

Introduction to Machine Learning and Deep Learning with Python

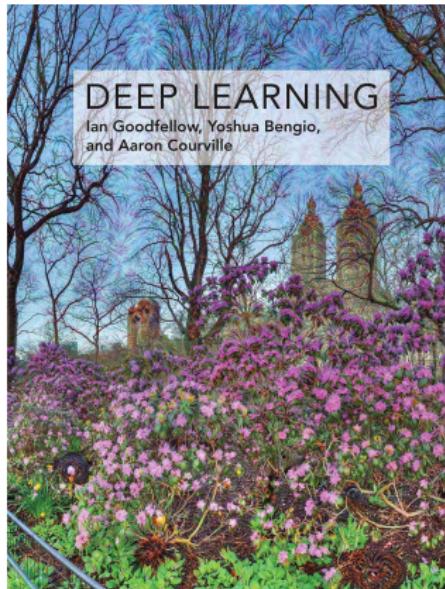
Day 3

Tabea Rebafka



CNRS Formation Entreprise
28 – 30 June, 2023, Paris

Day 3: Deep Learning



Outline of Day 3

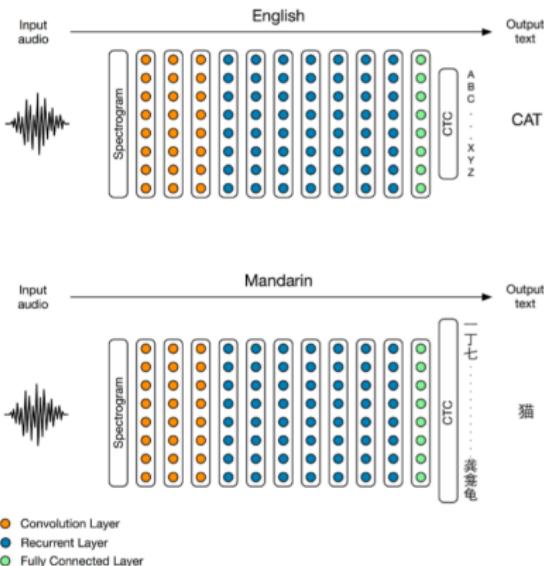
- 1 Deep learning: an introduction
- 2 Neural networks: basic principles and architecture
- 3 Tools for deep neural networks
 - Forward propagation
 - Backpropagation
 - Optimization with SGD and mini batches
 - Early stopping
 - Dropout
- 4 Convolutional neural networks
 - Convolutional layer
 - Pooling

- 1 Deep learning: an introduction
- 2 Neural networks: basic principles and architecture
- 3 Tools for deep neural networks
 - Forward propagation
 - Backpropagation
 - Optimization with SGD and mini batches
 - Early stopping
 - Dropout
- 4 Convolutional neural networks
 - Convolutional layer
 - Pooling

Where is deep learning today? I



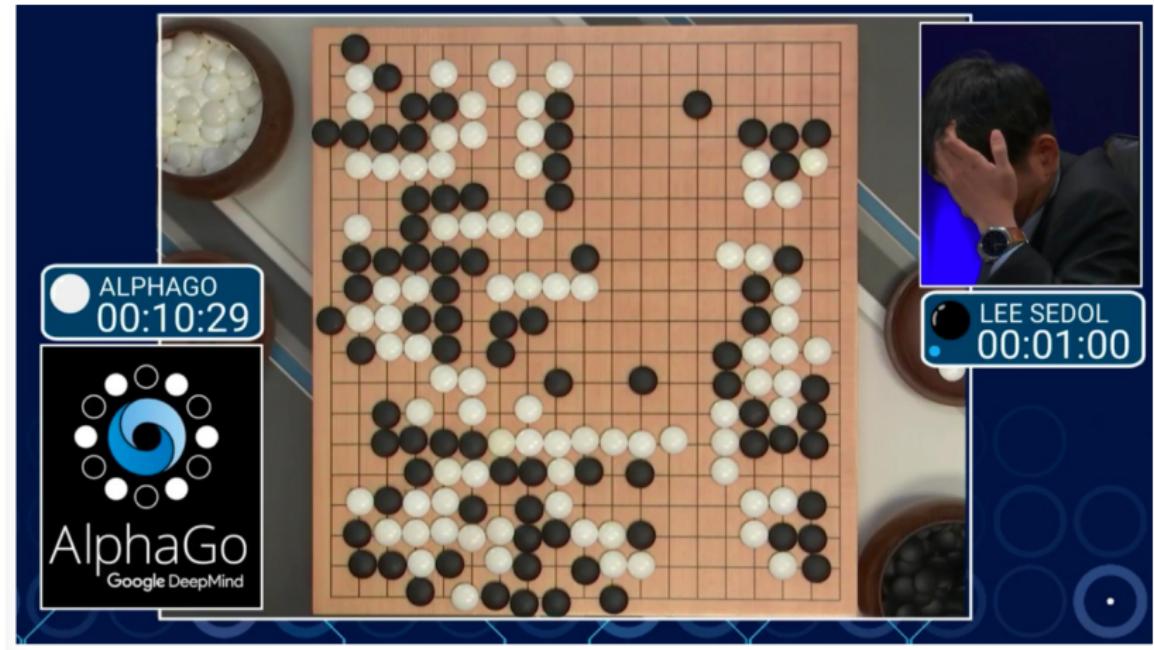
hello



[Baidu 2014]

Speech recognition

Where is deep learning today? II



Beat humans at Go (deep learning and reinforcement learning)

JANUARY 25, 2017

Deep learning algorithm does as well as dermatologists in identifying skin cancer

In hopes of creating better access to medical care, Stanford researchers have trained an algorithm to diagnose skin cancer.



BY TAYLOR KUBOTA



It's scary enough making a doctor's appointment to see if a strange mole could be cancerous. Imagine, then, that you were in that situation while also living far away from the nearest doctor, unable to take time off work and unsure you had the money to cover the cost of the visit. In a scenario like this, an option to receive a diagnosis through your smartphone could be lifesaving.

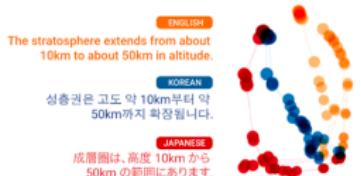


Universal access to health care was on the minds of computer scientists at Stanford when they set out to create an artificially intelligent diagnosis algorithm for skin cancer. They made a database of nearly 130,000 skin disease images and trained their algorithm to visually diagnose potential cancer. From the very first test, it performed with inspiring accuracy.

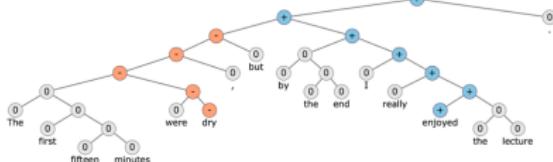


Analysis of medical pictures

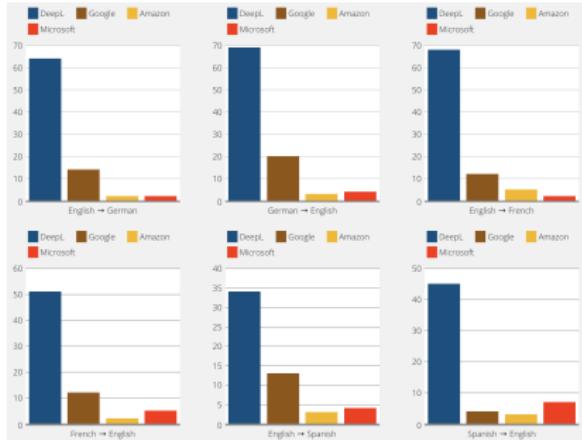
Where is deep learning today? IV



[Google Translate System - 2016]



[Socher 2015]



[DeepL 2020]

Automatic translation

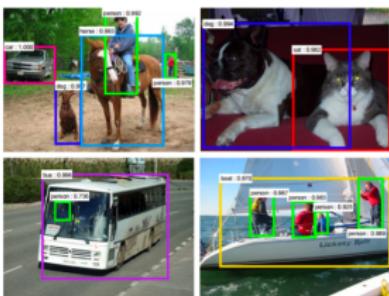
Where is deep learning today? V



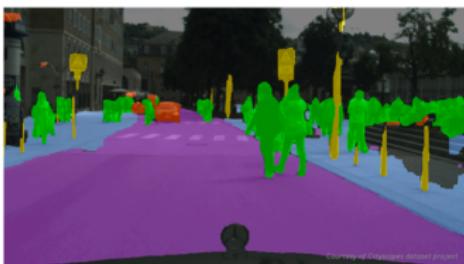
[Krizhevsky 2012]



[Ciresan et al. 2013]



[Faster R-CNN - Ren 2015]



[NVIDIA dev blog]

Image classification, entity naming, self-driving cars

Where is deep learning today? VI



It's all fake!

Where is deep learning today? VII



It's all fake!

Where is deep learning today? VIII



Even scary pictures!

