INTRODUCTION TO NEURAL NETWORKS

Lecture 6. Backpropagation

Dr. Jingxin Liu

School of AI and Advanced Computing

Xi'an Jiaotong-Liverpool University
西交利物浦大学

Updated at 13 March

Recap: Feedforward Neural Networks

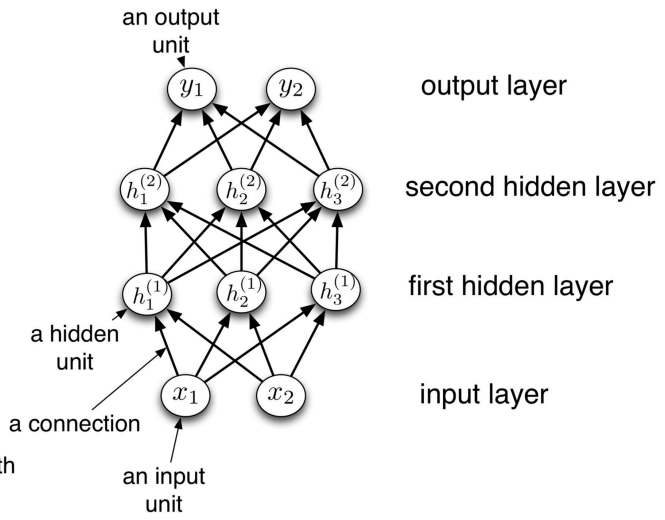
Build a feed-forward neural network which defines a **mapping** from **input x** to **output y**

$$\hat{y} = f(x; \boldsymbol{W})$$

$$= f^N\ (f^{N-1}(\dots f^2(f^1(x, W^1), W^2), W^{N-1}), W^N)$$

$$= f^N\ (h^{N-1}(\dots h^2(h^1(x, W^1), W^2), W^{N-1}), W^N)$$

depth



Final layers

Hidden layers

For the kth neuron in the nth hidden Layer:

$$h_k^n = g_k^n\ (\sum_{i=0} a_i w_{ki}^n)$$

Activation function

Weight

output of the previous layer

Recap: Feedforward Neural Networks

Define an activation function for the output layer:

- **Sigmoid:** $\frac{1}{1+e^{-z}}$
- **Softmax:** $\frac{\exp(z_k)}{\sum_p^K \exp(z_p)}$

Forward propagation: accept input x, pass through intermediate stages and obtain output $\hat{y}$

Define a loss function:

- **Squared loss:** $\sum_M \frac{1}{2}(\hat{y} - y)^2$
- **Cross-entropy loss:** $-\sum_{x \epsilon X} p(x) \cdot \log q(x)$

Recap: Gradient Descent

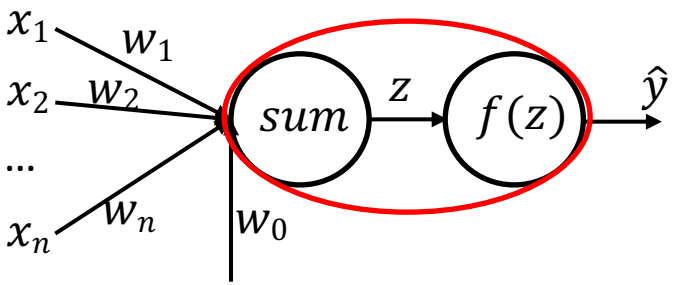
Gradient Descent for Linear regression or Logistic regression

- Initialize θ
- Repeat until convergence {

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} \mathcal{J}(\theta)$$

}

This also works for signal perceptron



Recap: Gradient Descent

We need a design a workflow with gradient descent for learning the weights (parameters) of multi-layer feedforward neural networks.

Gradient Descent for Feedforward Neural Networks

- Initialize $\boldsymbol{W}$
- Repeat until convergence {
  - Given input x, propagate activity forward (**forward propagation**), generate a training error
  - Change the weights in the direction of the negative gredient

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \frac{\partial J}{\partial \boldsymbol{W}}$$

}

Backpropagation

However, we don't have targets for hidden neurons.

It is impossible to compute error signal for internal neurons directly, because output values of these neurons are unknown. For many years the effective method for training multi-layer networks has been unknown. This is known as the credit assignment problem.

