Introduction to Machine Learning and Deep Learning with Python

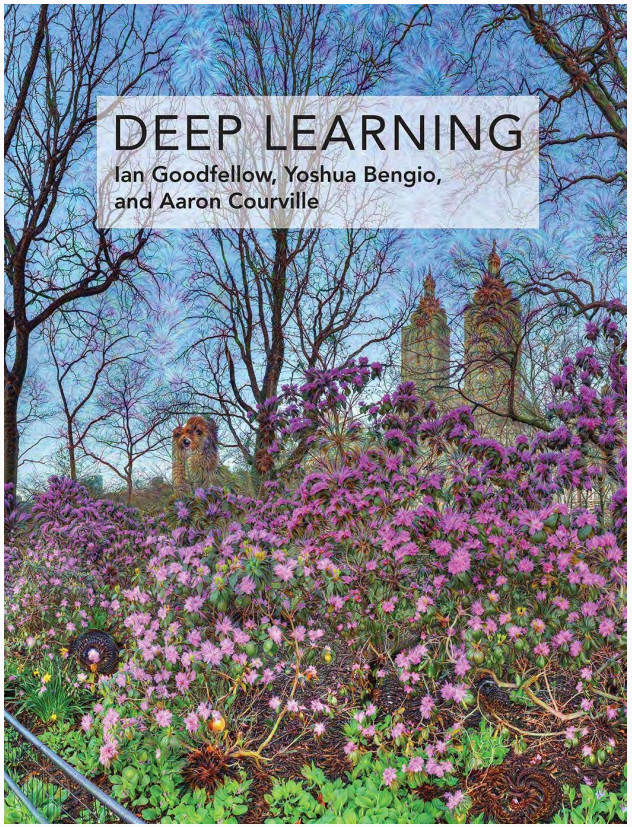
Day 3

CNRS Formation Entreprise

11 – 13 June, 2025, Paris

Tabea Rebafka

Day 3: Deep Learning

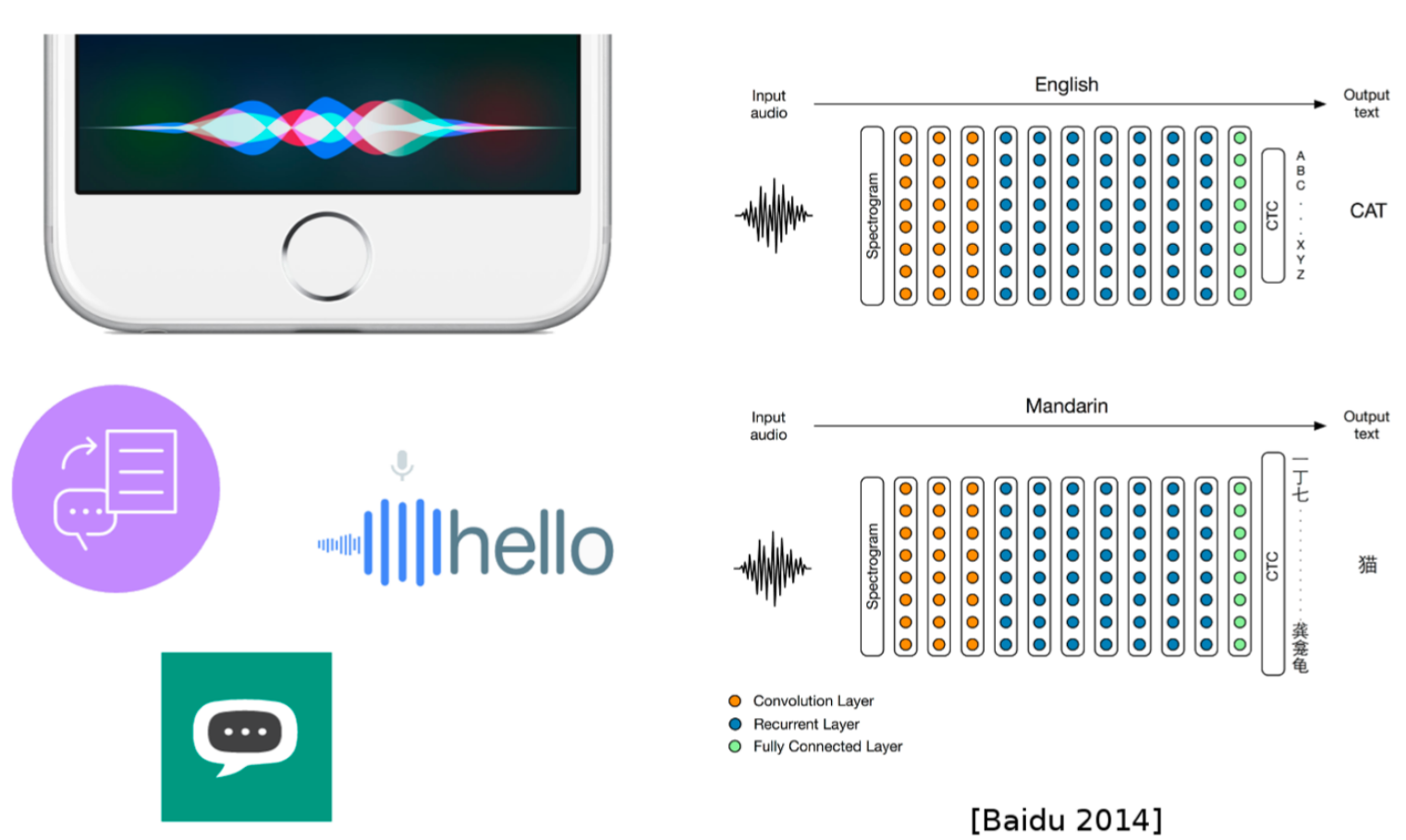


Outline of Day 3

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

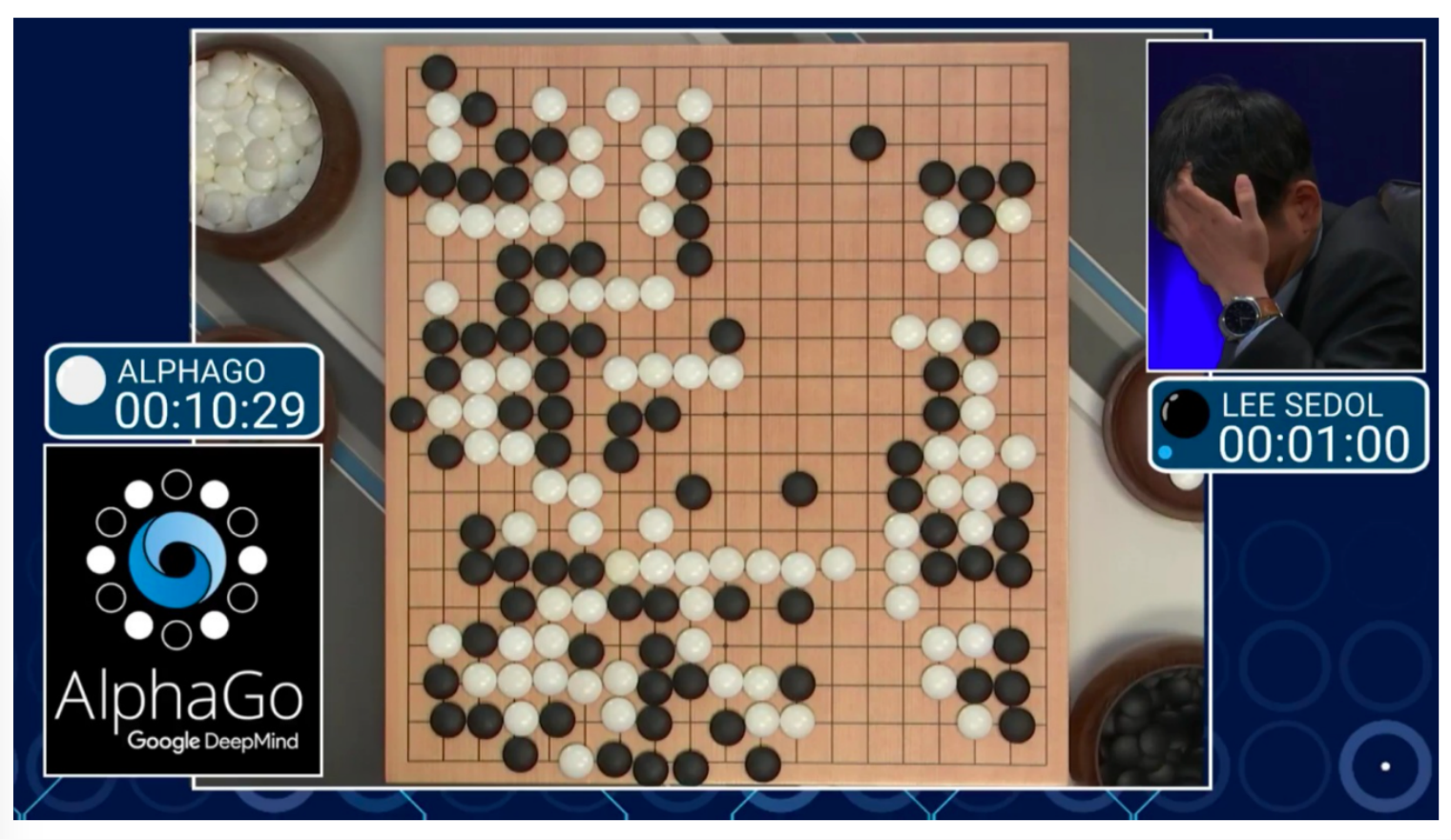
Outline Day 3

Where is deep learning today? i



Speech recognition

Where is deep learning today? ii



Beat humans at Go (deep learning and reinforcement learning)

Where is deep learning today? iii

Stanford | News

Search Stanford news...

Home Find Stories For Journalists Contact

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.

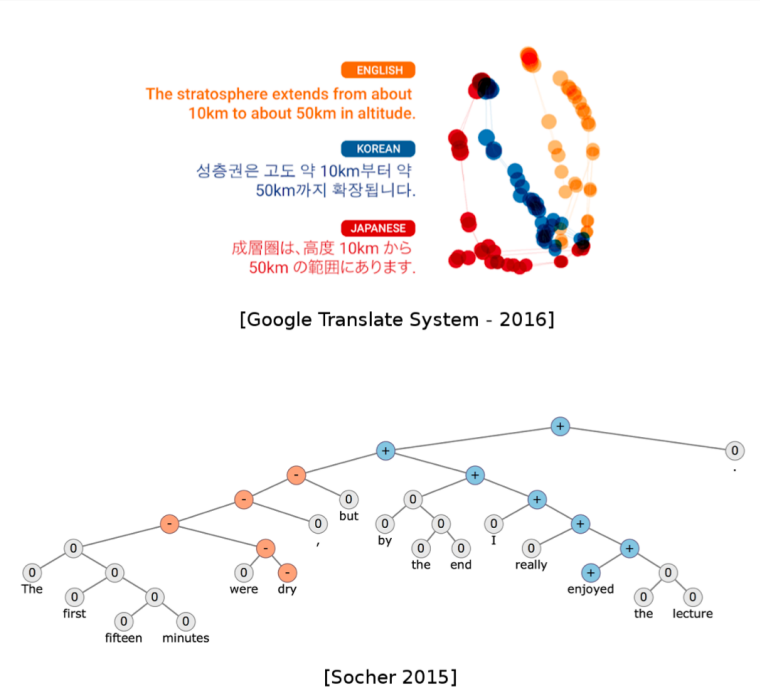
BY TAYLOR KUBOTA

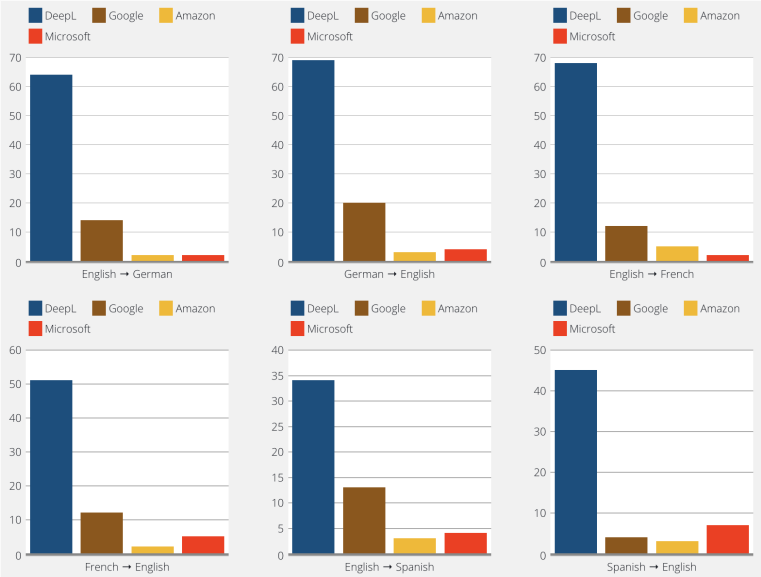
It's scary enough making a doctor's appointment to see if a strange mole could be cancerous. Imagine, then, that you were in that situation while also living far away from the nearest doctor, unable to take time off work and unsure you had the money to cover the cost of the visit. In a scenario like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





