = tanh(W_k \vec{k} + W_q \vec{q})$$

See Attention layer for the Keras implementation of this layer.

An application to translation

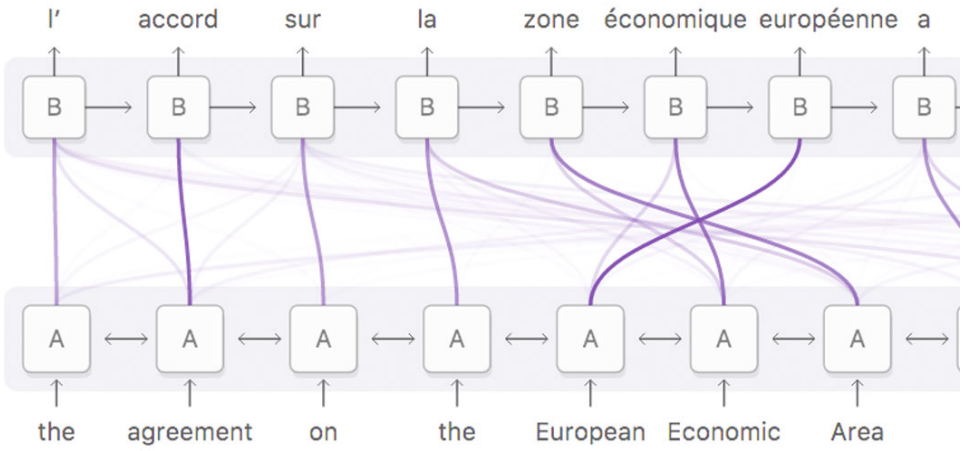
Neural Machine Translation by jointly learning to align and to translate, D.Bahdanau, K.Cho, Y.Bengio (2015)

alignment identify which parts of the input sequence are relevant to each word in the output

translation is the process of using the relevant information to select the appropriate output.