Only in the middle 80's the backpropagation algorithm has been worked out.

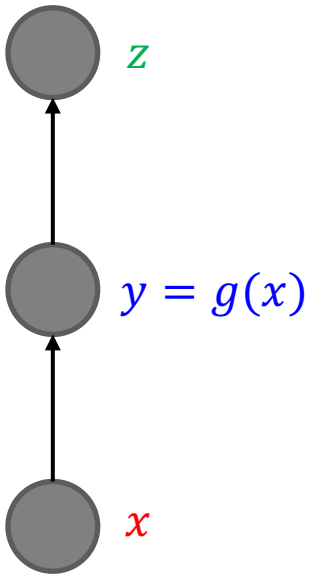
- *Learning Internal Representations by Error Propagation, Rumelhart, Hinton and Williams, 1986.*

The idea is to propagate error signal (computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.

Backpropagation so far is the most successful learning algorithm for training MLP's.

Backpropagation – Chain Rule

Suppose $y = g(x)$ and $z = f(y) = f(g(x))$



Univariate Chain Rule: $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$

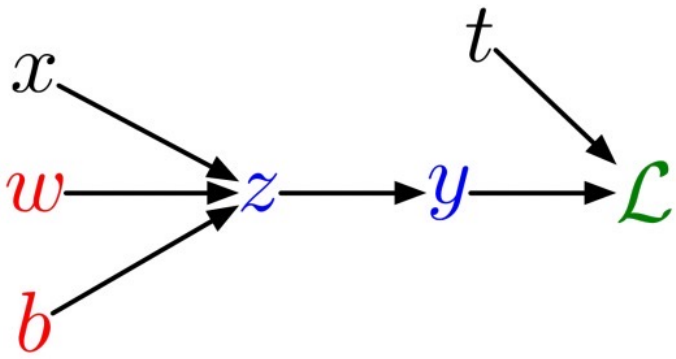
Backpropagation – Chain Rule

Suppose

$$z = wx + b$$

$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$



Use chain rule to compute the derivatives:

$$\frac{d\mathcal{L}}{dy} = \bar{y} = y - t$$

sometimes called the *error signal*

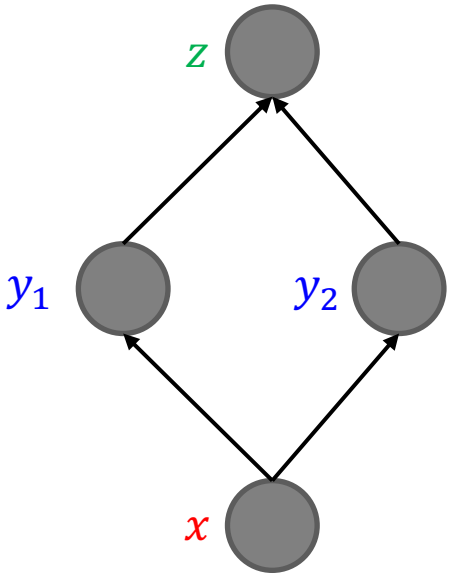
$$\frac{d\mathcal{L}}{dz} = \bar{z} = \frac{d\mathcal{L}}{dy}\frac{dy}{dz} = (y - t)\sigma'(z) = \bar{y}\sigma'(z)$$

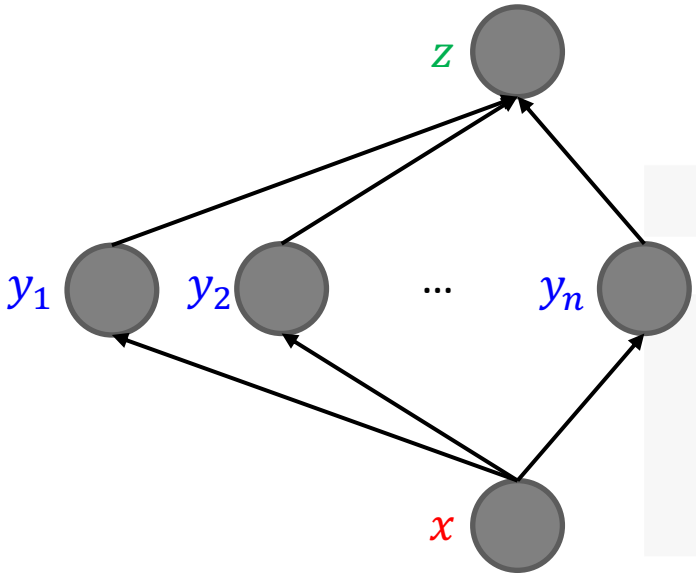
$$\frac{\partial\mathcal{L}}{\partial w} = \frac{d\mathcal{L}}{dy}\frac{dy}{dz}\frac{\partial z}{\partial w} = (y - t)\sigma'(z)x = \bar{z}x$$

