Latest Advances in NLP

Antske Fokkens & Pia Sommerauer

Tuesday 2 July 2019

Recap

Machine Learning

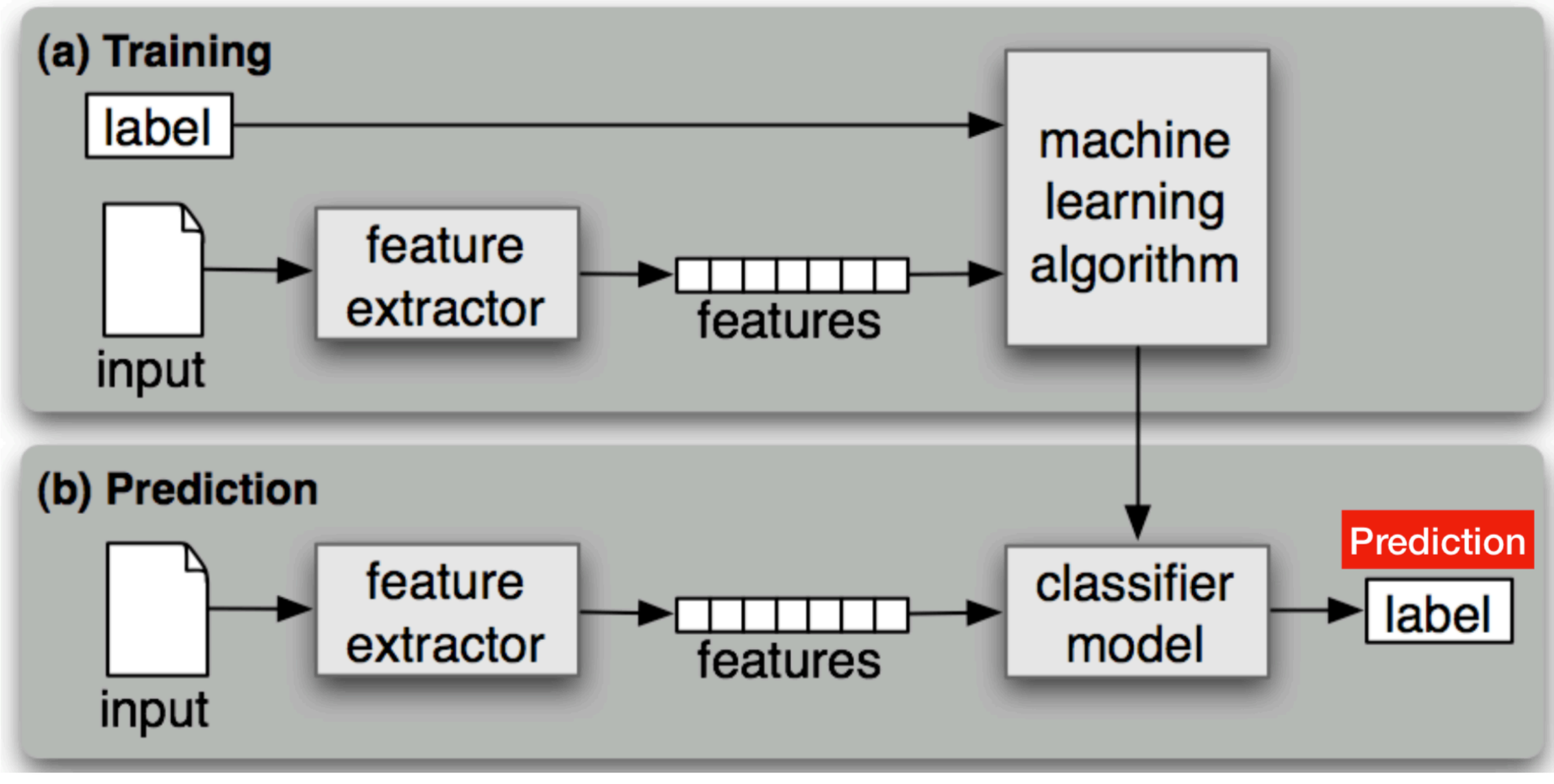


Image taken from slides Piek Vossen (zie lecture machine learning)

Logistic regression

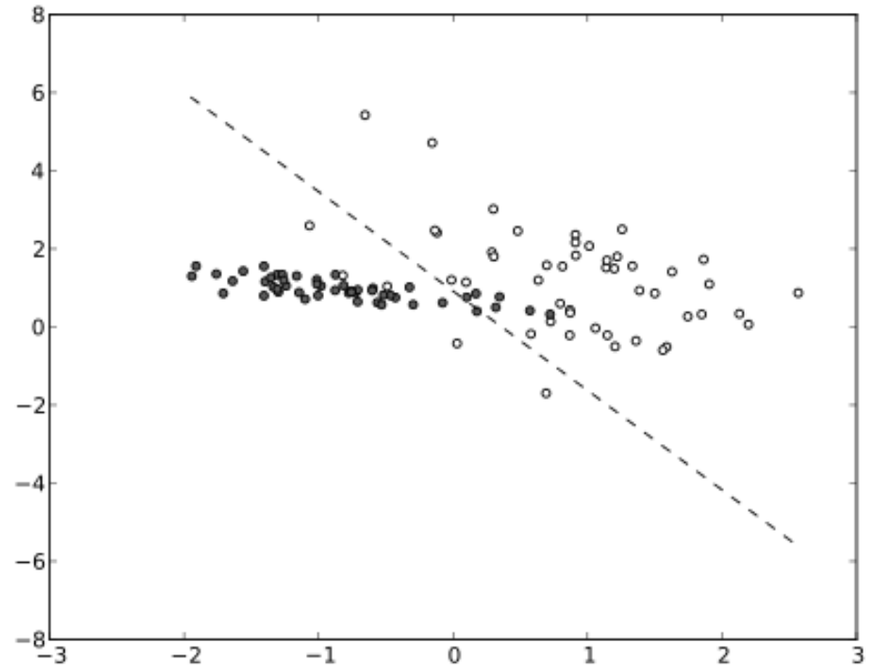
- Classifier that aims to optimize (y|X)
- Builds upon principles of linear regression, but instead of predicting values, it aims to optimize separation lines

Translating to numbers

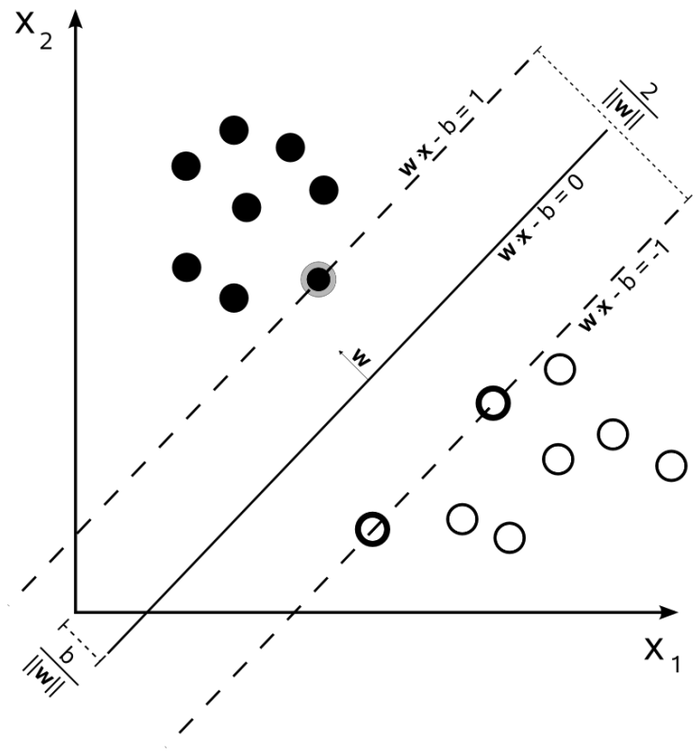
- Methods directly related to statistics (distribution & correlation)

- Many other forms: geometric representation & identify lines

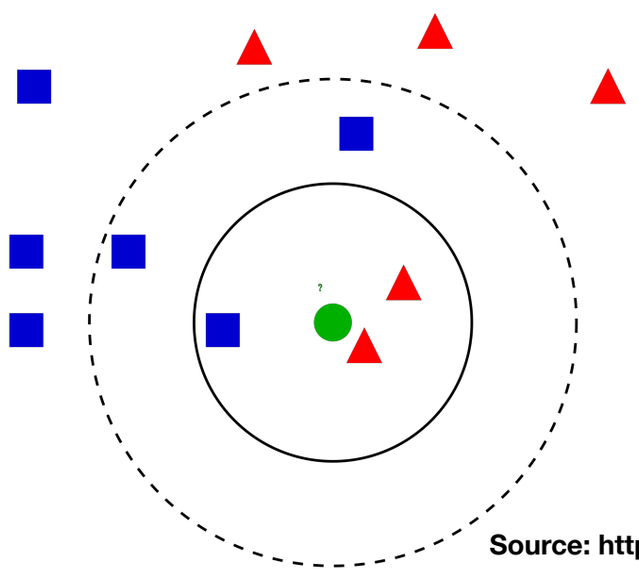
ML methods (examples)



Source: https://commons.wikimedia.org/wiki/File:Linear-svm-scatterplot.svg



Source: https://en.m.wikipedia.org/wiki/File:Svm_max_sep_hyperplane_with_margin.png



Source: https://commons.wikimedia.org/wiki/File:KnnClassification.svg

Linear Regression

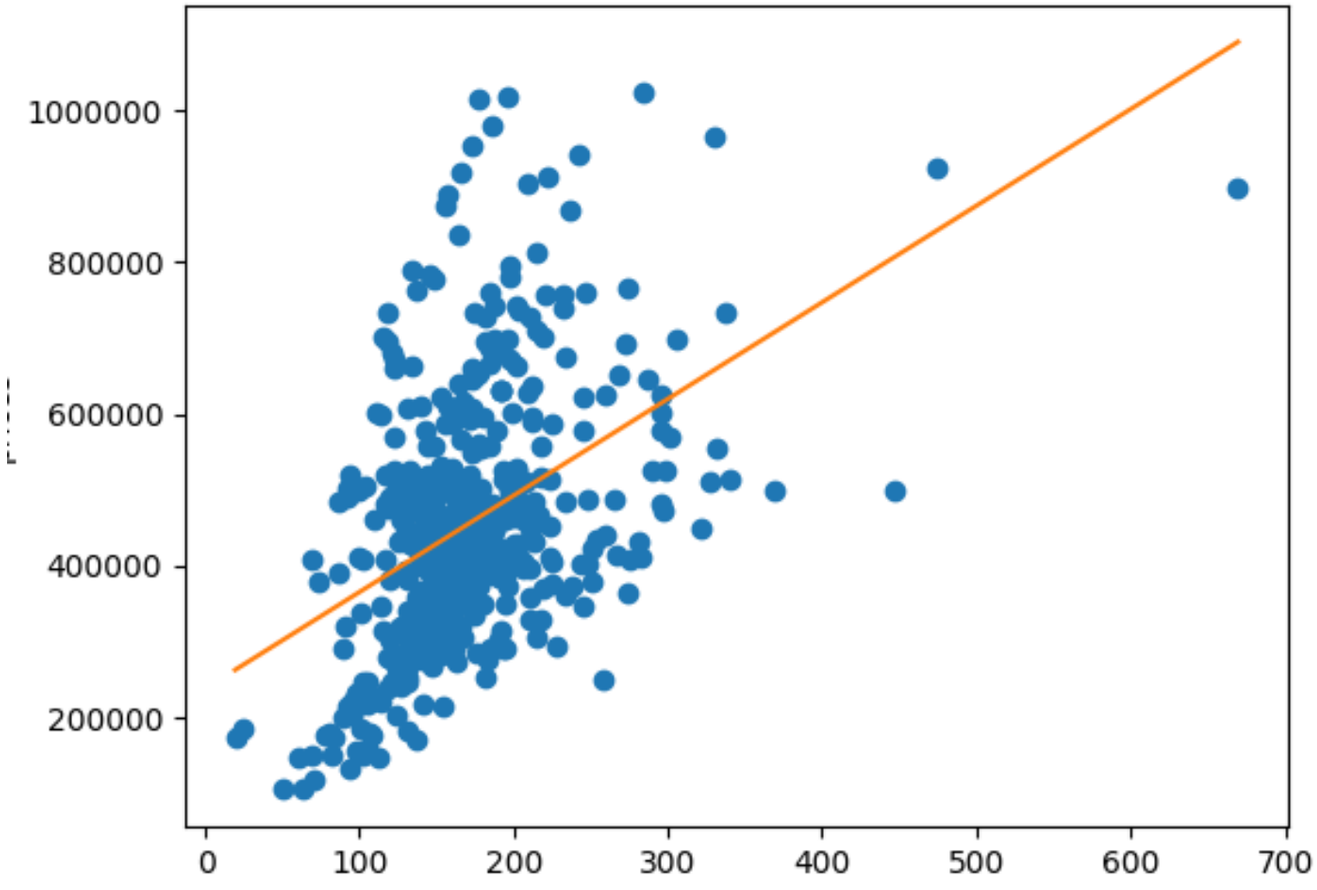
- *Predict the best Θ so as to make the least number of errors on the test data*

- We want to minimize the errors of *f(x)* against the true values of *y —> use of the training data*

- To obtain the best values for the parameters, i.e parameter estimation, we will apply a *cost function*

Predicting house price

- Information about the house:
  - Size (in m2)



Linear Regression

- Cost function = squared error function
- We can compute the error of *f(x)* against *y* for any point x_i

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

- m = number of training examples
- Θ_0 assumed to be 0

Linear Regression

- For each value Θ_1 of $f(x)$ there is a different value for $J(\Theta_1)$

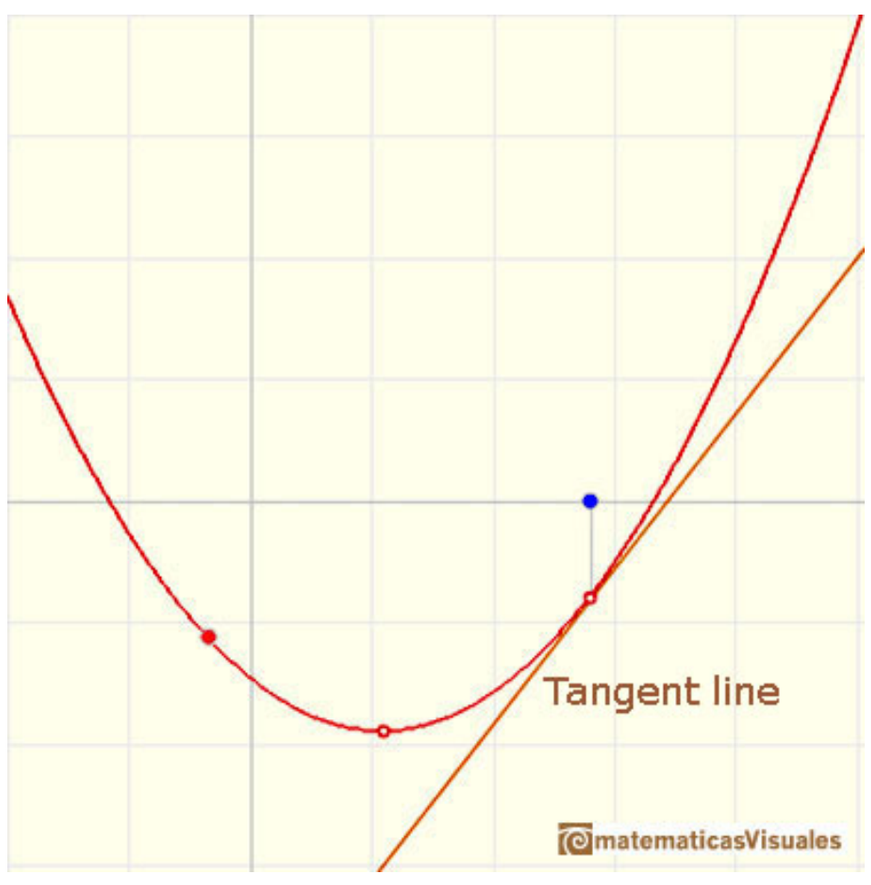
- Our optimization problem aims at minimizing $J(\Theta_1)$

