Aalto University

CS-E4890: Deep Learning
Lecture 1: Introduction

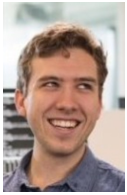
Alexander Ilin

Teaching assistants

Prakhar Verma

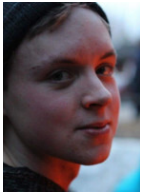
Miikka Runolinna

Kosta Stanojević



Sam Spilsbury

Trung Trinh



Joonatan Bergholm

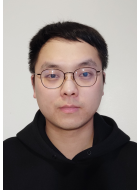
Çağatay Yıldız

Andrey Sukhobok

Lukas Prediger

Severi Rissanen

Paavo Reinikka



Rongtian Ye

Nicola Dainese

Contact information

If you have question regarding the course, please send an email to

cs-e4890@aalto.fi

Pre-requisites

- **Good knowledge of Python and numpy**
- Linear algebra: vectors, matrices, eigenvalues and eigenvectors.
- Basics of probability and statistics: sum rule, product rule, Bayes' rule, expectation, mean, variance, maximum likelihood, Kullback-Leibler divergence.
- Basics of machine learning (recommended): supervised and unsupervised learning, overfitting.

Course schedule

- Please study carefully the course schedule in mycourses.
  - 12 lectures (11 + 1 guest lecture on deep reinforcement learning)
  - 1 test assignment (deadline this Friday 23:00)
  - 10 assignments (the points are computed from 8 best)
  - Exercise sessions for assignments 2–9 (no exercise sessions for assignments 1, 10–11).
  - No exam (there is a placeholder for the exam in Oodi but no exam this year).

Test assignment

- We designed a very simple test assignment `01_test` to make sure that
  - you can write python code
  - you can understand the instructions in the notebooks of the course
  - you can understand documentation of machine learning libraries.
- **Important:** The deadline for `01_test` is this Friday 23:00.
- **If you find the test assignment difficult, the course level is not right for you.**
- The test assignment also includes important information and rules such as:
  - You give permission for proctoring your submissions.
  - **Solution sharing is strictly not allowed before, during and after the course.**

Communication channels

- Slack is the main communication channel: deeplearn21-aalto.slack.com
- Please ask questions about assignments in the dedicated channels.
- The teaching assistants (TAs) will look at slack regularly.

Course grading

- 5 credit points, 1-5 scale
- Grading is based on the number of points collected in eight best assignments. The grading rules are explained in mycourses.
- The course workload (5 credits) assumes solving eight assignments, the two extra assignments give you the possibility to improve your grade.

Assignments

- The assignments are released already.
- **Please read very carefully the instructions.**
- You can find the deadlines on the course schedule page in mycourses.
- **Strict deadlines, zero points for late submissions, no exceptions.**
- The feedback is returned on the same week after the deadline.
- If you plan to be away, submit your solutions early, no need to wait until the deadline.

Exercise sessions

- The exercise sessions are organized to help you solve the assignments, you do not have to attend them.
- The exercise sessions will be organized on Fridays and Mondays via zoom.
- **Please vote for your preferred day (Friday or Monday) in the #general channel in slack.**
- We will arrange exercise sessions according to your preferences (for example, one session on Friday and three sessions on Monday).
- The times of the sessions will be announced on the course schedule page.
- Please read carefully the protocol for the exercise sessions.

PyTorch

- We will not have special sessions on PyTorch, you should learn it by following online material.
  - If you know numpy (pre-requisite), PyTorch should be easy to learn.
  - Deep learning frameworks develop very quickly. If you do deep learning, you need to learn new frameworks/features all the time.
- If you need help with PyTorch and/or solving the assignments, please come to the exercise sessions!
- Why not Tensorflow or Keras?
  - Tensorflow is cumbersome, PyTorch is easier to learn and to use.
  - Keras is good if you want to try a deep learning model quickly. But it is more restrictive. If you want to be a professional deep learner, PyTorch is a better choice.

Course material

- Recommended book: Goodfellow, Bengio and Courville, 2016. Deep Learning.
- PyTorch tutorials
- Papers and links in the lecture slides and the assignments

Lectures

- The lectures are organized via zoom. They will be recorded and available from mycourses.
- The lecture slides are in mycourses.
- There will be changes in the slides, please download the latest version before each lecture.
- Credit to people whose material I used in the slides: Tapani Raiko, Kyunghyun Cho, Jyri Kivinen, Jorma Laaksonen, Antti Keurulainen, Sebastian Björkqvist.