$$\frac{\partial\mathcal{L}}{\partial b} = \frac{d\mathcal{L}}{dy}\frac{dy}{dz}\frac{\partial z}{\partial b} = (y - t)\sigma'(z) = \bar{z}$$

Backpropagation – Chain Rule

Multiple Paths / Multivariate Chain Rule:





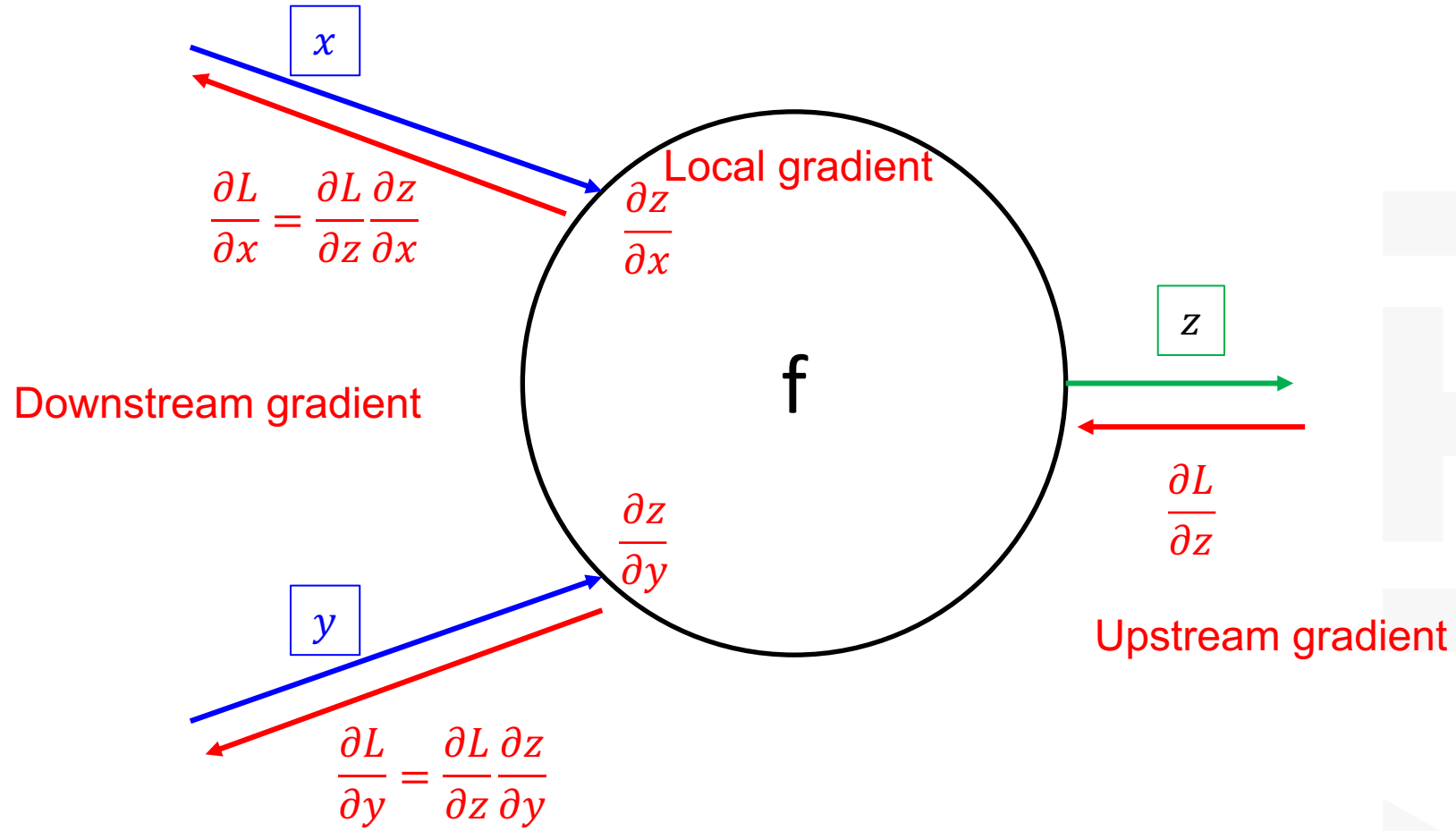
