Deep Learning

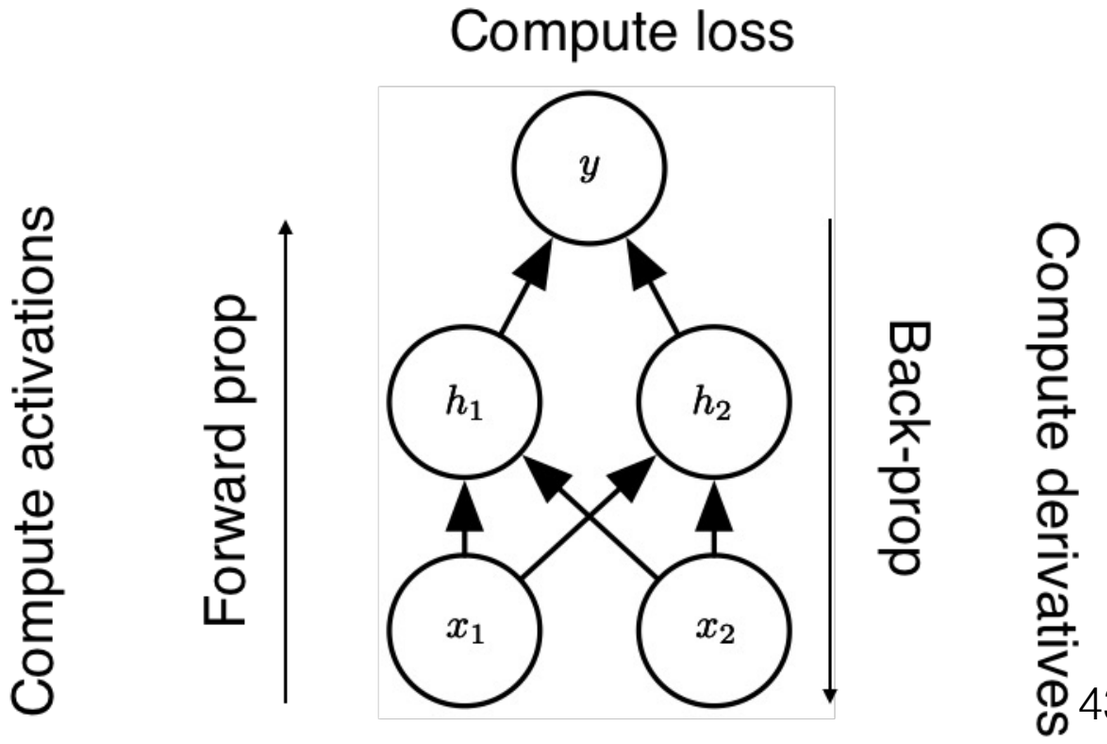
LM Computer Science, Data Science, Cybersecurity

2nd semester - 6 CFU

Luca Pasa, Nicolò Navarin & Alessandro Sperduti

Training NNs: back-propagation

- **Forward** propagation: the input is propagated through the network and produces the network output $\hat{y}$ and the cost $J(\boldsymbol{\theta})$
- **Back** propagation: information from the cost flows backward to compute the gradient for each parameter
  - It is the method for **computing the gradient**.
  - Another algorithm performs learning using gradient (e.g. stochastic gradient descent)



Back-Propagation

- Back-propagation is “just the chain rule” of calculus

$$x, y \in \mathbb{R}, f, g: \mathbb{R} \to \mathbb{R}, y = g(x), z = f\big(g(x)\big) = f(y)$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} = f'\big(g(x)\big)g'(x)$$

$$\boldsymbol{x} \in \mathbb{R}^m, \boldsymbol{y} \in \mathbb{R}^n, g: \mathbb{R}^m \to \mathbb{R}^n, f: \mathbb{R}^n \to \mathbb{R} \quad , \boldsymbol{y} = g(\boldsymbol{x}), z = f(\boldsymbol{y})$$

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i} \qquad \nabla_{\boldsymbol{x}} \mathrm{z} = \nabla_{\boldsymbol{y}} \mathrm{z} \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)$$

$$\nabla_{\boldsymbol{x}} z = \left[\frac{\partial}{\partial x_1} z, \dots, \frac{\partial}{\partial x_m} z\right]$$

Jacobian

- But it’s a particular implementation of the chain rule
  - Uses dynamic programming (table filling)
  - Avoids recomputing repeated subexpressions
  - Speed vs memory tradeoff

Warning: the DL book uses column notation for the gradient, you will find transposes and a different order

Let's recall the matrix notation

$$\nabla_x z = \left[\frac{\partial}{\partial x_1} z, \dots, \frac{\partial}{\partial x_m} z\right]$$

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x_i} + \cdots + \frac{\partial z}{\partial y_n}\frac{\partial y_n}{\partial x_i}$$

$$\nabla_x z = \nabla_y z \left(\frac{\partial y}{\partial x}\right) =$$

$\frac{\partial}{\partial y_1} z$	$\cdots$	$\frac{\partial}{\partial y_n} z$

$\times$

$\frac{\partial}{\partial x_1} y_1$	$\cdots$	$\frac{\partial}{\partial x_m} y_1$
$\cdots$		
$\frac{\partial}{\partial x_1} y_n$		$\frac{\partial}{\partial x_m} y_n$