- We can try all possible values of Θ_1 to identify the one which minimise $J(\Theta_1)$ i.e. get the straight line which best fit the data

- To do this we can use Gradient Descent

LR: Gradient Descent

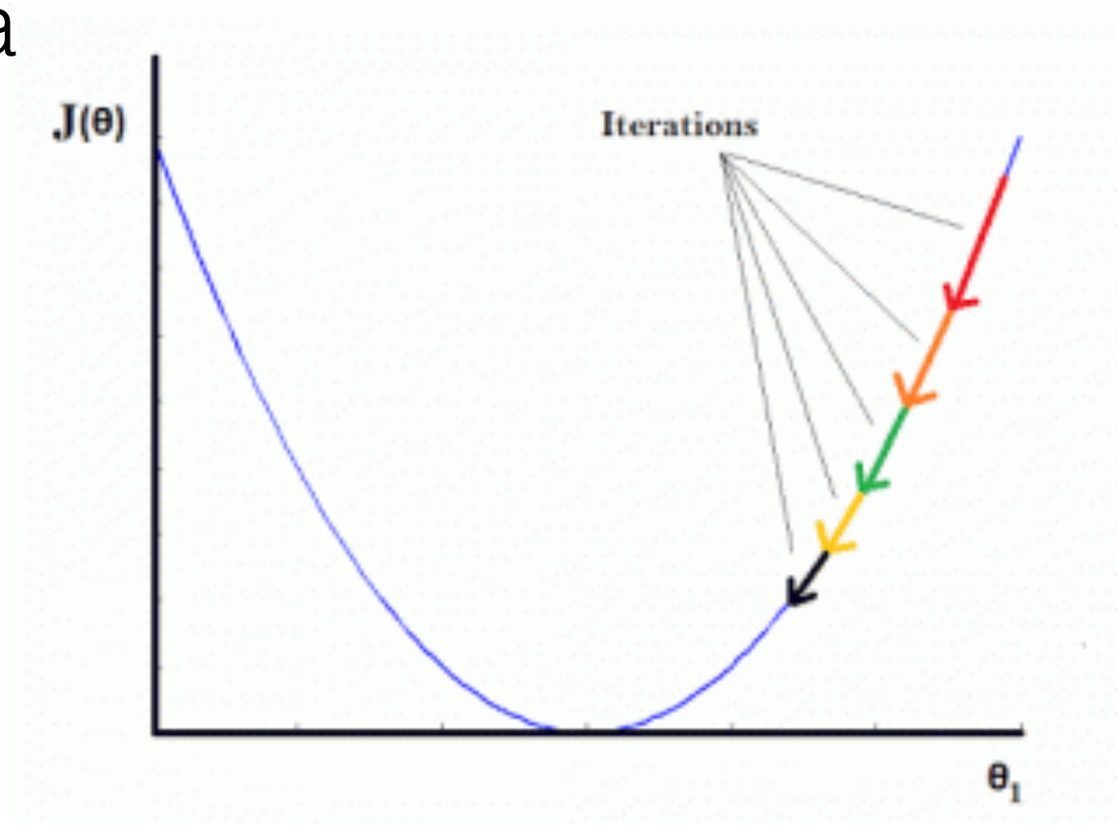
- Derivative: $\frac{\delta}{\delta\Theta_j} J(\Theta_0, \Theta_1)$
- Create a tangent on the cost function; the slope of the tangent tells us how/where to find he minimum



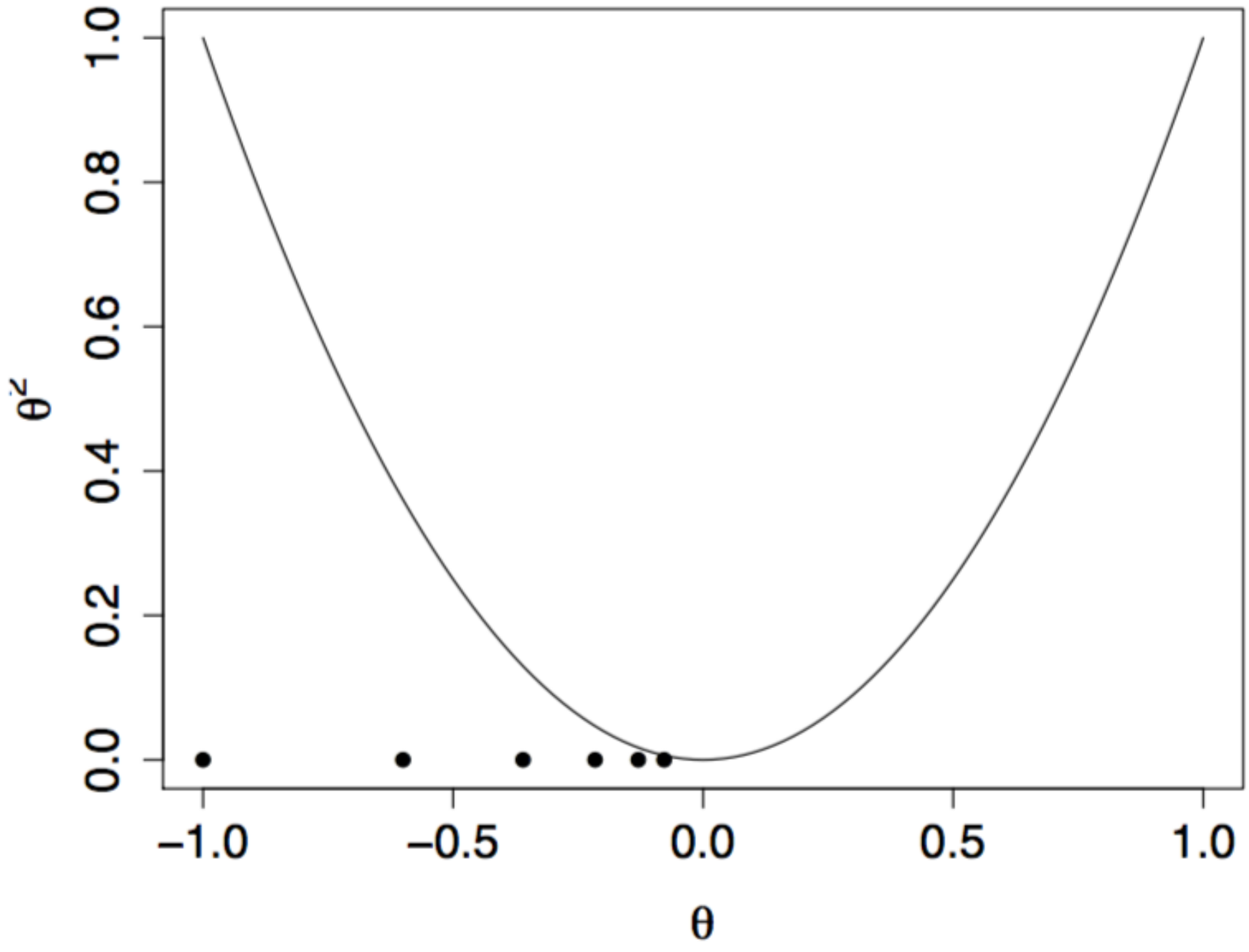
LR: Gradient Descent

- α is the learning rate, i.e. how "fast" we find the local minimum

- In Linear Regression the cost function is always a convex function where the local optima equals to the global optima



LR: Gradient Descent



Logistic Regression

- Hypothesis

$$f(x) = g(\Theta^T x)$$

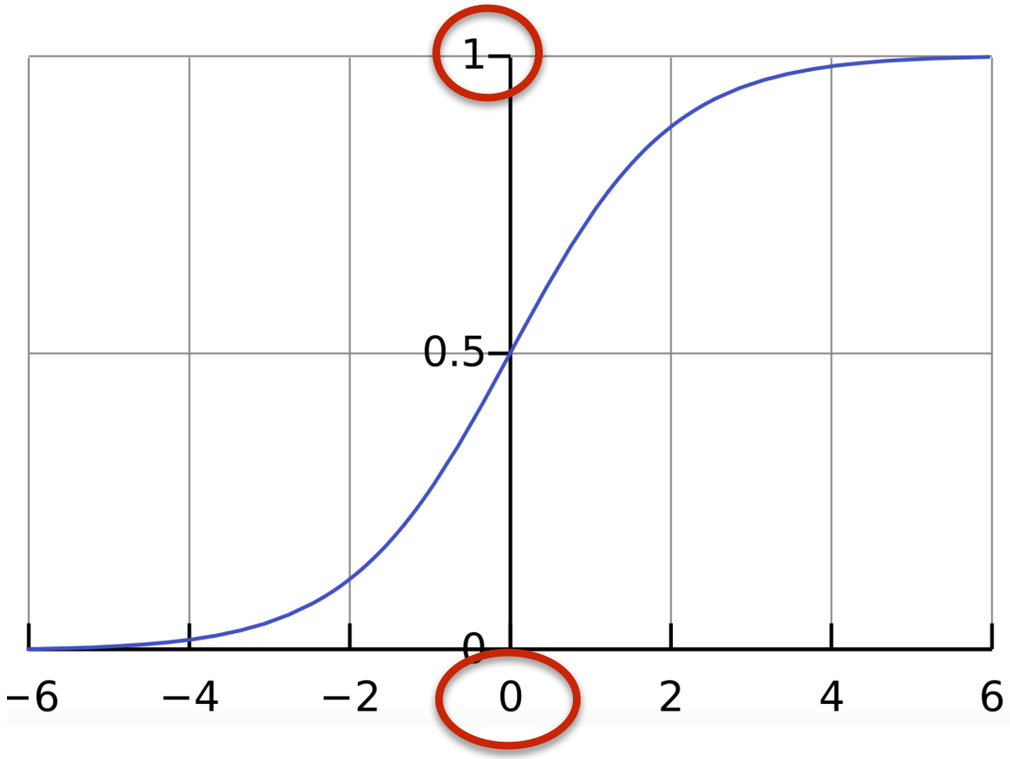
$$g(z) = \frac{1}{1+e^{-z}}$$ sigmoid / logistic function

$$f(x) = g(\Theta^T x) = \frac{1}{1+e^{-\theta^T X}}$$

Logistic Regression

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

$e =$ mathematical constant (2,718)



- $0 \leq y \leq 1$
- Allows modeling probabilities

Logistic Regression

$$P(Y=1 \mid x) = 1 - g(-\Theta^T x)$$

$$P(Y=0 \mid x) = -g(-\Theta^T x)$$

$$P(Y=1 \mid x) = \frac{e^{(\Theta_0 x_0 + \Theta_1 x_1)}}{1 + e^{(\Theta_0 x_0 + \Theta_1 x_1)}}$$

$$P(Y=0 \mid x) = \frac{1}{1 + e^{(\Theta_0 x_0 + \Theta_1 x_1)}}$$

Logistic Regression

- We still have to determine the parameters/weights Θ to solve our function

- Different interpretation of the output of *f(x):*

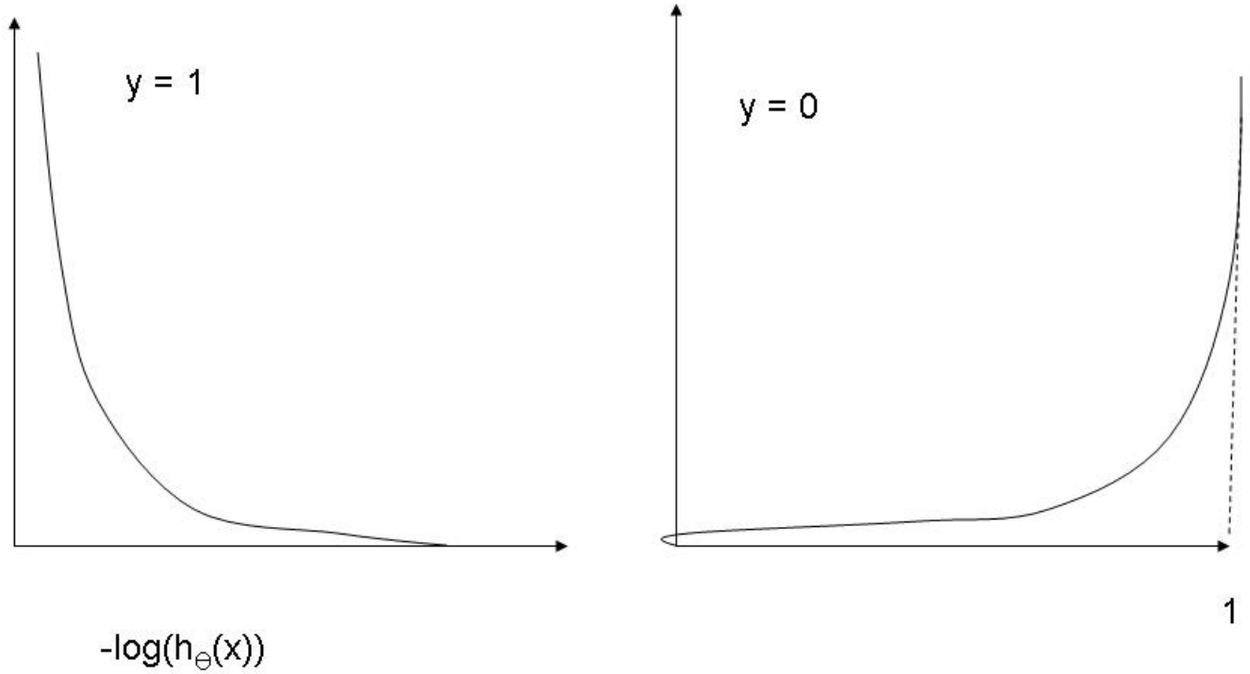
  - *f(x)* gives us the *estimated probability* of Y = 1 given input *x*