What is deep learning

Feature engineering

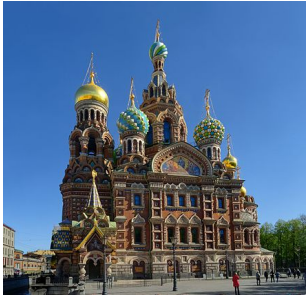
- Many machine learning tasks can be solved by designing the right set of features to extract for that task:

$$\text{Data} \rightarrow \text{Feature engineering} \rightarrow \text{Machine learning (e.g. classification)}$$

- Examples:
  - Spam detection: Useful features are counts of certain words.
  - Line item extraction from invoices: Useful features to classify a number as a line item or not are position on the invoice, words that appear in the proximity.
- Benefit of feature engineering: One can use domain knowledge to design features that are robust (for example, invariant to certain distortions).
- What are the problems with feature engineering?

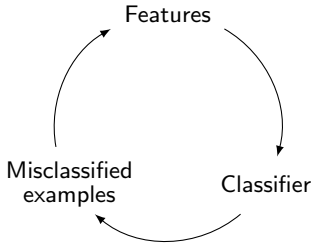
Feature engineering: Problem 1

- For many tasks, it is difficult to know what features should be extracted.
- Example: We want to detect certain buildings in images (two-dimensional maps of RGB values). What are useful features?
- Manually designing features for a complex task requires a great deal of human time and effort; it can take decades for an entire community of researchers to design good features.
  - Example: SIFT features in image classification.



Feature engineering: Problem 2

- Handcrafted features are not perfect. There are always examples that are not processed correctly, which motivates engineering of new features.



- Features can get very complex and difficult to maintain.

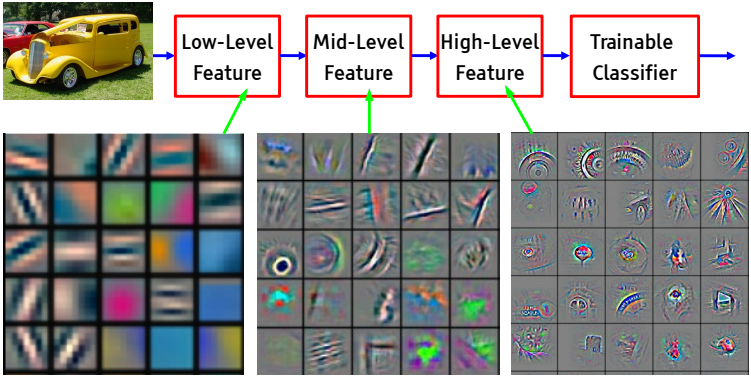
Representation learning

Data $\rightarrow$ Features (rerpresentation) $\rightarrow$ Classifier

- These problems can be overcome with **representation learning:** We use machine learning to discover not only the mapping from representation to output but also the representation itself.
- A representation learning algorithm can discover a good set of features much faster (in days instead of decades of efforts of an entire research community).
- Learned representations often result in much better performance compared to hand-designed representations.
- With learned representations, AI systems can rapidly adapt to new tasks, with minimal human intervention.

Deep learning = Learning hierarchical representations

- Deep learning does representation learning by introducing representations that are expressed in terms of other, simpler representations.



Feature visualization of convolutional net trained on ImageNet from (Zeiler and Fergus, 2013).

Deep learning = artificial neural networks

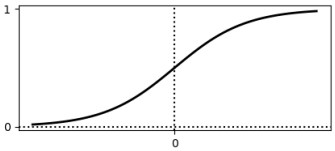
- Many ideas in deep learning models have been inspired by neuroscience:
  - The basic idea of having many computational units that become intelligent only via their interactions with each other is inspired by the brain.
  - The neocognitron (Fukushima, 1980) introduced a powerful model architecture for processing images that was inspired by the structure of the mammalian visual system and later became the basis for the modern convolutional networks.
- The name "deep learning" was invented to re-brand artificial neural networks which became unpopular in 2000s.
- Modern deep learning: A more general principle of learning multiple levels of composition, which can be applied in machine learning frameworks that are not necessarily neurally inspired.