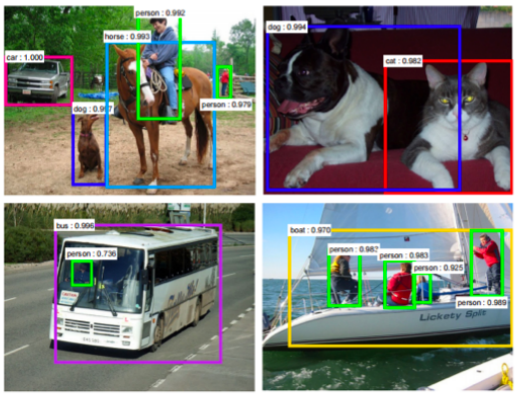
[DeepL 2020]

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

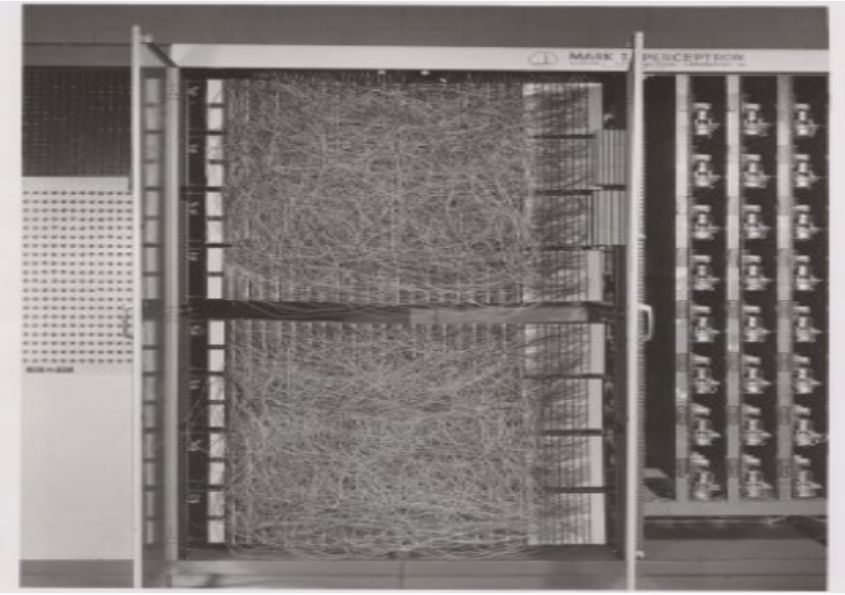
Even scary pictures!

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`

The roots of deep learning ii

Deep learning

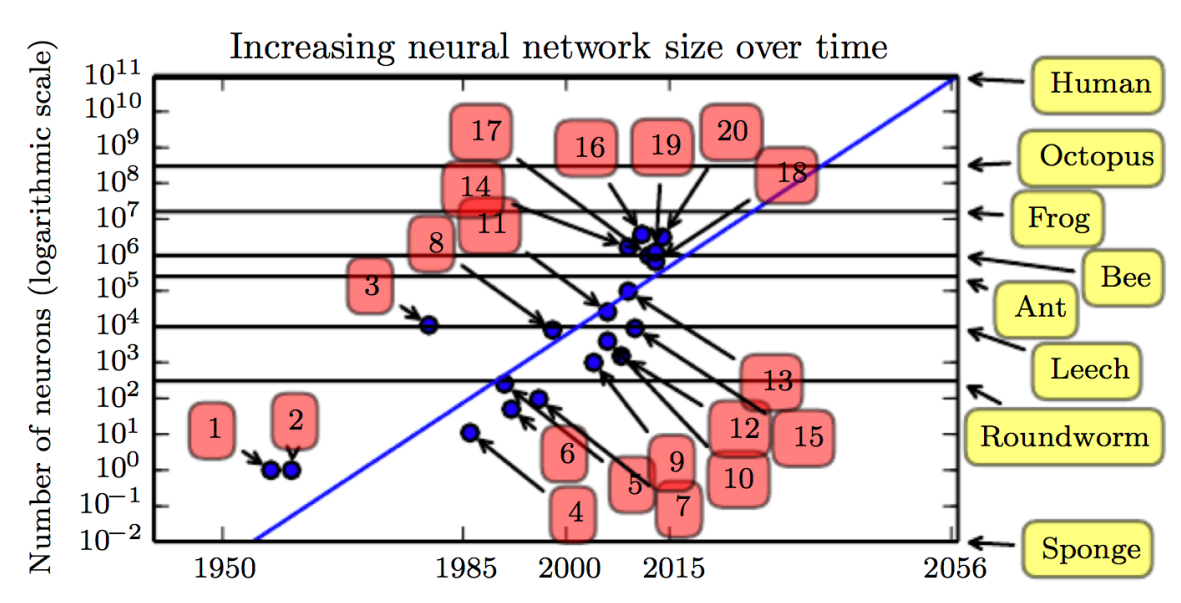
- is **good old neural networks** (NN)...

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



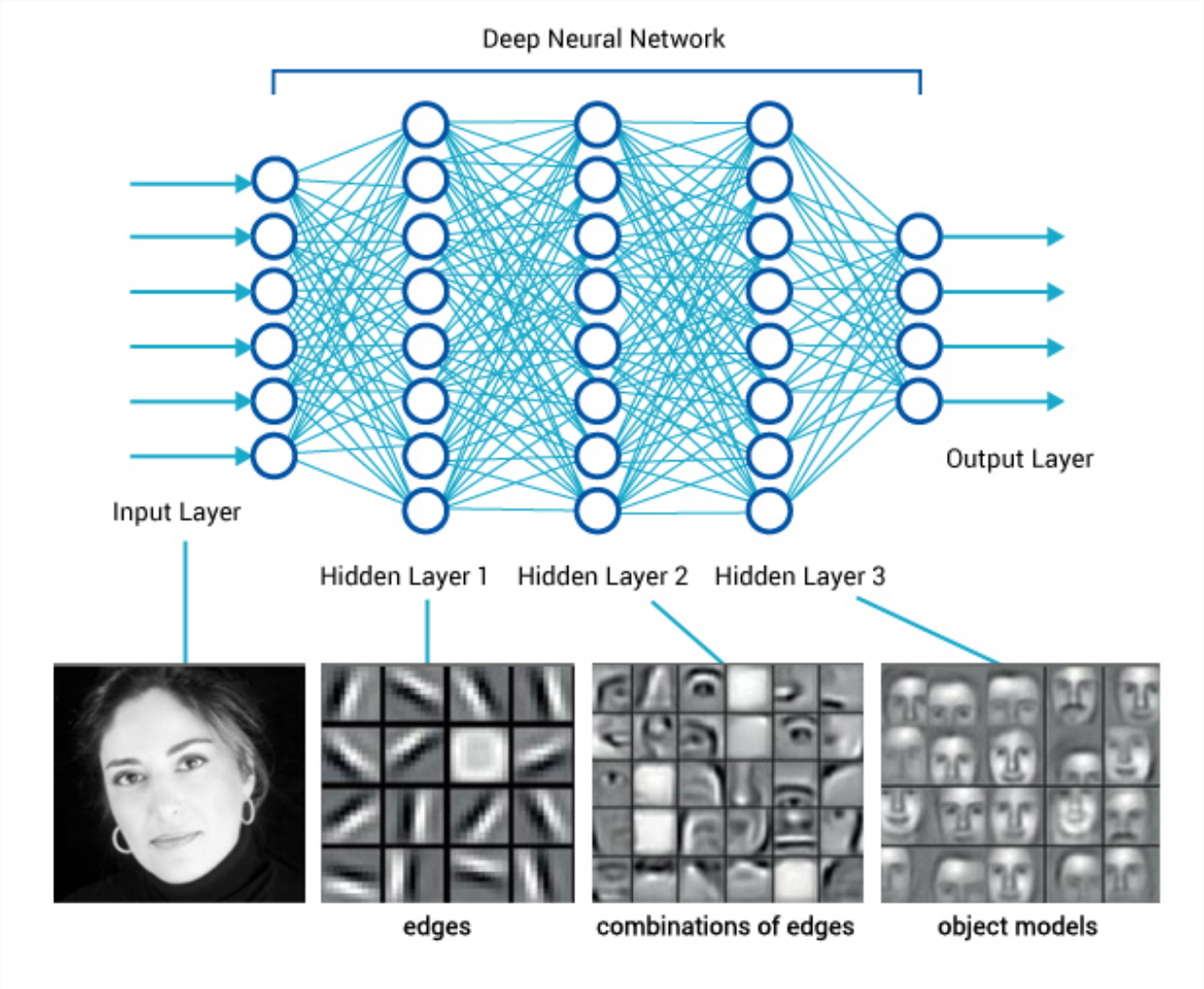
[from *Deep Learning*, Goodfellow, Bengio and Courville]

- 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!

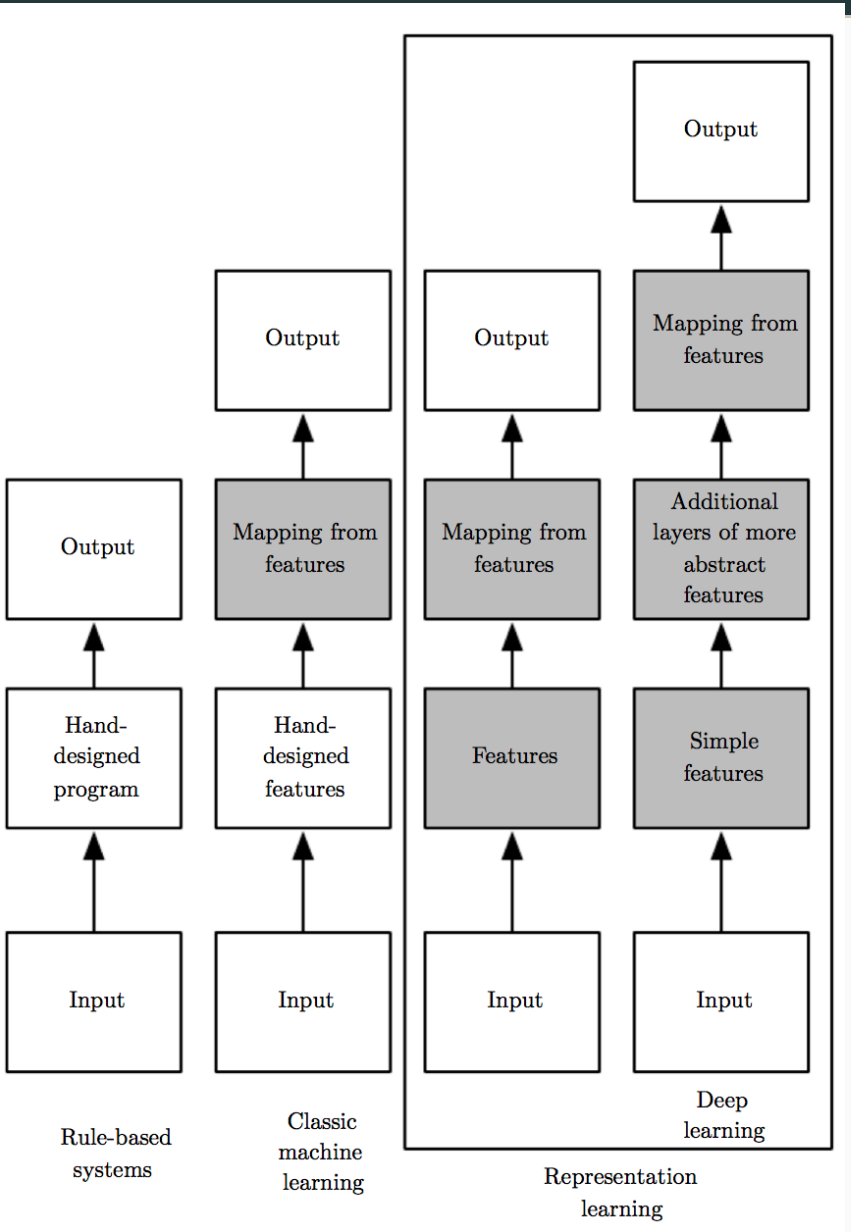
Representation learning i

- 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

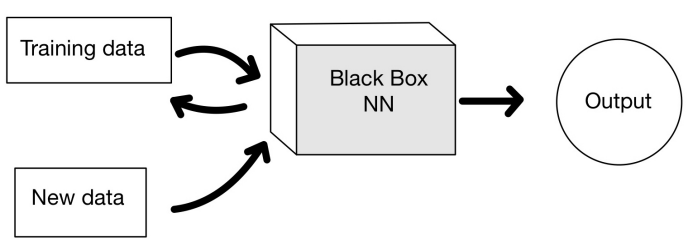


Grey rectangles are machine-learned

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

Outline Day 3

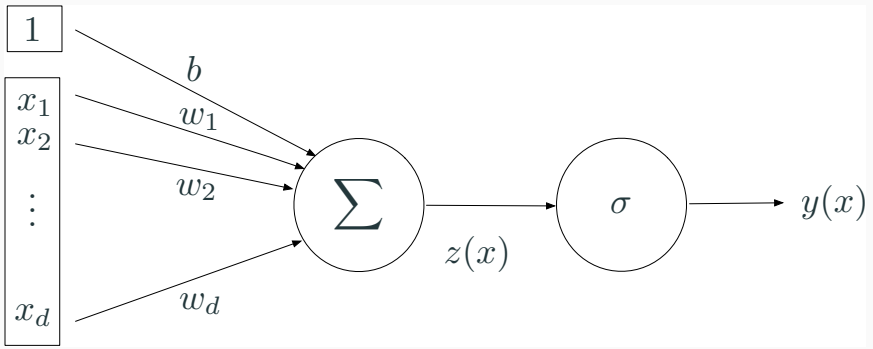
Interface of the neural network-black box



