Training

The goal of training

We have a loss function $L = L(x, \theta)$ depending on data x and parameters θ.

The goal is to minimize it.

Changing what?

We cannot change data: we have a fixed data set of examples. We can only change the parameters of the model.

For the purposes of training, you must uniquely think of the loss function as a function of the parameters: $L = L(\theta)$

The goal of training

We have a loss function $L = L(x, \theta)$ depending on data x and parameters θ.

The goal is to minimize it.

Changing what?

We cannot change data: we have a fixed data set of examples.
We can only change the parameters of the model.

For the purposes of training, you must uniquely think of the loss function as a function of the parameters: $L = L(\theta)$

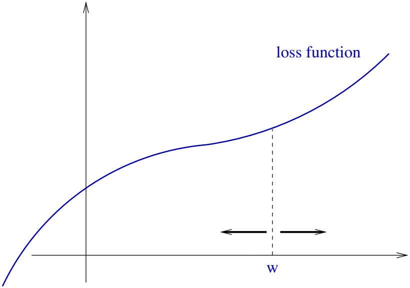
A naif approach: learning by trials

Evolutionary approach: *randomly perturb* weights and see if we get better results.
If so, save the change, else discharge.

- akin to reinforcement learning
- very inefficient
- high probability to make things worse

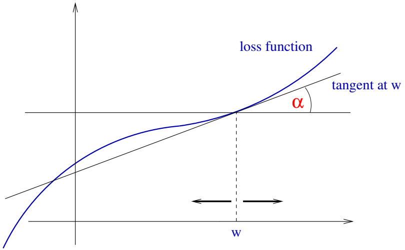
Predicting the adjustment

Instead of making a random adjustement of the parameters, can we predict it?



if the aim is to **decrease** the loss, should we move left or right?

Using derivatives



The mathematical tool we need are derivatives. The derivative is the tangent of the angle α

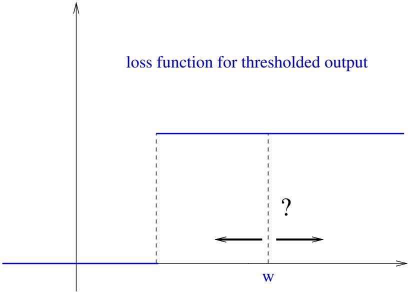
Sign and magnitude

The sign of the derivative provides orientation: it is positive if $\alpha < 90^\circ$ and negative if $90^\circ < \alpha < 180^\circ$.

If the derivative is positive we must decrease the parameter, if it is negative we must increase it (since we are descending).

The magnitude of the derivative is the related the steepness of the tangent: it is close to 0 if the angle is flat, and high when the angle is almost right.

Why binary threshold is no good for learning



Derivative is 0 everywhere (and infinite in correspondence of the jump).

The gradient

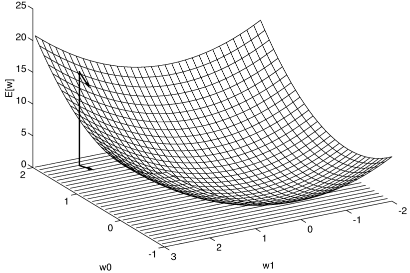
If we have many parameters, we have a different derivative for each of them (the so called partial derivatives).

The **vector** of all partial derivatives is called the gradient of the function.

$$\nabla_w[L(w)] = [\frac{\partial L(w)}{\partial w_1}, \ldots, \frac{\partial L(w)}{\partial w_n}]$$

With multiple parameters, the magnitude of partial derivatives becomes relevant, since it governs the orientation of gradient.

The gradient points in the direction of steepest ascent.



The gradient descent technique

The overall technique

1. start with a **random** configuration for the parameters
2. compute the gradient of the loss function
3. make a "small step" in the direction opposite to the gradient
4. iterate from step 2 until the loss is "sufficiently small"

- what is a small step?
- when should we stop iterating?

The overall technique

1. start with a **random** configuration for the parameters
2. compute the gradient of the loss function
3. make a "small step" in the direction opposite to the gradient
4. iterate from step 2 until the loss is "sufficiently small"

- what is a small step?
- when should we stop iterating?

