Gépi tanulás 2. – Neurális hálók

Simon Eszter

2021. február 22.

PIM DBK

Tartalom

Bevezetés

Gépi tanulás

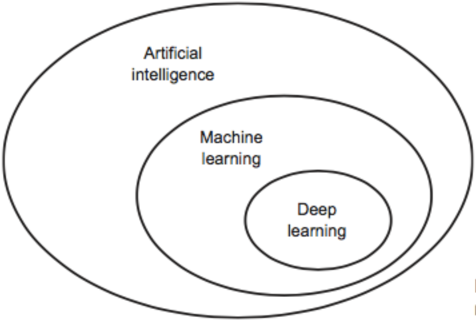


Figure 1.1 Artificial intelligence, machine learning, and deep learning

The ‘deep’ in deep learning

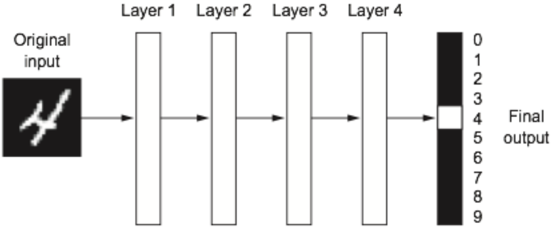
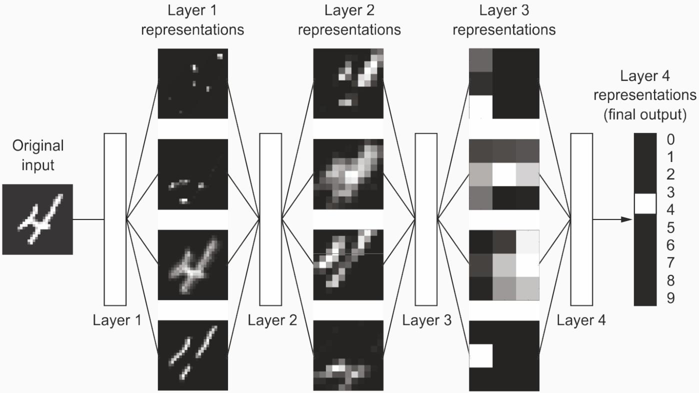


Figure 1.5 A deep neural network for digit classification

Layers and representations



Understanding how deep learning works 1.

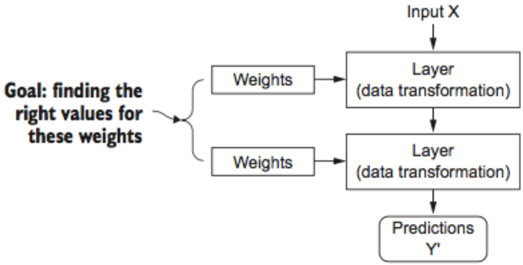


Figure 1.7 A neural network is parameterized by its weights.

Understanding how deep learning works 2.

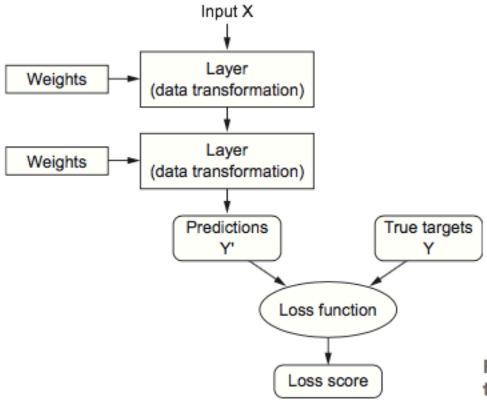


Figure 1.8 **A loss function measures the quality of the network's output.**

Understanding how deep learning works 3.

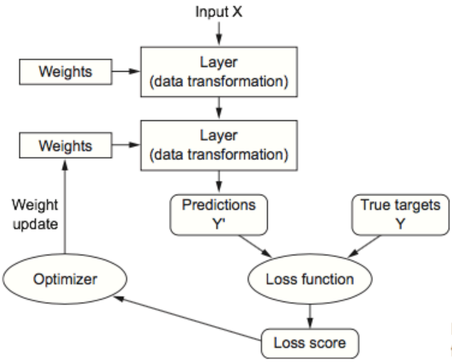


Figure 1.9 The loss score is used as a feedback signal to adjust the weights.

the fundamental trick is to use the loss score as a feedback signal to adjust the value of the weights a little, in a direction that will lower the loss score