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}

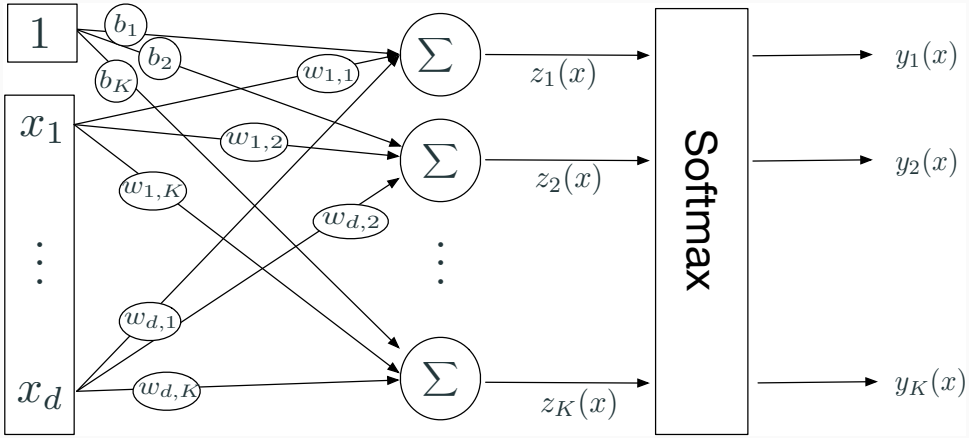
Artificial neuron



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

Network layer

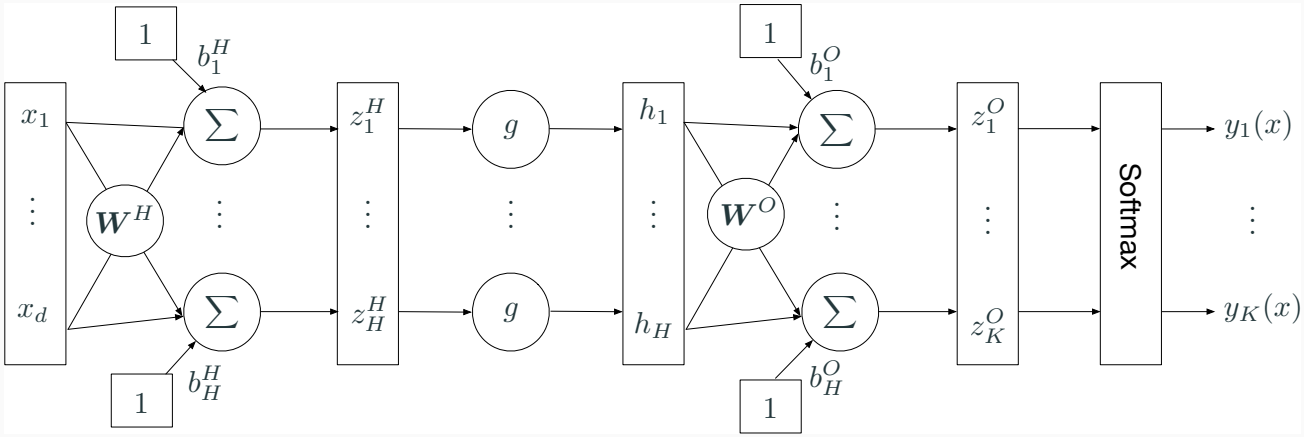


Every layer

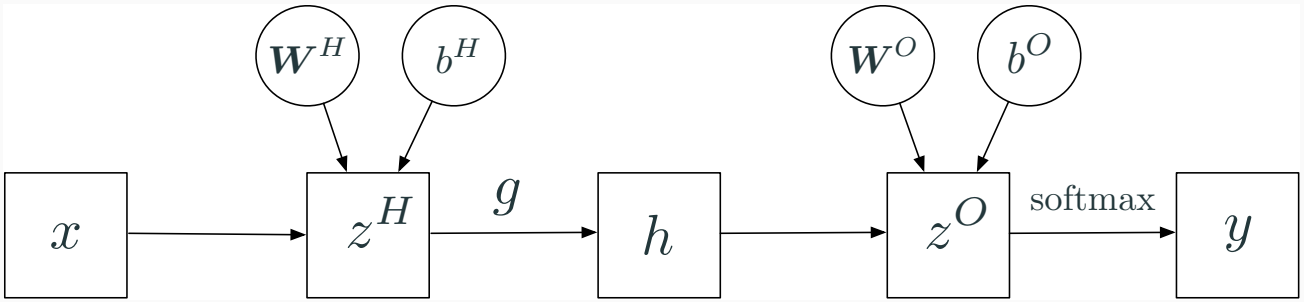
- is composed of **multiple neurons**
- applied to the same input vector
- the layer's output is the vector with
  - the values produced by the neurons or
  - the result of a function applied to those values
- the number of neurons is referred to the **width** of the layer
- A layer where **all** input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

Neural network i

Softmax regression with one hidden layer

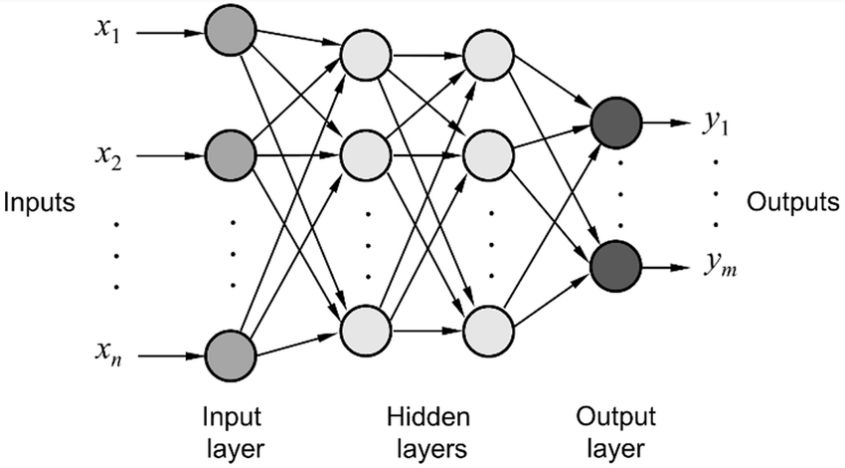


Alternative representation



Neural network ii

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**

Mathematical description of a NN i

Chain-based structure

$$
\begin{aligned}
h^{(1)} &= f^{(1)}(x) = g^{(1)}\left(\boldsymbol{W}^{(1)\top} x + \boldsymbol{b}^{(1)}\right) \\
h^{(2)} &= f^{(2)}\left(h^{(1)}\right) = g^{(2)}\left(\boldsymbol{W}^{(2)\top} h^{(1)} + \boldsymbol{b}^{(2)}\right) \\
&\vdots \\
h^{(L)} &= f^{(L)}\left(h^{(L-1)}\right) = g^{(L)}\left(\boldsymbol{W}^{(L)\top} h^{(L-1)} + \boldsymbol{b}^{(L)}\right) \\
y &= g^{(O)}\left(\boldsymbol{W}^{(O)\top} h^{(L)} + \boldsymbol{b}^{(O)}\right)
\end{aligned}
$$

- 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

Mathematical description of a NN ii

Function approximation

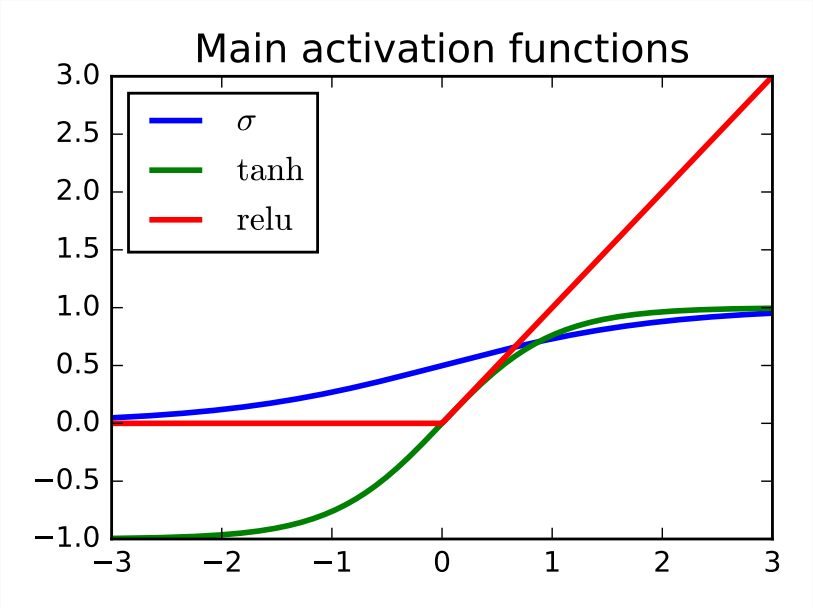
- The **universal approximation theorem** states that a basic NN is a universal approximator
- Roughly, if there is no limit on the widths of the layers, for any function f^*, there exists a NN f that approximates f^* with desired accuracy
- This also holds for shallow networks when the number of hidden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

Activation functions i

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



Activation functions ii

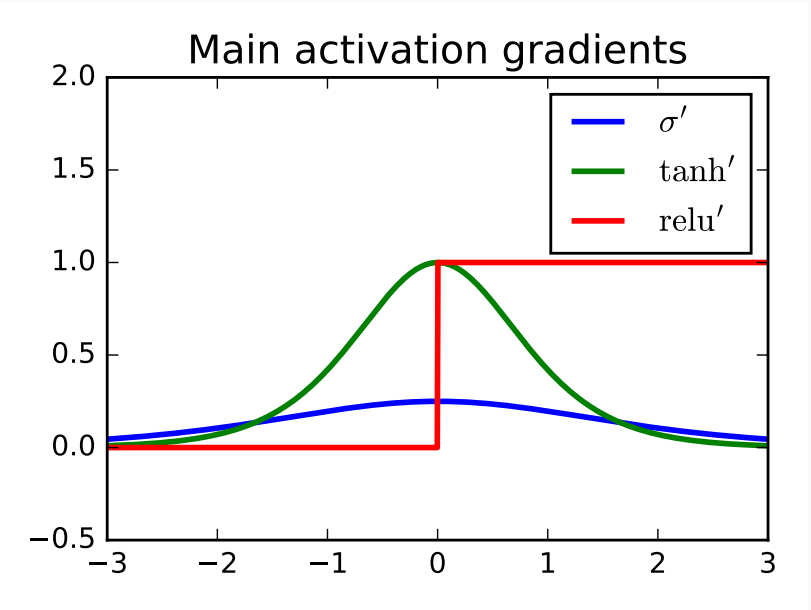
ReLu activation

- 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

Activation functions iii

Derivatives of activation functions

- **sigmoid**:
  $\sigma'(z) = \sigma(z)(1 - \sigma(z))$
- **tanh**:
  $\tanh'(z) = 1 - \tanh^2(z)$
- **ReLu**: $\text{ReLu}'(z) = \mathbf{1}_{z>0}$