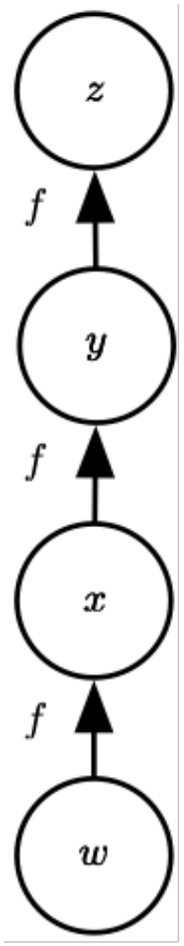
$=$

$\frac{\partial}{\partial x_1} z \frac{\partial}{\partial x_1} y_1 + \cdots + \frac{\partial}{\partial x_m} z \frac{\partial}{\partial x_1} y_n$
$\cdots$
$\frac{\partial}{\partial x_1} z \frac{\partial}{\partial x_m} y_1 + \cdots + \frac{\partial}{\partial x_m} z \frac{\partial}{\partial x_m} y_n$

$\frac{\partial}{\partial x_1} z$	$\cdots$	$\frac{\partial}{\partial x_m} z$

Back-Propagation

- Simple example. $w \in \mathbb{R}$, <u>same function at each step of the chain</u>



$$\frac{\partial z}{\partial w} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w}$$
$$= f'(y)f'(x)f'(w)$$
$$= f'(f(f(w)))f'(f(w))f'(w)$$

Back-prop avoids computing this twice

Back-propagation for (sigmoid) perceptron

- Similar to gradient descent for linear regression
- Training set $Tr = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(N_{Tr})}, y^{(N_{Tr})})\}$
- MSE error function $E = \frac{1}{2}\frac{1}{N_{Tr}}\sum_{d \in Tr}\left(t^{(d)} - out^{(d)}\right)^2$
- Let us derive it..

EX: Gradient Computation for Sigmoidal Units

Let consider a Perceptron with sigmoidal function:

$$out(\vec{x}) = \sigma(\sum_{i=0}^{n} w_i x_i) = \sigma(\vec{w} \cdot \vec{x})$$

where we recall that $\sigma(net) = \frac{1}{1+e^{-net}}$

Notice that for $\sigma()$ the following relation holds

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

and recall that (derivative of composition of functions)

$$\frac{d\ f(g(x))}{d\ x} = \frac{d\ f(g(x))}{d\ g(x)} \frac{d\ g(x)}{d\ x}$$

Gradient Computation for Sigmoidal Units

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2N_{Tr}}\sum_{d\in Tr}(t^{(d)} - out^{(d)})^2$$

Consider a single output neuron

$$= \frac{1}{2N_{Tr}}\sum_{d\in Tr}\frac{\partial}{\partial w_i}(t^{(d)} - out^{(d)})^2$$

Derivative of power

$$= \frac{1}{2N_{Tr}}\sum_{d\in Tr}2(t^{(d)} - out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)} - out^{(d)})$$

$$= \frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)} - out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)} - \sigma(\vec{w}\cdot\vec{x}^{(d)}))$$

Chain rule

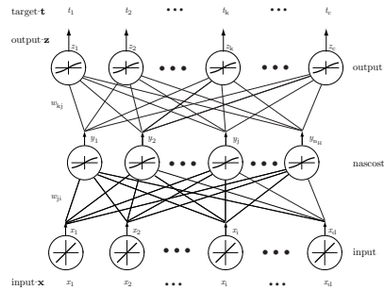
$$\frac{\partial E}{\partial w_i} = \frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)} - out^{(d)})(-\frac{\partial\sigma(\vec{w}\cdot\vec{x}^{(d)})}{\partial\vec{w}\cdot\vec{x}^{(d)}}\frac{\partial\vec{w}\cdot\vec{x}^{(d)}}{\partial w_i})$$

$$= -\frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)} - out^{(d)})\sigma(\vec{w}\cdot\vec{x}^{(d)})(1-\sigma(\vec{w}\cdot\vec{x}^{(d)}))x_i^{(d)}$$

Gradient Descent for Feed-forward Networks

- Notebook: Backpropagation

Gradient Descent for Feed-forward Networks



- d input units, size input $\vec{x} \equiv (x_1, \ldots, x_d)$
  ($d+1$ if the threshold is included into the weight vector $\vec{x}' \equiv (x_0, x_1, \ldots, x_d)$)
- N_H hidden units (with output $\vec{y} \equiv (y_1, \ldots, y_{N_H})$)
- c output units, output size $\vec{z} \equiv (z_1, \ldots, z_c)$
- c, size target vectors $\vec{t} \equiv (t_1, \ldots, t_c)$
- w_{ji} weight on the connection from input unit i to hidden unit j
- w_{kj} weight on the connection from hidden unit j to output unit k

