Acknowledgements: To my colleagues of this seminar and previous ones

Day 2 Lecture 1

Backpropagation

Elisa Sayrol

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

Training MLPs

With Multiple layers we need to minimize the **loss function** $\mathcal{L}(f_\theta(x), y)$ with respect to all the parameters of the model $\theta(W^{(k)}, b^{(k)})$:

$$W^* = argmin_\theta \mathcal{L}(f_\theta(x), y)$$

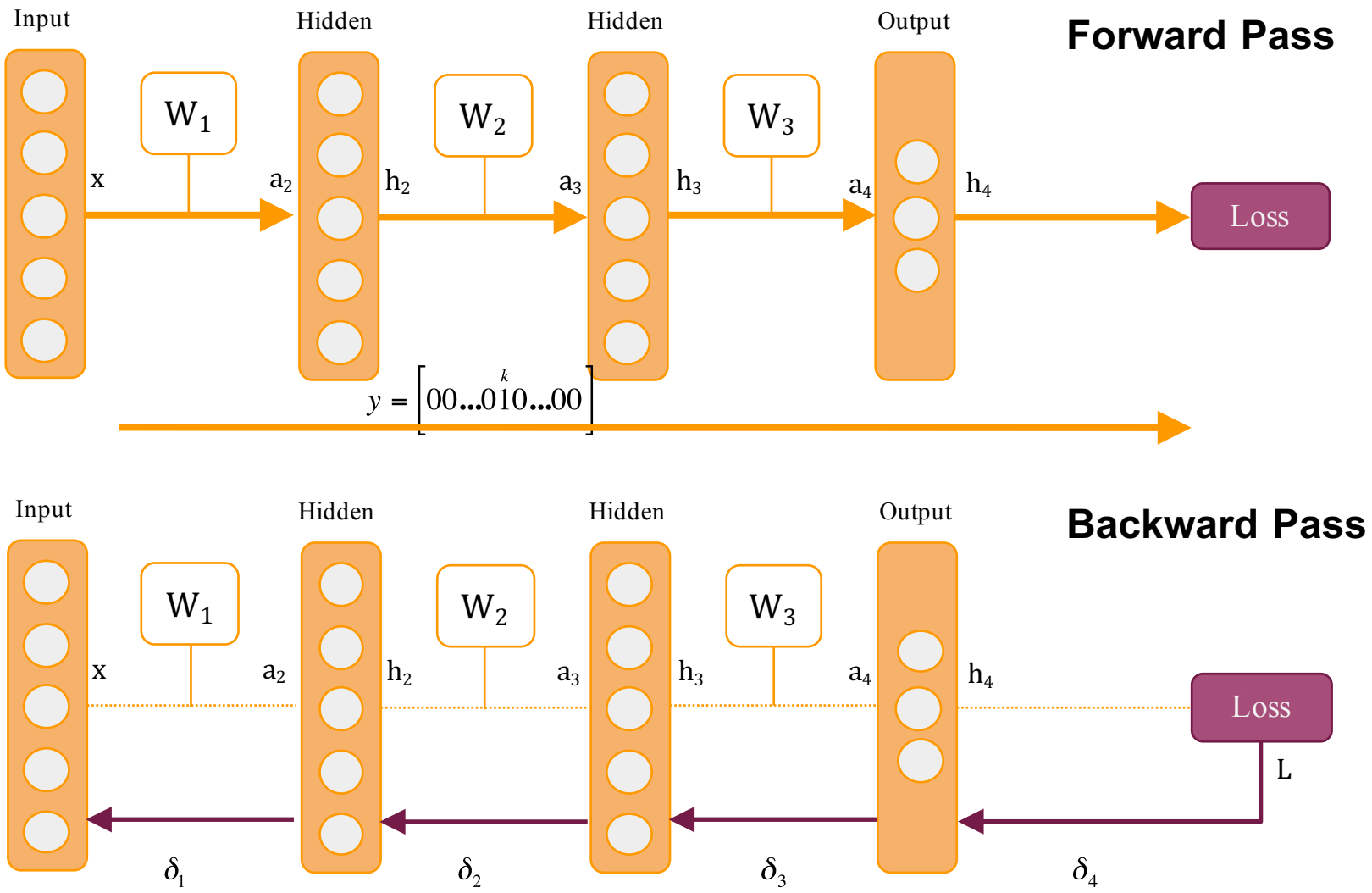
Gradient Descent: Move the parameter θ_j in small steps in the direction opposite sign of the derivative of the loss with respect j:

$$\theta_j^{(n)} = \theta_j^{(n-1)} - \alpha^{(n-1)} \cdot \nabla_{\theta_j} \mathcal{L}(y, f(x))$$

Stochastic gradient descent (SGD): estimate the gradient with one sample, or better, with a **minibatch** of examples.

