Deep learning: an introduction

Hugo Jair Escalante, Eduardo Morales

Contents

Deep learning in a nutshell

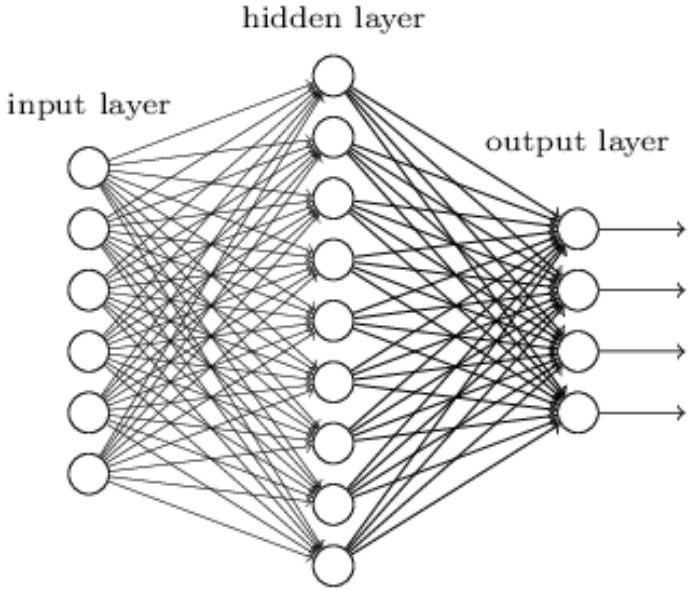
- Deep learning is a machine learning methodology that aims at solving (modeling) problems by building layer-wise models with several (many) levels of increasing abstraction
- Layers of these models capture discriminative/descriptive information from raw data
- Can be used for: supervised/unsupervised learning, reinforcement learning, feature extraction, …
- Examples: multi-layer perceptrons, deep neural networks, convolutional neural networks, deep belief nets, auto encoders, etc.

Deep learning in a nutshell

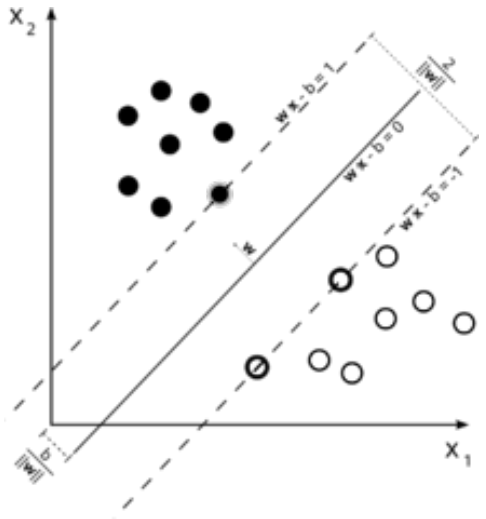
- Features of deep learning methods:
  - Large number of parameters (on the ranges of millions)
  - Require large amounts of data to be trained
  - Can extract features automatically
  - Can leverage unlabeled data
  - Extremely complex models
  - Require of specialized hardware for training them efficiently
  - Dominate the arenas of machine learning applications (e.g., computer vision, NLP)

Deep learning in a nutshell

- How does a non deep model looks like?



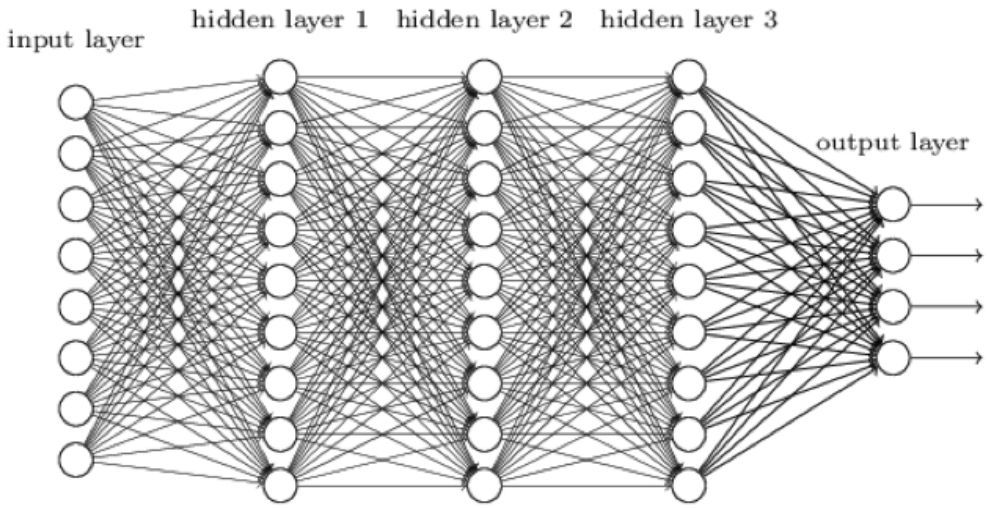
$$f(\mathrm{x}) = w\phi(x) + b$$



$$f(\mathrm{x}) = \sum_{i}^{N} \alpha_i y_i k(\mathrm{x}_i, \mathrm{x}) + b$$

Deep learning in a nutshell

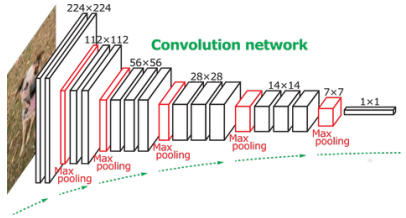
- How does a (not too deep) DL model looks like?

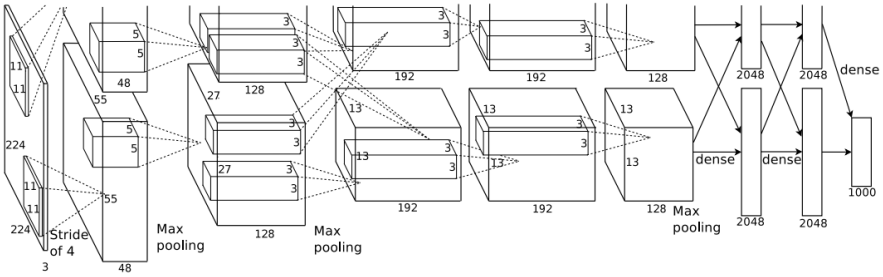


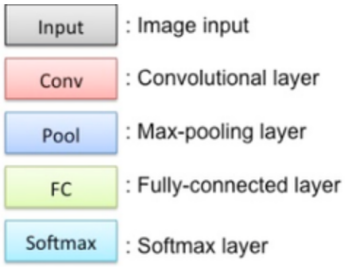
$$f(\mathrm{x}) = W_3\phi_3(W_2\phi_2(W_1\phi_1(X) + b) + b_2) + b_3$$