Feed-forward neural network (FNN) i

Feed-forward neural network

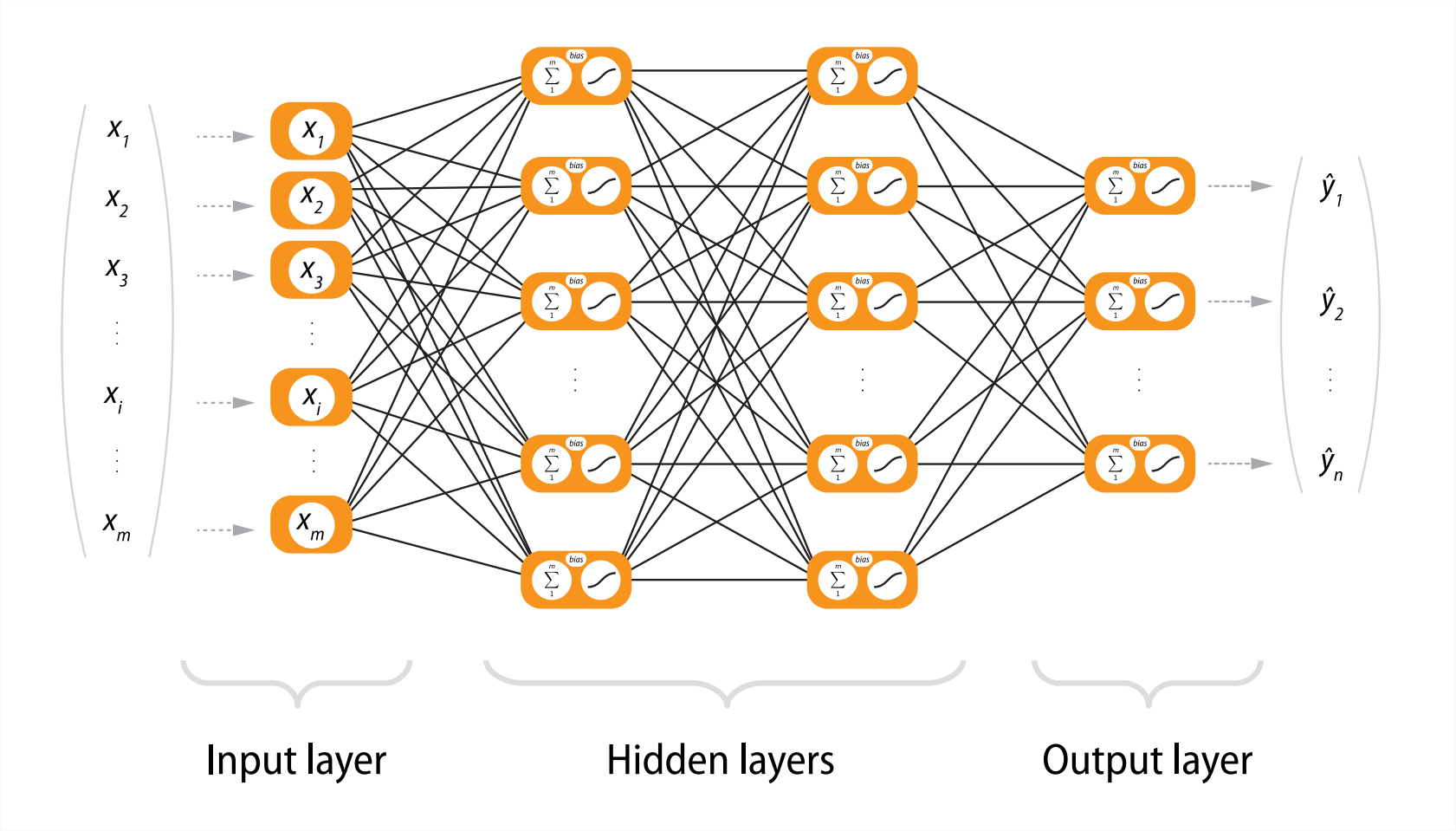
- was the first and simplest type of neural network devised
- **information moves in only one direction** - forward - from the input nodes through the hidden nodes (if any) and to the output nodes
- no cycles or loops in the network

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...

Architecture of a FFNN



[From FIDLE `https://fidle.cnrs.fr`]

Main categories of deep neural networks i

Three major categories of neural network architectures

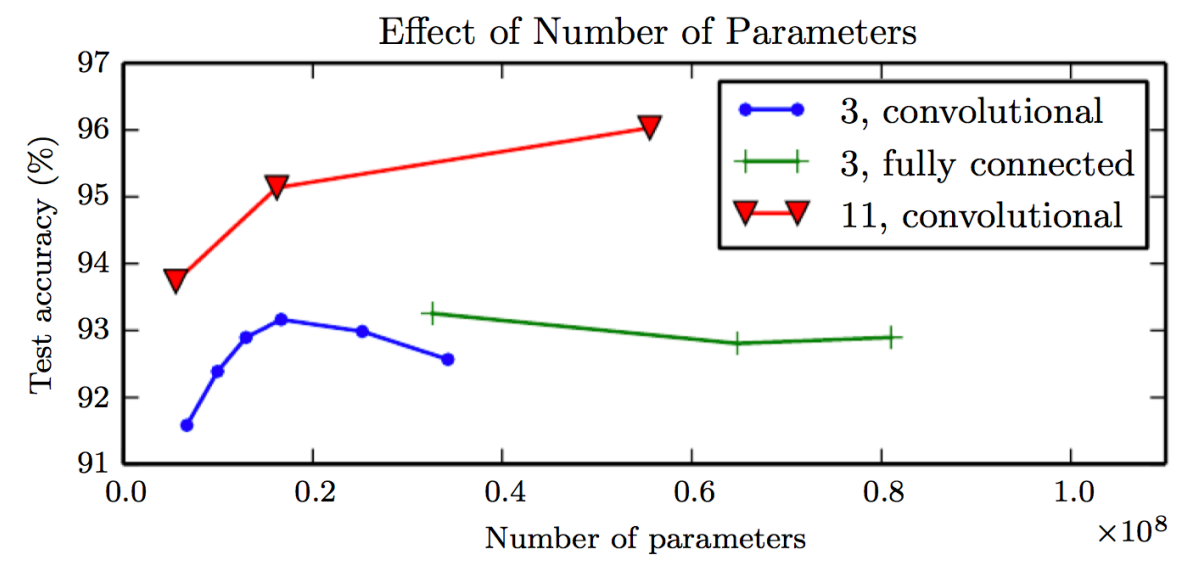
- Feed-forward neural networks (FNN)
  - classical neural networks
- Convolutional neural networks (CNN)
  - specifically designed for image recognition and processing
- Recurrent neural networks (RNN)
  - for temporal data, handwriting recognition, speech recognition

CNN and RNN

- not fully connected
- less parameters
- better exploitation of the data structure

Main categories of deep neural networks ii

Performance of different architectures



[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

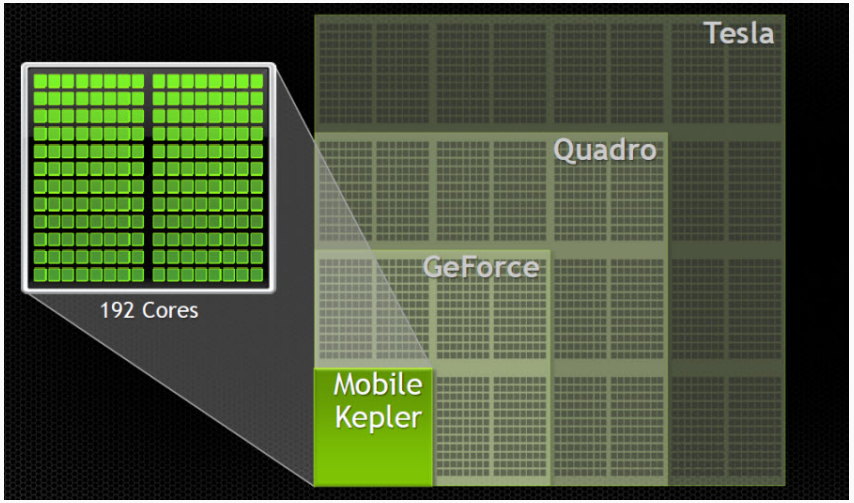
Training neural networks i

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



Training neural networks ii

Our goal today

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

Outline Day 3

Training of the network i

Objective function

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

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

where

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

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

Biases are not penalized

Training of the network ii

Example of an objective function

- Consider the one-hidden layer FNN

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

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

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

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

Training of the network iii

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 $\nabla_{\theta} L(\theta)$

Forward propagation i

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 :

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

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

Forward propagation ii

When is forward propagation performed?

- In the learning algorithm
  - At every iteration, for the **evaluation of the loss** $\ell(y_i, f(x_i; \boldsymbol{\theta}))$ for observations x_i and current parameter value $\boldsymbol{\theta}$
  - 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** $y^{\text{pred}} = f(x^{\text{new}}; \hat{\boldsymbol{\theta}})$ for a new datapoint x^{new} with trained parameter $\hat{\boldsymbol{\theta}}$

Outline Day 3

Backpropagation i

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 L
- This involves the computation of gradients of $f(x, \theta) = f^{(L)} \circ f^{(L-1)} \circ ... \circ f^{(1)}(x)$
- Backpropagation computes

$$\nabla_\theta f(x, \theta)$$
$$= f'^{(L)}(h^{(L-1)}(x)) \times f'^{(L-1)}(h^{(L-2)}(x)) \times ... \times f'^{(2)}(h^{(1)}(x)) \times f'^{(1)}(x)$$

Backpropagation ii

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 $\boldsymbol{\theta}$:

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

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

Outline Day 3

Stochastic gradient descent i

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 $m \in \{32, 64, \ldots, 256\}$
- Draw at random a subset $B \subset \{1, \ldots, n\}$ of indices of size $|B| = m$
- The **mini-batch gradient** is given by

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

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

Stochastic gradient descent ii

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

Stochastic gradient descent iii

Algorithm 1 SGD for training a neural network

Require: initial parameter $\boldsymbol{\theta}$, 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

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

4: Apply the update

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda_t \nabla_{\boldsymbol{\theta}} \boldsymbol{L}$$

5: **end while**

Stochastic gradient descent iv

Learning rates

- Learning rates have an impact on the **convergence** of the algorithm
- $(\lambda_t)_t$ 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

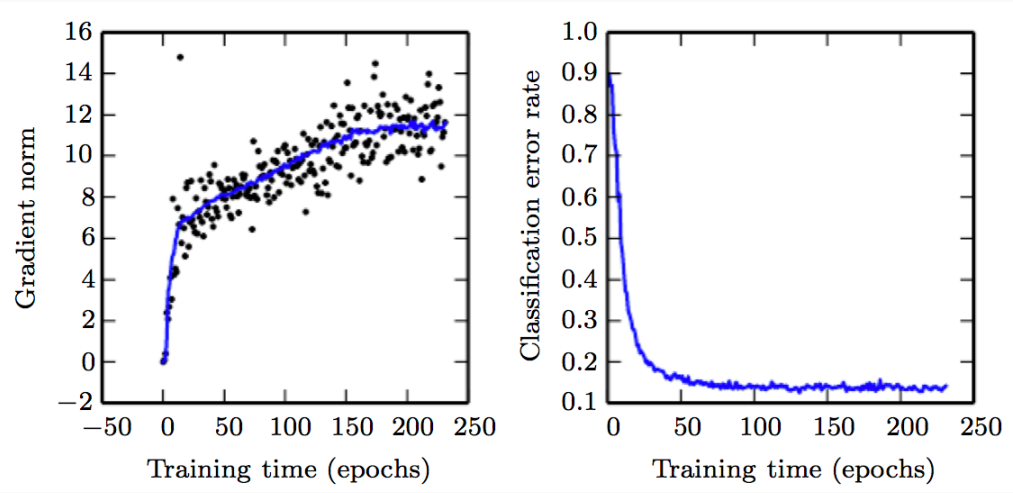
Popular implementations of SGD

- **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 i

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



Non convex optimisation ii

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!**

Non convex optimisation iii

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
  $\hookrightarrow$ What is the **parctical 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