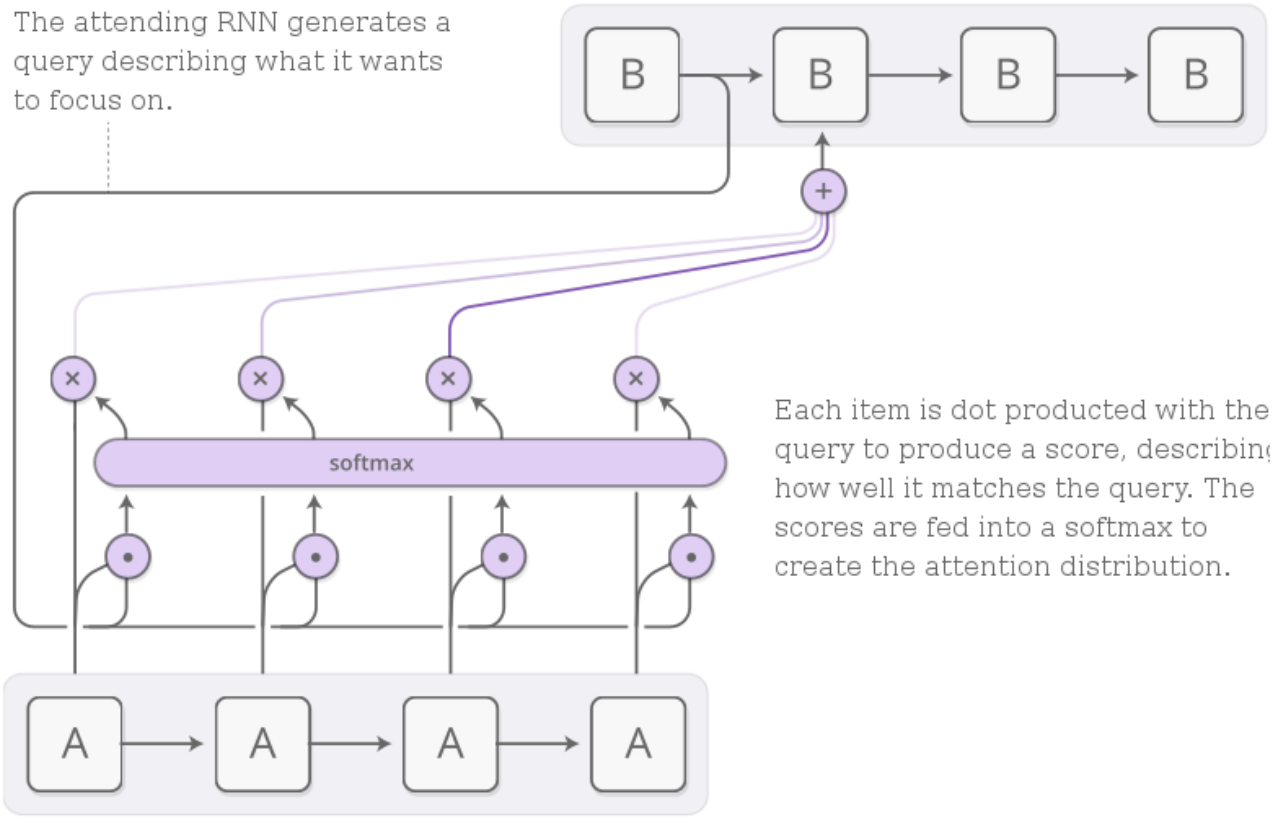
Attention to alignement

Alignement is a form of attention!



Picture from Attention and Augmented Recurrent Neural Networks, by C.Olah and S.Carter.

Producing attention maps



Picture from Attention and Augmented Recurrent Neural Networks.

Producing attention maps

> *"The decoder decides parts of the source sentence to pay attention to. By letting the decoder have an attention mechanism, we relieve the encoder from the burden of having to encode all information in the source sentence into a fixed- length vector. With this new approach the information can be spread throughout the sequence of annotations, which can be selectively retrieved by the decoder accordingly".*

Attention and Augmented Recurrent Neural Networks.

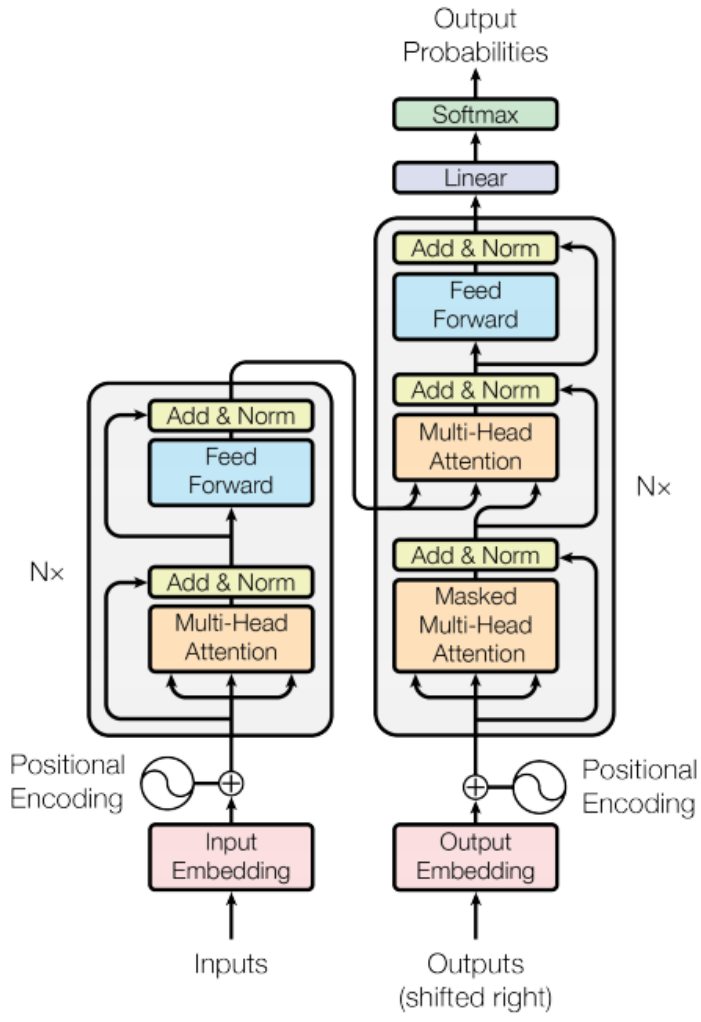
Transformers

Transformers

Transformers have been introduced in Attention is All You Need, one of the most influential works of recent years.