For MLP gradients 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.



Backpropagation is applied to the Backward Pass

From computational graphs to MLP

Applying the Chain Rule to Computational Graphs

Numerical Examples

$$f(x,y,z) = (x+y)z$$

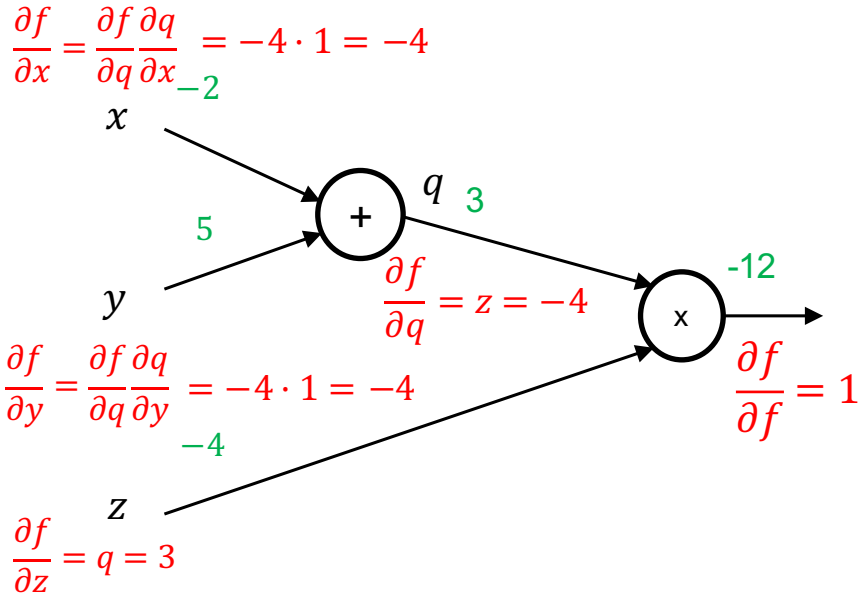
$$q = (x+y) \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z \qquad \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial f} = 1$$

$We\ want\ to\ compute$: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$Example\ x = -2, y = 5, z = -4$