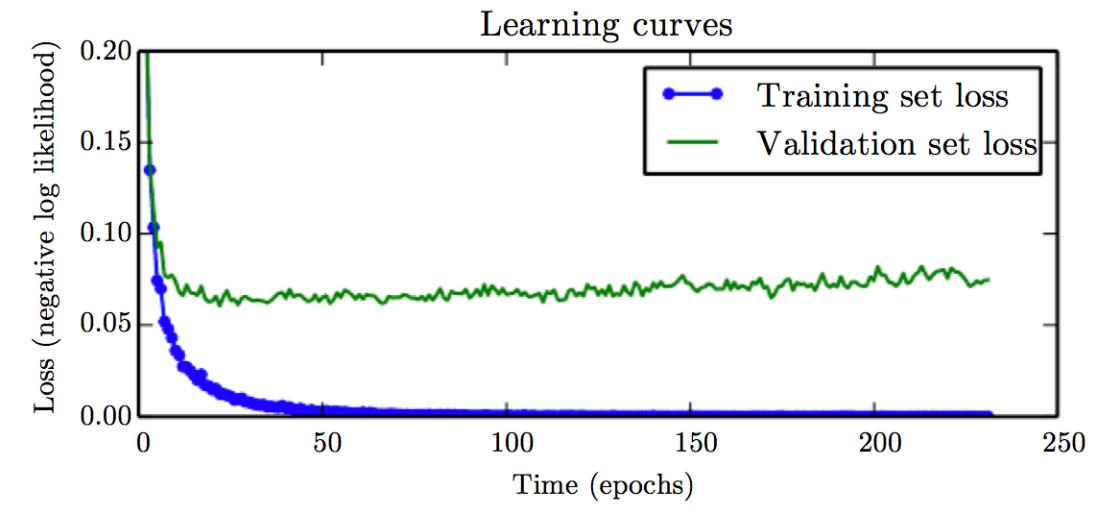
This is not the case anymore

- 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**

Regularization for deep learning i

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]

Outline Day 3

Regularization via early stopping

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

Outline Day 3

Regularization via dropout i

Ensemble learning

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

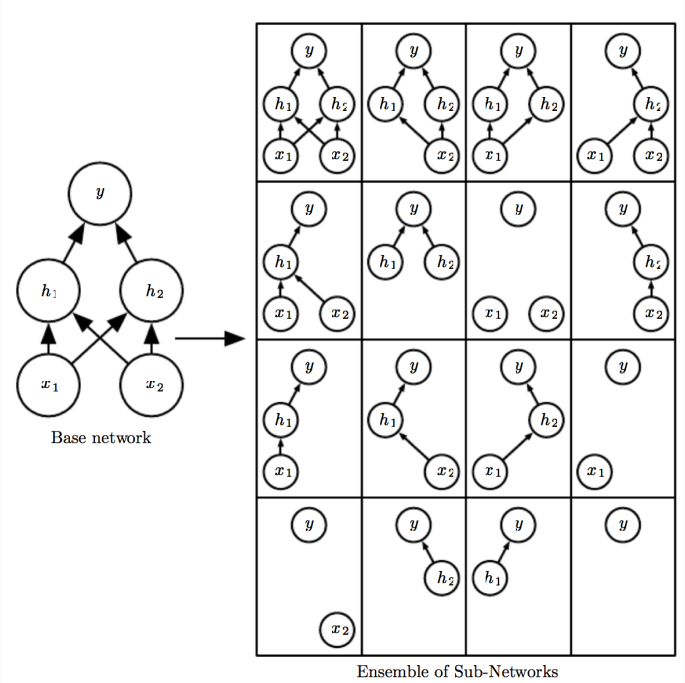
Regularization via dropout ii

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)

Regularization via dropout iii

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

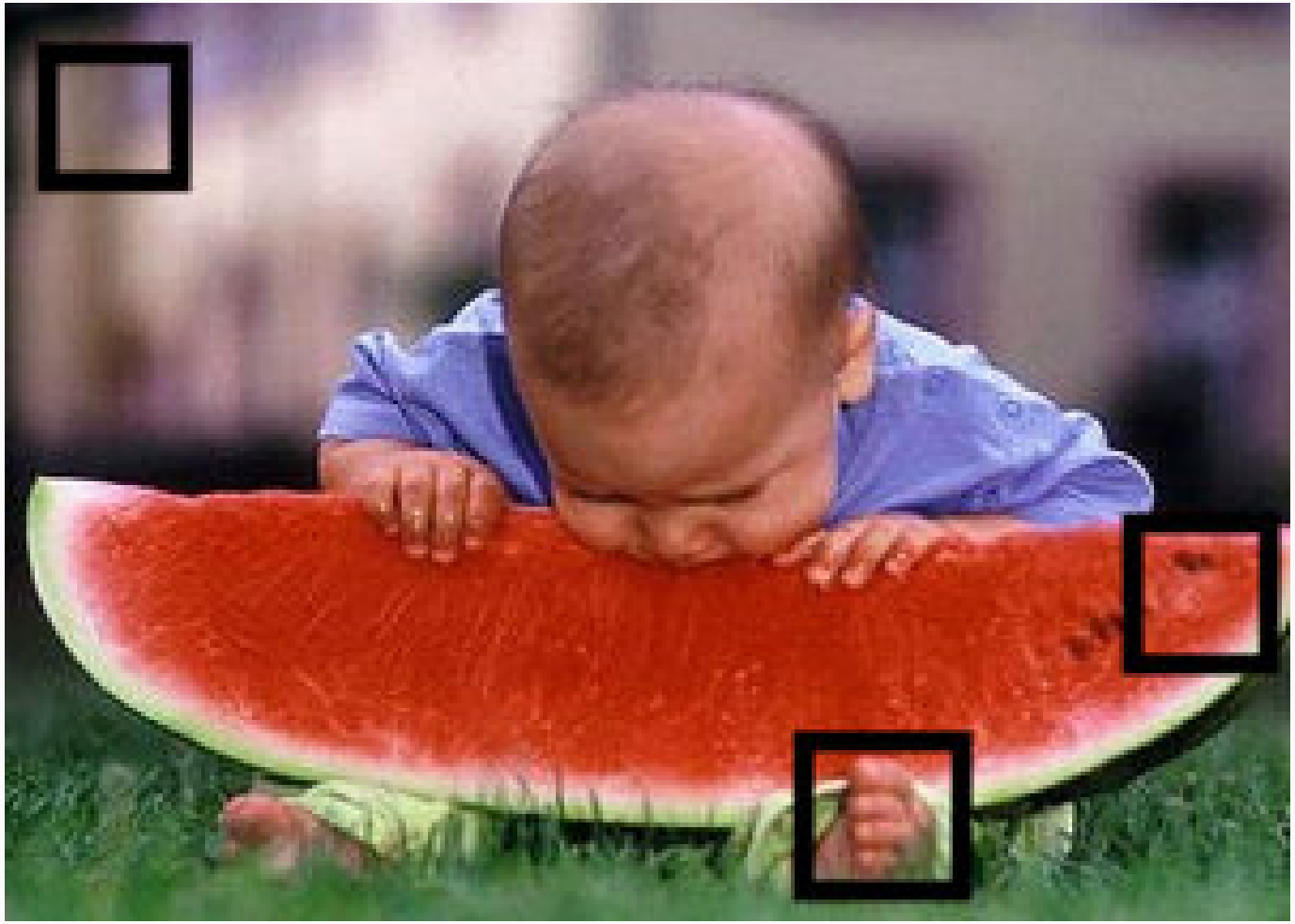
Outline Day 3

Playground time !!!

`http://playground.tensorflow.org`

Convolutional neural networks i

Convolutional neural networks ii



Convolutional neural networks iii

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

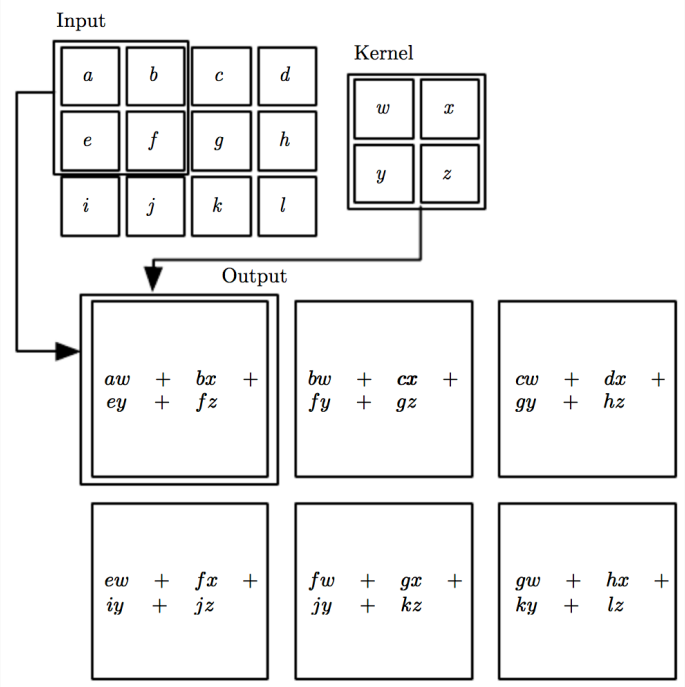
Convolutional neural networks

- 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 $f(x) = \boldsymbol{W}x$ with **constraint weight matrix $\boldsymbol{W}$**
  - Many zero values in $\boldsymbol{W}$: 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

Convolutional layers i

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

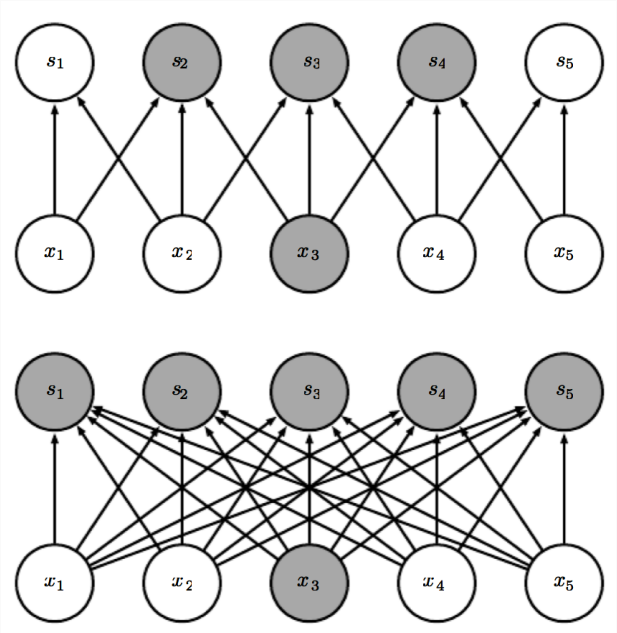
↪ **improves statistical efficiency**

↪ **improves memory**

↪ **faster computations**

Convolutional layers ii

Sparse connectivity

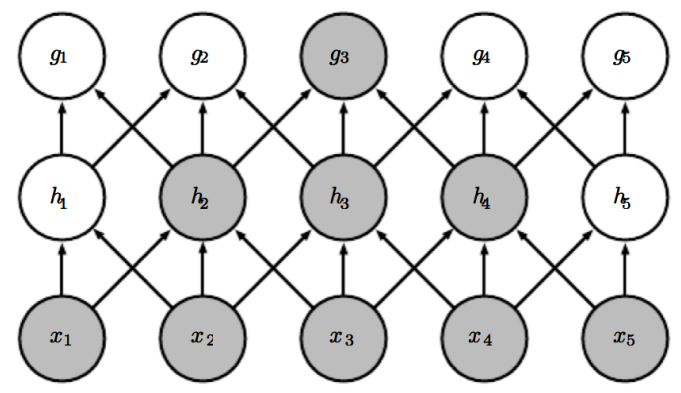


[from *Deep Learning*, Goodfellow, Bengio and Courville]

- **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

Convolutional layers iii

Multiple convolutional layers

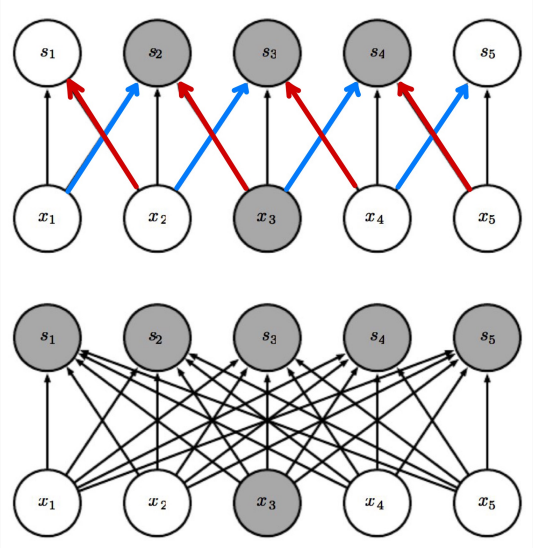


[from *Deep Learning*, Goodfellow, Bengio and Courville]