$z(y_1, y_2), y_1(x), y_2(x)$

$z(y_1, y_2, \dots), y_1(x), y_2(x), \dots$

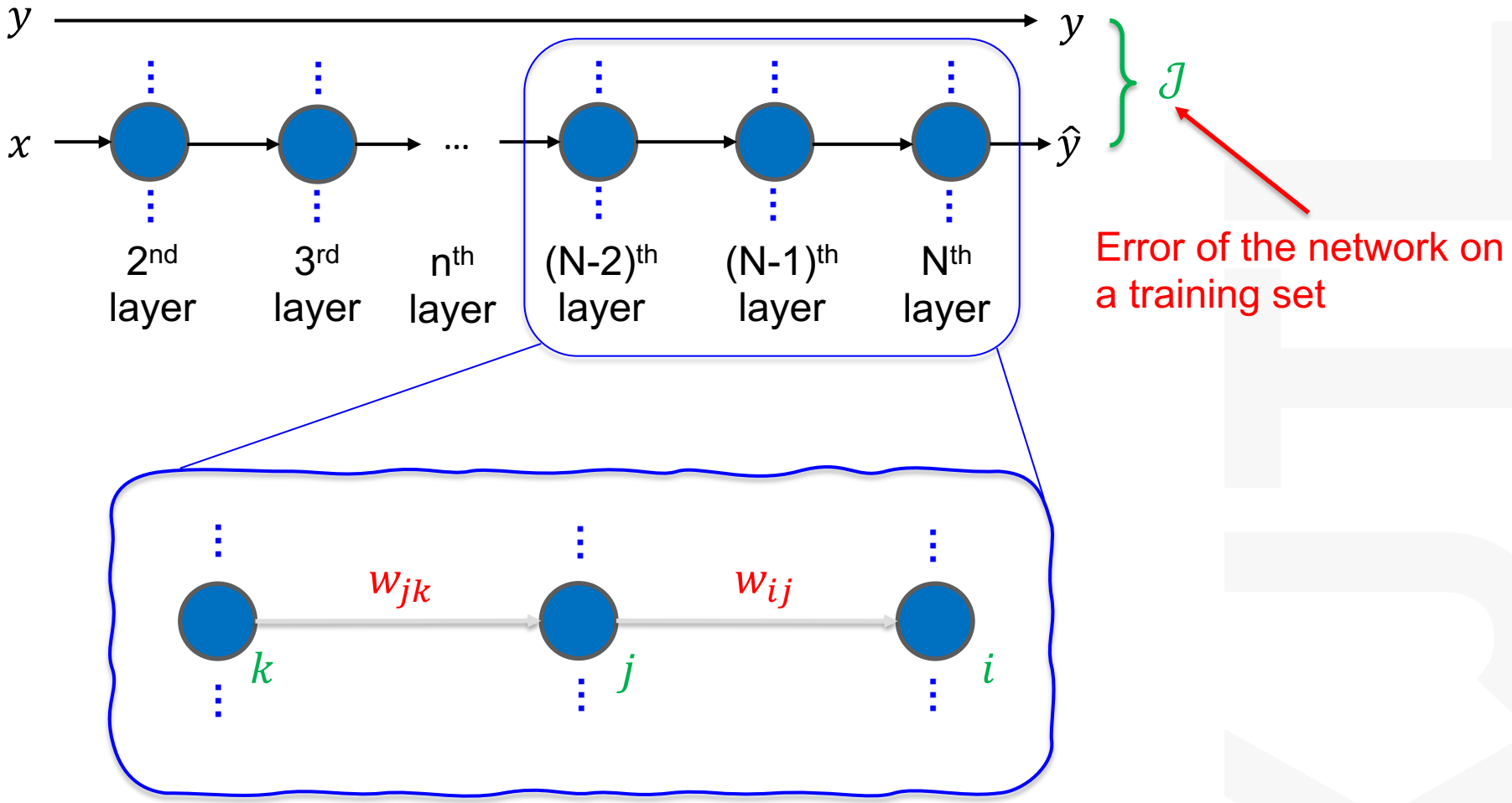
$$\frac{dz}{dx} = \frac{\partial z}{\partial y_1}\frac{dy_1}{dx} + \frac{\partial z}{\partial y_2}\frac{dy_2}{dx}$$

