Faculty for Informatics

Technical University of Munich

Natural Language Processing
IN2361

Prof. Dr. Georg Groh

Social Computing Research Group

Chapter 8
Neural Networks and Neural Language Models

- content is based on [1]
- certain elements (e.g. equations or tables) were taken over or taken over in a modified form from [1]
- citations of [1] or from [1] are omitted for legibility
- errors are fully in the responsibility of Georg Groh
- BIG thanks to Dan and James for a great book!

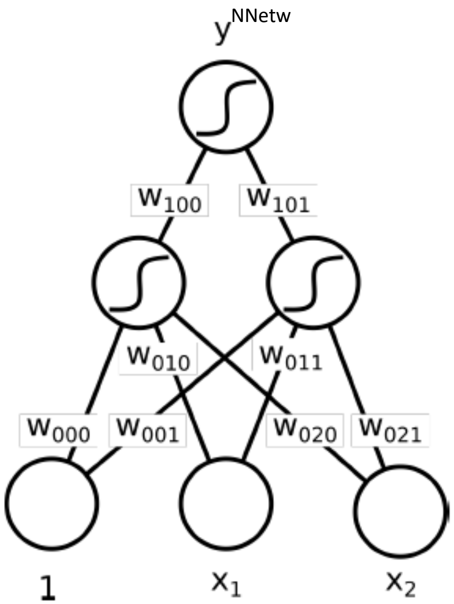
Repetition from ML1: Neural Networks

- Advantages:
  - no restrictions due to choice of particular model, NN are universal approximators
  - can autonomously learn appropriate features and feature representations (if enough data): no handcrafting of features
  - online learning, transfer learning, etc. easy

- Disadvantages:
  - usually require large amounts of data to work properly
  - computing ressources intensive
  - extremely sub-symbolic: model parameters hard to interpret

Notations

ML1 Lecture-Slides and Goodfellow

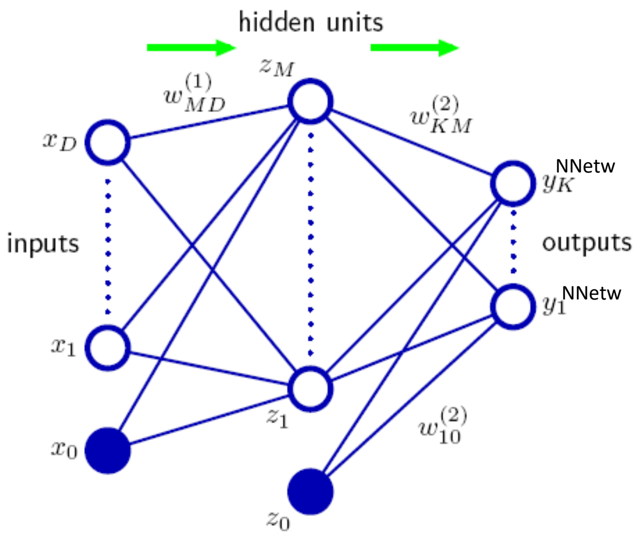


$$y^{\text{NNetw}} = f(\boldsymbol{x}, \boldsymbol{W}) = \sigma_1(\boldsymbol{W}_1^T \sigma_0(\boldsymbol{W}_0^T \boldsymbol{x}))$$

where $(\boldsymbol{W}_0^T \boldsymbol{x})_i = \sum_j (\boldsymbol{W}_0^T)_{ij}\, \boldsymbol{x}_j$

and $(\boldsymbol{W}_0^T)_{ij} = w_{0ji}$

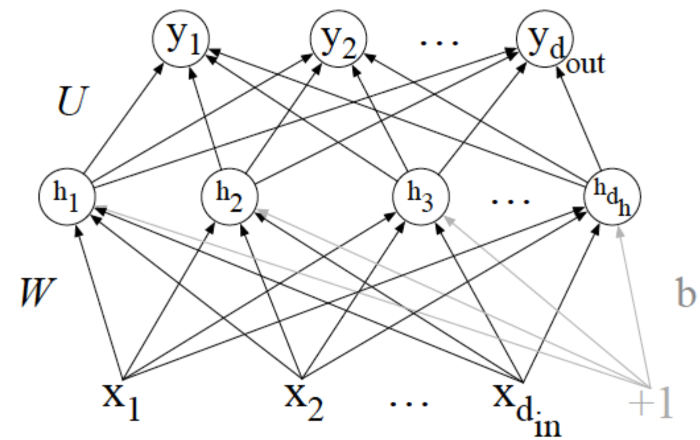
Bishop



$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right).$$

- counts layers from 1 (instead from 0)
- denotes layers as superscript
- does not have $()^T$ on the weight matrices

