Introduction to Machine Learning
NPFL 054

http://ufal.mff.cuni.cz/course/npfl054

Barbora Hladká
hladka@ufal.mff.cuni.cz

Martin Holub
holub@ufal.mff.cuni.cz

Charles University,
Faculty of Mathematics and Physics,
Institute of Formal and Applied Linguistics

Neural Networks fundamentals

. . . and key ideas of Deep Learning

- **Foreword on Neural Networks and Deep Learning**
  - Remarks on the relation of Machine Learning and Data Science
  - Motivations for deep architectures
  - Remarks on historical development and current visions
- **Perceptron**
  - Prerequisites for Neural Networks
  - Single unit perceptron learner, Single Layer Perceptron (SLP)
  - Basic NN topologies – one layer, multi-layered, feedforward, recurrent
- **Multi-Layer Perceptron (MLP)**
  - Nonlinear activation functions, output functions, loss functions
  - Back-propagation algorithm, Stochastic Gradient Descent
- **Magic power of deep architectures – key inventions of last decade**
  - Last decade: A Deep Learning breakthrough
    . . . when Deep Learning became convincing in practise
  - Example applications of Deep Learning that work
- **References**

Foreword on “Deep Learning tsunami”

What are Deep architectures / Deep learning methods?

- “Deep architectures are composed of multiple levels of non-linear operations, such as in neural nets with many hidden layers . . . ”
- “Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. Automatically learning features at multiple levels of abstraction allows a system to learn a complex functions mapping the input to the output directly from data.”

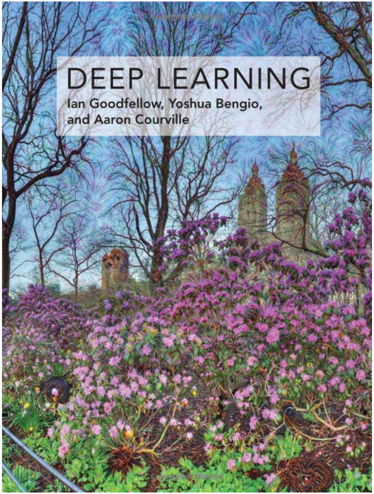
  — Bengio, 2009

In the same paper, Bengio claims that “insuficient depth can be detrimental for learning”: “an architecture with insuficient depth can require many more computational elements, potencially exponentially more (with respect to input size), than architectures whose depth is matched to the task”.

Deep Learning tsunami?

In 2015 a leading researcher in the field of Natural Language Processing (NLP), Chris Manning, reports that “Deep Learning tsunami” hit the major NLP conferences with its full force.

Deep Machine Learning – an excellent textbook!



Classical vs. deep Machine Learning

Cited from: *Deep Learning*, MIT Press, 2016.

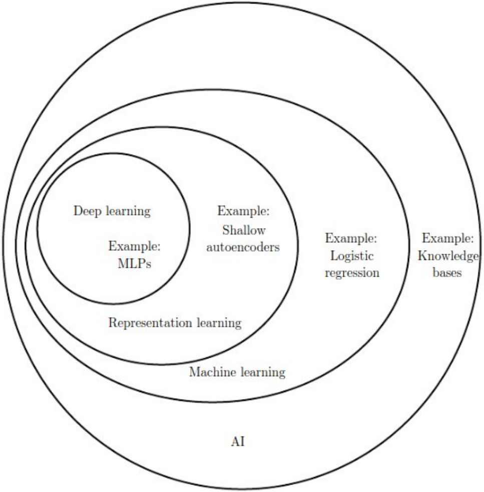
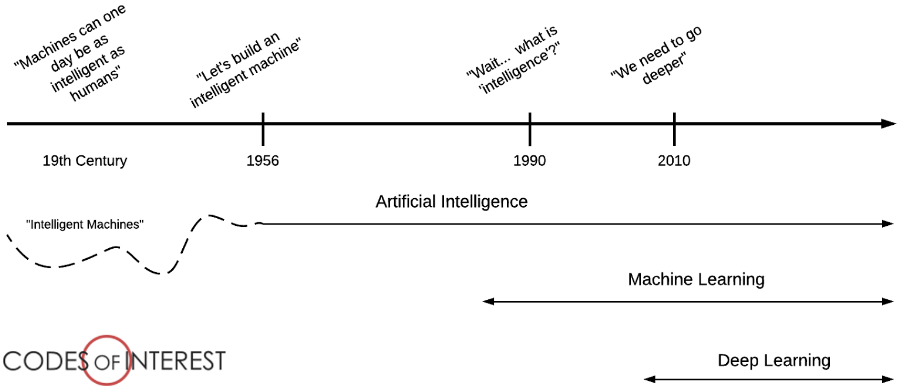