$$\frac{dz}{dx} = \sum_j \frac{\partial z}{\partial y_j}\frac{dy_j}{dx}$$

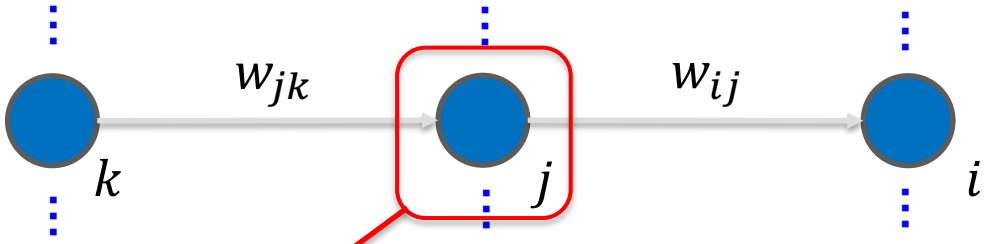
Backpropagation – Chain Rule

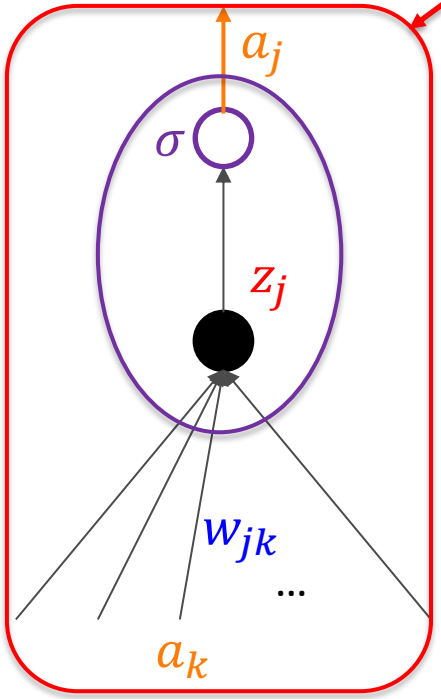


Backpropagation – Notations



Backpropagation – Notations





w_{jk} Weight of node k in layer $N-2$ to node j in layer $N-1$.

z_j Weighted summation of node j in layer $N-1$, the input to the activation function.

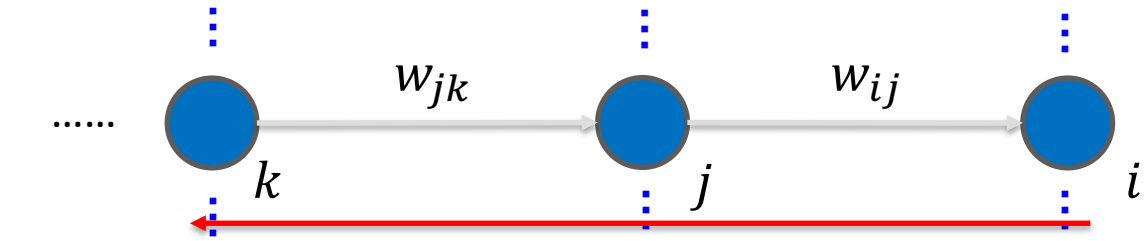
a_j Derived from the activation function σ.

$$z_j = \sum w_{jk} a_k$$

$$a_j = \sigma(z_j) = \sigma\left(\sum w_{jk} a_k\right)$$

Backpropagation

Start Backpropagation !