Linear classifiers

- Consider a binary classification problem: Our training data consist of examples $(\mathbf{x}^{(1)}, y^{(1)}), ..., (\mathbf{x}^{(n)}, y^{(n)})$ with $y^{(i)} \in \{0, 1\}$.
- We use the training data to build a linear classifier

$$f(\mathbf{x}) = \sigma\left(\sum_{j=1}^{m} w_j x_j + b\right) = \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right)$$

where m is the number of features in $\mathbf{x}$.



- Logistic regression model: $\sigma(x) = \frac{1}{1+e^{-x}}$ is a logistic function.
- Using the logistic function guarantees that the output is between 0 and 1 and it can be seen as the probability that $\mathbf{x}$ belongs to one of the classes: $p(y = 1 \mid \mathbf{x}) = f(\mathbf{x})$.

Likelihood function for logistic regression model

- We can tune the model assuming the Bernoulli distribution for the label y:

$$p(y \mid \mathbf{x}, \mathbf{w}, b) = f(\mathbf{x})^y (1 - f(\mathbf{x}))^{1-y} \qquad \text{where } f(\mathbf{x}) = \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right)$$

- For n training examples, the likelihood function is

$$p(\text{data} \mid \mathbf{w}, b) = \prod_{i=1}^{n} p(y^{(i)} \mid \mathbf{x}^{(i)}, \mathbf{w}, b)$$

  where $p(y^{(i)} \mid \mathbf{x}^{(i)})$ is a function of the model parameters $\mathbf{w}$ and b.

- This gives the following log-likelihood function:

$$\mathcal{F}(\mathbf{w}, b) = \log p(\text{data} \mid \mathbf{w}, b) = \sum_{i=1}^{n} y^{(i)} \log f(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log(1 - f(\mathbf{x}^{(i)}))$$

Tuning parameters of logistic regression model

- We can tune the parameters $\mathbf{w}$ and b of the classifier by maximizing the log-likelihood function $\mathcal{F}(\mathbf{w}, b)$ or by minimizing the negative of that:

$$\mathcal{L}(\mathbf{w}, b) = -\sum_{i=1}^{n} y^{(i)} \log f(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log(1 - f(\mathbf{x}^{(i)}))$$

  This loss function if often called *binary cross entropy*.

- For example, we can use the method of gradient descent, which iteratively updates the parameter values:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{w}} \qquad b \leftarrow b - \alpha \frac{\partial \mathcal{L}}{\partial b}$$

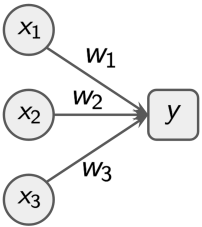
A historical note: First models of neurons

- First linear classifiers were proposed as a model of brain function by McCulloch and Pitts (1943).
- McCulloch-Pitts neuron is a linear binary classifier for binary inputs $x_j \in \{0, 1\}$

$$y = \phi\left(\sum_{j=1}^{m} w_j x_j + b\right)$$

where $\phi(\cdot)$ is a step function.

- To produce the correct output, the parameters w_j, b were set by a human operator.



$\phi(\cdot)$ is a step function

Perceptron (Rosenblatt, 1958)

- The perceptron was a machine (not a program) that also implemented a binary classifier

$$\hat{y}(\mathbf{x}; \mathbf{w}) = \text{sign}(\mathbf{w}^\top \mathbf{x})$$

- Rosenblatt proposed a training procedure using examples

$$(\mathbf{x}^{(i)}, y^{(i)}) \qquad \mathbf{x}^{(i)} \in \mathbb{R}^m, \quad y^{(i)} \in \{-1, +1\}$$

- The training procedure was inspired by Donald Hebb's rule: Weights between neurons whose activities are positively correlated are increased:
  - If $\mathbf{x}^{(i)}$ is misclassified, the weight vector is updated: $\mathbf{w} \leftarrow \mathbf{w} + y^{(i)}\mathbf{x}^{(i)}$
  - Training end when all data correctly classified
- Why can't we use gradients to find optimal $\mathbf{w}$?