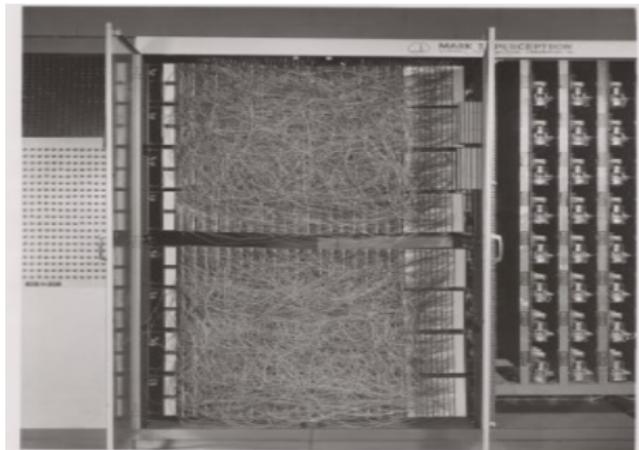
Where is deep learning today? IX



Amazing AI chatbot

The roots of deep learning I

- Actually based on old ideas from the 80s (and even earlier...)
- Started with the perceptron (50's, Frank Rosenblatt)



An old perceptron

Have a look at this video (**but not now!**)

<https://youtu.be/aygSMgK3BEM>

Deep learning

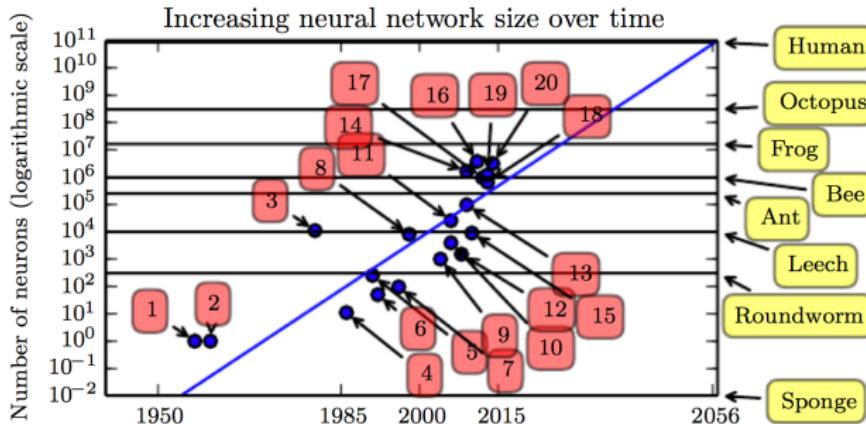
- is good old neural networks (NN)...

but with

- more data
- more computational power (GPU clusters)
- more layers
- more everything...

Their **depth** (number of layers) and **width** (number of the hidden units) has increased over the years.

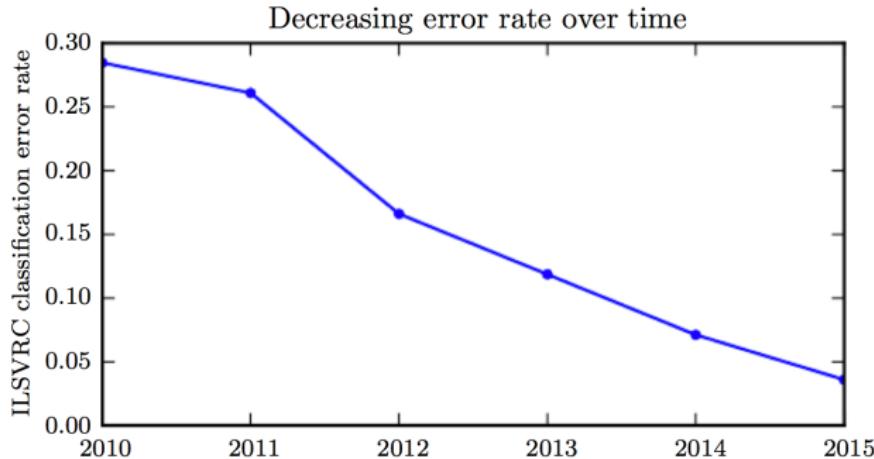
Increasing size of neural networks



[from *Deep Learning*, Goodfellow, Bengio and Courville]

- Comparison of
 - ▶ Neural networks in research papers (in red)
 - ▶ Connections in animal brains (in yellow)
- Neural networks have **doubled in size every 2.4 years**
- **⚠ Comparing computational and biological neurons is a very rough analogy!**

Strong improvement of the error rates

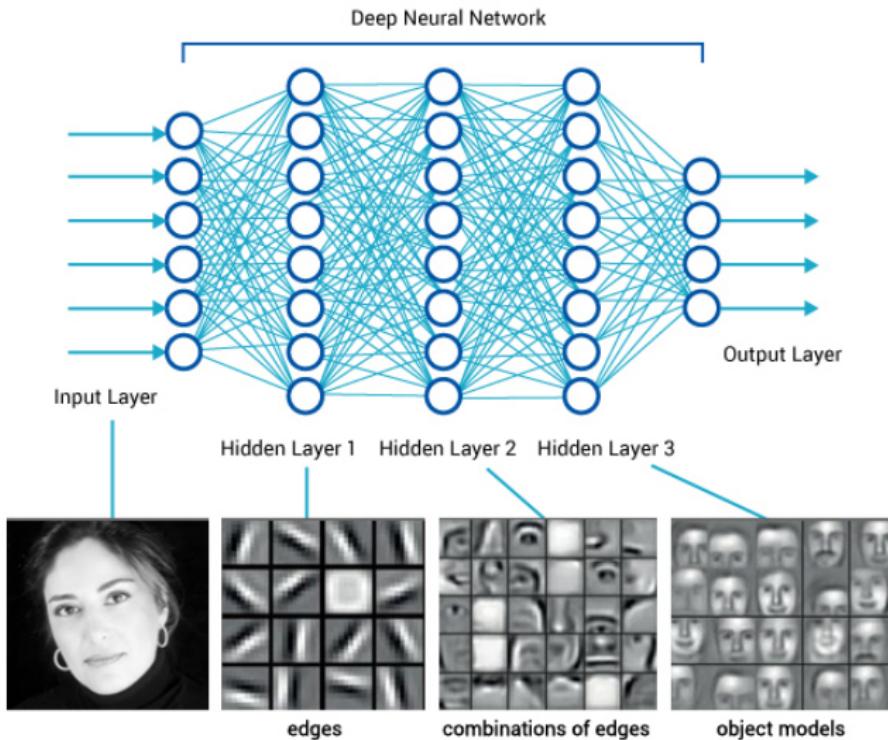


[from *Deep Learning*, Goodfellow, Bengio and Courville]

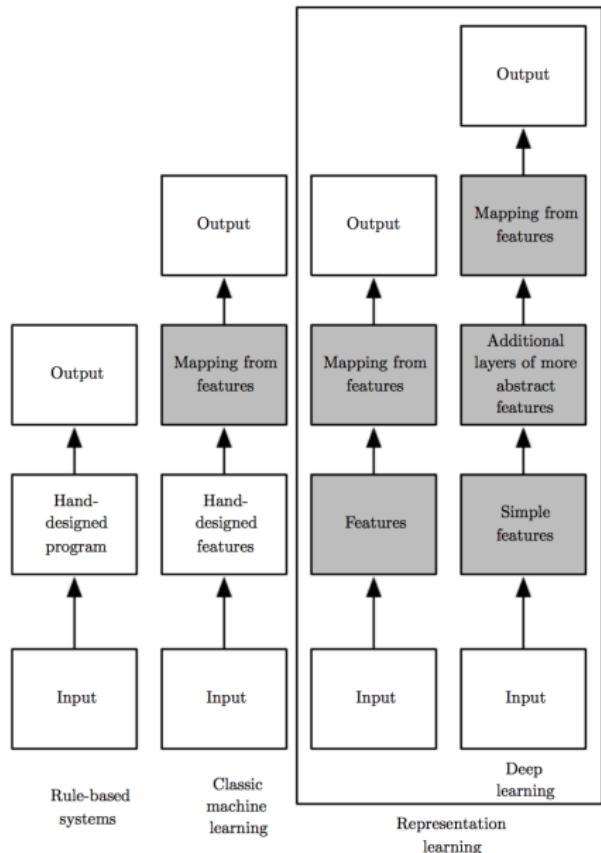
- Deep networks reached the goal of ImageNet Large Scale Visual Recognition Challenge
- Consistently won the competition and lower the error rate each year

- Deep learning is **representation learning**
- **Learn features**
- Computes **hierarchical, abstract representations** of the data
- Actually learns **several levels of abstraction** of the features

Representation learning II



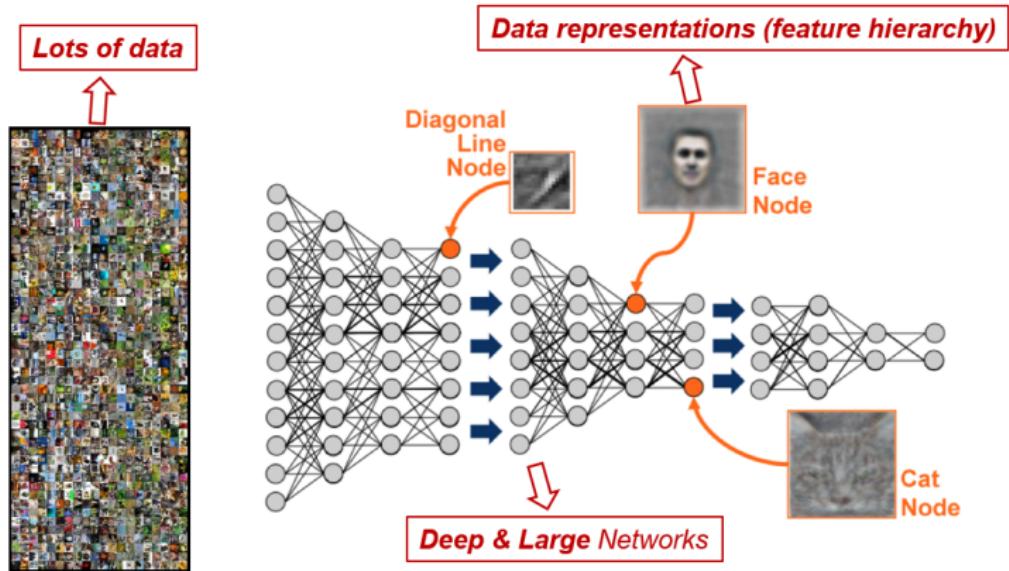
Representation learning III



Grey rectangles are machine-learned

[from *Deep Learning*, Goodfellow, Bengio and Courville]