Jurafsky

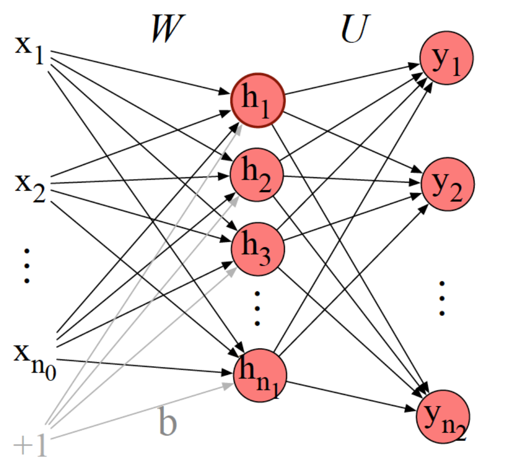


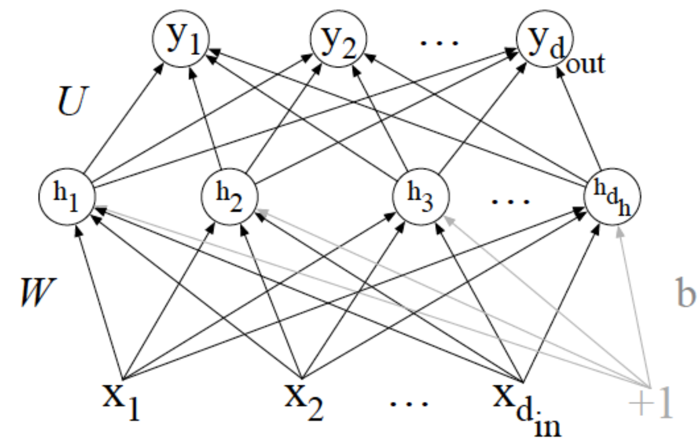
$$h = \sigma(Wx + b)$$
$$z = Uh$$
$$y = \text{softmax}(z)$$

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad 1 \le i \le D$$

Notations

horizontal diagram:





more general multi-layer notation:

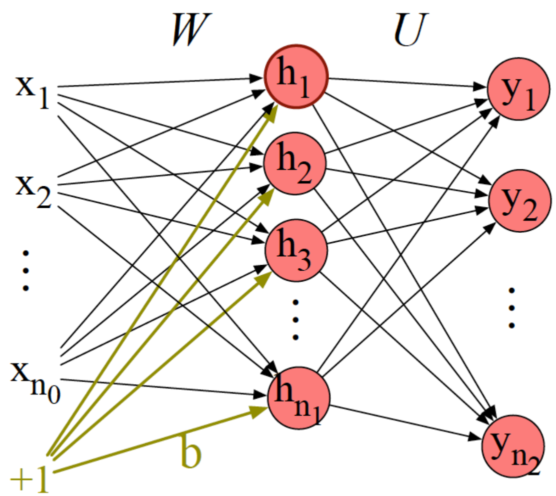
$$
\begin{aligned}
\mathbf{z}^{[1]} &= \mathbf{W}^{[1]}\mathbf{a}^{[0]} + \mathbf{b}^{[1]} \\
\mathbf{a}^{[1]} &= g^{[1]}(\mathbf{z}^{[1]}) \\
\mathbf{z}^{[2]} &= \mathbf{W}^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]} \\
\mathbf{a}^{[2]} &= g^{[2]}(\mathbf{z}^{[2]}) \\
\hat{\mathbf{y}} &= \mathbf{a}^{[2]}
\end{aligned}
$$

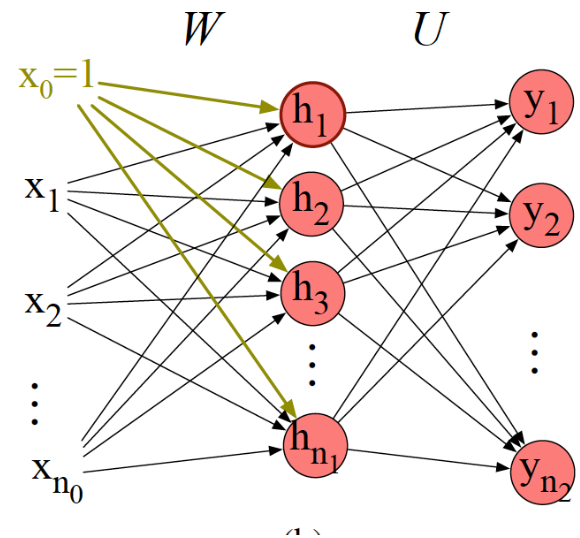
$$
\begin{aligned}
h &= \sigma(Wx+b) \\
z &= Uh \\
y &= \mathrm{softmax}(z)
\end{aligned}
$$

$$\mathrm{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad 1 \le i \le D$$