From Stanford Course: Convolutional Neural Networks for Visual Recognition 2017

From computational graphs to MLP

Applying the Chain Rule to Computational Graphs

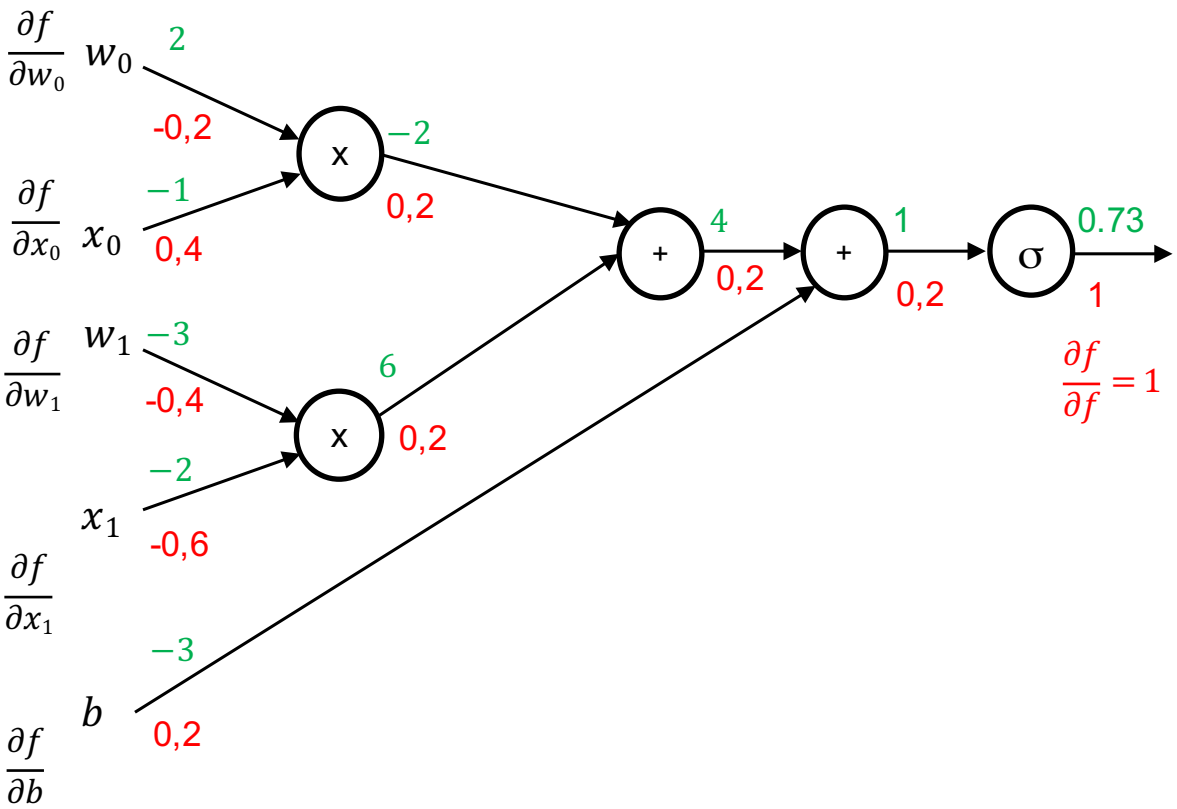
Numerical Examples

$$f(x, y, z) = \sigma(w_0 x_0 + w_1 x_1 + b)$$

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

$$\frac{d\sigma(x)}{x} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left(\frac{1 + e^{-x} - 1}{(1 + e^{-x})}\right)\left(\frac{1}{(1 + e^{-x})}\right)$$

$$\frac{d\sigma(x)}{x} = (1 - \sigma(x))(\sigma(x))$$



From Stanford Course: [Convolutional Neural Networks for Visual Recognition]

From computational graphs to MLP

Gates. Backward Pass

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

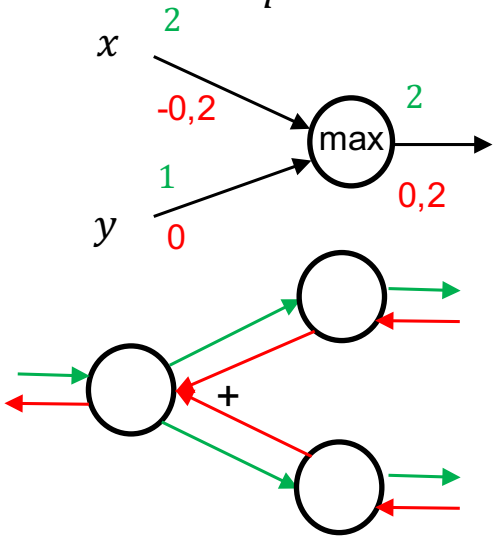
In general: Derivative of a function

$$q = (x+y) \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

Sum: Distributes the gradient to both branches

$$f = qz \qquad \frac{\partial f}{\partial q} = z \qquad \frac{\partial f}{\partial z} = q$$

Product: Switches gradient weigth values



Max: Routes the gradient only to the higher input branche (not sensitive to the lower branche)

Add branches: Branches that split in the forward pass and merge in the backward pass, add gradients

