# CS 224D: DEEP LEARNING FOR NLP MIDTERM REVIEW

Neural Networks: Terminology, Forward Pass, Backpropagation.

Rohit Mundra

May 4, 2015

# Overview

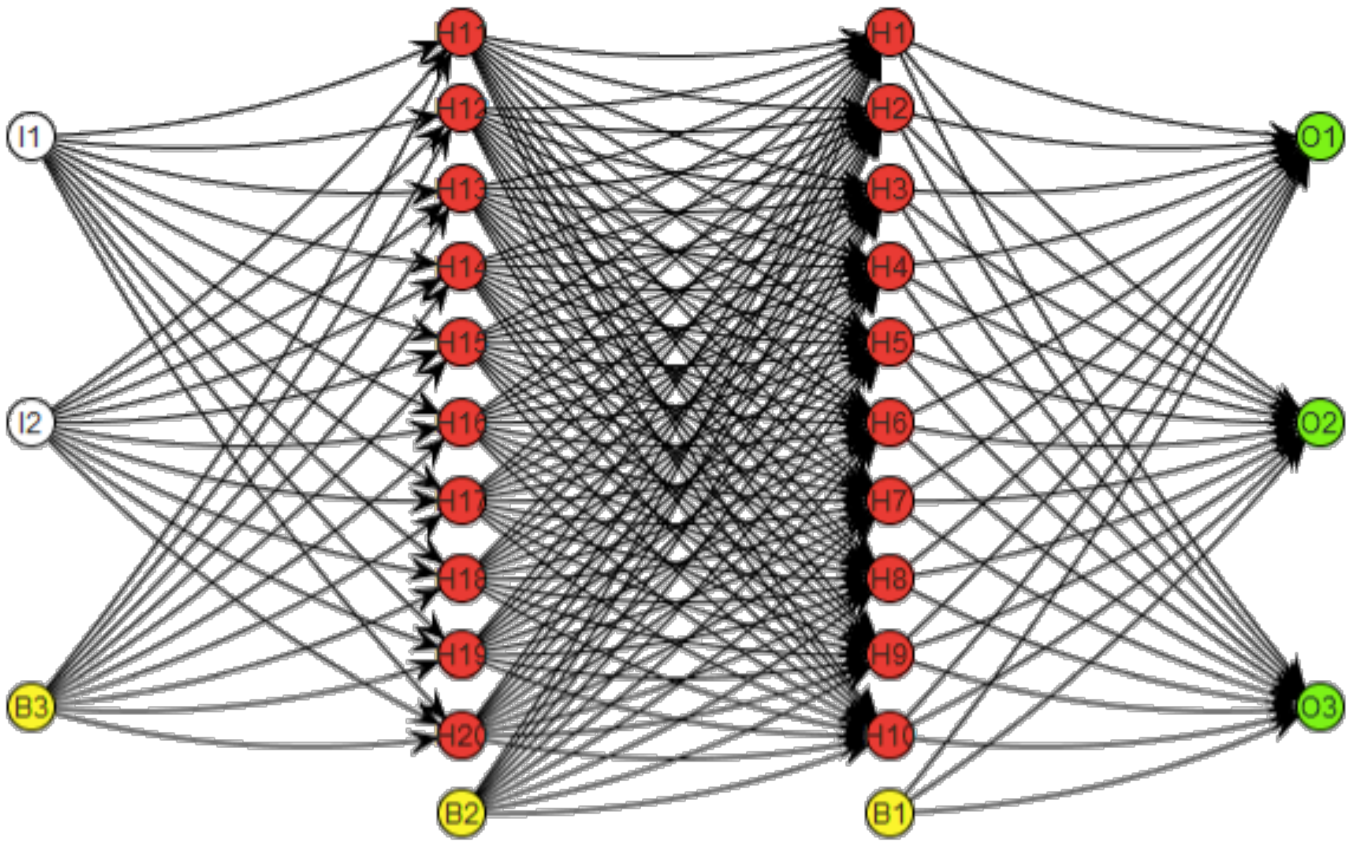
- Neural Network Example
- Terminology
- Example 1:
  - Forward Pass
  - Backpropagation Using Chain Rule
  - **What is delta? From Chain Rule to Modular Error Flow**
- Example 2:
  - Forward Pass
  - Backpropagation

# Neural Networks

- One of many different types of non-linear classifiers (i.e. leads to non-linear decision boundaries)

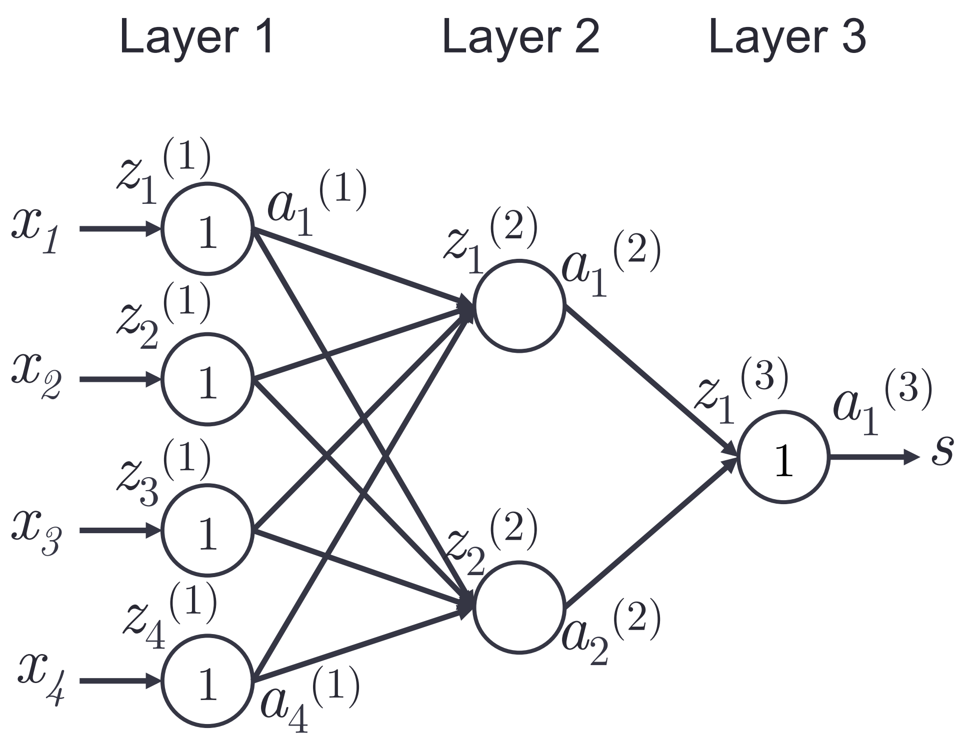
- Most common design involves the stacking of affine transformations followed by point-wise (element-wise) non-linearity

# An example of a neural network



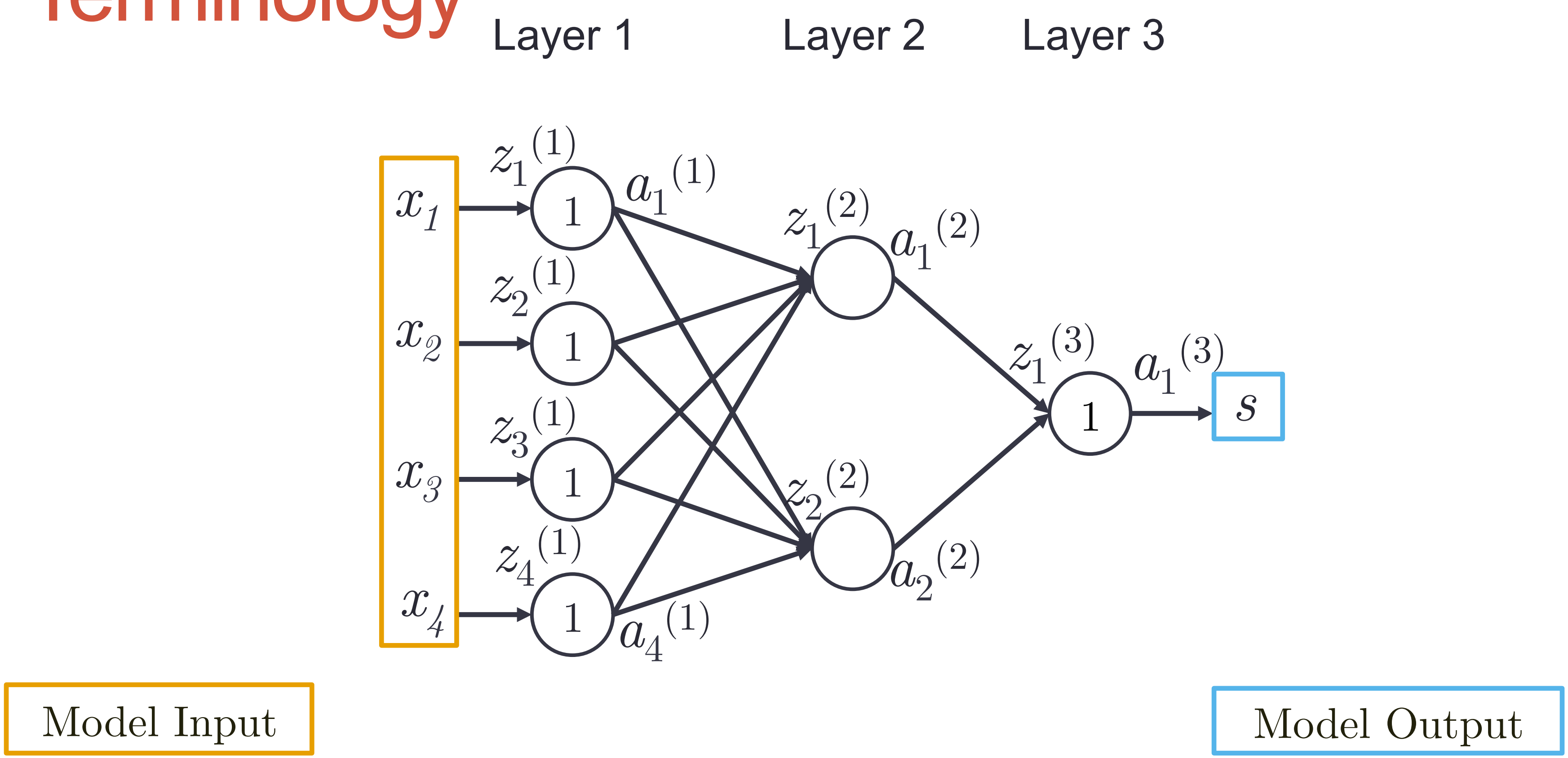

- This is a 4 layer neural network.
- 2 hidden-layer neural network.
- 2-10-10-3 neural network (complete architecture defn.)

# Our first example

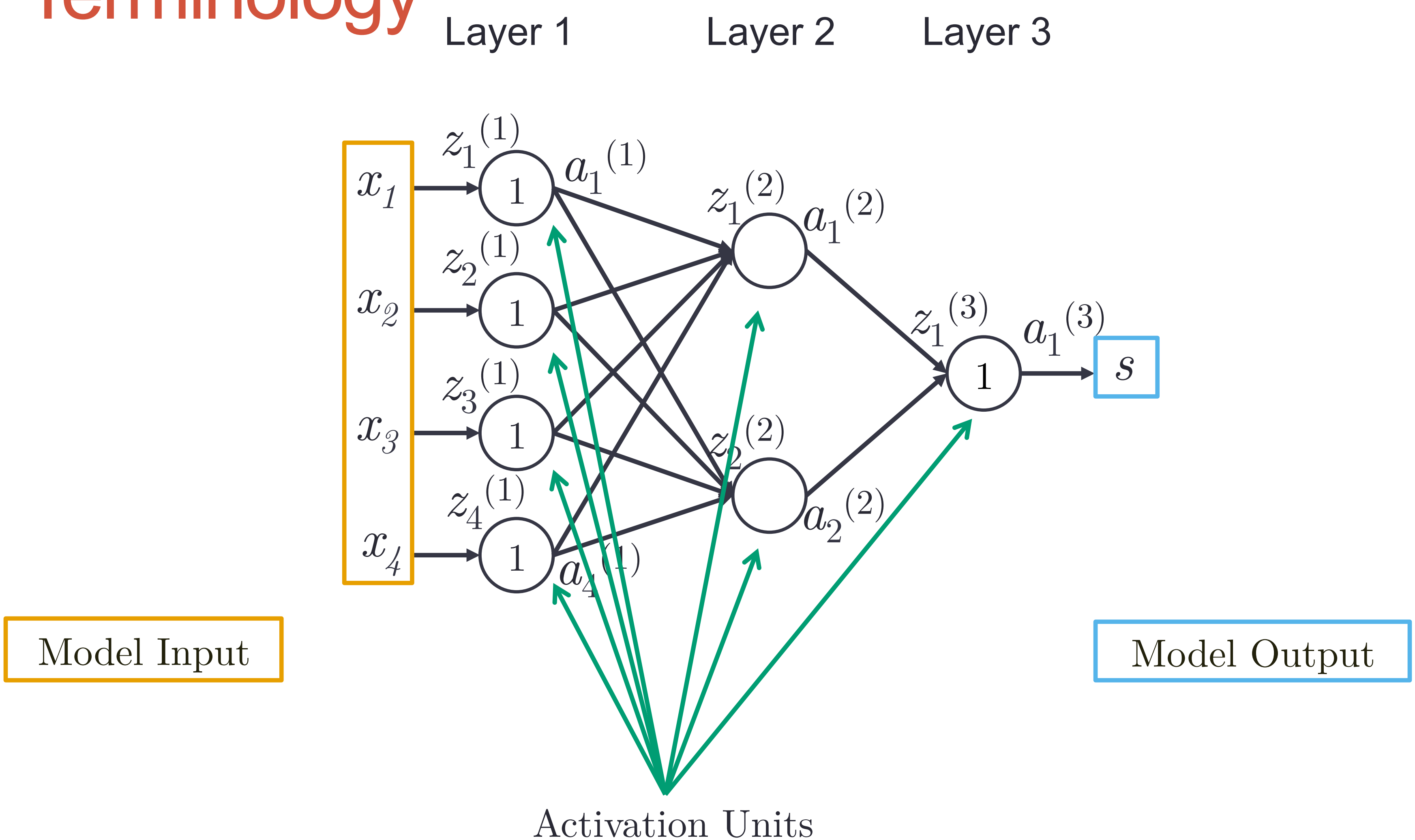



- This is a 3 layer neural network
- 1 hidden-layer neural network

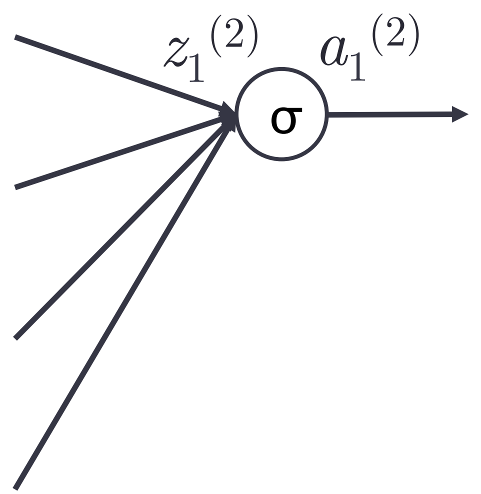
# Our first example: Terminology



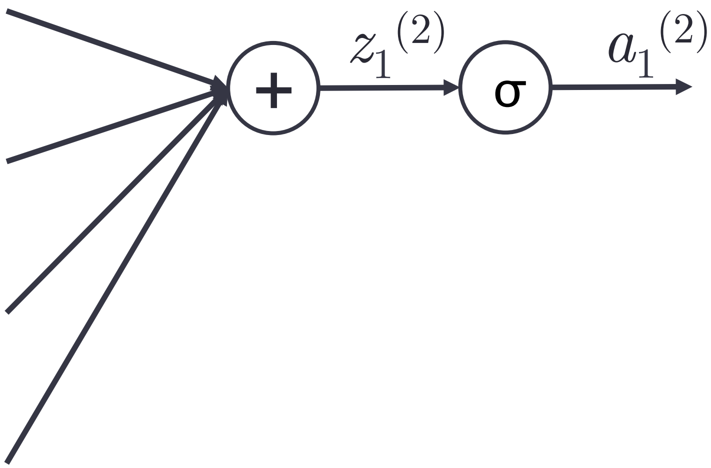
# Our first example: Terminology



# Our first example: Activation Unit Terminology




We draw this



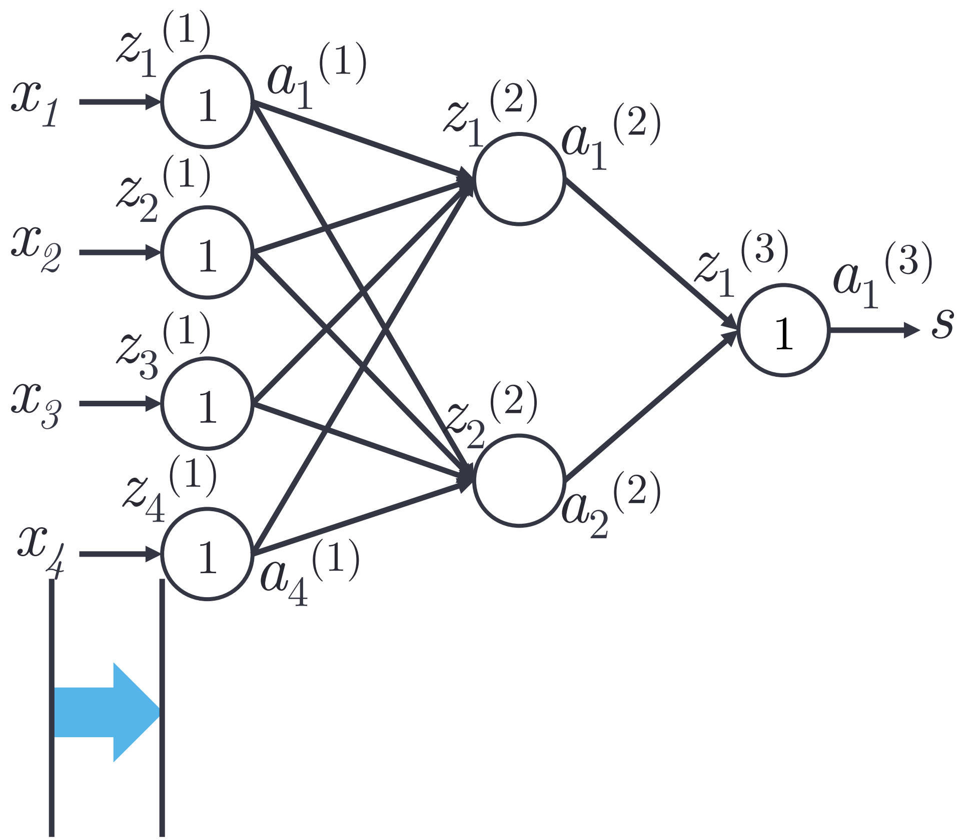

This is actually what's going on

$$z_1^{(2)} = W_{11}^{(1)} a_1^{(1)} + W_{12}^{(1)} a_2^{(1)} + W_{13}^{(1)} a_3^{(1)} + W_{14}^{(1)} a_4^{(1)}$$

$a_1^{(2)}$ is the $1^{\text{st}}$ activation unit of layer 2

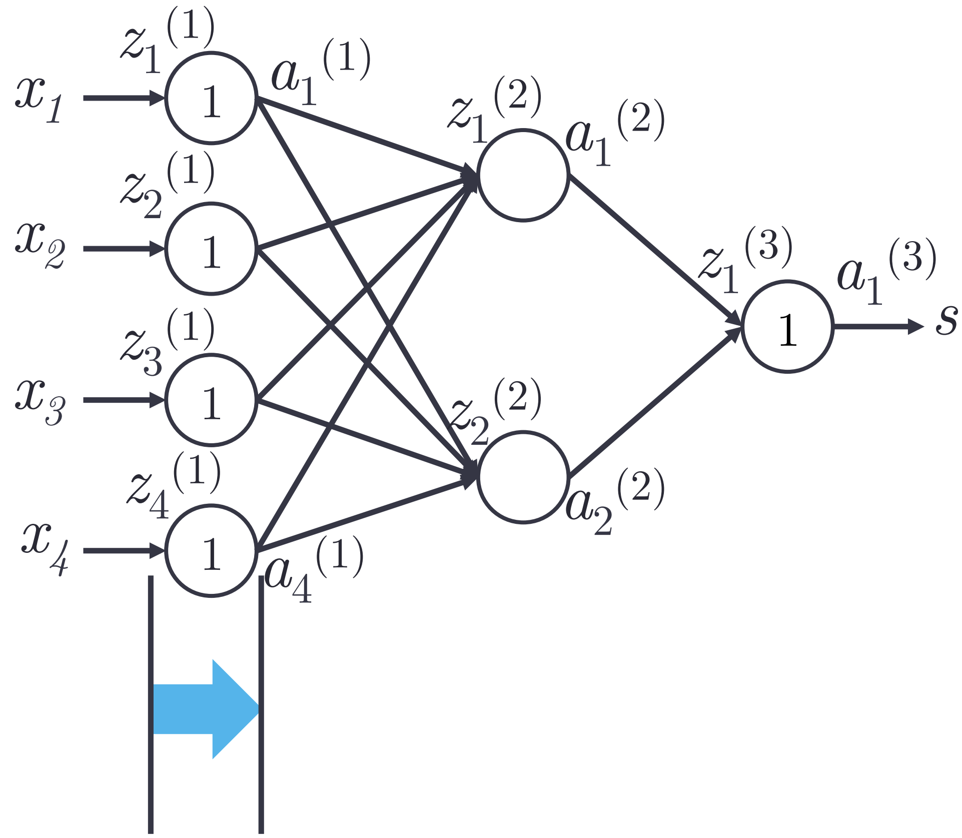
$$a_1^{(2)} = \sigma(z_1^{(2)})$$

# Our first example: Forward Pass



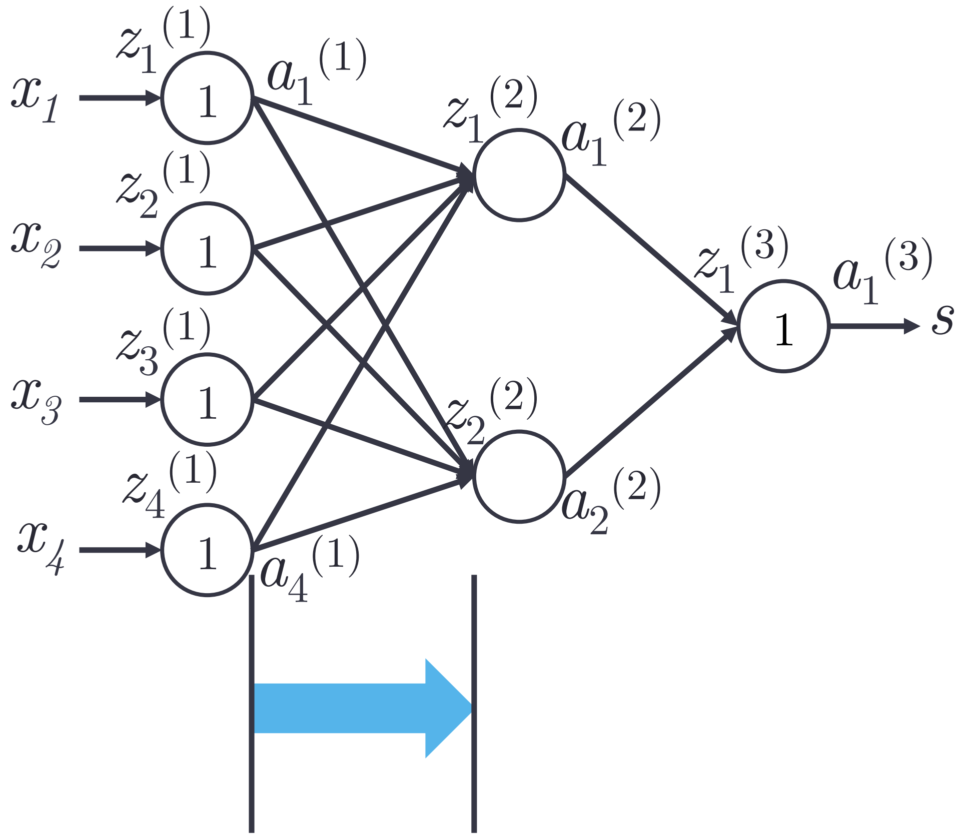

$$z_1^{(1)} = x_1$$
$$z_2^{(1)} = x_2$$
$$z_3^{(1)} = x_3$$
$$z_4^{(1)} = x_4$$

# Our first example: Forward Pass



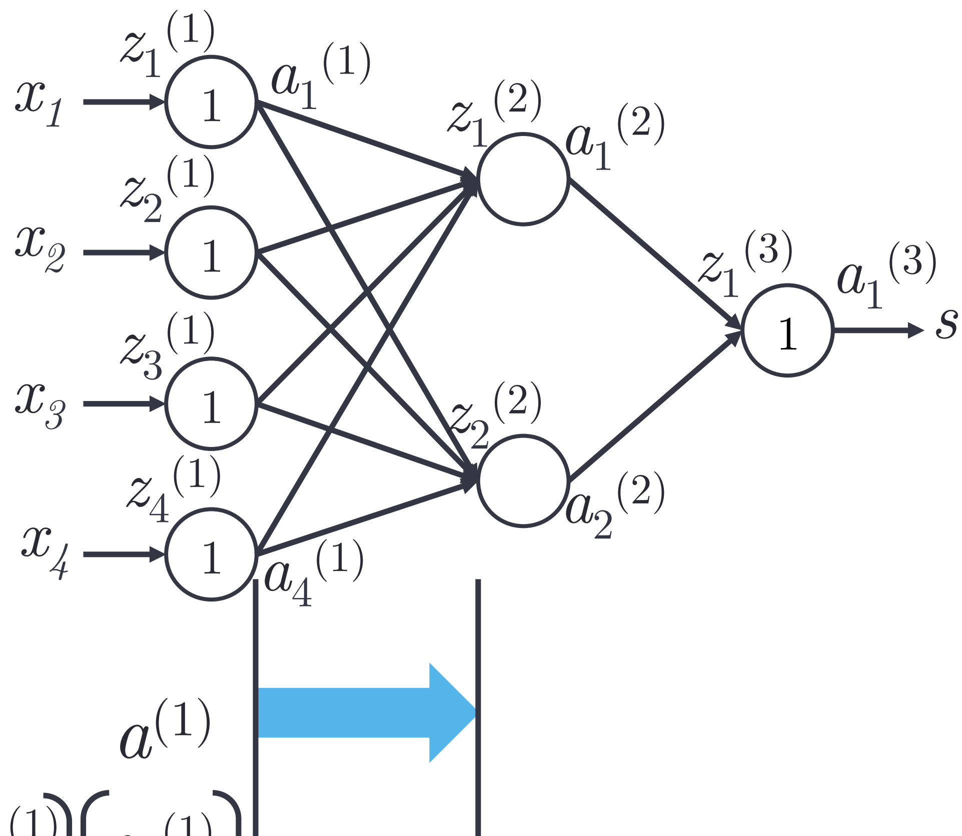

$$
\begin{aligned}
a_1^{(1)} &= z_1^{(1)} \\
a_2^{(1)} &= z_2^{(1)} \\
a_3^{(1)} &= z_3^{(1)} \\
a_4^{(1)} &= z_4^{(1)}
\end{aligned}
$$

# Our first example: Forward Pass



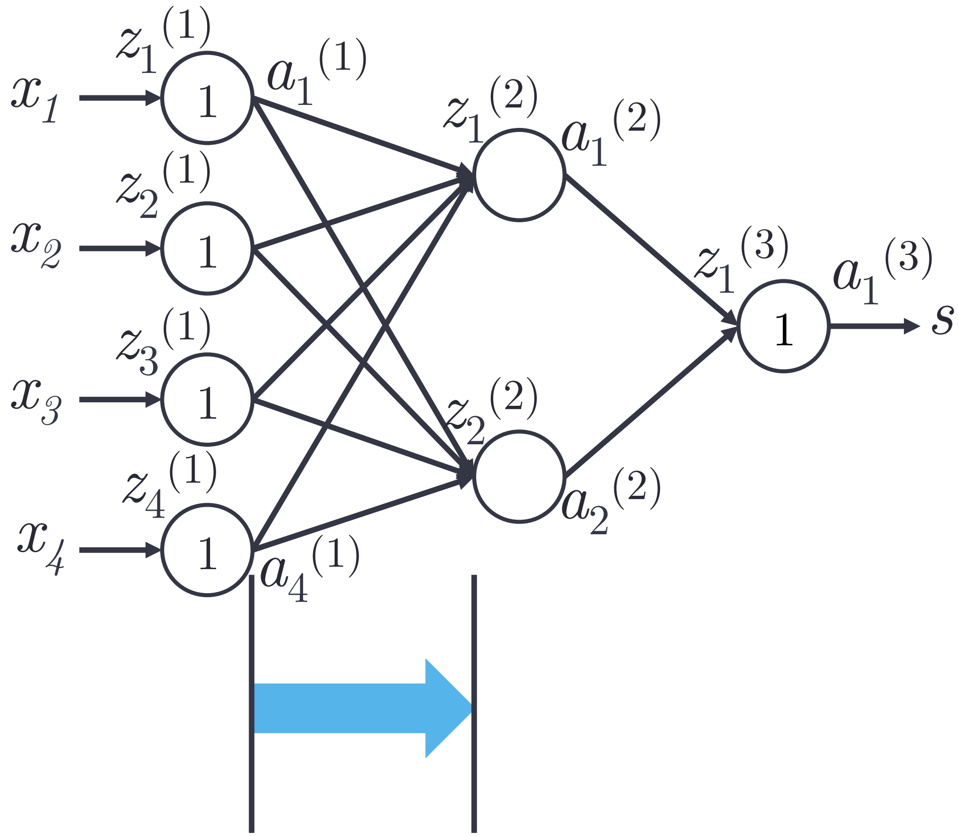

$$z_1^{(2)} = W_{11}^{(1)} a_1^{(1)} + W_{12}^{(1)} a_2^{(1)} + W_{13}^{(1)} a_3^{(1)} + W_{14}^{(1)} a_4^{(1)}$$
$$z_2^{(2)} = W_{21}^{(1)} a_1^{(1)} + W_{22}^{(1)} a_2^{(1)} + W_{23}^{(1)} a_3^{(1)} + W_{24}^{(1)} a_4^{(1)}$$

# Our first example: Forward Pass




$$\underset{z^{(2)}}{\begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \end{bmatrix}} = \underset{W^{(1)}}{\begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} & W_{14}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} & W_{24}^{(1)} \end{bmatrix}} \underset{a^{(1)}}{\begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \\ a_4^{(1)} \end{bmatrix}}$$

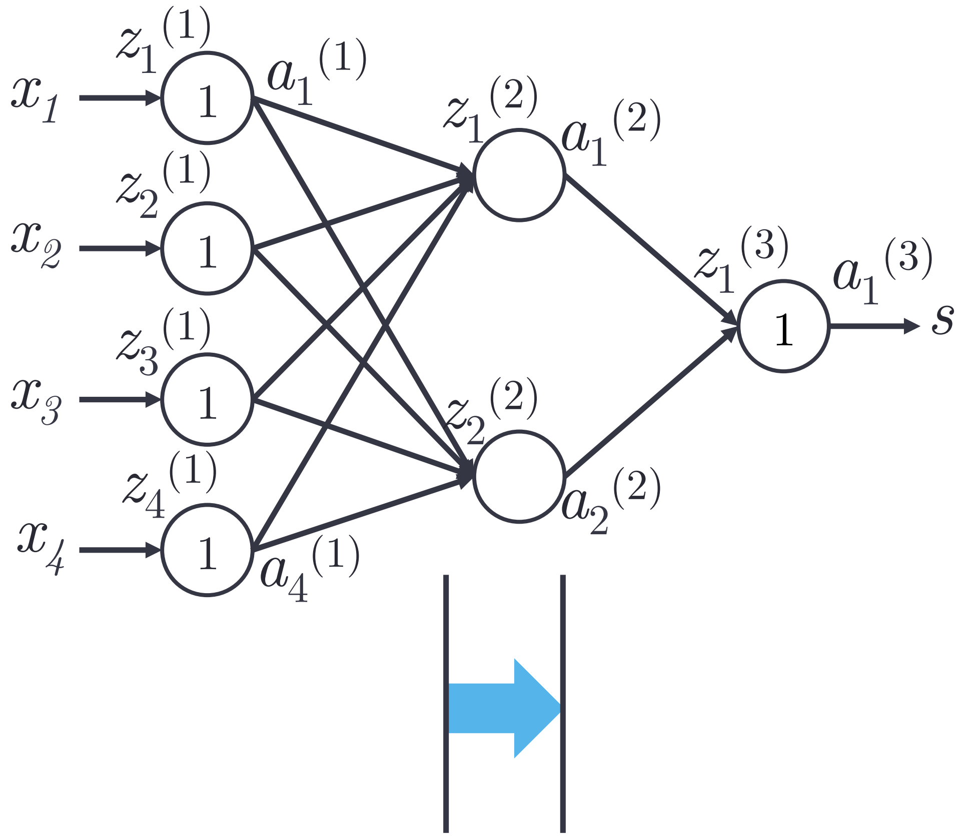
# Our first example: Forward Pass




$$z^{(2)} = W^{(1)} a^{(1)}$$

Affine transformation

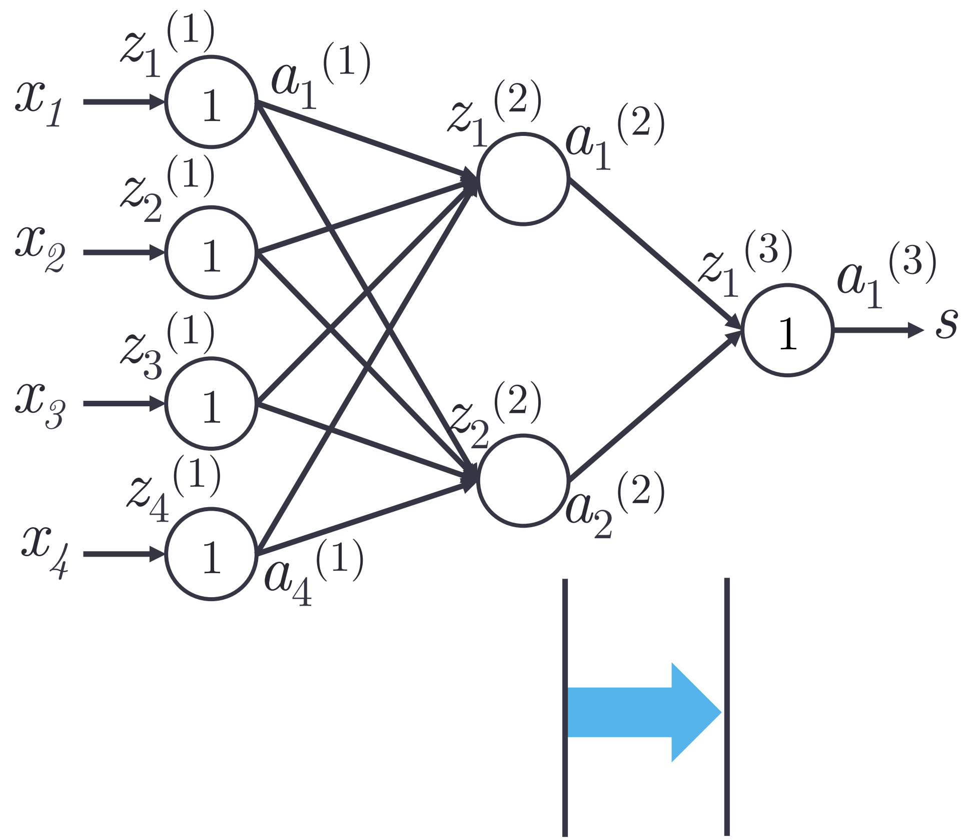
# Our first example: Forward Pass




$$a^{(2)} = \sigma(z^{(2)})$$

Point-wise/Element-wise non-linearity

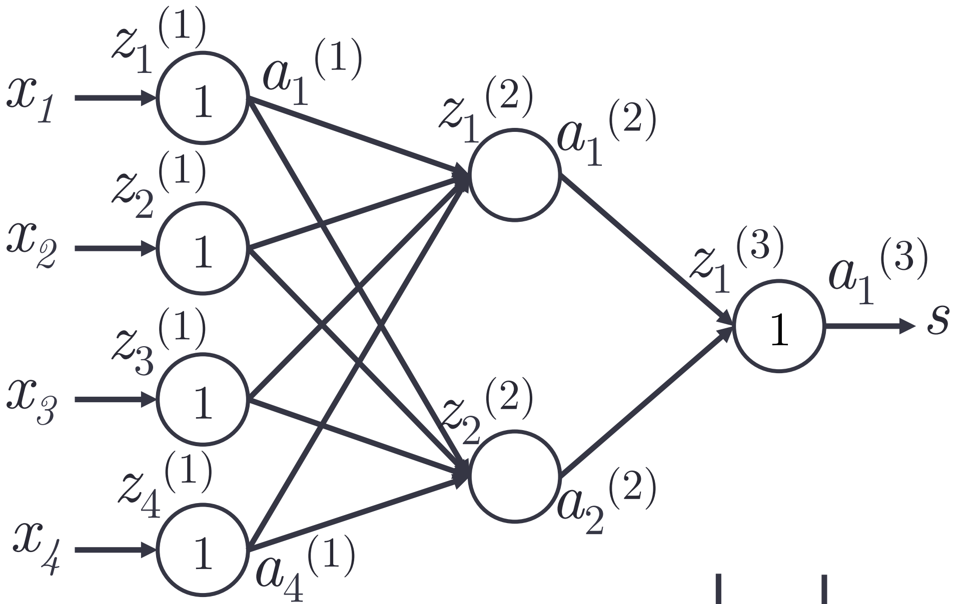
# Our first example: Forward Pass

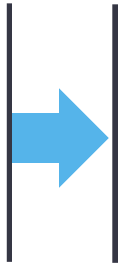



$z^{(3)} = W^{(2)} a^{(2)}$

Affine transformation

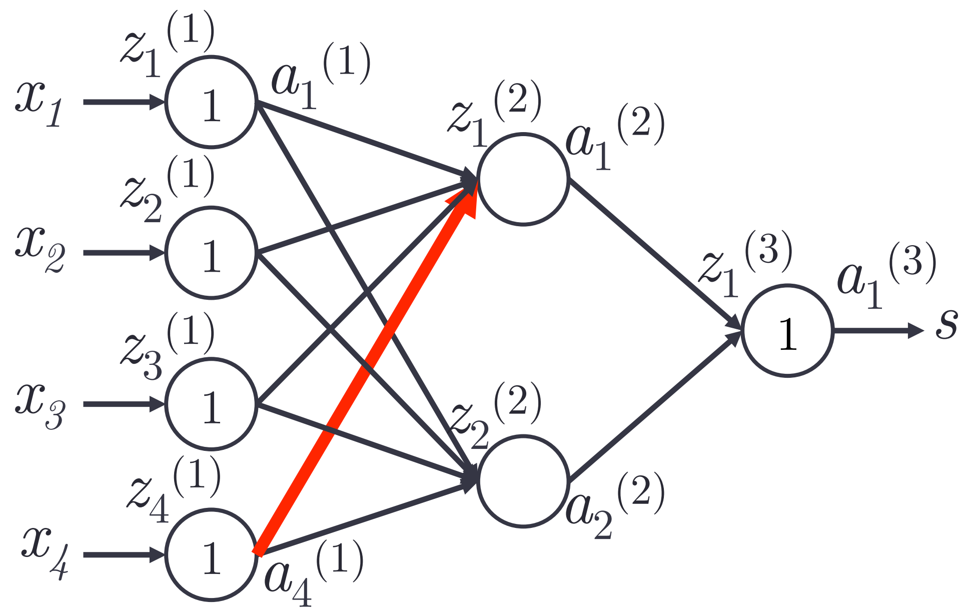
# Our first example: Forward Pass



$$a^{(3)} = z^{(3)}$$
$$s = a^{(3)}$$

# Our first example: Backpropagation using chain rule

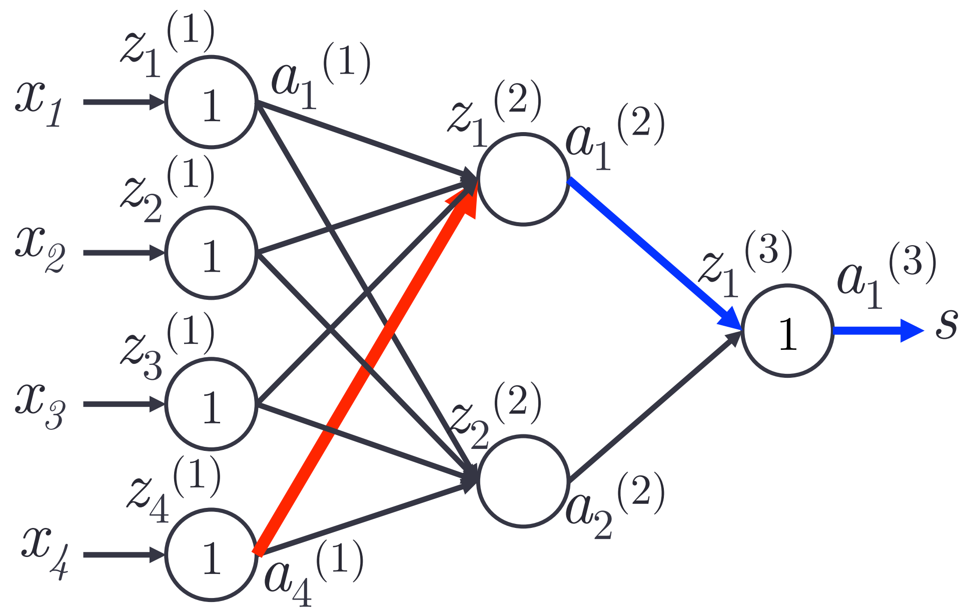



Let us try to calculate the error gradient wrt $W_{14}^{(1)}$

Thus we want to find:

$$\frac{\partial s}{\partial W_{14}^{(1)}}$$

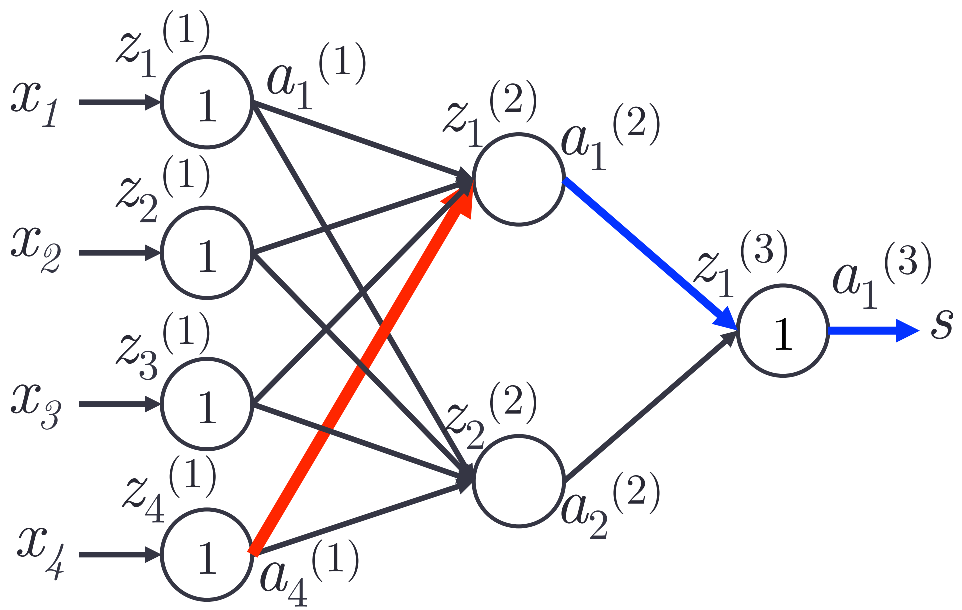
# Our first example: Backpropagation using chain rule




Let us try to calculate the error gradient wrt $W_{14}^{(1)}$
Thus we want to find:

$$\frac{\partial s}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial W_{14}^{(1)}}$$

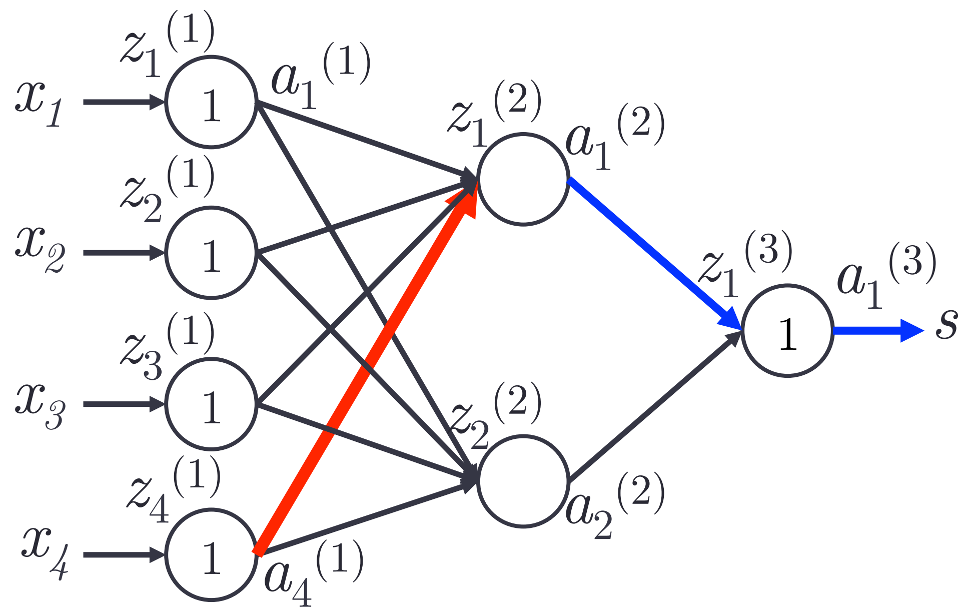
# Our first example: Backpropagation using chain rule




$\boxed{\text{This}}$ is simply 1

$$\boxed{\frac{\partial s}{\partial z_1^{(3)}}} \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial W_{14}^{(1)}}$$

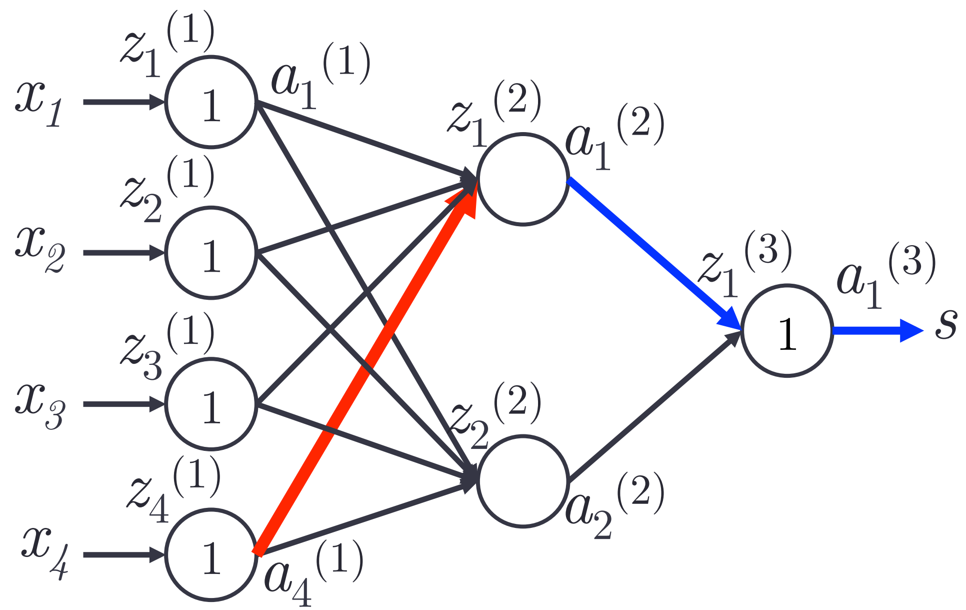
# Our first example: Backpropagation using chain rule




$$\frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial W_{14}^{(1)}}$$

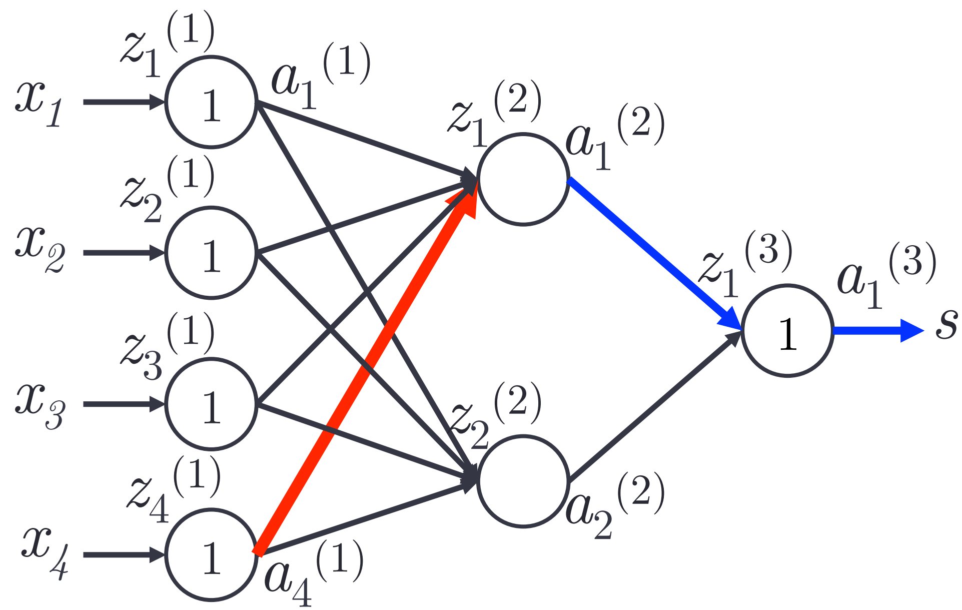
$$\frac{\partial (W_{11}^{(2)} a_1^{(2)} + W_{12}^{(2)} a_2^{(2)})}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial W_{14}^{(1)}}$$

# Our first example: Backpropagation using chain rule



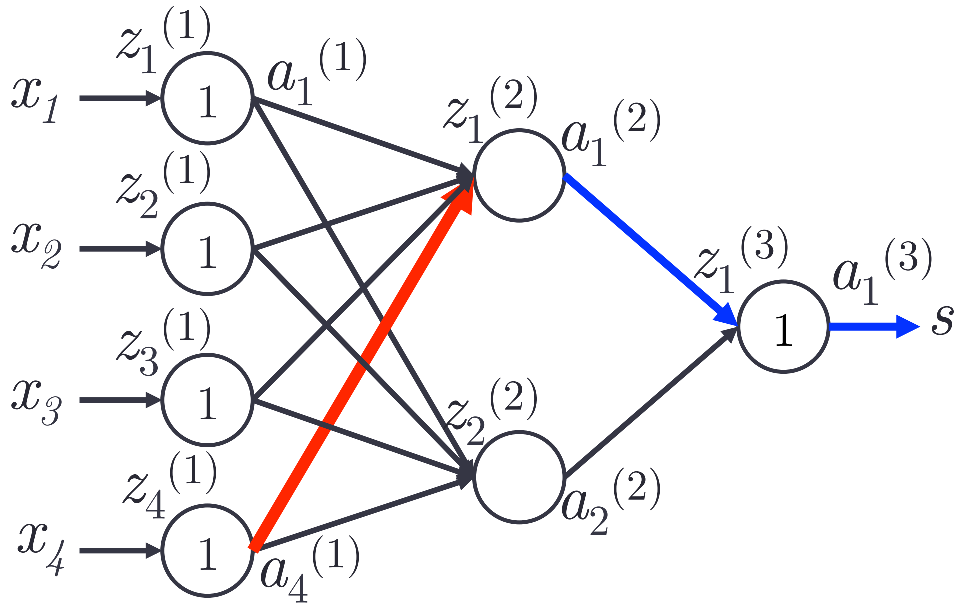

$$W_{11}^{(2)} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial W_{14}^{(1)}}$$

# Our first example: Backpropagation using chain rule



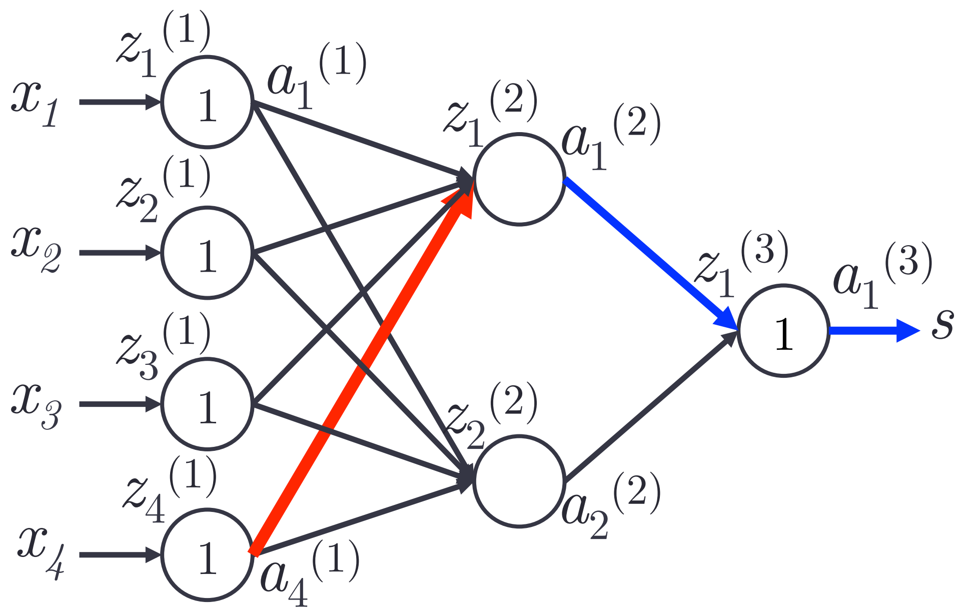

$$W_{11}^{(2)} \sigma'\left(z_1^{(2)}\right) \frac{\partial z_1^{(2)}}{\partial W_{14}^{(1)}}$$

# Our first example: Backpropagation using chain rule




$$W_{11}^{(2)}\sigma'\left(z_1^{(2)}\right)\frac{\partial(W_{11}^{(1)}a_1^{(1)} + W_{12}^{(1)}a_2^{(1)} + W_{13}^{(1)}a_3^{(1)} + W_{14}^{(1)}a_4^{(1)})}{\partial W_{14}^{(1)}}$$

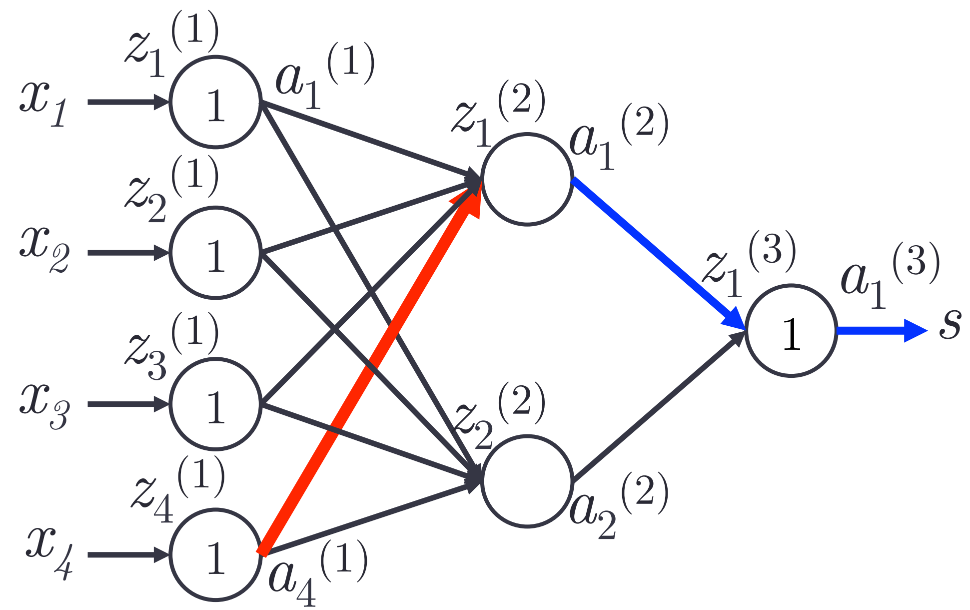
# Our first example: Backpropagation using chain rule




$$\underbrace{W_{11}^{(2)} \sigma'\left(z_1^{(2)}\right)}_{\delta_1^{(2)}} a_4^{(1)}$$

# Our first example: Backpropagation Observations

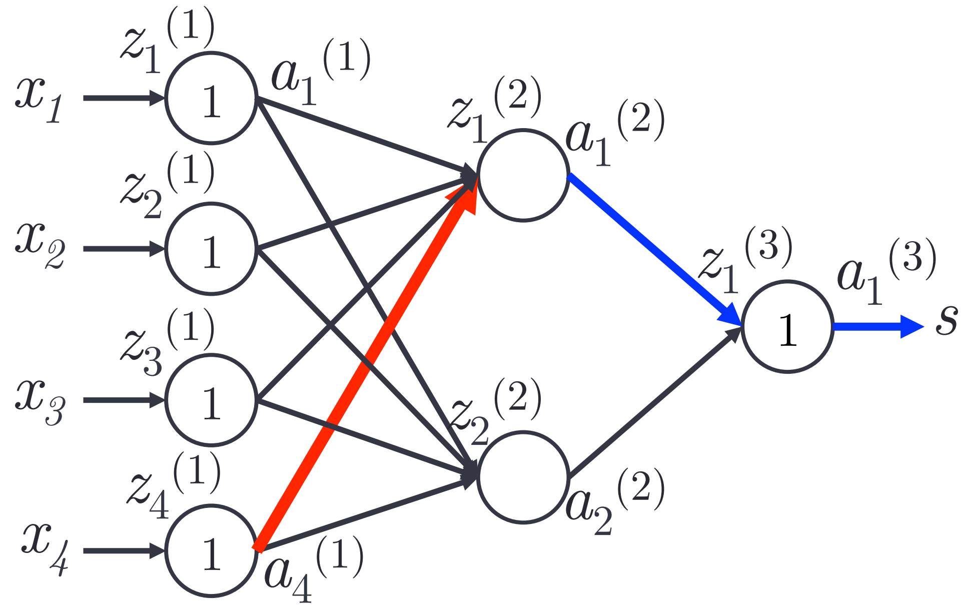
We got error gradient wrt $W_{14}^{(1)}$




Required:

- the signal forwarded by $W_{14}^{(1)} = a_4^{(1)}$
- the error propagating backwards $W_{11}^{(2)}$
- the local gradient $\sigma'(z_1^{(2)})$

# Our first example: Backpropagation Observations




We tried to get error gradient wrt $W_{14}^{(1)}$

Required:

- the signal forwarded by $W_{14}^{(1)} = a_4^{(1)}$
- the error propagating backwards $W_{11}^{(2)}$
- the local gradient $\sigma'(z_1^{(2)})$

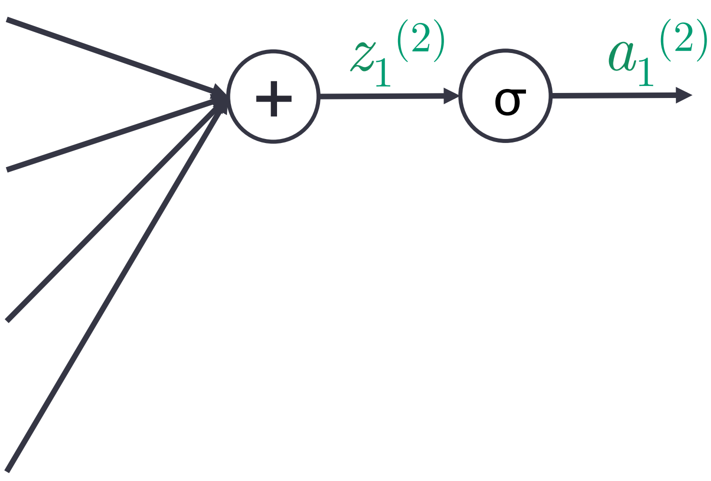
We can do this for all of $W^{(1)}$:

$$\begin{bmatrix} \delta_1^{(2)} a_1^{(1)} & \delta_1^{(2)} a_2^{(1)} & \delta_1^{(2)} a_3^{(1)} & \delta_1^{(2)} a_4^{(1)} \\ \delta_2^{(2)} a_1^{(1)} & \delta_2^{(2)} a_2^{(1)} & \delta_2^{(2)} a_3^{(1)} & \delta_2^{(2)} a_4^{(1)} \end{bmatrix}$$

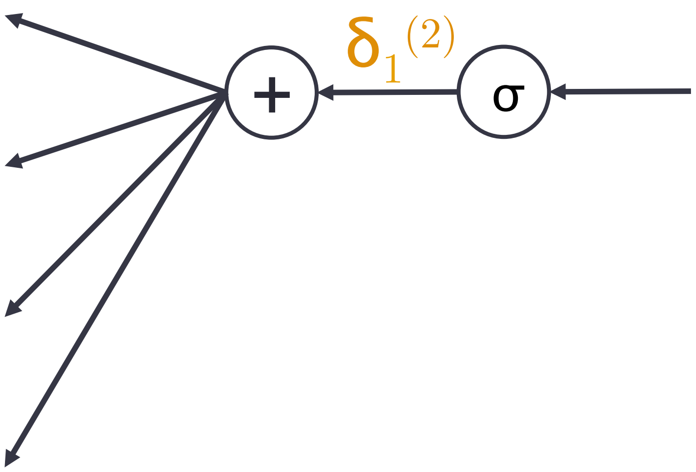
(as outer product)

$$\begin{bmatrix} \delta_1^{(2)} \\ \delta_2^{(2)} \end{bmatrix} \begin{bmatrix} a_1^{(1)} & a_2^{(1} & a_3^{(1)} & a_4^{(1)} \end{bmatrix}$$

# Our first example: Let us define δ




Recall that this is forward pass




This is the backpropagation

$\delta_1^{(2)}$ is the error flowing backwards at the same point where $z_1^{(2)}$ passed forwards. Thus it is simply the gradient of the error wrt $z_1^{(2)}$.

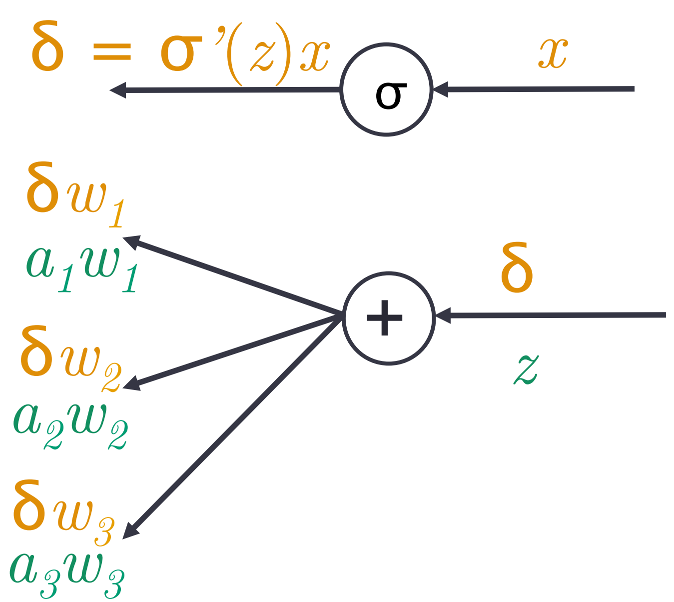
# Our first example: Backpropagation using error vectors

The chain rule of differentiation just boils down very simple patterns in error backpropagation:

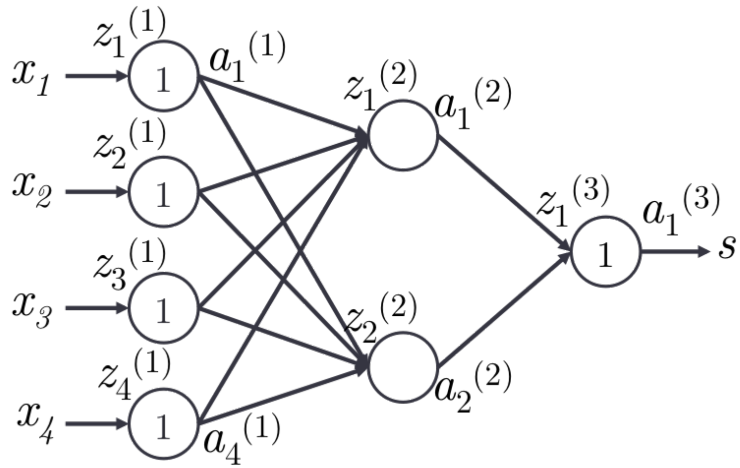
Orange = Backprop.
Green = Fwd. Pass

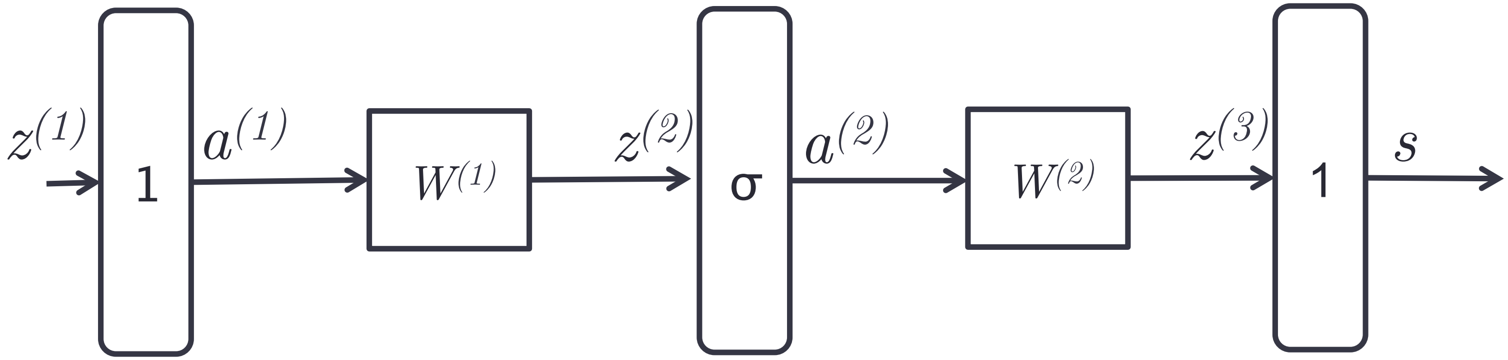
1. An error $x$ flowing backwards passes a neuron by getting amplified by the local gradient.

2. An error δ that needs to go through an affine transformation distributes itself in the way signal combined in forward pass.

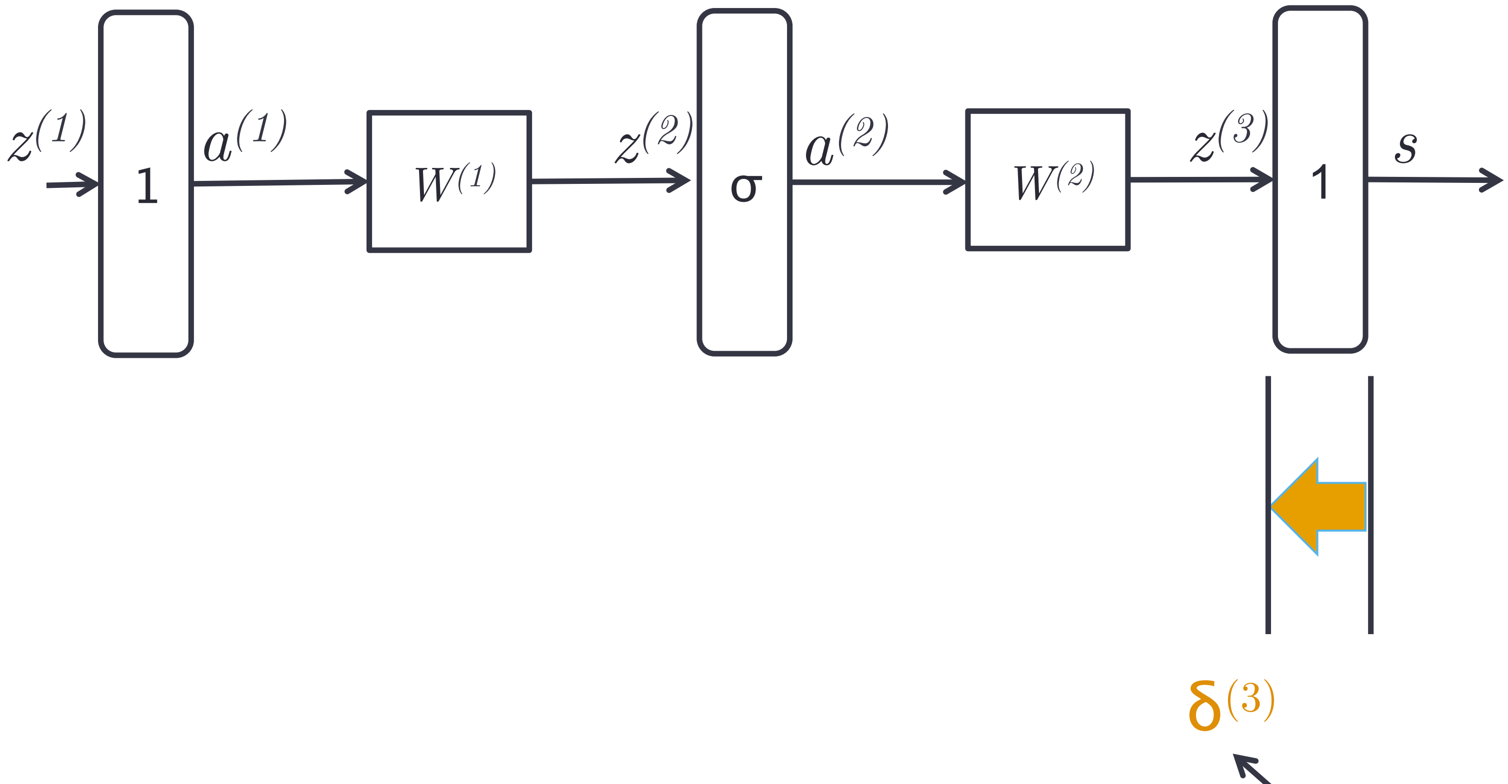


# Our first example: Backpropagation using error vectors



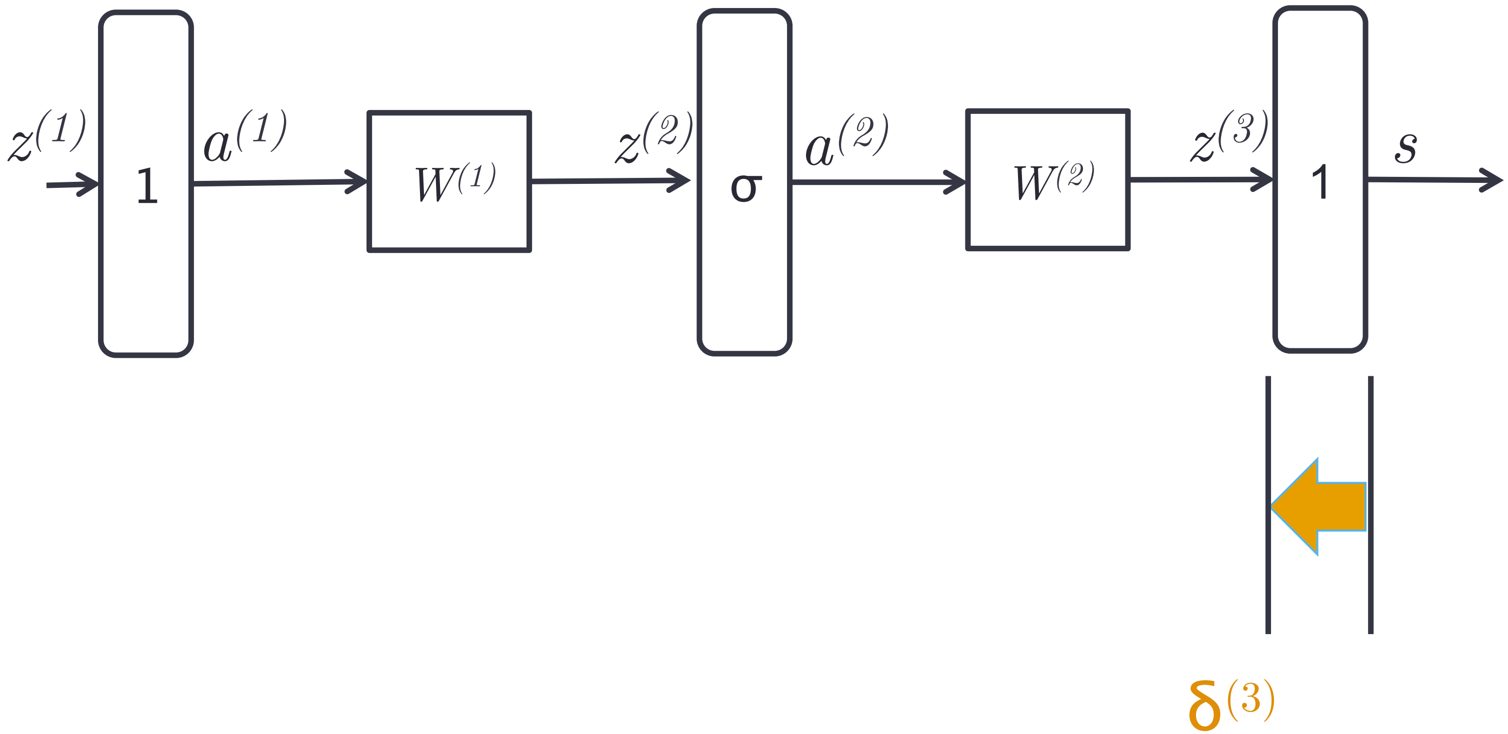



# Our first example: Backpropagation using error vectors



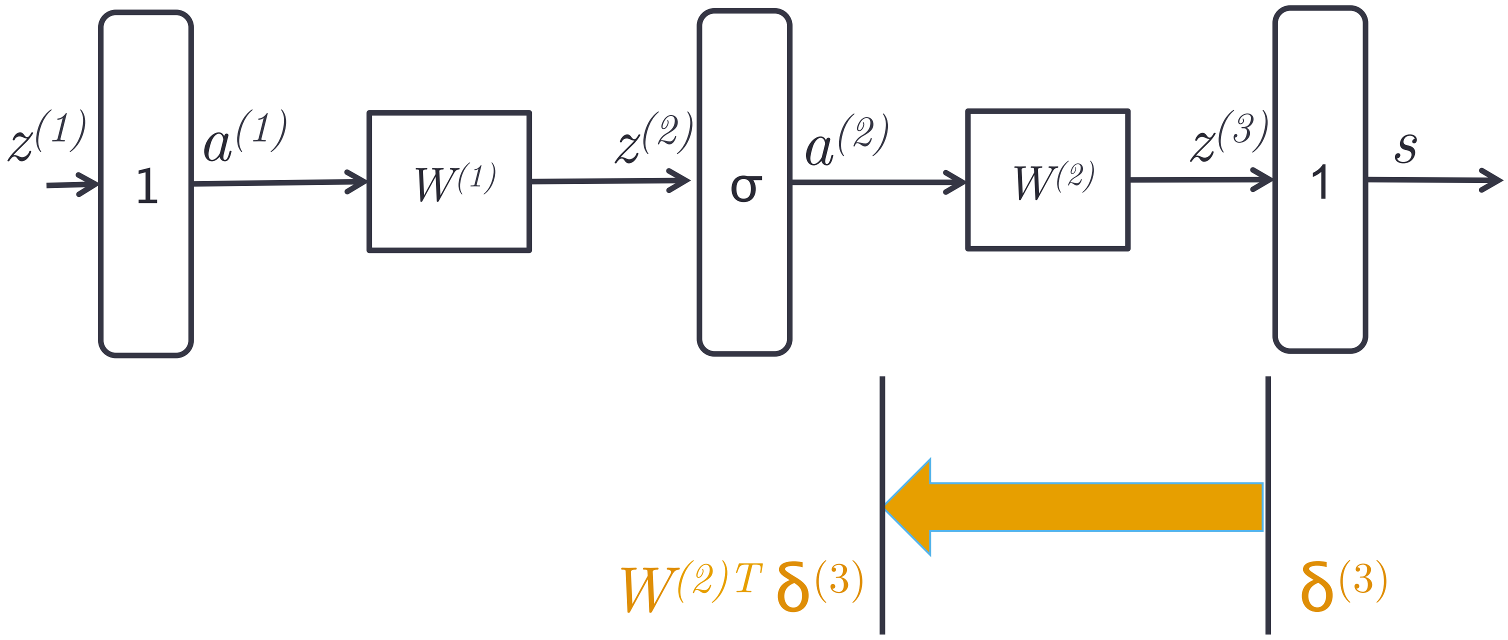

This is $\hat{y} - y$ for softmax

# Our first example: Backpropagation using error vectors



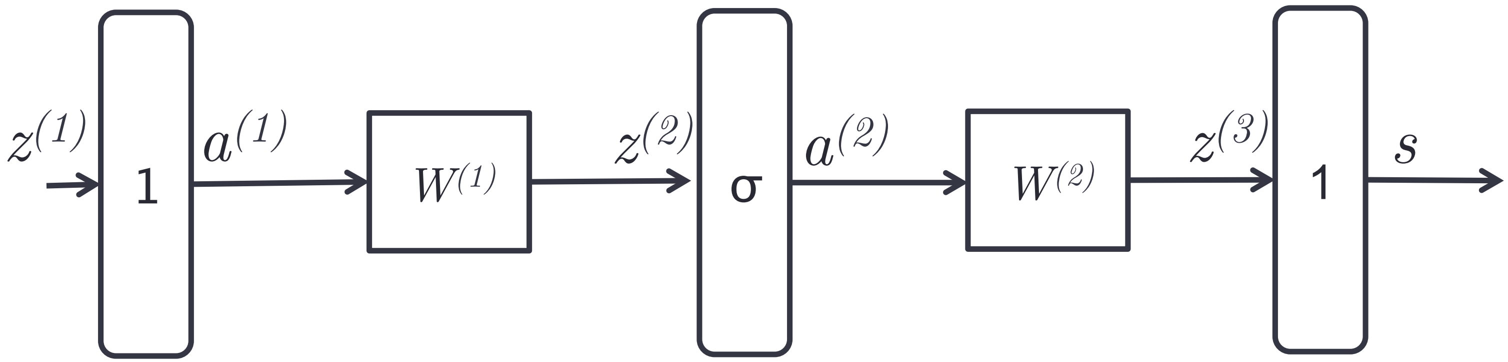

$$\text{Gradient w.r.t } W^{(2)} = \delta^{(3)} a^{(2)T}$$

# Our first example: Backpropagation using error vectors




--Reusing the $\delta^{(3)}$ for downstream updates.
--Moving error vector across affine transformation simply requires multiplication with the transpose of forward matrix
--Notice that the dimensions will line up perfectly too!

# Our first example: Backpropagation using error vectors

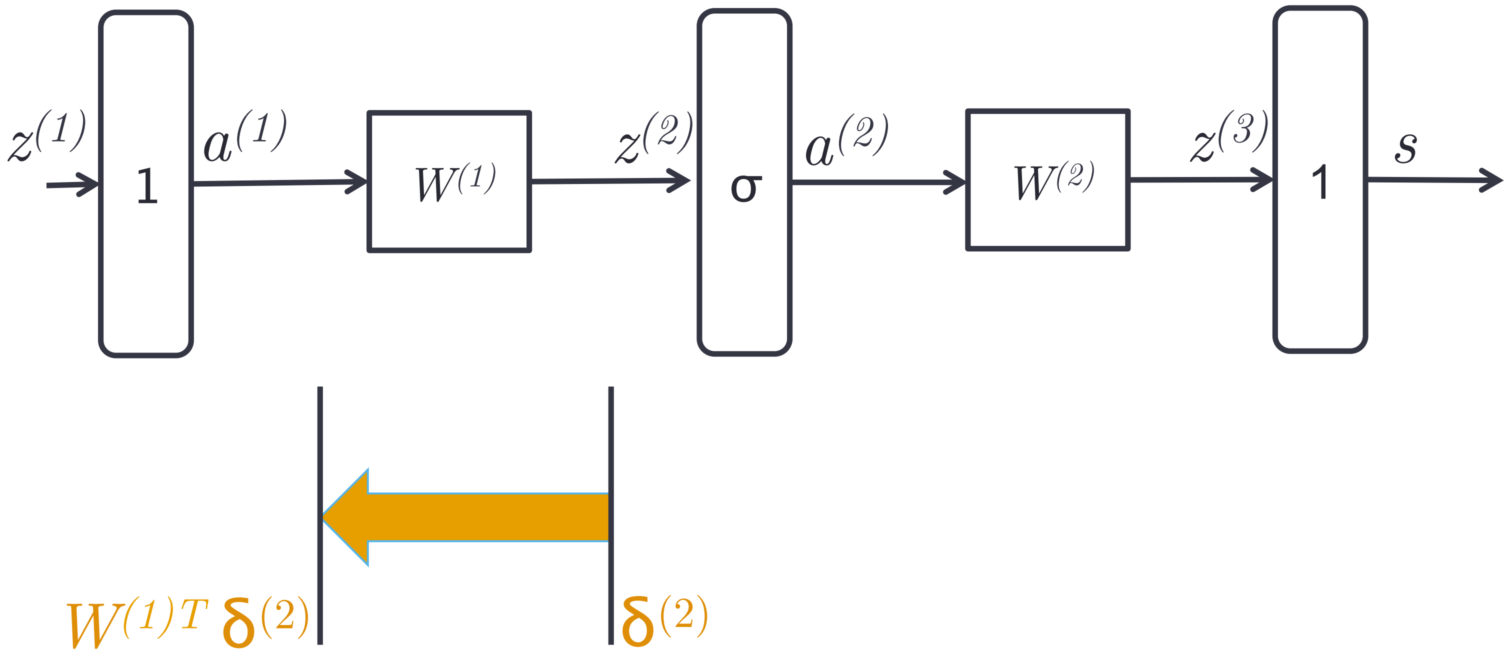



$$\sigma'(z^{(2)}) \odot W^{(2)T} \delta^{(3)} \qquad W^{(2)T} \delta^{(3)}$$

$$= \delta^{(2)}$$

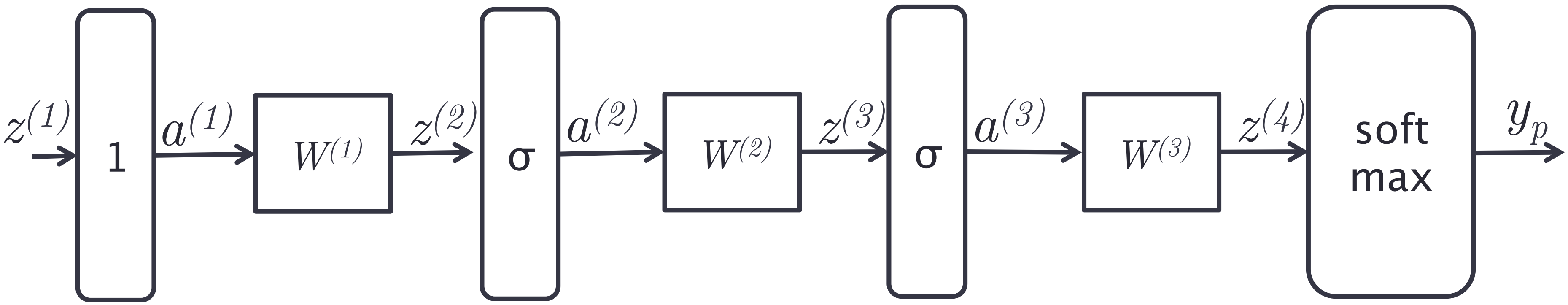
--Moving error vector across point-wise non-linearity requires point-wise multiplication with local gradient of the non-linearity

# Our first example: Backpropagation using error vectors

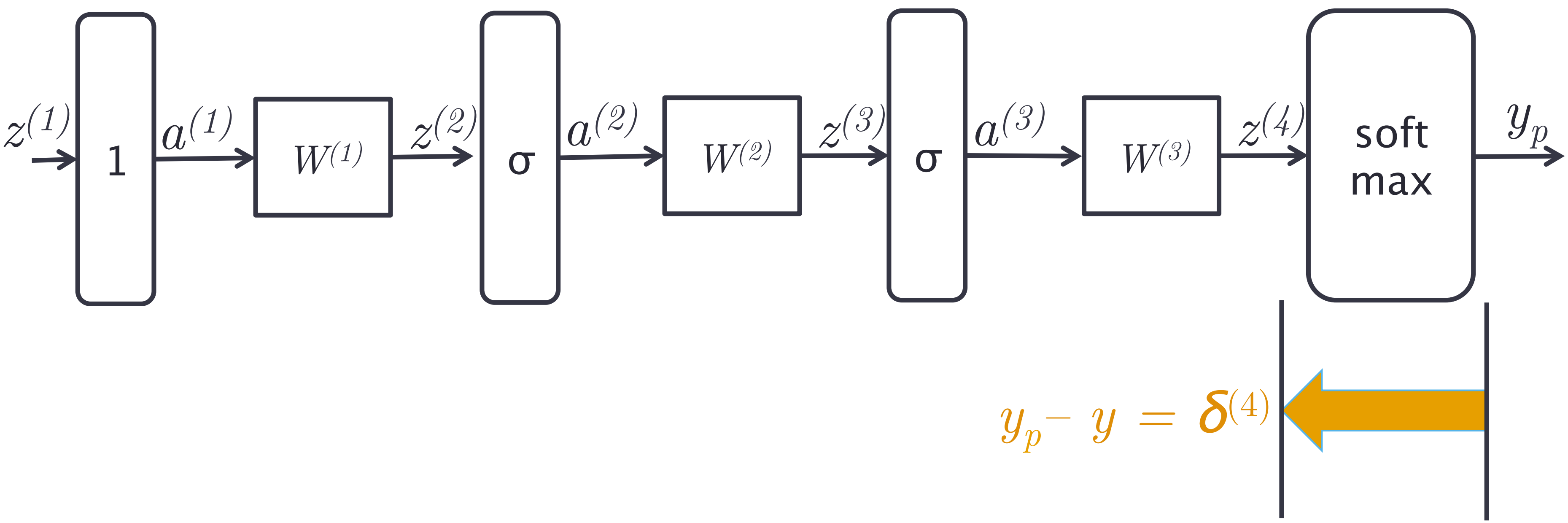



Gradient w.r.t $W^{(1)} = \delta^{(2)} a^{(1)\mathrm{T}}$

# Our second example (4-layer network): Backpropagation using error vectors

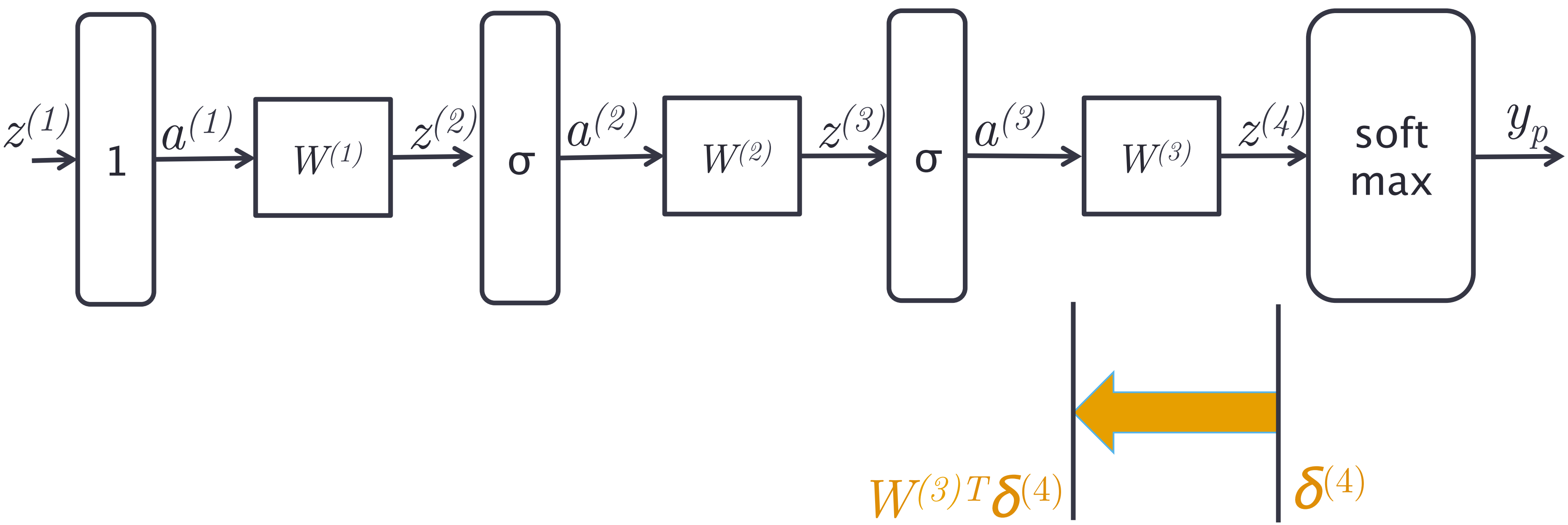


# Our second example (4-layer network): Backpropagation using error vectors

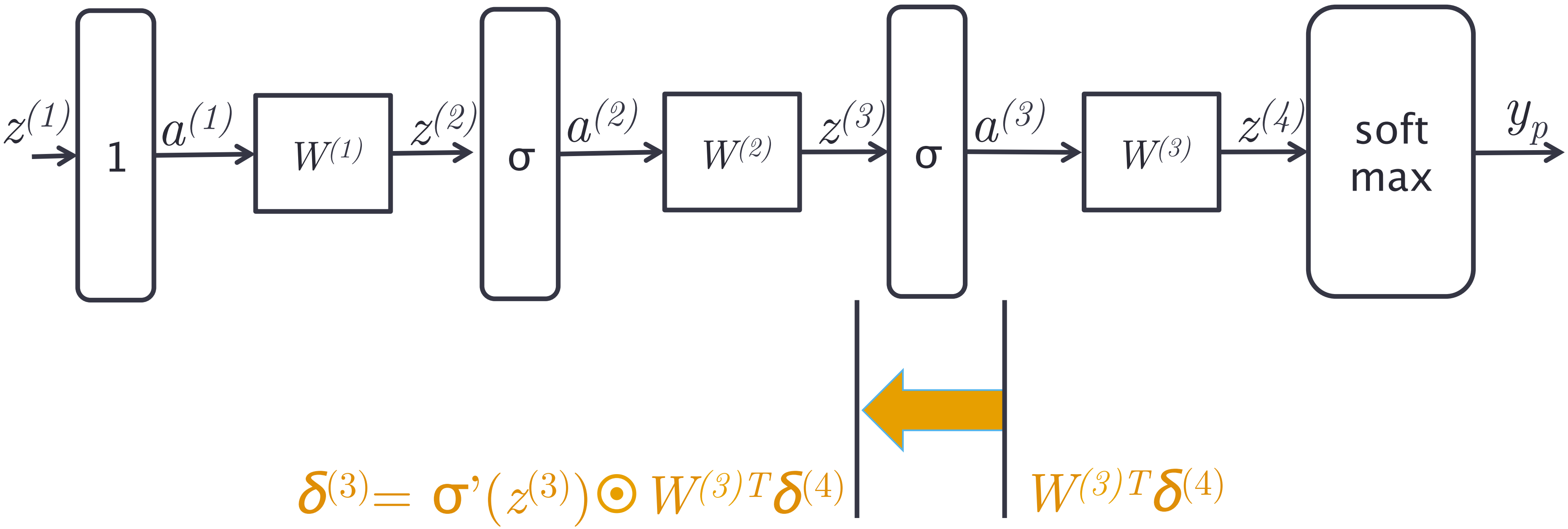


# Our second example (4-layer network): Backpropagation using error vectors

$$\text{Grad } W^{(3)} = \delta^{(4)} a^{(3)T}$$

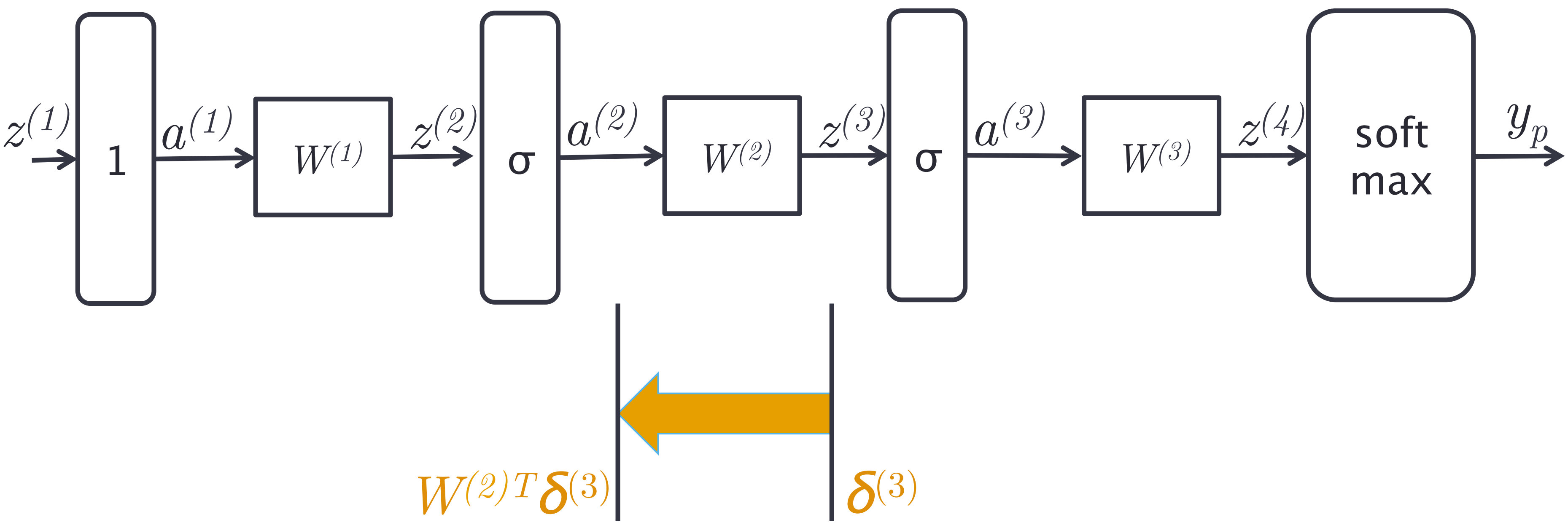


# Our second example (4-layer network): Backpropagation using error vectors

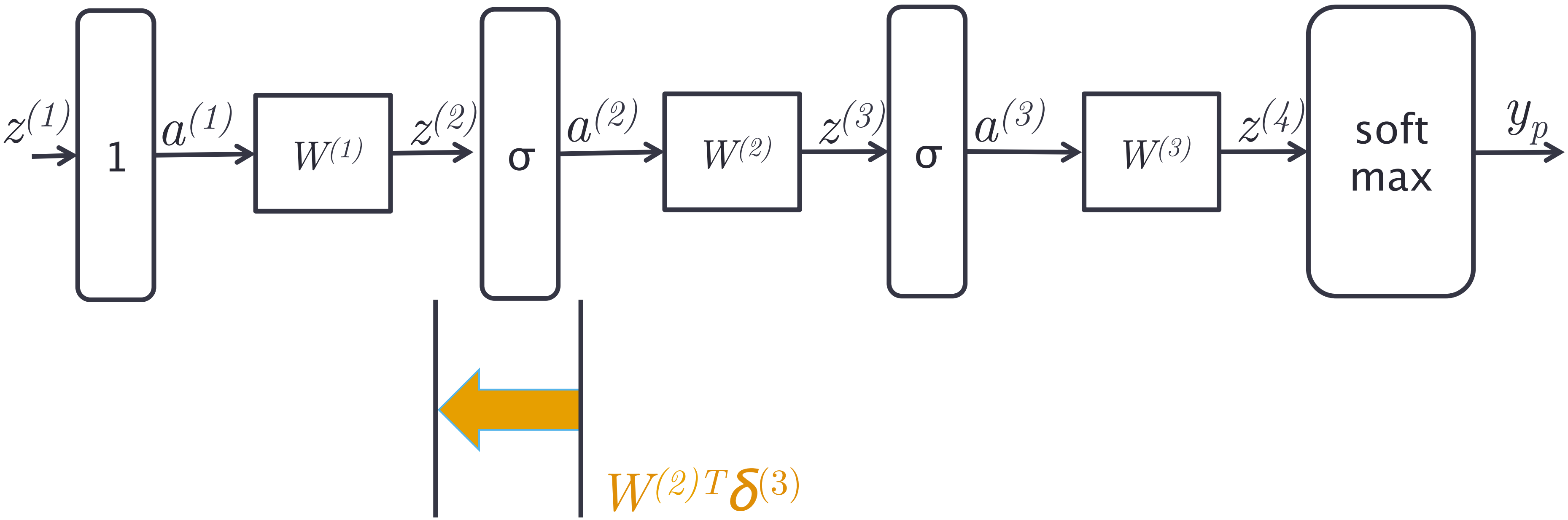


# Our second example (4-layer network): Backpropagation using error vectors

$$\text{Grad } W^{(2)} = \delta^{(3)} a^{(2)T}$$



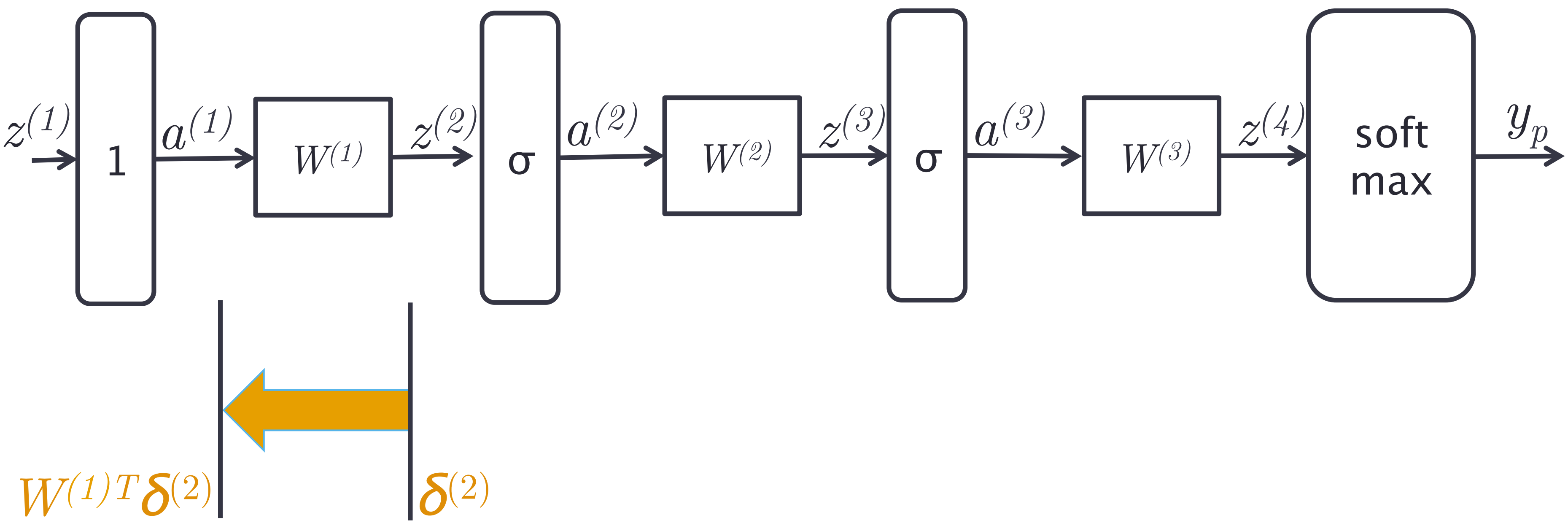
# Our second example (4-layer network): Backpropagation using error vectors




