PA164 Natural Language Learning

Lecture 07: Deep neural networks for NLP

Vít Nováček

Faculty of Informatics, Masaryk University

Autumn, 2022

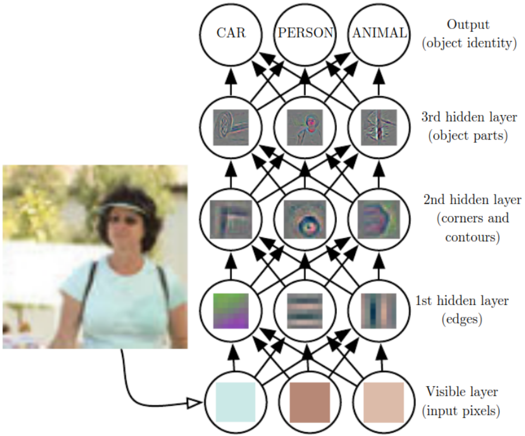
MUNI

Outline

History of neural networks

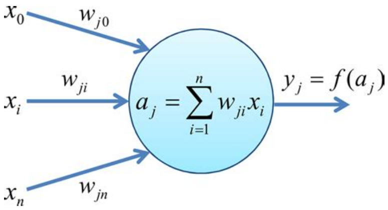
- Key motivating factors
  - The drawbacks of logics-based attempts at AI
    - Reliance on formal knowledge bases and rigid rules
    - Lots of manual work necessary
    - Some relevant problems can hardly ever be formalised
  - Drawing inspiration from nature
    - Machines acquiring their own knowledge
    - Extracting patterns from raw data
    - Learning not only patterns but the very features describing the data
    - Making use of neural architectures inspired by the human brain
- Selection of historical milestones
  - Single neural computation units: 1940s-1950s
  - Stochastic gradient descent for linear models: 1960s
  - Back-propagation: 1980s
  - Sequence modelling: 1990s
  - Deep learning boom: from 2010s on

The gist of DL: stacked representation learning



[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016. (Chap. 1)

Basic notions: perceptron

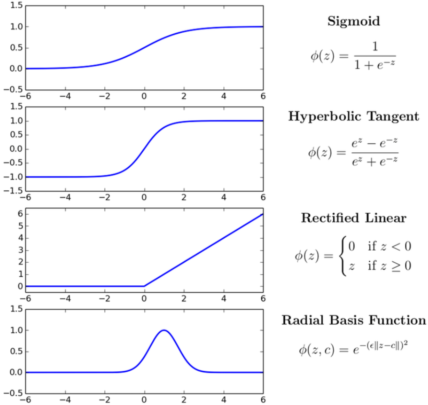


- Perceptron as a linear binary classifier:
  - $y_j = f(a_j) = f(\mathbf{w} \cdot \mathbf{x}) = 1$ if $\mathbf{w} \cdot \mathbf{x} > 0$
  - otherwise $y_j = 0$
- Learning process:
  - Init the **w** vector to random values
  - In each learning "epoch", randomly select one training example **x**
    - ★ If the example **x** is positive and $\mathbf{w} \cdot \mathbf{x} < 0$, then $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}$
    - ★ If the example **x** is negative and $\mathbf{w} \cdot \mathbf{x} > 0$, then $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{x}$
  - Repeat until (approximate) convergence

[2] Coop, Robert Austin. "Mitigation of Catastrophic Interference in Neural Networks and Ensembles using a Fixed Expansion Layer." (2013).

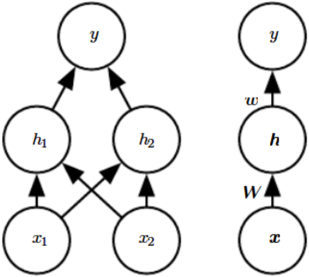
Basic notions: activation functions

- Alternatives of the f function from the perceptron example



[3] Hughes, Dana, and Nikolaus Correll. "Distributed machine learning in materials that couple sensing, actuation, computation and communication." arXiv preprint arXiv:1606.03508 (2016).