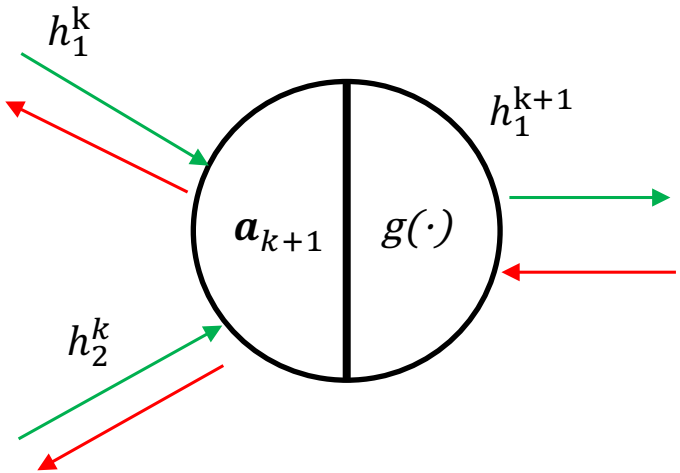
From computational graphs to MLP

For a single neuron with its linear and non-linear part

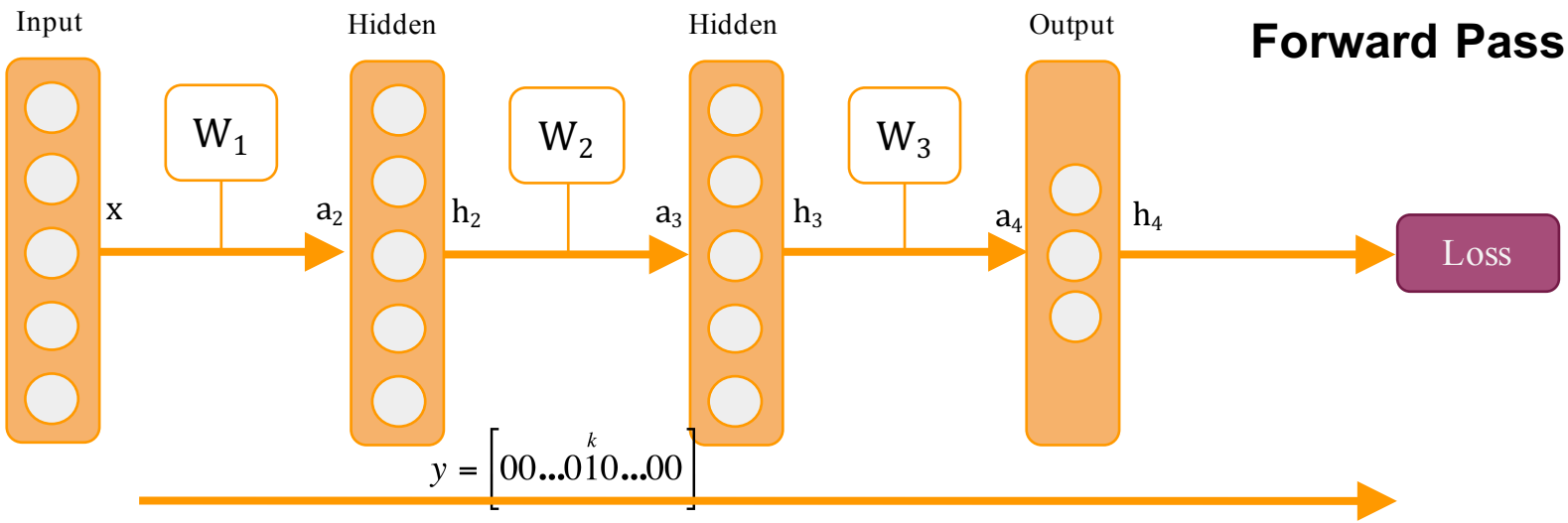
$$\boldsymbol{h}_{k+1} = g(\boldsymbol{W}_k \boldsymbol{h}_k + \boldsymbol{b}_k) = g(\boldsymbol{a}_{k+1})$$

$$\frac{\partial \boldsymbol{h}_{k+1}}{\partial \boldsymbol{a}_{k+1}} = g'(\boldsymbol{a}_{k+1})$$

$$\frac{\partial \boldsymbol{a}_{k+1}}{\partial \boldsymbol{h}_k} = \boldsymbol{W}_k$$