Representation learning IV



1 Deep learning: an introduction

2 Neural networks: basic principles and architecture

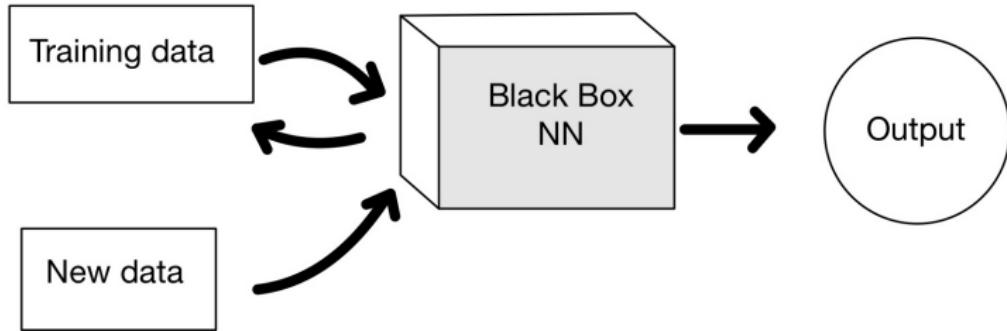
3 Tools for deep neural networks

- Forward propagation
- Backpropagation
- Optimization with SGD and mini batches
- Early stopping
- Dropout

4 Convolutional neural networks

- Convolutional layer
- Pooling

The general picture I



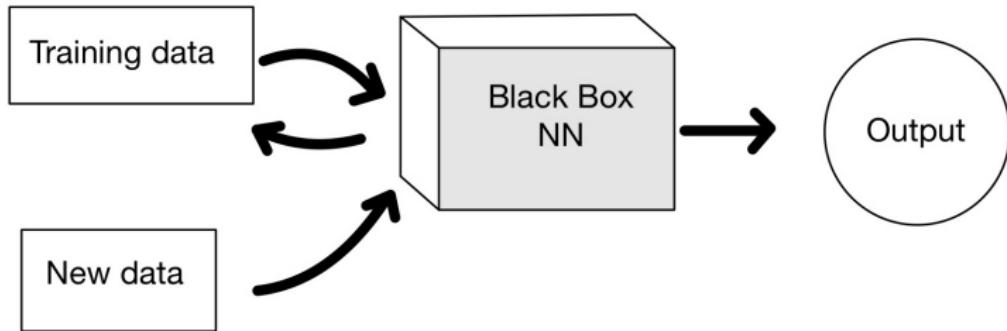
Ingredients for a deep neural network

- Input training data
 - ▶ supervised setting: $(x_i, y_i), 1 \leq i \leq n$
 - ▶ unsupervised setting: $x_i, 1 \leq i \leq n$
- NN-Black Box is defined of
 - ▶ an **architecture** \hookrightarrow chosen by the user
 - ▶ **parameters** \hookrightarrow learnt on training data by minimizing a loss
- **Loss** for training the model:
 - ▶ supervised setting: prediction accuracy on training data
 - ▶ unsupervised setting: cross entropy (negative log-likelihood) of training data
- With the trained model, for any **new** observation x^{new} :
 - ▶ supervised setting: the NN predicts an output y^{pred}
 - ▶ unsupervised setting: the NN computes the likelihood of x^{new}

Main categories of deep neural networks

- Feed-forward neural networks (FNN)
 - ▶ classical neural networks
- Convolutional neural networks (CNN)
 - ▶ specifically designed for image recognition and processing
- Recurrent neural networks (RNN)
 - ▶ for temporal data, handwriting recognition, speech recognition

What's in the box ?



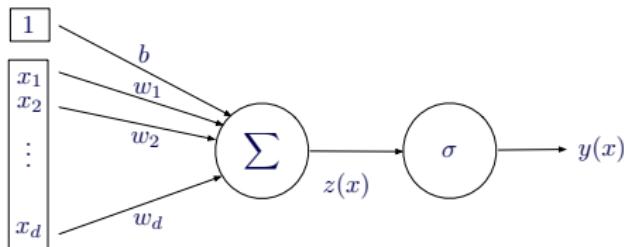
Example: Logistic regression

- Dataset $(x_1, y_1), \dots, (x_n, y_n)$ with $y_i \in \{0, 1\}$ and $x_i = (x_i^1, \dots, x_i^d) \in \mathbb{R}^d$
- Given a vector of input features $x \in \mathbb{R}^d$, consider

$$\mathbb{P}(Y = 1 | X = x) = \sigma(\langle w, x \rangle + b) = \frac{e^{\langle w, x \rangle + b}}{e^{\langle w, x \rangle + b} + 1}$$

- Model **weights** $w \in \mathbb{R}^d$ and intercept (or **bias**) $b \in \mathbb{R}$

Graphical representation of a logistic regression



Deep learning vocabulary

- $x = (x^1, \dots, x^d)$ is the **input** (a single feature vector)
- $z(x) = \langle w, x \rangle + b$ is called **pre-activation**
- $y(x) = \sigma(z(x))$ is the output (in $[0, 1]$ in this case)
- $w = (w^1, \dots, w^d)$ are **weights** and b is **bias**
- σ is an **activation** function

This is called a **unit** or an **artificial neuron**

Every artificial neuron

- takes a vector as input
- applies a **linear** mapping with weights that are to be learned
- applies a **nonlinear** known activation function
- outputs a single value

Softmax regression or Multinomial logistic regression

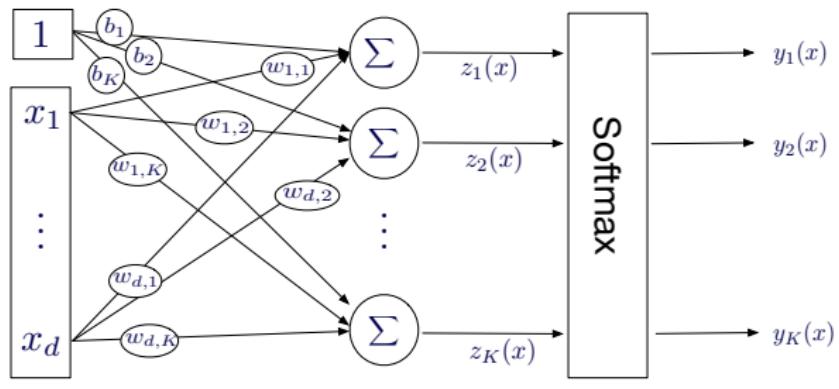
- Generalization of logistic regression to more than two classes
- Dataset $(x_1, y_1), \dots, (x_n, y_n)$ with $y_i \in \{1, \dots, K\}$ and $x_i = (x_i^1, \dots, x_i^d) \in \mathbb{R}^d$
- Given an input features vector $x \in \mathbb{R}^d$, consider

$$\mathbb{P}(Y = k | X = x) = \frac{e^{\langle w_k, x \rangle + b_k}}{\sum_{k'=1}^K e^{\langle w_{k'}, x \rangle + b_{k'}}}$$

for $k \in \{1, \dots, K\}$

- One vector of weights $w_k \in \mathbb{R}^d$ for each class k
- Stack all weights in a $d \times K$ matrix \mathbf{W} with $\mathbf{W}_{\bullet, k} = w_k$ and $\mathbf{b} = (b_1, \dots, b_K)$

Softmax regression



- input: x
- output: $y_1(x), \dots, y_K(x)$
- pre-activations or logits: $z_k(x) = \langle W_{\bullet,k}, x \rangle + b_k$
- **Softmax activation:**

$$y_k(x) = \frac{e^{z_k(x)}}{\sum_{k'=1}^K e^{z_{k'}(x)}}$$
- Matrix notation:

$$y(x) = \text{softmax}(z(x)) = \text{softmax}(W^\top x + b)$$

- This is called a **layer**
- A layer is composed of multiple neurons, also referred to as nodes
- Softmax is often the **output** layer in classification tasks

Every layer

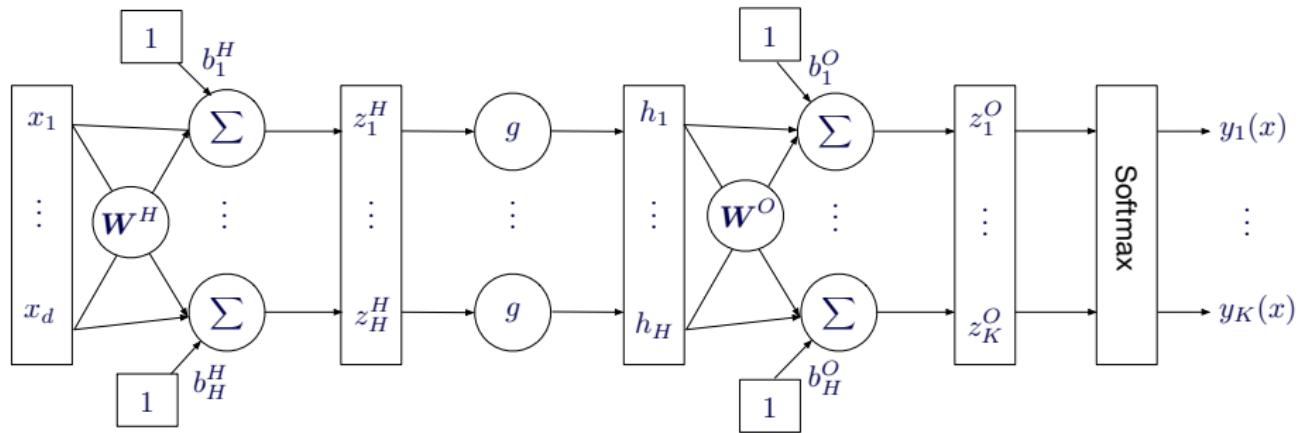
- is composed of **multiple neurons**
- applied to the same input vector
- the layer's output is the vector with
 - ▶ the values produced by the neurons or
 - ▶ the result of a function applied to those values
- the number of neurons is referred to the **width** of the layer
- A layer where **all** input neurons are connected to **all** output neurons is called **dense** or **fully connected**