Basic notions: multi-layer perceptron



[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016. (Chap. 6)

Basic notions: why are activation functions essential (the XOR example)

- First layer weights (**W**), bias (**c**), output weights (**w**), input batch (**X**):

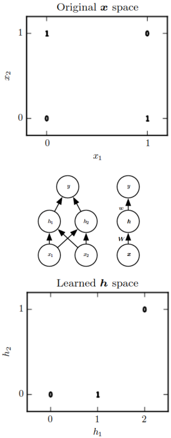
$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

$$\boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad \boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$\boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix},$$

- **XW**, **XW** with the bias **c** added row-wise, ReLU application and multiplication of the result by **w**:

$$\begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$



[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016. (Sec. 6.1)

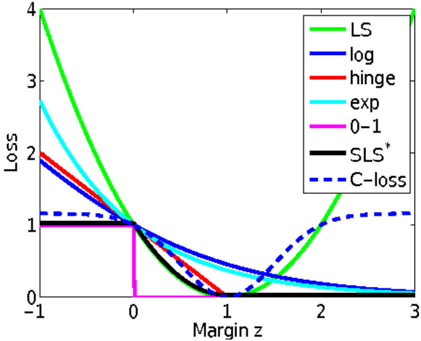
Basic notions: output units

- Quite like activation functions of the hidden units
- They have a special purpose, though:
  - First, they produce a model output $\hat{y}$ (usually a vector or a scalar, depending on the problem and the objective/loss function of choice)
  - The $\hat{y}$ value is then compared with the corresponding desired output y (i.e., label of the training example $\mathbf{x}$) via the loss function
  - The resulting error is back-propagated to update the model parameters
- Examples of often-used output units
  - Linear (simple final transformation): $\hat{\mathbf{y}} = \mathbf{W}^\top \mathbf{h} + \mathbf{b}$
  - Sigmoid (binary classification): First, use a linear layer to compute $z = \mathbf{w}^\top \mathbf{h} + b$, then convert z to a probability as $\hat{y} = \frac{1}{1-e^{-z}}$
  - Softmax (multiclass problems): First, a linear layer predicts unnormalised log probabilities $\mathbf{z} = \mathbf{W}^\top \mathbf{h} + \mathbf{b}$, where $z_i = log\ \tilde{P}(y = i|\mathbf{x})$, which is then normalised to obtain the desired $\hat{\mathbf{y}}$ probabilities as $\mathrm{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$

[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016. (Sec. 6.2)

Basic notions: loss/objective functions

- Examples of loss functions:

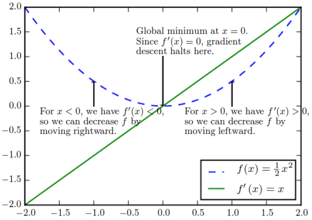


- In deep learning, the cross-entropy loss is often used
- Compares whole produced and desired distributions

[4] Xu, Guibiao, Bao-Gang Hu, and Jose C. Principe. "An asymmetric stagewise least square loss function for imbalanced classification." 2014 International Joint Conference on Neural Networks (IJCNN). IEEE, 2014.

Basic notions: gradient-based learning

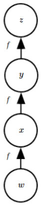
- The goal: minimise an objective (i.e., loss) function f with multiple inputs (i.e., find such vector $\mathbf{x}$ that $f(\mathbf{x})$ is the lowest possible number)
- The solution:
  - Pick a random $\mathbf{x}$ value
  - Find the direction from $\mathbf{x}$ in which f decreases the fastest
  - In other words, move to a new point $\mathbf{x}' = \mathbf{x} - \epsilon \nabla_{\mathbf{x}} f(\mathbf{x})$, where:
    - ϵ is the learning rate,
    - $\nabla_{\mathbf{x}} f(\mathbf{x})$ is the vector of all partial derivatives $\frac{\delta}{\delta x_i} f(\mathbf{x})$ (i.e., the gradient)
- A simple example for a function of one variable ($\frac{1}{2}x^2$):



[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016. (Sec. 4.3)

Basic notions: the gist of back-propagation

- An efficient method for computing the gradient in practice
- A differentiable loss function computes the error, i.e., the difference between the actual and the desired output y of the network based on the input vector $\mathbf{x}$
- The error is then back-propagated through the network by means of the chain rule of calculus, as in the following simple example:

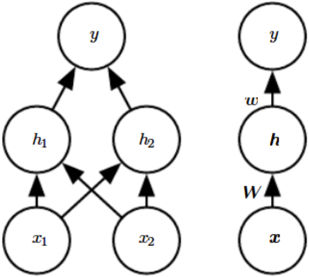


- $\frac{\delta z}{\delta w} =$
- $= \frac{\delta z}{\delta y} \frac{\delta y}{\delta x} \frac{\delta x}{\delta w} =$
- $= f'(y) f'(x) f'(w) =$
- $= f'(f(f(w))) f'(f(w)) f'(w)$

[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016. (Sec. 6.5)

Outline

Feedforward neural networks: synonym for MLPs

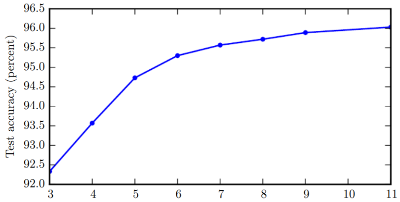


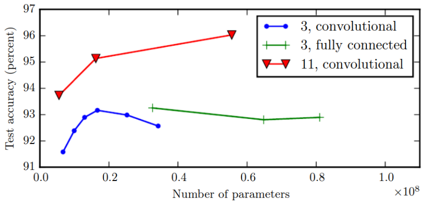
[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016. (Chap. 6)

Feedforward neural networks: practical considerations

- Universal approximation
  - A feedforward network with a linear output layer and at least one hidden layer with any "squashing" activation (such as logistic sigmoid)...
  - ...can approximate virtually any practical function with any desired non-zero amount of error...
  - ...given enough hidden units.
- That doesn't necessarily mean the network can also efficiently learn the function, though
- In practice, depth often wins over breadth

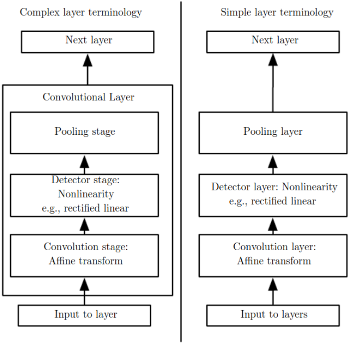
- Depth vs. number of parameters





[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016. (Sec 6.4)

Convolutional neural networks

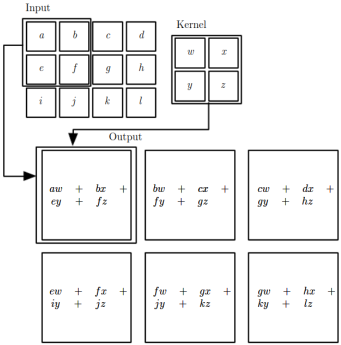


- Great for grid-like input (e.g., image tensors)
- Replacing (some) expensive matrix multiplications by convolutions
  - Affine linear transformation of the input via a much smaller kernel
- Non-linear "detection" stage on top of the linear convolution
- Pooling (e.g., maximum value within a rectangular region) then makes the representation approximately invariant to translations in the input

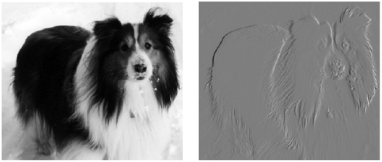
[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016. (Sec. 9.3)

Convolution examples

- Sample kernel and its application



- Subtraction of neighbouring pixels for edge detection

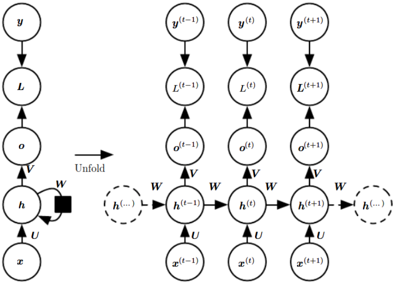


[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016. (Sec. 9.2)