Deep learning in a nutshell

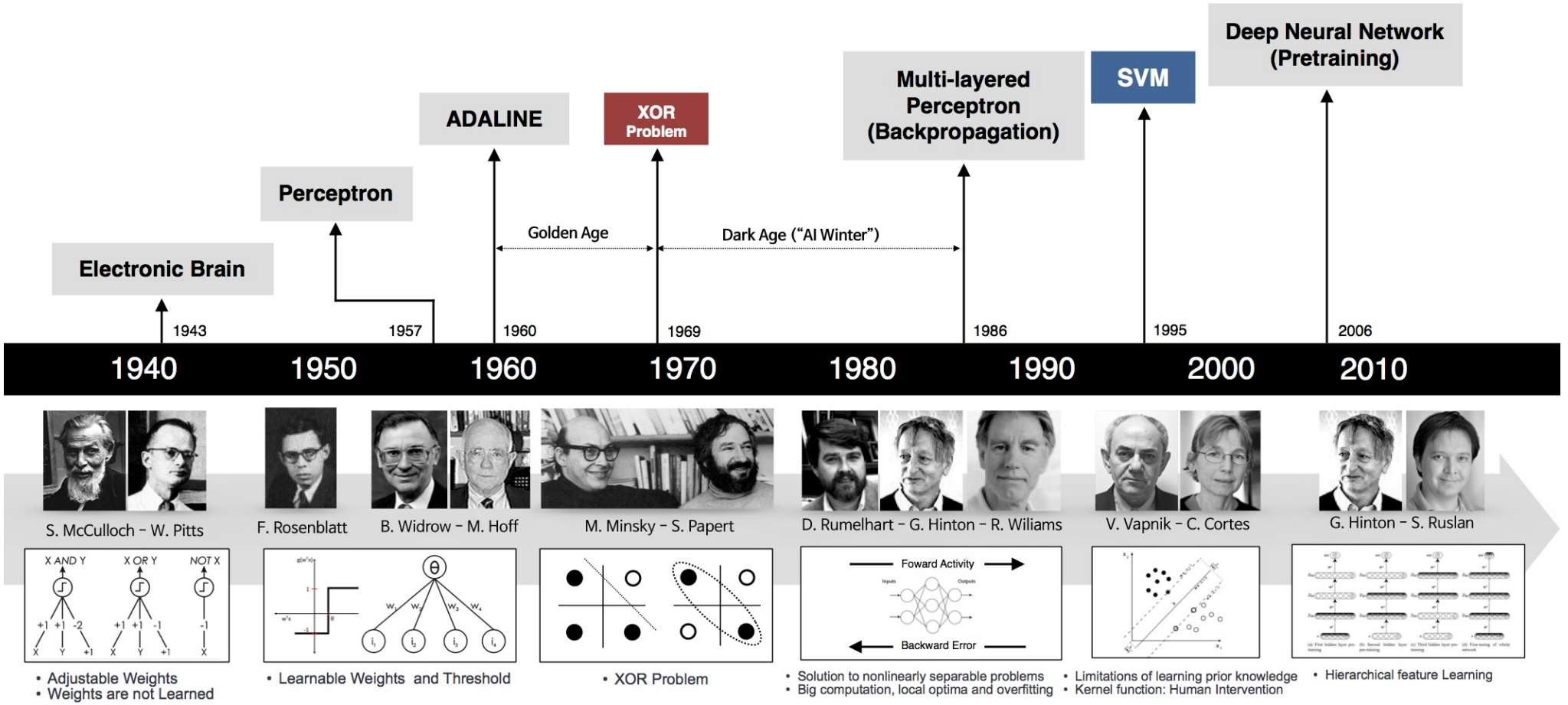
- Going deeper (CNNs)







The boom of DL: brief history



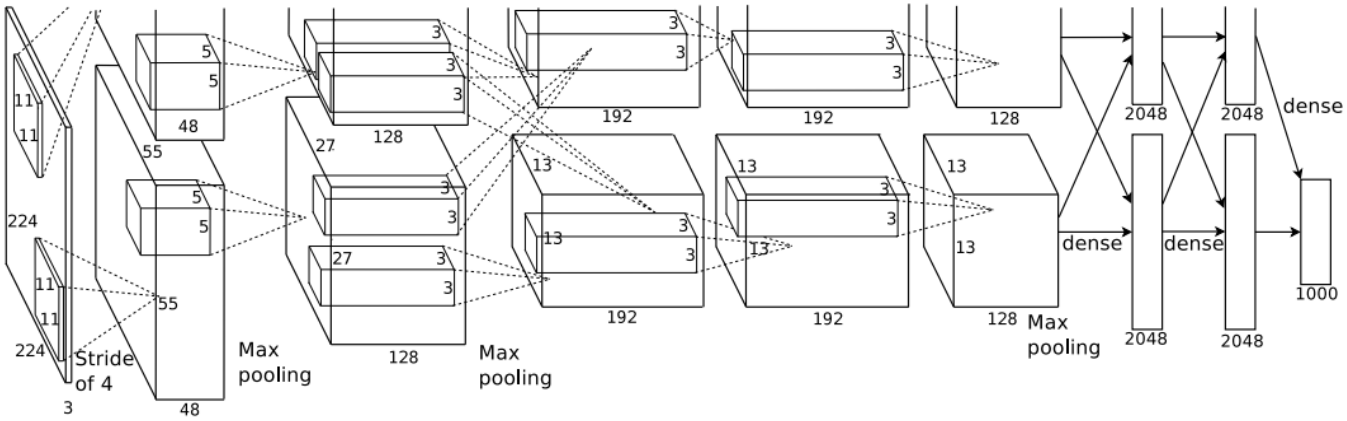
http://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

The boom of DL: noticeable achivements

- Large scale image classification
- Speech recognition
- Face recognition
- Deep reinforcement learning
- Other achievements
  - Image captioning
  - Word embeddings
  - Gesture / action recognition
  - Super resolution
  - …

Breakthrough achievements I (ImageNET)

- In 2012, Krizhevsky et al. succeded at training a convolutional neural network with about 1 million images, approaching the ImageNET large scale classification challenge (1000 of classes, millions of images)



IMAGENET

Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. **ImageNet Classification with Deep Convolutional Neural Networks.** Advances in Neural Information Processing Systems 25 (NIPS 2012)

The ImageNET challenge

- ImageNET: A huge resource comprising millions of images.
- Images were downloaded from the web using synsets from WordNet
- The ImageNET challenge is organized since 2011
  - Classification
  - Object detection
  - Object localization

IMAGENET

Year	Train images (per class)	Val images (per class)	Test images (per class)
Image classification annotations (1000 object classes)			
ILSVRC2010	1,261,406 (668–3047)	50,000 (50)	150,000 (150)
ILSVRC2011	1,229,413 (384–1300)	50,000 (50)	100,000 (100)
ILSVRC2012-14	1,281,167 (732–1300)	50,000 (50)	100,000 (100)

The numbers in parentheses correspond to (minimum per class–maximum per class). The 1000 classes change from year to year but are consistent between image classification and single-object localization tasks in the same year. All images from the image classification task may be used for single-object localization

Breakthrough achievements I (ImageNET)

- Performance improvement with solutions from those days was impressive
- Key for success:
  - GPU based training
  - RELU activation functions
  - Dropout regularization
  - Big data / complex model

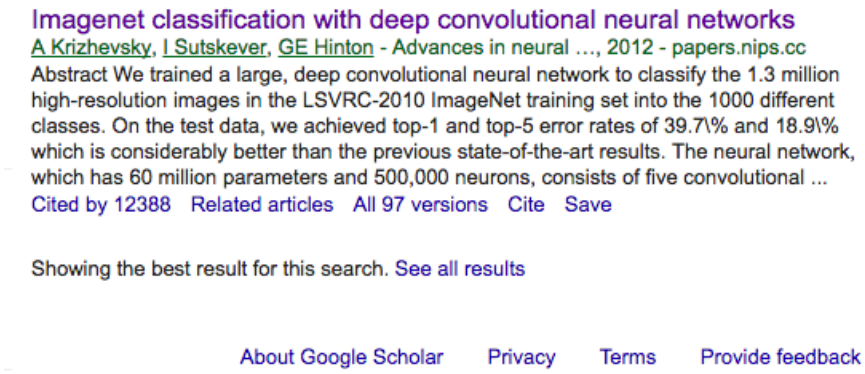
Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**

IMAGENET

Breakthrough achievements I (ImageNET)

- AlexNet is nowadays (perhaps) the most used architecture for image classification and related tasks, it is included in most DL toolkits
- This success inspired other major achievements in computer vision with DL (... Ultimately establishing DL as the *de facto* methodology in CV)



Imagenet classification with deep convolutional neural networks
A Krizhevsky, I Sutskever, GE Hinton - Advances in neural ..., 2012 - papers.nips.cc
Abstract We trained a large, deep convolutional neural network to classify the 1.3 million high-resolution images in the LSVRC-2010 ImageNet training set into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 39.7\% and 18.9\% which is considerably better than the previous state-of-the-art results. The neural network, which has 60 million parameters and 500,000 neurons, consists of five convolutional ...
Cited by 12388 Related articles All 97 versions Cite Save