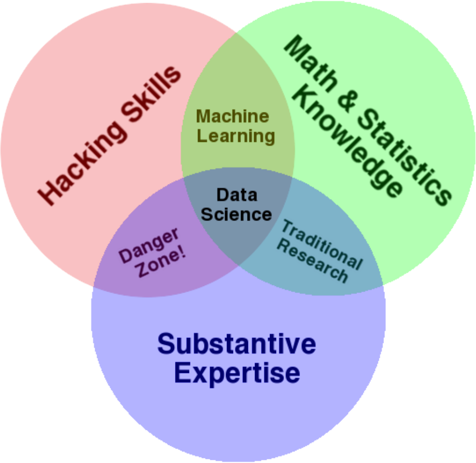
Figure 1.4: A Venn diagram showing how deep learning is a kind of representation learning, which is in turn a kind of machine learning, which is used for many but not all approaches to AI. Each section of the Venn diagram includes an example of an AI technology.

Deep learning – history

Cited from: `www.codesofinterest.com/p/what-is-deep-learning.html`



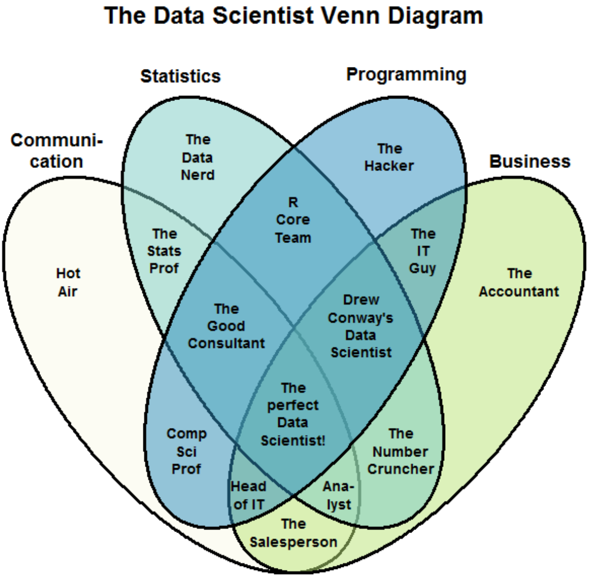
Machine Learning in the context of Data Science



How to read the Data Science Venn Diagram

For more comments see `http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram`

The perfect Data Scientist



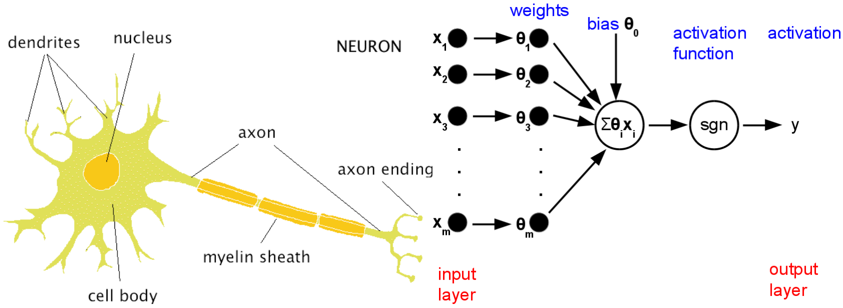
Prerequisites for Neural Networks

You should already know

- Linear and Logistic Regression
- Gradient Descent algorithm
- Maximum Likelihood Estimation