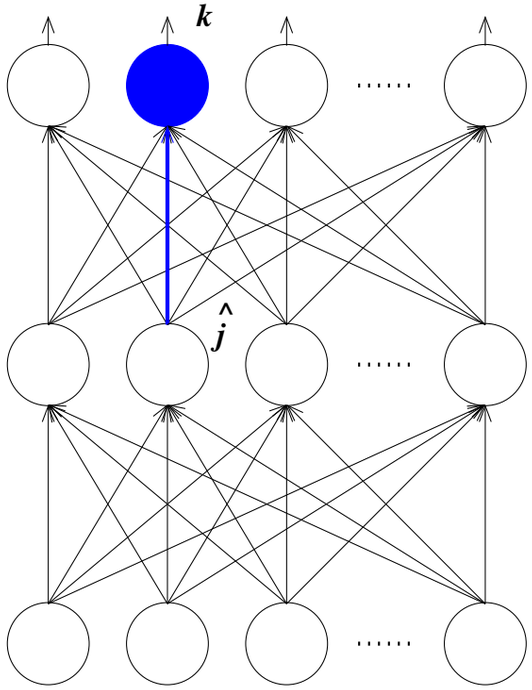
Function error, with c output units:

$$E[\vec{w}] = \frac{1}{2cN_{Tr}} \sum_{(\vec{x}^{(p)}, \vec{t}^{(p)}) \in Tr} \sum_{k=1}^{c} \left(t_k^{(p)} - z_k(\vec{x}^{(p)}) \right)^2$$

Gradient Computation for Multiple Output Units

Let fix $\hat{k}$ and $\hat{j}$:

$$\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} = \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

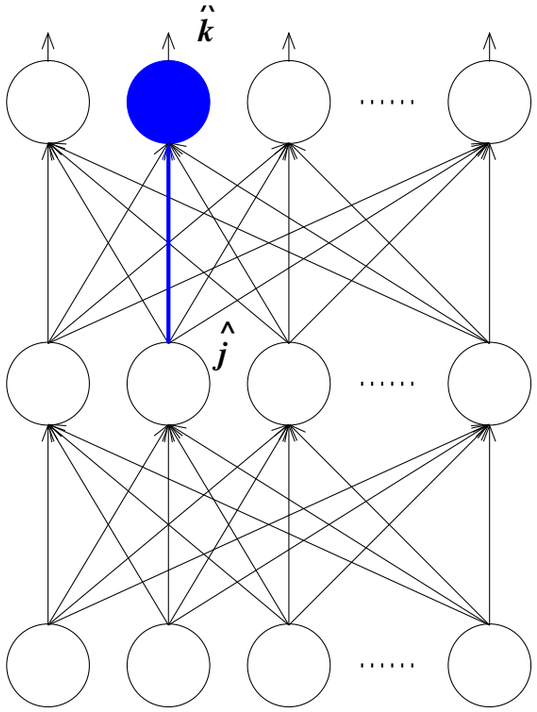


Gradient Computation for Output Units

Let fix $\hat{k}$ and $\hat{j}$:

$$\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} = \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

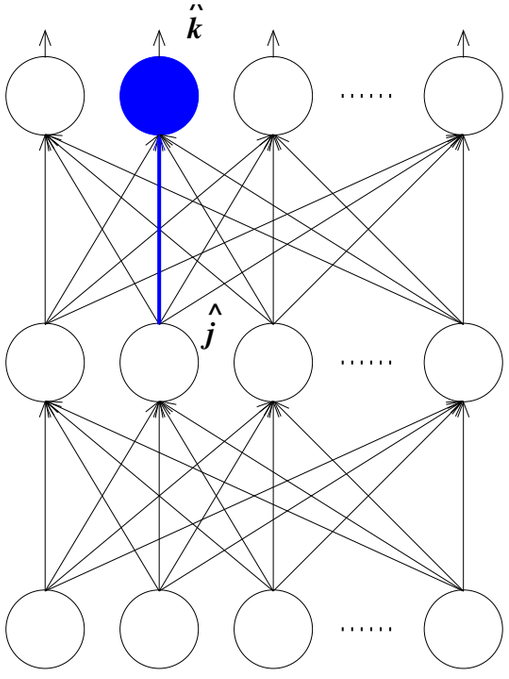


Gradient Computation for Output Units

Let fix $\hat{k}$ and $\hat{j}$:

$$\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} = \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$



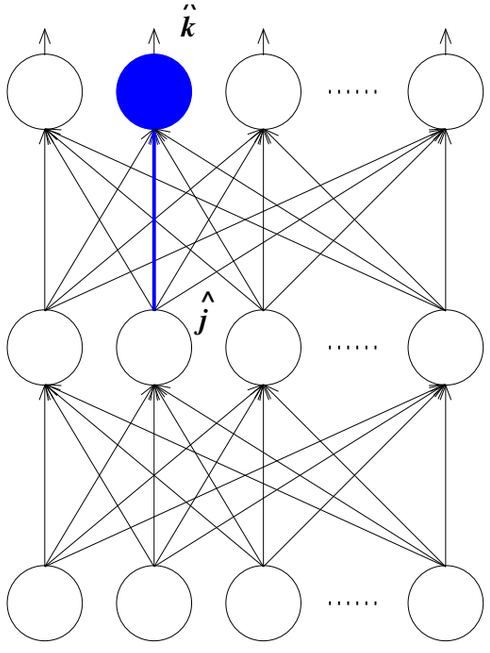
$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} 2(t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)})$$

Gradient Computation for Output Units

Let fix $\hat{k}$ and $\hat{j}$:

$$\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} = \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$