Showing the best result for this search. See all results

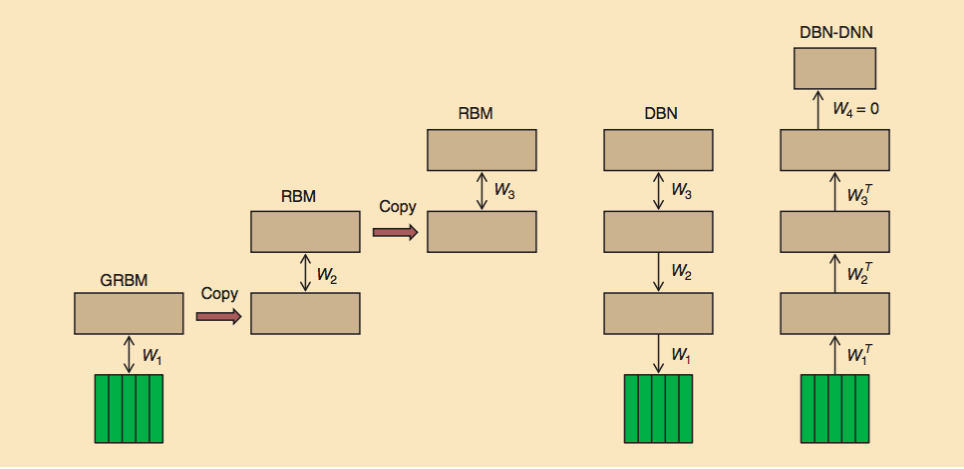
About Google Scholar Privacy Terms Provide feedback

[PDF] nips.cc

Checked, June 14
(one day after: 12437 citations, today?)

Breakthrough achievements II (Speech recognition)

- Around 2012, the most important IT companies converged to the use of Restricted Boltzman Machines for Speech Recognition
- Key idea: RBM-pretraining + fine tunning + HMM



Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. **Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups.** IEEE Signal Processing Magazine, Vol 29(6):82 - 97, 2012

Breakthrough achievements II (Speech recognition)

- Impressive results were attained by such type of models:

[TABLE 1] COMPARISONS AMONG THE REPORTED SPEAKER-INDEPENDENT (SI) PHONETIC RECOGNITION ACCURACY RESULTS ON TIMIT CORE TEST SET WITH 192 SENTENCES.

METHOD	PER
CD-HMM [26]	27.3%
AUGMENTED CONDITIONAL RANDOM FIELDS [26]	26.6%
RANDOMLY INITIALIZED RECURRENT NEURAL NETS [27]	26.1%
BAYESIAN TRIPHONE GMM-HMM [28]	25.6%
MONOPHONE HTMS [29]	24.8%
HETEROGENEOUS CLASSIFIERS [30]	24.4%
MONOPHONE RANDOMLY INITIALIZED DNNs (SIX LAYERS) [13]	23.4%
MONOPHONE DBN-DNNs (SIX LAYERS) [13]	22.4%
MONOPHONE DBN-DNNs WITH MMI TRAINING [31]	22.1%
TRIPHONE GMM-HMMs DT W/ BMMI [32]	21.7%
MONOPHONE DBN-DNNs ON FBANK (EIGHT LAYERS) [13]	20.7%
MONOPHONE MCRBM-DBN-DNNs ON FBANK (FIVE LAYERS) [33]	20.5%
MONOPHONE CONVOLUTIONAL DNNs ON FBANK (THREE LAYERS) [34]	20.0%

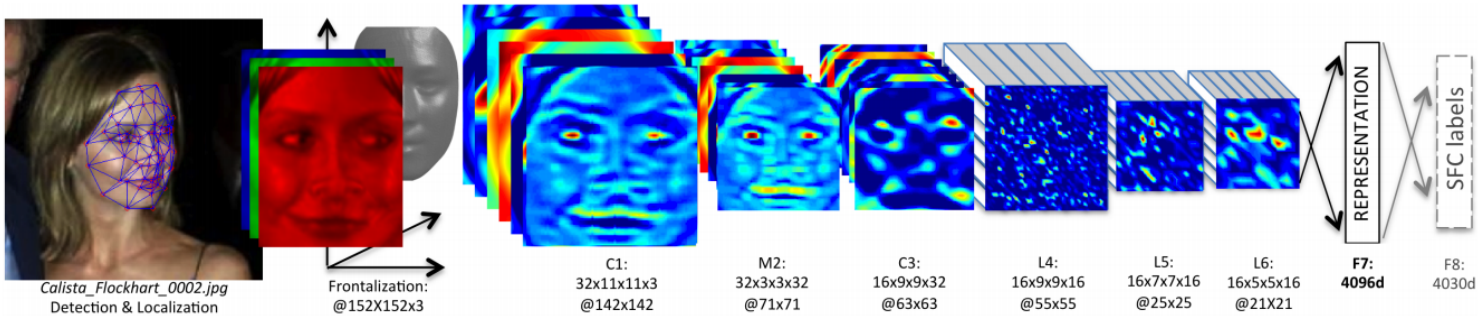
[TABLE 3] A COMPARISON OF THE PERCENTAGE WERs USING DNN-HMMs AND GMM-HMMs ON FIVE DIFFERENT LARGE VOCABULARY TASKS.

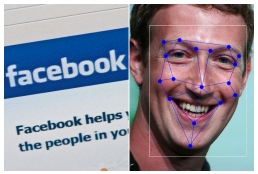
TASK	HOURS OF TRAINING DATA	DNN-HMM	GMM-HMM WITH SAME DATA	GMM-HMM WITH MORE DATA
SWITCHBOARD (TEST SET 1)	309	18.5	27.4	18.6 (2,000 H)
SWITCHBOARD (TEST SET 2)	309	16.1	23.6	17.1 (2,000 H)
ENGLISH BROADCAST NEWS	50	17.5	18.8	
BING VOICE SEARCH (SENTENCE ERROR RATES)	24	30.4	36.2	
GOOGLE VOICE INPUT	5,870	12.3		16.0 (>> 5,870 H)
YOUTUBE	1,400	47.6	52.3	

Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. **Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups.** IEEE Signal Processing Magazine, Vol 29(6):82 - 97, 2012

Breakthrough achievements ++: Deepface

- In 2014, it was announced deep face a deep neural net trained on 4.44 million images

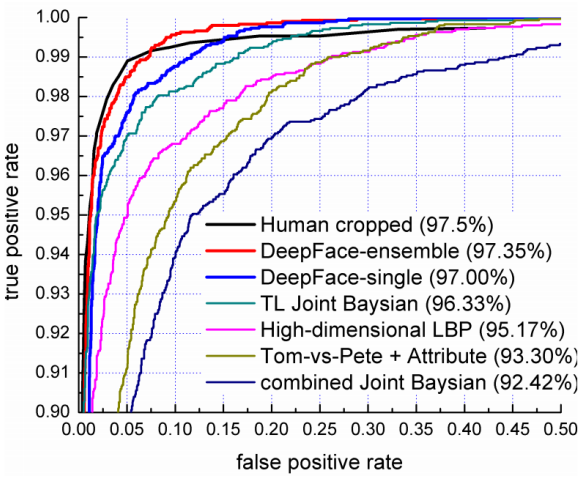


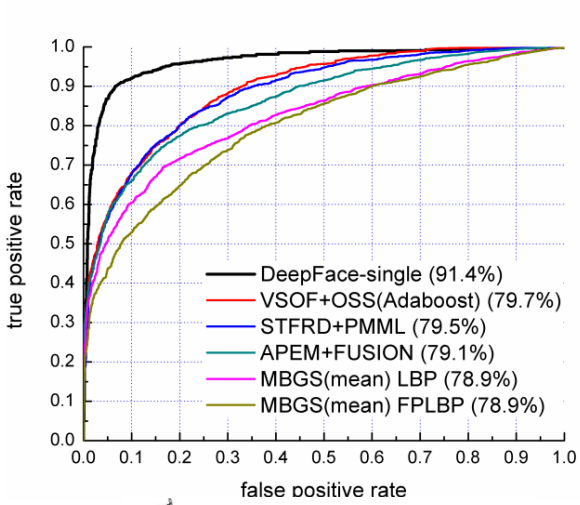


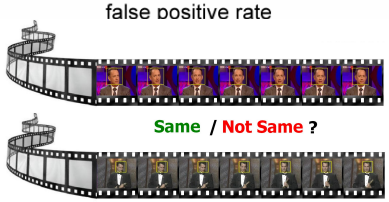
Yaniv Tagiman, Ming Yang, Marc Aurelio Ranzato, Lior Wolf**. DeepFace: Closing the Gap to Human-Level Performance in Face Verification**, CVPR 2014

Breakthrough achievements ++: Deepface

- Deepface achieved a 97.35% of recognition accuracy in the LFW data set (human 97.5%) and 91.4% in the youtube faces data set.