Single-layer perceptron (SLP)

biological inspiration	**machine learning**
neuron	SLP algorithm
other neurons	$\mathbf{x} = \langle x_1, \dots, x_m \rangle$
connection weights	feature weights $\Theta_1, \dots, \Theta_m$
amount of neuron activation	$\sum_{i=1}^{m} x_i \Theta_i$
"firing" neuron	output positive classification
"not firing" neuron	output negative classification



SLP

- is **on-line learning**
  - Look at one example, process it and go to the next example
- is **error-driven learning**
  - One example at a time
  - Make prediction, compare it with true prediction
  - Update Θ if different

Binary classification with SLP

Neuron

- **Input**: instance $\mathbf{x}$
- **Incoming connections**: feature weights $\Theta_1, \dots, \Theta_m$, $\mathbf{\Theta} =< \Theta_1, \dots, \Theta_m >$
- **Action**: compute activation $a = \mathbf{\Theta}^{\mathrm{T}}\mathbf{x}$
- **Output**: if $a > 0$ output $+1$ otherwise -1

Prefer **non-zero threshold $\mathbf{\Theta}^{\mathrm{T}}\mathbf{x} > \Delta$**

- Introduce a **bias** term Θ_0 into the neuron.
- Then $a = \Theta_0 + \mathbf{\Theta}^T\mathbf{x}$
- **Output**: if $a > 0$ output $+1$ otherwise -1

Binary classification with SLP
Training algorithm

$Data = \{\langle \mathbf{x}, y \rangle : \mathbf{x} = \langle x_1, \ldots, x_m \rangle, y \in \{-1, +1\}\}$

1. Initialize weights $\Theta_i \leftarrow 0$ for all $i = 1, \ldots, m$
2. Initialize bias $\Theta_0 \leftarrow 0$
3. for $iter = 1, \ldots, MaxIter$ do
4. for all $\langle \mathbf{x}, y \rangle \in Data$ do
5. $a \leftarrow \Theta_0 + \boldsymbol{\Theta}^T \mathbf{x}$ // compute activation for x
6. if $ya \leq 0$ then // update weights and bias
7. $\Theta_i \leftarrow \Theta_i + yx_i$ for all $i = 1, \ldots, m$
8. $\Theta_0 \leftarrow \Theta_0 + y$
9. end if
10. end for
11. end for
12. Return $\Theta_0^\star, \boldsymbol{\Theta}^\star$

Binary classification with SLP
Test

1. $a \leftarrow \Theta_0^\star + \boldsymbol{\Theta}^{\star T}\mathbf{x}$ `// compute activation for x`
2. `return sgn(`a`)`

Error driven learning – updating Θ parameters

If we can see the given example in the future, we should do a better job.

For illustration

- Assume a positive example $\langle \mathbf{x}, +1 \rangle$, and current Θ_0 and $\mathbf{\Theta}$.
- Do prediction and $y(\Theta_0 + \mathbf{\Theta}^T \mathbf{x}) < 0$, i.e. misclassification
- Update Θ_0 and $\mathbf{\Theta}$
    - $\Theta_0' = \Theta_0 + 1$
    - $\Theta_i' = \Theta_i + 1 * x_i,\ i = 1, \dots, m$
- Process the next example which is by chance the same example $\mathbf{x}$
- Compute

$$\begin{aligned} a' &= \Theta_0' + (\mathbf{\Theta'})^T \mathbf{x} = \Theta_0 + 1 + (\mathbf{\Theta} + \mathbf{x})^T \mathbf{x} = \\ &= \Theta_0 + 1 + \mathbf{\Theta}^T \mathbf{x} + \mathbf{x}^T \mathbf{x} > \Theta_0 + \mathbf{\Theta}^T \mathbf{x} = a \end{aligned}$$

- $\mathbf{x}$ is a positive example so we have moved the activation in the proper direction

Perceptron learning – the underlying idea

The perceptron learning algorithm tries to find a separating hyperplane by minimizing the distance of misclassified points to the decision boundary.