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

Convolutional layers iv

Parameter sharing



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

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

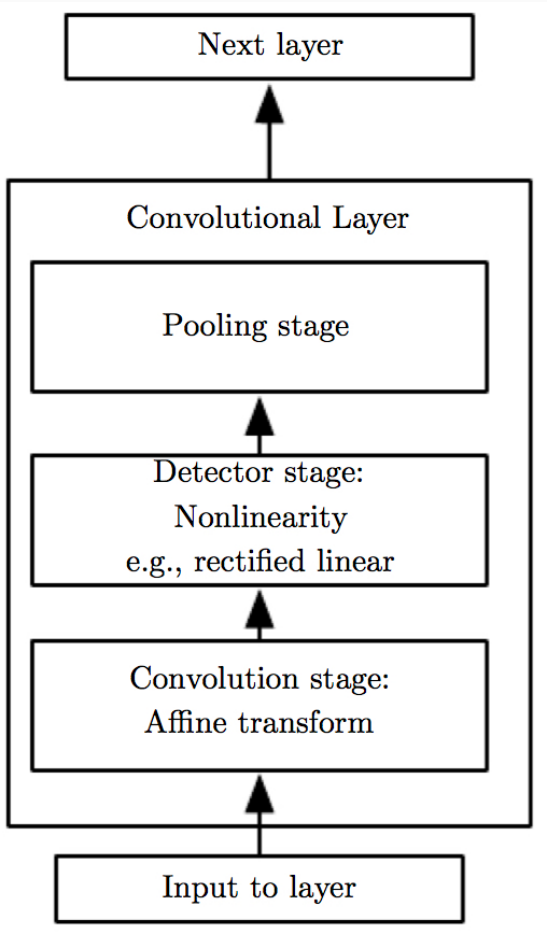
Outline Day 3

Pooling i

A CNN layer

- typically includes 3 operations:

convolution + activation + pooling



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

Pooling ii

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

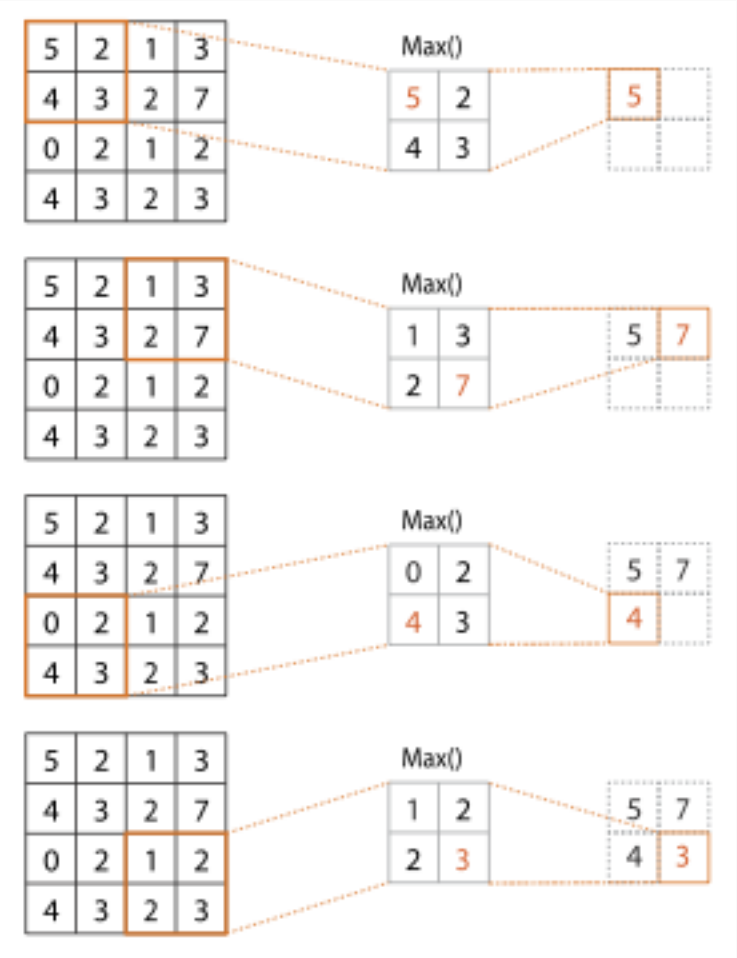
Pooling iii

Pooling

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

MaxPooling

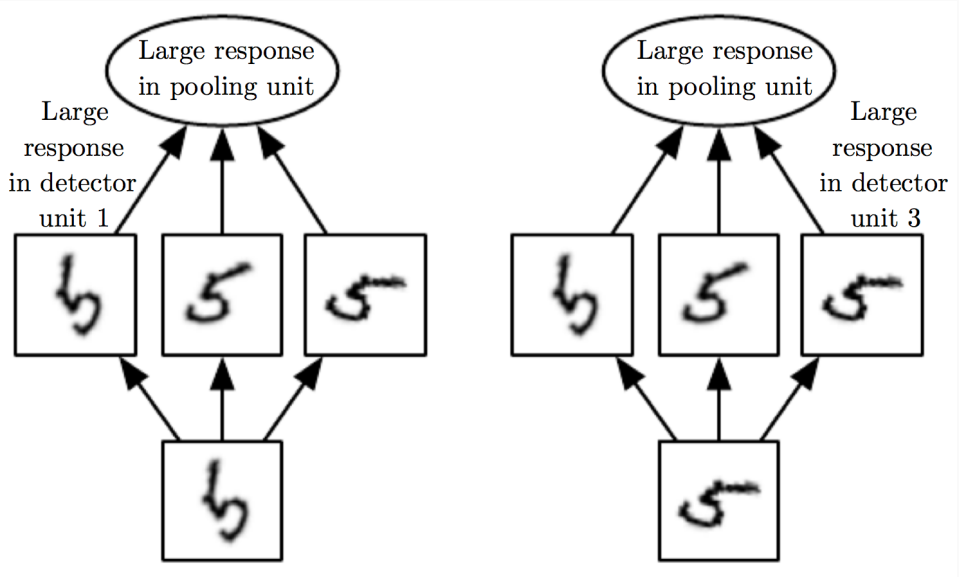
- most widely used is **max aggregation**, also called max-pooling



[From FIDLE `https://fidle.cnrs.fr`]

Pooling iv

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**

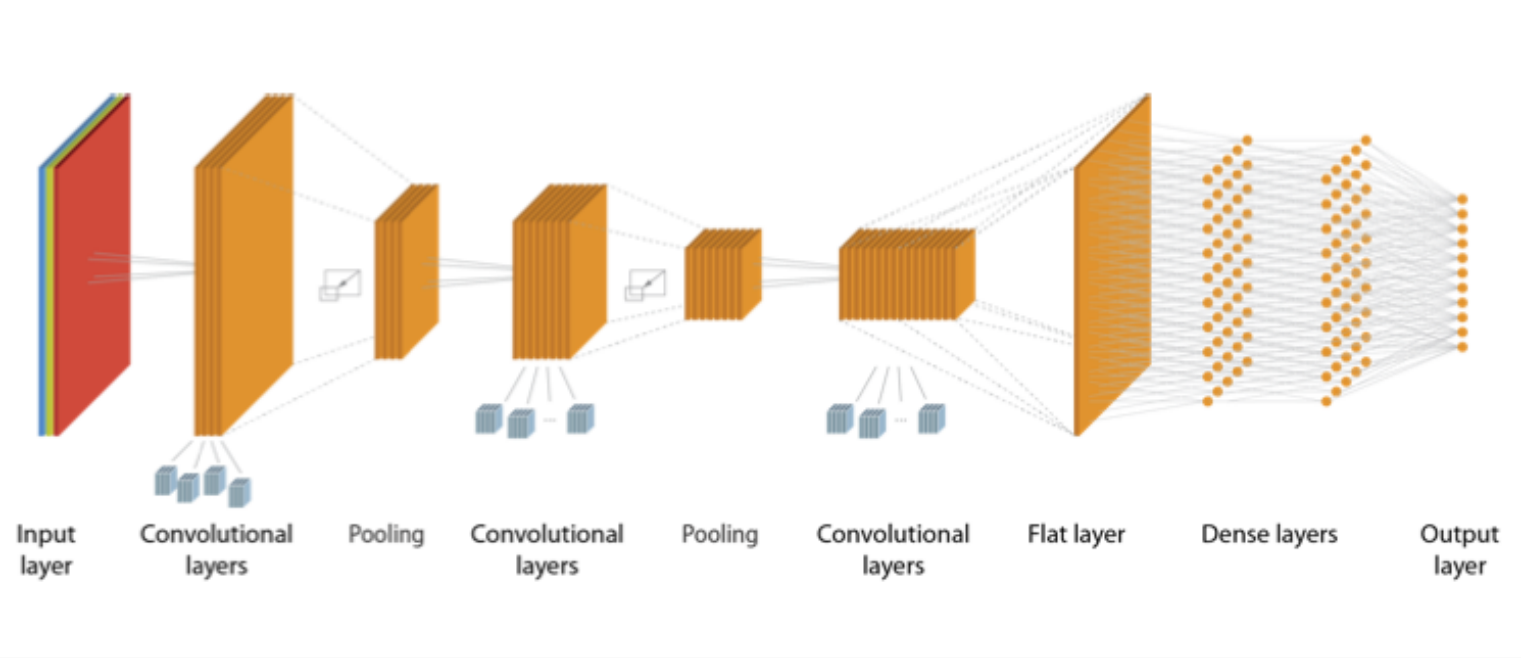
Convolutional neural networks i

Many tricks and variants of convolutional layers

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

Convolutional neural networks ii

CNN design



[From FIDLE `https://fidle.cnrs.fr`]

Outline Day 3

Recurrent Neural Networks i

Goal

- analyse **sequential data** $\mathbf{x}_i = (x_i^{(1)}, x_i^{(2)}, ..., x_i^{(T)})$
- Applications: time series, text, language, genomic data

Problems with a traditional dense feedforward NN

- when processing **long** sequences
- when sequences $\mathbf{x}_i$ have **different lengths**
- Redundancy not considered
  - Example: sentences "I went to London in 2010", "In 2010 I went to London"
  - Goal: extract year
  - FFNN has different parameters for each input $x^{(t)}$, hence has to learn all the rules of the language separately at each position in the sentence

Recurrent Neural Networks ii

Solution

- **Parameter sharing**

1D CNN

- allows parameter sharing across time
- but is shallow, no long-term dependencies
- *each member of the output is a function of a small number of neighboring members of the input*

Recurrent Neural Networks iii

RNN

- parameter sharing in a different way
- *each member of the output is a function of previous members of the output*
- assume that passing from state $s^{(t-1)}$ to $s^{(t)}$ follows the same mechanism as passing from $s^{(t)}$ to $s^{(t+1)}$
- for each output apply the same function on the previous outputs
- consequence: share parameters through a deep computational graph
- include **cycles** that represent the influence of the present value of a variable on its own value at a future time step

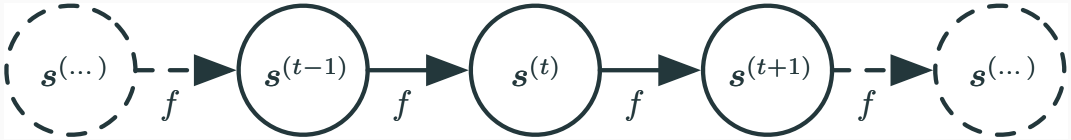
Recurrent units i

Simple dynamical system

Evolution of the state $\mathbf{s}^{(t)}$ of a system:

$$\begin{aligned}\mathbf{s}^{(t)} &= f(\mathbf{s}^{(t-1)};\theta), \qquad t = 1, 2, ... \\ &= f(f(\mathbf{s}^{(t-2)};\theta);\theta) \\ &= f(f(f(\mathbf{s}^{(t-3)};\theta);\theta);\theta)\end{aligned}$$

Same parameter θ at each step



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

Recurrent units ii

Dynamical system with an external signal $\mathbf{x}^{(t)}$

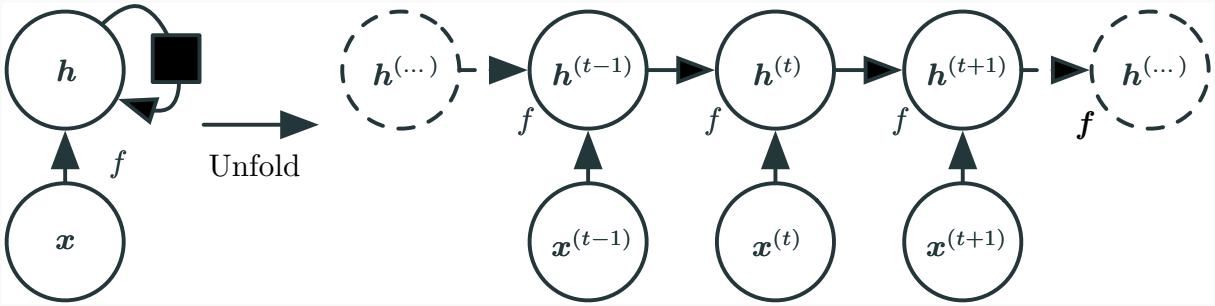
$$\mathbf{s}^{(t)} = f\left(\mathbf{s}^{(t-1)}, \mathbf{x}^{(t)}; \theta\right)$$