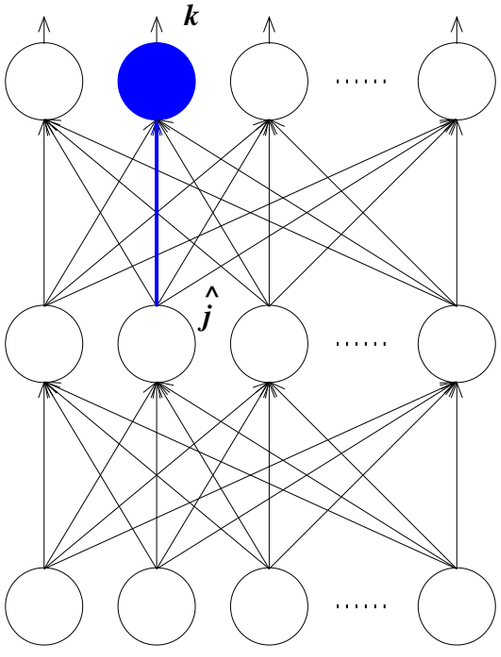
$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} 2(t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)})$$

$$= \frac{1}{cN_{Tr}} \sum_{p \in Tr} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - \sigma(\vec{w}_{\hat{k}} \cdot \vec{y}^{(p)}))$$

Gradient Computation for Output Units

Let fix $\hat{k}$ and $\hat{j}$:

$$\begin{aligned}
\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} &= \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2 \\
&= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2
\end{aligned}$$



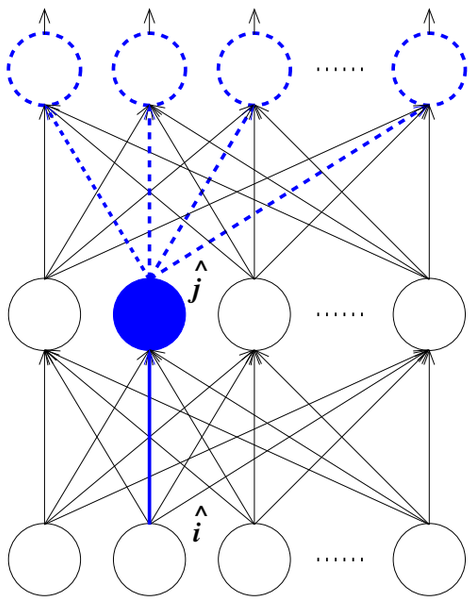
$$\begin{aligned}
&= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} 2(t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \\
&= \frac{1}{cN_{Tr}} \sum_{p \in Tr} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - \sigma(\vec{w}_{\hat{k}} \cdot \vec{y}^{(p)})) \\
&= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \sigma'(\vec{w}_{\hat{k}} \cdot \vec{y}^{(p)}) y_{\hat{j}}^{(p)}
\end{aligned}$$

Gradient Computation for Hidden Units

Let fix $\hat{j}$ and $\hat{i}$:

Output neurons

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

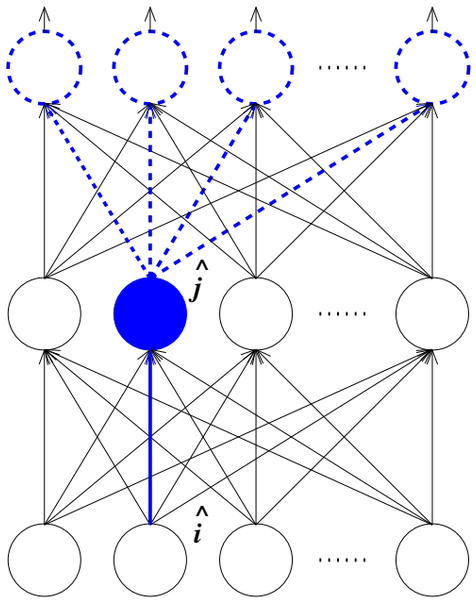


Gradient Computation for Hidden Units

Let fix $\hat{j}$ and $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2$$



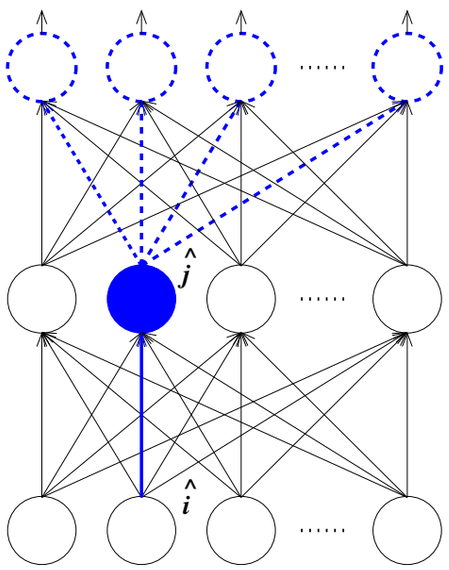
Gradient Computation for Hidden Units

Let fix $\hat{j}$ and $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)})$$

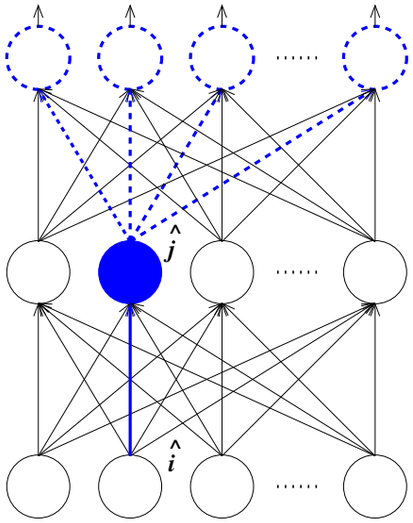


Gradient Computation for Hidden Units

Let fix $\hat{j}$ and $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2$$



$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)})$$

$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \vec{w}_k \cdot \vec{y}^{(p)}$$