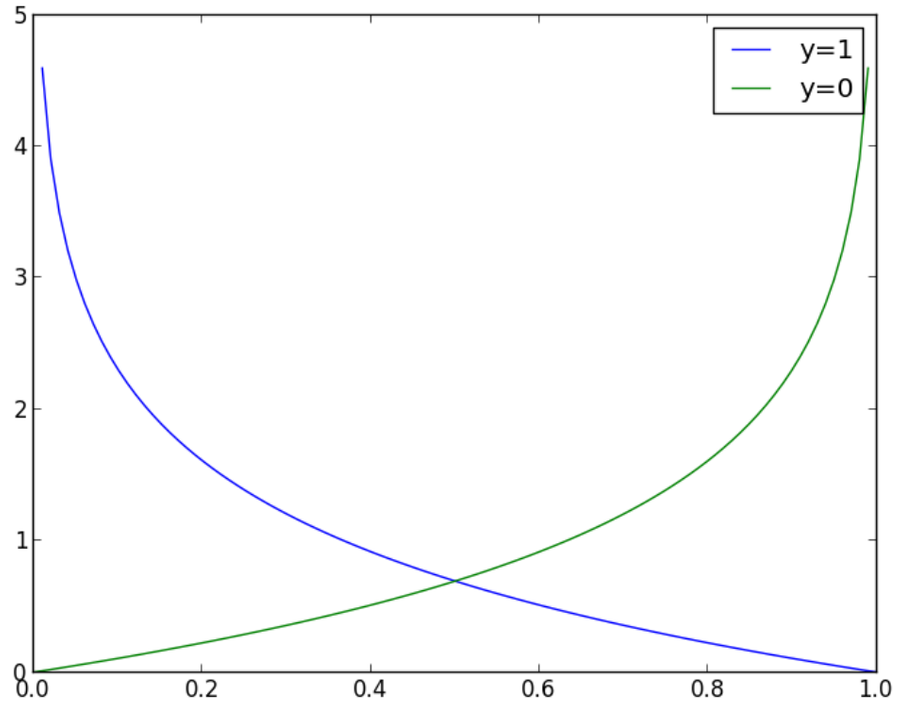
Logistic Regression

- Cost function (for a single variable):

$$J(\theta) = \frac{1}{m}\sum_{1}^{m} Cost(f(x), y)$$

$$Cost(f(x), y) = \begin{cases} -\log(f(x)) & if \quad y = 1 \\ -\log(1 - f(x)) & if \quad y = 0 \end{cases}$$





Logistic regression

- Cost function searches for a global minimum given all evidence

- Impact of correlating features is captured by adjusted weights

Feature Representation

Input representation

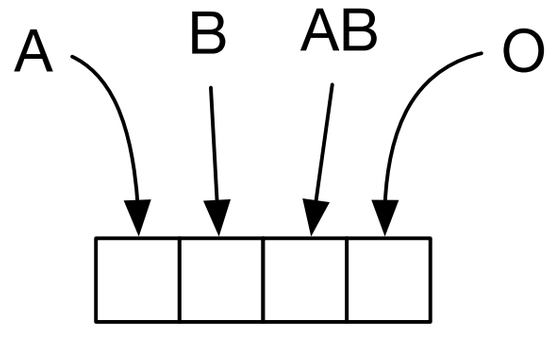
- Many Machine Learning Classifiers use **input representations** that correspond to vectors

- How can we translate linguistic information to space?

Representations

- A number in one dimension, e.g. size of houses: represent size as number (e.g. square meters)

- Different dimensions allocated to various values, e.g. blood type of a person: represent value in a 4-dimensional vector



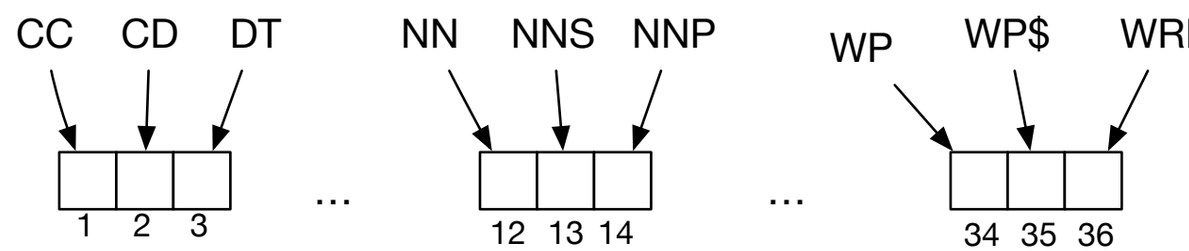
Linguistic features

How to represent?

- POS-tags
- Words/lemmas
- Chunks
- Dependencies

Penn Treebank Pos-tags

- 36 tags: 36 dimensions (one-hot representation)



CC <1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0>

CD <0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0>

DT <0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0>

NN <0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0>

NNS <0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0>

NNP <0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0>

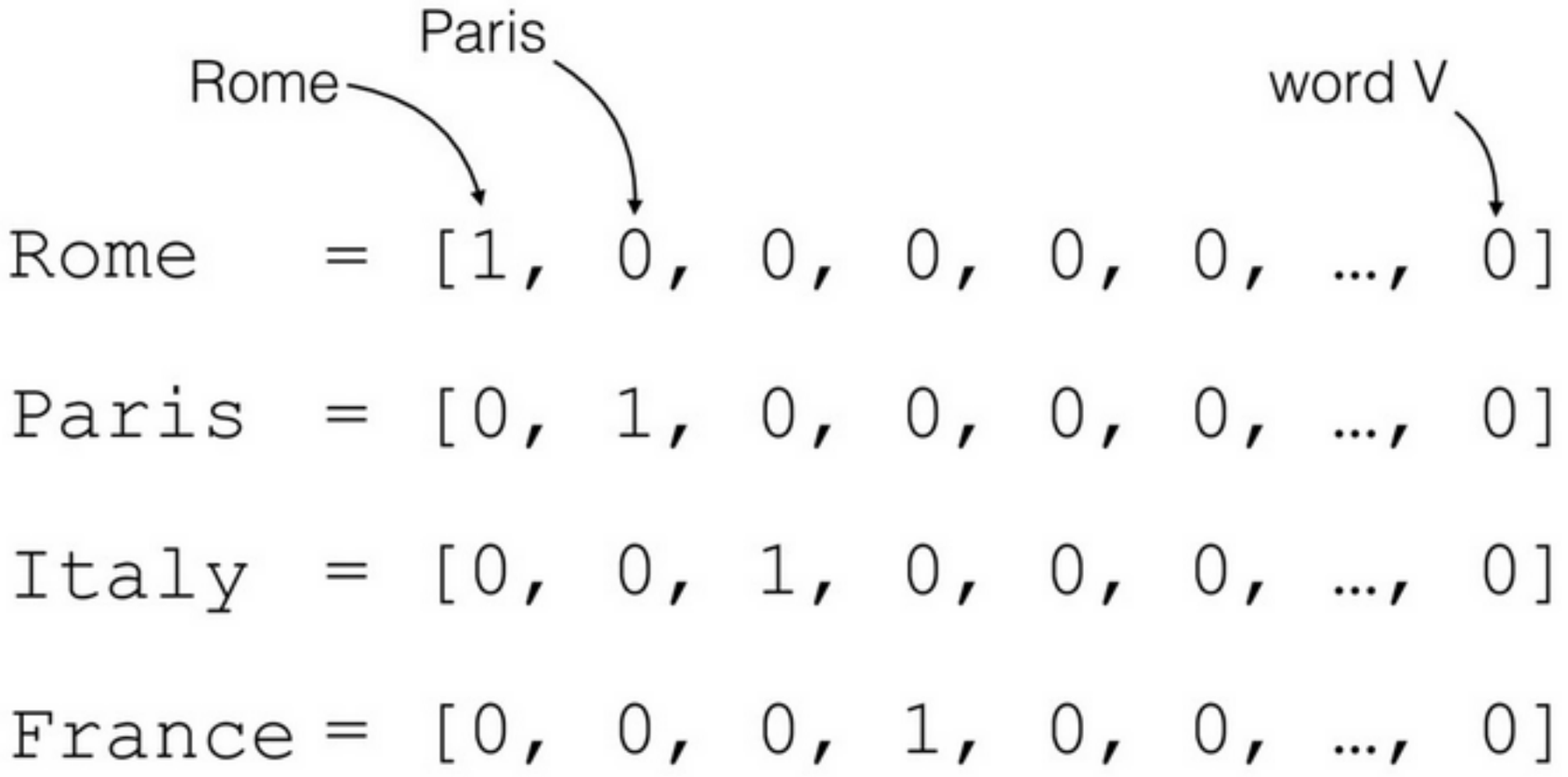
WP <0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0>

WP$ <0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0>

WRB <0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1>

Word - Lemmas

- One-hot representations: vocabulary-size dimensions



Source image: Shaffy, Athif (2017) Vector Representation of Text for Machine Learning. https://medium.com/@athif.shaffy/one-hot-encoding-of-text-b69124bef0a7

Why are Word Embeddings so useful?

how about more complex features?

Chunks

[the mailman$_{NP}$] [has already delivered$_{VP}$] [his letter$_{NP}$]

- Kind of information:
  - which words form a chunk
  - order (structure) of these words
  - label of the chunk

Chunks

- which words form a chunk:
  - sum over vectors representing words:
    => captures which words are present, but no structure, no differentiation between individual words within the chunk
  - concatenation is problematic:

    => different lengths of chunk
    => maximum expected length? problem: 2nd word in 4 word chunk is not the same as 2nd word in 2 word chunk

Chunks

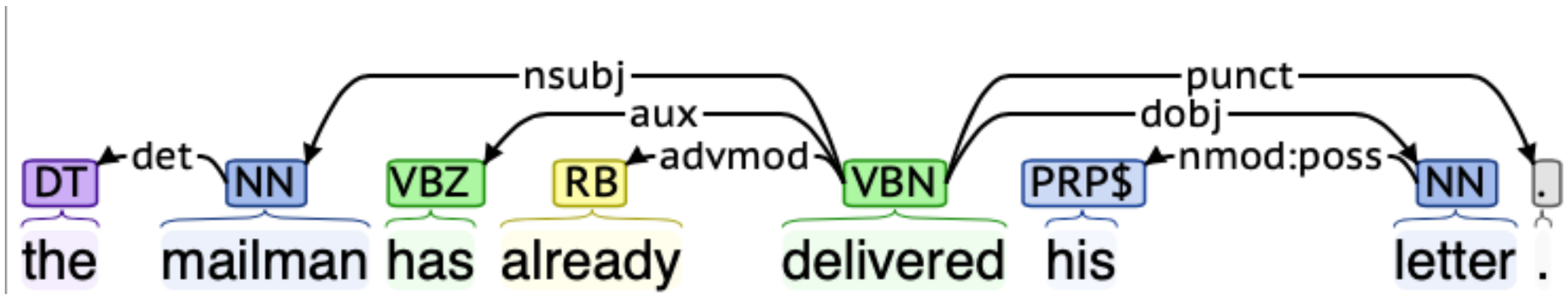
- order (structure) of these words:
    - what structural information matters?
        - position:
            - last word or not, first word or not, middle word or not
            - preceding head word, following head word
        - function:
            - head word or not
        - morphological marking:
            - dependency marking?

Chunks

- label of the chunk:
- one-hot vectors

  => usually closed class of categories with limited variation (comparable to POS-tags)

Dependencies



- Information:
  - head word & label
  - dependents (& label)
  - siblings (& labels)

Dependencies

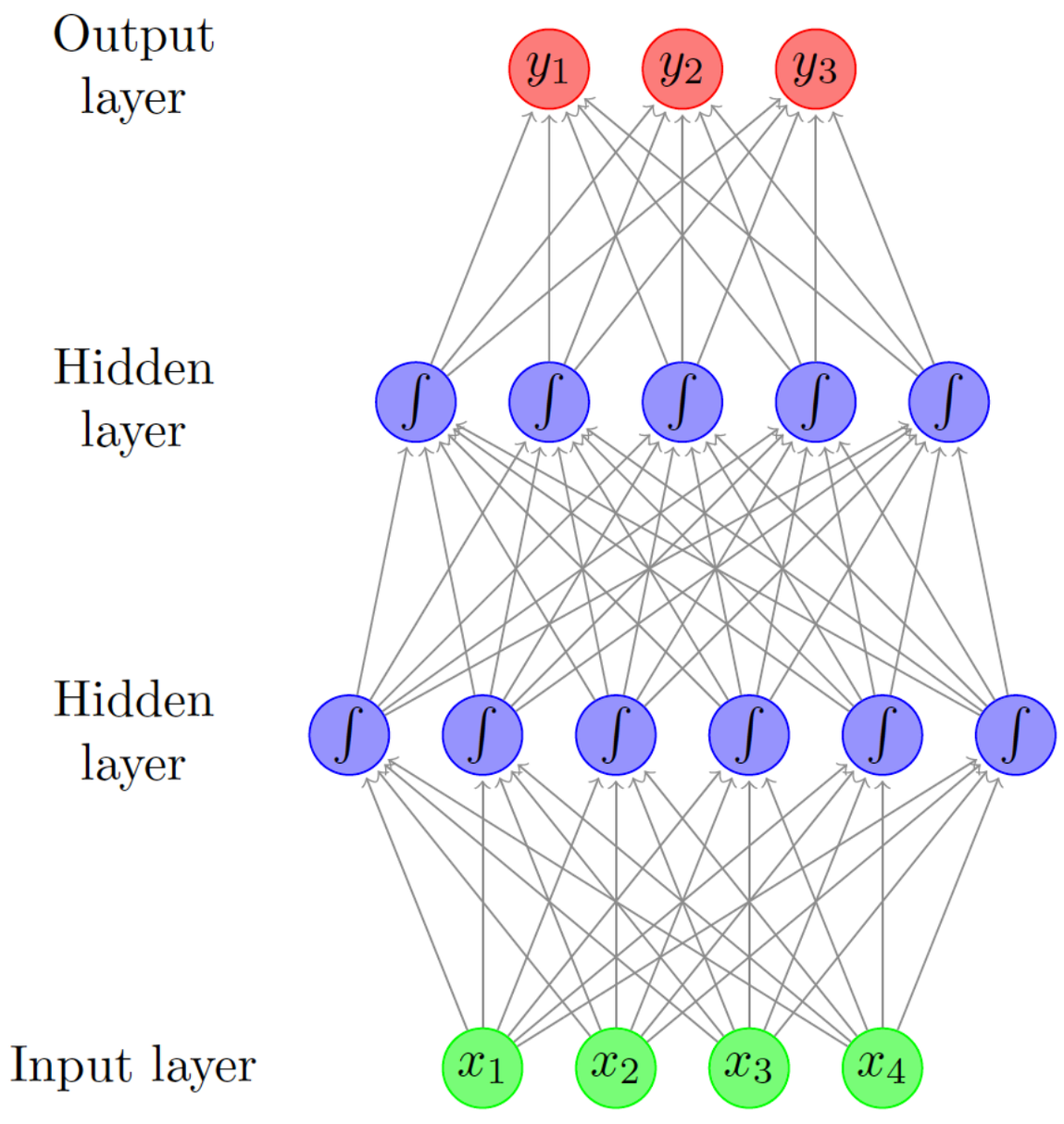
- Information:
  - **head word & label <= most typically represented**
  - dependents (& label)
  - siblings (& labels)

Dependencies

- head word & label representation, concatenated:
  - vector representation of head word (one hot or higher density embedding)
  - label representation (e.g. one hot vector)