Learning rate

The dimension of the step in the direction of the gradient is the so called learning rate, traditionally denoted with μ.

$$w \Leftarrow w - \mu \nabla L(w)$$

The learning rate is an hyperparameter that can be configured by the user.

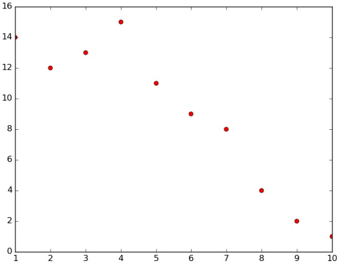
Its evolution during training is governed by software components called optimizers (more about them later).

Examples

Example 1: fitting a line

We want to fit a line through a set of points $\langle x_i, y_i \rangle$

- Model: a line $y = ax + b$
- Loss: $1/2 * \sum_i (y_i - (ax_i + b))^2$
- $\frac{\partial L}{\partial a} = -\sum_i ((y_i - ax_i - b)x_i)$
- $\frac{\partial L}{\partial b} = -\sum_i (y_i - ax_i - b)$



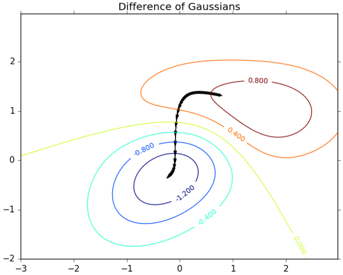
Demo

Why don't we solve it analytically?

The previous problem is a linear optimization problem, that can be easily solved analytically. Why taking a different approach?

- the analytic solution only works in the linear case, and for fixed error functions
- usually, it is not compatible with regularizers
- the backpropagation method can be generalized to multi-layer non-linear networks

Example 2: a general technique, but ...



Gradient descent is a general minimization technique, but it can

- end up in local minima
- get lost in plateau

Only guaranteed to work if the surface is concave

Demo

Optimizations

- Stochastic Gradient Descent
- Momentum

Optimizations

How often to update the weights

- Online: for each training sample
- Full batch: full sweep through the training data
- Mini-batch: for a small random set of training cases

How fast to update

- Use a fixed learning rate?
- Adapt the global learning rate?
- Adapt the learning rate on each connection separately?
- Use momentum?

suggested lecture: Geoffrey Hinton's lecture

Stochastic Gradient Descent

The gradient of the Loss function should be computed over all training samples (**fullbatch**).

This can be expensive, since the size of the dataset can be huge.

What happens if instead we use a small random subset? (**minibatch**)

- the direction of the gradient could be less precise
- in the limit case, it converges to the same value of the fullbatch case

Stochastic Gradient Descent

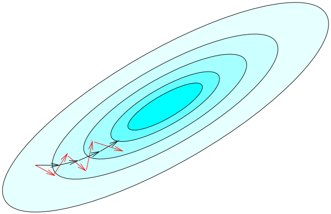
The gradient of the Loss function should be computed over all training samples (**fullbatch**).

This can be expensive, since the size of the dataset can be huge.

What happens if instead we use a small random subset? (**minibatch**)

- the direction of the gradient could be less precise
- in the limit case, it converges to the same value of the fullbatch case

Online vs Batch learning



Fullbatch 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.

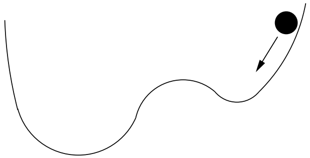
Minibatch (random subset of training samples): a good compromise.

Momentum

If, during consecutive training steps, the gradient seems to follow a stable direction, we could improve its magnitude, simulating the fact that it is acquiring a momentun along that direction, similarly to a ball rolling down a surface.

The hope is to reduce the risk to get stuck in a local minimum, or a plateau.

No theoretical justification



parameters updating with momentum

The momentum corrects the update at time t with a fraction of the update at time $t-1$.

Calling v^t the vector of updates at time t, we have the rules:

$$v^t = \underbrace{\mu * \nabla L(w)}_{\text{gradient step}} + \underbrace{\alpha * v^{t-1}}_{\text{momentum}}$$

Nesterov momentum

Nesterov momentum is a variant of the previous technique.
The difference is just the position at which the gradient is computed: before or after the momentum step: