Natural Language Processing

Lecture 14:
Machine Learning: Feed-forward Neural Networks, Autoencoders/embeddings, Dense networks

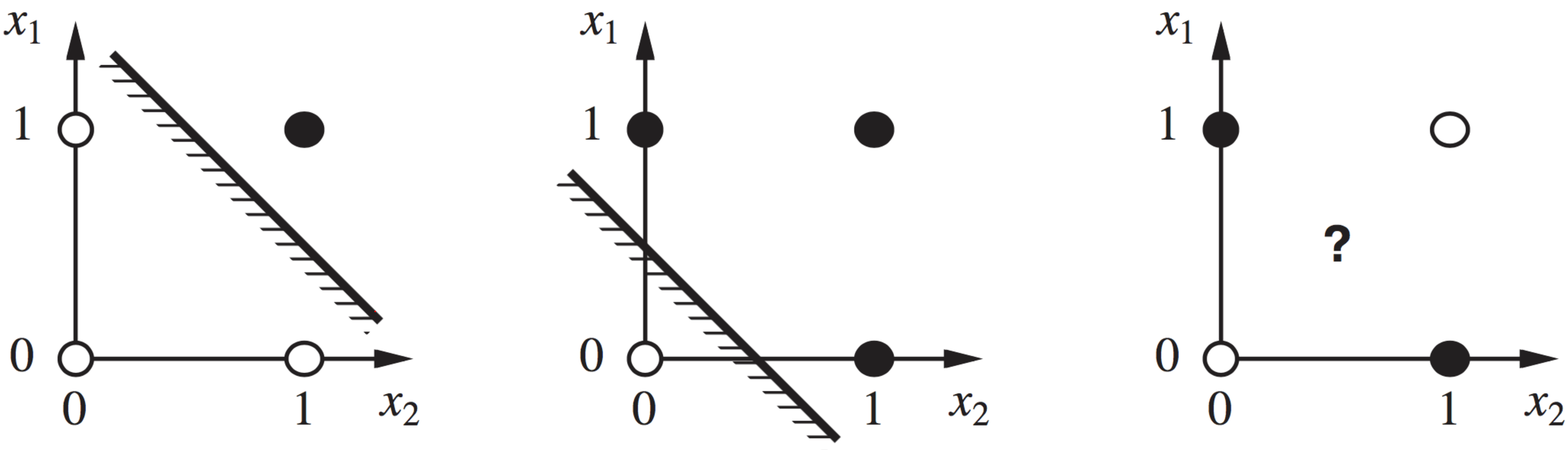
12 /7/2019

COMS W4705
Yassine Benajiba

Perceptron Expressiveness

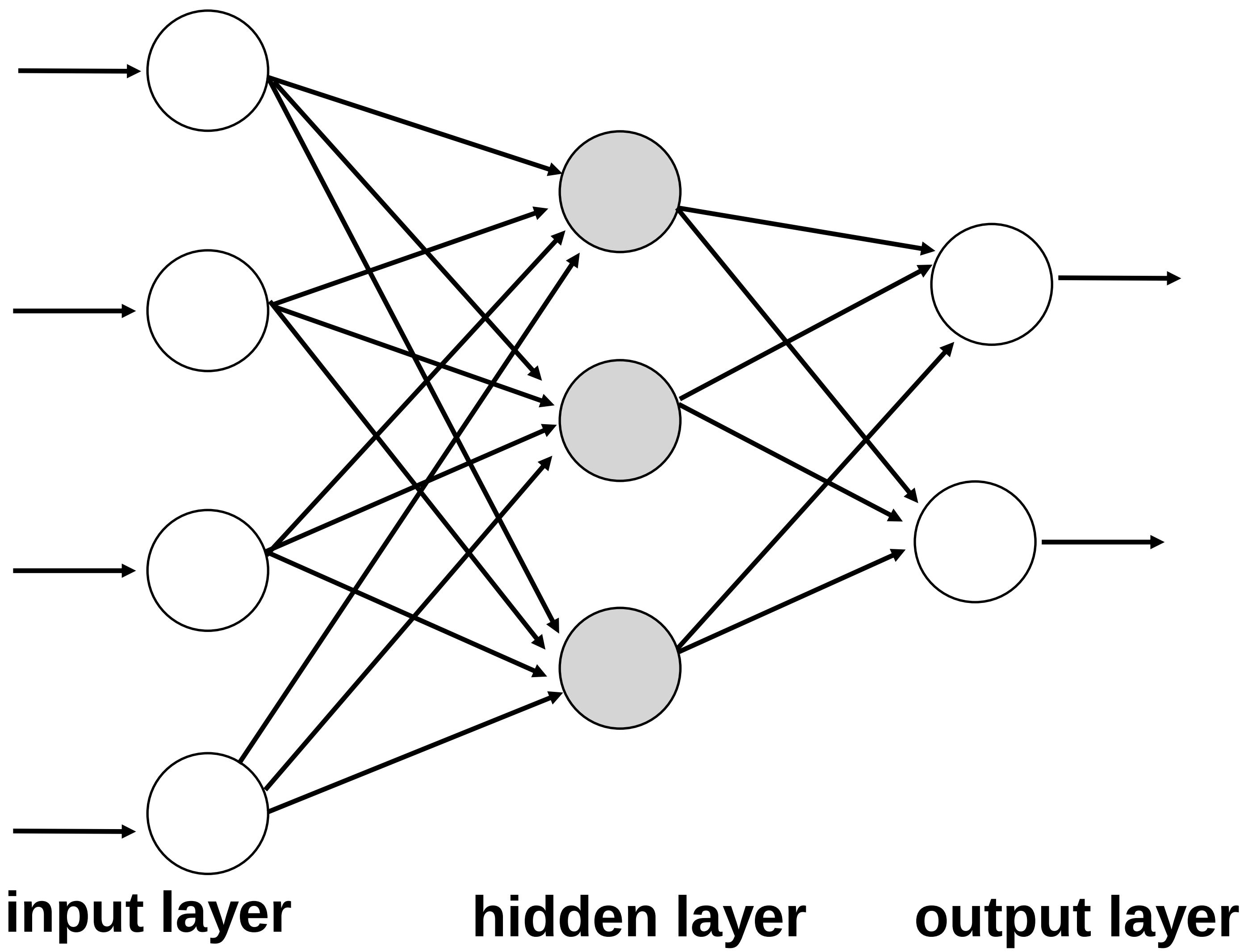
- Simple perceptron learning algorithm, starts with an arbitrary hyperplane and adjusts it using the training data.
  - Step function is not differentiable, so no closed-form solution.
- Perceptron produces a linear separator.
  - Can only learn linearly separable patterns.
- Can represent boolean functions like **and, or, not** but not the **xor** function.