Training loop

- kezdetben a súlyok random értékek → az output távol van az ideálistól, a loss score nagyon magas
- a súlyok minden egyes tanulási kör során egy kicsit módosulnak → a loss score kisebb lesz
- ha ezt a tanulási kört elégszer iteráljuk, akkor elérjük a loss score minimumát
- a minimális loss score-ral rendelkező rendszer kimenete lesz a legközelebb a gold standardhez

Történeti áttekintés

The promise of AI

mottó:

"Don't believe in the short-term hype, but do believe in the long-term vision."

a deep learning sok mindenre jó, de nem mindenre a legjobb eszköz:

- kevés az adat
- más algoritmus jobban használható az adott feladatra

AI winter: high expectations for the short term $\rightarrow$ technology fails to deliver $\rightarrow$ research investment dries up, slowing progress for a long time

1. 1960s: symbolic AI
   Marvin Minsky 1967: "Within a generation … the problem of creating artificial intelligence will substantially be solved."
   1969-70: first AI winter
2. 1980s: expert systems
   a few initial success stories $\rightarrow$ expensive to maintain, difficult to scale, and limited in scope
   early 1990s: second AI winter

- 1940s: McCulloch–Pitts neuron: a simplified model of the human neuron as a kind of computing element
- 1950/60s: perceptron (Rosenblatt, 1958), bias (Widrow and Hoff, 1960), XOR (Minsky and Papert, 1969)
- 1980s: backpropagation (Rumelhart et al., 1986), handwriting recognition with backpropagation and convolutional neural networks (LeCun et al., 1989)
- 1990s: recurrent networks (Elman, 1990), Long Short-Term Memory (1997)
- 2010s: Geoffrey Hinton et al., Yoshua Bengio et al.

Why now?

Hardware

- Graphical Processing Unit (GPU): developed for gaming
- 2007: NVIDIA launched CUDA, a programming interface for its line of GPUs
- a small number of GPUs can replace massive clusters of CPUs
- parallelizable matrix multiplications
- 2016: Tensor Processing Unit (TPU) by Google

Data

"if deep learning is the steam engine of this revolution, then data is its coal"

Why now? – cont.

Algorithms

The feedback signal used to train neural networks would fade away as the number of layers increased.

- better activation functions
- better weight-initialization schemes
- better optimization schemes

Only when these improvements began to allow for training models with 10 or more layers did deep learning start to shine.

A new wave of investment

total investment in AI: 2011: \$19 million $\rightarrow$ 2014: \$394 million

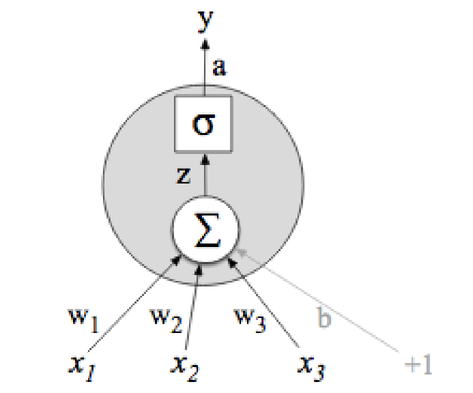
Why now? – cont.

The democratization of deep learning

early days: doing deep learning required significant programming expertise $\rightarrow$ now: basic Python scripting skills are sufficient (PyTorch, TensorFlow, Keras) $\rightarrow$ no feature engineering

Units

A neural unit



The building block of a neural network is a single computational unit. A unit takes a set of real valued numbers as input, performs some computation on them, and produces an output.

Bias

a neural unit is taking a weighted sum of its inputs, with one additional term in the sum called a bias term

$$z = b + \sum_i w_i x_i$$

expressing this weighted sum using vector notation: replacing the sum with dot product ($z \in \mathbb{R}$):

$$z = w \cdot x + b$$

Activation

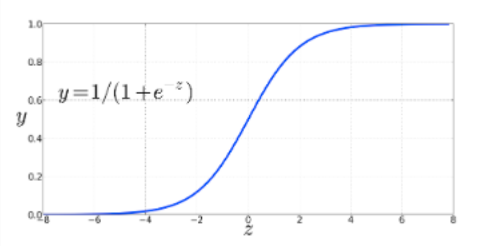
instead of using z, neural units apply a non-linear function f to $z \rightarrow$ the output of this function is the activation value for the unit a

$$y = a = f(z)$$

the final output of the network is y, and since here we have a single unit, y and a are the same