Notations

Notationally absorbing the bias:





$$\mathbf{h}_j = \sigma\left(\sum_{i=1}^{n_0} \mathbf{W}_{ji}\mathbf{x}_i + \mathbf{b}_j\right)$$

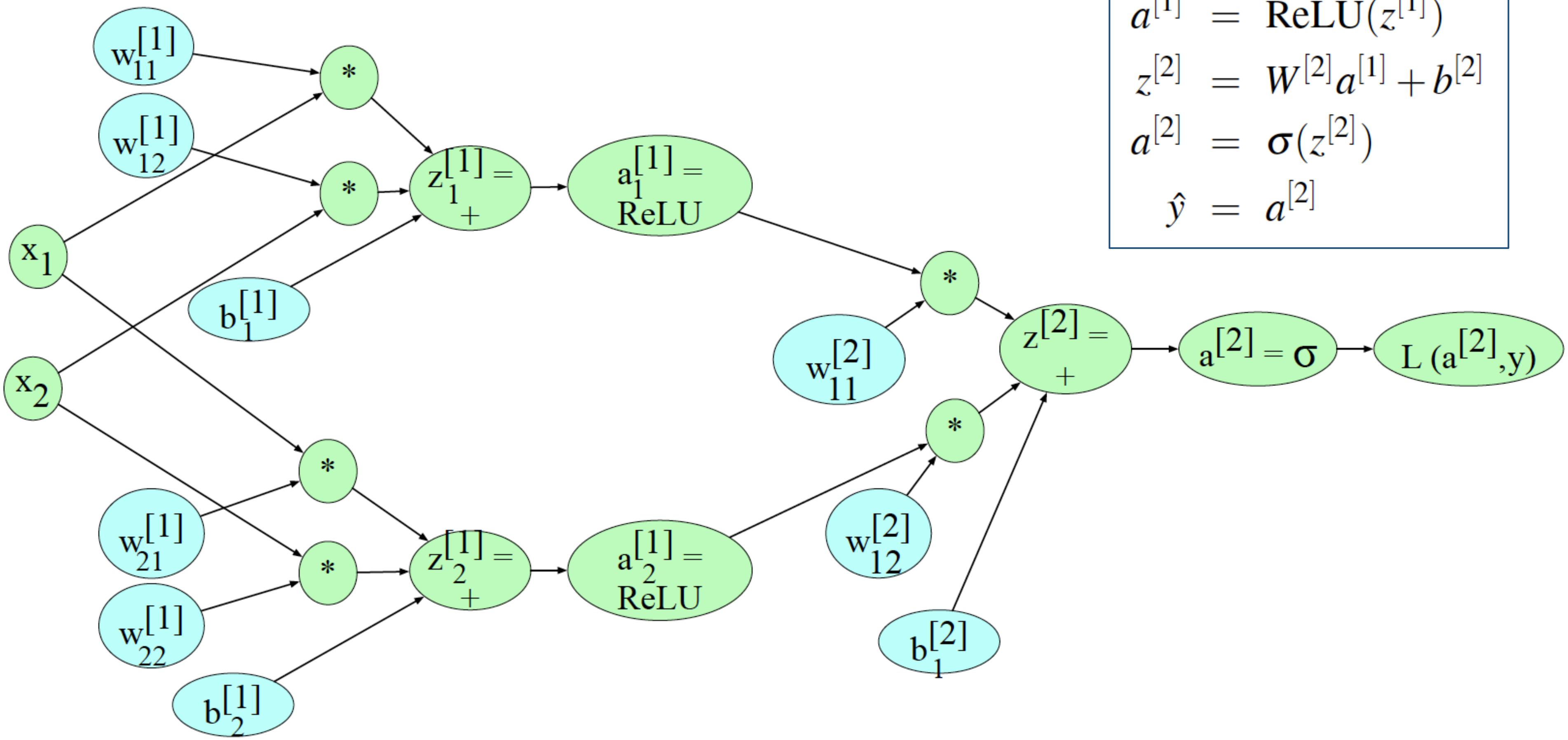
$$\mathbf{h}_j = \sigma\left(\sum_{i=0}^{n_0} \mathbf{W}_{ji}\mathbf{x}_i\right)$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x})$$

Forward Pass: Computation Graphs

$$
\begin{aligned}
z^{[1]} &= W^{[1]}\mathbf{x} + b^{[1]} \\
a^{[1]} &= \mathrm{ReLU}(z^{[1]}) \\
z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
a^{[2]} &= \sigma(z^{[2]}) \\
\hat{y} &= a^{[2]}
\end{aligned}
$$



Computation graph for simple 2-layer NN (two input units, two hidden units)

Repetition from ML1: Loss functions

- Loss function in general:

$$L(\hat{y}, y) = L(f(x; \boldsymbol{\theta}), y) = \text{How much } f(x) \text{ differs from the true } y$$