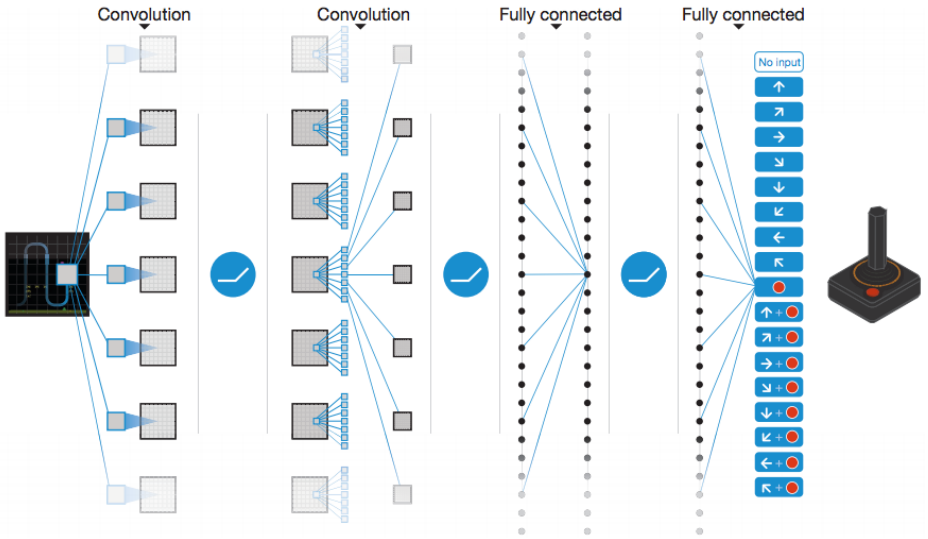






Breakthrough achievements ++: DeepRL

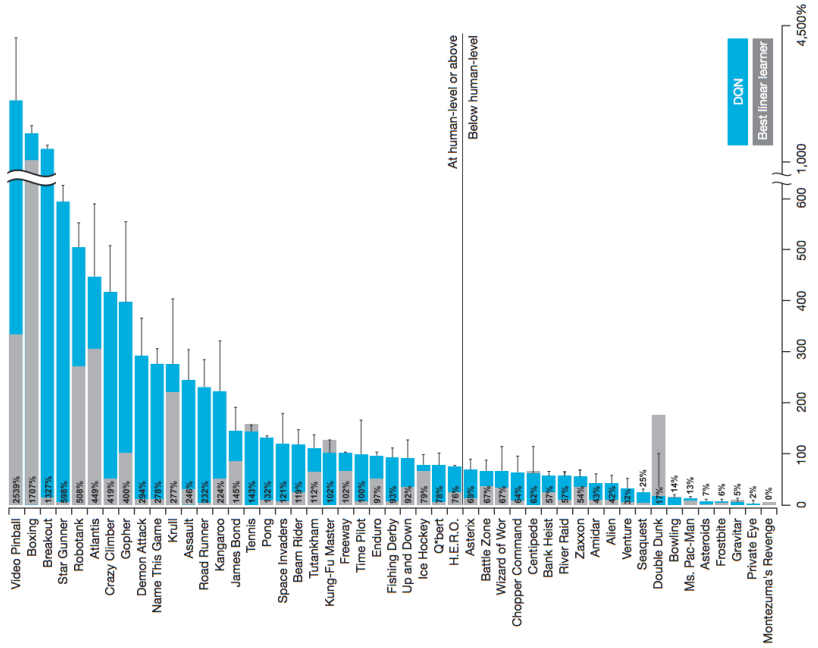
- In 2015, the deepmind team published their Deep-Q network: a DL architecture that by ”looking” at the pixels produced in videogames and using game scores, was able to learn to play Atari



Volodymyr Mnih, et al**. Human-level Control through Deep Reinforcement Learning** In Nature, 518: 529–533, 2015.

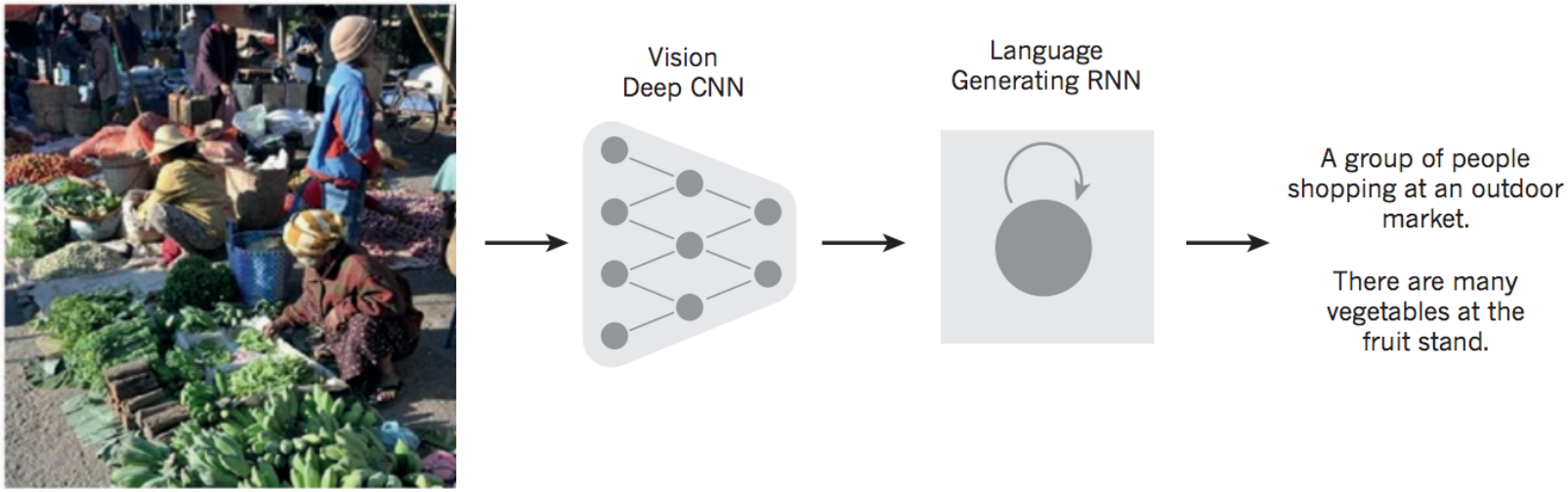
Breakthrough achievements ++: DeepRL

- DQ outperformed all previous solutions in a suite of 50 Atari games
- Achieving human (expert) level performance in a large portion of the games



https://deepmind.com/blog/deep-reinforcement-learning/

Breakthrough achievements ++: Image Captioning



https://pdollar.wordpress.com/2015/01/21/image-captioning/

Breakthrough achievements ++: Image Captioning

A woman is throwing a **frisbee** in a park.

A **dog** is standing on a hardwood floor.

A **stop** sign is on a road with a mountain in the background

A little **girl** sitting on a bed with a teddy bear.

A group of **people** sitting on a boat in the water.