- If a response y_i is misclassified, then $y_i(\mathbf{\Theta}^T \mathbf{x_i} + \Theta_0) < 0$.
- Therefore the learning goal is to minimize the sum of distances

$$D(\mathbf{\Theta}, \Theta_0) = -\sum_{i \in \mathcal{M}} y_i(\mathbf{\Theta}^T \mathbf{x_i} + \Theta_0) < 0,$$

  where $\mathcal{M}$ indexes the set of misclassified points.

In fact, the algorithm uses the gradient descent method to minimize this piecewise linear criterion. The gradient (assuming $\mathcal{M}$ is fixed) is given by

$$\frac{\delta D(\mathbf{\Theta}, \Theta_0)}{\delta \mathbf{\Theta}} = -\sum_{i \in \mathcal{M}} y_i \mathbf{x_i}, \quad \text{and} \quad \frac{\delta D(\mathbf{\Theta}, \Theta_0)}{\delta \Theta_0} = -\sum_{i \in \mathcal{M}} y_i,$$

which is the rationale for parameter updates (the learning rate α is taken to be 1):

$$\mathbf{\Theta} \longleftarrow \mathbf{\Theta} + \alpha y_i \mathbf{x_i}, \quad \text{and} \quad \Theta_0 \longleftarrow \Theta_0 + \alpha y_i.$$

Geometric interpretation of SLP

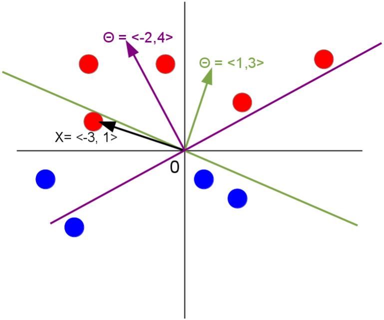
A hyperplane of an m-dimensional space is a flat subset with dimension $m - 1$. Any hyperplane can be written as the set of points $\mathbf{x}$ satisfying

$$\Theta_0 + \mathbf{\Theta}^T\mathbf{x} = 0, \text{ where } \mathbf{\Theta} = \begin{pmatrix} \Theta_1 \\ \vdots \\ \Theta_m \end{pmatrix}, \mathbf{x} = \langle x_1, \ldots, x_m \rangle$$

Geometric interpretation of SLP

Assume $\Theta_0 = 0$

- $\mathbf{\Theta}$ points in the direction of the positive examples and away from the negative examples.



- Having $\mathbf{\Theta}$ normalized, $\mathbf{\Theta}^T\mathbf{x}$ is the length of projection of $\mathbf{x}$ onto $\mathbf{\Theta}$, i.e. the activation of $\mathbf{x}$ with no bias.

Geometric interpretation of SLP

Assume $\Theta_0 \neq 0$

- After the projection is computed, Θ_0 is added to get the overall activation.
- Then $\mathbf{\Theta}^T\mathbf{x} + \Theta_0 > 0$?
  - If $\Theta_0 < 0$, Θ_0 shifts the hyperplane away from $\mathbf{\Theta}$.
  - If $\Theta_0 > 0$, Θ_0 shifts the hyperplane towards $\mathbf{\Theta}$.

Properties of SLP algorithm

Learning parameter `MaxIter`

- many passes $\rightarrow$ overfitting
- only one pass $\rightarrow$ underfitting

Convergence

- Does the SLP algorithm converge?
  - If the training data IS linearly separable, the SLP algorithm yields a hyperplane that classifies all the training examples correctly.
  - If the training data IS NOT linearly saparable, the SLP algorithm could never possibly classify each example correctly.
- After how many updates the algorithm converges?