Perceptron (Rosenblatt, 1958)

- The perceptron was a machine (not a program) that also implemented a binary classifier

$$\hat{y}(\mathbf{x}; \mathbf{w}) = \text{sign}(\mathbf{w}^\top \mathbf{x})$$

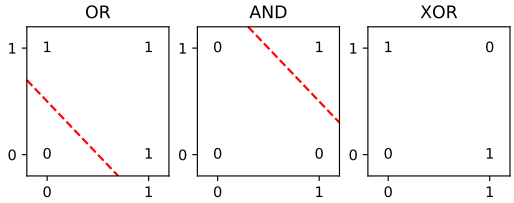
- Rosenblatt proposed a training procedure using examples

$$(\mathbf{x}^{(i)}, y^{(i)}) \qquad \mathbf{x}^{(i)} \in \mathbb{R}^m, \quad y^{(i)} \in \{-1, +1\}$$

- The training procedure was inspired by Donald Hebb's rule: Weights between neurons whose activities are positively correlated are increased:
  - If $\mathbf{x}^{(i)}$ is misclassified, the weight vector is updated: $\mathbf{w} \leftarrow \mathbf{w} + y^{(i)}\mathbf{x}^{(i)}$
  - Training end when all data correctly classified
- Why can't we use gradients to find optimal $\mathbf{w}$?
  - Because the sign() function is not differentiable.

Linear classifiers cannot solve complex classification problems

- The problem with perceptrons: Since they are linear classifiers, they can solve a very limited set of classification problems.
- They cannot separate linearly inseparable classes, for example, solve the XOR problem:

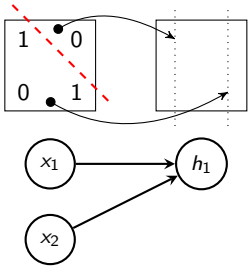


- This problem was emphasized in the influential book "Perceptrons" by Minsky and Papert (1969). They argued that more complex (nonlinear) problems have to be solved with multiple layers of perceptrons (what we now call multilayer neural nets).

Multilayer perceptrons

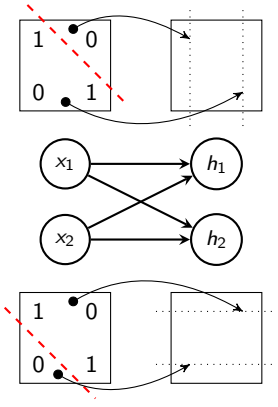
Solving the XOR problem

- The XOR problem can be solved with multiple layers of perceptrons (neurons).
- One neurons can linearly separate the input space as shown on the figure.



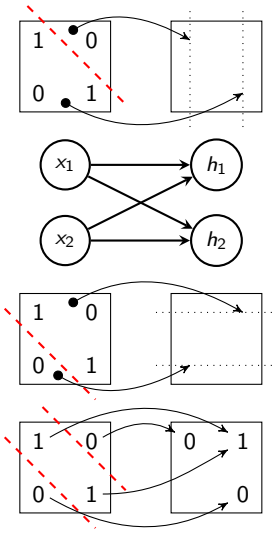
Solving the XOR problem

- The XOR problem can be solved with multiple layers of perceptrons (neurons).
- One neurons can linearly separate the input space as shown on the figure.
- We can add another neuron h_2 which can do another kind of separation of the input space.



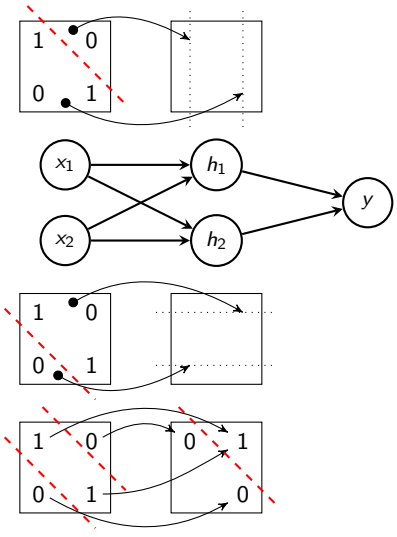
Solving the XOR problem

- The XOR problem can be solved with multiple layers of perceptrons (neurons).
- One neurons can linearly separate the input space as shown on the figure.
- We can add another neuron h_2 which can do another kind of separation of the input space.
- Now we mapped original two-dimensional data into a new two-dimensional space where linear separation is possible.