A giraffe standing in a forest with **trees** in the background.

https://pdollar.wordpress.com/2015/01/21/image-captioning/

Breakthrough achievements ++

- Image captioning https://pdollar.wordpress.com/2015/01/21/image-captioning/
- Distributed representations for words
- Translation
- …

References

- Book:
  - Ian Goodfellow, Yoshua Bengio, Aaron Courville. **Deep Learning.** MIT Press, 2016
- Overviews:
  - Yann LeCun, Yoshua Bengio & Geoffrey Hinton. **Deep learning**. Nature 521, 436–444 (28 May 2015)
  - Andrew L. Beam. Deep Learning 101 - Part 1: History and Background, Blog post, Feb 23, 2017

- Breakthrough:
  - Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. **ImageNet Classification with Deep Convolutional Neural Networks.** Advances in Neural Information Processing Systems 25 (NIPS 2012)
  - Yaniv Tagiman, Ming Yang, Marc Aurelio Ranzato, Lior Wolf. **DeepFace: Closing the Gap to Human-Level Performance in Face Verification**, CVPR 2014
  - Volodymyr Mnih, et al. **Human-level Control through Deep Reinforcement Learning** In Nature, 518: 529–533, 2015.
  - Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. **ImageNet Large Scale Visual Recognition Challenge.** IJCV, 2015.
  - Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. **Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups.** IEEE Signal Processing Magazine, Vol 29(6):82 - 97, 2012
  - https://pdollar.wordpress.com/2015/01/21/image-captioning/

Deep learning in a nutshell

- Deep learning is a machine learning methodology that aims at solving (modeling) problems by building layer-wise models with several (many) levels of increasing abstraction
- Layers of these models capture discriminative/descriptive information from raw data
- Can be used for: supervised/unsupervised learning, reinforcement learning, feature extraction, …
- Examples: multi-layer perceptrons, deep neural networks, convolutional neural networks, deep belief nets, auto encoders, etc.

Deep neural networks

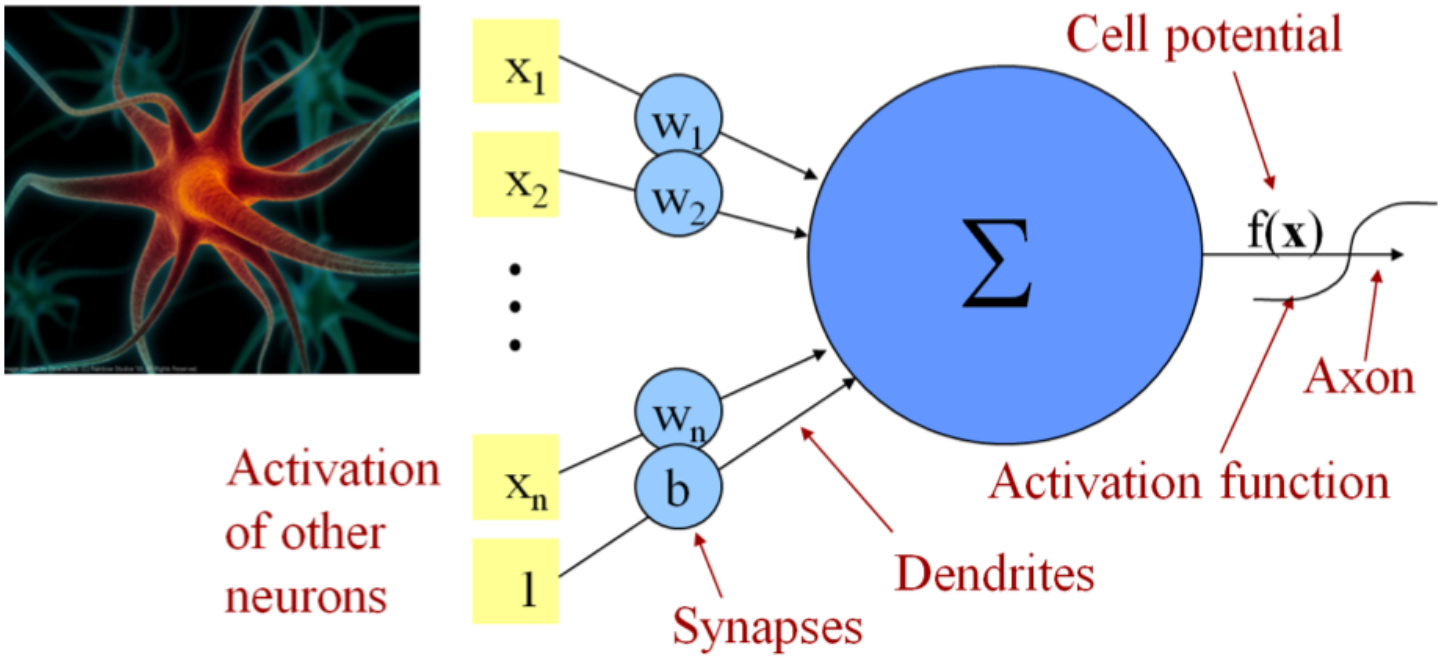
- Deep feedforward networks are the ``essential" deep learning models
- Conventionally, a neural network is said to be deep if it has at least 2 hidden layers
- Hence, feedforward neural networks comprise the fundamentals of deep learning
- How much do you know about NNs?

Neural networks – recap.

- A feedforward neural network is a model:
  - That approximates functions of the form $y = f(x; \Theta)$
  - Is formed by multiple (nonlinear) functions arranged in layers
  - Layers form a network
  - In which information flows in a single (forward) direction

Neural networks – recap.

- Neuron analogy



Slide from I. Guyon

Neural networks – recap. (from perceptrons to DNNs)

- In general neural networks are built of units that resemble the perceptron (linear units activated by a differentiable function)

- We will revisit the perceptron, linear units and will arrive to MLPs or DNNs

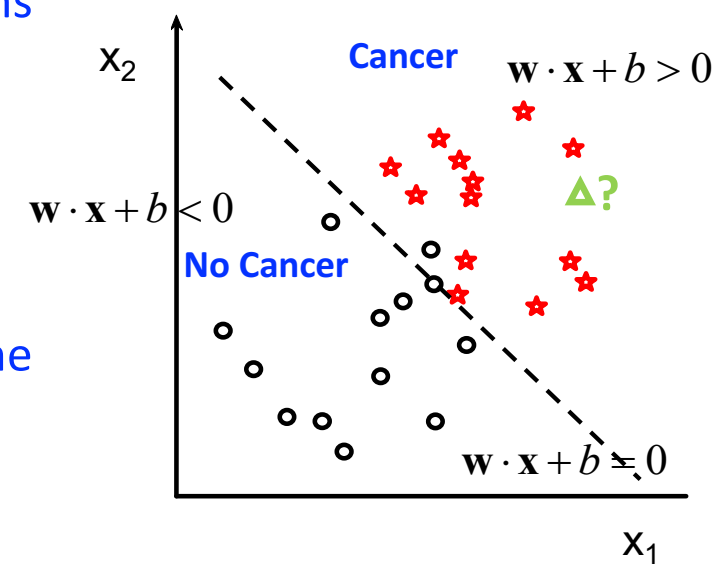
Neural networks – recap. (from perceptrons to DNNs)

- **Perceptron.** A simple, linear classifier that can solve linearly separable classification problems (grandparent of NNs and the SVM)

- Given:
  - $D = \{(\mathbf{x}_i, y_i)_{1,\ldots,N}\}$, with $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$

- A perceptron learns a discriminative function of the form:
  - $f(\mathbf{x}) = sign(\mathbf{w}\mathbf{x} + b)$, with $\mathbf{w} \in \mathbb{R}^d$

- The fundamental unit of DNNs!



Neural networks – recap. (from perceptrons to DNNs)

- How to determine the weights **w**?

- The Perceptron learning algorithm

1. **w** ← randomly initialize weights
2. Repeat until stop criterion meet
   I. For each $\mathbf{x}_i \in D$
      a) $o_i \leftarrow \mathbf{w}\mathbf{x}_i + b$ // estimate perceptron's prediction
      b) $\Delta\mathbf{w} \leftarrow \eta(y_i - o_i)$ // estimate the rate of change
      c) $\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$ //Update *w*
3. *Return* **w**

Intuition?