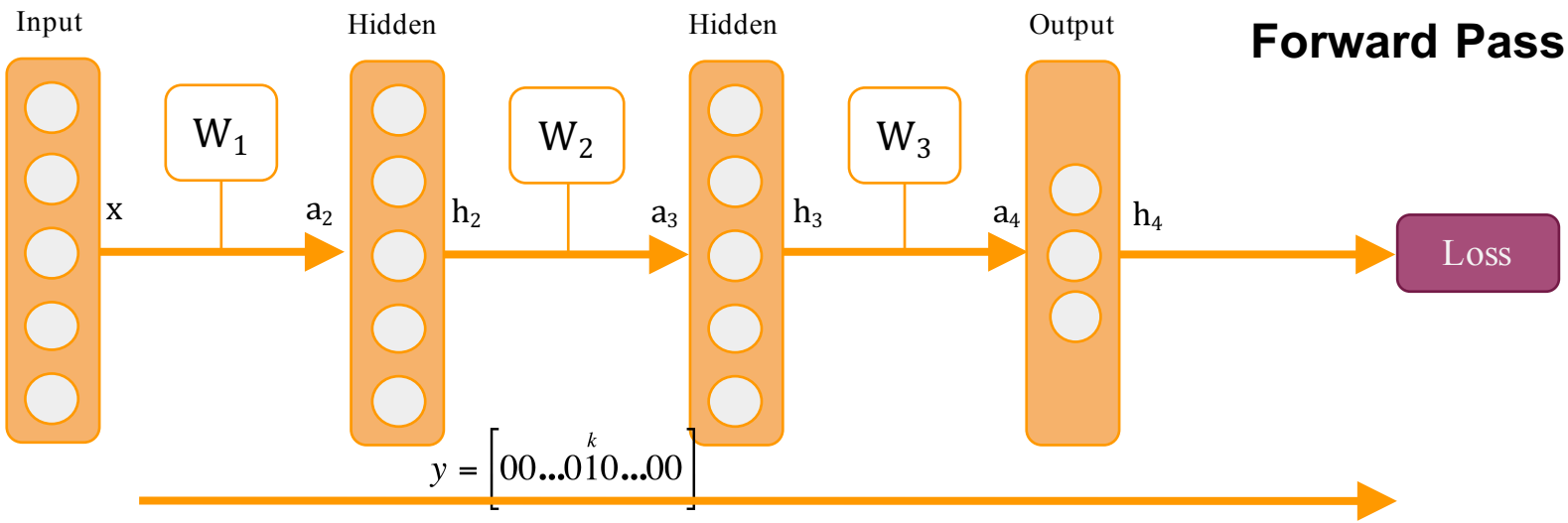
Usually we apply the back-propagation algorithm to tensors of arbitrary dimensionality



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



Probability Class given an input (softmax)

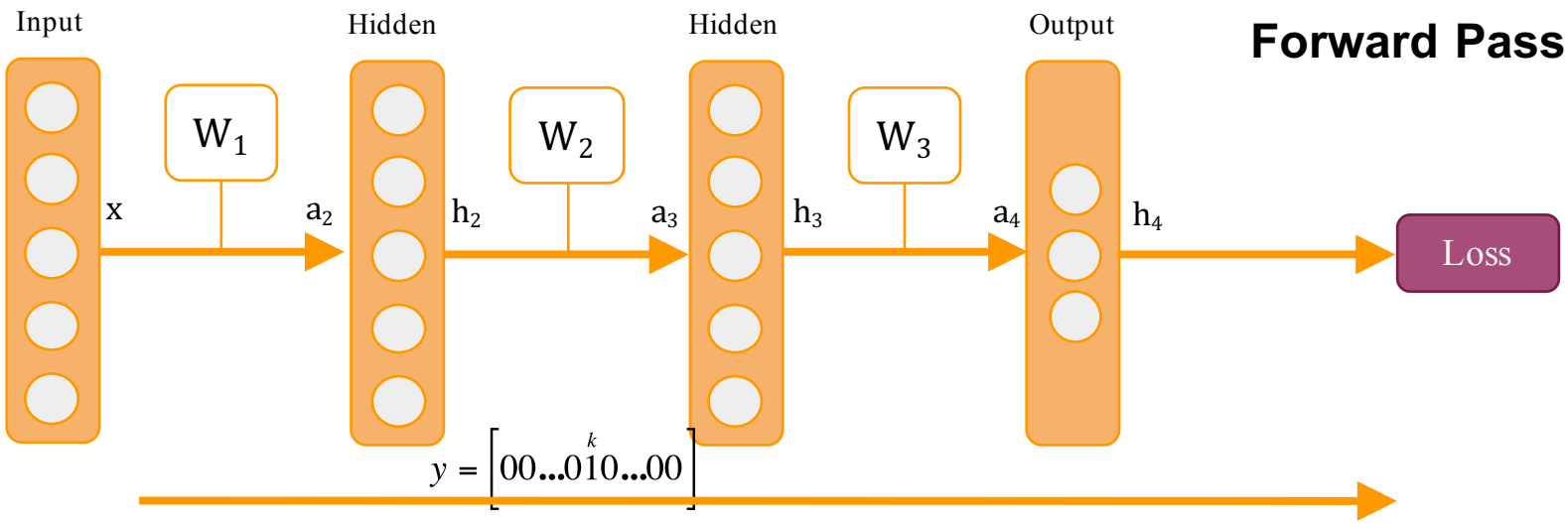
$$p(c_k = 1|\mathbf{x}) = \frac{\exp(a_k)}{\sum_c \exp(a_c)}$$