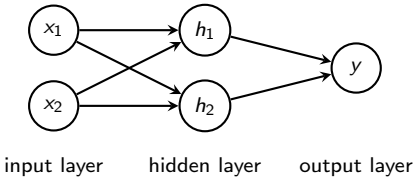
Solving the XOR problem

- The XOR problem can be solved with multiple layers of perceptrons (neurons).
- One neurons can linearly separate the input space as shown on the figure.
- We can add another neuron h_2 which can do another kind of separation of the input space.
- Now we mapped original two-dimensional data into a new two-dimensional space where linear separation is possible.
- Adding another neuron y on top of neurons h_1 and h_2 solves the classification problem.



Multilayer perceptrons

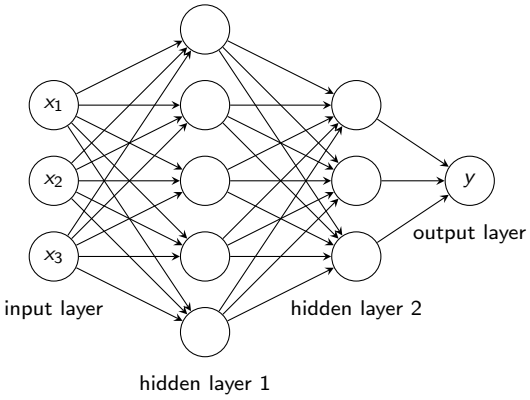
- Now we have a network with two layers of neurons: hidden layer h_1, h_2 and output layer y.
- A neural network with this architecture is called a *multilayer perceptron* (MLP).



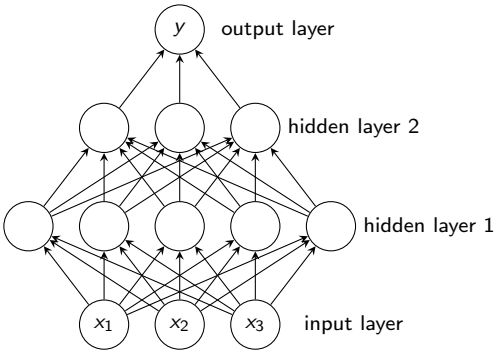
- An MLP can of course have more layers and many more neurons.
- Each neuron implements a function

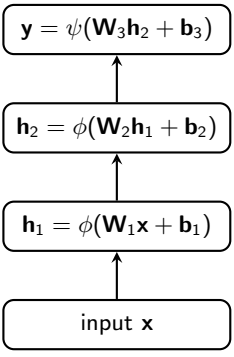
$$y = \phi\left(\sum_{j=1}^{m} w_j x_j + b\right) = \phi\left(\mathbf{w}^\top \mathbf{x} + b\right)$$

  which resembles a simple linear classifier that we considered before.
- The layers in an MLP are called *fully-connected* because each neuron is connected to each neuron in the previous layer.

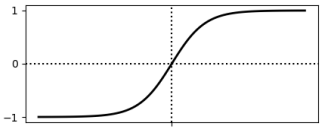


- A more compact style: A node in the graph corresponds to an entire layer.

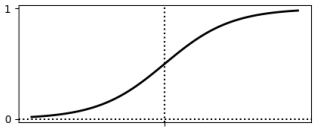




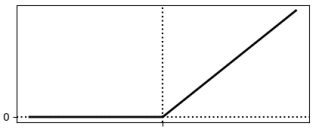
Activation functions



$\tanh(x)$



$\sigma(x) = 1/(1 + e^{-x})$

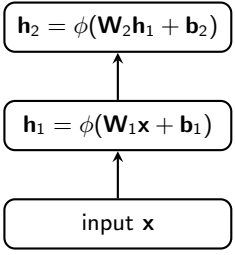


$\text{relu}(x) = \max(0, x)$

- Nonlinearities used after an affine transformation of inputs are often called *activation functions*.
- Nonlinearities used before 2010: $\tanh(x)$ and $\sigma(x) = 1/(1 + e^{-x})$.
- Since 2010, $\text{relu}(z) = \max(0, z)$ is very popular.

What if one does not use any nonlinearity?

- What if one does not use any nonlinearity?