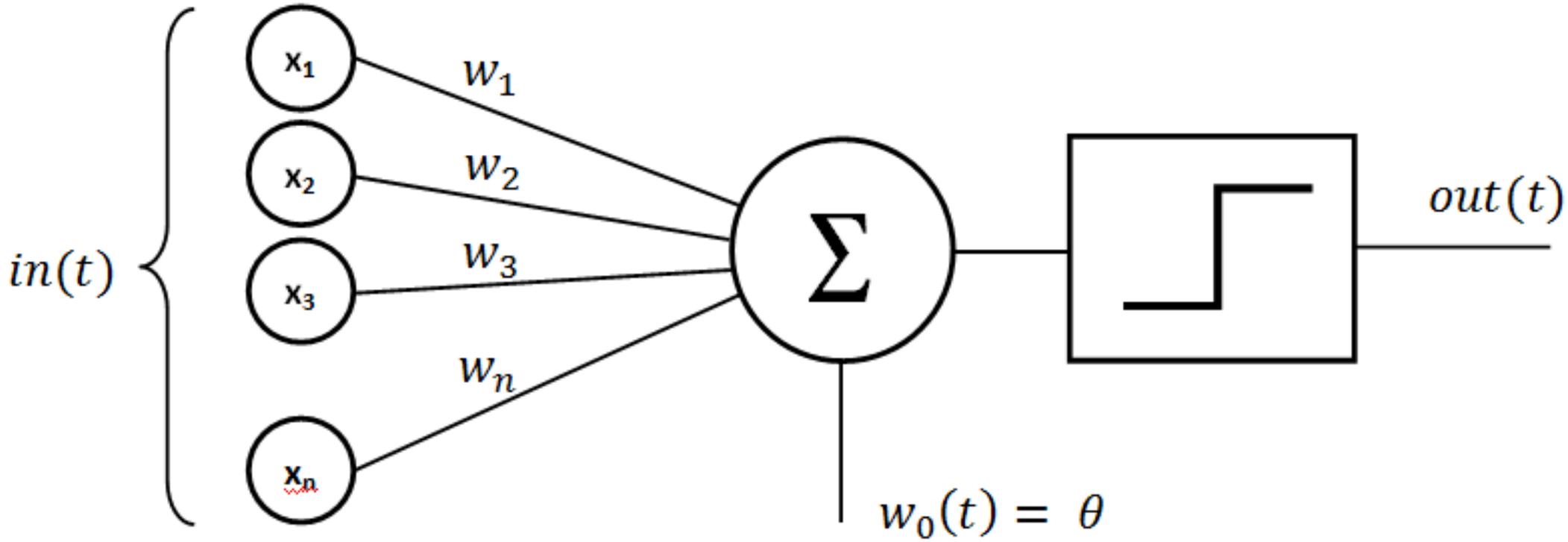
Neural Networks

Feed-forward Network



Goldberg (2015) p. 11

Perceptron

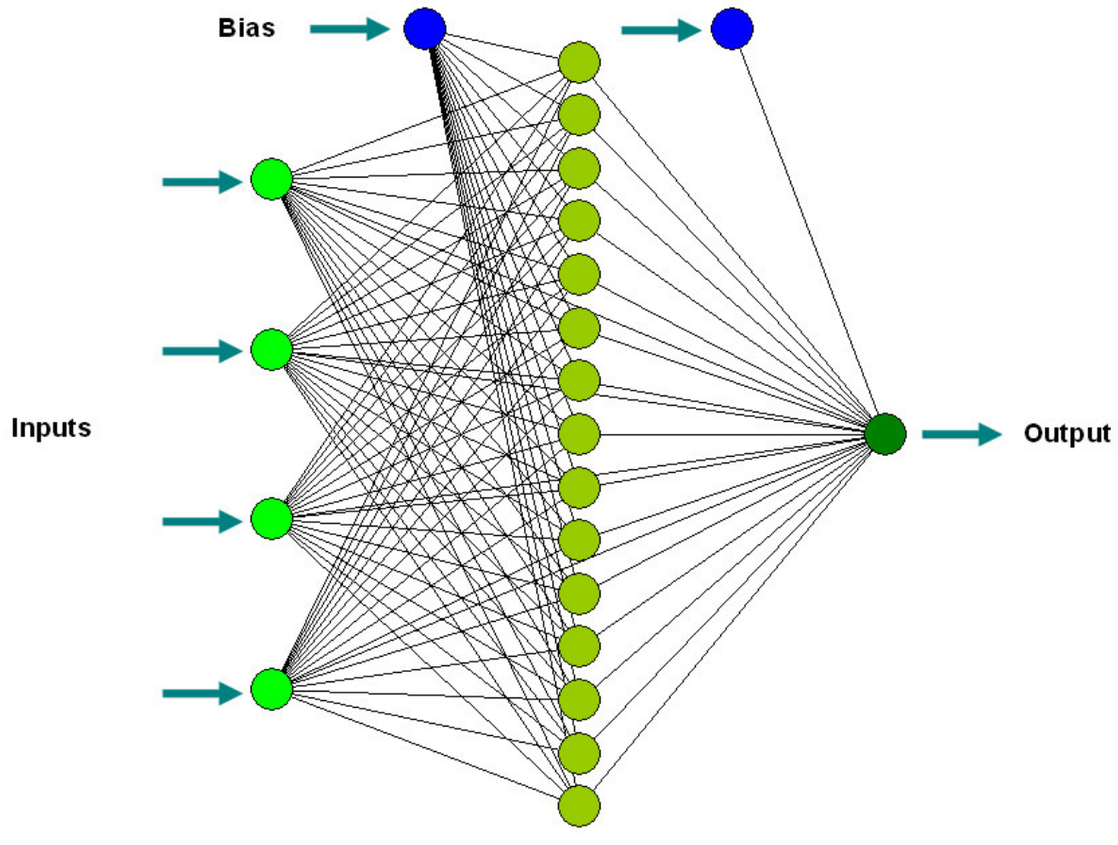


Source: https://commons.wikimedia.org/wiki/File:Perceptron_moj.png
Accessed November 2018

$$NN_{Perceptron}(x) = xW + b$$

$$x \in \mathbb{R}^{d_{in}}, W \in \mathbb{R}^{d_{in} \times d_{out}}, b \in \mathbb{R}^{d_{out}}$$

One layer neural network



Source: https://www.flickr.com/photos/worldworldworld/7880863848
Accessed: November 2018

$$NN_{MLP1}(x) = g(xW_1 + b_1)W_2 + b_2$$

$$x \in \mathbb{R}^{d_{in}}, W_1 \in \mathbb{R}^{d_{in} \times d_1}, b_1 \in \mathbb{R}^{d_1},$$

$$W_2 \in \mathbb{R}^{d_1 \times d_{out}}, b_2 \in \mathbb{R}^{d_{out}}$$

Two layer neural network

$$NN_{MLP2}(x) = (g_2(g_1(xW_1 + b_1)W_2 + b_2))W_3 + b_3$$

$$x \in \mathbb{R}^{d_{in}}, W_1 \in \mathbb{R}^{d_{in} \times d_1}, b_1 \in \mathbb{R}^{d_1}, \quad W_2 \in \mathbb{R}^{d_1 \times d_2}, b_2 \in \mathbb{R}^{d_2}, W_3 \in \mathbb{R}^{d_2 \times d_{out}}, b_3 \in \mathbb{R}^{d_{out}}$$

Alternative representation:

$$NN_{MLP2}(x) = y$$

$$h_1 = g_1(xW_1 + b_1)$$

$$h_2 = g_2(h_1W_2 + b_2)$$

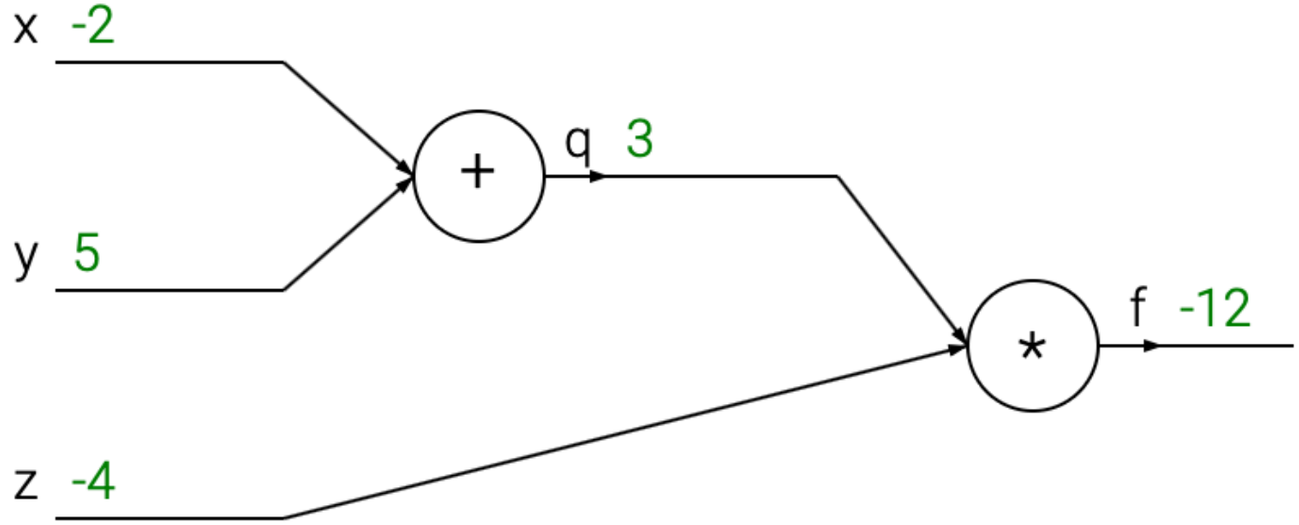
$$y = h_2W_3 + b_3$$

Training

- Stochastic Gradient Descent (SGD) algorithm
- Goal (as before): set parameters θ to minimize the Loss (*L*)
- Approach (as before):
    1. determine gradient descents of $L(f(x_i;\theta), y_i)$, w.r.t.
    2. update based on gradient descent and learning rate