- Convergence guaranteed (linearly separable problems)

What if the problem is non linearly separable?

Neural networks – recap. (from perceptrons to DNNs)

- For non-linearly separable problems, weights for a *similar* unit can be optimized with gradient descent

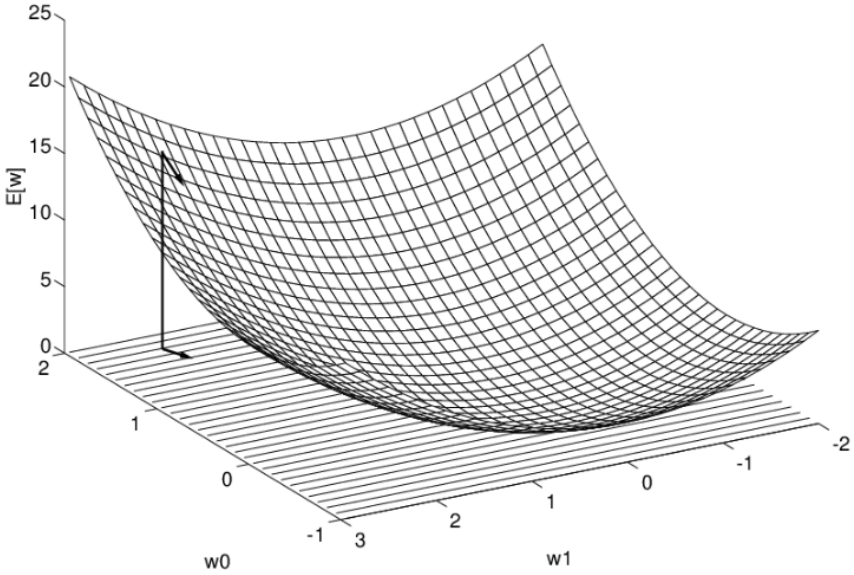
- Suppose we want to learn weights for a perceptron without threshold
  - $f(\mathbf{x}) = (\mathbf{w}\mathbf{x} + b)$, with $\mathbf{w} \in \mathbb{R}^d$
  - $f(\mathbf{x}) = (\mathbf{w}\mathbf{x})$, if we augment the input space with a 1, and include b into w

- And suppose we want to minimize:
  - $E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - o_i)^2$

- How to learn these weights?

Neural networks – recap. (from perceptrons to DNNs)

- Problem:
  - Minimize $E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - o_i)^2$ w.r.t **w**
- Idea: to explore the space of possible values that **w** can take. Starting with an initial w and updating it in the direction that decreases the error
  - (**hint**: the gradient of $E(\mathbf{w})$ indicates the direction that produces the highest increase in E starting in **w**)



Neural networks – recap. (from perceptrons to DNNs)

- The math:

$$W \leftarrow W + \Delta W$$

$$\Delta W = -\alpha \nabla E$$

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\ &= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) \\ &= \sum_{d \in D} (t_d - o_d)(-x_{i,d}) \end{aligned}$$

$$\Delta w_i = \alpha \sum_{d \in D} (t_d - o_d) x_{i,d}$$

Neural networks – recap. (from perceptrons to DNNs)

- The delta rule learning algorithm (gradient descend)

1. $\mathbf{w} \leftarrow$ randomly initialize weights
2. Repeat until stop criterion meet
   - *I.* $\Delta\mathbf{w} \leftarrow$ initialize to 0
   - II. For each $\mathbf{x}_i \in D$
     - *a)* $o_i \leftarrow \mathbf{w}\mathbf{x}_i + b$ // estimate perceptron's prediction
     - b) For each weight j estimate
       - *1.* $\Delta\mathbf{w}_j \leftarrow \Delta\mathbf{w}_j + \eta(y_i - o_i)\, x_{\boldsymbol{i},j}$ // estimate the rate of change
   - **III.** $\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$ //Update *w*
3. *Return* **w**

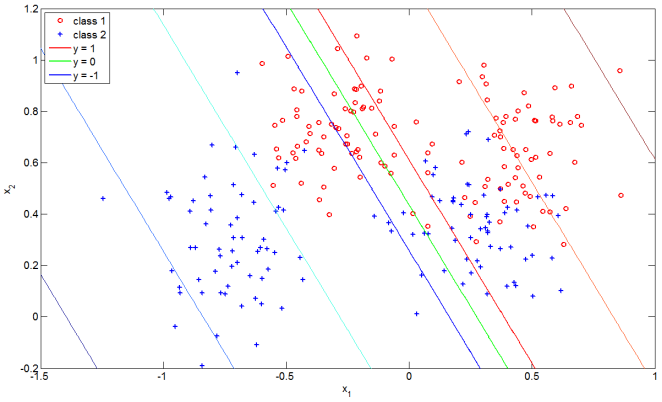
Neural networks – recap. (from perceptrons to DNNs)

- SGD: in practice, a stochastic version of the algorithm is used, in which weights are updated after processing each input

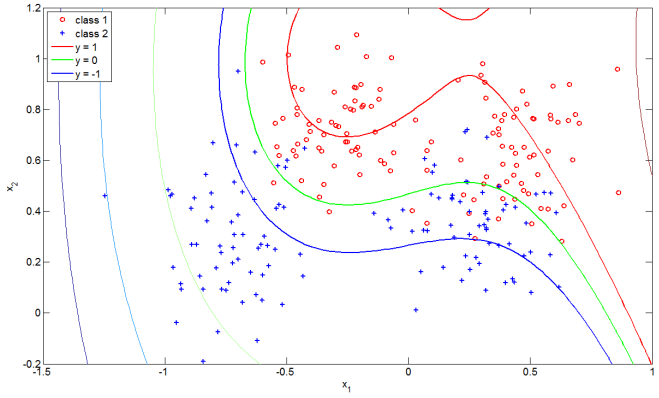
1. $\mathbf{w} \leftarrow$ randomly initialize weights
2. Repeat until stop criterion meet
   I. $\Delta\mathbf{w} \leftarrow$ initialize to 0
   II. For each $\mathbf{x}_i \in D$
      a) $o_i \leftarrow \mathbf{w}\mathbf{x}_i + b$ // estimate perceptron's prediction
      b) For each weight j estimate
         1. $\Delta\mathbf{w}_j \leftarrow \Delta\mathbf{w}_j + \eta(y_i - o_i)\, x_{\boldsymbol{i},j}$ // estimate the rate of change
      c) $\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$ //Update *w*
3. *Return* **w**

Neural networks – recap. (from perceptrons to DNNs)

- The problem with preceptrons et al.: They can only learn linear functions. When the data is not linearly separable the best one can do is to expect to have a *good fit*



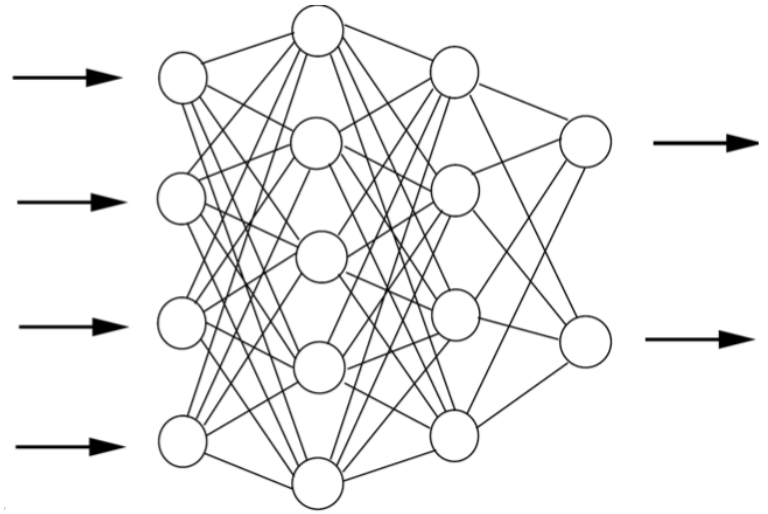
- Solutions?
  - To map the data into a non linear feature space in which the problem can become linearly separable
  - How?