The problem with *xor*



(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

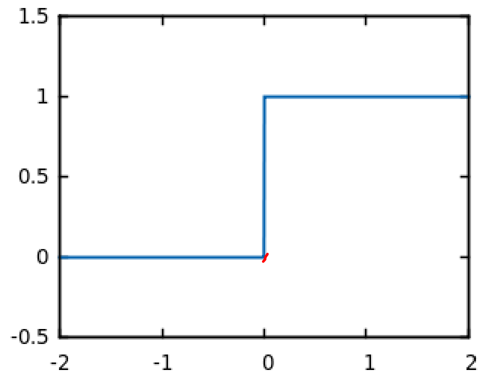
Multi-Layer Neural Networks



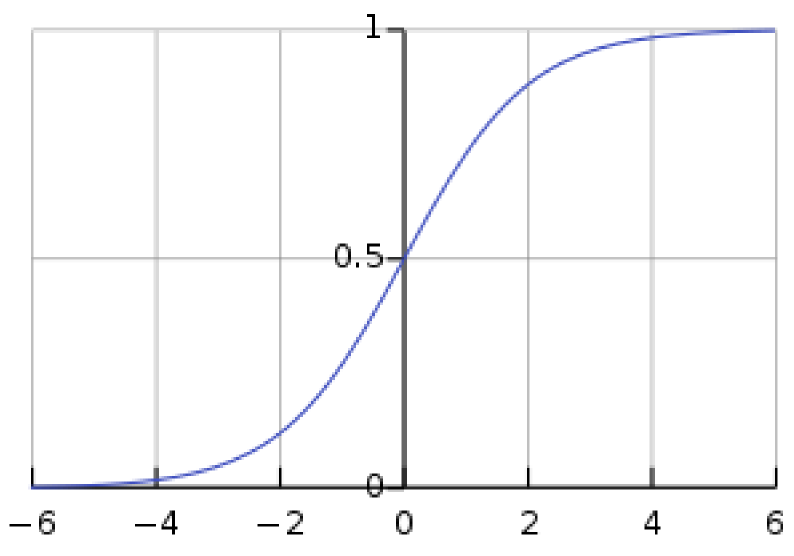
- Basic idea: represent any (non-linear) function as a composition of soft-threshold functions. This is a form of non-linear regression.
- Lippmann 1987: Two hidden layers suffice to represent any arbitrary region (provided enough neurons), even discontinuous functions!

Activation Functions

- One problem with perceptrons is that the **threshold function (step function)** is undifferentiable.



- It is therefore unsuitable for gradient descent.
- One alternative is the **sigmoid (logistic) function:**



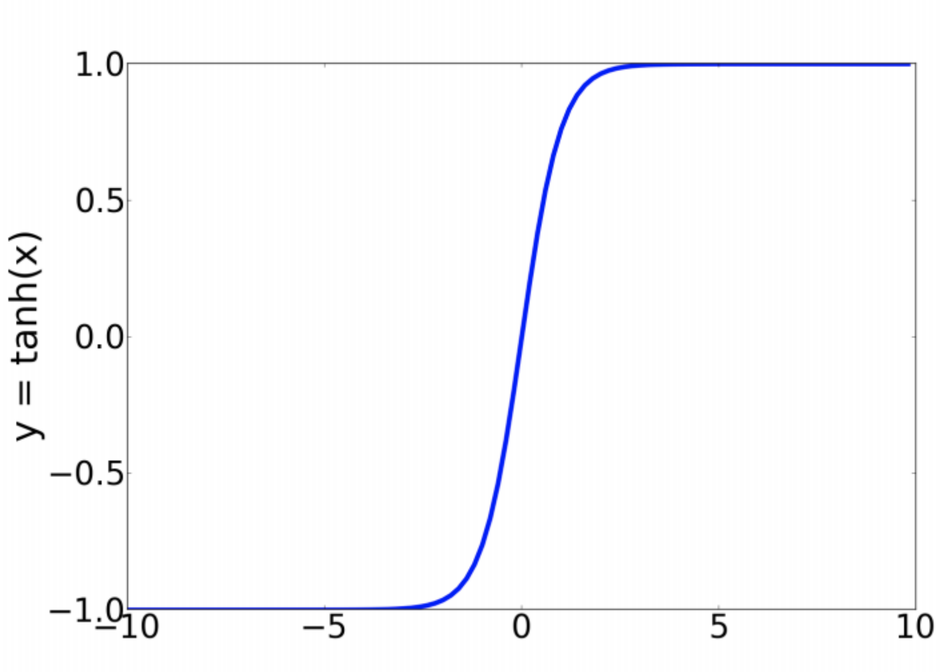
$$g(z) = \frac{1}{1 + e^{-z}}$$

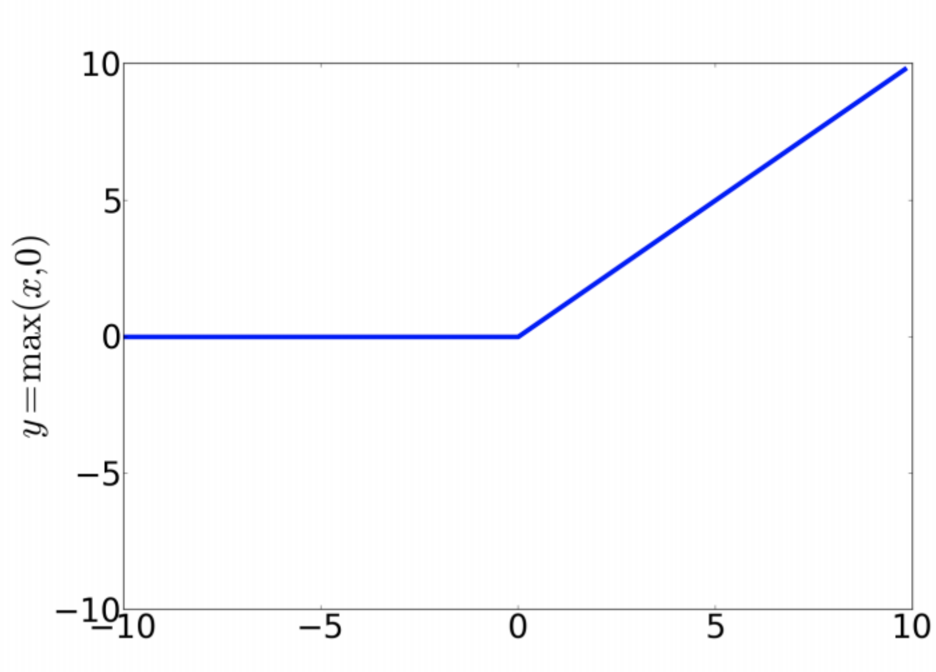
$g(z) = 0$ if $z \rightarrow -\infty$

$g(z) = 1$ if $z \rightarrow \infty$

Activation Functions

- Two other popular activation functions:



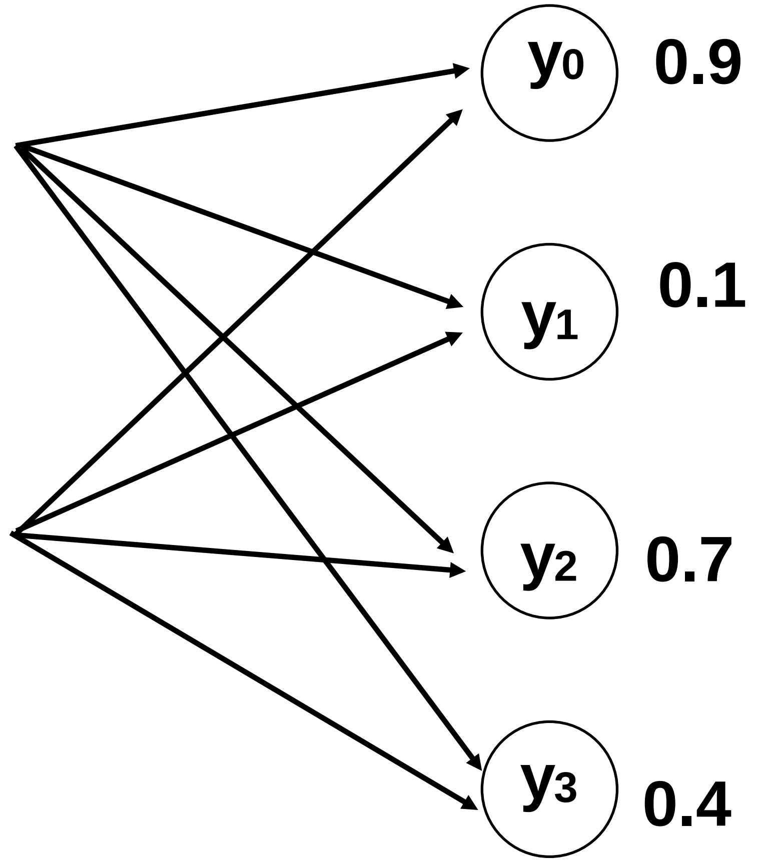


$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$relu(z) = max(z, 0)$$

Output Representation

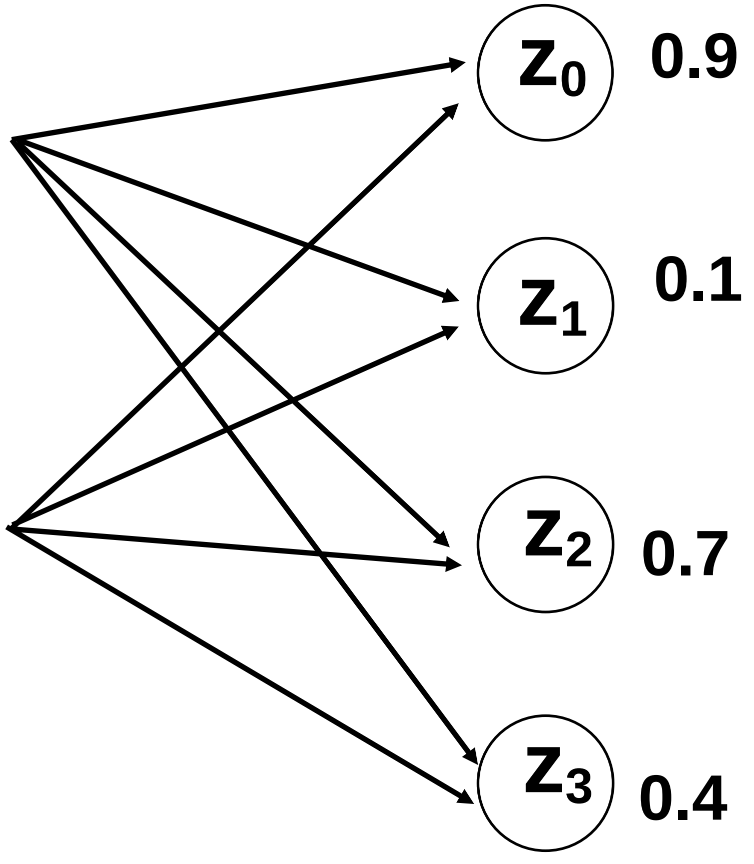
- Many NLP Problems are multi-class classification problems.
- Each output neuron represents one class. Predict the class with the highest activation.



Softmax

- We often want the activation at the output layer to represent probabilities.

- Normalize activation of each output unit by the sum of all output activations (as in log-linear models).



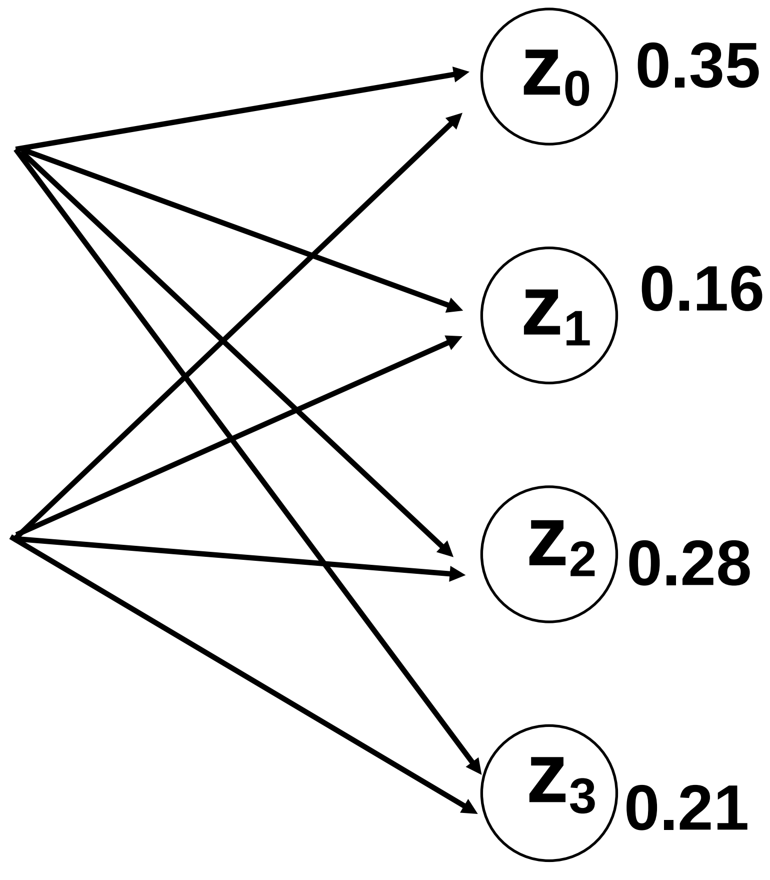
$$softmax(z_i) = \frac{exp(z_i)}{\sum_{j=1}^{k} exp(z_j)}$$