Computation Graphs

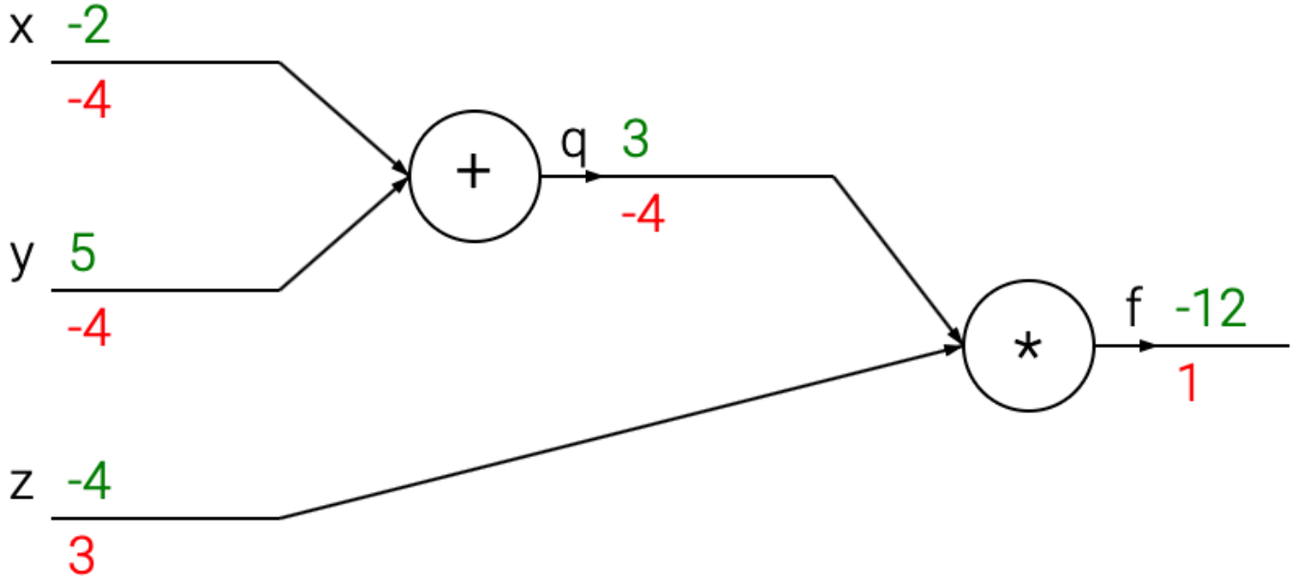
$$f(x, y, z) = (x + y)z$$



Adapted from: http://cs231n.github.io/optimization-2/
Accessed 2 December 2018

Computation Graphs

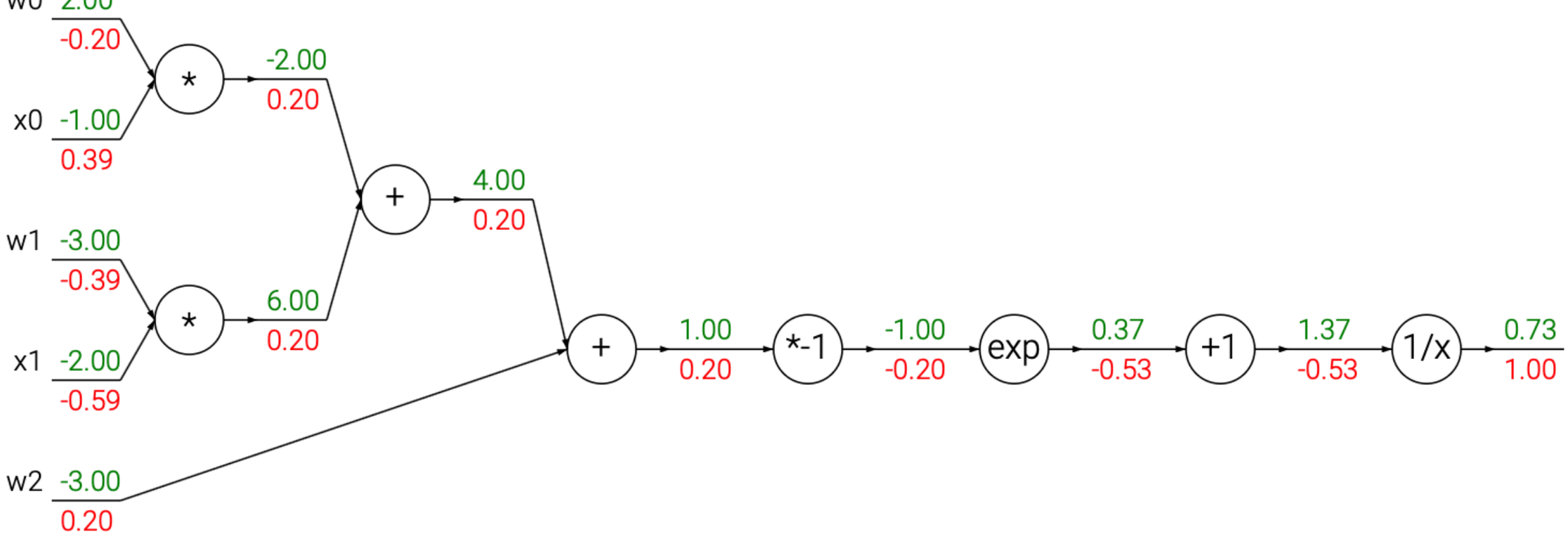
$$f(x, y, z) = (x + y)z$$



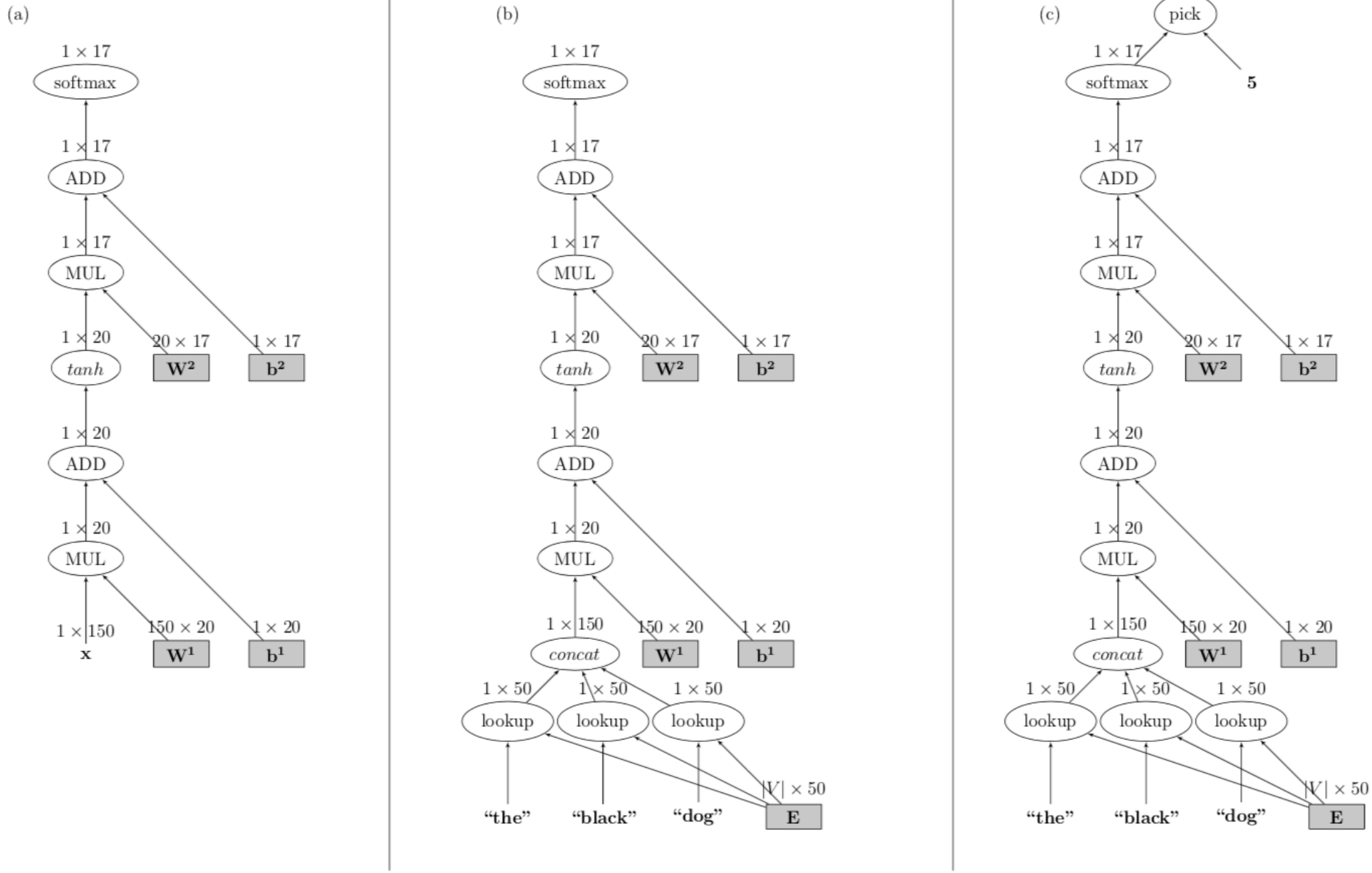
From: http://cs231n.github.io/optimization-2/
Accessed 2 December 2018

A more elaborate example

$$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



Adapted from: http://cs231n.github.io/optimization-2/
Accessed 2 December 2018



Goldberg (2015) p. 29

Stochastic Gradient Descent Training

Algorithm 1 Online Stochastic Gradient Descent Training

Input: Function $f(\mathbf{x};\theta)$ parameterized with parameters θ.
Input: Training set of inputs $\mathbf{x_1}, \ldots, \mathbf{x_n}$ and outputs $\mathbf{y_1}, \ldots, \mathbf{y_n}$.
Input: Loss function L.
while stopping criteria not met **do**
    Sample a training example $\mathbf{x_i}, \mathbf{y_i}$
    Compute the loss $L(f(\mathbf{x_i};\theta), \mathbf{y_i})$
    $\hat{\mathbf{g}} \leftarrow$ gradients of $L(f(\mathbf{x_i};\theta), \mathbf{y_i})$ w.r.t θ
    $\theta \leftarrow \theta + \eta_k \hat{\mathbf{g}}$
return θ

Goldberg (2015) p. 26

With Computation Graph Abstraction

Algorithm 5 Neural Network Training with Computation Graph Abstraction (using mini-batches of size 1)

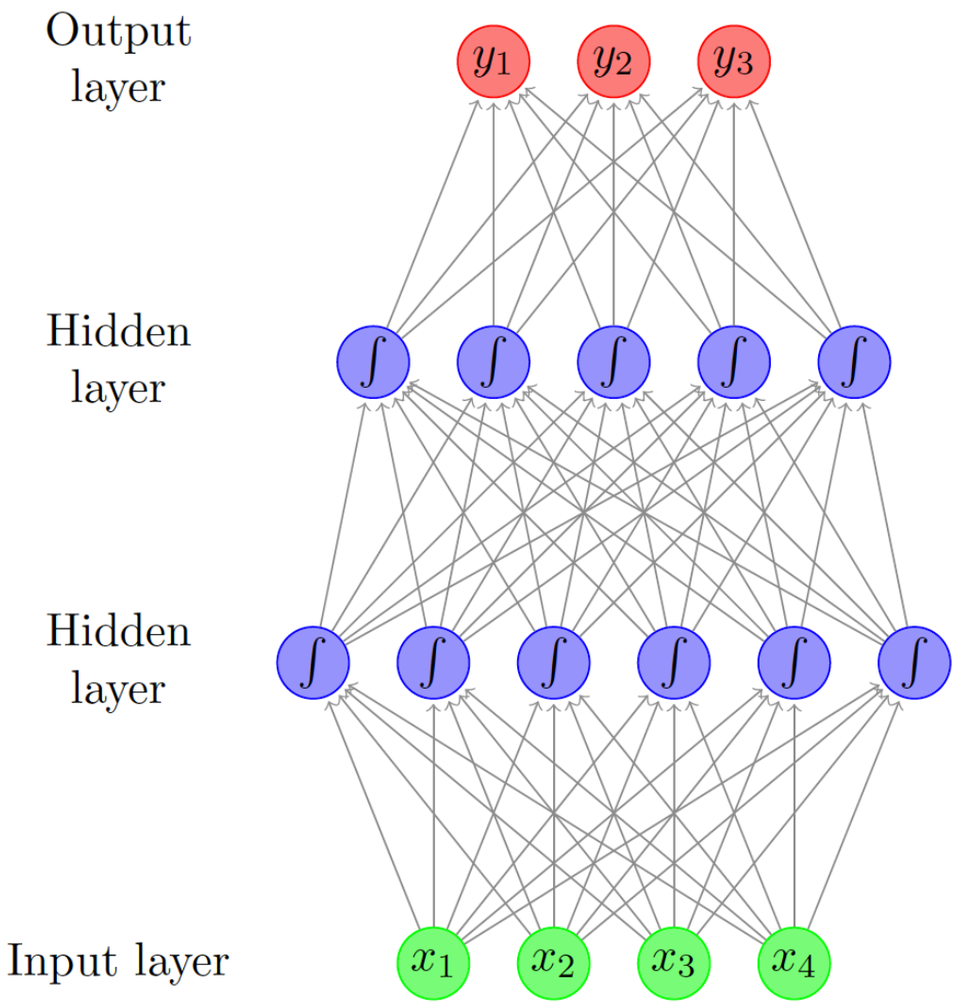
```
Define network parameters.
for iteration = 1 to N do
    for Training example x_i, y_i in dataset do
        loss_node ← build_computation_graph(x_i, y_i, parameters)
        loss_node.forward()
        gradients ← loss_node().backward()
        parameters ← update_parameters(parameters, gradients)
return parameters.
```

Goldberg (2015) p.32

Issues/points of attention

- Initialization
- Vanishing and Exploding gradients
- Saturation and Dead Neurons
- Example order
- Learning Rate
- Minibatches
- Overfitting

Stacking up neural networks



Parse tree: update with loss

PoS-tags: update with loss

Image from Goldberg (2015) p. 11

Sequences & structures

Parsing

LEFT$_l$ adds an arc $\sigma_1 \xrightarrow{l} \sigma_2$ to A and removes σ_2 from the stack.

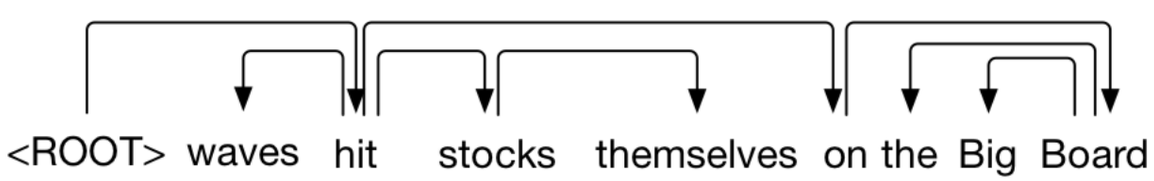
RIGHT$_l$ adds an arc $\sigma_2 \xrightarrow{l} \sigma_1$ to A and removes σ_1 from the stack.

SHIFT moves β_1 to the stack.

Example Parsing

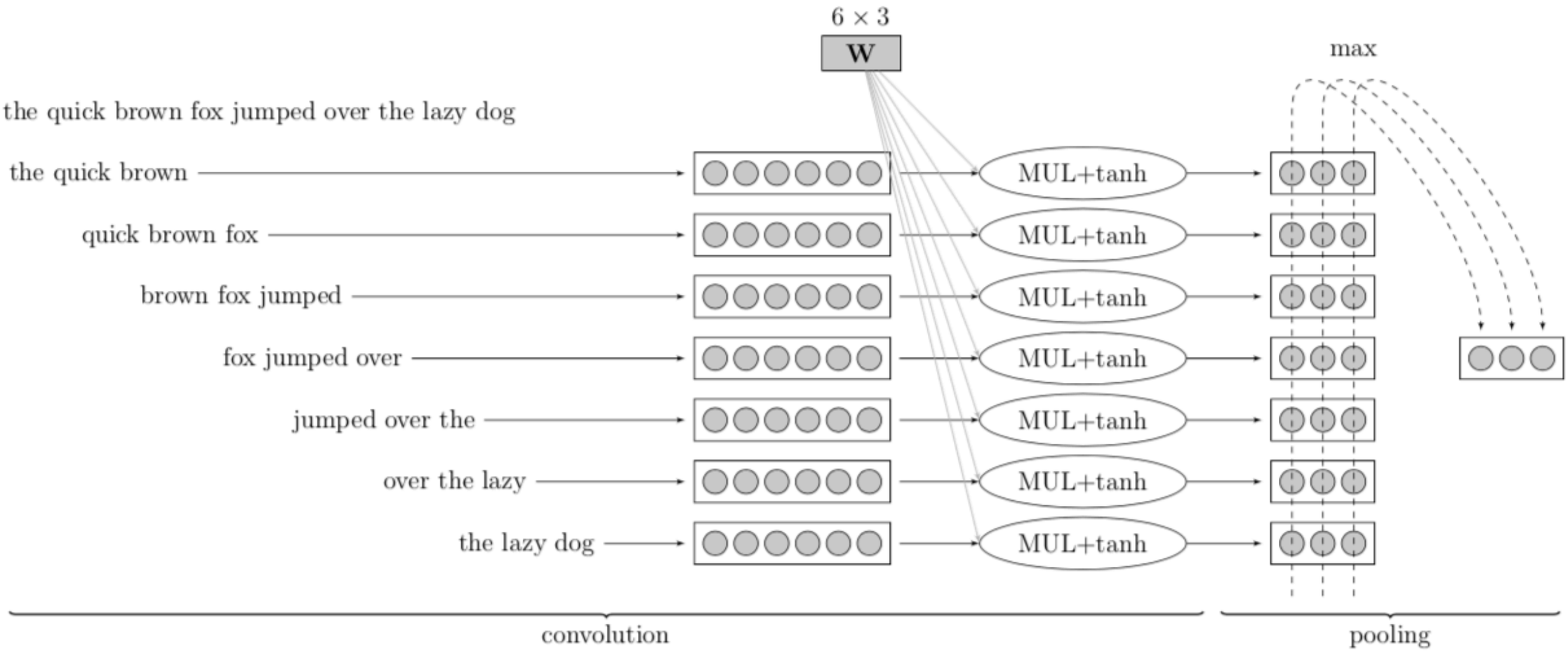
Step	Transition	Stack	Buffer	Arcs
0		<ROOT>	*waves hit ... Big Board*	$\emptyset$
1	SHIFT	<ROOT> *waves*	*hit stocks ... Big Board*	$\emptyset$
2	SHIFT	<ROOT> *waves hit*	*stocks themselves ... Big Board*	$\emptyset$
3	$\text{LEFT}_{\text{nsubj}}$	<ROOT> *hit*	*stocks themselves ... Big Board*	$A_1 = \{ hit \xrightarrow{\text{nsubj}} waves \}$
4	SHIFT	<ROOT> *hit stocks*	*themselves on the Big Board*	A_1
5	SHIFT	<ROOT> *hit stocks themselves*	*on the Big Board*	A_1
6	$\text{RIGHT}_{\text{dep}}$	<ROOT> *hit stocks*	*on the Big Board*	$A_2 = A_1 \cup \{ stock \xrightarrow{\text{dep}} themselves \}$
7	$\text{RIGHT}_{\text{dobj}}$	<ROOT> *hit*	*on the Big Board*	$A_3 = A_2 \cup \{ hit \xrightarrow{\text{dobj}} stock \}$

Table 1: Parsing oracle walk-through



Example from Le & Fokkens (2014)