Properties of SLP algorithm

Recall the notion of margin of hyperplane

— Assume a hyperplane g: $\Theta_0 + \boldsymbol{\Theta}^{\mathrm{T}}\mathbf{x} = 0$

- The **geometric margin** of $\langle \mathbf{x}, y \rangle$ w.r.t. a hyperplane g is

$$\rho_g(\mathbf{x}, y) = y(\Theta_0 + \boldsymbol{\Theta}^T\mathbf{x})/||\Theta||$$

- The **margin** of *Data* w.r.t. a hyperplane g is

$$\rho_g(Data) = \mathrm{argmin}_{\langle \mathbf{x}, y \rangle \in Data} \rho_g(\mathbf{x}, y)$$

- **Define optimal hyperplane g***

$$g^\star = \mathrm{argmax}_g \rho_g(Data), g^\star : \Theta_0^\star + {\boldsymbol{\Theta}^\star}^{\mathrm{T}}\mathbf{x} = 0$$

Let $\gamma = \rho_{g^\star}(Data)$

Perceptron Convergence Theorem

Suppose the perceptron algorithm is run on a linearly separable data set *Data* with margin $\gamma > 0$. Assume that $||\mathbf{x}|| \leq 1$ for all examples in *Data*. Then the algorithm will converge after at most $\frac{1}{\gamma^2}$ updates.

Proof: The perceptron algorithm is trying to find $\mathbf{\Theta}$ that points roughly in the same direction as $\mathbf{\Theta}^\star$. We are interested in the angle α between $\mathbf{\Theta}$ and $\mathbf{\Theta}^\star$. Every time the algorithm makes an update, α changes. Thus we approve that α decreases. We will show that

1. $\mathbf{\Theta}^T\mathbf{\Theta}^\star$ increases a lot
2. $||\mathbf{\Theta}||$ does not increase much

Perceptron Convergence Theorem

$\boldsymbol{\Theta}^0$ is the initial weight vector, $\boldsymbol{\Theta}^k$ is the weight vector after k updates.

1. We will show that $\boldsymbol{\Theta}^\star\boldsymbol{\Theta}^k$ grows as a function of k:

$$\boldsymbol{\Theta}^\star\boldsymbol{\Theta}^k \overset{\text{definition of } \boldsymbol{\Theta}^k}{=} \boldsymbol{\Theta}^\star(\boldsymbol{\Theta}^{k-1} + y\mathbf{x}) \overset{\text{vector algebra}}{=}$$
$$= \boldsymbol{\Theta}^\star\boldsymbol{\Theta}^{k-1} + y\boldsymbol{\Theta}^\star\mathbf{x} \overset{\boldsymbol{\Theta}^\star \text{has margin} \gamma}{\geq} \boldsymbol{\Theta}^\star\boldsymbol{\Theta}^{k-1} + \gamma$$

Therefore $\boldsymbol{\Theta}^\star\boldsymbol{\Theta}^k \geq k\gamma$

Perceptron Convergence Theorem

2. We update $\boldsymbol{\Theta}^k$ because $y(\boldsymbol{\Theta}^{k-1})^T\mathbf{x} < 0$

$$||\boldsymbol{\Theta}^k||^2 = ||\boldsymbol{\Theta}^{k-1} + y\mathbf{x}||^2 \overset{\text{quadratic rule of vectors}}{=}$$
$$||\boldsymbol{\Theta}^{k-1}||^2 + y^2||\mathbf{x}||^2 + 2y\boldsymbol{\Theta}^{k-1}\mathbf{x} \overset{\text{assumption on}||\mathbf{x}||\text{ and } a<0}{\leq} ||\boldsymbol{\Theta}^{k-1}||^2 + 1 + 0$$

Therefore $||\boldsymbol{\Theta}^k||^2 \leq k$

Perceptron Convergence Theorem

Putting 1. and 2. together, we can write

$$\sqrt{k} \overset{2.}{\geq} ||\mathbf{\Theta}^k|| \overset{\mathbf{\Theta}^\star \text{ is a unit vector}}{\geq} (\mathbf{\Theta}^\star)^T \mathbf{\Theta}^k \overset{1.}{\geq} k\gamma \Rightarrow k \leq 1/\gamma^2$$

Perceptron Convergence Theorem

- The proof says that if the perceptron gets linearly separable data with γ, then it will converge to a solution that separates the data.
- The proof does not speak about the solution, other than the fact that it separates the data. The proof makes use of the maximum margin hyperplane. But the perceptron is not guaranteed to find m.m. hyperplane.