The network computes a probability

$$P(c_i | \mathbf{x}; \mathbf{w})$$

Softmax

- We often want the activation at the output layer to represent probabilities.
- Normalize activation of each output unit by the sum of all output activations (as in log-linear models).



$$softmax(z_i) = \frac{exp(z_i)}{\sum_{j=1}^{k} exp(z_j)}$$

The network computes a probability

$$P(c_i | \mathbf{x}; \mathbf{w})$$

Learning in Multi-Layer Neural Networks

- Network structure is fixed, but we want to train the weights. Assume **feed-forward** neural networks: no connections that are loops.

- **Backpropagation Algorithm:**

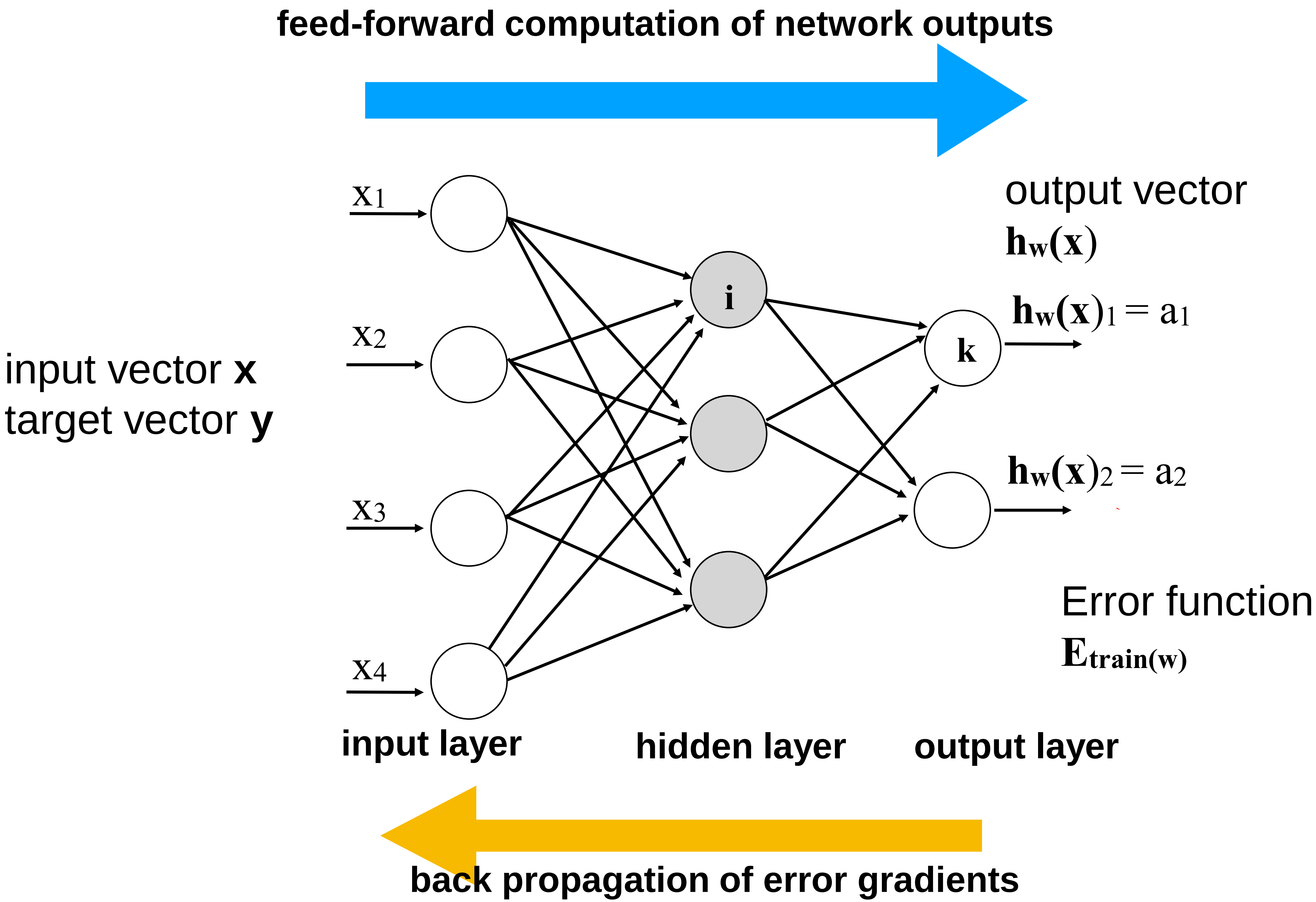
  - Given current weights, get network output and compute loss function (assume multiple outputs / a vector of outputs).

  - Can use gradient descent to update weights and minimize loss.

  - Problem: We only know how to do this for the last layer!

  - Idea: Propagate error backwards through the network.

Backpropagation



Negative Log-Likelihood

(also known as cross-entropy)

- Assume target output is a one-hot vector and *c(y)* is the target class for target **y**.

- Compute the negative log-likehood for a single example

$$Loss(\mathbf{y}, h_{\mathbf{w}}(x)) = -logP(c(\mathbf{y})|\mathbf{x};\mathbf{w})$$

- Empirical error for the entire training data:

$$E_{train}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} -logP(c(\mathbf{y}^{(i)})|\mathbf{x}^{i};\mathbf{w})$$

Stochastic Gradient Descent (for a single unit)

- Goal: Learn parameters that minimize the empirical error.