Convolutional Neural Network



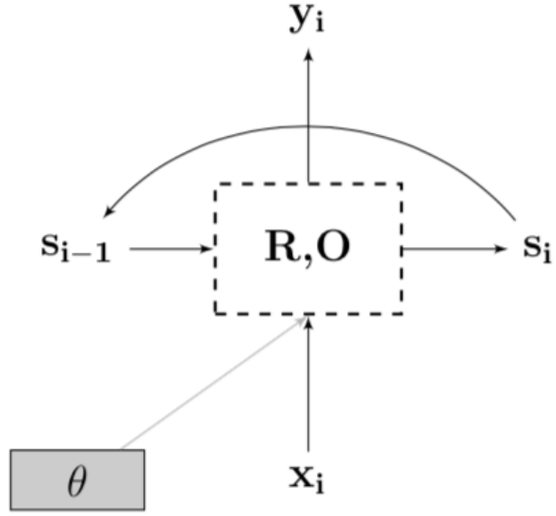
Goldberg (2015) p. 44

Recurrent Neural Networks

$$RNN(S_0, X_{1:n}) = s_{1:n}, y_{1:n}$$

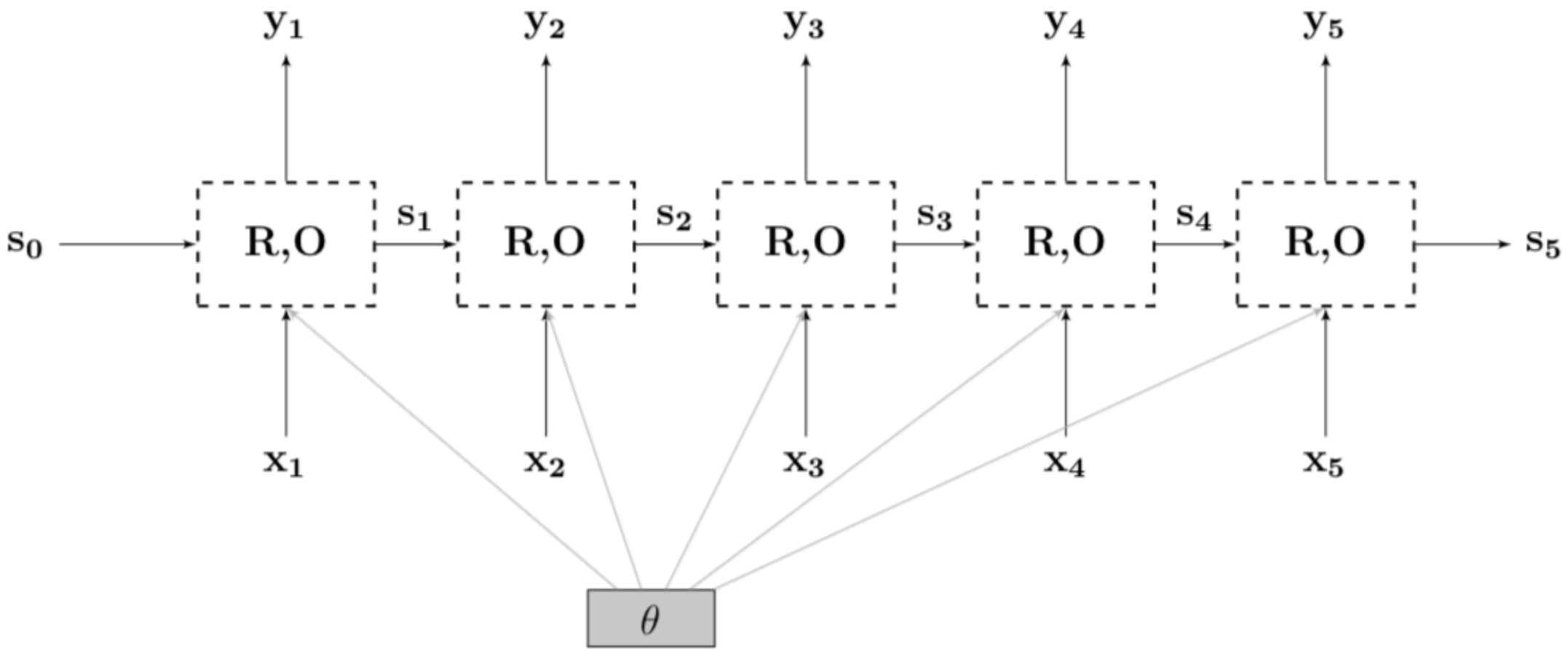
$$s_i = R(s_{i-1}, x_i)$$

$$y_i = O(s_i)$$



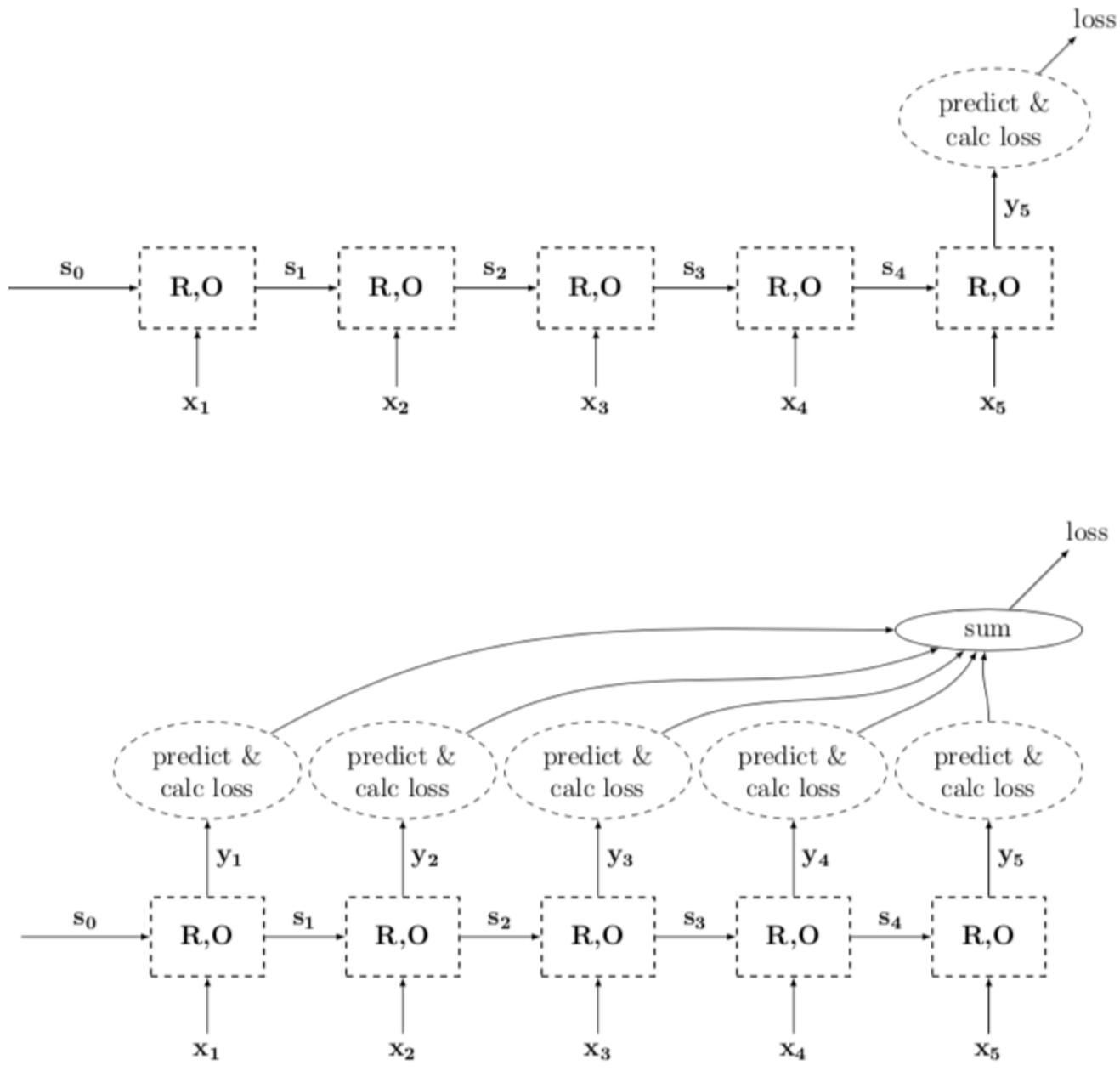
Goldberg (2015) p. 47

Unrolled representation



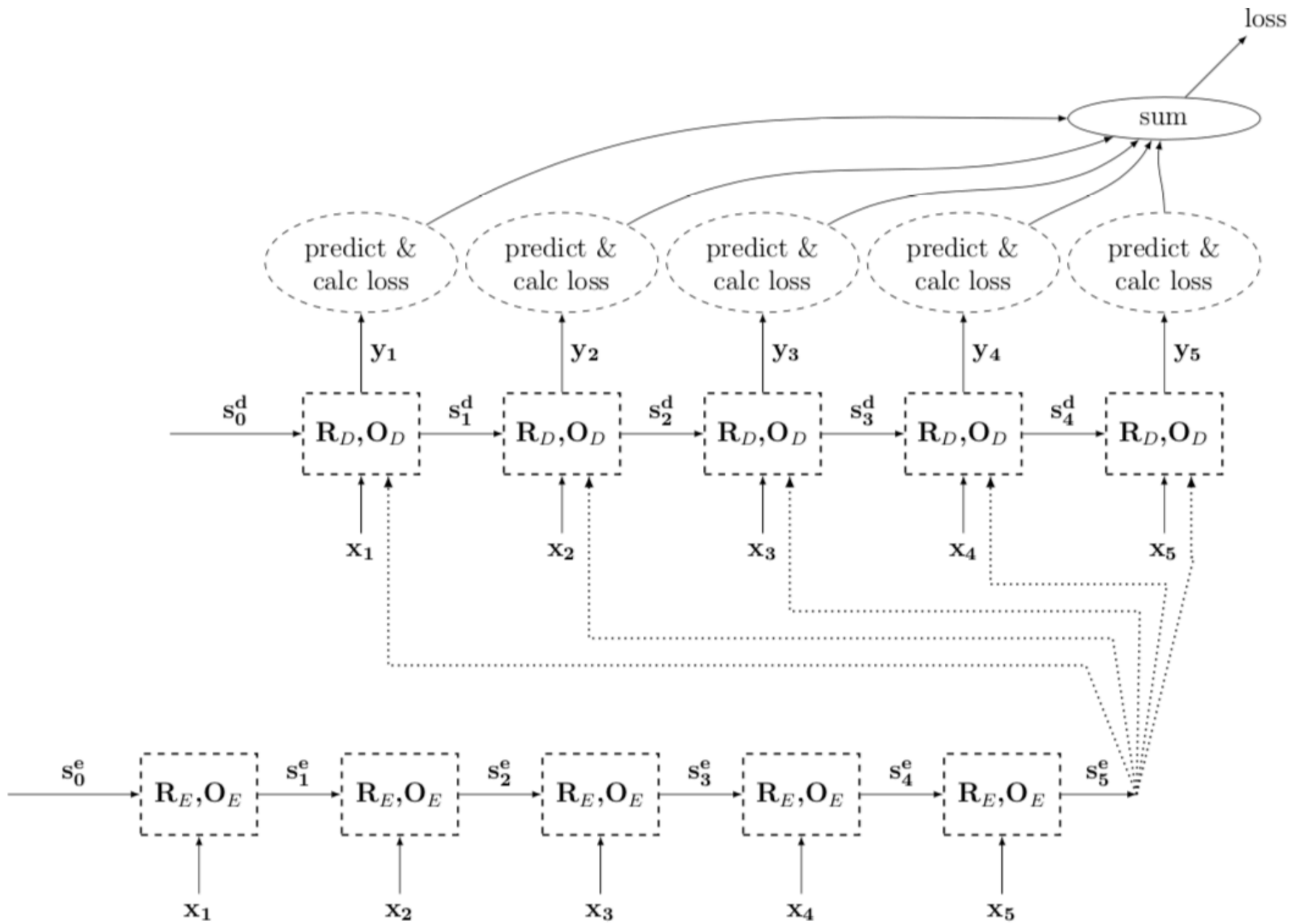
Goldberg (2015) p. 47

Acceptor & Transducer



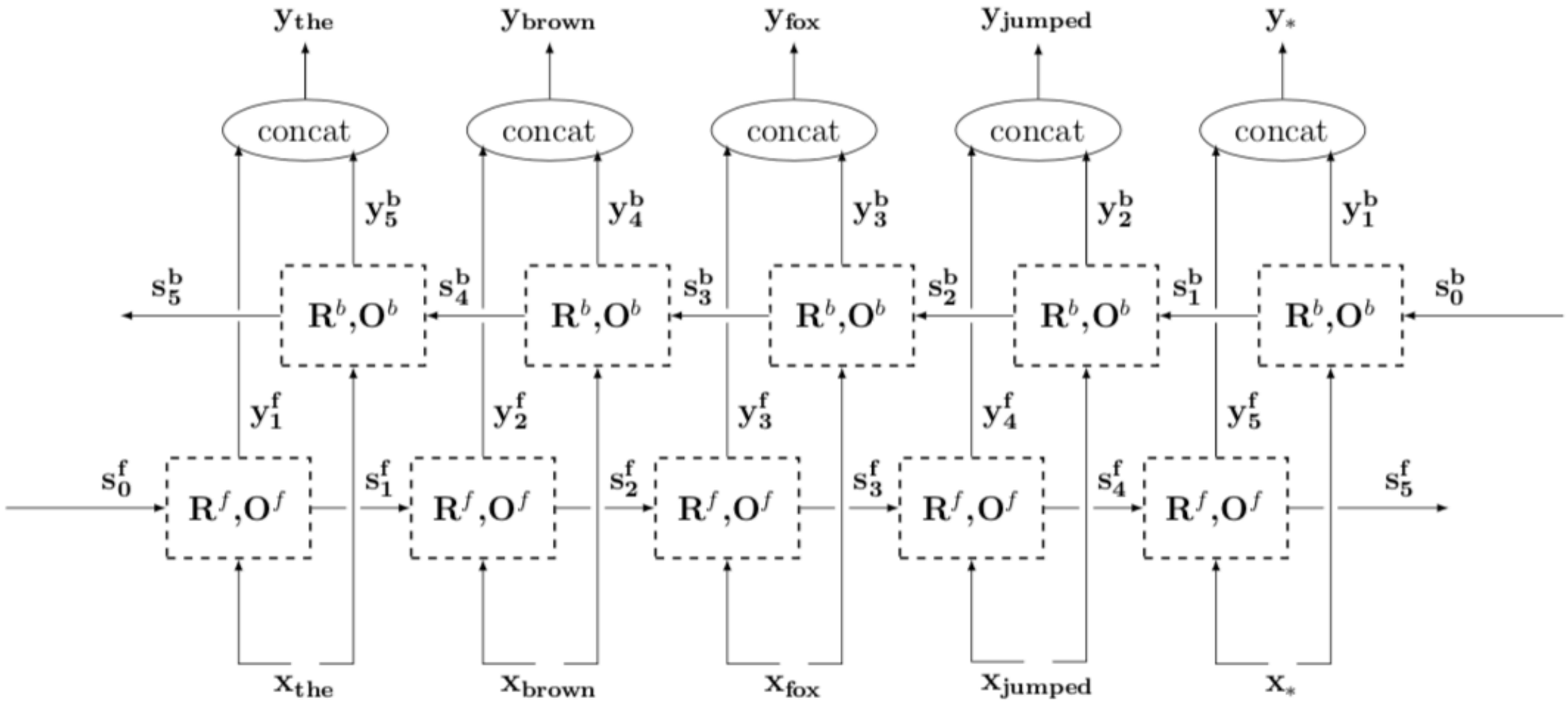
Goldberg (2015) p. 49, p.50

Encoder - Decoder



Goldberg (2015) p. 51

BiRNN



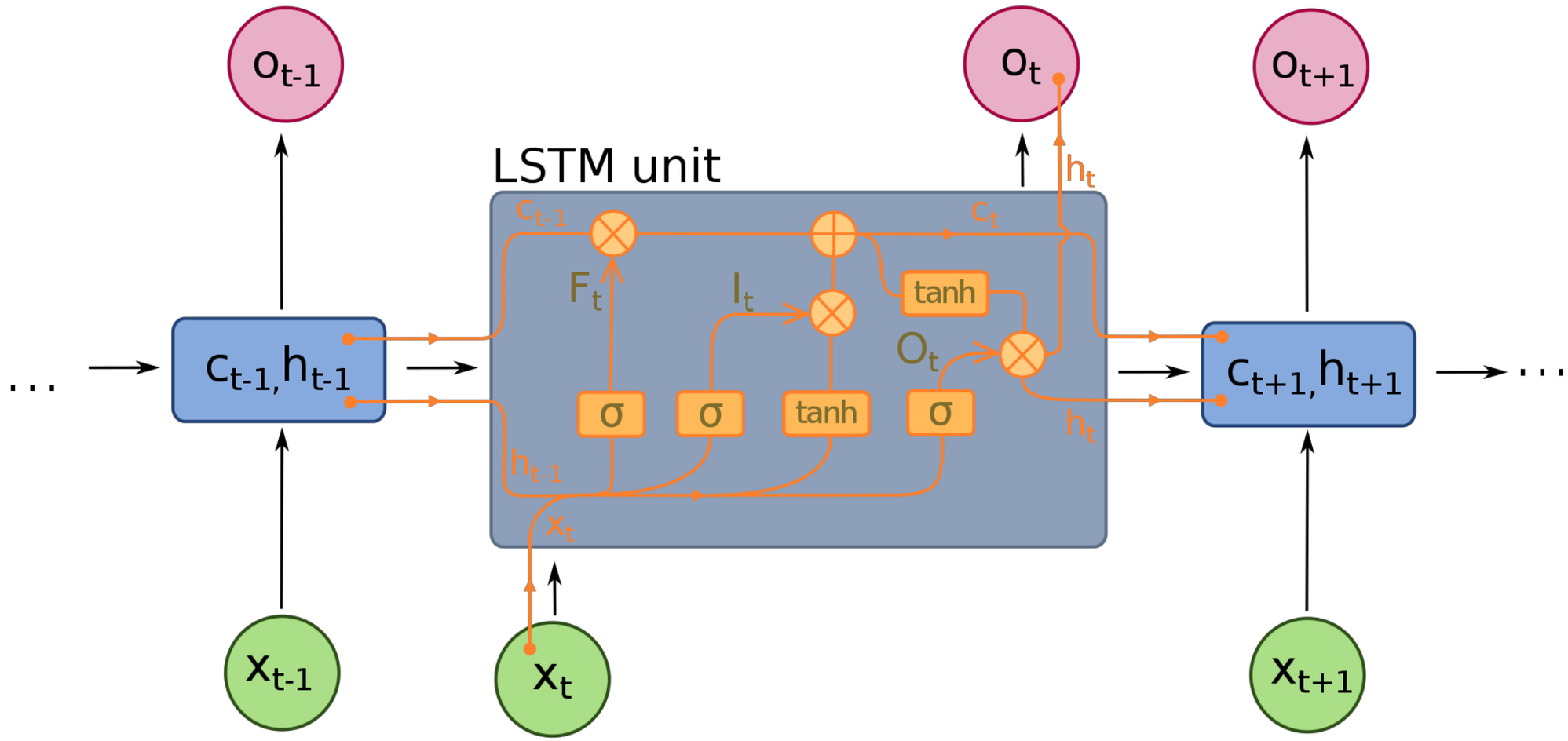
Goldberg (2015) p. 53

Concrete Architectures

- Simple RNN:

$$s_i = R_{SRNN}(s_{i-1}, x_i) = g(x_i W^x + s_{i-1} W^s + b)$$

LSTM



Source: https://commons.wikimedia.org/wiki/File:Long_Short-Term_Memory.svg
Accessed: 3 December 2018

LSTM

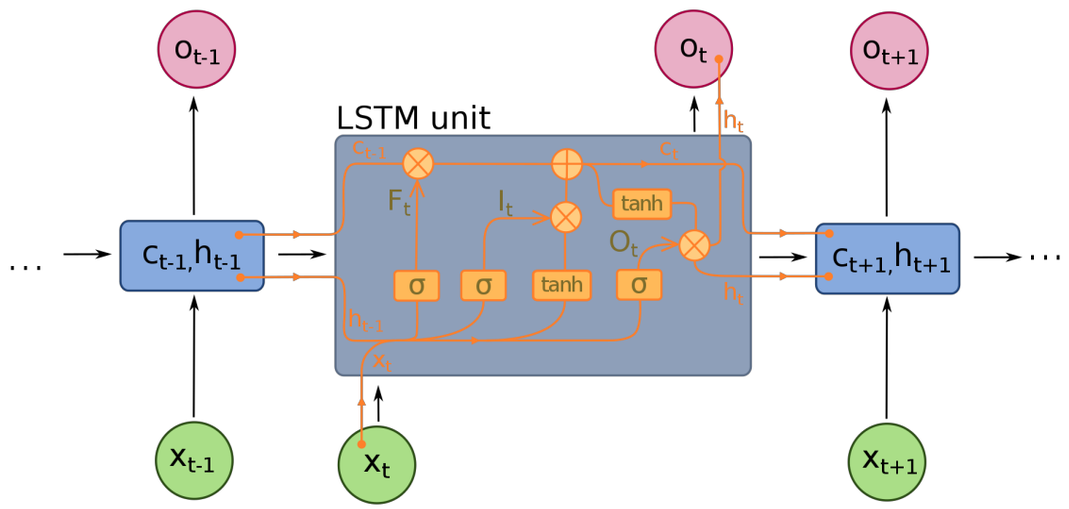
$$
\begin{aligned}
\mathbf{s_j} = R_{LSTM}(\mathbf{s_{j-1}}, \mathbf{x_j}) &= [\mathbf{c_j}; \mathbf{h_j}] \\
\mathbf{c_j} &= \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{h_j} &= \tanh(\mathbf{c_j}) \odot \mathbf{o} \\
\mathbf{i} &= \sigma(\mathbf{x_j}\mathbf{W^{xi}} + \mathbf{h_{j-1}}\mathbf{W^{hi}}) \\
\mathbf{f} &= \sigma(\mathbf{x_j}\mathbf{W^{xf}} + \mathbf{h_{j-1}}\mathbf{W^{hf}}) \\
\mathbf{o} &= \sigma(\mathbf{x_j}\mathbf{W^{xo}} + \mathbf{h_{j-1}}\mathbf{W^{ho}}) \\
