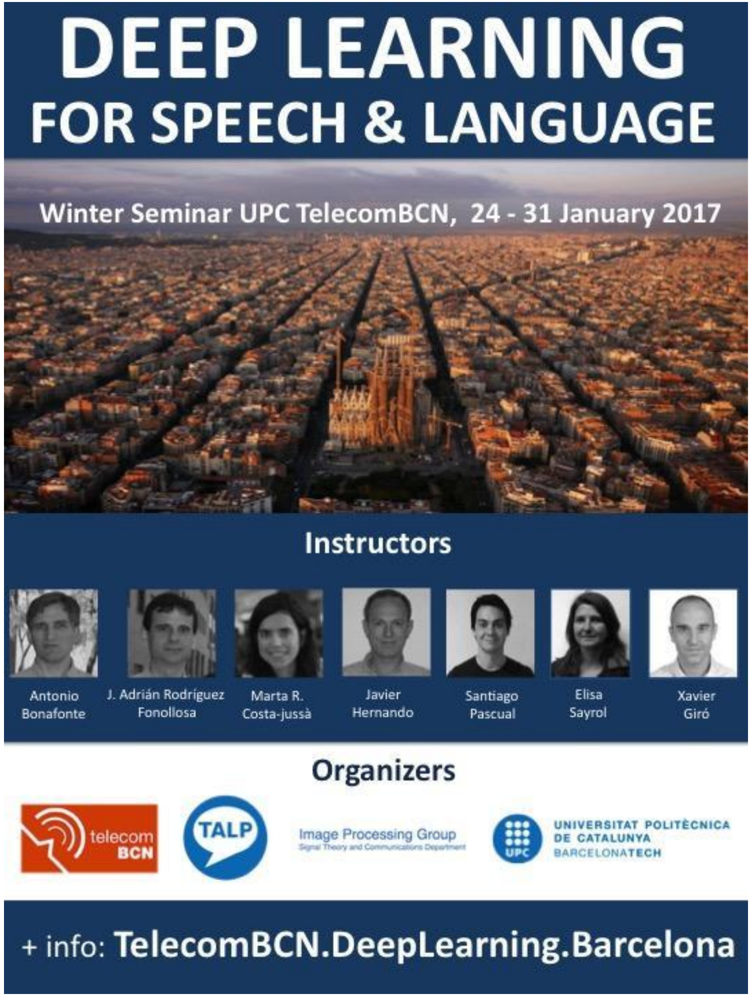



[course site]

Day 1 Lecture 5

# Backpropagation in Deep Networks

Elisa Sayrol

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Department of Signal Theory and Communications
*Image Processing Group*

# From previous lectures

<u>L Hidden Layers</u>

<u>Hidden pre-activation (*k*>0)</u>

$$\mathbf{a}^{(k+1)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k)}(\mathbf{x})$$

$$\mathbf{h}^{(1)}(\mathbf{x}) = \mathbf{x}$$

<u>Hidden activation (*k*=1,…*L*)</u>

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

<u>Output activation (*k*=*L*+1)</u>

$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$

$$o(\mathbf{a}) = \mathrm{softmax}(\mathbf{a}) = \left[ \frac{\exp(a_1)}{\sum_c \exp(a_c)} \cdots \frac{\exp(a_C)}{\sum_c \exp(a_c)} \right]^{\mathrm{T}}$$