Outline

Recurrent neural networks

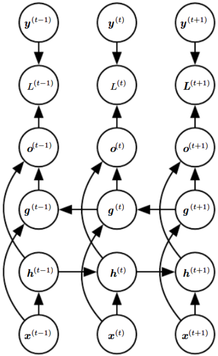
- Motivated by the need for sequence modelling (e.g., in NLP)
- Generalising the computational graphs for NN representation
  - Loops to represent influence of node values on their future values
  - Unfolding of the computational graph into a sequence of steps (corresponding to minibatches in which RNNs typically process inputs)
  - The information flow in such networks allows to learn patterns of relationships between sequence elements (very useful in NLP)



[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016. (Sec. 10.2)

Bidirectional RNNs

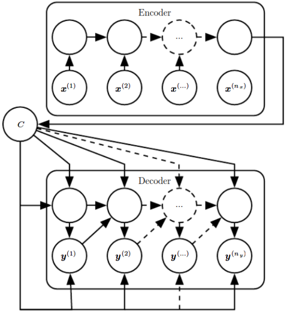
- Generalisation of recurrent neural networks that lets the information flow in both directions
- Allows for learning more complex relationships (both past and future influences between sequence elements)



[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016. (Sec. 10.3)

Encoder-decoder models

- Sequence-to-sequence mapping, for instance in machine translation
  - One model (usually a RNN, sometimes also a CNN) converts the input sequence to an intermediate semantic representation (a context summary)
  - Another model (typically another RNN) then converts the semantic representation to an output sequence



[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016. (Sec. 10.4)

The challenge of long-term dependencies

- Major practical limitation of RNNs
- Gradients propagated over long sequences tend to vanish (or, less often, explode):
  - Consider recurrence relation modelled as $\mathbf{h}^{(t)} = \mathbf{W}^\top \mathbf{h}^{(t-1)}$
  - This can be simplified to $\mathbf{h}^{(t)} = (\mathbf{W}^t)^\top \mathbf{h}^{(0)}$
  - If $\mathbf{W}$ can be eigen-decomposed to $\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$, then the recurrence can be further simplified to $\mathbf{h}^{(t)} = \mathbf{Q}^\top \mathbf{\Lambda}^t \mathbf{Q} \mathbf{h}^{(0)}$
- In the scalar case of weight w, this is analogous to vanishing/exploding w^t, depending on whether $w < 1$ or $w > 1$, respectively

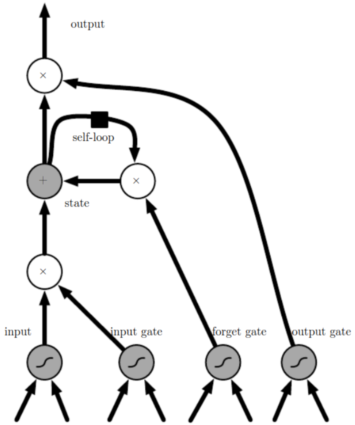
[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016. (Sec. 10.7)

Coping with the long-term dependencies

- Multiple time-scale models
  - Adding skip connections across multiple time steps to allow for more coarse-grained flow of information
  - Adding linear self-connections to nodes on critical paths and keeping the corresponding weights close to one (so called leaky units)
  - Removing fine-grained time connections
- Gated RNN architectures
  - Similar to the leaky units idea
  - Creating paths through time where gradients don't vanish/explode
  - Two key innovations, though:
    - ⋆ The "safe" weights are not manually set but learned like any other parameter
    - ⋆ Information is not only accumulated, but also forgotten (i.e., set to zero) when not needed anymore
  - Achieved by self-loops producing long gradient flow paths
  - The self-loops conditioned based on context – gating (weight controlled by another hidden unit)

[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016. (Sec. 10.10)

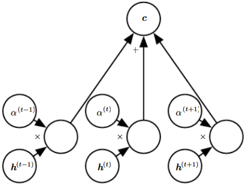
The long short-term memory (LSTM) gated model schema



[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016. (Sec. 10.10)

Attention mechanism

- Originally proposed to improve performance of encoder-decoder architectures in machine translation (2015-2017)
- Became a basis of virtually every neural model for NLP since then, though
- The gist of the approach:

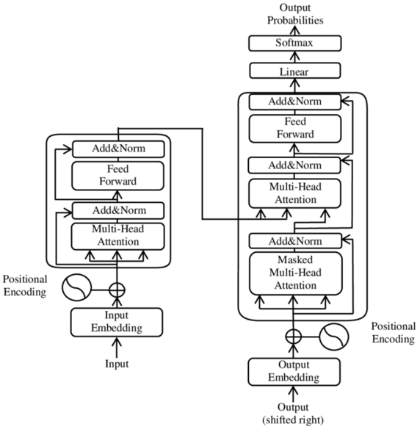


- The α weights produce a weighted average of the hidden feature vectors, forming the context representation of the input **c**
- The attention weights are usually computed as a softmax of relevance scores produced by a different portion of the model
- The mechanism can dynamically highlight portions of the sequence relevant for producing desired output
- This can often work better than arbitrarily complex RNN or CNN architecture

[1] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016. (Sec. 12.4.5.1)

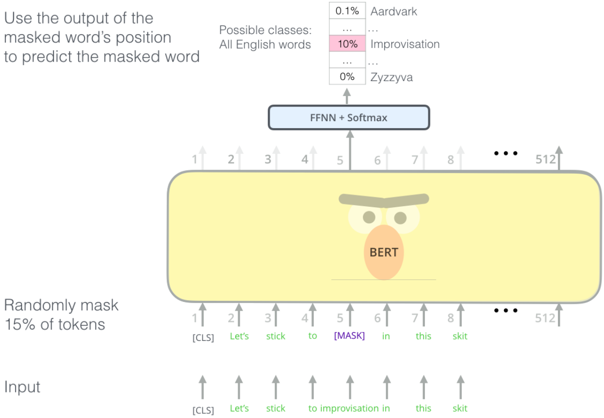
Transformers

- The state-of-the-art neural NLP models of today
- Encoder-decoder overall design (decoder-only possible as well)
- Example of a transformer architecture schema:



[5] Original image (DOI:10.1088/1742-6596/1314/1/012186) created by Yuening Jia, available under the CC BY-SA 3.0 license.

Modern language models: the gist



[6] Jay Alammar. "The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)."

`http://jalammar.github.io/` blog (2018-2021).

Selected language models

- BERT
  - A Google model based on the original attention-enabled encoder-decoder paper
  - Widely used and "forked" (many bespoke variants of pretrained BERT)
- GPT*
  - A series of large OpenAI models
- BLOOM
  - A large and free model initiated by a co-founder of Hugging Face
- OPT
  - A large, open language model released by Meta AI to the scientific community
- DALL-E and CLIP
  - Multimodal OpenAI models for creating images from prompts, and vice versa
- Hugging Face – a company and a portal making many SoA language models available to the public

Outline

Further readings on deep learning in general

- Deep learning overview
  - Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016.
  - LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." nature 521.7553 (2015): 436-444.
- Selected historical works
  - Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." Psychological review 65.6 (1958): 386.
  - Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." nature 323.6088 (1986): 533-536.

Further readings on specific architectures

- The classical architectures
  - LeCun, Yann, et al. "Backpropagation applied to handwritten zip code recognition." Neural computation 1.4 (1989): 541-551.
  - Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, 2010.
  - Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Communications of the ACM 60.6 (2017): 84-90.
- The modern architectures
  - Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).
  - Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).
  - Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

Further readings on language models

- Textual models
  - Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).
  - Peters, Matthew E., et al. "Deep contextualized word representations." arXiv preprint arXiv:1802.05365 (2018).
  - Brown, Tom, et al. "Language models are few-shot learners." Advances in neural information processing systems 33 (2020): 1877-1901.
  - Petroni, Fabio, et al. "Language Models as Knowledge Bases?." Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). 2019.
- Multimodal models
  - Kiros, Ryan, Ruslan Salakhutdinov, and Rich Zemel. "Multimodal neural language models." International conference on machine learning. PMLR, 2014.
  - Kiros, Ryan, Ruslan Salakhutdinov, and Richard S. Zemel. "Unifying visual-semantic embeddings with multimodal neural language models." arXiv preprint arXiv:1411.2539 (2014).