Non-linear functions – sigmoid

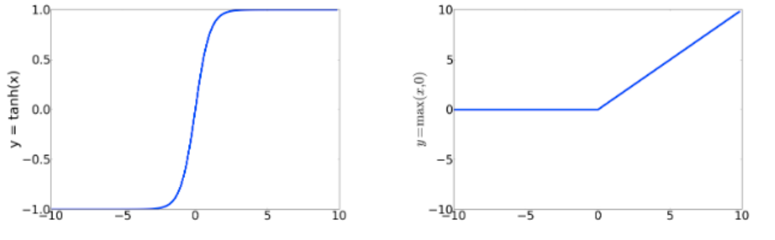
$$y = \sigma(z) = \frac{1}{1 + e^{-z}}$$



Non-linear functions – tanh and ReLU

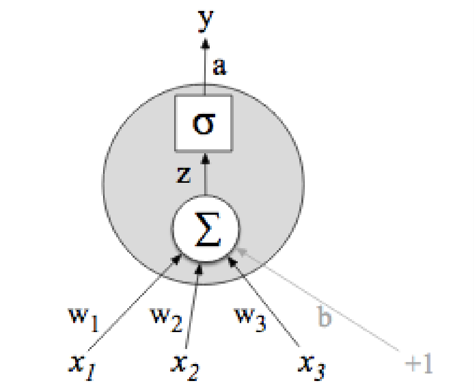
$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$y = max(x, 0)$$



(a)
(b)

Summary – a unit



Feedforward Neural Networks

A feedforward network

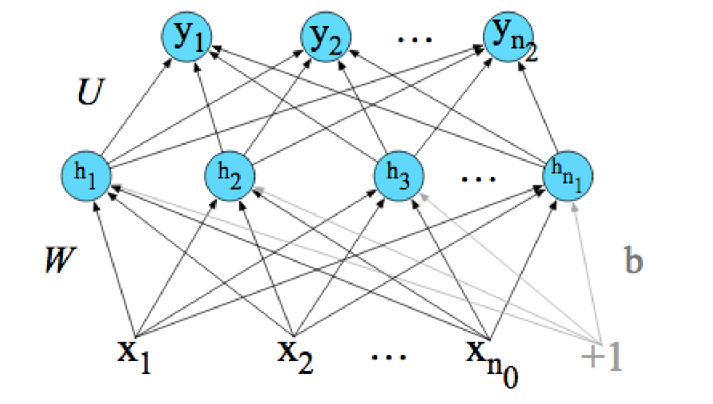
a feedforward network

is a multilayer network

- in which the units are connected with no cycles;
- the outputs from units in each layer are passed to units in the next higher layer, and
- no outputs are passed back to lower layers

(networks with cycles are called recurrent neural networks (RNNs))

Three kinds of nodes



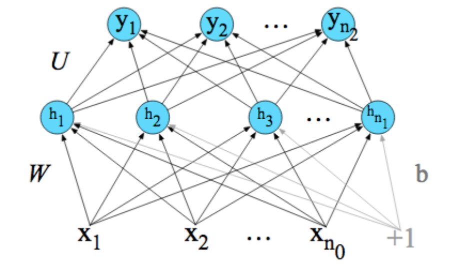
input units, hidden units, and output units

The hidden layer

- the hidden layer is formed of hidden units, each of which is a neural unit, taking a weighted sum of its inputs and then applying a non-linearity
- fully-connected: each hidden unit sums over all the input units

Weight matrix

We represent the parameters for the entire hidden layer by combining the weight vector w_i and bias b_i for each unit i into a single weight matrix W and a single bias vector b for the whole layer. Each element W_{ij} of the weight matrix W represents the weight of the connection from the ith input unit x_i to the jth hidden unit h_j.



The W matrix

	x_1	x_2	x_3
h_1	w_{11}	w_{12}	w_{13}
h_2	w_{21}	w_{22}	w_{23}
h_3	w_{31}	w_{32}	w_{33}
h_4	w_{41}	w_{42}	w_{43}

Matrix operations

3 steps:

1. multiplying the weight matrix by the input vector x
2. adding the bias vector b
3. applying the activation function g

$$h = \sigma(Wx + b)$$

- the number of inputs: n_0
- x is a vector of real numbers of dimension n_0: $x \in \mathbb{R}^{n_0}$
- the hidden layer has dimensionality n_1, so $h \in \mathbb{R}^{n_1}$
- $W \in \mathbb{R}^{n_1 \times n_0}$

The role of the output layer

- the resulting value *h* forms a representation of the input
- the role of the output layer: to take this representation and compute the final output
- the output can be a real-valued number, but it is rather a probability distribution across the output nodes