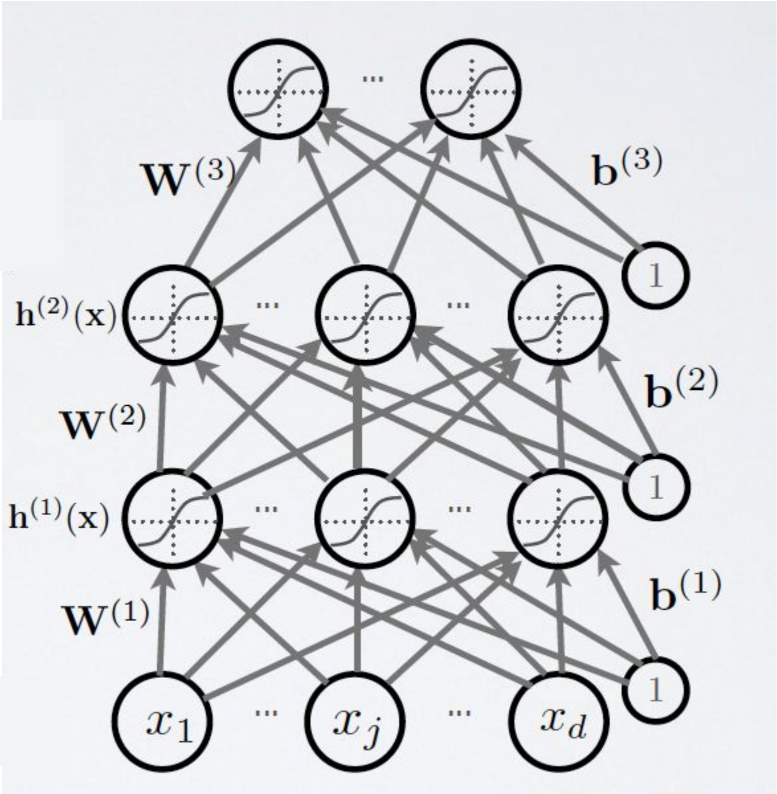



*Figure Credit: Hugo Laroche NN course*

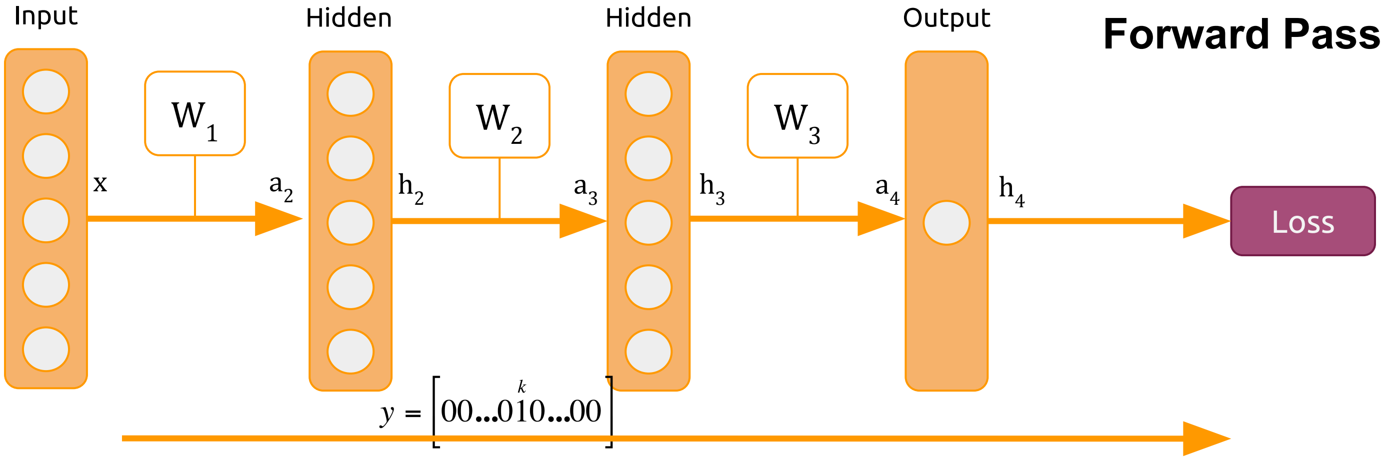
# Backpropagation algorithm

The output of the Network gives class **scores** that depens on the input and the parameters

$$f(\mathbf{x}) = \mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{o}(\mathbf{b}^{(L)} + \mathbf{W}^{(L)}\mathbf{h}^{(L)}(\mathbf{x}))$$

- Define a **loss function** that quantifies our unhappiness with the scores across the training data.
- Come up with a way of efficiently finding the parameters that minimize the loss function (**optimization**)