Mutliclass classification with SLP Training

$Y = \{1, \ldots, k\}$
There will be a weight vector for each class $\mathbf{\Theta}^1, \ldots, \mathbf{\Theta}^k$

```
1.  Initialize weights  Θ^k ← ⟨0, ..., 0⟩          for all  i = 1, ..., k
2.      for  iter  =  1, ..., MaxIter  do
3.          for all  ⟨x, y⟩ ∈ Data  do
4.              compute  Θ^{i^T} x                 for all  i = 1, ..., k
5.              ŷ  =  argmax_i  Θ^{i^T} x
6.              if y and ŷ are different then
7.                  Θ^y  ←  Θ^y  −  x
8.                  Θ^ŷ  ←  Θ^ŷ  +  x
9.              end if
10.         end for
11.     end for
12. Return  Θ^{1*}, ..., Θ^{k*}
```

Mutliclass classification with SLP
Test

1. `return` $argmax_i \ (\mathbf{\Theta}^{i^\star})^T \mathbf{x}$

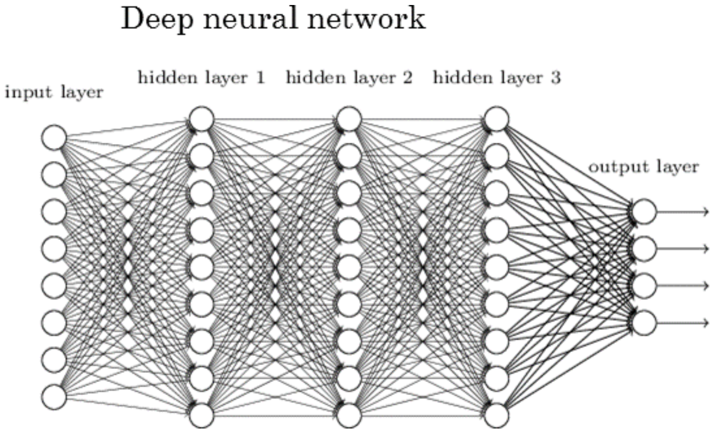
Perceptron – different activation/output functions

Generally, output of each network layer is produced by an activation function f:

$$\mathbf{h} = f(\mathbf{W}^T\mathbf{x} + \mathbf{b})$$

- identity $\sim$ linear regression
  - traditional MSE loss for regression
- **sigmoid family**
  - logistic (sigmoid) $\sim$ logistic regression
  - tanh, "hard tanh"
- **ReLU**
  - often appears to be better than a sigmoid
- **softmax**
  - most often used as the output of a classifier
  - to model probability distribution over k classes
  - used in connection with negative log-likelihood loss

Deep feedforward architecture



Fully connected layers have their own

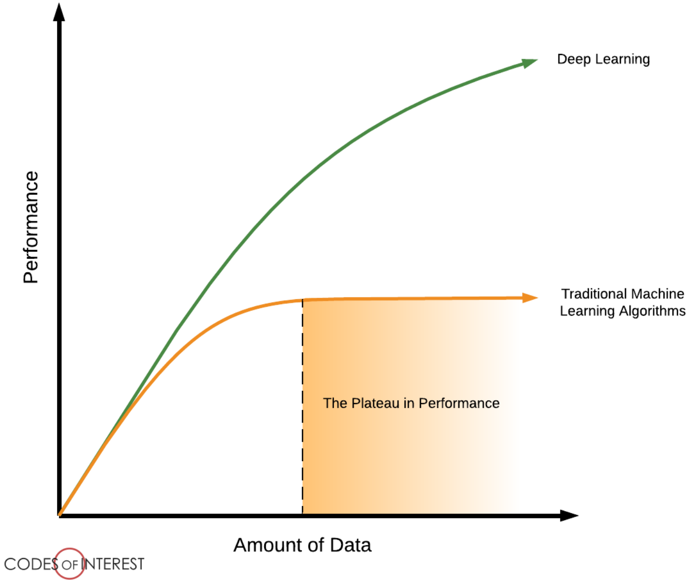
- sets of parameters (weights and biases)
- outputs (activation values)