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

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



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)

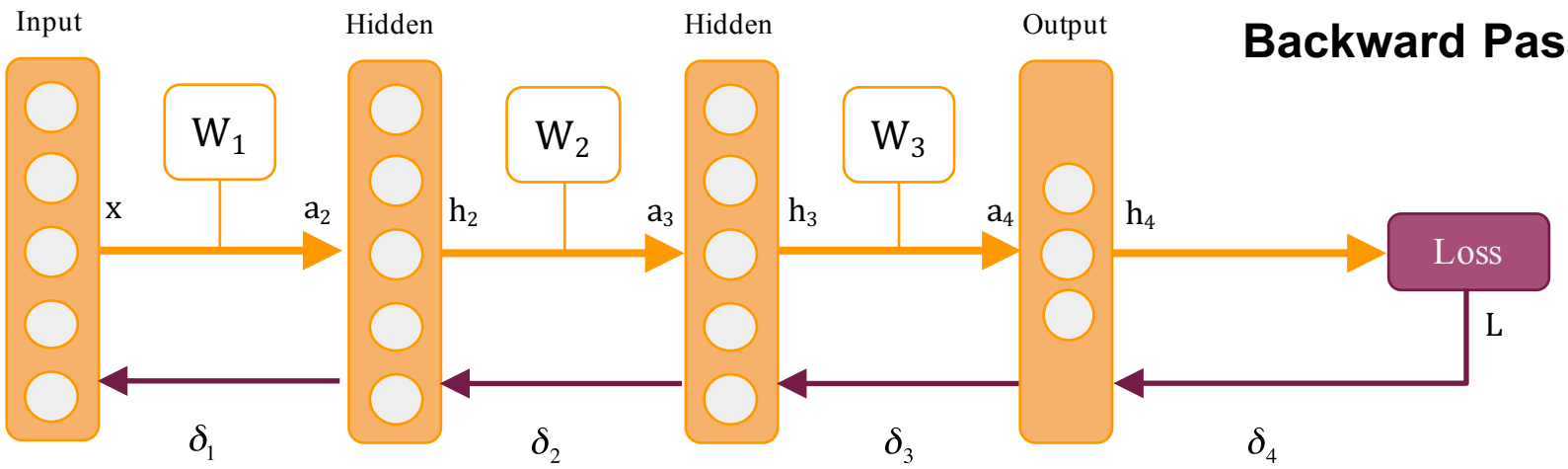
$$L(\mathbf{x}, y; \mathbf{W}) = -\sum_j y_j \log(p(c_j|\mathbf{x}))$$

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.