.....

$$\frac{\partial \mathcal{J}}{\partial w_{jk}} \qquad \frac{\partial \mathcal{J}}{\partial w_{ij}}$$

$$\Delta w_{jk} = -\alpha \frac{\partial \mathcal{J}}{\partial w_{jk}} \qquad \Delta w_{ij} = -\alpha \frac{\partial \mathcal{J}}{\partial w_{ij}}$$

Backpropagation

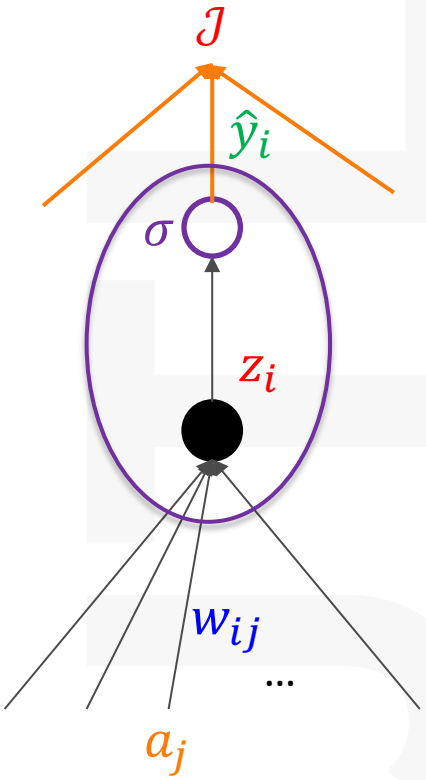
$$\frac{\partial \mathcal{J}}{\partial w_{ij}} = \frac{\partial \mathcal{J}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_i} \frac{\partial z_i}{\partial w_{ij}}$$

$$= \underbrace{\frac{\partial \mathcal{J}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_i}} a_j$$

Sensitivity

$$z_i = \sum w_{ij} a_j$$

Sensitivity of $\mathcal{J}$ to the net activation of i



Backpropagation

$$\frac{\partial \mathcal{J}}{\partial w_{ij}} = \frac{\partial \mathcal{J}}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial z_i} a_j$$

$$\delta_i = \frac{\partial \mathcal{J}}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial z_i}$$ Sensitivity

$$\frac{\partial \mathcal{J}}{\partial w_{ij}} = \delta_i a_j$$

$$\delta_i = \frac{\partial \mathcal{J}}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial z_i} = \frac{\partial \mathcal{J}}{\partial \hat{y}_i}\sigma'(z_i)$$

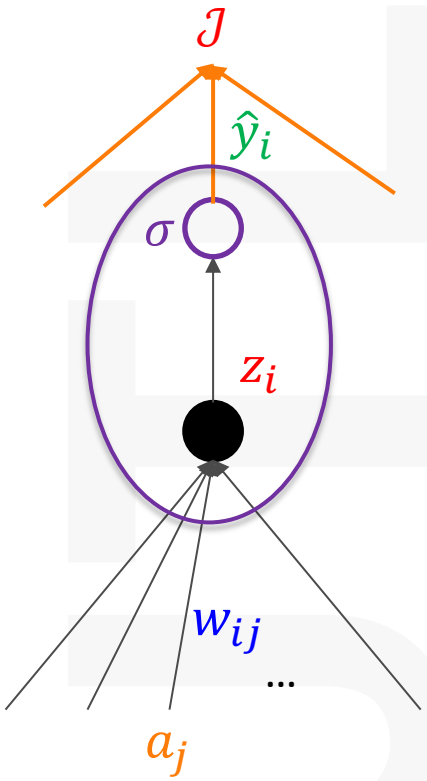
$$y_i = \sigma(z_i)$$

The derivative of $\sigma(z_i)$

$$\frac{\partial \mathcal{J}}{\partial \hat{y}_i} = \frac{-y_i}{\hat{y}_i} + \frac{1-y_i}{1-\hat{y}_i}$$

If we use binary cross entropy loss:

$$\mathcal{J} = -\sum (y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i))$$



https://towardsdatascience.com/deriving-backpropagation-with-cross-entropy-loss-d24811edeaf9