Universal approximation function theorem

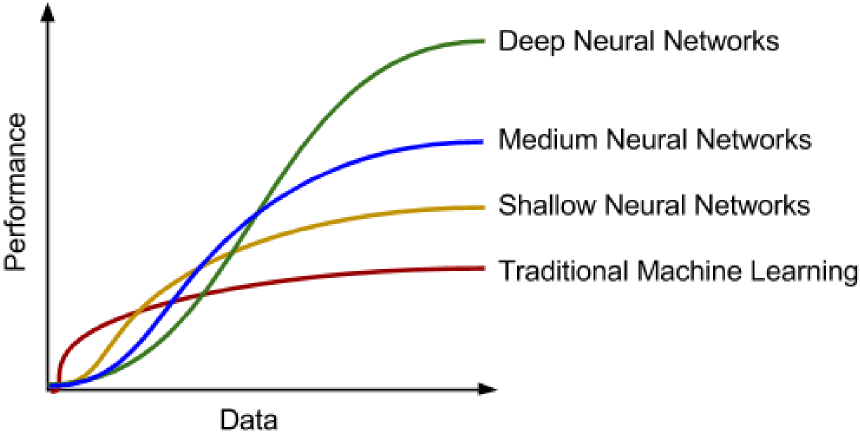
The *universal approximation theorem* (Hornik et al., 1989; Cybenko, 1989) states that a feedforward network with a linear output layer and at least one hidden layer with any "squashing" activation function (such as the logistic sigmoid activation function) can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units.
[...]
The universal approximation theorem means that regardless of what function we are trying to learn, we know that a large MLP will be able to **represent** this function. However, we are not guaranteed that the training algorithm will be able to **learn** that function.

— *The Deep Learning book*

ML performance – traditional vs. deep



The deeper the better?



Back-propagation learning

back-propagation = gradient descent + chain rule

Fitting a neural network

- is generally a very computationally intensive procedure
- key algorithm: *"back-propagating erors"* (mid 1980s)
- back-propagation algorithm iterates through many cycles of two processes: forward phase and backward phase
- each iteration during which all training examples are processed = *"epoch"*

Forward phase – neurons are activated in sequence from the input layer to the output layer using the existing weights and output signal is produced

Backward phase – the value of cost function is computed by comparing the output with the true value, then gradient descent method is applied and derivatives are propagated from neurons in the output layer backwards in the network successively to the input layer

Historical overview – last decade

It is a matter of fact that during last decade
Deep Learning indisputably proved its "magic" power . . .

- **Why it "suddenly" works? . . .**
  . . . while earlier it didn't?

- What were
  the milestones,
  the breakthrough ideas,
  and the key inventions
  that effectively made the drammatical development possible
  and changed the machine learning world forever?

If you want to learn more — take the opportunity
$\Longrightarrow$ Attend the great Milan Straka's course on *Deep Learning*!

Key inventions in last decade

- ReLU activation
  and its modifications like LReLU, PReLU, SReLU, ELU, ...
- softmax output + negative log likelihood loss
- better regularization techniques
  – e.g. "dropout"
- Gradient Descent with adaptive gradient
  – e.g. SGD with momentum, AdaGrad, RMSProp, Adam

Key inventions in last decade (cont.)