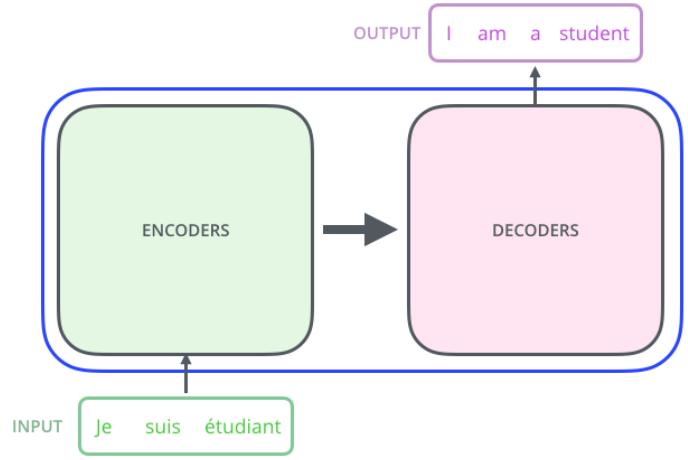
Transformers have rapidly become the model of choice for NLP.

Applications like Bert and GPT, (with all relative families) are based on Transformers.

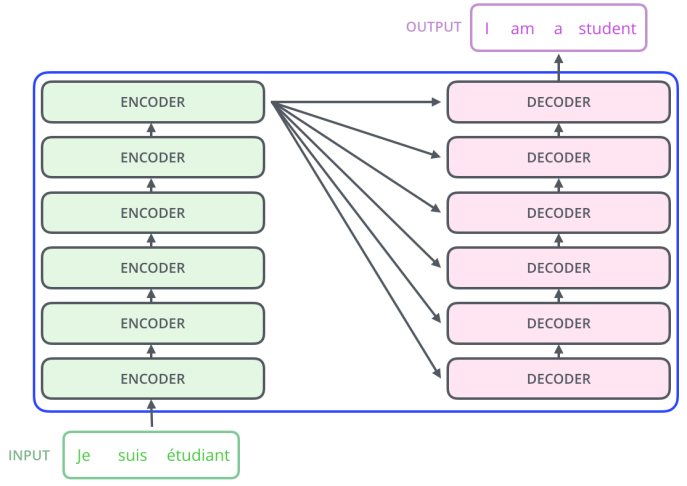


Encoder and decoder

A transformer has a traditional encoder-decoder structure, with connections between them.



The encoding component is a stack of **encoders**. Similarly, the decoding component is a stack of **decoders**.

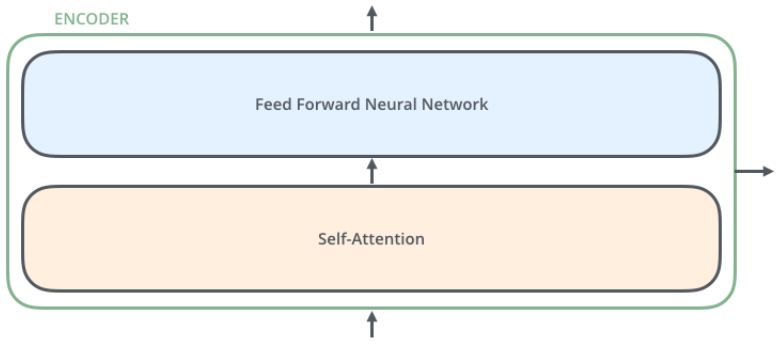


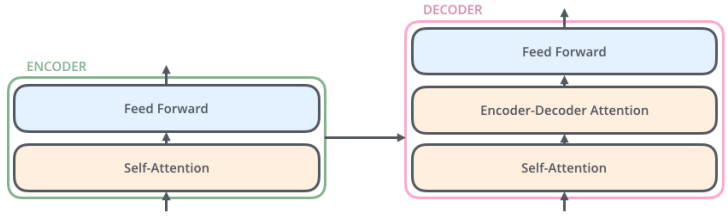
Pictures from The annotated transformer

Encoder and decoder modules

The encoder is organized as a self-attention layer (query, key and value are shared), followed by feedfoward component (a couple of dense layers).

The decoder is similar, with an additional attention layer that helps the decoder to focus on relevant parts of the input sentence.