# Forward Pass

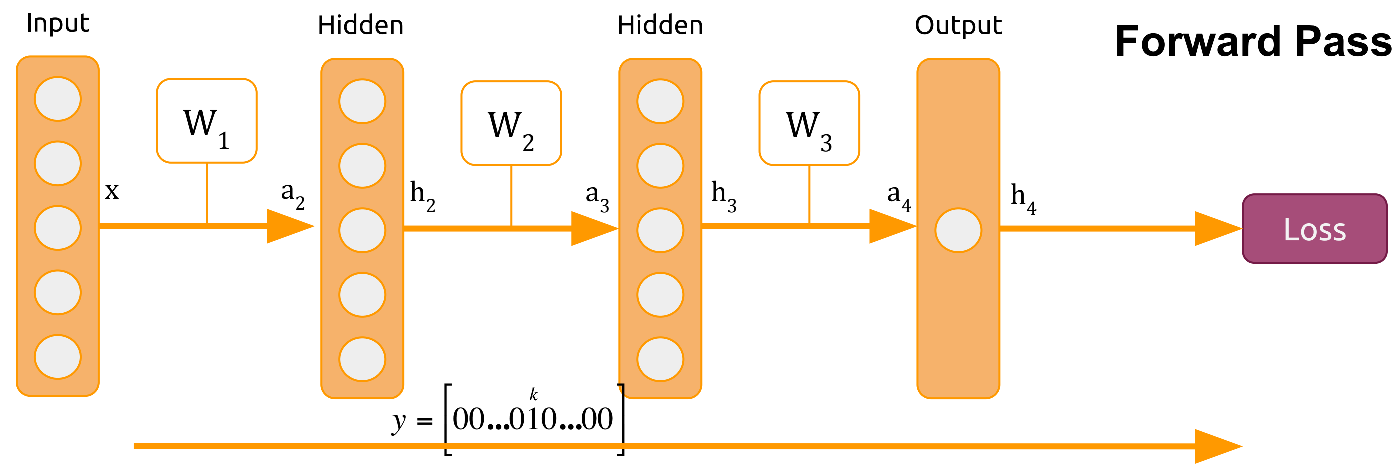



**Probability Class given an input (softmax)**

$$p(c_k = 1|\mathbf{x}) = \frac{\exp(a_k)}{\sum_c \exp(a_c)}$$

*Figure Credit: Kevin McGuiness*

## Forward Pass




### Probability Class given an input (softmax)

$$p(c_k = 1|\mathbf{x}) = \frac{\exp(a_k)}{\sum_c \exp(a_c)}$$

### Loss function; e.g., negative log-likelihood (good for classification)

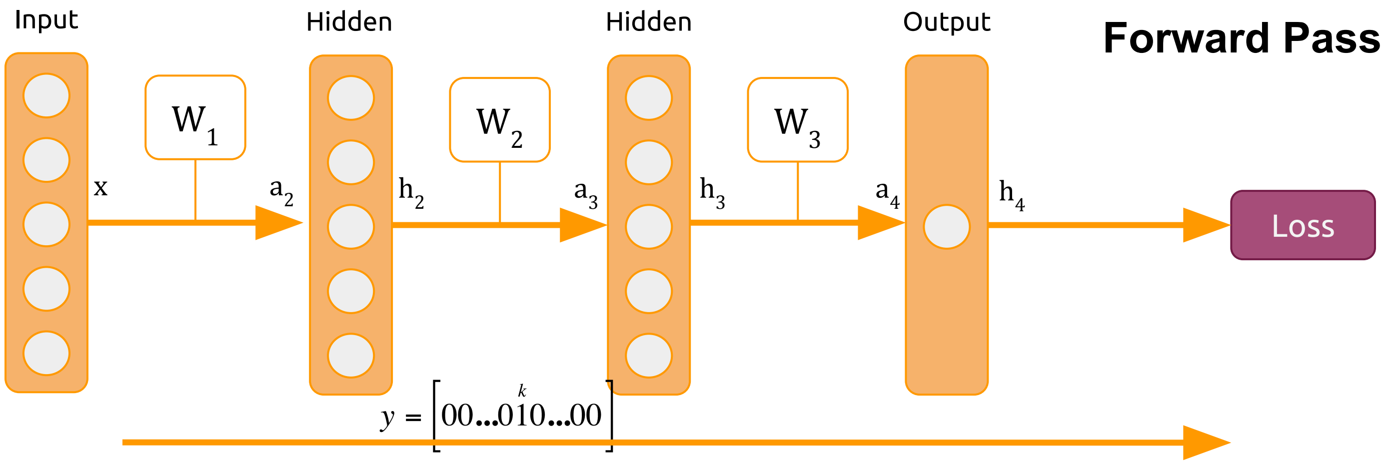
$$L(\mathbf{x}, y; \mathbf{W}) = -\sum_j y_j \log(p(c_j|\mathbf{x}))$$