Intermediate output

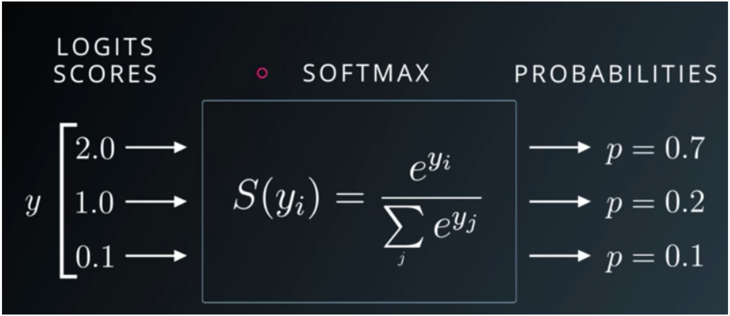
- the output layer also has a weight matrix (U)
- some models don't include a bias vector b, so here we eliminate it
- the weight matrix U is multiplied by the vector h to produce the intermediate output z:

$$z = Uh$$

- $U \in \mathbb{R}^{n_2 \times n_1}$
- element U_{ij} is the weight from unit j in the hidden layer to unit i in the output layer

converting a vector of real-valued numbers to a vector encoding a probability distribution:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}} 1 \leq i \leq d$$



Summary – feedforward network

the final equations for a feedforward network with a single hidden layer, which takes an input vector x, outputs a probability distribution y, and is parameterized by weight matrices W and U and a bias vector b:

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

activation functions:

- at the internal layers: ReLU or tanh
- at the final layer:
  - for binary classification: sigmoid
  - for multinomial classification: softmax

Training Neural Nets

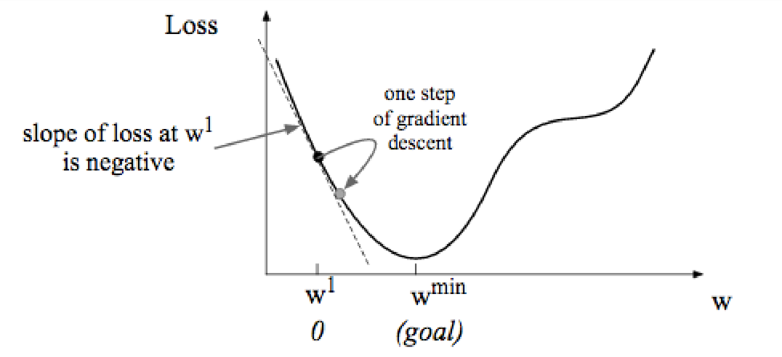
Supervised machine learning

- the correct output: y
- the system's estimate of the true y: $\hat{y}$
- the goal of the training procedure: to learn parameters $W^{[i]}$ and $b^{[i]}$ for each layer i that make $\hat{y}$ as close as possible to the true y

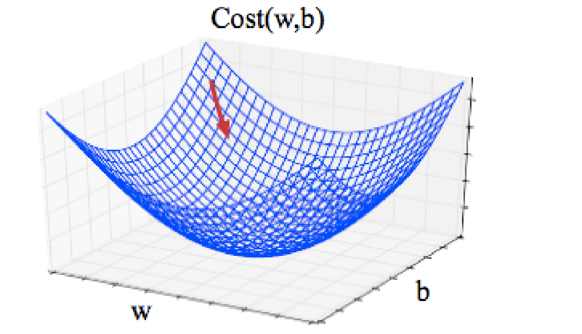
How to do that?

1. we need a loss function that models the distance between $\hat{y}$ and $y \rightarrow$ cross-entropy loss
2. we have to minimize the loss function $\rightarrow$ an optimization algorithm for iteratively updating the weights: gradient descent
3. we have to know the gradient of the loss function $\rightarrow$ error backpropagation

Computing the Gradient – one parameter



Computing the Gradient – two parameters



for more parameters → error backpropagation or backward differentiation → all parameters can be calibrated together

non-convex optimization problem with possible local minima

More details on learning

- to prevent overfitting $\rightarrow$ dropout: randomly dropping some units and their connections from the network during training
- tuning hyperparameters:
    - the number of layers
    - the number of hidden nodes per layer
    - the choice of activation functions
    - ...

Irodalom

Irodalom

- Jurafsky 3rd edition 7. chapter: https://web.stanford.edu/~jurafsky/slp3/7.pdf
- Francois Chollet: Deep Learning with Python. Manning, Shelter Island, 2018.: https://www.manning.com/books/deep-learning-with-python