- Example for loss function: e.g. for regression case: → MSE

$$L_{MSE}(\theta) = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}^{(i)}(\theta) - y^{(i)})^2$$

- Example for loss function:
multi-class classification: for a softmax'ed K-neuron output:

$$p(y = k | \mathbf{x}, \theta) = \frac{\exp(a_k(\mathbf{x}, \theta))}{\sum_{k'=1}^{K} \exp a_{k'}(\mathbf{x}, \theta)} = \hat{y}_k(\mathbf{x}, \theta)$$

→ cross entropy:

$$L_{CE}(\theta) = \sum_{n=1}^{N} L_{CE n}(\theta) = -\sum_{n=1}^{N} \sum_{k=1}^{K} y_k^{(n)} \log \hat{y}_k(\mathbf{x}^{(n)}, \theta)$$

$$\overset{\text{for a one-hot encoded true } y^{(n)}}{=} -\sum_{n=1}^{N} \log \hat{y}_{correctclass}(\mathbf{x}^{(n)}, \theta)$$

Repetition from ML1: Gradient Descent

- NN learning: preferrably in (mini-)batches of M training examples: $x^{(n)}$ → forward pass → $\hat{y}^{(n)}(\theta)$

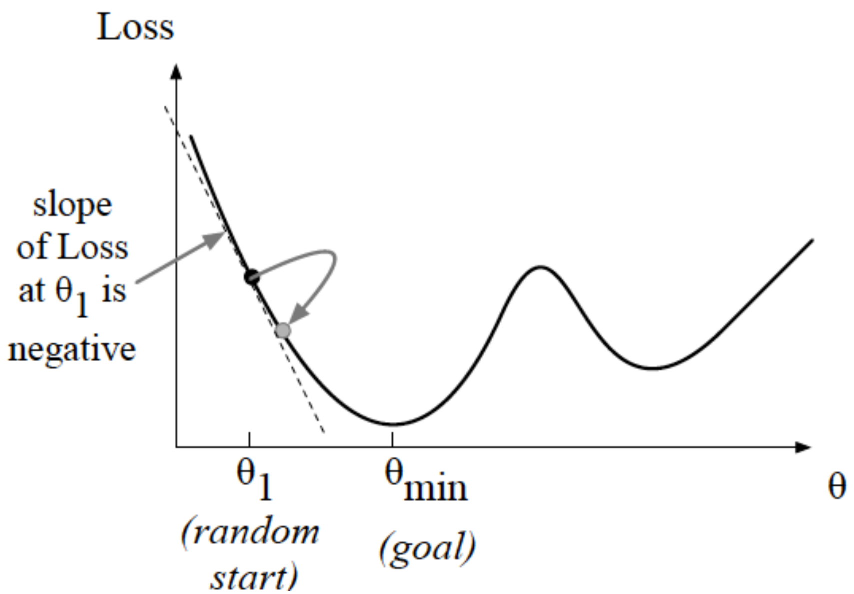
- compute loss $\mathrm{L}(\hat{y}^{(n)}(\theta), y^{(n)})$

- backward pass / Backpropagation → compute

$$\nabla_\theta L = \sum_{i=1}^{N} \nabla_\theta \, \mathrm{L}(\hat{y}^{(n)}(\theta), y^{(n)})$$

- stochastic gradient descent, ADAM etc.: randomly sample some $(x^{(i)}, y^{(i)})$ (or a whole minibatch) and do smth. similar to

$$\theta_{t+1} = \theta_t - \eta \, \nabla \mathrm{L}\big(\hat{y}^{(n)}(\theta), y^{(n)}\big)$$