New NN architectures

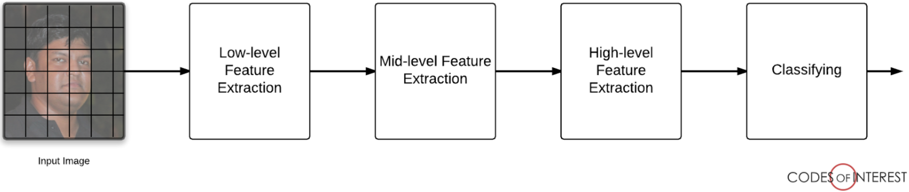
- convolution NN
  - AlexNet, VGG, GoogLeNet (also called Inception), ResNet, ...
- recurrent NN
  - LSTM, GRU
- residual connections
  - in CNN: ResNet
  - in fully connected layers: highway networks

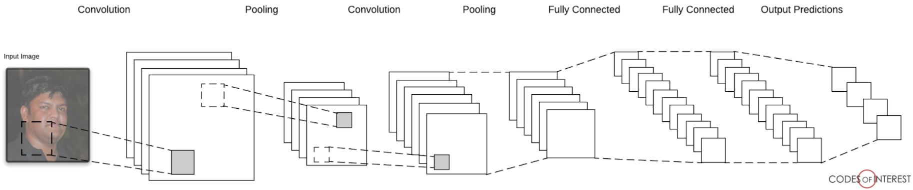
Distributed representations

– e.g. so called "embeddings"

Convolutional neural network – example architecture

Cited from: `www.codesofinterest.com/p/what-is-deep-learning.html`

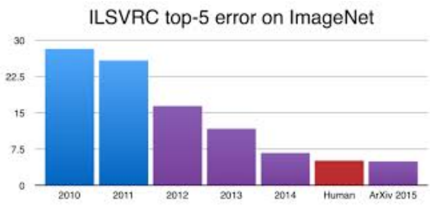


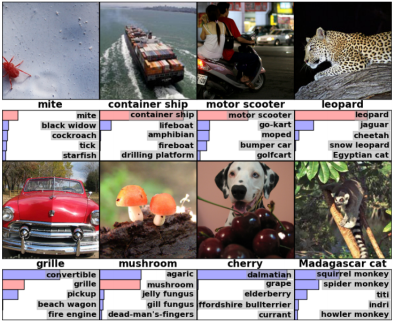


Examples of successful applications
Image recognition

ILSVCR contest, domination of CNN since 2012

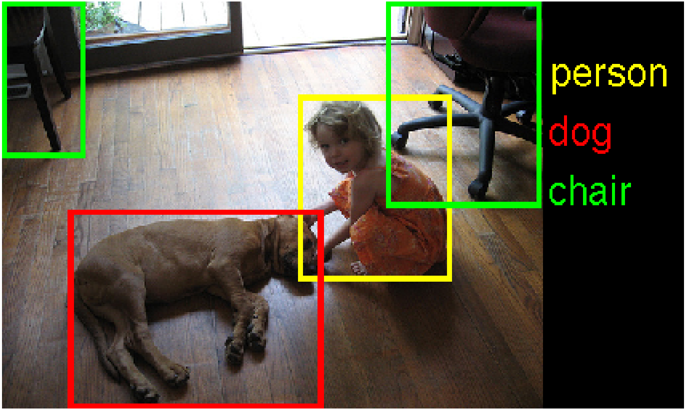
Second image from ImageNet Classification with Deep Convolutional Neural Networks by Alex Krizhevsky et al.





Examples of successful applications
Object detection

http://image-net.org/challenges/LSVRC/2014/



Examples of successful applications
Image segmentation

http://mscoco.org/dataset/#detections-challenge2016

A few other examples of successful applications

- Speech recognition, speech synthesis
- Handwriting recognition
- Neural machine translation
- Multimodal tasks: visual question answering, image labelling, image description translation
- Video game playing
- Maze navigation, Precise robot control

- AlphaGo
  "Mastering the game of Go with deep neural networks and tree search"
  – by Silver D., et al. (*Nature*, 529 (7587): 484–489. 2016)

References (not only) for beginners

- Bengio, Y: *Learning Deep Architectures for AI.* 2009.
  - available online
- Manning, C.: *Computational Linguistics and Deep Learning.* 2015.
  - available online
- Goodfellow I., Bengio Y., Courville A.: *Deep Learning Book*. 2016.
  - `http://www.deeplearningbook.org`
- Hal Daume III.: *A Course on Machine Learning.* Second printing, 2017.
  - An online set of introductory materials on Machine Learning
  - `http://ciml.info`