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

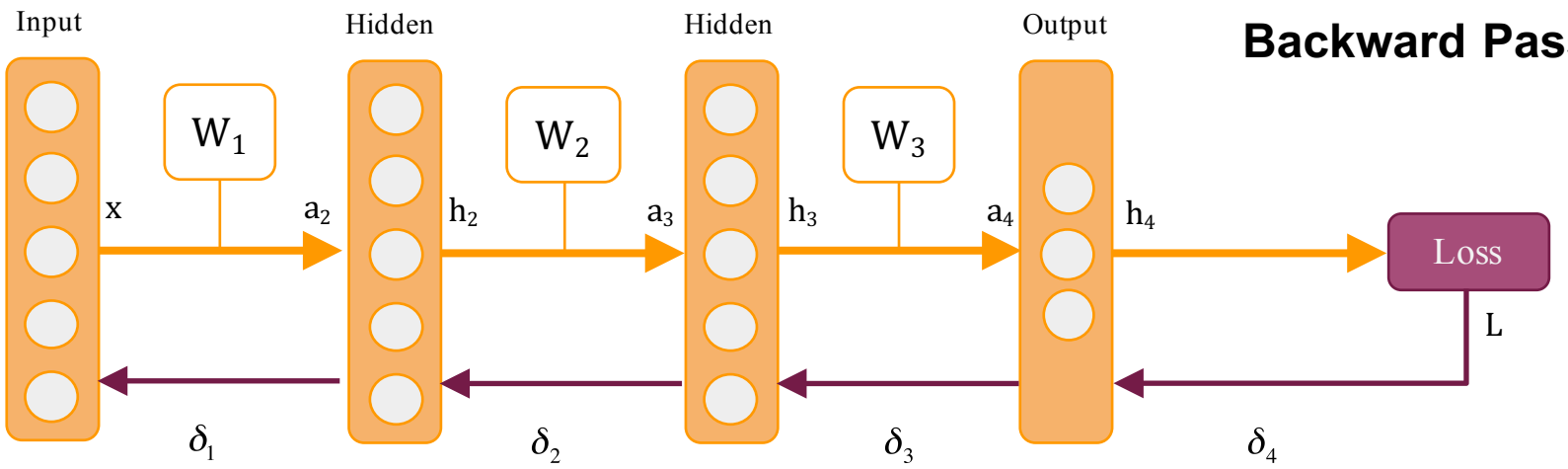


1. Find the error in the top layer:

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

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

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



1. Find the error in the top layer:

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

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

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

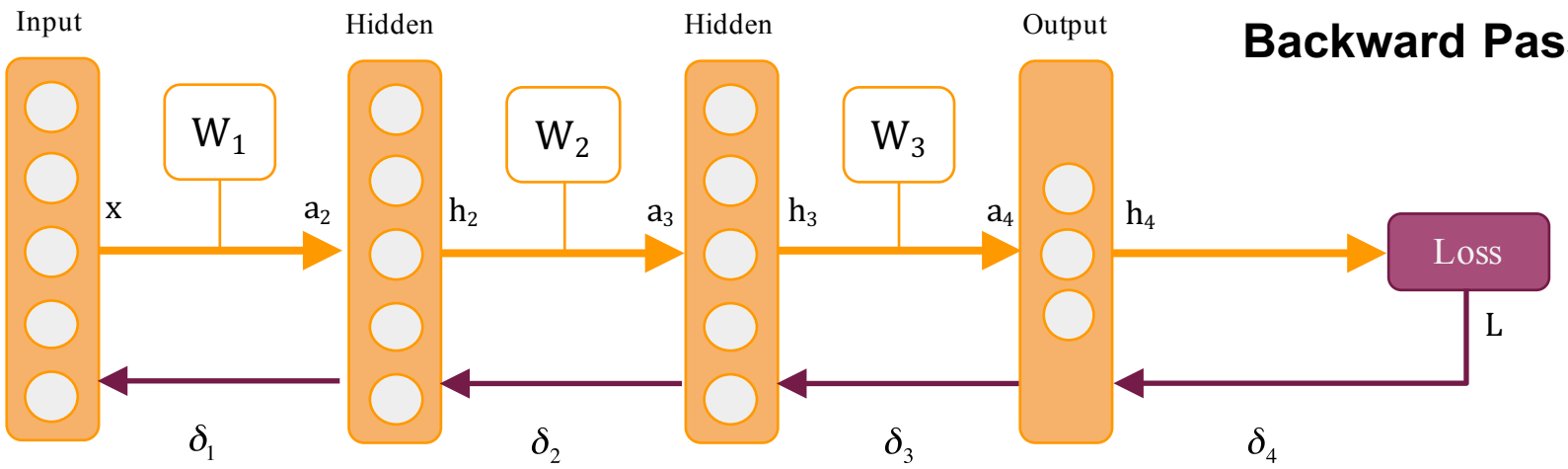
2. Compute weight updates

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

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

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

To simplify we don't consider the bias



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

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

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

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

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

Weight initialization

Need to pick a starting point for gradient descent: an initial set of weights

Zero is a very **bad idea**!