Softmax regression

- Model parameters: \mathbf{W}, \mathbf{b}
- To be learnt from the training data
- Training by **optimization of a criterion** or loss function
- Here: cross entropy (negative log-likelihood)

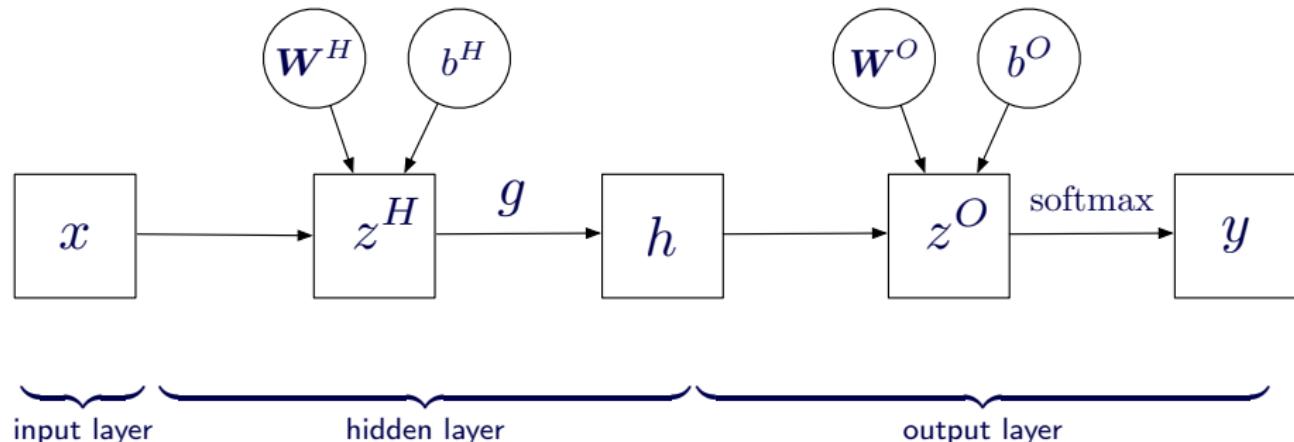
$$\ell(\mathbf{W}, \mathbf{b}) = - \sum_{i=1}^n \sum_{k=1}^K 1_{y_i=k} \log \left(\frac{e^{\langle \mathbf{w}_k, \mathbf{x} \rangle + b_k}}{\sum_{k'=1}^K e^{\langle \mathbf{w}_{k'}, \mathbf{x} \rangle + b_{k'}}} \right)$$

Softmax regression with one hidden layer



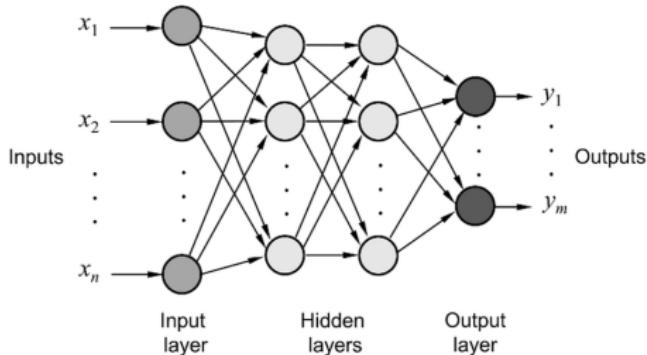
- g is a nonlinear **activation function** applied entrywise on z_1^H, \dots, z_H^H
- The **width** of the first layer is H

Alternative representation of the same network



$$\begin{aligned}
 y(x) &= \text{softmax}(z^{(O)}) = \text{softmax}(\mathbf{W}^{(O)\top} h + \mathbf{b}^{(O)}) \\
 &= \text{softmax}(\mathbf{W}^{(O)\top} g(z^{(H)}) + \mathbf{b}^{(O)}) \\
 &= \text{softmax}(\mathbf{W}^{(O)\top} g(\mathbf{W}^{(H)\top} x + \mathbf{b}^{(H)}) + \mathbf{b}^{(O)})
 \end{aligned}$$

Neural network with several hidden layers



A neural network

- is composed of multiple layers: an **input layer**, **hidden layers** and an **output layer**
- the **depth** of a neural network is the number of layers
- each layer has a specific **width**

Feed-forward neural network

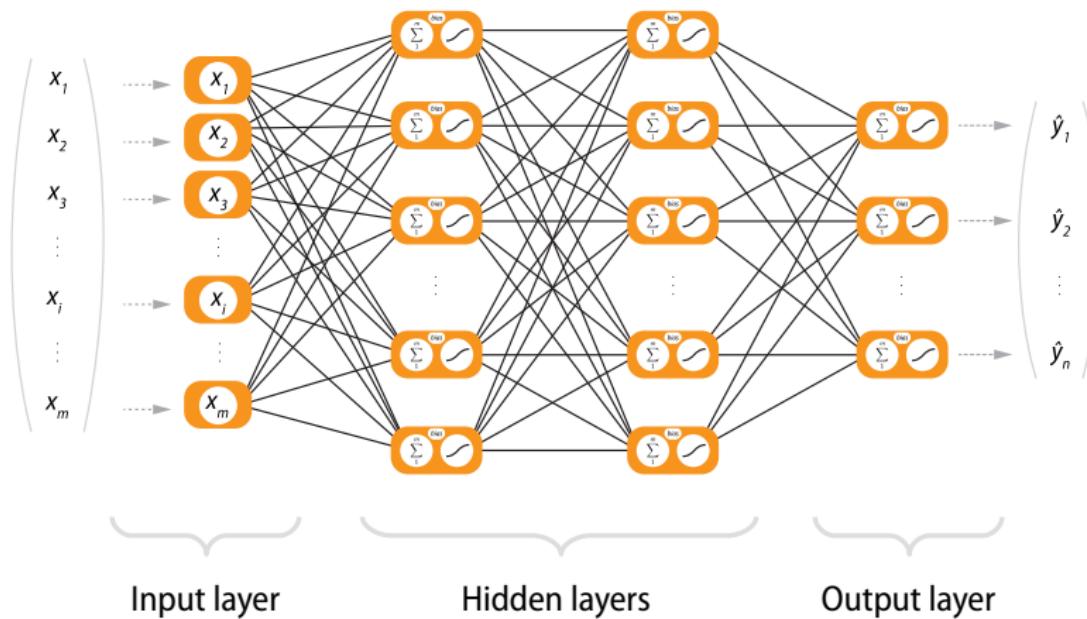
- was the first and simplest type of neural network devised
- **information moves in only one direction** - forward - from the input nodes through the hidden nodes (if any) and to the output nodes
- no cycles or loops in the network

There is ambiguity about terms and definitions...

Multi-layer perceptron (MLP)

- MLP are networks with multiple layers, interconnected in a feed-forward way, typically fully connected
- MLP and FNN are often considered to be identical, but not always...

The architecture of a FFNN I



[From FIDLE <https://fiddle.cnrs.fr>]

- Aim: **build an approximation f** of some function f^* that maps an input x to an output y , i.e. $y = f^*(x)$
- The neural network is the function f
- Layers correspond to functions that are composed in some order:
 $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$ has 3 layers
- The network describes how the functions are composed together (**computational graph**)
- $f^{(1)}$ is the first layer, while $f^{(3)}$ is the last one
- Each layer $f^{(i)}$ is the composition of
 - ▶ a **linear function**: $u \mapsto \mathbf{W}^{(i)\top} u + b^{(i)}$ and
 - ▶ a nonlinear **activation function** $g^{(i)}$ applied entrywise

Chain-based structure

$$h^{(1)} = f^{(1)}(x) = g^{(1)} \left(\mathbf{W}^{(1)\top} x + \mathbf{b}^{(1)} \right)$$

$$h^{(2)} = f^{(2)} \left(h^{(1)} \right) = g^{(2)} \left(\mathbf{W}^{(2)\top} h^{(1)} + \mathbf{b}^{(2)} \right)$$

⋮

$$h^{(L)} = f^{(L)} \left(h^{(L-1)} \right) = g^{(L)} \left(\mathbf{W}^{(L)\top} h^{(L-1)} + \mathbf{b}^{(L)} \right)$$

$$y = g^{(O)} \left(\mathbf{W}^{(O)\top} h^{(L)} + \mathbf{b}^{(O)} \right)$$

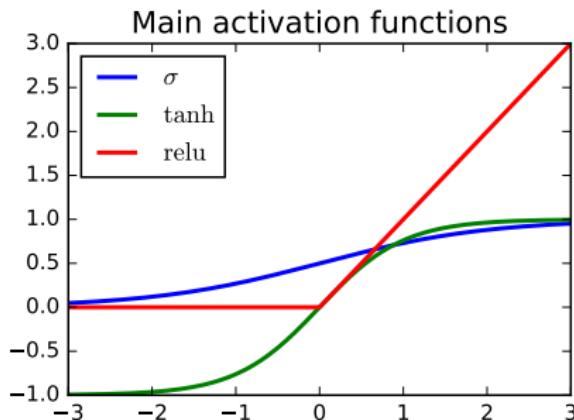
- Linear functions combined with non-linear activation functions
- Choose the width of each layer (= dimension of output vector $h^{(l)}$)
- Choose the depth L of the network