Randomly initialize $\boldsymbol{w}$

for a set number of iterations T:

shuffle training data $\mathcal{D} = (x^{(j)}, y^{(j)})|_{j=1}^{n}$

for $j = 1...N$:

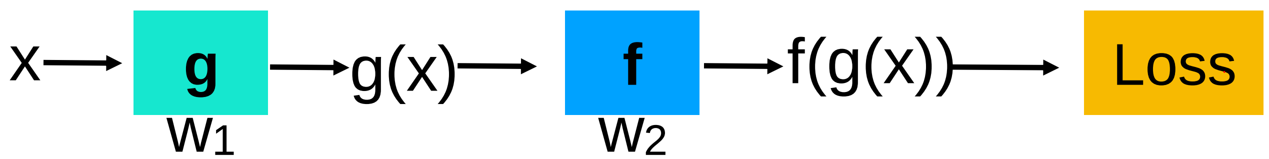
for each w_i (all weights in the network):

$$w_i \leftarrow w_i - \eta \frac{\partial}{\partial w_i} Loss(y^{(j)}, h_{\mathbf{w}}(x^{(j)}))$$

- η is the learning rate.
- It often makes sense to compute the gradient over batches of examples, instead of just one ("mini-batch").

Backpropgation

- Simplified multi-layer case (a single unit per layer):



- Stochastic Gradient Descent should perform the following update:

$$w_2 \leftarrow w_2 - \eta \frac{\partial Loss(y, f(g(x)))}{\partial w_2}$$

$$w_1 \leftarrow w_1 - \eta \frac{\partial Loss(y, f(g(x)))}{\partial w_1}$$

- Problem: How do we compute the gradient for parameters w_1 and w_2?

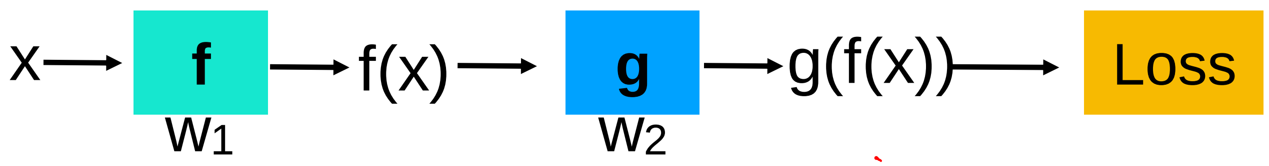
Chain Rule of Calculus

- To compute gradients for hidden units, we need to apply the chain rule of calculus:

The derivative of $f(g(x))$ is

$$\frac{df(g(x))}{dx} = \frac{df(g(x))}{dg(x)} \cdot \frac{dg(x)}{dx}$$

Backpropagation

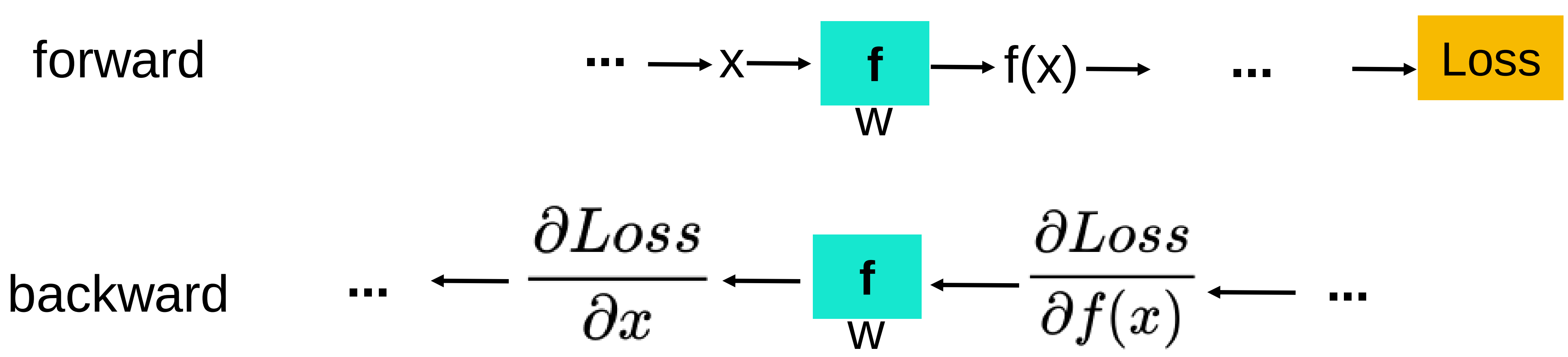


$$\frac{\partial Loss}{w_2} = \left(\frac{\partial Loss}{g(f(x))}\right)\left(\frac{\partial g(f(x))}{\partial w_2}\right)$$

$$\frac{\partial Loss}{w_1} = \left(\frac{\partial Loss}{\partial f(x)}\right)\left(\frac{\partial f(x)}{\partial w_1}\right)$$

$$= \left(\frac{\partial Loss}{g(f(x))}\right)\left(\frac{\partial g(f(x))}{\partial f(x)}\right)\left(\frac{\partial f(x)}{\partial w_1}\right)$$

Backpropagation



Assume we know $\frac{\partial Loss}{\partial f(x)}$

We want to compute $\frac{\partial Loss}{\partial x}$ to propagate it back.

and $\frac{\partial Loss}{\partial w}$ (for the weight update)

Backpropagation

forward $\cdots \longrightarrow x \longrightarrow \underset{w}{\boxed{f}} \longrightarrow f(x) \longrightarrow \cdots \longrightarrow \boxed{\text{Loss}}$

backward $\cdots \longleftarrow \dfrac{\partial Loss}{\partial x} \longleftarrow \underset{w}{\boxed{f}} \longleftarrow \dfrac{\partial Loss}{\partial f(x)} \longleftarrow \cdots$

$$\frac{\partial Loss}{\partial x} = \left(\frac{\partial Loss}{\partial f(x)}\right)\boxed{\left(\frac{\partial f(x)}{\partial x}\right)}$$

$$\frac{\partial Loss}{\partial w} = \left(\frac{\partial Loss}{\partial f(x)}\right)\boxed{\left(\frac{\partial f(x)}{\partial w}\right)}$$

