- Overfitting
- Entropy, CrossEntropy, Kullabck-Leibler divergence

Overfitting

Overfitting and underfitting

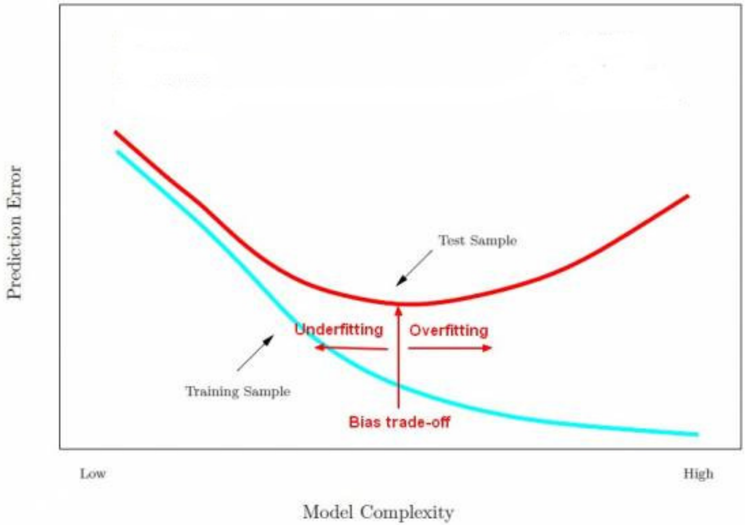
overfitting the model is too complex and specialized over the peculiarities of the samples in the training set

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

remark

Deep models are good at fitting, but the real goal is generalization

Overfitting and model complexity



About overfitting

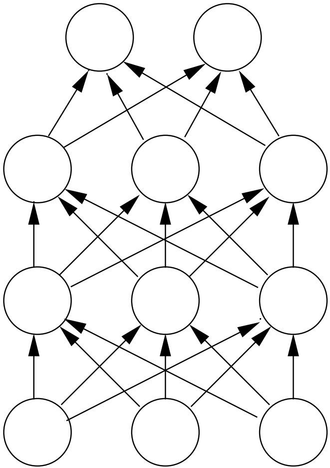
Ways to reduce overfitting

- Collect more data
- Reduce the model capacity
- Early stopping
- Regularization, e.g. Weight-decay
- Model averaging
- Data augmentation
- Dropout

Dropout

Idea: "cripple" the neural network stocastically removing hidden units

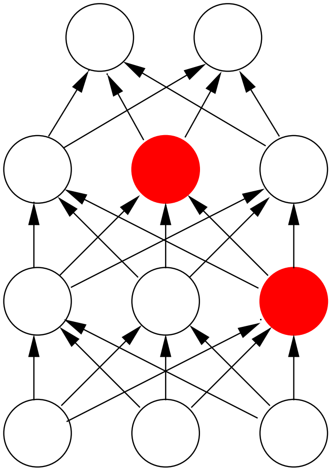
- during training, at each iteration hidden units are disabled with probability p (e.g. 0.5)
- hidden units cannot co-adapts with other units
- similar to train many networks and averaging between them



Dropout

Idea: "cripple" the neural network stocastically removing hidden units

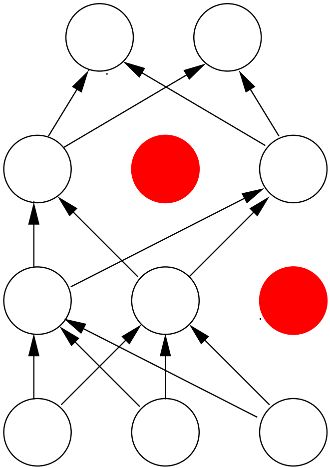
- during training, at each iteration hidden units are disabled with probability p (e.g. 0.5)
- hidden units cannot co-adapts with other units
- similar to train many networks and averaging between them



Dropout

Idea: "cripple" the neural network stocastically removing hidden units

- during training, at each iteration hidden units are disabled with probability 1-p (e.g. 0.5)
- hidden units cannot co-adapts with other units
- similar to train many networks and averaging between them



Geometric averaging

At each stage of training, only the crippled network is trained by means of backpropagation. Then, the omitted units are reinserted and the process repeated (hence weights are shared among the crippled networks).

At test time, we weight each unit with its expectaction p.

For a single layer, this is equivalent to take a geometric average among all different crippled networks.

A form of regularization

With Dropout, we are randomly sampling from an exponential number of different architectures
- all architectures share weights

Sharing weights means that every model is very strongly regularized, by all the other models
- A good alternative to L2 or L1 penalties that pull the weights towards zero.

Demonstrating Overfitting

Example 1: The IMDB Movie reviews data set

A Dataset of 25,000 movies reviews from IMDB, labeled by sentiment (positive/negative).