\mathbf{g} &= \tanh(\mathbf{x_j}\mathbf{W^{xg}} + \mathbf{h_{j-1}}\mathbf{W^{hg}})
\end{aligned}
$$

$$
\mathbf{y_j} = O_{LSTM}(\mathbf{s_j}) = \mathbf{h_j}
$$

Goldberg (2015) p. 57

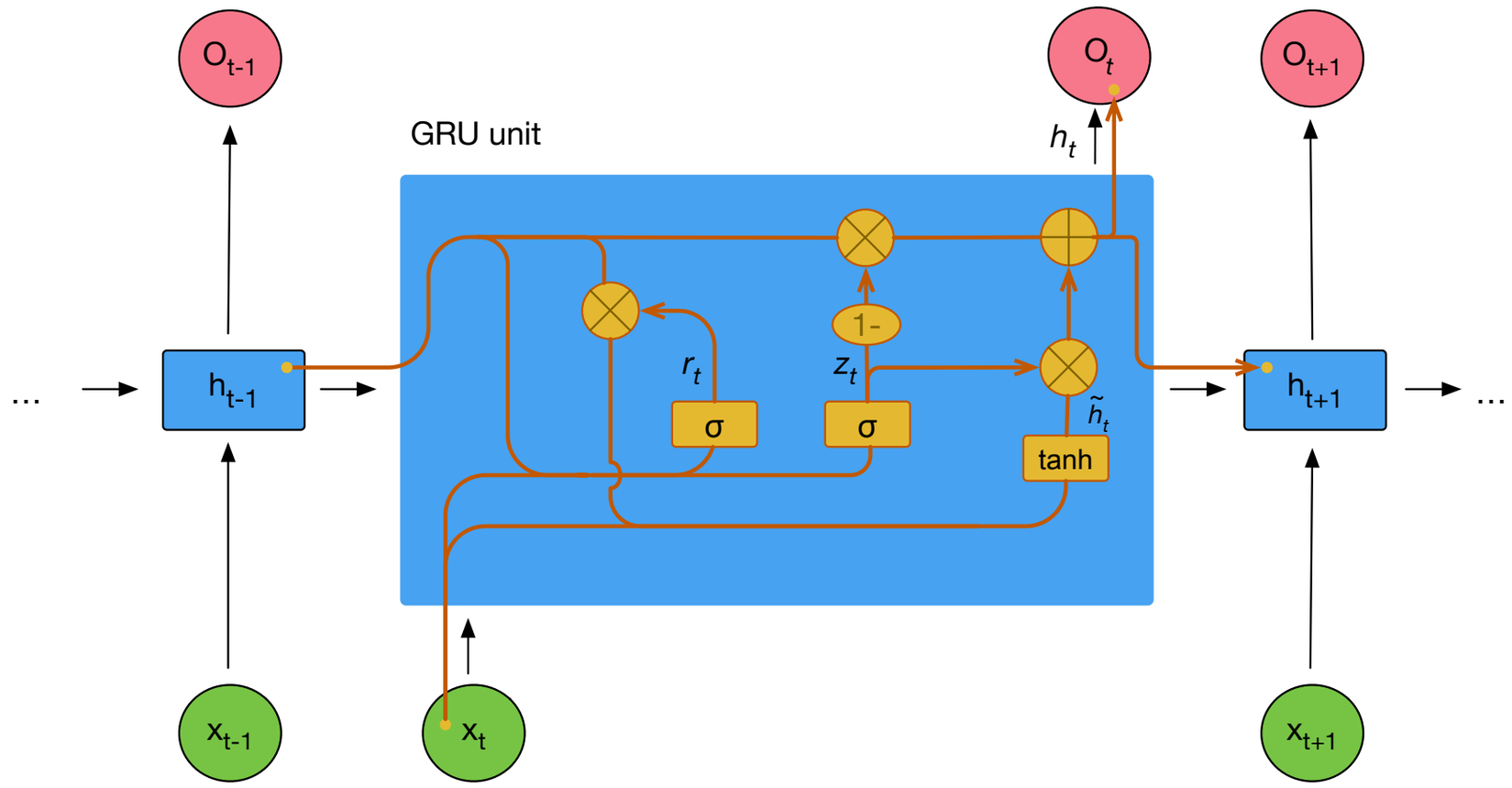
$$
\begin{aligned}
\mathbf{s_j} = R_{LSTM}(\mathbf{s_{j-1}}, \mathbf{x_j}) =& [\mathbf{c_j}; \mathbf{h_j}] \\
\mathbf{c_j} =& \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{h_j} =& \tanh(\mathbf{c_j}) \odot \mathbf{o} \\
\mathbf{i} =& \sigma(\mathbf{x_j W^{xi}} + \mathbf{h_{j-1} W^{hi}}) \\
\mathbf{f} =& \sigma(\mathbf{x_j W^{xf}} + \mathbf{h_{j-1} W^{hf}}) \\
\mathbf{o} =& \sigma(\mathbf{x_j W^{xo}} + \mathbf{h_{j-1} W^{ho}}) \\
\mathbf{g} =& \tanh(\mathbf{x_j W^{xg}} + \mathbf{h_{j-1} W^{hg}})
\end{aligned}
$$

$$
\mathbf{y_j} = O_{LSTM}(\mathbf{s_j}) = \mathbf{h_j}
$$



Source: https://commons.wikimedia.org/wiki/File:Long_Short-Term_Memory.svg
Accessed: 3 December 2018

GRU



le:Gated_Recurrent_Unit.svg, Accessed: 3 December 2018

s-rnn-bidirectional-rnn-lstm-gru-73927ec9df15, Accessed: 11 December 2018

GRU

$$h_t = R_{GRU}(h_{t-1}, x) = (1 - z_t) \odot h_{t-1} + z_t \odot h'_t$$

$$z_t = \sigma(x_t W^{xz} + h_{t-1} W^{hz})$$

$$r_t = \sigma(x_t W^{xr} + h_{t-1} W^{hr})$$

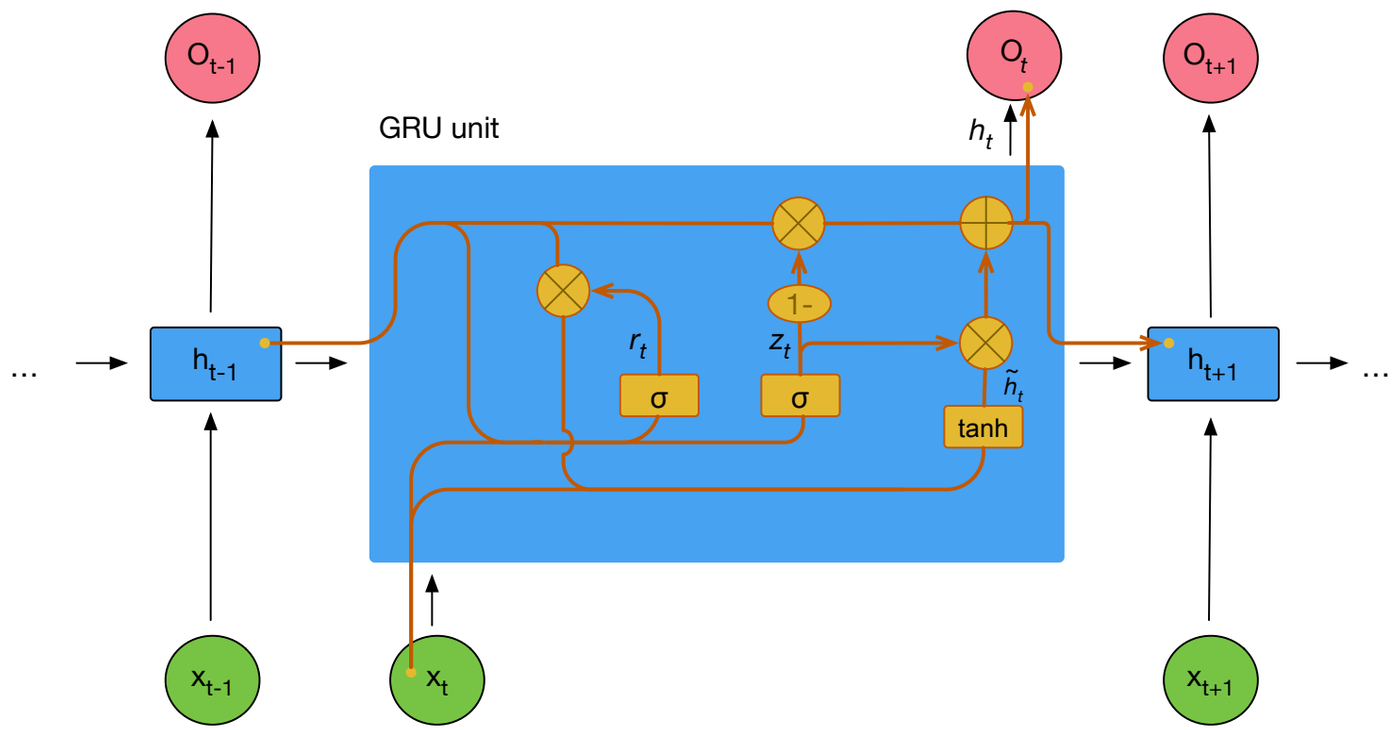
$$h'_t = tanh(x_t W^{xg} + (r_t \odot h_{t-1}) W^{hg}$$