Combine information from the past with information on the present

Recurrent units iii

Hidden units in RNNs

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \theta)$$



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

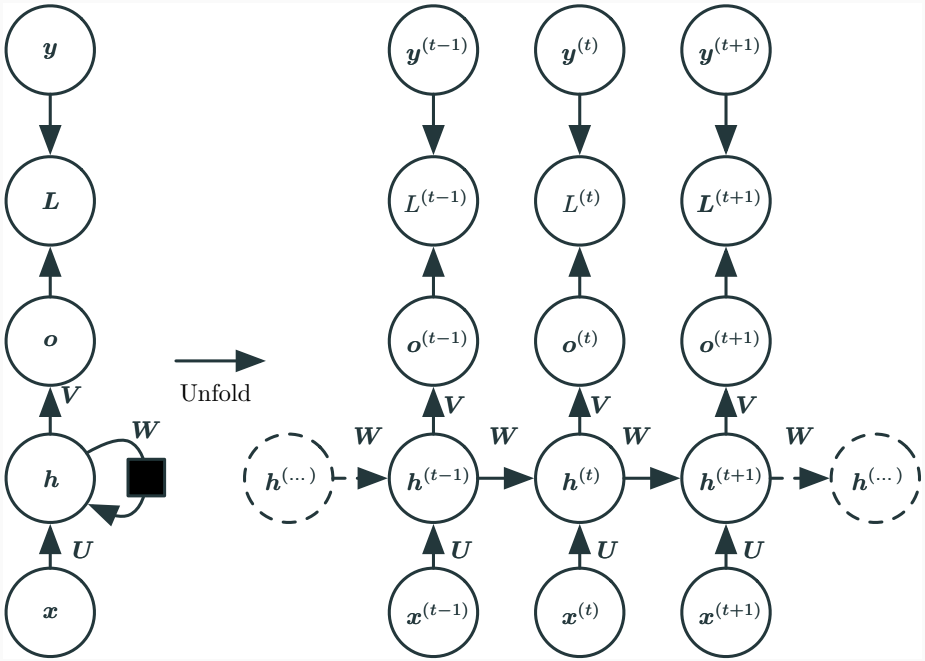
- $h^{(t)}$ is a **summary** of the past or a specific feature present (or not) in the past $x^{(1)}, ..., x^{(t-1)}$
- black square ■ in a circuit diagram indicates that an interaction takes place with a delay of a single time step

Recurrent units iv

Advantages

- **sequences of different lengths** can be processed: the model is specified in terms of *transition from one state to another*, rather than specified in terms of a variable-length history of states
- **parameter sharing** by using the same transition function f with the same parameters at every time step

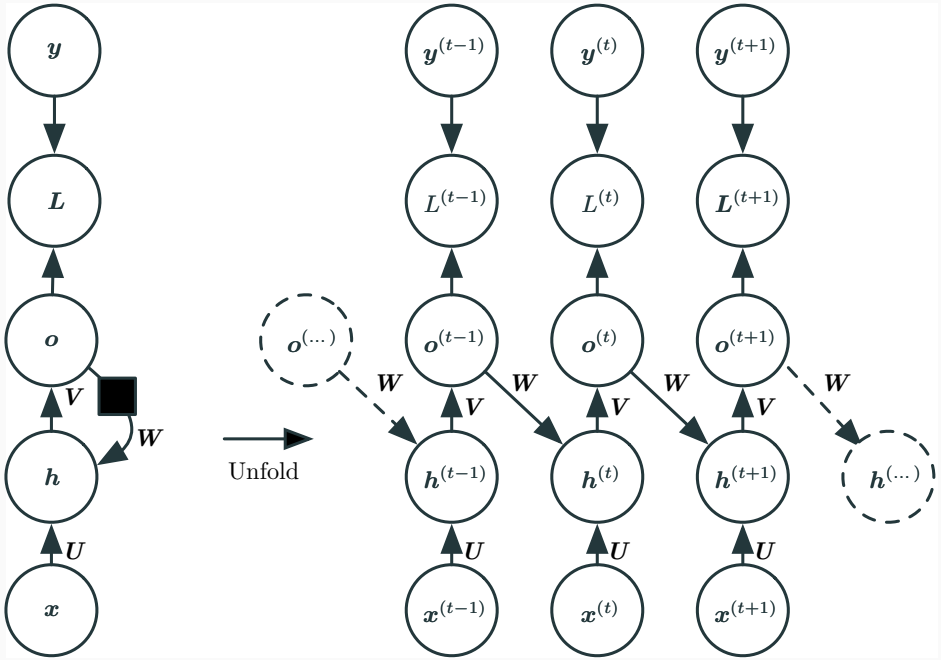
Various design patterns i



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

- produce an output at each time step
- have a recurrent connection between hidden units

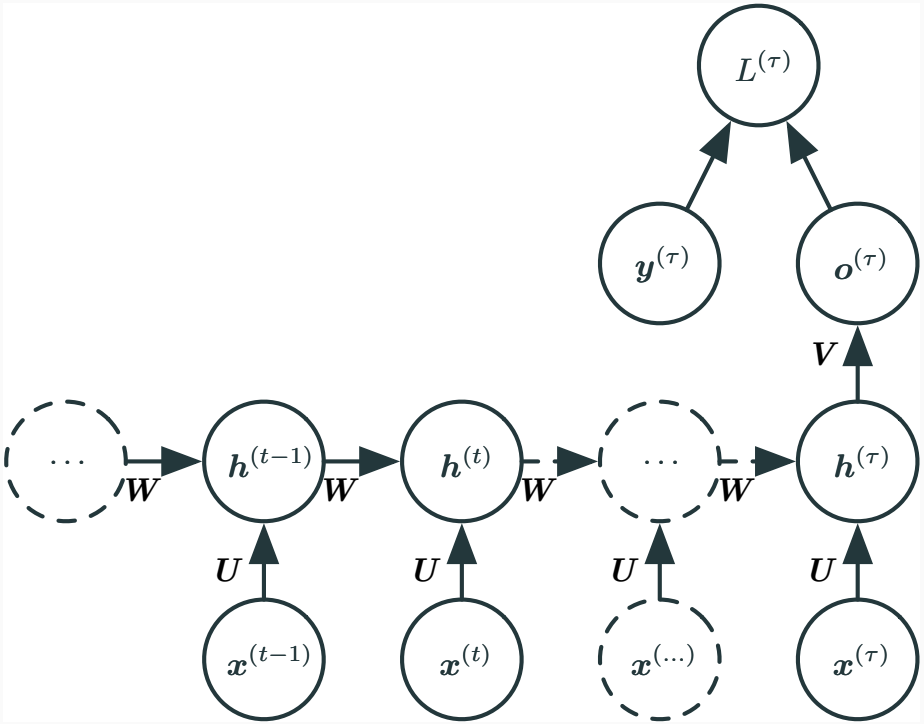
Various design patterns ii



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

- produce an output at each time step
- have recurrent connections only from the output at one time step to the hidden units at the next time step

Various design patterns iii



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

- recurrent connections between hidden units
- produce a single output after reading the entire sequence

Optimization i

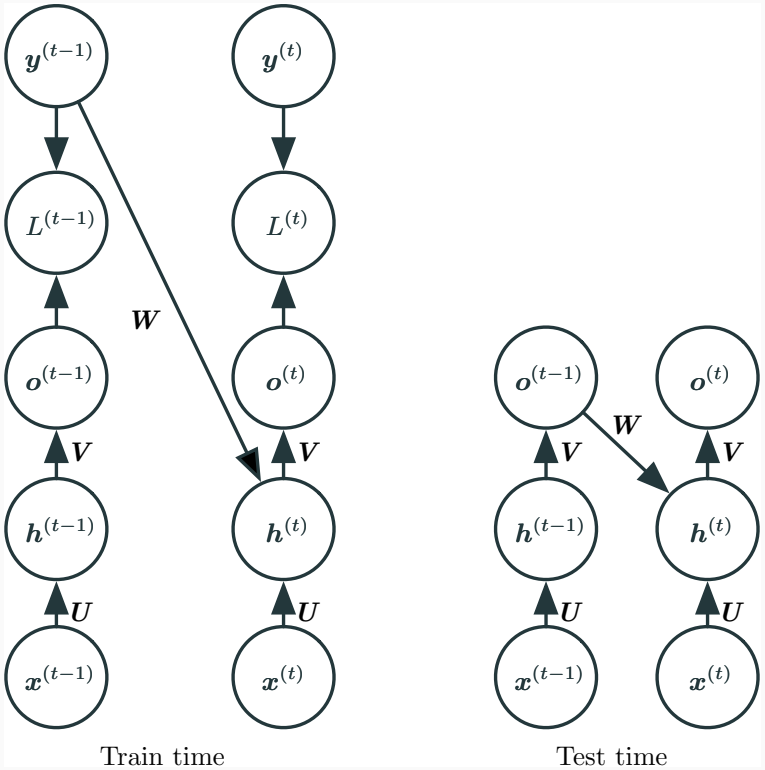
Numerical issues

- depending on the design
- compute gradients of the loss with respect to the parameters can be an expensive operation
- gradients can explode or vanish

Solutions

- gradient clipping
- skip connections
- teacher forcing

Optimization ii



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

Teacher forcing: simplify training by replacing outputs predicted by the model by true values $y^{(t)}$

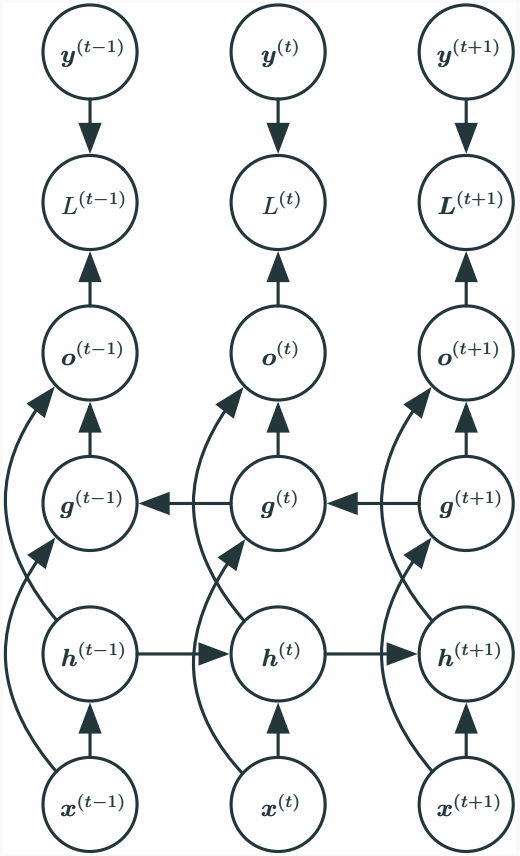
Extensions i

Bidirectional RNNs

- prediction depends on the *whole input sequence*, not only on the past
- speech recognition, handwriting recognition, genomic data
- 2 RNNs: one for each direction

Extensions ii

Bidirectional RNNs

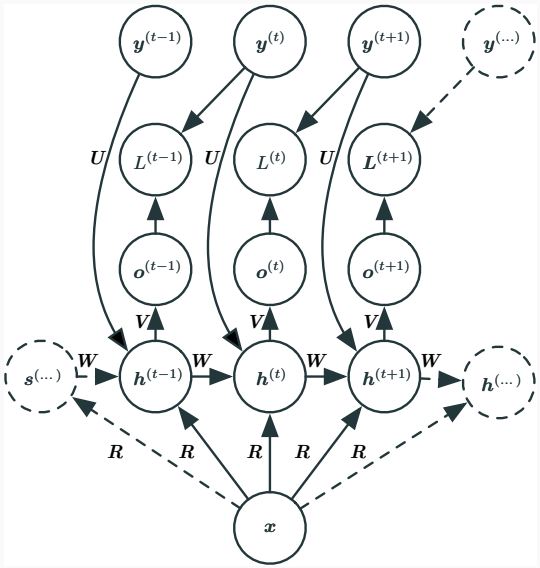


[from *Deep Learning*, Goodfellow, Bengio and Courville]

Extensions iii

Modeling sequences conditioned on context

- produce a sequence (of variable length) from an input with fixed length
- image captioning: observe an image and produce a sequence of words describing the image