Backpropagation

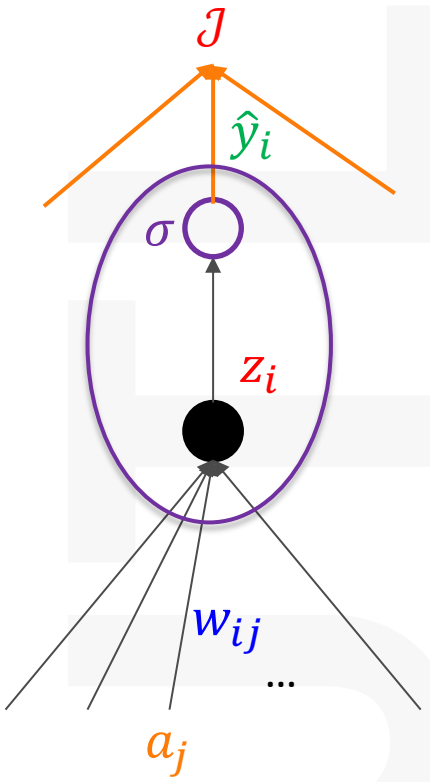
Hence,

$$\Delta w_{ij} = -\alpha \frac{\partial \mathcal{J}}{\partial w_{ij}}$$

$$= -\alpha \frac{\partial \mathcal{J}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_i} \frac{\partial z_i}{\partial w_{ij}}$$

$$= -\alpha \delta_i a_j$$

$$= -\alpha \frac{\partial \mathcal{J}}{\partial \hat{y}_i} \sigma'(z_i) a_j$$



Above is the update for the output layer.

Backpropagation

For the $N-1$ layer,

$$\frac{\partial \mathcal{J}}{\partial w_{jk}} = \sum \frac{\partial \mathcal{J}}{\partial a_j} \frac{\partial a_j}{\partial z_j} \frac{\partial z_j}{\partial w_{jk}}$$

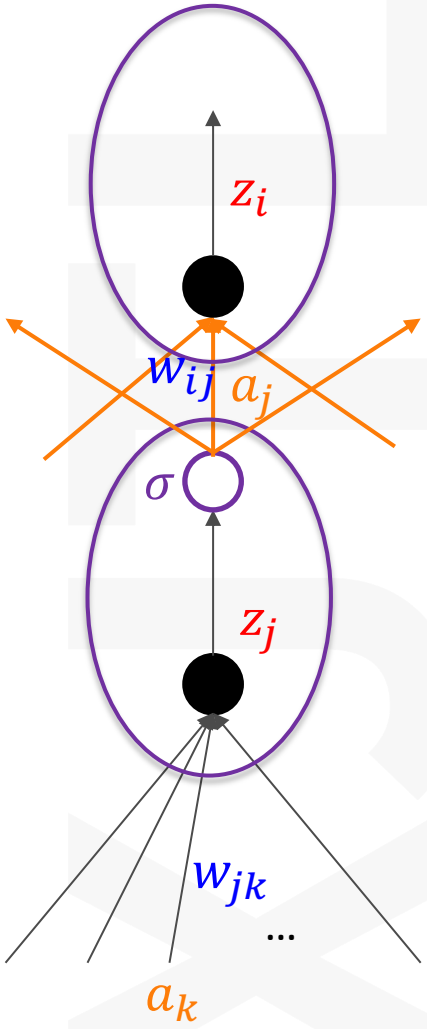
Note that $\frac{\partial \mathcal{J}}{\partial a_j}$ involves all output units

$$\frac{\partial \mathcal{J}}{\partial a_j} = \frac{\partial \mathcal{J}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_i} \frac{\partial z_i}{\partial a_j}$$

$$= \delta_i \frac{\partial z_i}{\partial a_j}$$

$$= \delta_i w_{ij}$$

$$z_i = \sum w_{ij} a_j$$



Backpropagation

$$\frac{\partial \mathcal{J}}{\partial w_{jk}} = \sum \frac{\partial \mathcal{J}}{\partial a_j} \frac{\partial a_j}{\partial z_j} \frac{\partial z_j}{\partial w_{jk}}$$