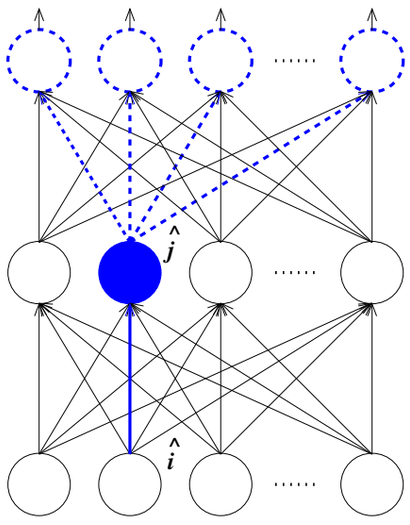
Same term as before

Gradient Computation for Hidden Units

Let fix $\hat{j}$ and $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2$$



$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)})$$

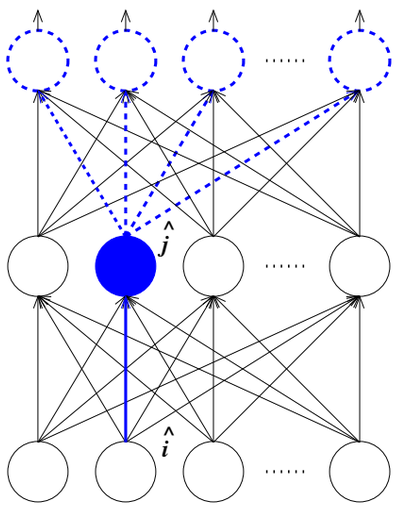
$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \sum_{j=1}^{N_H} w_{kj} y_j^{(p)}$$

Gradient Computation for Hidden Units

Let fix $\hat{j}$ and $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2$$



$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)})$$

$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \sum_{j=1}^{N_H} w_{kj} y_j^{(p)}$$

$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} y_{\hat{j}}^{(p)}$$

Predicted output for k-th neuron

$\sigma(\vec{w}_k \cdot \vec{y}^{(p)})$

Hidden representation

$\sigma(\vec{w}_j \cdot \vec{x}^{(p)})$

Gradient Computation for Hidden Units

Let fix $\hat{j}$ and $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)})$$

$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \sum_{j=1}^{N_H} w_{kj} y_j^{(p)}$$

$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} y_{\hat{j}}^{(p)}$$

Chain rule

$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \sigma'(\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)})$$

Gradient Computation for Hidden Units

Let fix $\hat{j}$ and $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)})$$

$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \sum_{j=1}^{N_H} w_{kj} y_j^{(p)}$$

$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} y_{\hat{j}}^{(p)}$$

$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \sigma'(\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)})$$

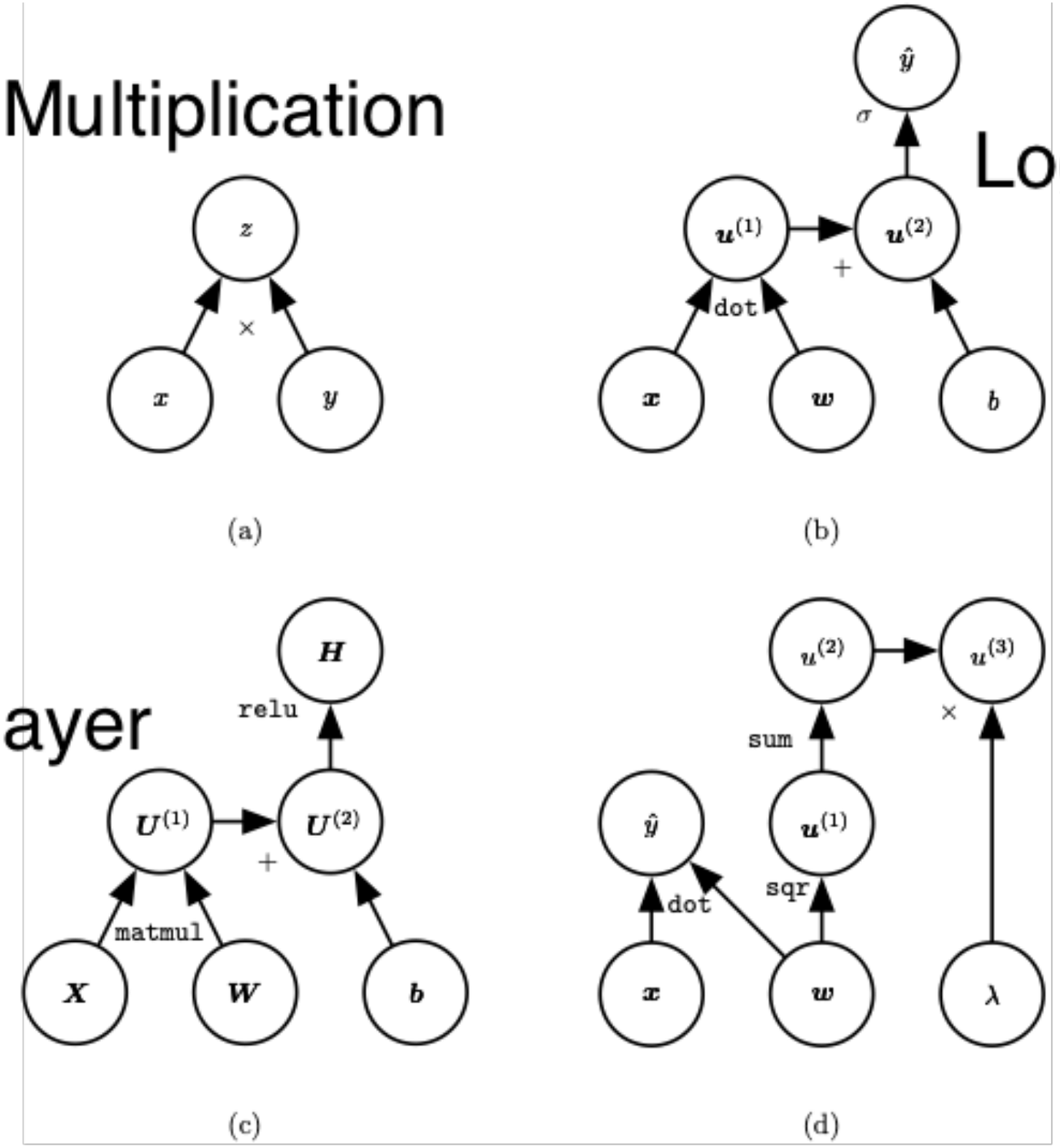
$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \sigma'(\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)}) x_{\hat{i}}^{(p)}$$