Function approximation

- The **universal approximation theorem** states that a basic FNN is a universal approximator
- Roughly, if there is no limit on the widths of the layers, for any function f^* , there exists a FNN f that approximates f^* with desired accuracy
- This also holds for shallow networks when the number of hidden units is sufficiently large
- **Nonlinear activations** functions are key

Activation functions

- Activations introduce **nonlinearity** in the model
- Without activation, the whole network represents only a simple linear function
- Main activation functions:
 - ▶ **sigmoid**: $\sigma(z) = \frac{1}{1+e^{-z}}$
 - ▶ **tanh**: $\tanh(z) = \frac{e^{2z}-1}{e^{2z}+1}$
 - ▶ **ReLu**: $\text{ReLU}(z) = \max(0, z)$
- Note that $\tanh(z) = 2\sigma(2z) - 1$
- Applied entrywise to the inputs



ReLU activation

- ReLu = rectified linear unit
- Easy to optimize, since its behaviour is closer to linear
- Its gradient is not defined at $z = 0$: not a problem since during training, it's unlikely that many inputs are equal to 0
- Recent research: works very well in practice for deep neural networks
- Sigmoid activations σ and \tanh saturate for large positive or large negative values, while ReLu doesn't. This is why sigmoid activations in hidden units are now not recommended

Derivatives of activation functions

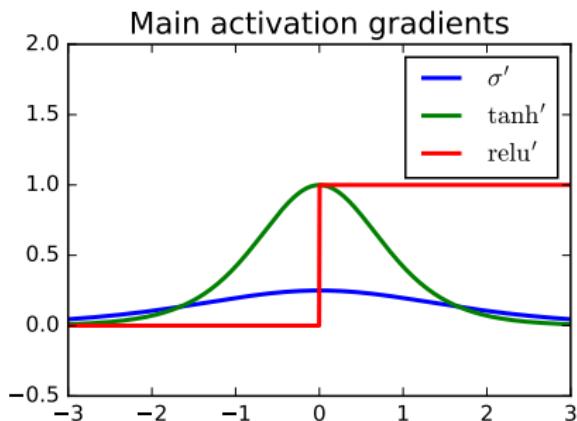
- **sigmoid:**

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

- **tanh:**

$$\tanh'(z) = 1 - \tanh^2(z)$$

- **ReLU:** $\text{ReLu}'(z) = 1_{z>0}$



Questions

- how many layers?
- width of the different layers?
- shallow versus deep neural networks

Shallow neural networks

- on simple problems a single layer can definitely get the job done
- on more challenging tasks one might not reach acceptable levels of accuracy
- generally need more units per layer and so more parameters to accommodate the complexity
- easier to optimize

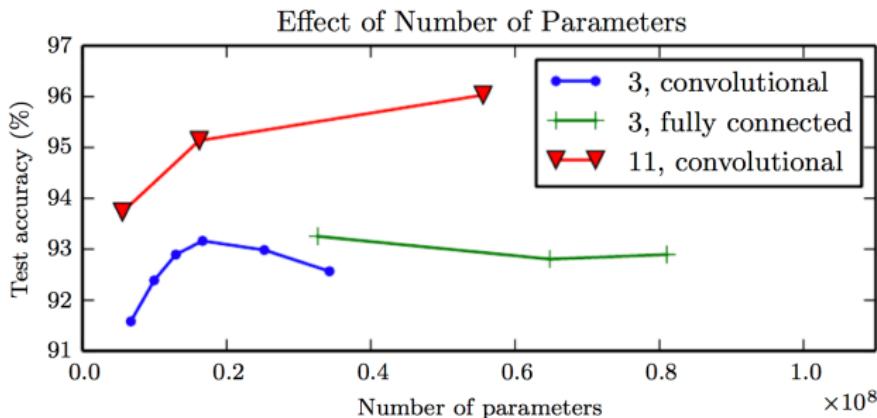
Deeper neural networks

- can get the job done with **fewer units per layer** and far fewer parameters
- leads to **good generalizations** on a test set
- can be difficult to train

Historically

- First neural networks were explored with shallow architectures and performance was unsatisfactory
- The breakthrough of neural networks came when deeper architectures were devised

Shallow versus deep neural networks



[from *Deep Learning*, Goodfellow, Bengio and Courville]

- Performance of a fully connected model is achieved with a CNN with same depth but fewer parameters
- A deep CNN clearly outperforms a shallow CNN with the same number of parameters

Outline Day 3

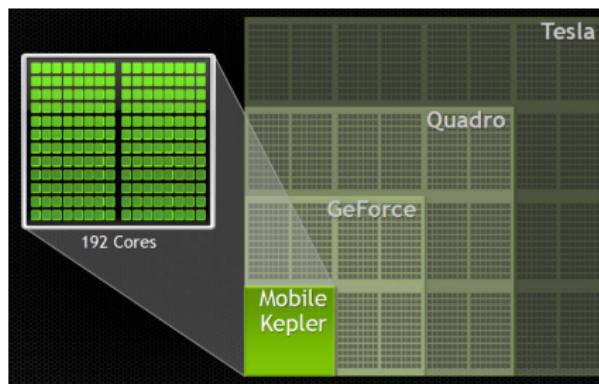
- 1 Deep learning: an introduction
- 2 Neural networks: basic principles and architecture
- 3 Tools for deep neural networks
 - Forward propagation
 - Backpropagation
 - Optimization with SGD and mini batches
 - Early stopping
 - Dropout
- 4 Convolutional neural networks
 - Convolutional layer
 - Pooling

Algorithmic tools

- **Stochastic optimization algorithm**
- **Forward propagation** to evaluate the loss and for prediction
- **Backpropagation** to compute gradients (uses forward propagation at the beginning)

Computational tools

- Nice **open source libraries**:
pytorch, tensorflow, caffe,
keras
- **GPU** to perform massive parallel computations



Our goal today

- understand these concepts and methods sufficiently
- be able to make good choices for the corresponding options and parameters of the functions in the standard libraries

- 1 Deep learning: an introduction
- 2 Neural networks: basic principles and architecture
- 3 Tools for deep neural networks
 - Forward propagation
 - Backpropagation
 - Optimization with SGD and mini batches
 - Early stopping
 - Dropout
- 4 Convolutional neural networks
 - Convolutional layer
 - Pooling

The objective function

- Minimization of an **objective function**
- In general, the objective is an **average loss** evaluated over all training points

$$L(\theta) = \frac{1}{n} \sum_{i=1}^n \ell(y_i, f(x_i; \theta)) + \text{pen}(\theta)$$

where

- ▶ ℓ is a **loss function**
- ▶ $f(\cdot; \theta)$ is the output of the network
- ▶ $\text{pen}(\theta)$ is a **penalty** possibly added for regularization to avoid overfitting, such as ridge

$$\text{pen}(\theta) = \text{pen}(\mathbf{W}^{(O)}, \mathbf{W}^{(H)}) = \lambda (\|\mathbf{W}^{(O)}\|_F^2 + \|\mathbf{W}^{(H)}\|_F^2)$$

Biases are not penalized

Example of an objective function

- Consider the one-hidden layer FNN

$$y = \text{softmax}(\mathbf{W}^{(O)\top} g(\mathbf{W}^{(H)\top} x + \mathbf{b}^{(H)}) + \mathbf{b}^{(O)})$$

- The **empirical loss** or goodness of fit of parameters $\theta = (\mathbf{W}^H, b^H, \mathbf{W}^O, \mathbf{b}^O)$ is the negative log-likelihood (also called cross-entropy):

$$L(\theta) = -\text{LogLik}(\theta)$$

$$= -\frac{1}{n} \sum_{i=1}^n \sum_{k=1}^K 1_{y_i=k} \log \left(\frac{\exp \left(\langle \mathbf{W}_{\bullet,k}^{(O)}, g(\mathbf{W}^{(H)\top} x_i + \mathbf{b}^{(H)}) \rangle + b_k^{(O)} \right)}{\sum_{k'=1}^K \exp \left(\langle \mathbf{W}_{\bullet,k'}^{(O)}, g(\mathbf{W}^{(H)\top} x_i + \mathbf{b}^{(H)}) \rangle + b_{k'}^{(O)} \right)} \right)$$

Minimization of the objective function

- Apply an **iterative** procedure based on **gradient descent**
- At each step, the gradient $\nabla_{\theta} L(\theta)$ has to be evaluated
- This involves
 - ▶ **backpropagation**
 - ▶ **forward propagation** and
 - ▶ **mini-batch gradients** as a fast approximation of $\nabla_{\theta} L(\theta)$

What is forward propagation?

- The **evaluation of the neural network f** on a given input x
- Process of **propagating values in the network**
- Consider the network as a **computational graph**
- The input data is fed in the forward direction through the network
- Each hidden layer accepts the input data, processes it and passes to the successive layer :

$$f(x) = f^{(L)} \circ \dots \circ f^{(2)} \circ f^{(1)}(x)$$

- For backpropagation it is useful to store the intermediate results of the layers $h^{(k)}(x) = f^{(k)}(h^{(k-1)}(x))$ for $k = 1, \dots, L$

When is forward propagation performed?

- In the learning algorithm
 - ▶ At every iteration, for the **evaluation of the loss** $\ell(y_i, f(x_i; \theta))$ for observations x_i and current parameter value θ
 - ▶ The loss is evaluated only on a mini batch of data points x_i (see below)
- After training the neural network
 - ▶ To compute the **prediction** $y^{\text{pred}} = f(x^{\text{new}}; \hat{\theta})$ for a new datapoint x^{new} with trained parameter $\hat{\theta}$

- 1 Deep learning: an introduction
- 2 Neural networks: basic principles and architecture
- 3 Tools for deep neural networks
 - Forward propagation
 - Backpropagation
 - Optimization with SGD and mini batches
 - Early stopping
 - Dropout
- 4 Convolutional neural networks
 - Convolutional layer
 - Pooling

What is backpropagation?