### Regularization term (L2 Norm) aka as weight decay

$$L(\mathbf{x}, y; \mathbf{W}) = -\sum_j y_j \log(p(c_j|\mathbf{x})) + \frac{\lambda}{2}\|\mathbf{W}\|_2^2$$

*Figure Credit: Kevin McGuiness*

# Forward Pass




**Probability Class given an input (softmax)**

$$p(c_k = 1|\mathbf{x}) = \frac{\exp(a_k)}{\sum_c \exp(a_c)}$$

**Loss function; e.g., negative log-likelihood (good for classification)**

$$L(\mathbf{x}, y; \mathbf{W}) = -\sum_j y_j \log(p(c_j|\mathbf{x}))$$

**Regularization term (L2 Norm) aka as weight decay**

$$L(\mathbf{x}, y; \mathbf{W}) = -\sum_j y_j \log(p(c_j|\mathbf{x})) + \frac{\lambda}{2}\|\mathbf{W}\|_2^2$$

**Minimize the loss (plus some regularization term) w.r.t. Parameters over the whole training set.**

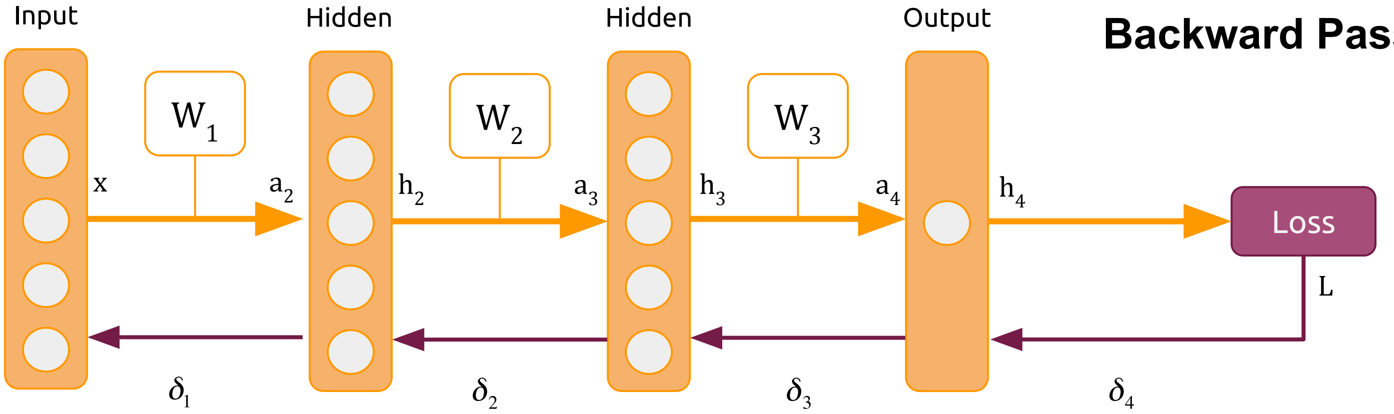
$$\mathbf{W}^* = argmin_\theta \sum_j L(\mathbf{x}^n, y^n; \mathbf{W})$$

*Figure Credit: Kevin McGuiness*

# Backpropagation algorithm

- We need a way to fit the model to data: find parameters ($\mathbf{W}^{(k)}$, $\mathbf{b}^{(k)}$) of the network that (locally) minimize the loss function.
- We can use **stochastic gradient descent**. Or better yet, mini-batch stochastic gradient descent.
- To do this, we need to find the gradient of the loss function with respect to all the parameters of the model ($\mathbf{W}^{(k)}$, $\mathbf{b}^{(k)}$)
- These can be found using the **chain rule** of differentiation.
- The calculations reveal that the gradient wrt. the parameters in layer k only depends on the error from the above layer and the output from the layer below.
- This means that the gradients for each layer can be computed iteratively, starting at the last layer and propagating the error back through the network. This is known as the **backpropagation** algorithm.