Each review is a sequence of words in a vocabulary of 10000 different words. Each word is encoded by an index (integer) in the range [0,9999].

Words are indexed by overall frequency in the dataset; for instance, the integer "3" encodes the 3rd most frequent word in the data.

This allows for quick filtering operations such as: “only consider the top 10,000 most common words, but eliminate the top 20 most common words”.

Bag of words approach

Encode each review r as a boolean vector x_r of dimension 10000 (number of different words).

We neglect the order and the multiplicity.

$x_r[i] = 1$ if the word with index i appears in the review r, and 0 otherwise.

DEMO

Example 2: CIFAR-10

DEMO
(data augmentation)

Suggested reading:

Do CIFAR-10 Classifiers Generalize to CIFAR-10?

Activation and loss functions for classification

Sigmoid

When the result of the network is a value between 0 and 1, e.g. a probability for a binary classification problem, it is customary to use the sigmoid function

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

as activation funtion.
If

$$P(Y=1|x) = \sigma(f(x)) = \frac{e^{f(x)}}{1+e^{f(x)}}$$

then

$$P(Y=0|x) = 1-\sigma(f(x)) = \frac{1}{1+e^{f(x)}}$$

Softmax

When the result of the network is a probability distribution, e.g. over K different categories, the softmax function is used as activation:

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

It is easy to see that

$$0 < \text{softmax}(j, x_1, \dots, x_k) < 1$$

and most importantly

$$\sum_{j=1}^{k} \text{softmax}(j, x_1, \dots, x_k)$$

since we expect probablilites to sum up 1.

Softmax vs Sigmoid

It is easy to prove that for any c,

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

in particular, we can always assume one argument (corresponding to a "reference category") is null, taking e.g. $c = -x_k$.

In the binary case, we would be left with a single argument, and in particular

$$\sigma(x) = \text{softmax}(x, 0) = \frac{e^x}{e^x + e^0} = \frac{e^x}{e^x + 1}$$

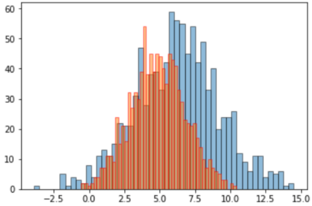
Cross entropy

Loss functions for probability distributions

If the intended output of the network is a probability distribution, we should find ways to compare it with the ground truth distribution (usually, but not necessarily, a categorical distribution).

Loss functions

What loss functions should we use for comparing probability distributions?



We could treat them as "normal functions", and use e.g. quadratic distance between true and predicted probabilities.

Can we do better? For instance, in logistic regression we do not use mean squared error, but use negative loglikelihood. Why?

Comparing distributions

Probability distributions can be compared according to many different metrics.
There are two main techniques:

- you consider their **difference** P - Q (e.g. Wasserstein distance)
- you consider their **ratio** P/Q (e.g. Kullback Leibler divergence)

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:

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

We call Cross-Entropy between P and Q the quantity

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

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!

Cross entropy and log-likelihood

Predicted log-likelyhood that X has label Y

$$logQ(Y|X)$$

We want to split it according to the possibile labels ℓ of Y:

$$logQ(\ell_1|X) + logQ(\ell_2|X) \ldots logQ(\ell_n|X)$$

but weighted in which way?

According to the **actual** probability that X has label ℓ:

$$P(\ell_1|X)logQ(\ell_1|X) + P(\ell_2|X)logQ(\ell_2|X) \ldots P(\ell_n|X)logQ(\ell_n|X)$$

or

$$\sum_{\ell} P(\ell|X)log(Q(\ell|X)$$

Cross entropy and log-likelihood

Predicted log-likelyhood that X has label Y

$$logQ(Y|X)$$

We want to split it according to the possibile labels ℓ of Y:

$$logQ(\ell_1|X) + logQ(\ell_2|X) \ldots logQ(\ell_n|X)$$

but weighted in which way?

According to the **actual** probability that X has label ℓ:

$$P(\ell_1|X)logQ(\ell_1|X) + P(\ell_2|X)logQ(\ell_2|X) \ldots P(\ell_n|X)logQ(\ell_n|X)$$

or

$$\sum_{\ell} P(\ell|X)log(Q(\ell|X)$$

Summing up

For **binary classification** use:

- sigmoid as activation function
- binary crossentropy (aka log-likelihod) as loss function

For **multinomial classification** use:

- softmax as activation function
- categorical crossentropy as loss function

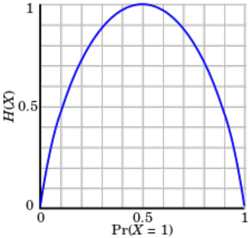
Appendix: Entropy recap

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 can be understood as the 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?

$$log(n)$$

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!!!

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!!!

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!!!

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!!!

Related notions:

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:

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