- Algorithm to **compute gradients**
- Uses the **chain rule** to compute the derivative of composed functions

$$f(g(x))' = f'(g(x)) \times g'(x)$$

- Not specific to neural networks

Backpropagation for training neural networks

- Perform gradient descent to minimize the objective function F
- This involves the computation of gradients of
 $f(x, \theta) = f^{(L)} \circ f^{(L-1)} \circ \dots \circ f^{(1)}(x)$
- Backpropagation computes

$$\nabla_{\theta} f(x, \theta)$$

$$= f'^{(L)}(h^{(L-1)}(x)) \times f'^{(L-1)}(h^{(L-2)}(x)) \times \dots \times f'^{(2)}(h^{(1)}(x)) \times f'^{(1)}(x)$$

When is backpropagation performed?

- In the learning algorithm
 - ▶ At each iteration, evaluate the gradient of the loss for observations x_i with current parameter value θ :

$$\nabla_{\theta} \ell(y_i, f(x_i; \theta)) = \ell'(y_i, f(x_i; \theta)) \times \nabla_{\theta} f(x_i; \theta)$$

- ▶ This is done only for a **mini batch** of data points x_i (see below)

- 1 Deep learning: an introduction
- 2 Neural networks: basic principles and architecture
- 3 Tools for deep neural networks
 - Forward propagation
 - Backpropagation
 - Optimization with SGD and mini batches
 - Early stopping
 - Dropout
- 4 Convolutional neural networks
 - Convolutional layer
 - Pooling

Stochastic gradient descent (SGD)

- Same algorithm as the one we have already seen
- Deep learning is powered by this very important algorithm
- With stochastic gradients based on **mini batches**

Mini-batch gradients

- Choose a mini-batch size m
- Often m is a power of two to accommodate GPU, typically
 $m \in \{32, 64, \dots, 256\}$
- Draw at random a subset $B \subset \{1, \dots, n\}$ of indices of size $|B| = m$
- The **mini-batch gradient** is given by

$$\frac{1}{m} \sum_{i \in B} \nabla_{\theta} \ell(y_i, f_{\theta}(x_i)) + \nabla_{\theta} \text{pen}(\theta)$$

- **Recipe** If the network is deep, use m small (≈ 16) and waaaaaiiiit !

Mini batches and epochs

- With mini batches only a fraction (m/n) of the dataset is visited during one iteration
- An **epoch** corresponds to one pass through the dataset on average, that is, to n/m iterations
- E.g. with mini batches of size $\alpha = 20\%$ of the data, an epoch corresponds to $1/\alpha = 1/0.2 = 5$ iterations
- In practice, we choose the mini-batch size m and the number of epochs N (see below early stopping)
- During the entire SGD algorithm $m \times N$ data points are visited

Algorithm 1 SGD for training a neural network

Require: initial parameter θ , mini-batch size m , learning rates η_t

- 1: **while** stopping criterion is not met **do**
- 2: Sample a mini batch B of size m
- 3: Compute the gradient estimate

$$\nabla_{\theta} \mathcal{L} \leftarrow \frac{1}{m} \sum_{i \in B} \nabla_{\theta} \ell(y_i, f(x_i; \theta)) + \nabla_{\theta} \text{pen}(\theta)$$

- 4: Apply the update

$$\theta \leftarrow \theta - \eta_t \nabla_{\theta} \mathcal{L}$$

- 5: **end while**
-

Learning rates

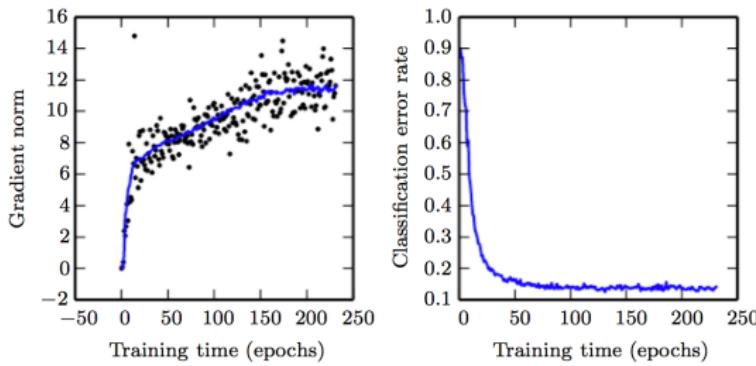
- Learning rates have an impact on the **convergence** of the algorithm
- $(\eta_t)_t$ should be decreasing along iterations t
- Choosing learning rates is more an art than science...
- Some SGD implementations include **adaptive** choices, others just rely on **default values**
- Try to modify this parameter and look at the impact

Popular implementations of SGD

- **AdaGrad**: scales training rates using a cumulative sums of squared past values of the gradients
- **RMSProp**: modifies AdaGrad by replacing cumulative sums by exponentially weighted averages
- **Adam, Adadelta**: some extra tricks...

Non convex optimisation problem

- The objective function of a neural network has an **extremely large number of local minima**
- SGD often **does not converge to a critical point** (neither to a local minimum nor a saddle point)
- Gradient norm **increases (!!)** along iterations while training seems successful
- Strong departure from convex optimization



To make things work...

- Deep neural networks are extremely complex objects and many choices have to be done
- Strong interplay between **network design, learning algorithm and regularization**
- Success depends on many, many factors and also on the **quality of the data**
- If training is not successful **spend more time on tuning!**

Open questions on training neural networks

- Due to non convexity, the learning algorithm generally ends up in a local minima instead of the global minimum
 - ↪ What is the **practical interest of local minima** for neural networks?
- Are there guarantees that optimization algorithms **converge to local minima**?

For many years

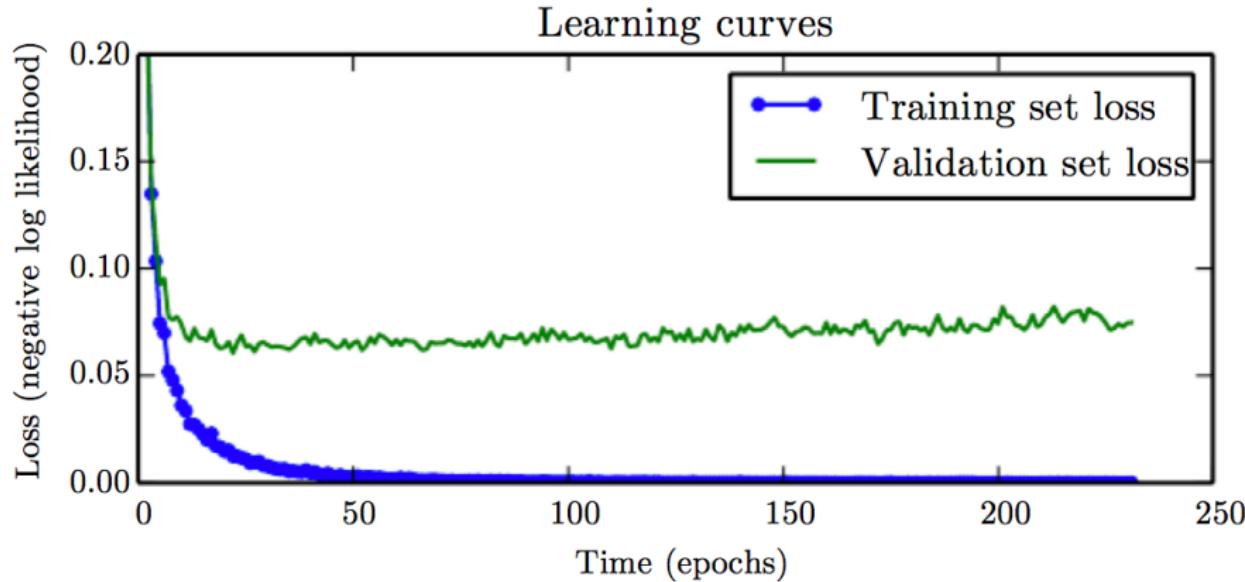
- We were **frightened of local minima** (especially mathematicians)
- We thought that local minima were plaguing neural networks

This is not the case anymore

- Now common belief is that for sufficiently large networks, **local minima are valuable and come with low computational cost**
- It is not important to find a global minimum, but a **good local minimum**

Overfitting

- Deep neural networks have very many parameters and thus **high representational power**, called **model capacity**
 - Training large models might lead to **overfitting** the training dataset
 - Training error goes down, while validation error rises
 - Often occurs after only a couple of passes over the data
 - Apply a form of **regularization**



[from *Deep Learning*, Goodfellow, Bengio and Courville]

- 1 Deep learning: an introduction
- 2 Neural networks: basic principles and architecture
- 3 Tools for deep neural networks
 - Forward propagation
 - Backpropagation
 - Optimization with SGD and mini batches
 - Early stopping
 - Dropout
- 4 Convolutional neural networks
 - Convolutional layer
 - Pooling

Early stopping

- Principle: Stop training when the validation error starts to increase
- In theory, when the validation error starts to increase, you should use the parameter θ obtained many iterations ago
- Simple to implement: save parameters along the epochs, and compute validation error at every epoch
- Probably the most widely used form of regularization in deep learning: very simple and efficient
- Number of epochs becomes a meta-parameter of the algorithm: can be saved and re-used later