Back-Propagation

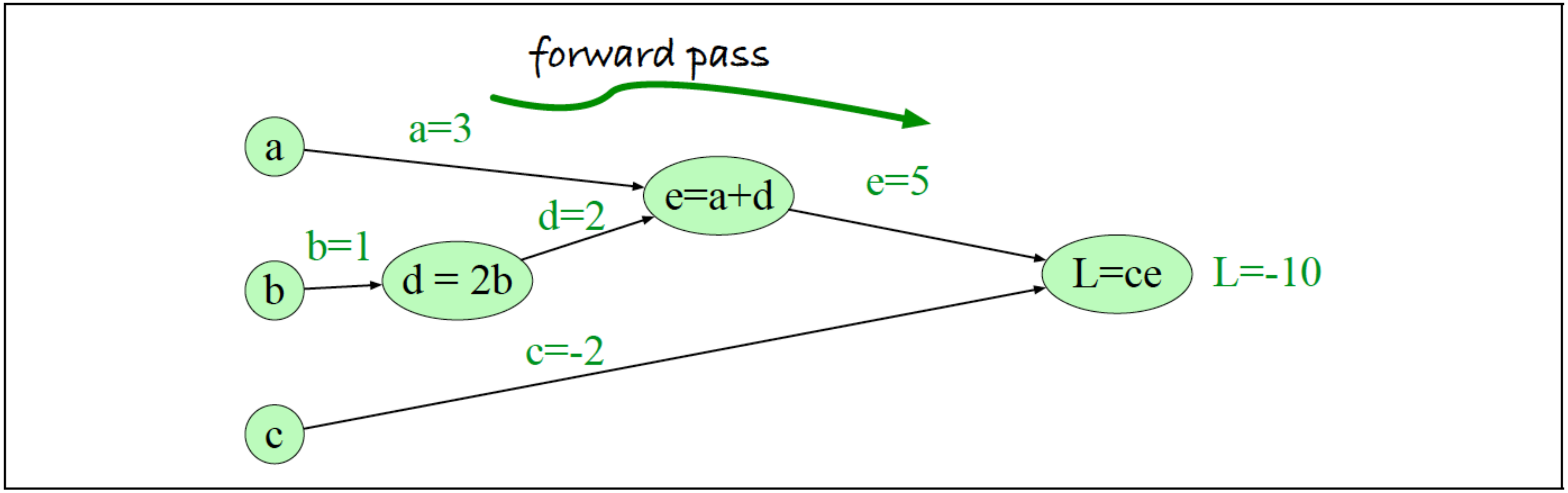


Figure 7.14 Computation graph for the function $L(a,b,c) = c(a+2b)$, with values for input nodes $a = 3$, $b = 1$, $c = -2$, showing the forward pass computation of L.

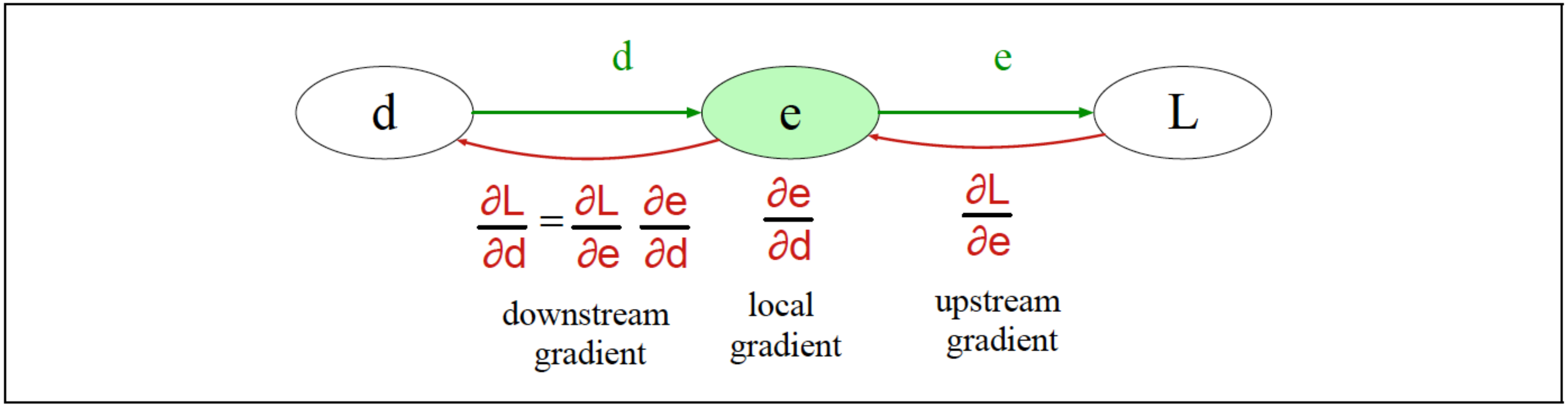
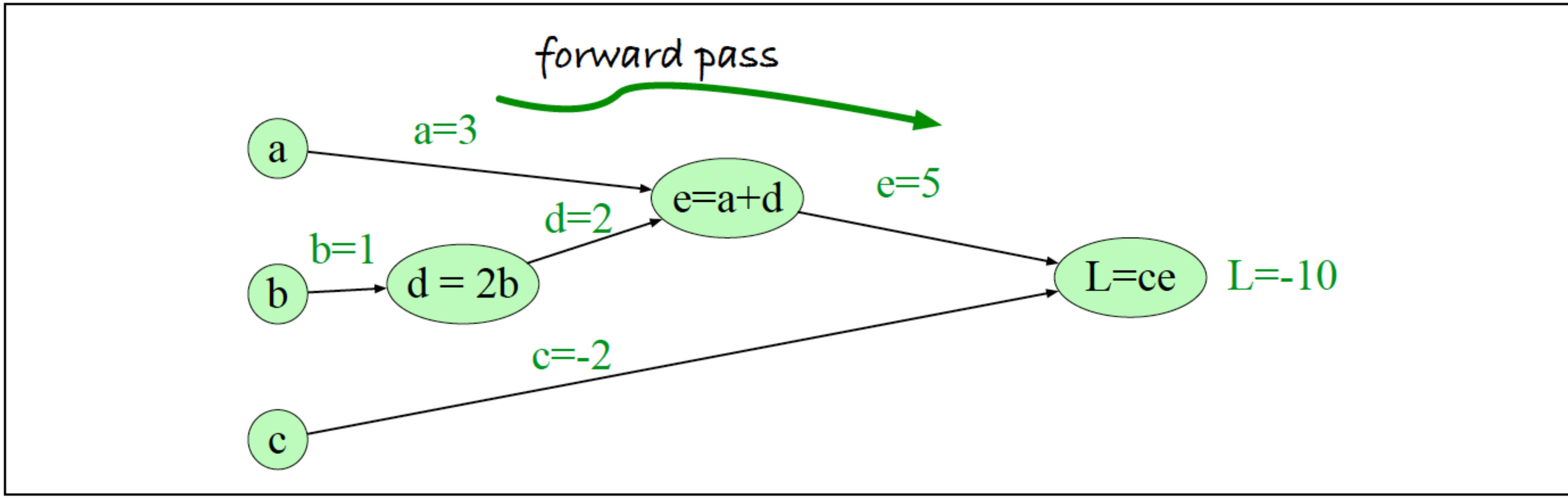