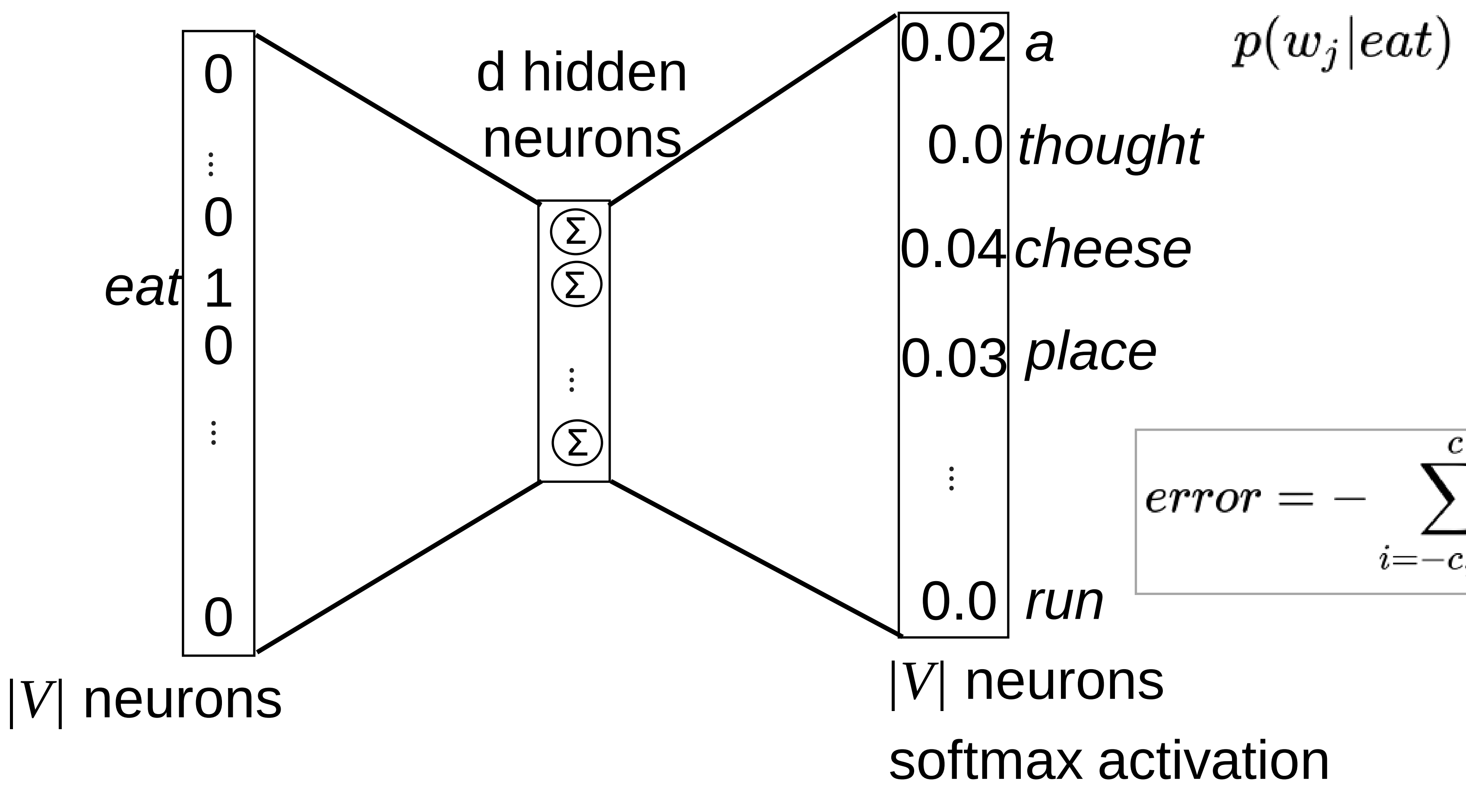
to compute these we have to know the derivate of the function f

Autoencoders
Embeddings
(Word level semantics)

Skip-Gram Model

- Input:
  A single word in one-hot representation.

- Output: probability to see any single word as a context word.