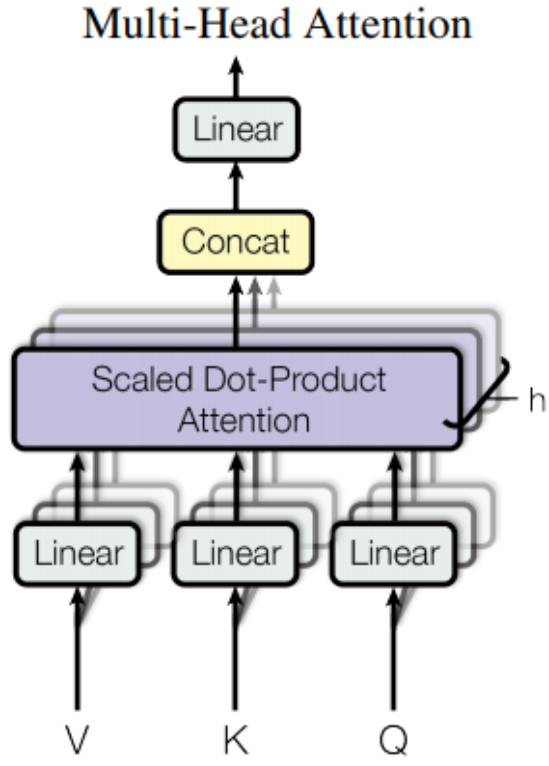




Multi head attention

Using multiple heads for attention expands the model's ability to focus on different positions, for different purposes.

As a result, multiple "representation subspaces" are created, focusing on potentially different aspects of the input sequence.



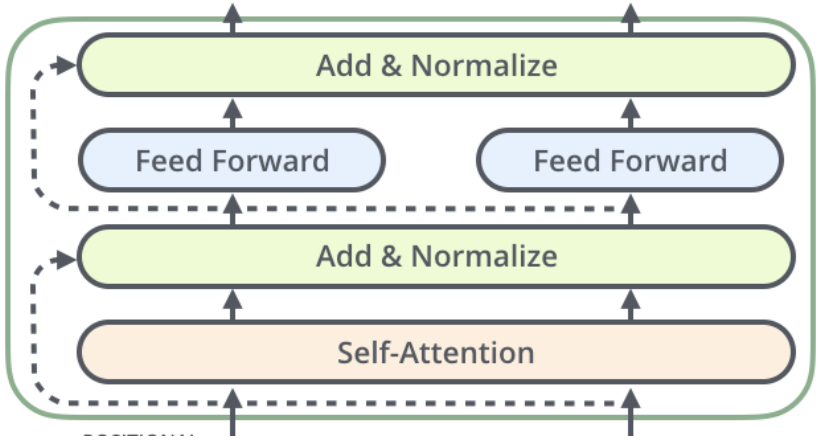
Masking the future

Typically, in a Transformer module, we can apply a boolean mask to the input, to hide part of its content.

This is frequently used in the decoder to prevent it to attend at future positions during generation.

Residual connections

Each sub-layer (self-attention, ffnn) in each encoder has a residual connection around it, and it is followed by a layer-normalization step.



Positional encoding

Positional encoding is added to word embeddings to give the model some information about the **relative position** of the words in the sentence.

The positional information is a vector of the same dimensions d_{model}, of the word embedding.

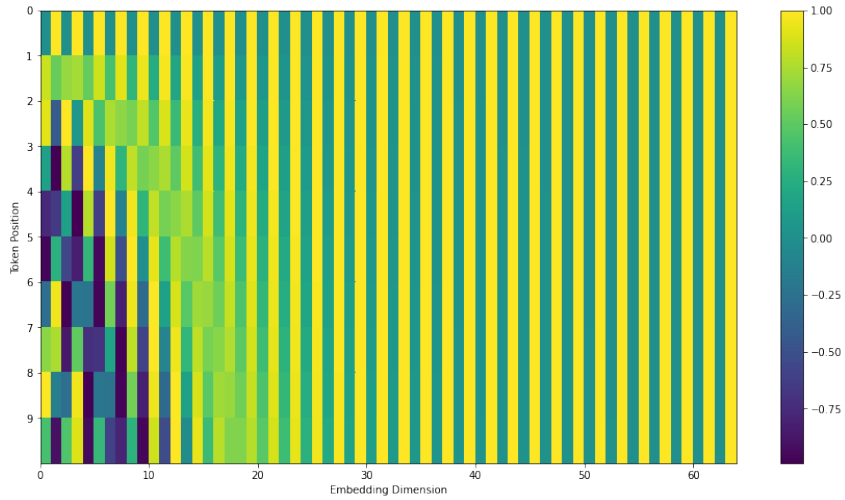
The authors use sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i+1/d_{model}})$$

Sinusoids at work

According to the previous encoding, each dimension i of the PE vector corresponds to a sinusoid, where the wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$.



"We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a **linear function** *of PE_{pos}."*

The annotated transformer

Additional links

- Attention is all you need
- The annotated transfomer
- Tensorflow transformer tutorial
- D2L lesson on transformers
- The illustrated Transfomer
- The positional encoding