Figure 7.15 Each node (like e here) takes an upstream gradient, multiplies it by the local gradient (the gradient of its output with respect to its input), and uses the chain rule to compute a downstream gradient to be passed on to a prior node. A node may have multiple local gradients if it has multiple inputs.

Back-Propagation



Figu r input

node

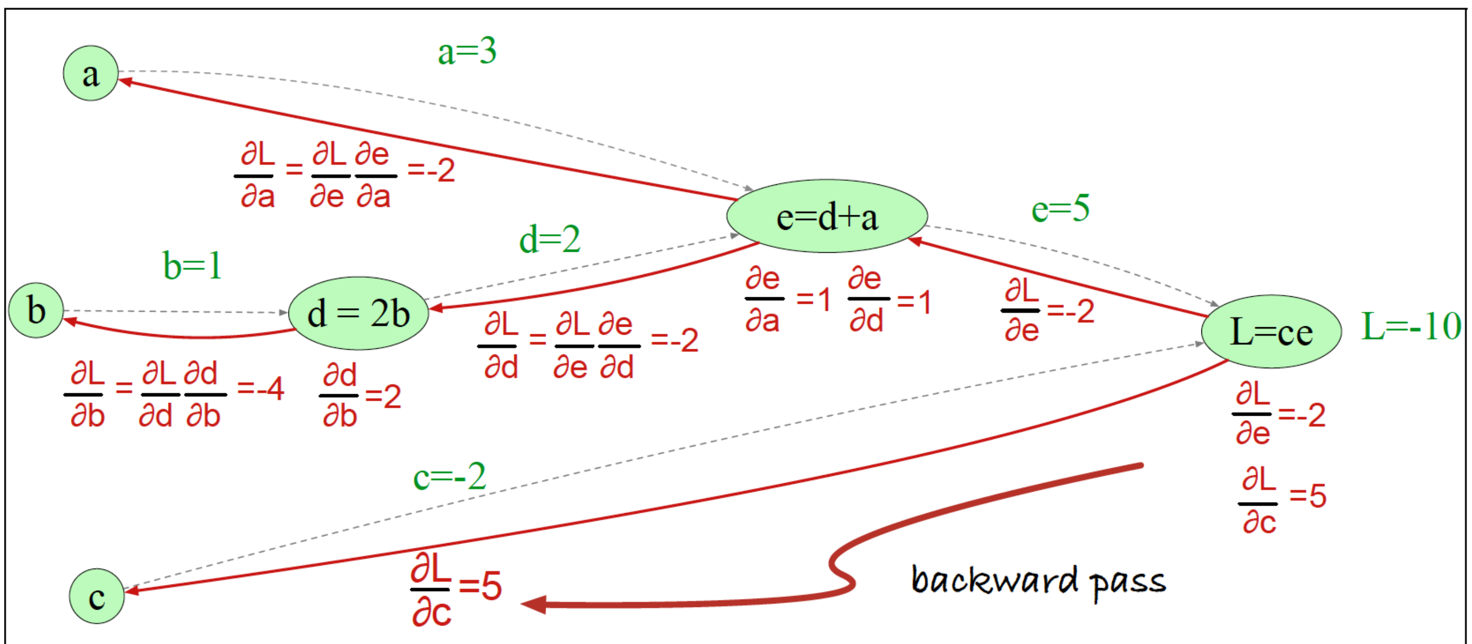


Figure 7.16 Computation graph for the function $L(a,b,c) = c(a+2b)$, showing the backward pass computation of $\frac{\partial L}{\partial a}$, $\frac{\partial L}{\partial b}$, and $\frac{\partial L}{\partial c}$.