Computational Graph

- Each node is a variable
  - scalar
  - Vector
  - tensor..
- Edges encode operations
  - Simple functions of one or more variables
  - Returns just one output variable

Computational Graph

Multiplication



Logistic regression

ReLU layer

Linear regression and weight decay

Back-Propagation on tensors

- Think about tensors just as vectors
  - Flattening + reshape of the gradient
- Gradient w.r.t. a tensor $\mathbf{X}$ is $\nabla_{\mathbf{X}} z$
  - A 3-D tensor has 3 coordinates. We abstract it away using a single index i
- Chain rule for tensors, with $\mathbf{Y} = g(\mathbf{X})$ and $z = f(\boldsymbol{Y})$

$$\nabla_{\mathbf{X}} z = \sum_j \frac{\partial z}{\partial \boldsymbol{Y}_j} \nabla_{\mathbf{X}} \boldsymbol{Y}_j$$

Forward propagation algorithm

Algorithm 6.3 Forward propagation through a typical deep neural network and the computation of the cost function. The loss $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ depends on the output $\hat{\boldsymbol{y}}$ and on the target $\boldsymbol{y}$ (see section 6.2.1.1 for examples of loss functions). To obtain the total cost J, the loss may be added to a regularizer $\Omega(\theta)$, where θ contains all the parameters (weights and biases). Algorithm 6.4 shows how to compute gradients of J with respect to parameters $\boldsymbol{W}$ and $\boldsymbol{b}$. For simplicity, this demonstration uses only a single input example $\boldsymbol{x}$. Practical applications should use a minibatch. See section 6.5.7 for a more realistic demonstration.

Require: Network depth, l
Require: $\boldsymbol{W}^{(i)}, i \in \{1, \ldots, l\}$, the weight matrices of the model
Require: $\boldsymbol{b}^{(i)}, i \in \{1, \ldots, l\}$, the bias parameters of the model
Require: $\boldsymbol{x}$, the input to process
Require: $\boldsymbol{y}$, the target output

$\boldsymbol{h}^{(0)} = \boldsymbol{x}$
for $k = 1, \ldots, l$ **do**
  $\boldsymbol{a}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)} \boldsymbol{h}^{(k-1)}$
  $\boldsymbol{h}^{(k)} = f(\boldsymbol{a}^{(k)})$
end for
$\hat{\boldsymbol{y}} = \boldsymbol{h}^{(l)}$
$J = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \lambda \Omega(\theta)$

Back propagation algorithm

Algorithm 6.4 Backward computation for the deep neural network of algorithm 6.3, which uses in addition to the input $\boldsymbol{x}$ a target $\boldsymbol{y}$. This computation yields the gradients on the activations $\boldsymbol{a}^{(k)}$ for each layer k, starting from the output layer and going backwards to the first hidden layer. From these gradients, which can be interpreted as an indication of how each layer's output should change to reduce error, one can obtain the gradient on the parameters of each layer. The gradients on weights and biases can be immediately used as part of a stochastic gradient update (performing the update right after the gradients have been computed) or used with other gradient-based optimization methods.

After the forward computation, compute the gradient on the output layer:

$\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$

for $k = l, l-1, \ldots, 1$ **do**

Convert the gradient on the layer's output into a gradient into the pre-nonlinearity activation (element-wise multiplication if f is element-wise):

$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = \boldsymbol{g} \odot f'(\boldsymbol{a}^{(k)})$

Compute gradients on weights and biases (including the regularization term, where needed):

$\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g} + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega(\theta)$

$\nabla_{\boldsymbol{W}^{(k)}} J = \boldsymbol{g}\, \boldsymbol{h}^{(k-1)\top} + \lambda \nabla_{\boldsymbol{W}^{(k)}} \Omega(\theta)$