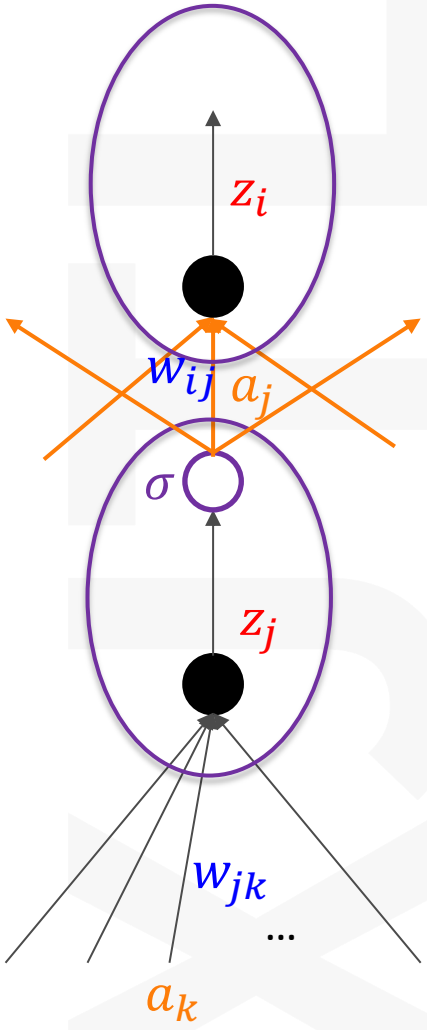
Sensitivity

$$= \sum \delta_i w_{ij} \frac{\partial a_j}{\partial z_j} \frac{\partial z_j}{\partial w_{jk}}$$

We can define δ_j as the hidden unit sensitivity:

$$\delta_j = \frac{\partial \mathcal{J}}{\partial a_j} \frac{\partial a_j}{\partial z_j}$$

The output sentivities are propagated back to the hidden unit sensitivies. Hence ‘Backpropagation of errors’.



Backpropagation

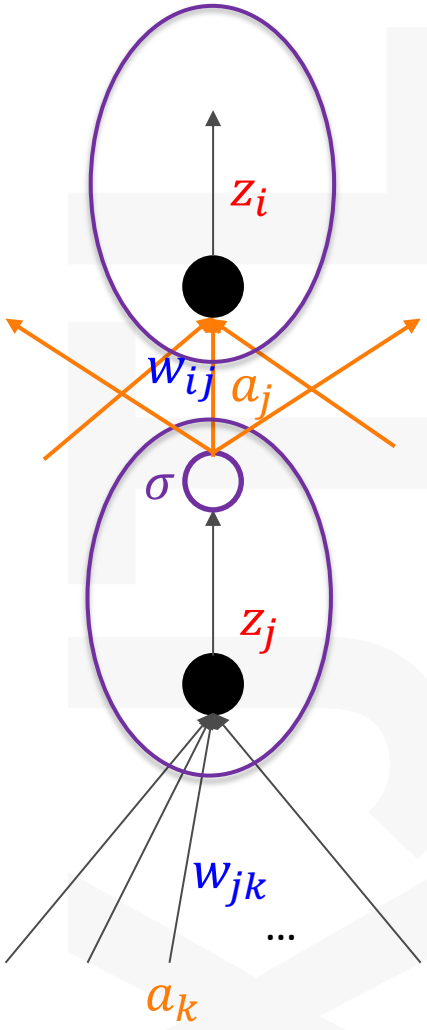
$$\frac{\partial \mathcal{J}}{\partial w_{jk}} = \sum \delta_i w_{ij} \frac{\partial a_j}{\partial z_j} \frac{\partial z_j}{\partial w_{jk}}$$

$$= \sum \delta_i w_{ij} \sigma'(z_j) \frac{\partial z_j}{\partial w_{jk}} \qquad a_j = \sigma(z_j)$$

$$= \sum \delta_i w_{ij} \sigma'(z_j) a_k \qquad z_j = \sum w_{jk} a_k$$

So,

$$\Delta w_{jk} = -\alpha \frac{\partial \mathcal{J}}{\partial w_{jk}} = -\alpha \sum \delta_i w_{ij} \sigma'(z_j) a_k$$



Backpropagation

- “Backpropagation” is the process of using the chain rule of differentiation in order to find the derivative of the loss with respect to each of the learnable weights and biases of the network.
- Training is done using gradient descent.
- The backpropagation algorithm can be easily generalized to any network with feed-forward connections, e.g. more layers, different nonlinearities in different layers, etc.

Exercise:

https://peterroelants.github.io/posts/neural-network-implementation-part01/