$\delta^{(2)} = \sigma'(z^{(2)}) \odot W^{(2)T}\delta^{(3)}$

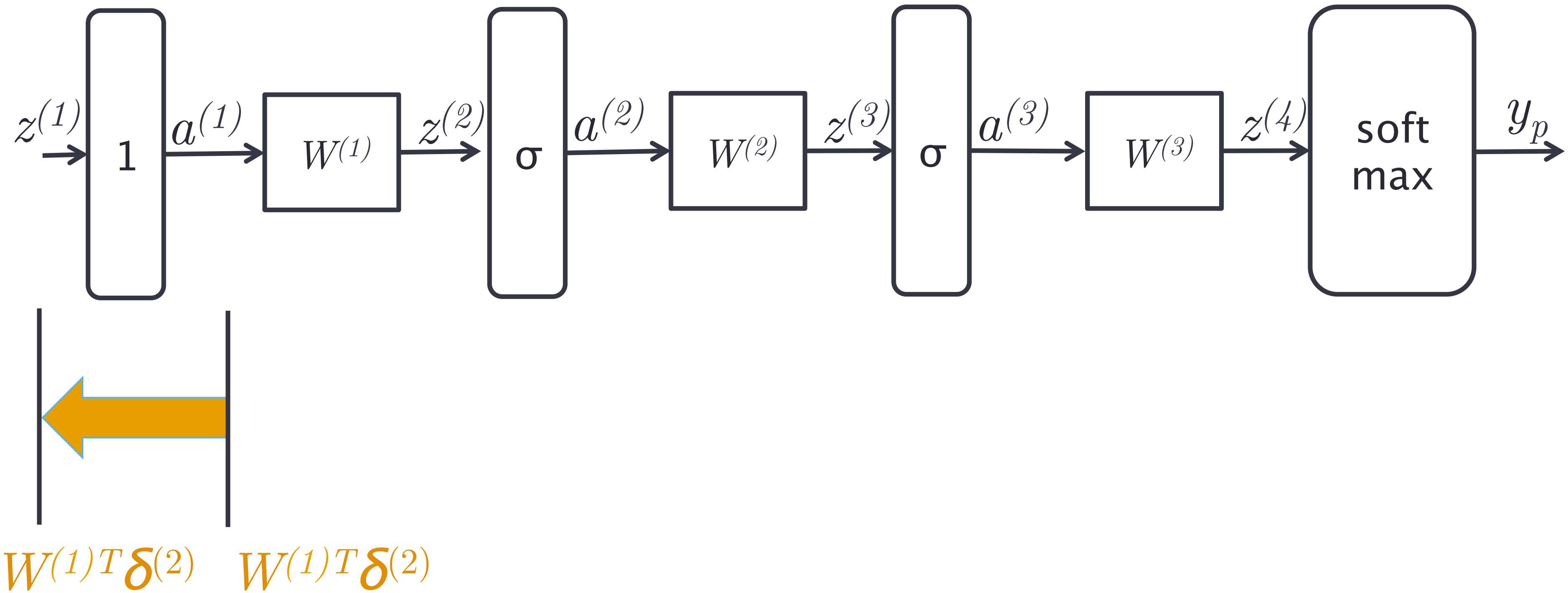
# Our second example (4-layer network): Backpropagation using error vectors

$$\text{Grad } W^{(1)} = \delta^{(2)} a^{(1)T}$$



# Our second example (4-layer network): Backpropagation using error vectors

$\text{Grad wrt input vector} = W^{(1)T}\delta^{(2)}$



# CS224D Midterm Review

Ian Tenney

May 4, 2015

# Outline

## Backpropagation (continued)

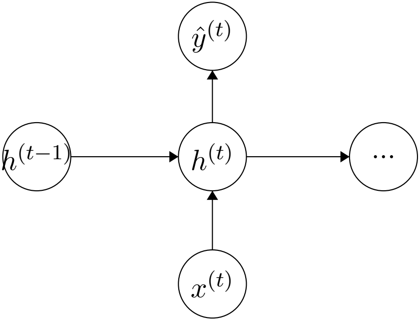
- RNN Structure
- RNN Backpropagation

## Backprop on a DAG

- Example: Gated Recurrent Units (GRUs)
- GRU Backpropagation

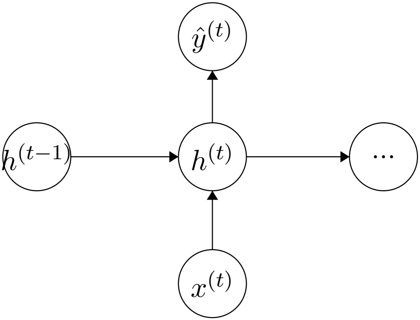
# Outline

# Basic RNN Structure




- Basic RNN ("Elman network")
- You've seen this on Assignment #2 (and also in Lecture #5)

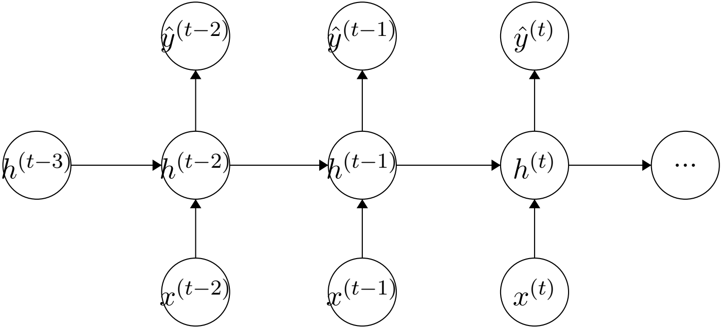
# Basic RNN Structure




- Two layers between input and prediction, plus hidden state

$$
\begin{aligned}
h^{(t)} &= \text{sigmoid}\left(Hh^{(t-1)} + Wx^{(t)} + b_1\right) \\
\hat{y}^{(t)} &= \text{softmax}\left(Uh^{(t)} + b_2\right)
\end{aligned}
$$

# Unrolled RNN




- Helps to think about as "unrolled" network: distinct nodes for each timestep
- Just do backprop on this! Then combine shared gradients.

# Backprop on RNN

- Usual cross-entropy loss ($k$-class):

$$\begin{aligned} \bar{P}(y^{(t)} = j \mid x^{(t)}, \ldots, x^{(1)}) &= \hat{y}_j^{(t)} \\ J^{(t)}(\theta) &= -\sum_{j=1}^{k} y_j^{(t)} \log \hat{y}_j^{(t)} \end{aligned}$$

- Just do backprop on this! First timestep ($\tau = 1$):

$$\frac{\partial J^{(t)}}{\partial U} \qquad \frac{\partial J^{(t)}}{\partial b_2}$$

$$\left.\frac{\partial J^{(t)}}{\partial H}\right|_{(t)} \qquad \frac{\partial J^{(t)}}{\partial h^{(t)}} \qquad \left.\frac{\partial J^{(t)}}{\partial W}\right|_{(t)} \qquad \frac{\partial J^{(t)}}{\partial x^{(t)}}$$

# Backprop on RNN

- First timestep ($s = 0$):

$$\frac{\partial J^{(t)}}{\partial U} \qquad \frac{\partial J^{(t)}}{\partial b_2}$$

$$\left.\frac{\partial J^{(t)}}{\partial H}\right|_{(t)} \qquad \frac{\partial J^{(t)}}{\partial h^{(t)}} \qquad \left.\frac{\partial J^{(t)}}{\partial W}\right|_{(t)} \qquad \frac{\partial J^{(t)}}{\partial x^{(t)}}$$

- Back in time ($s = 1, 2, \ldots, \tau - 1$)

$$\left.\frac{\partial J^{(t)}}{\partial H}\right|_{(t-s)} \qquad \frac{\partial J^{(t)}}{\partial h^{(t-s)}} \qquad \left.\frac{\partial J^{(t)}}{\partial W}\right|_{(t-s)} \qquad \frac{\partial J^{(t)}}{\partial x^{(t-s)}}$$

# Backprop on RNN

## Yuck, that's a lot of math!

- Actually, it's not so bad.
- Solution: error vectors ($\delta$)

# Making sense of the madness

- Chain rule to the rescue!
- $a^{(t)} = Uh^{(t)} + b_2$
- $\hat{y}^{(t)} = \text{softmax}(a^{(t)})$
- Gradient is *transpose* of Jacobian:

$$\nabla_a J = \left(\frac{\partial J^{(t)}}{\partial a^{(t)}}\right)^T = \hat{y}^{(t)} - y^{(t)} = \delta^{(2)(t)} \quad \in \mathbb{R}^{k\times 1}$$

- Now dimensions work out:

$$\frac{\partial J^{(t)}}{\partial a^{(t)}} \cdot \frac{\partial a^{(t)}}{\partial b_2} = (\delta^{(2)(t)})^T I \quad \in \mathbb{R}^{(1\times k)\cdot(k\times k)} = \mathbb{R}^{1\times k}$$

# Making sense of the madness

- Chain rule to the rescue!
- $a^{(t)} = Uh^{(t)} + b_2$
- $\hat{y}^{(t)} = \text{softmax}(a^{(t)})$
- Gradient is *transpose* of Jacobian:

$$\nabla_a J = \left(\frac{\partial J^{(t)}}{\partial a^{(t)}}\right)^T = \hat{y}^{(t)} - y^{(t)} = \delta^{(2)(t)} \quad \in \mathbb{R}^{k\times 1}$$

- Now dimensions work out:

$$\frac{\partial J^{(t)}}{\partial a^{(t)}} \cdot \frac{\partial a^{(t)}}{\partial b_2} = (\delta^{(2)(t)})^T I \quad \in \mathbb{R}^{(1\times k)\cdot(k\times k)} = \mathbb{R}^{1\times k}$$

# Making sense of the madness

- Chain rule to the rescue!
- $a^{(t)} = Uh^{(t)} + b_2$
- $\hat{y}^{(t)} = \text{softmax}(a^{(t)})$
- Gradient is *transpose* of Jacobian:

$$\nabla_a J = \left(\frac{\partial J^{(t)}}{\partial a^{(t)}}\right)^T = \hat{y}^{(t)} - y^{(t)} = \delta^{(2)(t)} \qquad \in \mathbb{R}^{k\times 1}$$

- Now dimensions work out:

$$\frac{\partial J^{(t)}}{\partial a^{(t)}} \cdot \frac{\partial a^{(t)}}{\partial b_2} = (\delta^{(2)(t)})^T I \qquad \in \mathbb{R}^{(1\times k)\cdot(k\times k)} = \mathbb{R}^{1\times k}$$

# Making sense of the madness

- Chain rule to the rescue!
- $a^{(t)} = Uh^{(t)} + b_2$
- $\hat{y}^{(t)} = \text{softmax}(a^{(t)})$
- Matrix dimensions get weird:

$$\frac{\partial a^{(t)}}{\partial U} \quad \in \mathbb{R}^{k \times (k \times D_h)}$$

- But we don't need fancy tensors:

$$\nabla_U J^{(t)} = \left( \frac{\partial J^{(t)}}{\partial a^{(t)}} \cdot \frac{\partial a^{(t)}}{\partial U} \right)^T = \delta^{(2)(t)} (h^{(t)})^T \quad \in \mathbb{R}^{k \times D_h}$$

- NumPy: `self.grads.U += outer(d2, hs[t])`

# Making sense of the madness

- Chain rule to the rescue!
- $a^{(t)} = Uh^{(t)} + b_2$
- $\hat{y}^{(t)} = \text{softmax}(a^{(t)})$
- Matrix dimensions get weird:

$$\frac{\partial a^{(t)}}{\partial U} \quad \in \mathbb{R}^{k \times (k \times D_h)}$$

- But we don't need fancy tensors:

$$\nabla_U J^{(t)} = \left( \frac{\partial J^{(t)}}{\partial a^{(t)}} \cdot \frac{\partial a^{(t)}}{\partial U} \right)^T = \delta^{(2)(t)} (h^{(t)})^T \quad \in \mathbb{R}^{k \times D_h}$$

- NumPy: `self.grads.U += outer(d2, hs[t])`

# Going deeper

- Really just need one simple pattern:
- $z^{(t)} = Hh^{(t-1)} + Wx^{(t)} + b_1$
- $h^{(t)} = f(z^{(t)})$
- Compute error delta ($s = 0, 1, 2, \ldots$):
  - From top: $\delta^{(t)} = \left[h^{(t)} \circ (1 - h^{(t)})\right] \circ U^T \delta^{(2)(t)}$
  - Deeper: $\delta^{(t-s)} = \left[h^{(t-s)} \circ (1 - h^{(t-s)})\right] \circ H^T \delta^{(t-s+1)}$
- These are just chain-rule expansions!

$$\frac{\partial J^{(t)}}{\partial z^{(t)}} = \frac{\partial J^{(t)}}{\partial a^{(t)}} \cdot \frac{\partial a^{(t)}}{\partial h^{(t)}} \cdot \frac{\partial h^{(t)}}{\partial z^{(t)}} = (\delta^{(t)})^T$$

# Going deeper

- Really just need one simple pattern:
- $z^{(t)} = Hh^{(t-1)} + Wx^{(t)} + b_1$
- $h^{(t)} = f(z^{(t)})$
- Compute error delta ($s = 0, 1, 2, \ldots$):
  - From top: $\delta^{(t)} = \left[h^{(t)} \circ (1 - h^{(t)})\right] \circ U^T \delta^{(2)(t)}$
  - Deeper: $\delta^{(t-s)} = \left[h^{(t-s)} \circ (1 - h^{(t-s)})\right] \circ H^T \delta^{(t-s+1)}$
- These are just chain-rule expansions!

$$\frac{\partial J^{(t)}}{\partial z^{(t)}} = \frac{\partial J^{(t)}}{\partial a^{(t)}} \cdot \frac{\partial a^{(t)}}{\partial h^{(t)}} \cdot \frac{\partial h^{(t)}}{\partial z^{(t)}} = (\delta^{(t)})^T$$

# Going deeper

- These are just chain-rule expansions!

$$\left.\frac{\partial J^{(t)}}{\partial b_1}\right|_{(t)} = \left(\frac{\partial J^{(t)}}{\partial a^{(t)}} \cdot \frac{\partial a^{(t)}}{\partial h^{(t)}} \cdot \frac{\partial h^{(t)}}{\partial z^{(t)}}\right) \cdot \frac{\partial z^{(t)}}{\partial b_1} = (\delta^{(t)})^T \frac{\partial z^{(t)}}{\partial b_1}$$

$$\left.\frac{\partial J^{(t)}}{\partial H}\right|_{(t)} = \left(\frac{\partial J^{(t)}}{\partial a^{(t)}} \cdot \frac{\partial a^{(t)}}{\partial h^{(t)}} \cdot \frac{\partial h^{(t)}}{\partial z^{(t)}}\right) \cdot \frac{\partial z^{(t)}}{\partial H} = (\delta^{(t)})^T \frac{\partial z^{(t)}}{\partial H}$$

$$\frac{\partial J^{(t)}}{\partial z^{(t-1)}} = \left(\frac{\partial J^{(t)}}{\partial a^{(t)}} \cdot \frac{\partial a^{(t)}}{\partial h^{(t)}} \cdot \frac{\partial h^{(t)}}{\partial z^{(t)}}\right) \cdot \frac{\partial z^{(t)}}{\partial h^{(t-1)}} = (\delta^{(t)})^T \frac{\partial z^{(t)}}{\partial z^{(t-1)}}$$