- 1 Deep learning: an introduction
- 2 Neural networks: basic principles and architecture
- 3 Tools for deep neural networks
 - Forward propagation
 - Backpropagation
 - Optimization with SGD and mini batches
 - Early stopping
 - Dropout
- 4 Convolutional neural networks
 - Convolutional layer
 - Pooling

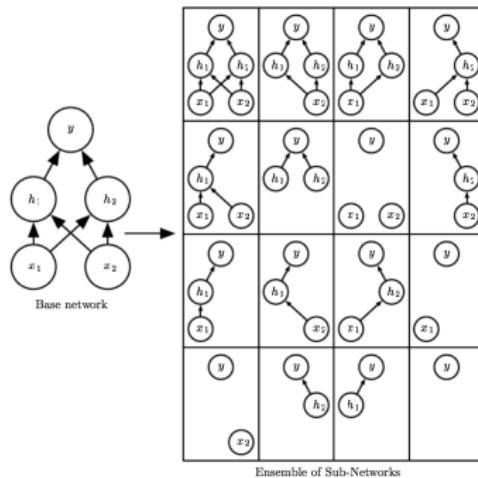
Ensemble learning

- Learning several models in parallel and averaging them (bagging) reduces overfitting
- Requires the computational expense of training and maintaining multiple models
↪ not feasible for neural networks

Dropout

- A single model can be used to simulate having a large number of different network architectures by **randomly dropping out nodes** during training
- At any iteration, remove any hidden unit with probability p (often $p = 1/2$)
- Dropout has the effect of reducing the model capacity
- Computationally very cheap and remarkably effective regularization method
- Patented by Google! (but we don't care)

Example of a network and all possible sub-networks



[from *Deep Learning*, Goodfellow, Bengio and Courville]

- Sixteen possible subgraphs
- Only keep the ones with connections between the input and output

Outline Day 3

- 1 Deep learning: an introduction
- 2 Neural networks: basic principles and architecture
- 3 Tools for deep neural networks
 - Forward propagation
 - Backpropagation
 - Optimization with SGD and mini batches
 - Early stopping
 - Dropout
- 4 Convolutional neural networks
 - Convolutional layer
 - Pooling

- 1 Deep learning: an introduction
- 2 Neural networks: basic principles and architecture
- 3 Tools for deep neural networks
 - Forward propagation
 - Backpropagation
 - Optimization with SGD and mini batches
 - Early stopping
 - Dropout
- 4 Convolutional neural networks
 - Convolutional layer
 - Pooling

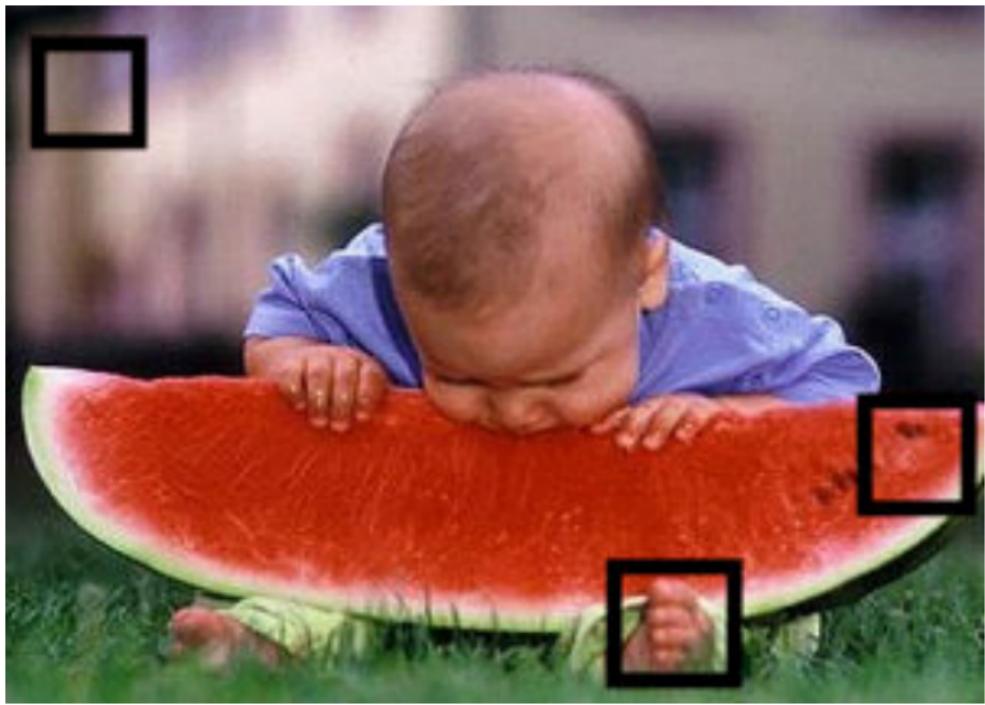
Playground time !!!

<http://playground.tensorflow.org>

Convolutional neural networks I



Convolutional neural networks II



General idea of convolutional neural networks (CNN)

- A convolutional layer slides a **filter** over the image and **extracts features** resulting in a feature map that can be fed to the next convolutional layer to extract **higher-level features**
- Stacking multiple convolutional layers allows CNNs to recognize increasingly complex structures and objects in an image

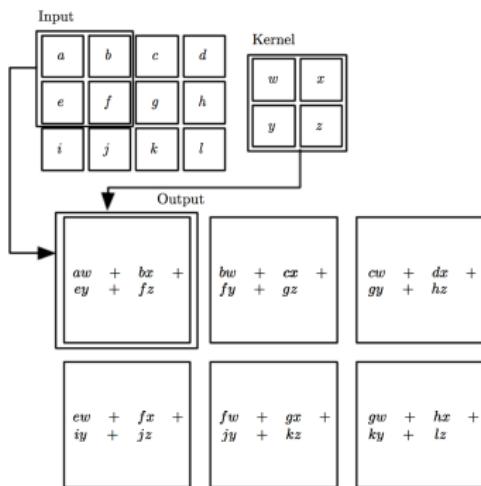
Convolutional neural networks

- CNN or ConvNet
- Neural networks that use convolution
- Convolution operator in neural networks:

$$S(i, j) = (\mathcal{I} \star K)(i, j) = \sum_k \sum_l \mathcal{I}(i + k, j + l)K(k, l)$$

- ▶ \mathcal{I} is the input (an image)
- ▶ K is called the **kernel** function, which is non zero only on a small region
- ▶ The kernel K has **to be learned**
- A convolution over an entire image is a linear function $f(x) = \mathbf{W}x$ with **constraint weight matrix \mathbf{W}**
 - ▶ Many zero values in \mathbf{W} : to process only small batches of the image
 - ▶ Many identical non zero values coming from the kernel K

Example of 2D convolution



[from *Deep Learning*, Goodfellow, Bengio and Courville]

- Only kernel positions where the entire kernel lies within the image are considered (called **valid** convolution)
- Parameter sharing: the same kernel is used for all positions

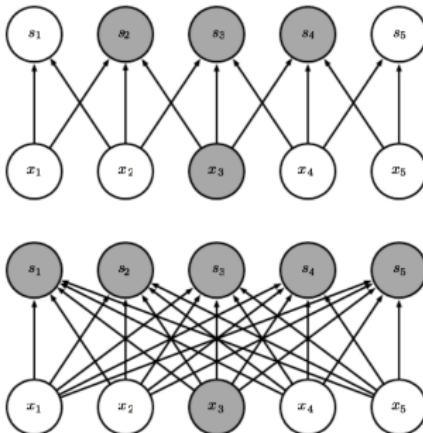
A convolutional layer

- contains only few connections between input and output nodes (**not dense**)
- uses only few parameters as the **same** kernel is used for all neurons (**parameter sharing**)
- This is a way of regularization
- reduces the dimension of the image (thus the width of the layer)
- is prone to detect small meaningful **features** such as edges in images with millions of pixels

Very few parameters

- ↪ **improves statistical efficiency**
- ↪ **improves memory**
- ↪ **faster computations**

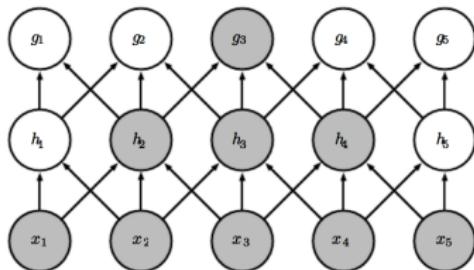
Sparse connectivity



[from *Deep Learning*, Goodfellow, Bengio and Courville]

- **Top:** convolution with a kernel of width 3: only three outputs are affected by input $x_3 \hookrightarrow$ **connectivity is sparse**
- **Bottom:** fully connected layer: all outputs depend on $x_3 \hookrightarrow$ dense

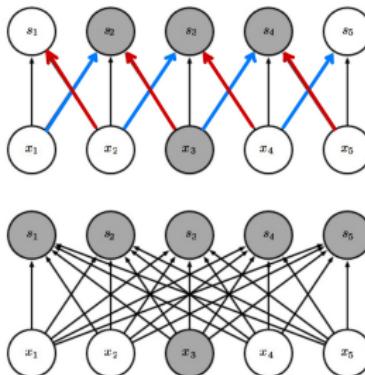
Multiple convolutional layers



[from *Deep Learning*, Goodfellow, Bengio and Courville]

- Deeper layers are indirectly connected to the whole input image
- Learn **hierarchical features** and representations of the image

Parameter sharing



[from *Deep Learning*, Goodfellow, Bengio and Courville]

- **Top:** convolution layer: the same weights are used in all 5 neurons
↪ a total of 3 parameters
- **Bottom:** fully connected layer: every connection comes with its own weight
↪ a total of 25 parameters