Application: Language Modelling

- in chapter 4: smoothed N-Gram models for language modelling

$$P(w_t|w_1,\ldots,w_{t-1}) \approx P(w_t|w_{t-N+1},\ldots,w_{t-1})$$

- now use deep FF NN (later: RNN LSTM, Transformers etc.) for that task:
  - Input: N previous words.
  - Output: probability distribution over all $|V|$ possible candidate words

- standard representation for words: word embeddings: vectors ∈ $\mathbb{R}^d$ → vector space model of meaning: word-vectors that are ‚near' in vector space correspond to words with similar or related meaning

I forgot when I got home to feed the…

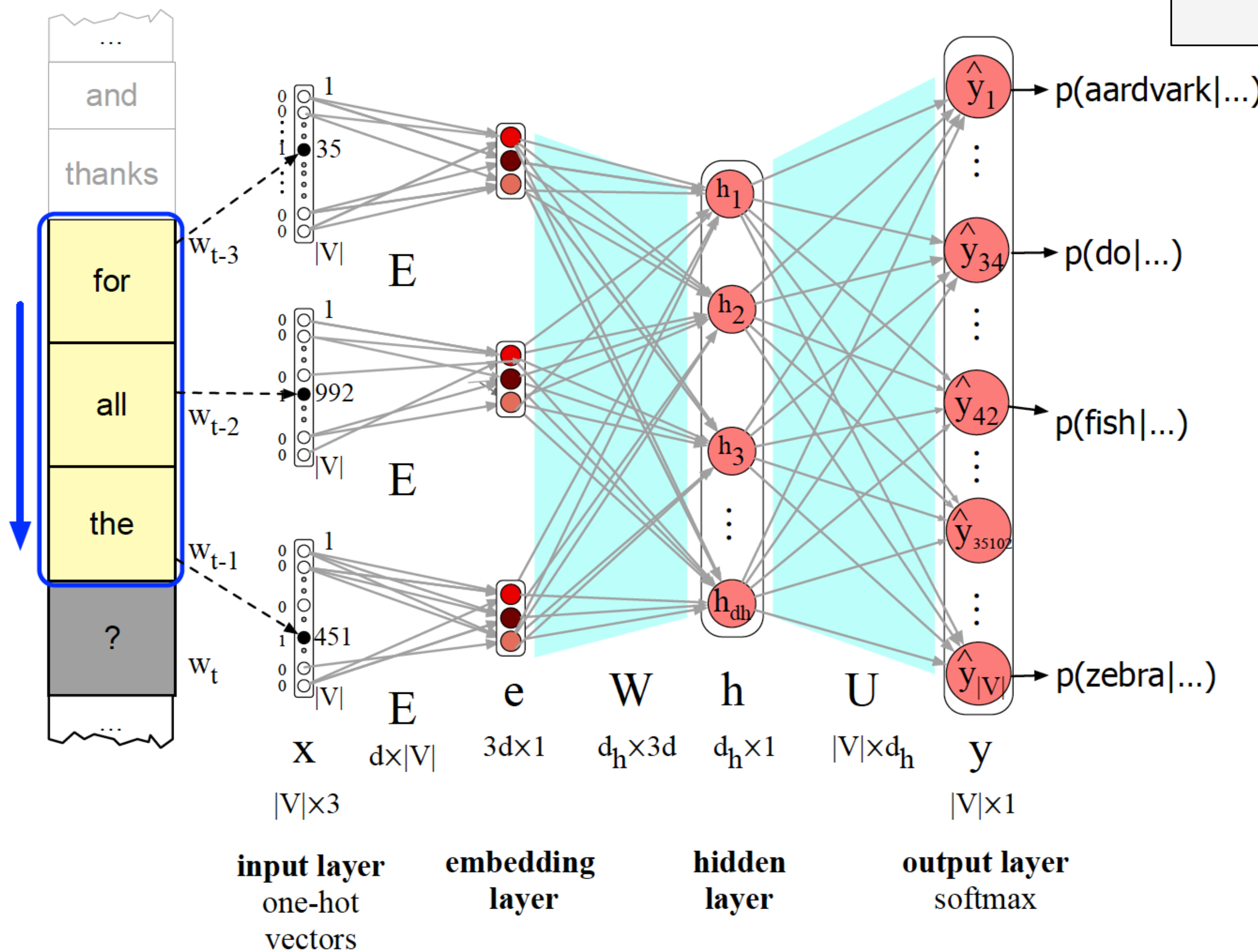
standard smoothed N-Gram language model:
p(cat|…) large, p(dog|…) small
if we haven't seen dog in this context in the corpus.