$$y_t = O_{GRU}(h_t) = h_t$$

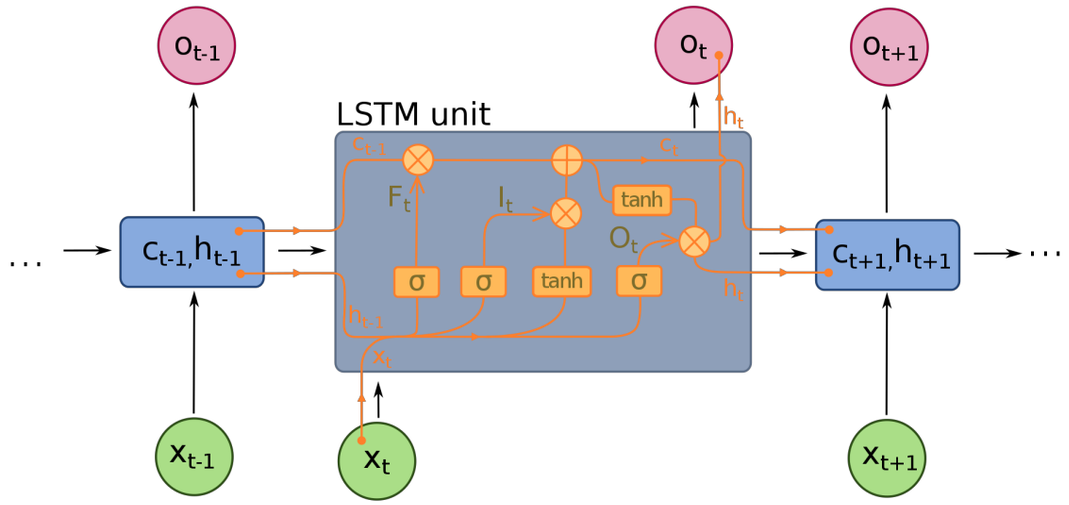
Based on Goldberg (2015) p. 58 and Gandhi (2018)

GRU

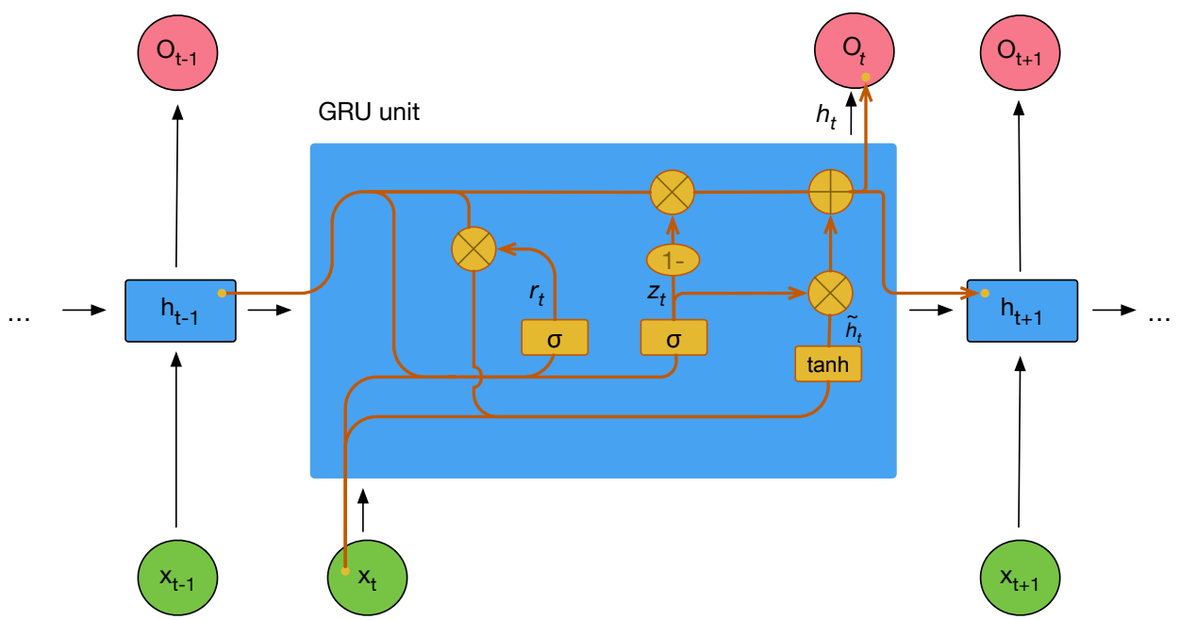
$$h_t = R_{GRU}(h_{t-1}, x) = (1 - z_t) \odot h_{t-1} + z_t \odot h'_t$$
$$z_t = \sigma(x_t W^{xz} + h_{t-1} W^{hz})$$
$$r_t = \sigma(x_t W^{xr} + h_{t-1} W^{hr})$$
$$h'_t = tanh(x_t W^{xg} + (r_t \odot h_{t-1}) W^{hg})$$
$$y_t = O_{GRU}(h_t) = h_t$$



LSTM vs GRU



Source: https://commons.wikimedia.org/wiki/File:Long_Short-Term_Memory.svg
Accessed: 3 December 2018



Feature Learning

Feature Learning

- Scenario:
  - you have limited data for your task (often the case)
  - you have data for other tasks that (partially) rely on the same information
- Idea:
  - use data from other tasks to support your target task

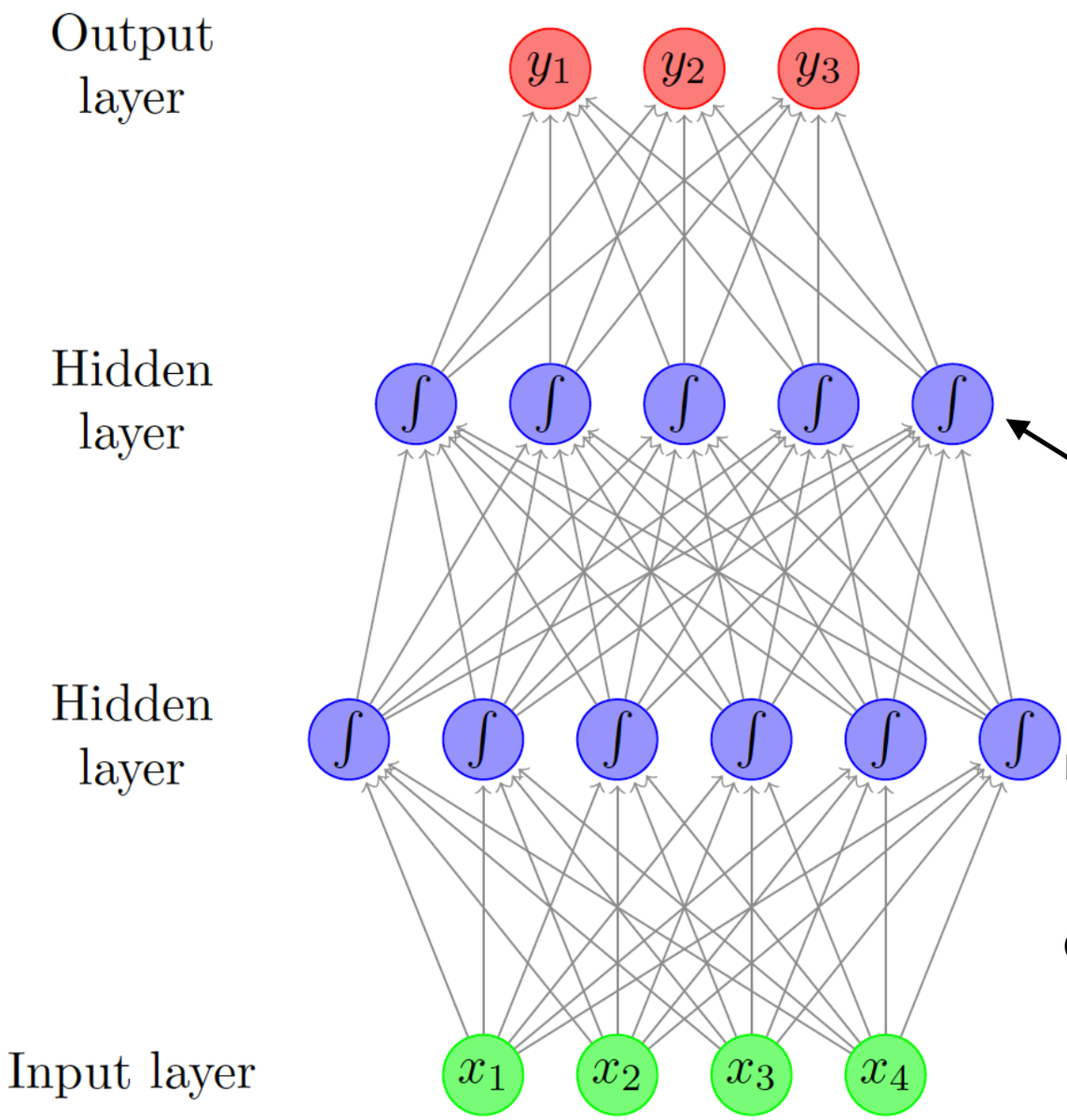
Feature Learning

- Auxiliary tasks
  - as in the old days:
    - supervised learning of relevant information
  - different:
    - use embeddings
    - training on useful information and ultimate task in the same model

Word Embeddings

- Standardly trained to predict the next word
- Alternative options (e.g.):
  - train to optimize pos-tagging (for parsing)
  - train to optimize sentiment prediction (for opinion mining)
  - train to optimize for syntactic parsing (for semantic role labeling)?

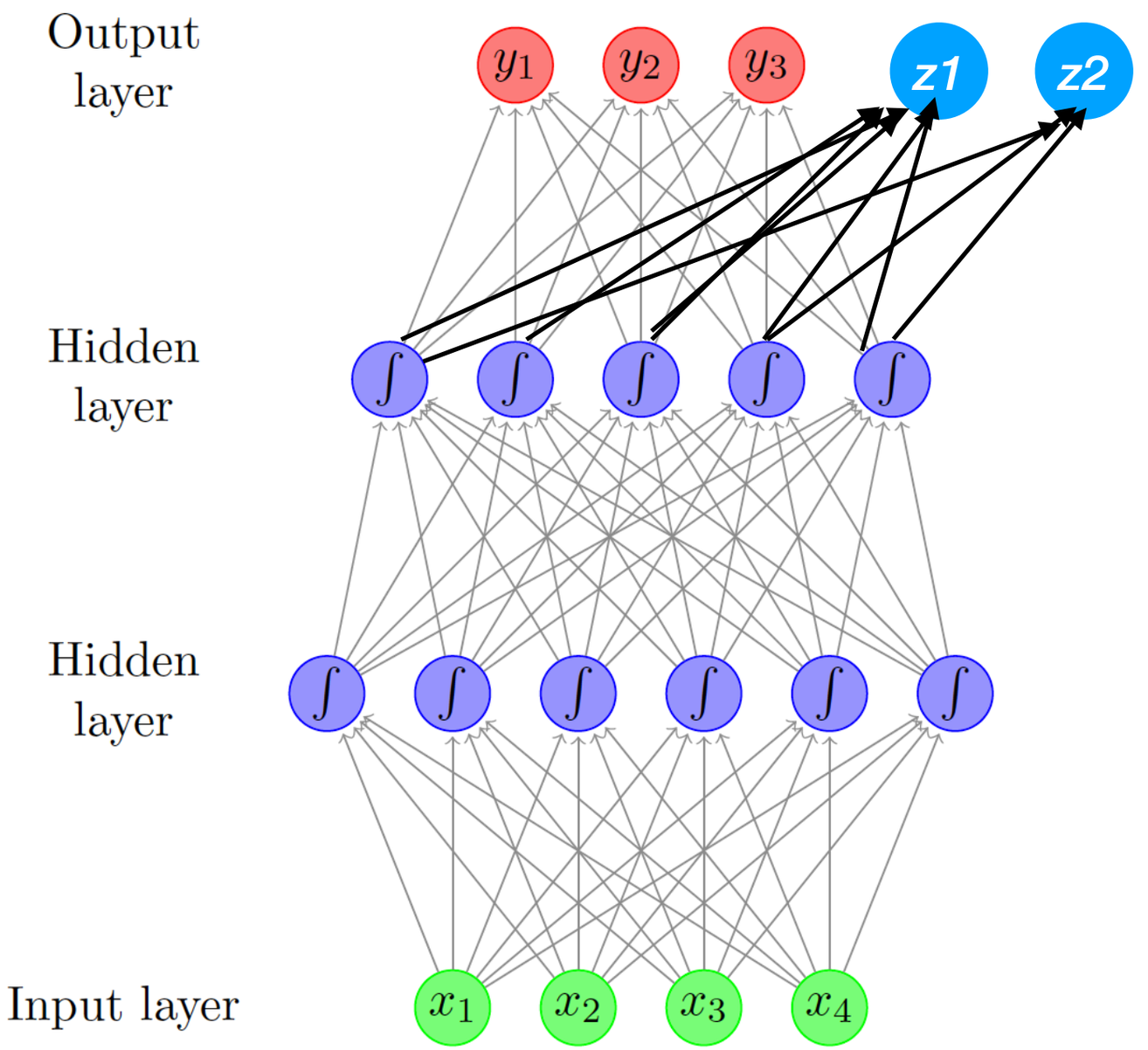
Feeding in useful information



optimize by syntactic parser

optimize by pos-tagger representation

Joint Learning



References

- Gandhi, Rohith (2018). Introduction to Sequence Models - RNN, Bidirectional RNN, LSTM, GRU https://towardsdatascience.com/introduction-to-sequence-models-rnn-bidirectional-rnn-lstm-gru-73927ec9df15 (accessed 11 december 2018)

- Goldberg, Yoav (2015). A Primer on Neural Network Models for Natural Language Processing. https://arxiv.org/pdf/1510.00726.pdf (last access December 3rd, 2018)

- Convolutional Neural Networks for Visual Recognition: http://cs231n.github.io/optimization-2/#grad (last access December 3rd, 2018)

- Le and Fokkens. 2017. Tackling Error Propagation through Reinforcement Learning: A Case of Greedy Dependency Parsing. In *Proceedings of EACL.* Valencia, Spain.