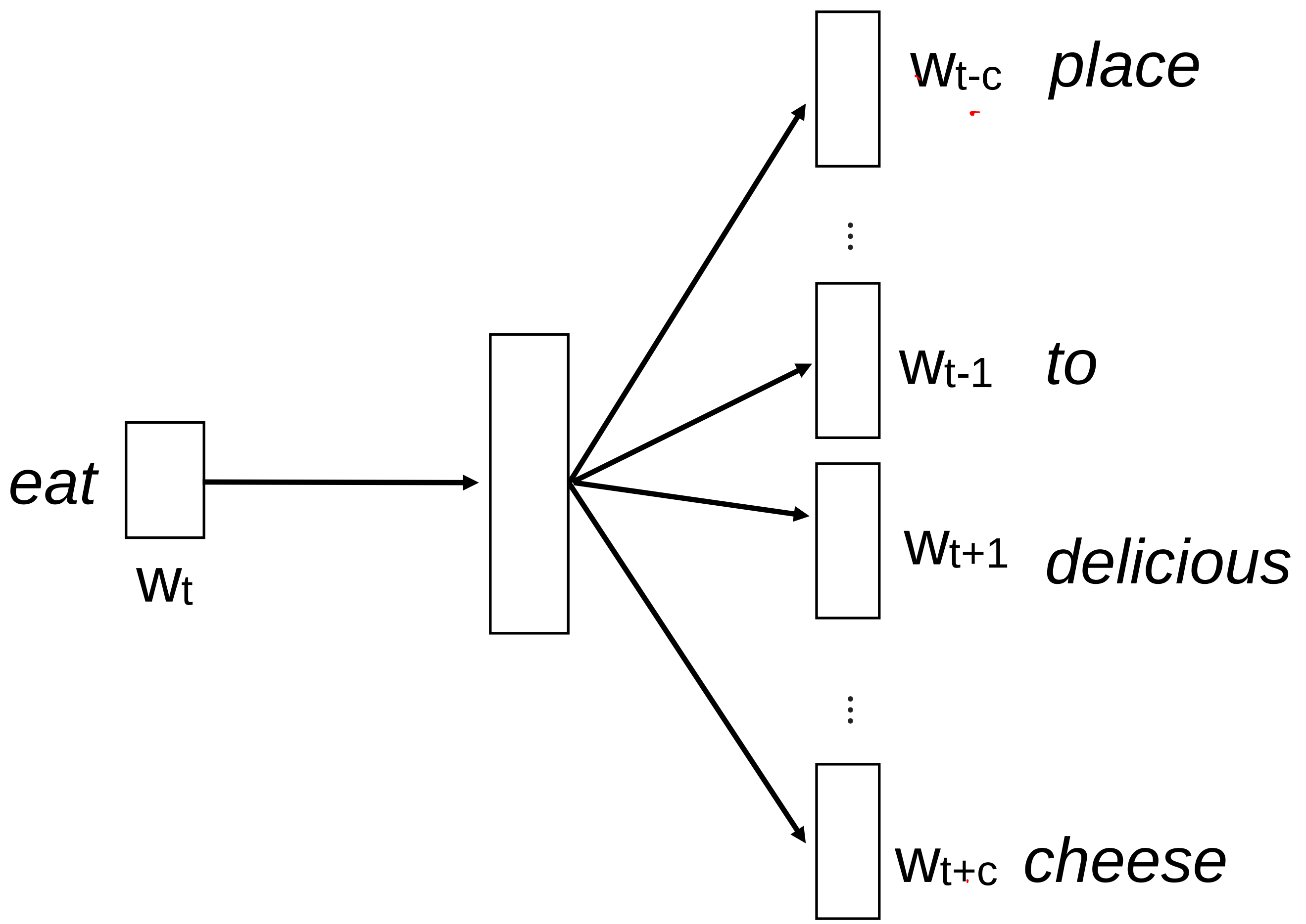


$$error = -\sum_{i=-c, i\neq 0}^{c} \log p(w_{t+i}|w_t)$$

- Softmax function normalizes the activation of the output neurons to sum up to 1.0.

Skip-Gram Model

- Compute error with respect to each context word.



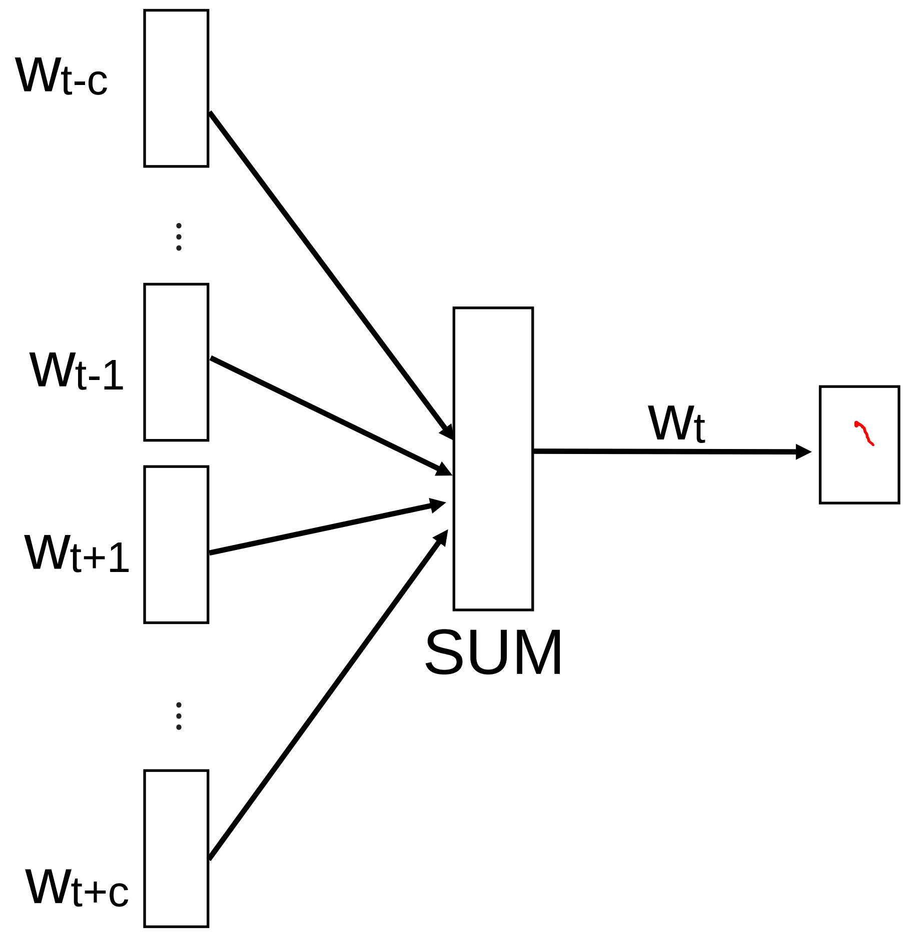
...a place to ***eat*** *delicious cheese .*

(eat, place)
(eat, to)
(eat, delicious)
(eat,cheese)

- Combine errors for each word, then use combined error to update weights using back-propagation.

$$error = -\sum_{i=-c, i\neq 0}^{c} \log p(w_{t+i}|w_t)$$

Continuous Bag-of-Words Model (CBOW)