Equivariance of convolutional layers

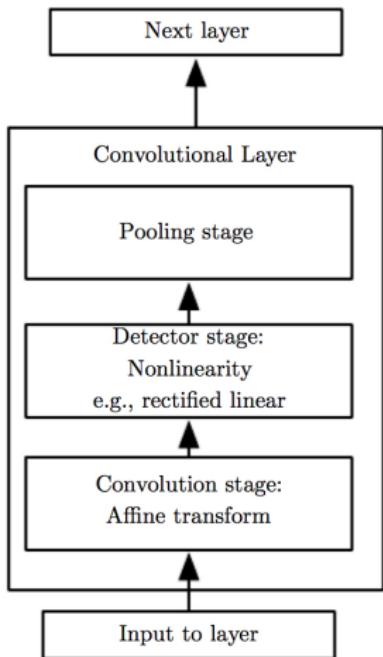
- Convolution has the property of **equivariance to translation**
- The same function is applied to a small number of neighboring pixels at different places of the image
- If the input is changed by a translation, then the output changes in the same way
- If an object is moved inside an image, the output changes accordingly
- Can be useful for **feature detection**
- Sometimes it is less useful, e.g. when images are cropped and centered on individuals' faces for instance

- 1 Deep learning: an introduction
- 2 Neural networks: basic principles and architecture
- 3 Tools for deep neural networks
 - Forward propagation
 - Backpropagation
 - Optimization with SGD and mini batches
 - Early stopping
 - Dropout
- 4 Convolutional neural networks
 - Convolutional layer
 - Pooling

A CNN layer

- typically includes 3 operations:

convolution + activation + pooling



[from *Deep Learning*, Goodfellow, Bengio and Courville]

Pooling

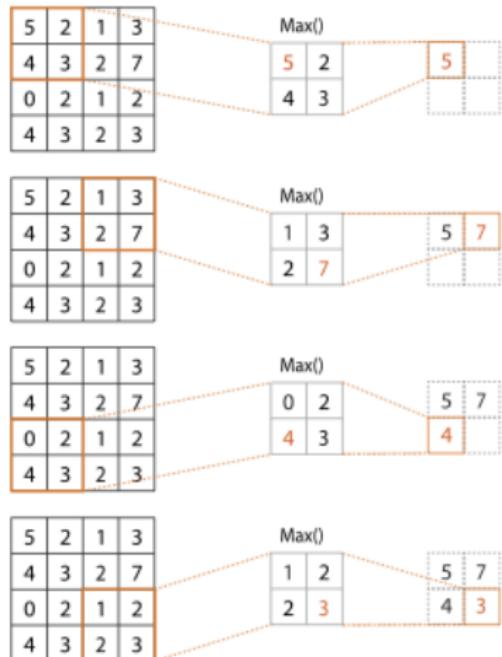
- Technique for **generalizing features** extracted by convolutional filters and recognizing features **independent of their location** in the image
- Helps the representation to become approximately invariant to small translations of the input
- If a small translation is applied, the output of the layer is almost unchanged
- Very useful for **feature detection** when the specific position in the image is less important
- Allows to handle **inputs with different sizes**: pictures can have different sizes, but the output classification layer must be of fixed size

Pooling

- Sliding a window over the outputs of a convolutional layer
- Compute a **summary statistic** such as average or max

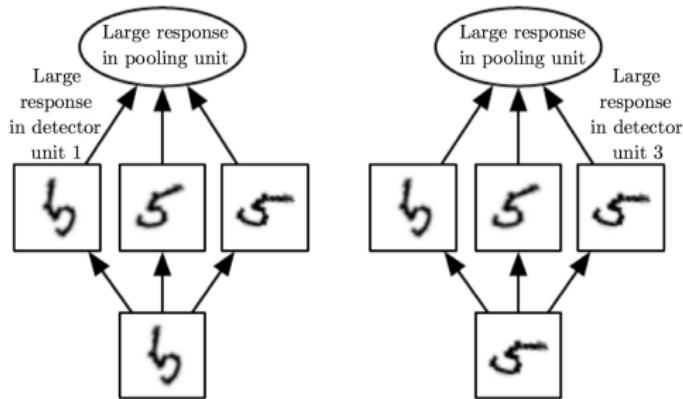
MaxPooling

- most widely used is **max aggregation**, also called max-pooling



[From FIDLE <https://fidle.cnrs.fr>]

Pooling over multiple convolutions



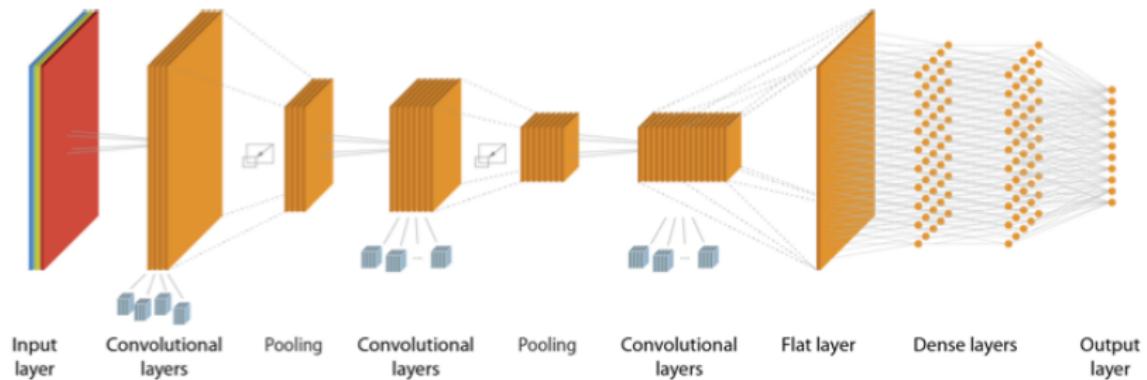
[from *Deep Learning*, Goodfellow, Bengio and Courville]

- Max pooling over multiple features can make the network **invariant to transformations** of the input such as rotations
- In general, pooling helps in **learning invariances**

Many tricks and variants of convolutional layers

- downsampling
- convolution with a stride
- unshared convolution or locally connected layers
- tiled convolution
- zero padding
- ...

CNN design



[From FIDLE <https://fiddle.cnrs.fr>]

- (○) Backfed Input Cell
- (●) Input Cell
- (△) Noisy Input Cell
- (●) Hidden Cell
- (●) Probabilistic Hidden Cell
- (△) Spiking Hidden Cell
- (●) Output Cell
- (○) Match Input Output Cell
- (●) Recurrent Cell
- (○) Memory Cell
- (△) Different Memory Cell
- (●) Kernel
- (○) Convolution or Pool

A mostly complete chart of

Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

Perceptron (P)



Feed Forward (FF)



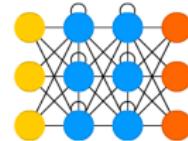
Radial Basis Network (RBF)



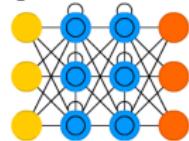
Deep Feed Forward (DFF)



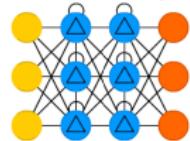
Recurrent Neural Network (RNN)



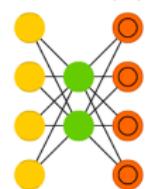
Long / Short Term Memory (LSTM)



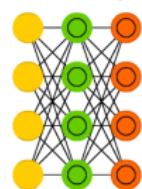
Gated Recurrent Unit (GRU)



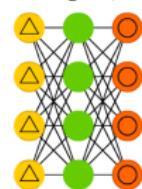
Auto Encoder (AE)



Variational AE (VAE)



Denoising AE (DAE)



Sparse AE (SAE)



The jungle of NN II

Markov Chain (MC)



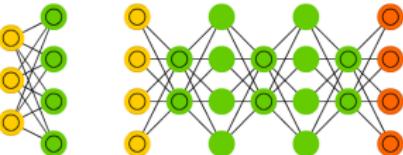
Hopfield Network (HN)



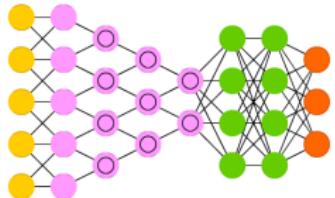
Boltzmann Machine (BM) / Restricted BM (RBM)



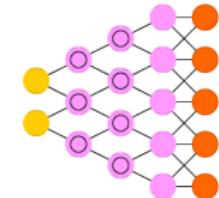
Deep Belief Network (DBN)



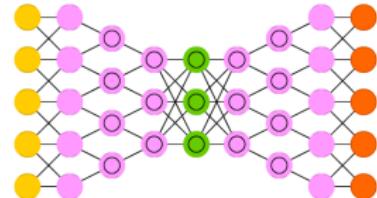
Deep Convolutional Network (DCN)



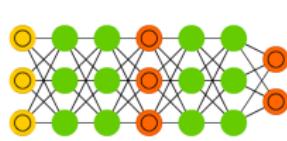
Deconvolutional Network (DN)



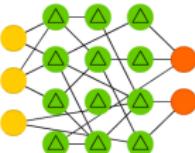
Deep Convolutional Inverse Graphics Network (DCIGN)



Generative Adversarial Network (GAN)



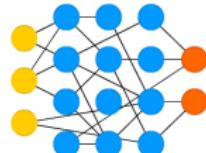
Liquid State Machine (LSM)



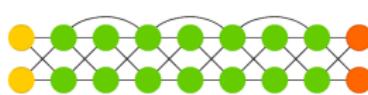
Extreme Learning Machine (ELM)



Echo State Network (ESN)



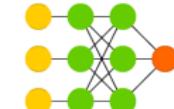
Deep Residual Network (DRN)



Kohonen Network (KN)



Support Vector Machine (SVM)



Neural Turing Machine (NTM)



Thank you!