NN language model with pretrained embeddings (word vectors) as features:
p(cat|…) large, p(dog|…) large
even if we haven't seen dog in this context in the corpus.

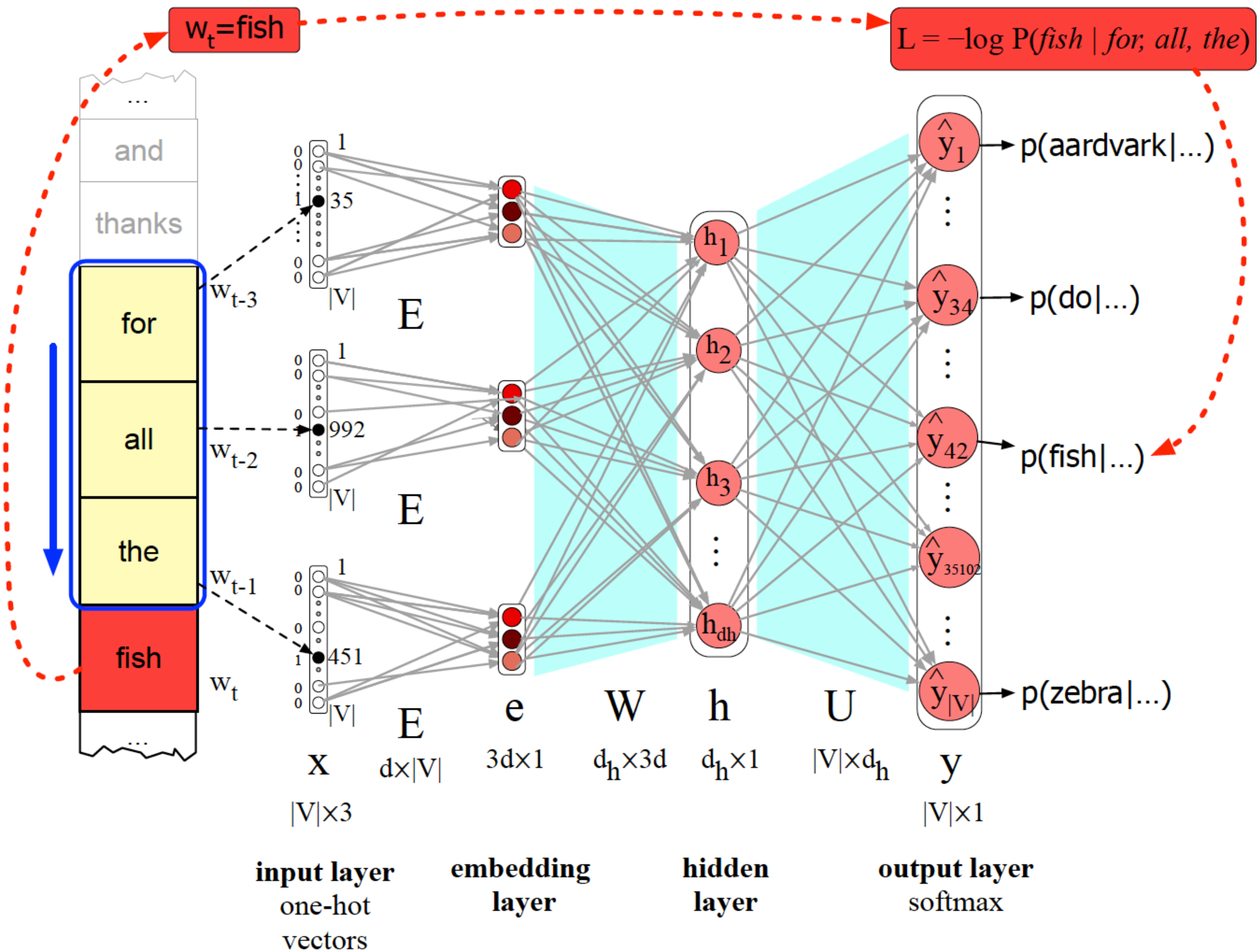
Application: Language Modelling

$$e = (Ex_1, Ex_2, ..., Ex)$$
$$h = \sigma(We + b)$$
$$z = Uh$$
$$y = \text{softmax}(z)$$

- example: N=3, known (pretrained) or unknown (trained here) embeddings : learn $P(w_t = i | w_{t-1}, w_{t-2}, w_{t-3})$



Application: Language Modelling: Training:



Bibliography

(1) Dan Jurafsky and James Martin: Speech and Language Processing (3rd ed. draft); Online: https://web.stanford.edu/~jurafsky/slp3/ (URL, May 2018)

(2)

Recommendations for Studying

- minimal approach:

  work with the slides and understand their contents! Think beyond instead of merely memorizing the contents

- standard approach:

  minimal approach + read the corresponding pages in Jurafsky [1]

- interested students

  == standard approach