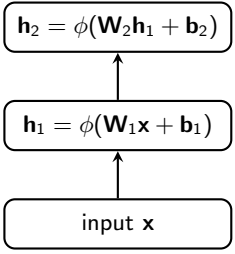


What if one does not use any nonlinearity?

- What if one does not use any nonlinearity?
- The identity activation function would lead to:

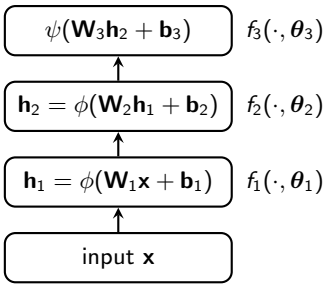
$$\begin{aligned}\mathbf{h}_2 &= \mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2 = \mathbf{W}_2(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 \\ &= (\mathbf{W}_2\mathbf{W}_1)\mathbf{x} + (\mathbf{W}_2\mathbf{b}_1 + \mathbf{b}_2) = \mathbf{W}'\mathbf{x} + \mathbf{b}'\end{aligned}$$

Thus, we get a linear model.



- Our neural network represents a function which is composed of several functions:

$$f(\mathbf{x}) = f_3(f_2(f_1(\mathbf{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2), \boldsymbol{\theta}_3)$$



- If we solve a binary classification problem, we can use the same loss function that we used before:

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{i=1}^{n} y^{(i)} \log f(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log(1 - f(\mathbf{x}^{(i)}))$$

- Again, we can tune the parameters $\boldsymbol{\theta}_k$ of the classifier by maximizing the log-likelihood, for example, using gradient descent:

$$\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_k - \alpha \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_k}.$$

A historical note on multilayer perceptrons

- The idea of using multilayer perceptrons for solving nonlinear classification problems existed already in the 1960s (Minsky and Papert, 1969). However, no one knew how to train multilayer perceptrons.
- Rosenblatt's learning algorithm

$$\text{if } \mathbf{x}^{(i)} \text{ is misclassified, the weight vector is updated: } \mathbf{w} \leftarrow \mathbf{w} + y^{(i)}\mathbf{x}^{(i)}$$

did not work for multiple layers. A training example $(\mathbf{x}^{(i)}, y^{(i)})$ only specifies the correct output for the final output layer, so there was no way to know how to adjust the weights of Perceptrons in layers before the last one.

A historical note on multilayer perceptrons

- How to train MLP networks was well understood only in the mid 80s after an influential paper by Rumelhart, Hinton and Williams (1986). Specifically, they showed how to compute the gradients $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$ wrt network parameters efficiently using the backpropagation algorithm.
  - Backpropagation is basically the application of the chain rule of differentiation to models with multiple layers. It was proposed by several researchers even earlier (Linnainmaa, 1970; Werbos, 1982) but became popular after the 1986 paper.
- The introduction of the backpropagation algorithm for training MLPs as well as other neural models (self-organizing maps, Hopfield networks and Boltzmann machines) caused a second wave of interest in neural networks in late 80s to mid-90s.

Deep learning era

A second winter for neural networks

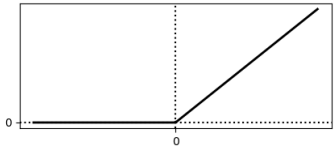
- By mid-90s, the interest in neural networks has dropped. New methods showed same or better performance in supervised learning tasks (such as support vector machines, random forests).
- There were no theoretical results that deep networks have better representational power.
  - Universal approximation theorem: A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of R^n (Cybenko, 1989).
- In early 2000s, there was a lot interest in probabilistic machine learning and kernel methods. The amounts of data were relatively small and those methods were favourable in the low-data regime.
- In early 2000s, researchers avoided to use terms "neural network" or "multilayer perceptron" in research proposals.

Deep belief networks

- Deeper networks (with more than two-three hidden layers) did not provide much better results until early 2000s when Hinton et al. (2006) demonstrated a better performance of a deep neural network over shallow networks on MNIST dataset.
- The model was called *deep belief networks*, in which the key elements were:
  - unsupervised layer-wise pretraining using restricted Boltzmann machines
  - supervised fine-tuning
- At that time, the term "deep learning" was invented to re-brand neural networks.