- Zero is a **critical point**
- Error signal will not propagate
- Gradients will be zero: no progress

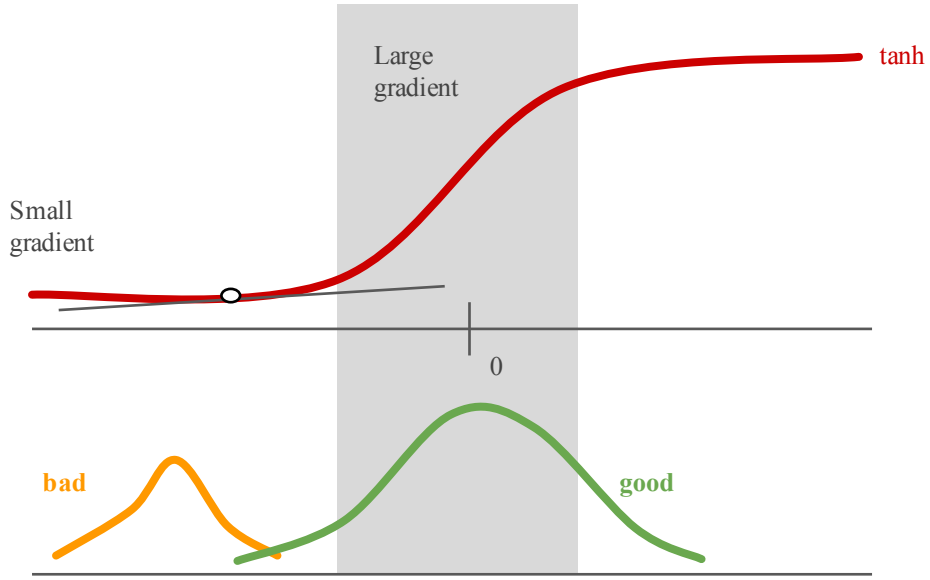
Constant value also bad idea:

- Need to break symmetry

Use **small random values:**

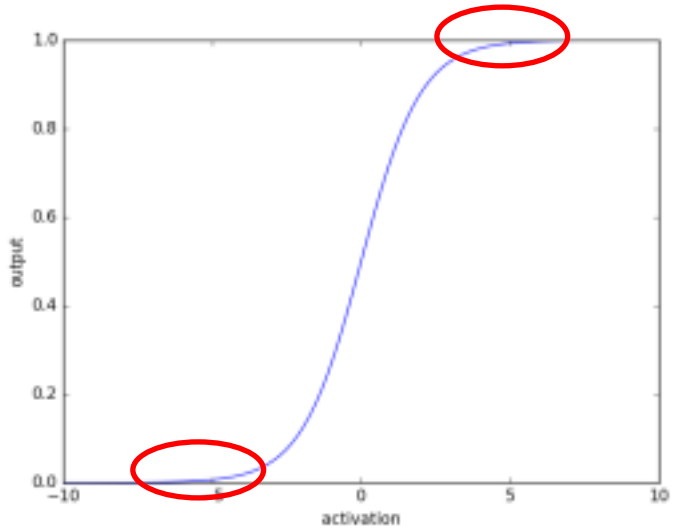
- E.g. zero mean Gaussian noise with constant variance

Ideally we want inputs to activation functions (e.g. sigmoid, tanh, ReLU) to be mostly **in the linear area** to allow larger gradients to propagate and converge faster.



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



Summary

- Backpropagation is applied during the Backward pass while training
- Computational graphs help to understand the chain rule of differentiation
- Parameters in layer k only depend 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.
- You should take of weight inizialization and the problem of vanishing gradients
- For a “deeper” study: http://www.deeplearningbook.org/