*Slide Credit: Kevin McGuiness*




# Backward Pass

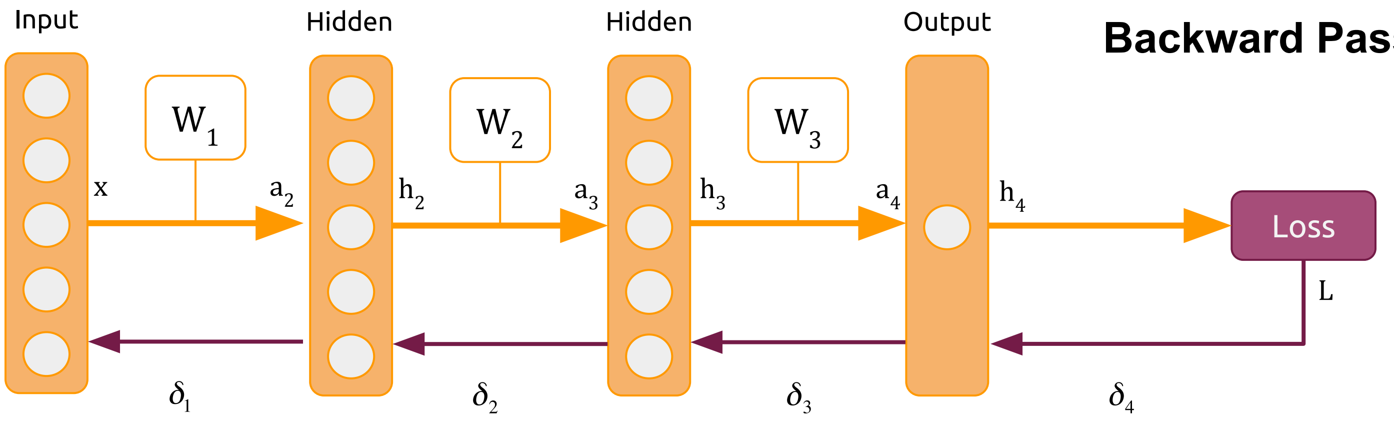
**1. Find the error in the top layer:**

$$\delta_K = \frac{\partial L}{\partial a_K}$$

$$\delta_K = \frac{\partial L}{\partial h_K}\frac{\partial h_K}{\partial a_K}$$

$$\delta_K = \frac{\partial L}{\partial h_K} \bullet g'(a_K)$$

*Figure Credit: Kevin McGuiness*




# Backward Pass

### 1. Find the error in the top layer:

$$\delta_K = \frac{\partial L}{\partial a_K}$$

$$\delta_K = \frac{\partial L}{\partial h_K}\frac{\partial h_K}{\partial a_K}$$

$$\delta_K = \frac{\partial L}{\partial h_K} \bullet g'(a_K)$$

### 2. Compute weight updates

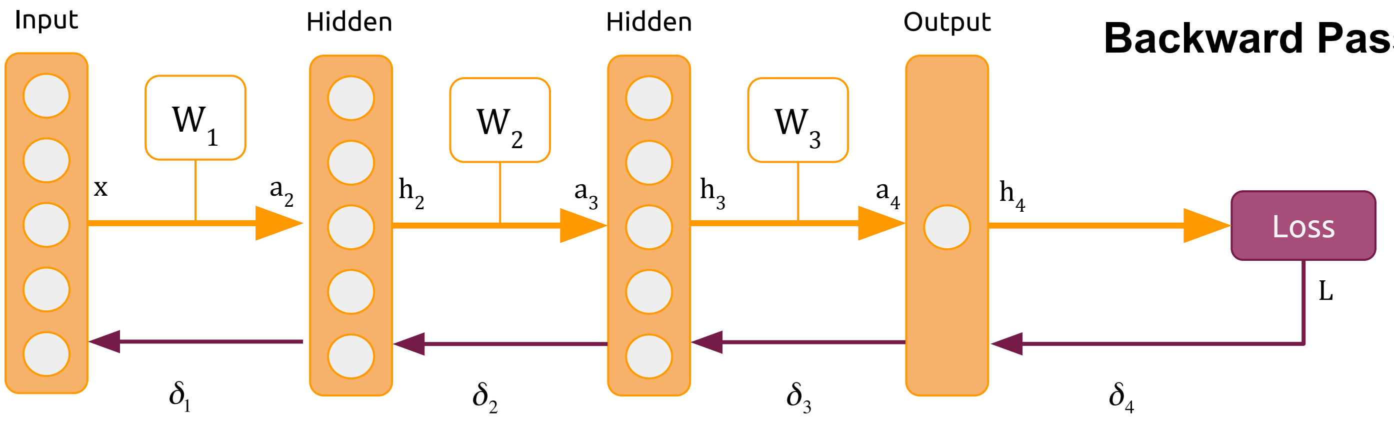
$$\frac{\partial L}{\partial W_k} = \frac{\partial L}{\partial a_{k+1}}\frac{\partial a_{k+1}}{\partial W_k}$$