Deep learning starts

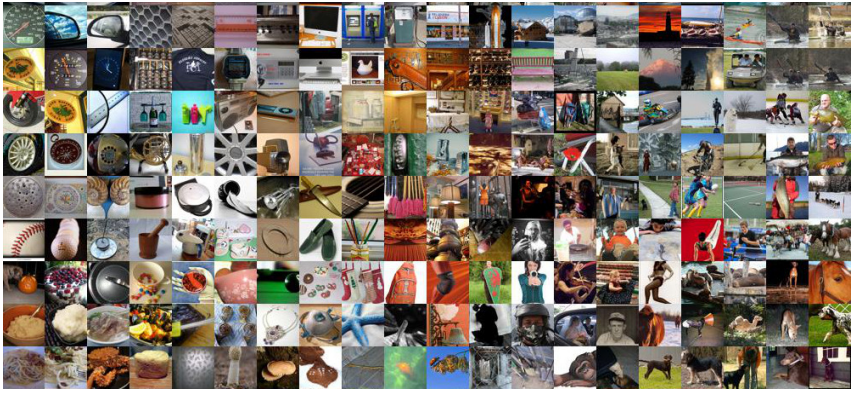
- The good results of deep belief networks have attracted many researchers.
- Schmidhuber's group (Ciresan et al., 2010) showed that pre-training in deep neural networks is not necessary, one just needs more time to train (they used GPUs).
- Several research groups around 2009-2010 showed that ReLU activation function works better.



- Hinton et al. (2012) proposed dropout, a method to reduce overfitting.

Imagenet: The beginning of the deep learning era

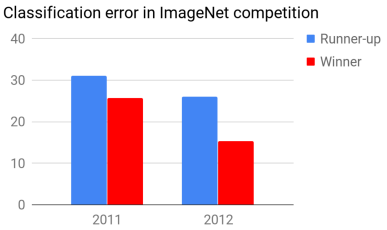
- Imagenet: Yearly competition in computer vision

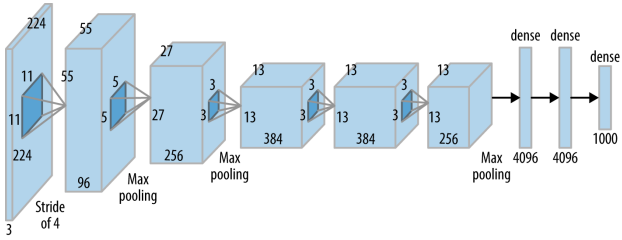


- Before 2012: The winning solutions used hand-crafted features.

AlexNet: The beginning of the deep learning era

- Krizhevsky, Sutskever and Hinton (2012) won the Imagenet competition by a large margin using a deep convolutional neural network.





Deep learning improved results almost in every domain

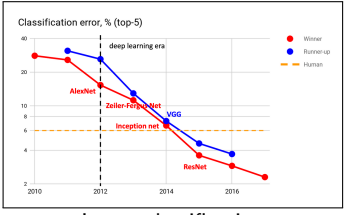
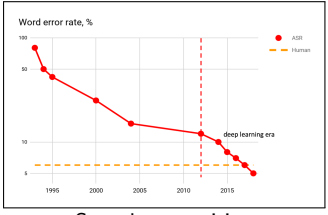
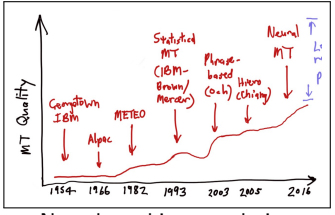


Image classification
(Krizhevsky et al., 2012)



Speech recognition
(Graves and Jaitly, 2014)



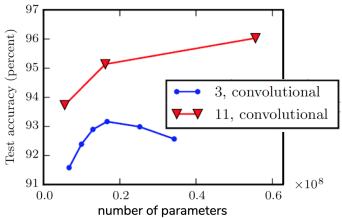
Neural machine translation
(Cho et al., 2014)

What did deep learning start only in 2010-2012?

- Many components of deep learning have been invented much earlier:
  - 1958: Perceptrons
  - 1986: Backpropagation
  - 1989: Convolutional networks
- Why did it start only in 2010-2012?
- Geoffrey Hinton gave four reasons for that:
  - Our labeled datasets were thousands of times too small.
  - Our computers were millions of times too slow.
  - We initialized the weights in a stupid way.
  - We used the wrong type of non-linearity.