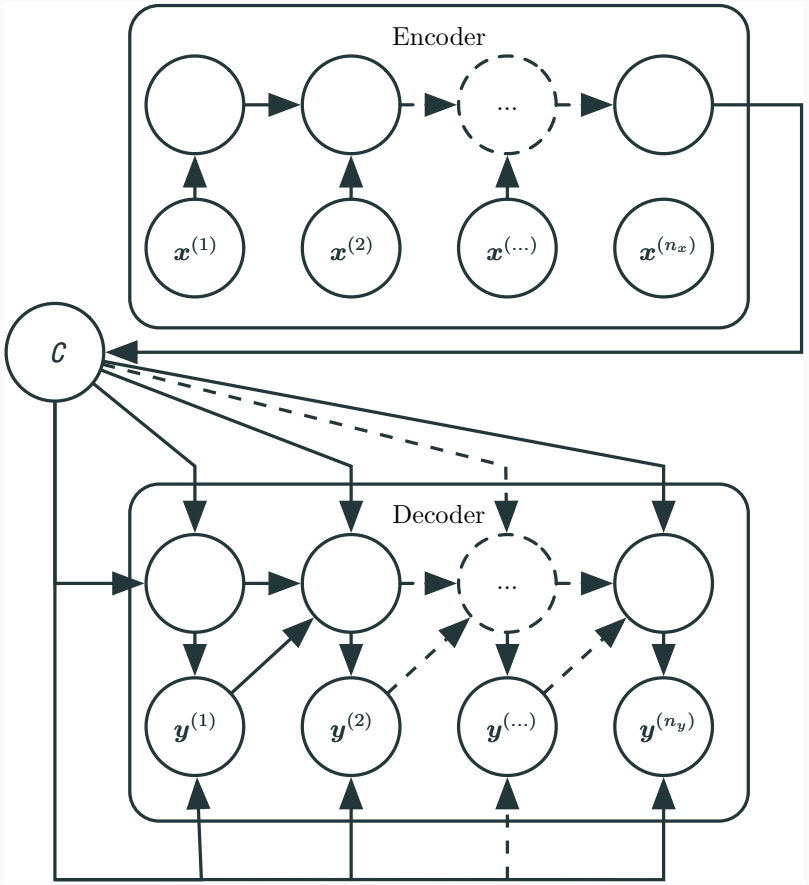
[from *Deep Learning*, Goodfellow, Bengio and Courville]

Extensions iv

Sequence-to-sequence architectures

- Goal: map an input sequence to an *output* sequence which is *not necessarily of the same length*
- machine translation, question answering
- encoder-decoder architecture:
  - encoder: an RNN processes the input sequence and outputs a *context* (of fixed length)
  - decoder: RNN applied to output of the encoder (of fixed length) and produces a sequence of variable length

Encoder-decoder architecture



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

Long-term dependencies i

Issues with RNNs

- does not learn long-term dependencies
- does not operate at multiple time scales

Solutions

- **skip connections** across time to transfer information from the distant past to the present more efficiently
- **leaky units** that integrate signals with different time constants
- remove some connections to model fine-grained time scales

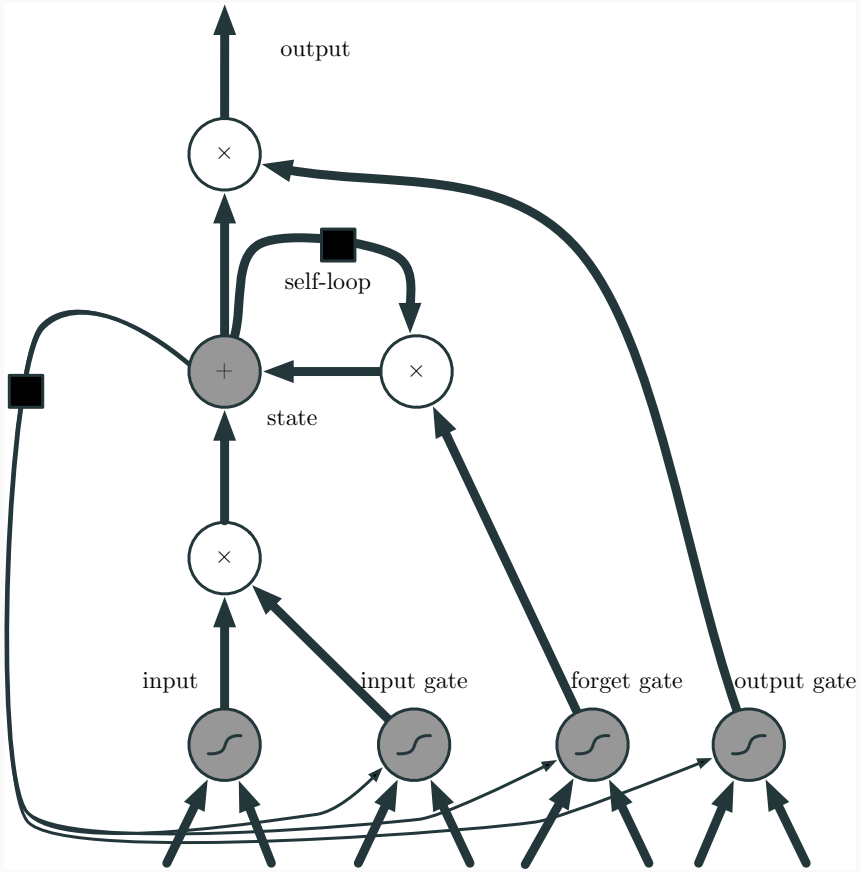
Long-term dependencies ii

LSTM = Long short-term memory

- introduce **forget gates**: the weight of a self-loop is gated, i.e. controlled by another hidden unit
  $\hookrightarrow$ time scale is changed **dynamically**

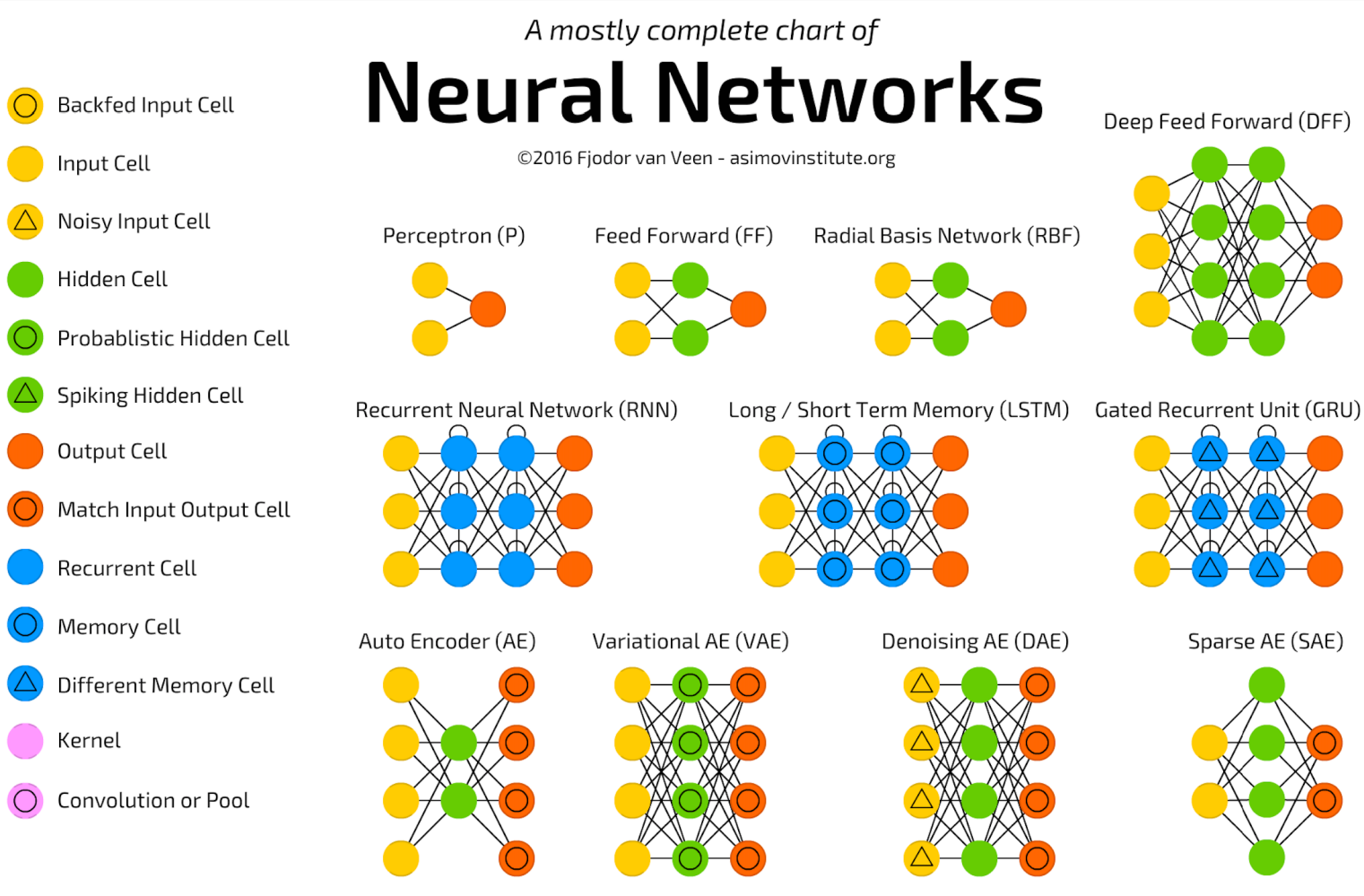
Long-term dependencies iii

LSTM

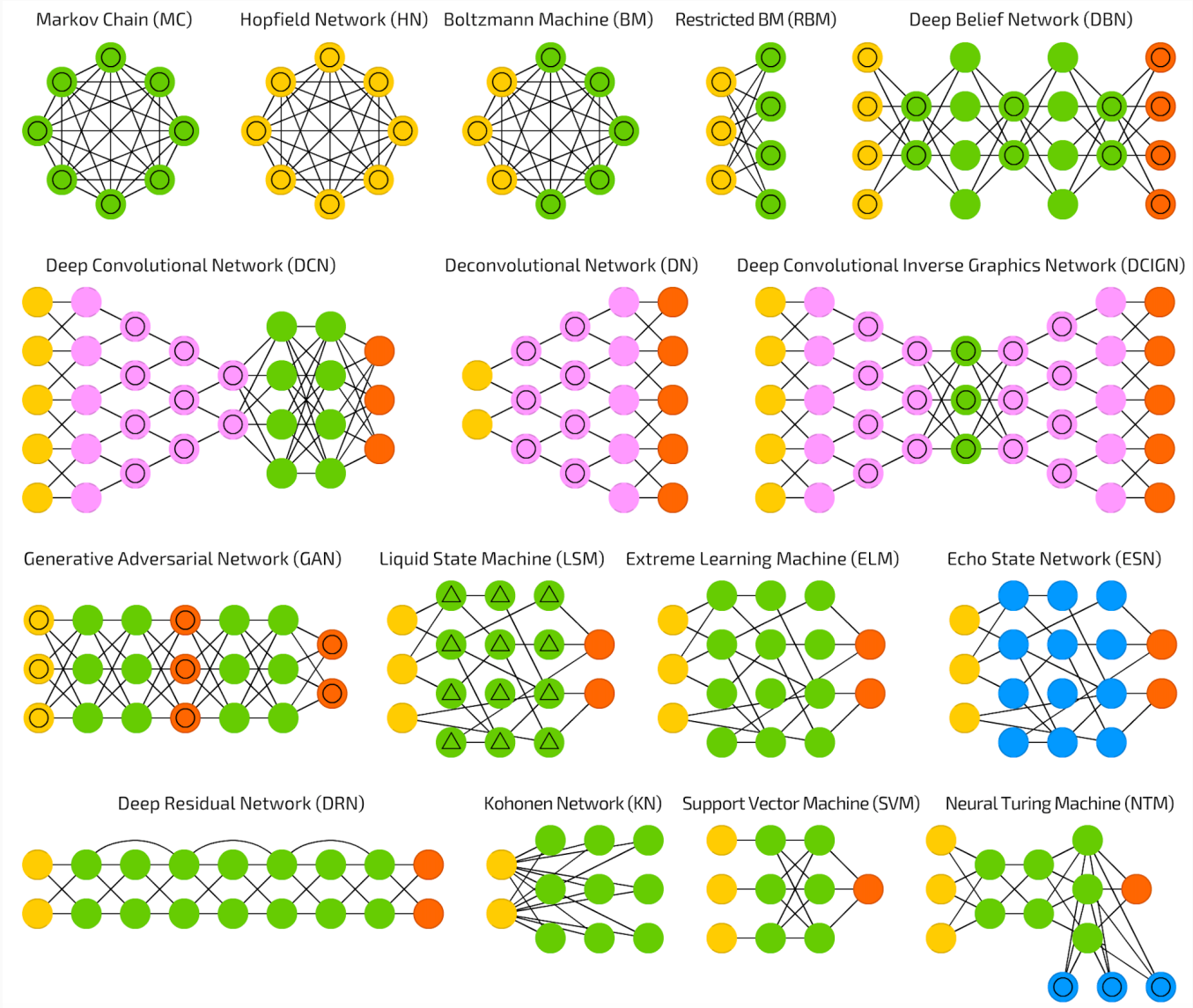


[from *Deep Learning*, Goodfellow, Bengio and Courville]

The jungle of NN i



The jungle of NN ii



Thank you!