Neural networks – recap. (from perceptrons to DNNs)

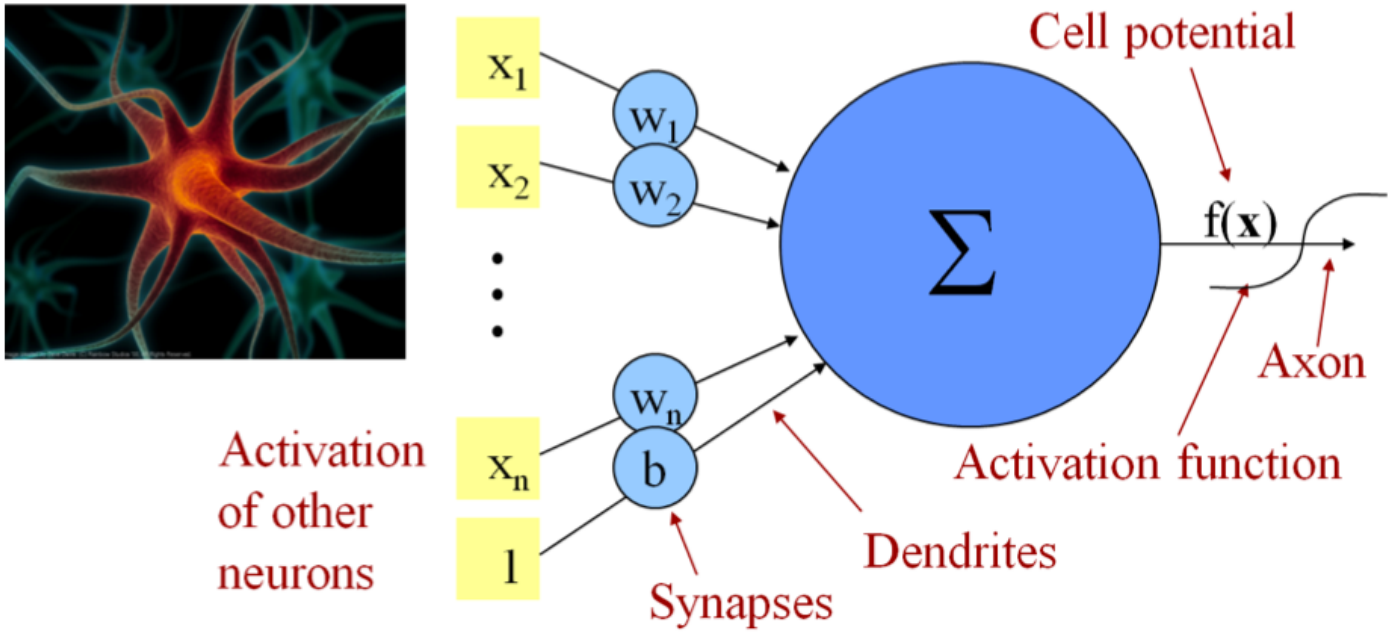
- What about stacking multiple layers of linear units?
  - Still will produce only linear functions

- Idea: stacking multiple layers of linear units *activated* with non linear functions



Neural networks – recap. (from perceptrons to DNNs)

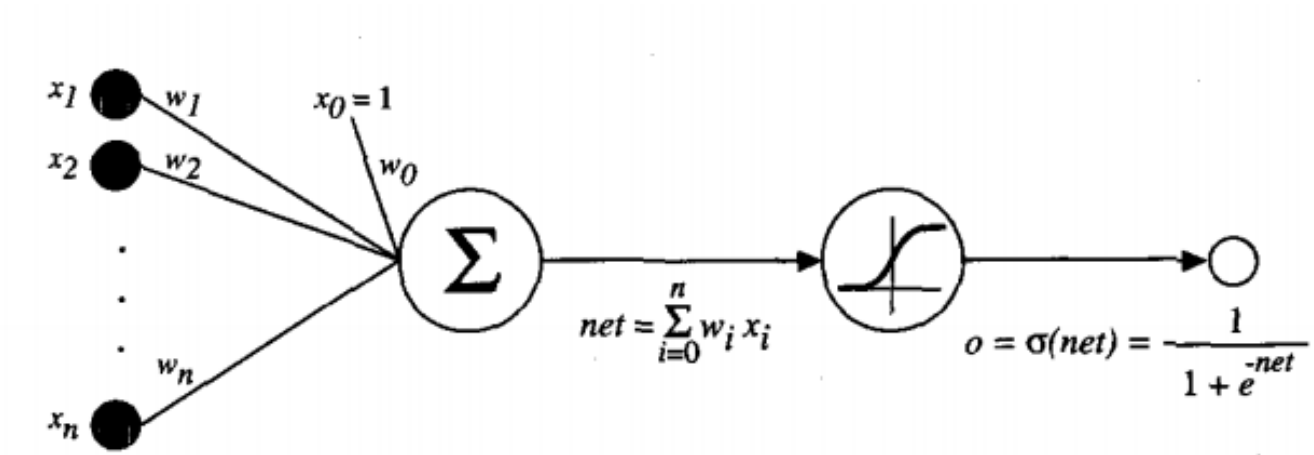
- Introducing non linearities in units



Slide from I. Guyon

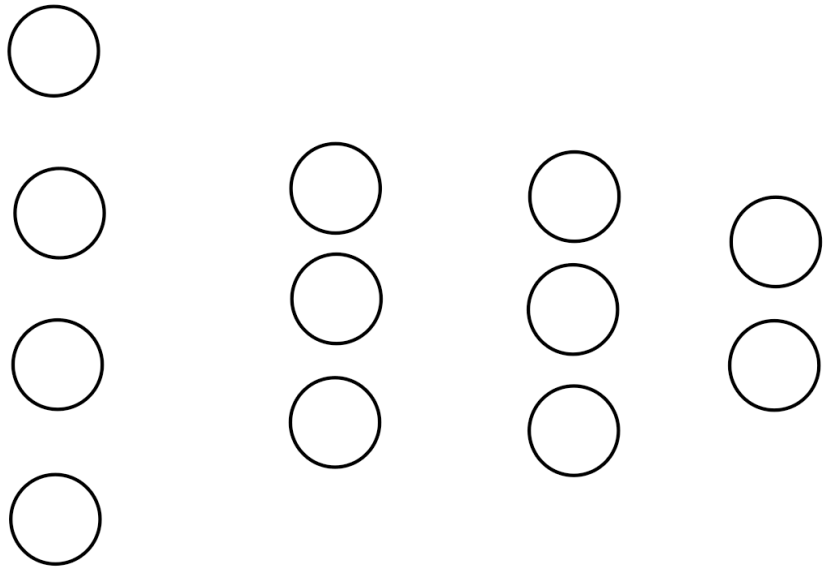
Neural networks – recap. (from perceptrons to DNNs)

- Introducing non linearities in units



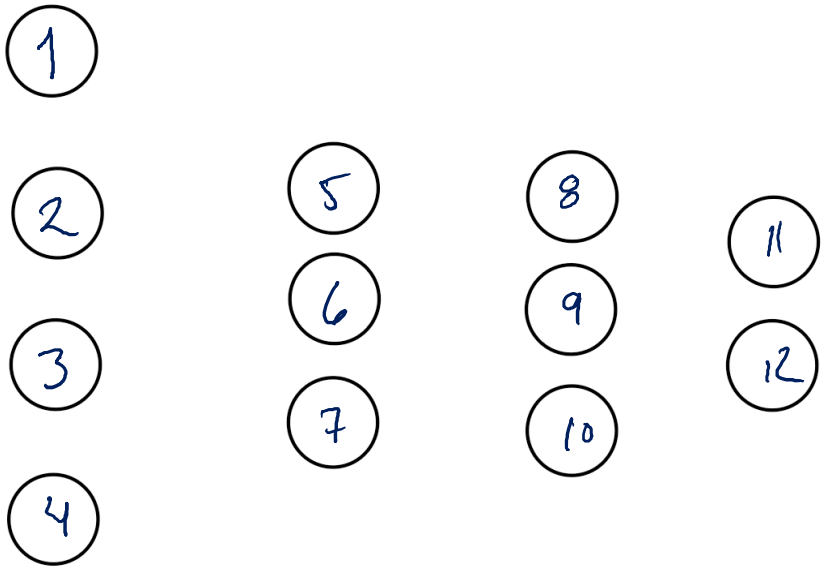
T. Mitchell. **Machine Learning**, McGrawHill 1997,