Deeper models tend to perform better

- Theory: Although feed-forward networks with a single hidden layer are universal approximators, the width of such networks has to be exponentially large.
- Practice: increasing the number of parameters in layers without increasing their depth is not nearly as effective at increasing test set performance:
  - Shallow models overfit at around 20 million parameters while deep ones can benefit from having over 60 million
- Deep models express a belief that the learned function should consist of many simpler functions composed together, which turns out to be a good assumption.



Goodfellow et al (2014). Multi-digit number recognition from Street View imagery using deep convolutional neural networks

Representation learning vs feature engineering

Traditional way: Data $\rightarrow$ Feature engineering $\rightarrow$ Machine learning (e.g. classification)

Deep learning way: Data $\rightarrow$ End-to-end learning

- Deep learning has become the state of the art in many applications. Do we need feature engineering at all?

Representation learning vs feature engineering

Traditional way: Data $\rightarrow$ Feature engineering $\rightarrow$ Machine learning (e.g. classification)

Deep learning way: Data $\rightarrow$ End-to-end learning

- Deep learning has become the state of the art in many applications. Do we need feature engineering at all?
- Deep learning needs a lot of data. In some applications, the amount of available data can be limited. In such cases, feature engineering can be useful.

Finland has been strong in
neural networks research

Finland has been strong in neural networks research

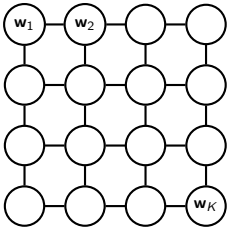
Self-organizing maps
(Kohonen, 1981)

Neuron principal component analysis (Oja, 1982)
Bottleneck autoencoder (Oja, 1991)

- Hyvärinen and Oja (1997): Fast algorithms for independent component analysis (FastICA)
- Valpola and Honkela (2000): Predecessor model of variational autoencoders
- Kyunghyun Cho (GRU, NMT) was a Macadamia student, did his PhD in deep learning in Aalto.

Self-organizing map (SOM) (Kohonen, 1981)

- Unsupervised learning method that can be used, for example, for data visualization.
- Data samples $\mathbf{x}^{(i)}$ are mapped to a grid of neurons $\mathbf{w}_k$ arranged in a 2D square or hexagonal grid.



SOM square grid

- Training procedure:
  - Take a training sample $\mathbf{x}^{(i)}$ and select the neuron whose weight vector $\mathbf{w}_k$ has the shortest Euclidean distance to $\mathbf{x}^{(i)}$.
  - The weight vectors of the winning neuron and the neurons in its neighborhood are updated:

$$\mathbf{w}_j \leftarrow \mathbf{w}_j + \eta(j,k)\left(\mathbf{x}^{(i)} - \mathbf{w}_j\right)$$

  where $\eta(j,k)$ is the neighborhood function which depends on the distance on the grid.

Self-organizing map (SOM)

- SOM can be used as a data visualization tool.
- Data: $\mathbf{x}^{(i)}$ is a collection of votes by one member of Congress, each vote is yes/no/abstain.

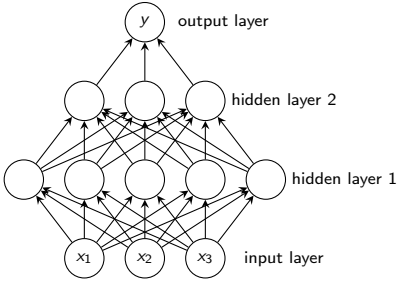
Home assignments

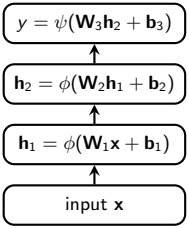
Assignment `01_test`

- The deadline is this Friday 23:00.

Assignment 02_mlp

1. Implement and train a multilayer perceptron (MLP) network in PyTorch.





2. Implement the backpropagation algorithm and train a multilayer perceptron (MLP) network in numpy (we will study backpropagation on the next lecture on Thursday).

Recommended reading

- Sections 1, 6.1–6.4 of the deep learning book.
- A. Kurenkov. A 'Brief' History of Neural Nets and Deep Learning.