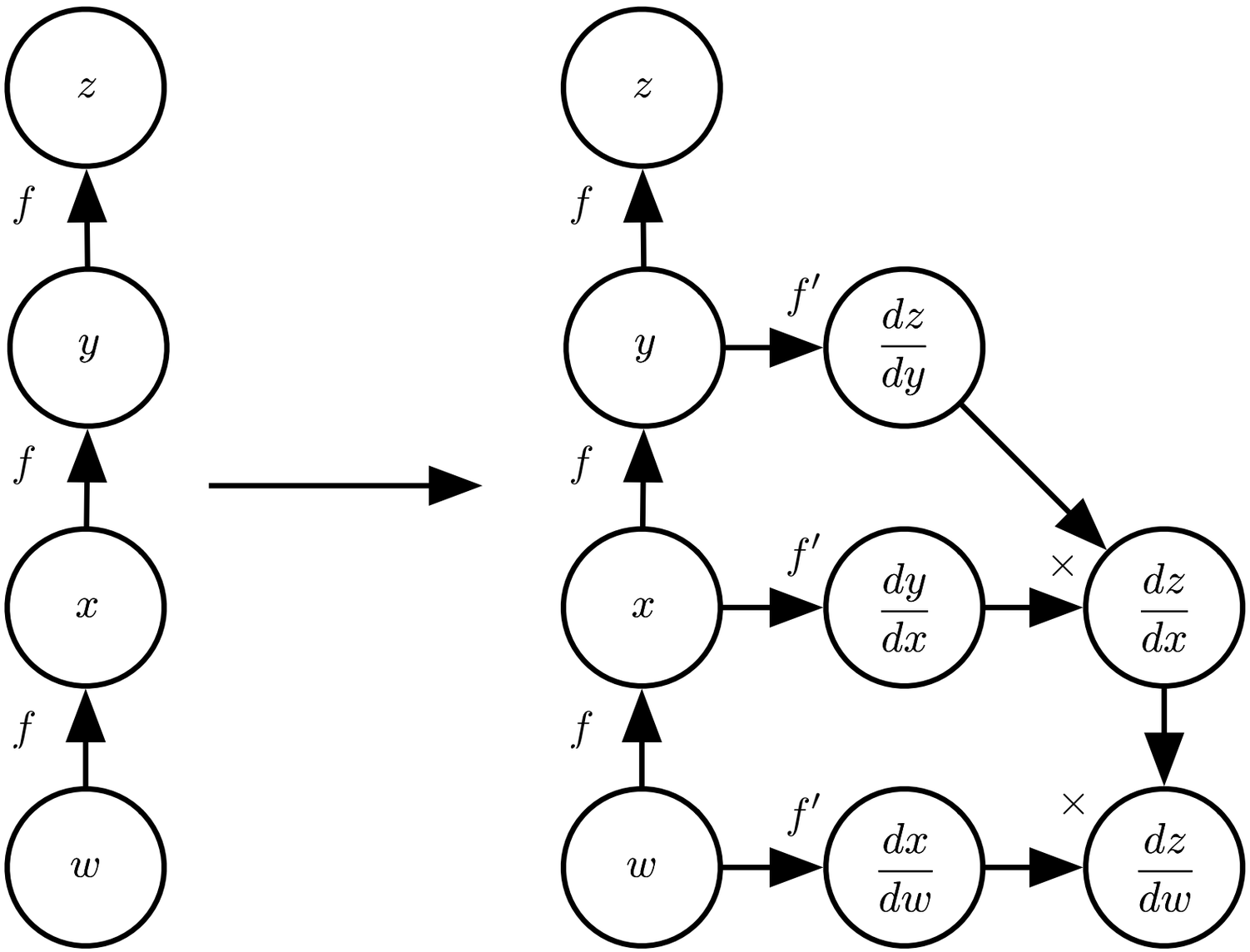
Propagate the gradients w.r.t. the next lower-level hidden layer's activations:

$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)\top} \boldsymbol{g}$

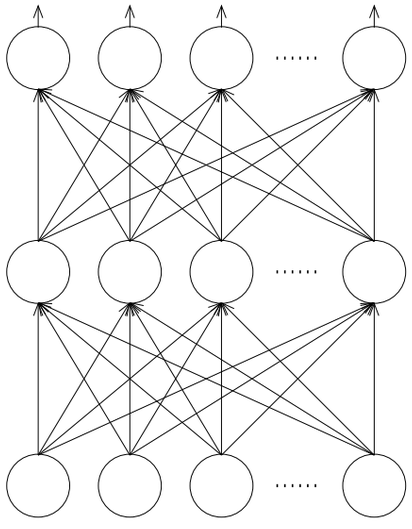
end for

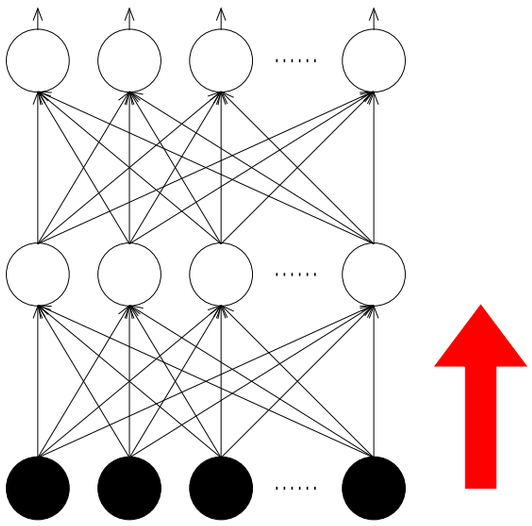
Autodiff

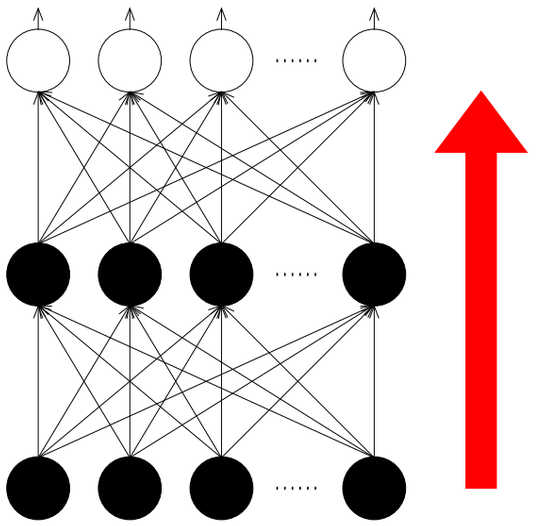
- Backpropagation described in the computational graph (TensorFlow)

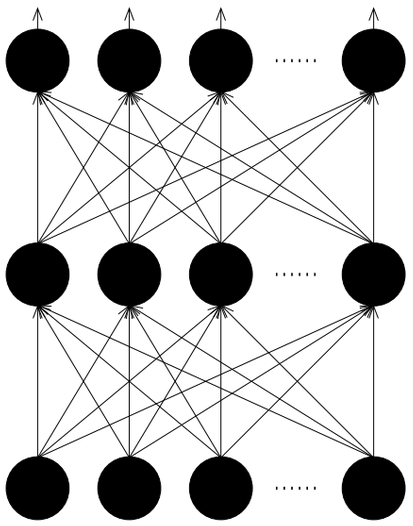


Forward Phase

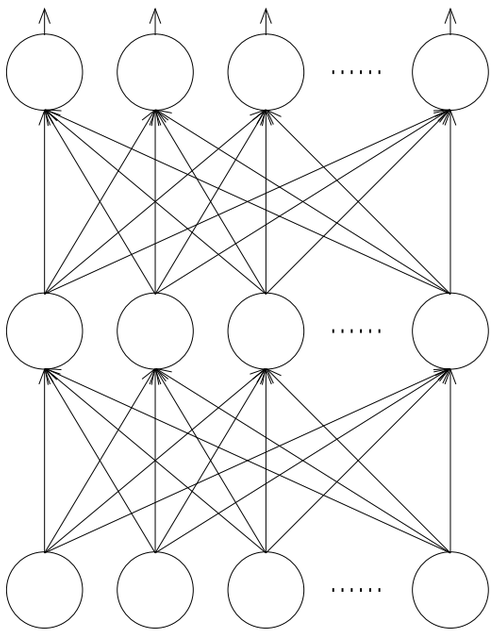


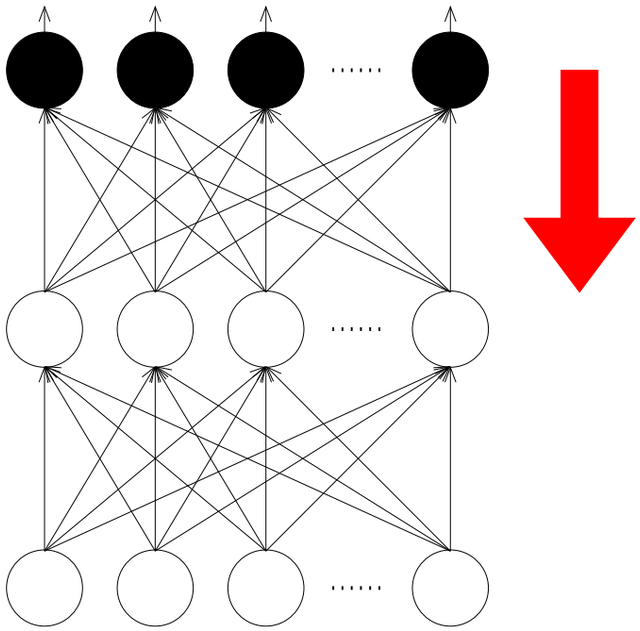


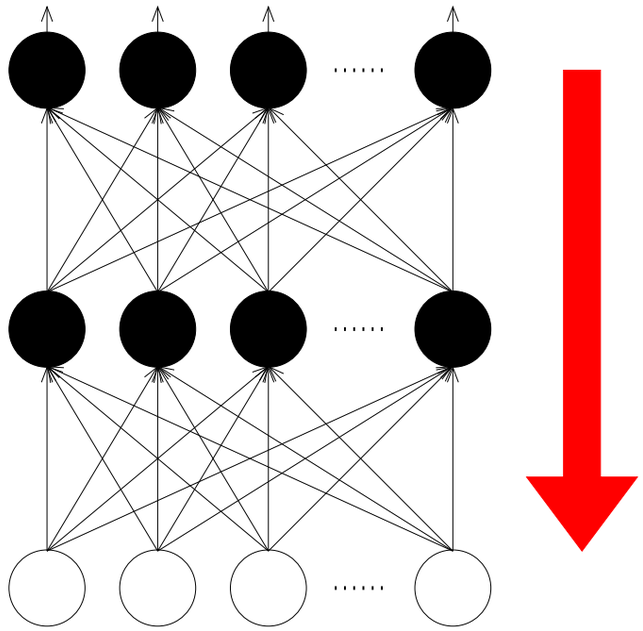




Backward Phase







END