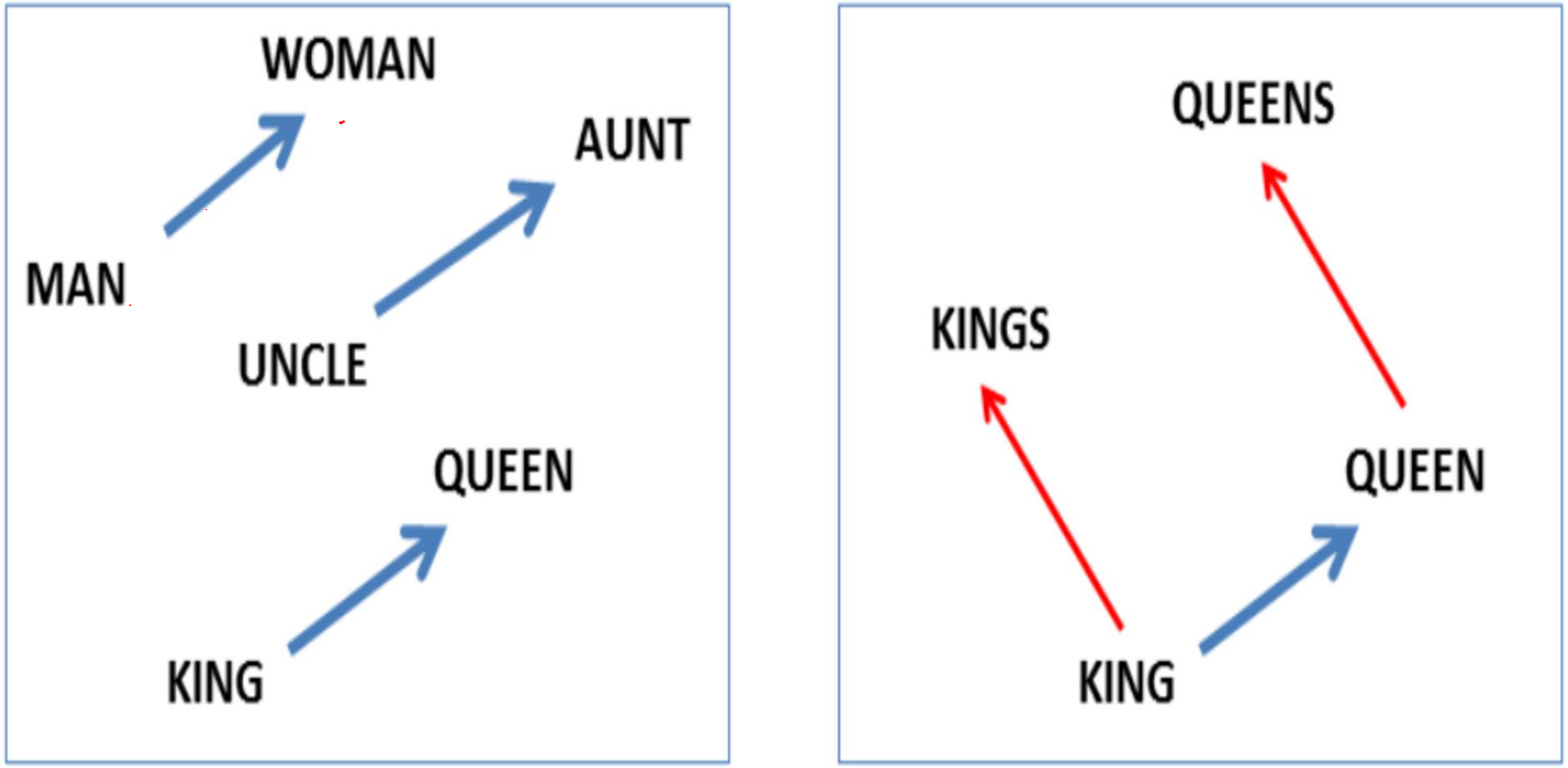


- Input: Context words. Averaged in the hidden layer.
- Output: Probability that each word is the target word.

Embeddings are Magic

(Mikolov 2016)

vector(‘king’) - vector(‘man’) + vector(‘woman’) ≈ vector(‘queen’)



Application: Word Pair Relationships

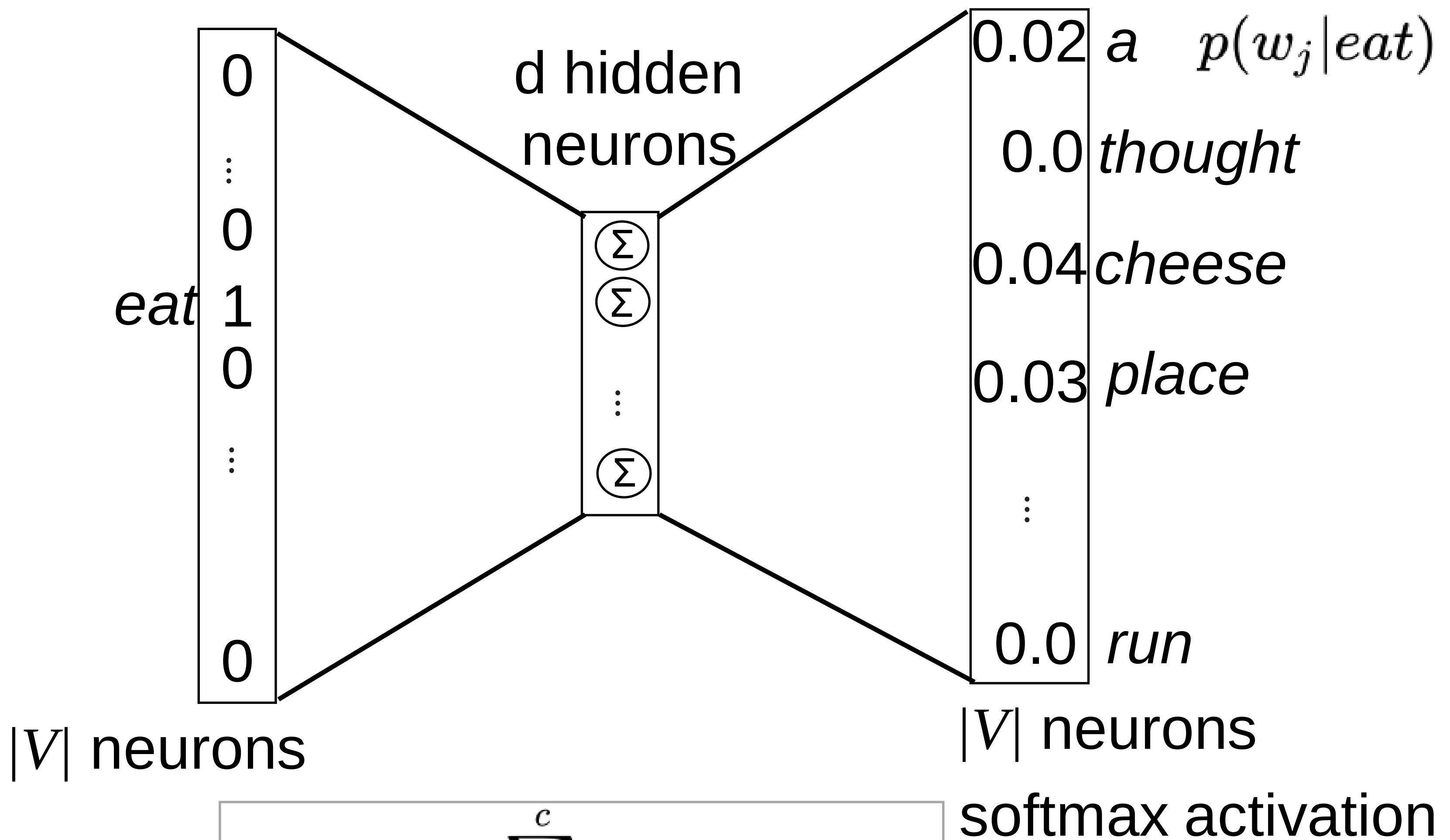
Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Using Word Embeddings

- Word2Vec:
  - https://code.google.com/archive/p/word2vec/
- GloVe: Global Vectors for Word Representation
  - https://nlp.stanford.edu/projects/glove/
- Can either use pre-trained word embeddings or train them on a large corpus.

Word embeddings



$$error = -\sum_{i=-c, i\neq 0}^{c} \log p(w_{t+i}|w_t)$$

Word embeddings

Pros

- **Groups semantically similar words together**
- **A simple way to measure similarity**
- **Great approach to better deal with unseen words in the training**

Cons

- **Doesn't make a difference between function and content words**
- **Only one representation for polysemous words**
- **Non interpretable semantic dimensions**

How can we build a sentence representation using word-level distributional representations?

Acknowledgments

- Some slides by Chris Kedzie