TODOs
- Add sign and magnitude slide
- Add first slide info
- Add learning rate info from lesson
- Review part of SGD

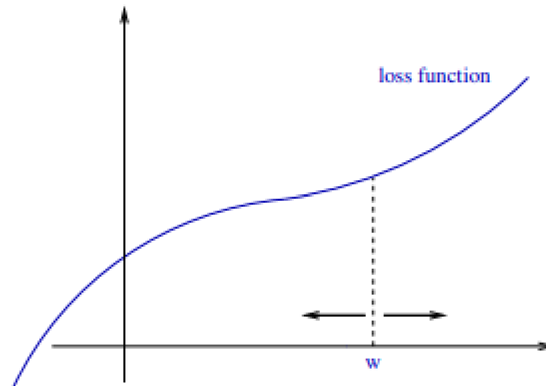
The loss depends on the dataset, and it also depends on the current model, which is defined by a set of parameters.

For the purposes of training, you should consider the loss function as a functino of the parameters $L = L(\theta)$.

Learning by trials (naif approach)

Evolutionary approach: *randomly perturb weights* and see if *we get better results*. If so, save the change, else discharge. This is akin to reinforcement learning, but is *very inefficient* and has a high probability to make things worse.

Instead of making a random adjustement of the parameters, we try to predict the parameters.

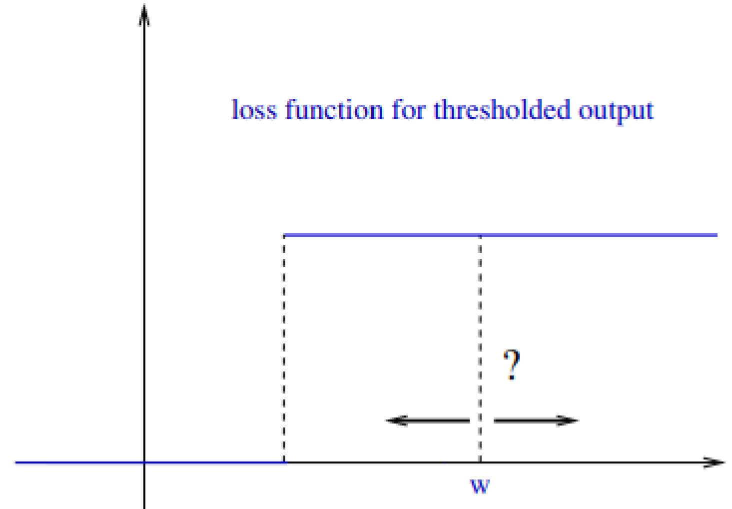


w is the current value of the loss function, and we should move left (in this example) to decrease the loss. We also have to compute how long we have to move to decrease the loss.

Obviously, the mathematical tool we need to find the movement direction are *derivatives*. The derivative is the tangent of the angle α. The magnitude of the derivative tells us how much we are close to a stationary point. - If the derivative is 0, we are either close to the minimum, or maybe we're are in a plateu.

[insert slide?]

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: […]

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

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

The gradient points in the direction of *steepest* ***ascent***.

The gradient descent technique (AGAIN)

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"

We have to answer to these questions first though: - what is a small step? - when should we stop iterating?

Learning rate

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

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

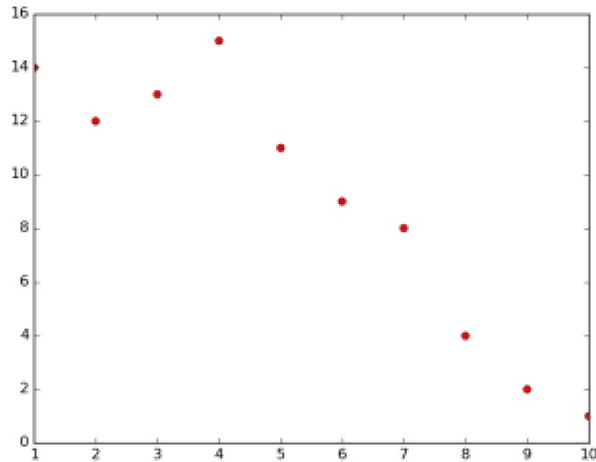
The learning rate is an hyperparameter that can be configured by the user. Its evolution during training is governed by software components called *optimizers* (more about them later).

[riguardare lezione in questa slide, sono manco 5 minuti bro dai]

Examples

Linear regression example (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: $1/2 * \sum_i (y_i - (ax_i + b))^2$
- $\dfrac{\partial L}{\partial a} = -\sum_i ((y_i - ax_i + b)x_i)$



- $\dfrac{\partial L}{\partial b} = -\sum_i (y_i - ax_i + b)$
  The previous problem is a linear optimization problem, that can be easily solved analytically. Why taking a different approach? I the analytic solution only works in the linear case, and for fixed error functions I usually, it is not compatible with regularizers I the backpropagation method can be generalized to multi-layer non-linear networks

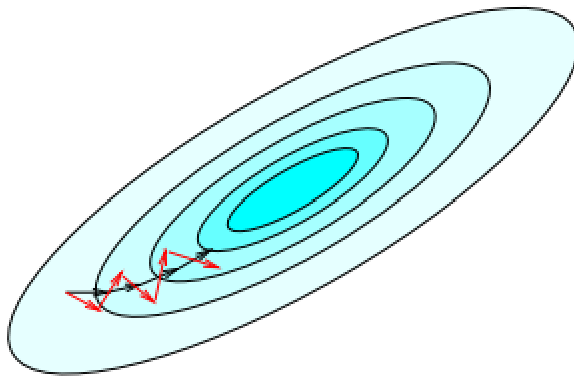
Some notes on gradient descent Gradient descent is a general minimization technique, but it can - end up in local minima - get lost in plateau Only guaranteed to work (find an absolute minimum) if the surface is *concave*.

Optimizations

How often to update the weights (in an epoch)? - 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? Each asnwer to these questions returns an new *optimizer* definition.

Online vs Batch learning



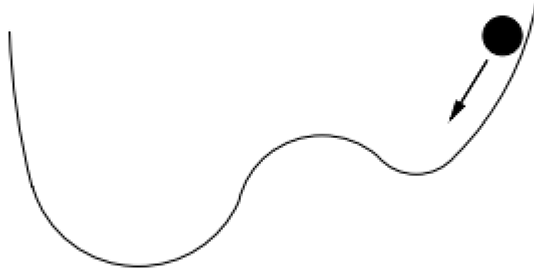
- Fullbatch on all training samples: gradient points to the direction of steepest descent on the error surface (perpendicular to contour lines of the error surface) - Online (one sample at a time) gradient zig-zags around the direction of the steepest descent. - Minibatch (random subset of training samples): a good compromise.

Stochastic Gradient Descent

[…] We make less precise updates more frequently.

Momentum

If, during consecutive training steps, the gradient seems to follow a stable direction, we could improve (increase) its magnitude (we increase the gradient), 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. There's no theoretical justification.

Most of the optimizers have some sort of momentum implemented.