$$\frac{\partial L}{\partial W_k} = \frac{\partial L}{\partial a_{k+1}} \bullet h_k$$

$$\frac{\partial L}{\partial W_k} = \delta_{k+1} \bullet h_k$$

**To simplify we don't consider the biass**

*Figure Credit: Kevin McGuiness*




# Backward Pass

### 1. Find the error in the top layer:

$$\delta_K = \frac{\partial L}{\partial a_K}$$

$$\delta_K = \frac{\partial L}{\partial h_K}\frac{\partial h_K}{\partial a_K}$$

$$\delta_K = \frac{\partial L}{\partial h_K} \bullet g'(a_K)$$

### 2. Compute weight updates

$$\frac{\partial L}{\partial W_k} = \frac{\partial L}{\partial a_{k+1}}\frac{\partial a_{k+1}}{\partial W_k}$$

$$\frac{\partial L}{\partial W_k} = \frac{\partial L}{\partial a_{k+1}} \bullet h_k$$

$$\frac{\partial L}{\partial W_k} = \delta_{k+1} \bullet h_k$$

**To simplify we don't consider the biass**

### 3. Backpropagate error to layer below

$$\delta_k = \frac{\partial L}{\partial a_k}$$

$$\delta_k = \frac{\partial L}{\partial a_{k+1}}\frac{\partial a_{k+1}}{\partial h_k}\frac{\partial h_k}{\partial a_k}$$

$$\delta_k = W_k^T \frac{\partial L}{\partial a_{k+1}} \bullet g'(a_k)$$

$$\delta_k = W_k^T \delta_{k+1} \bullet g'(a_k)$$

*Figure Credit: Kevin McGuiness*

# Optimization

## Stochastic Gradient Descent

$$\mathbf{W} \longleftarrow \mathbf{W} - \eta \frac{\partial L}{\partial W}$$

$\eta$ : learning rate

## Stochastic Gradient Descent with momentum

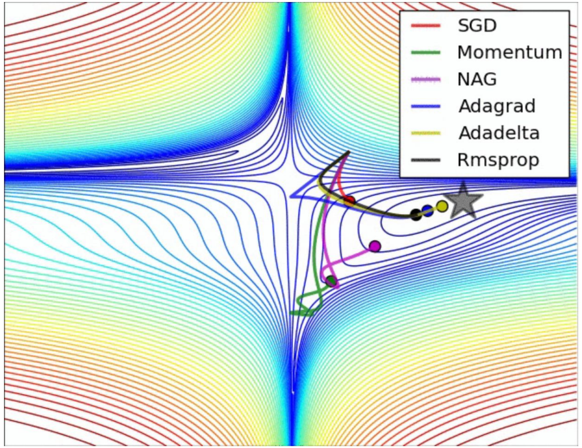
$$\mathbf{W} \longleftarrow \mathbf{W} - \eta \Delta$$

$$\Delta \longleftarrow 0.9\Delta + \frac{\partial L}{\partial \mathbf{W}}$$

## Stochastic Gradient Descent with L2 regularization

$$\mathbf{W} \longleftarrow \mathbf{W} - \eta \frac{\partial L}{\partial W} - \lambda |\mathbf{W}|$$

$\lambda$: weight decay




Sebastian Ruder Blog

Recommended lectures:

[Backpropagation: http://cs231n.github.io/optimization-2/](http://cs231n.github.io/optimization-2/)

[Optimization: http://sebastianruder.com/optimizing-gradient-descent/](http://sebastianruder.com/optimizing-gradient-descent/)

# “Vanishing Gradients”

In the backward pass you might be in the flat part of the sigmoid (or any other activation function like tanh) so derivative tends to zero and your training loss will not go down