# Going deeper

- And there's shortcuts for them too:

$$
\begin{aligned}
\left( \left. \frac{\partial J^{(t)}}{\partial b_1} \right|_{(t)} \right)^T &= \delta^{(t)} \\
\left( \left. \frac{\partial J^{(t)}}{\partial H} \right|_{(t)} \right)^T &= \delta^{(t)} \cdot (h^{(t-1)})^T \\
\left( \frac{\partial J^{(t)}}{\partial z^{(t-1)}} \right)^T &= \left[ h^{(t-1)} \circ (1 - h^{(t-1)}) \right] \circ H^T \delta^{(t)} = \delta^{(t-1)}
\end{aligned}
$$

# Outline

# Motivation

- Gated units with “reset” and “output” gates
- Reduce problems with vanishing gradients

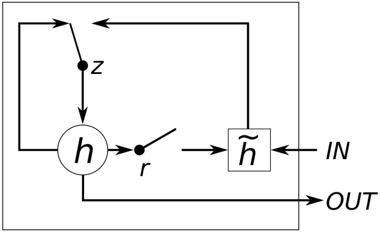



Figure : *You are likely to be eaten by a GRU.* (Figure from Chung, et al. 2014)

# Intuition

- Gates $z_i$ and $r_i$ for *each* hidden layer neuron
- $z_i, r_i \in [0, 1]$
- $\tilde{h}$ as "candidate" hidden layer
- $\tilde{h}$, $z$, $r$ all depend on on $x^{(t)}$, $h^{(t-1)}$
- $h^{(t)}$ depends on $h^{(t-1)}$ mixed with $\tilde{h}^{(t)}$

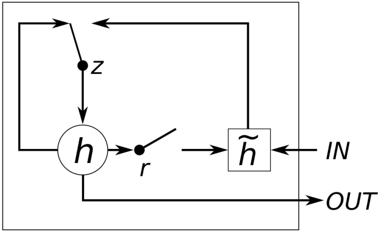



Figure : *You are likely to be eaten by a GRU.* (Figure from Chung, et al. 2014)

# Equations

- $z^{(t)} = \sigma\left(W_z x^{(t)} + U_z h^{(t-1)}\right)$
- $r^{(t)} = \sigma\left(W_r x^{(t)} + U_r h^{(t-1)}\right)$
- $\tilde{h}^{(t)} = \tanh\left(W x^{(t)} + r^{(t)} \circ U h^{(t-1)}\right)$
- $h^{(t)} = z^{(t)} \circ h^{(t-1)} + (1 - z^{(t)}) \circ \tilde{h}^{(t)}$
- Optionally can have biases; omitted for clarity.

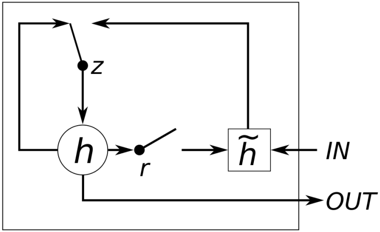



Figure : *You are likely to be eaten by a GRU.* (Figure from Chung, et al. 2014)

*Same eqs. as Lecture 8, subscripts/superscripts as in Assignment #2.*

# Backpropagation

Multi-path to compute $\frac{\partial J}{\partial x^{(t)}}$

- Start with $\delta^{(t)} = \left(\frac{\partial J}{\partial h^{(t)}}\right)^T \quad \in \mathbb{R}^d$
- $h^{(t)} = z^{(t)} \circ h^{(t-1)} + (1 - z^{(t)}) \circ \tilde{h}^{(t)}$
- Expand chain rule into sum (*a.k.a. product rule*):

$$\begin{aligned} \frac{\partial J}{\partial x^{(t)}} &= \frac{\partial J}{\partial h^{(t)}} \cdot \left[ z^{(t)} \circ \frac{\partial h^{(t-1)}}{\partial x^{(t)}} + \frac{\partial z^{(t)}}{\partial x^{(t)}} \circ h^{(t-1)} \right] \\ &+ \frac{\partial J}{\partial h^{(t)}} \cdot \left[ (1 - z^{(t)}) \circ \frac{\partial \tilde{h}^{(t)}}{\partial x^{(t)}} + \frac{\partial (1 - z^{(t)})}{\partial x^{(t)}} \circ \tilde{h}^{(t)} \right] \end{aligned}$$

# It gets (a little) better

Multi-path to compute $\frac{\partial J}{\partial x^{(t)}}$

- Drop terms that don't depend on $x^{(t)}$:

$$\begin{aligned}
\frac{\partial J}{\partial x^{(t)}} &= \frac{\partial J}{\partial h^{(t)}} \cdot \left[ z^{(t)} \circ \frac{\partial h^{(t-1)}}{\partial x^{(t)}} + \frac{\partial z^{(t)}}{\partial x^{(t)}} \circ h^{(t-1)} \right] \\
&\quad + \frac{\partial J}{\partial h^{(t)}} \cdot \left[ (1 - z^{(t)}) \circ \frac{\partial \tilde{h}^{(t)}}{\partial x^{(t)}} + \frac{\partial (1 - z^{(t)})}{\partial x^{(t)}} \circ \tilde{h}^{(t)} \right] \\
&= \frac{\partial J}{\partial h^{(t)}} \cdot \left[ \frac{\partial z^{(t)}}{\partial x^{(t)}} \circ h^{(t-1)} + (1 - z^{(t)}) \circ \frac{\partial \tilde{h}^{(t)}}{\partial x^{(t)}} \right] \\
&\quad - \frac{\partial J}{\partial h^{(t)}} \frac{\partial z^{(t)}}{\partial x^{(t)}} \circ \tilde{h}^{(t)}
\end{aligned}$$

# Almost there!

Multi-path to compute $\frac{\partial J}{\partial x^{(t)}}$

- Now we really just need to compute two things:
- Output gate:

$$\frac{\partial z^{(t)}}{\partial x^{(t)}} = z^{(t)} \circ (1 - z^{(t)}) \circ W_z$$

- Candidate $\tilde{h}$:

$$\begin{aligned} \frac{\partial \tilde{h}^{(t)}}{\partial x^{(t)}} = & \quad (1 - (\tilde{h}^{(t)})^2) \circ W \\ & + \quad (1 - (\tilde{h}^{(t)})^2) \circ \frac{\partial r^{(t)}}{\partial x^{(t)}} \circ U h^{(t-1)} \end{aligned}$$

- Ok, I lied - there's a third.
- Don't forget to check all paths!

# Almost there!

Multi-path to compute $\frac{\partial J}{\partial x^{(t)}}$

- Now we really just need to compute two things:
- Output gate:

$$\frac{\partial z^{(t)}}{\partial x^{(t)}} = z^{(t)} \circ (1 - z^{(t)}) \circ W_z$$

- Candidate $\tilde{h}$:

$$\begin{aligned} \frac{\partial \tilde{h}^{(t)}}{\partial x^{(t)}} = & \quad (1 - (\tilde{h}^{(t)})^2) \circ W \\ & + \quad (1 - (\tilde{h}^{(t)})^2) \circ \frac{\partial r^{(t)}}{\partial x^{(t)}} \circ U h^{(t-1)} \end{aligned}$$

- Ok, I lied - there's a third.
- Don't forget to check all paths!

# Almost there!

Multi-path to compute $\frac{\partial J}{\partial x^{(t)}}$

- Now we really just need to compute two things:
- Output gate:

$$\frac{\partial z^{(t)}}{\partial x^{(t)}} = z^{(t)} \circ (1 - z^{(t)}) \circ W_z$$

- Candidate $\tilde{h}$:

$$\begin{aligned} \frac{\partial \tilde{h}^{(t)}}{\partial x^{(t)}} = & \quad (1 - (\tilde{h}^{(t)})^2) \circ W \\ & + \quad (1 - (\tilde{h}^{(t)})^2) \circ \frac{\partial r^{(t)}}{\partial x^{(t)}} \circ U h^{(t-1)} \end{aligned}$$

- Ok, I lied - there's a third.
- Don't forget to check all paths!

# Almost there!

Multi-path to compute $\frac{\partial J}{\partial x^{(t)}}$

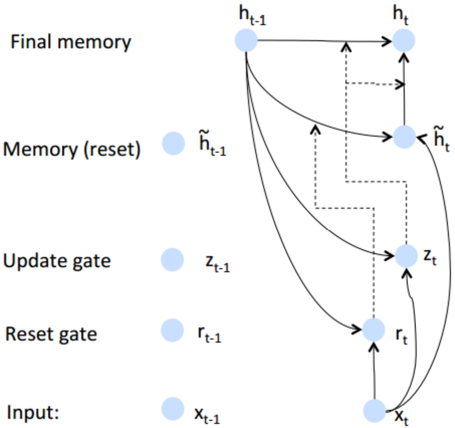
- Last one:

$$\frac{\partial r^{(t)}}{\partial x^{(t)}} = r^{(t)} \circ (1 - r^{(t)}) \circ W_r$$

- Now we can just add things up!
- (I'll spare you the pain...)

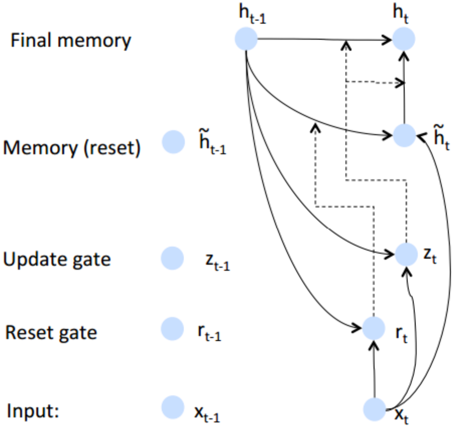
# Whew.

- Why three derivatives?
- Three arrows from $x^{(t)}$ to distinct nodes
- Four paths total ($\frac{\partial z^{(t)}}{\partial x^{(t)}}$ appears twice)



# Whew.

- GRUs are complicated
- All the pieces are simple
- Same matrix gradients that you've seen before



# Summary

- Check your dimensions!
- Write error vectors $\delta$; just parentheses around chain rule
- Combine simple operations to make complex network
  - Matrix-vector product
  - Activation functions (tanh, sigmoid, softmax)

# Good luck on Wednesday!

May the 4th be with you!

# CS 224d Midterm Review (word vectors)

Peng Qi

May 5, 2015

# Outline

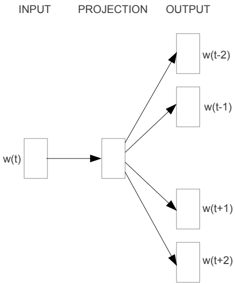
- `word2vec` and GloVe revisited
- `word2vec` with backpropagation

# Outline

- word2vec and GloVe revisited
    - Skip-gram revisited
    - (Optional) CBOW and its connection to Skip-gram
    - (Optional) word2vec as matrix factorization (conceptually)
    - GloVe v.s. word2vec
- word2vec with backpropagation

# Skip-gram

- *Task:* given a center word, predict its context words
- For each word, we have an "input vector" $v_w$ and an "output vector" $v'_w$



# Skip-gram

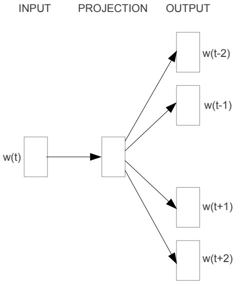
- We have seen two types of costs for an expected word given a vector prediction $r$

$$CE(w_i|r) = -\log\left(\frac{\exp(r^\top v'_{w_i})}{\sum_{j=1}^{|V|}\exp(r^\top v'_{w_j})}\right)$$

$$NEG(w_i|r) = -\log(\sigma(r^\top v'_{w_i})) - \sum_{k=1}^{K}\log(\sigma(-r^\top v'_{w_k}))$$

In the case of skip-gram, the vector prediction $r$ is just the "input vector" of the center word, $v_{w_i}$.

$\sigma(\cdot)$ is the sigmoid (logistic) function.



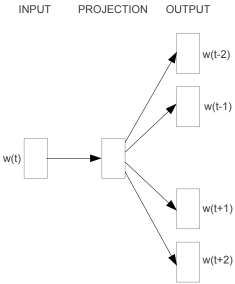
# Skip-gram

- Now we have all the pieces of skip-gram, the cost for a context window $[w_{i-C}, \cdots, w_{i-1}, w_i, w_{i+1}, \cdots, w_{i+C}]$ is ($w_i$ is the center word)

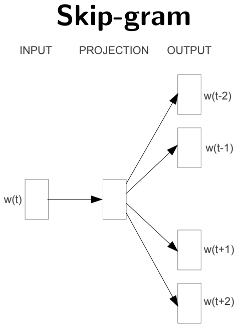
$$J_{\mathrm{skip-gram}}([w_{i-C}, \cdots, w_{i+C}]) = \sum_{i-C \le j \le i+C, i \ne j} F(w_j | v_{w_i})$$

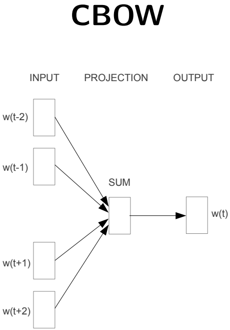
  where $F$ is one of the cost functions we defined in the previous slide.

- You might ask: but why are we introducing so many notations?



# Skip-gram v.s. CBOW




| | Skip-gram | CBOW |
|---|---|---|
| Task | Center word $\rightarrow$ Context | Context $\rightarrow$ Center word |
| $r$ | $v_{w_i}$ | $f(v_{w_{i-C}}, \cdots, v_{w_{i-1}}, v_{w_{i+1}}, \cdots, v_{w_{i+C}})$ |

All `word2vec` figures are from http://arxiv.org/pdf/1301.3781.pdf

# `word2vec` as matrix factorization (conceptually)

- Matrix factorization

$$\begin{bmatrix} & M & \end{bmatrix}_{n\times n} \approx \begin{bmatrix} A^\top \end{bmatrix}_{n\times k} \begin{bmatrix} & B & \end{bmatrix}_{k\times n}$$

$$M_{ij} \approx a_i^\top b_j$$

- Imagine $M$ is a matrix of counts for events co-occurring, but we only get to observe the co-occurrences one at a time. E.g.

$$M = \begin{bmatrix} 1 & 0 & 4 \\ 0 & 0 & 2 \\ 1 & 3 & 0 \end{bmatrix}$$

but we only see
*(1,1), (2,3), (3,2), (2,3), (1,3), ...*

# `word2vec` as matrix factorization (conceptually)

$$M_{ij} \approx a_i^\top b_j$$

- Whenever we see a pair $(i, j)$ co-occur, we try to increasing $a_i^\top b_j$
- We also try to make all the other inner-products smaller to account for pairs never observed (or unobserved yet), by decreasing $a_{\neg i}^\top b_j$ and $a_i^\top b_{\neg j}$
- Remember from the lecture that the word co-occurrence matrix usually captures the semantic meaning of a word? For `word2vec` models, roughly speaking, $M$ is the windowed word co-occurrence matrix, $A$ is the output vector matrix, and $B$ is the input vector matrix.
- Why not just use one set of vectors? It's equivalent to $A = B$ in our formulation here, but less constraints is usually easier for optimization.

# GloVe v.s. `word2vec`

| | Fast training | Efficient usage of statistics | Quality affected by size of corpora | Captures complex patterns |
|---|---|---|---|---|
| Direct prediction (`word2vec`) | Scales with size of corpus | No | No* | Yes |
| GloVe | Yes | Yes | No | Yes |

* Skip-gram and CBOW are qualitatively different when it comes to smaller corpora

# Outline

- `word2vec` and GloVe revisited
- `word2vec` with backpropagation

# word2vec with backpropagation

- $$CE(w_i|r) = -\log\left(\frac{\exp(r^\top v'_{w_i})}{\sum_{j=1}^{|V|}\exp(r^\top v'_{w_j})}\right)$$

- $$\begin{aligned} CE(w_i|r) &= CE(\hat{y}, y_i) \\ \hat{y} &= \mathrm{softmax}(\theta) \\ \theta &= (V')^\top r \end{aligned}$$

- $$\delta = \frac{\partial CE}{\partial \theta} = \hat{y} - y_i$$

- $$\begin{aligned} \frac{\partial CE}{\partial V'} &= r\delta^\top \\ \frac{\partial CE}{\partial r} &= V'\delta \end{aligned}$$

Thanks for your attention
and best of luck with the mid-term!


Any questions?