Neural networks – recap. (from perceptrons to DNNs)



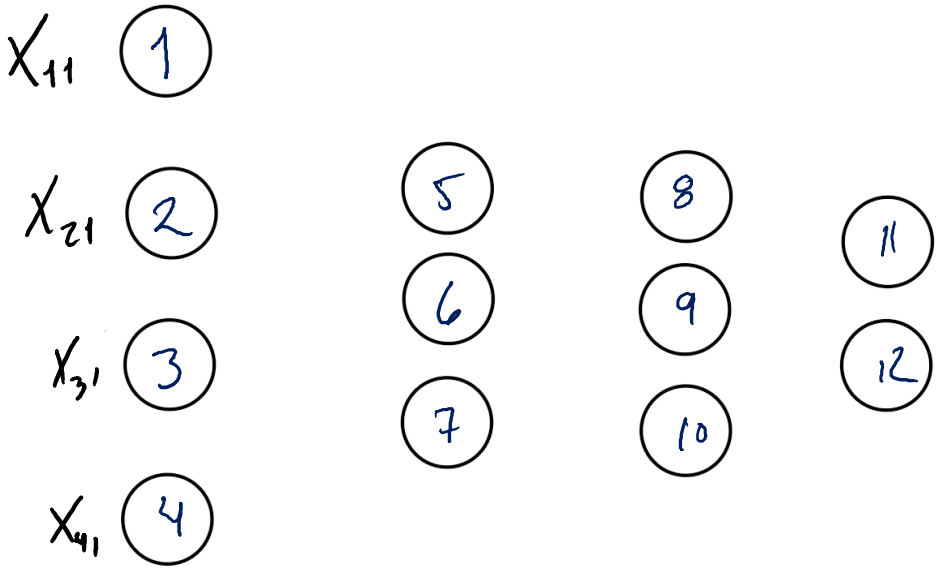
- Disentangling NNs

Neural networks – recap. (from perceptrons to DNNs)



- Disentangling NNs

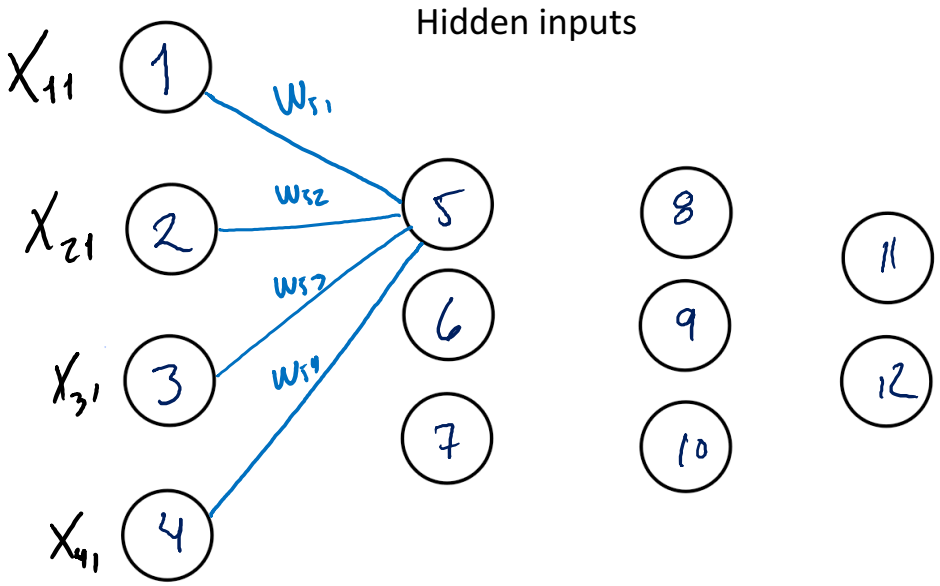
Neural networks – recap. (from perceptrons to DNNs)



Input units

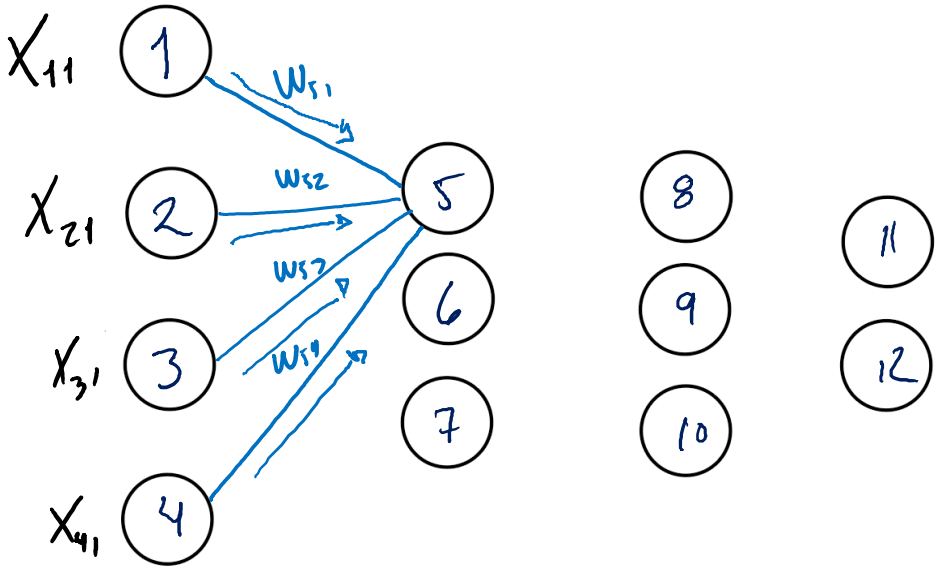
- Disentangling NNs

Neural networks – recap. (from perceptrons to DNNs)



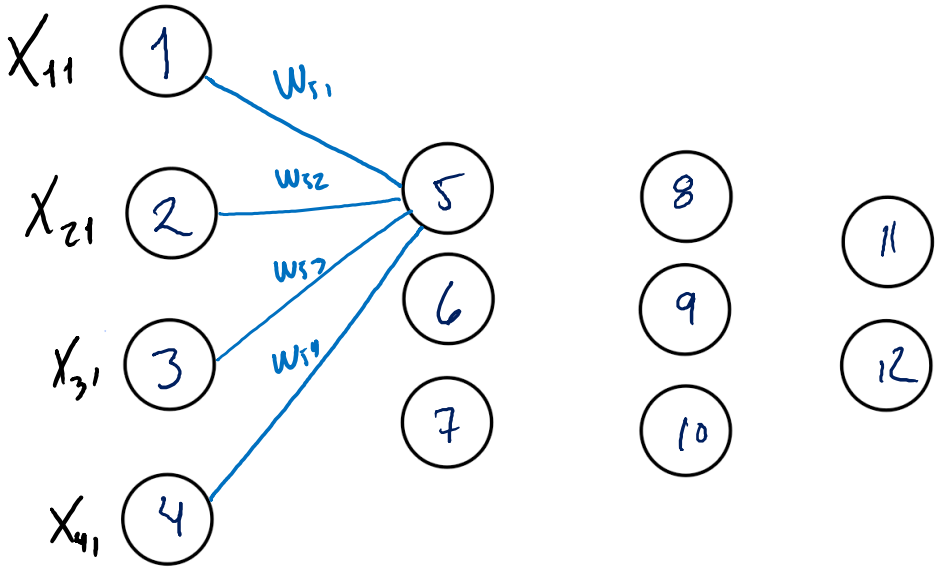
- Disentangling NNs

Neural networks – recap. (from perceptrons to DNNs)



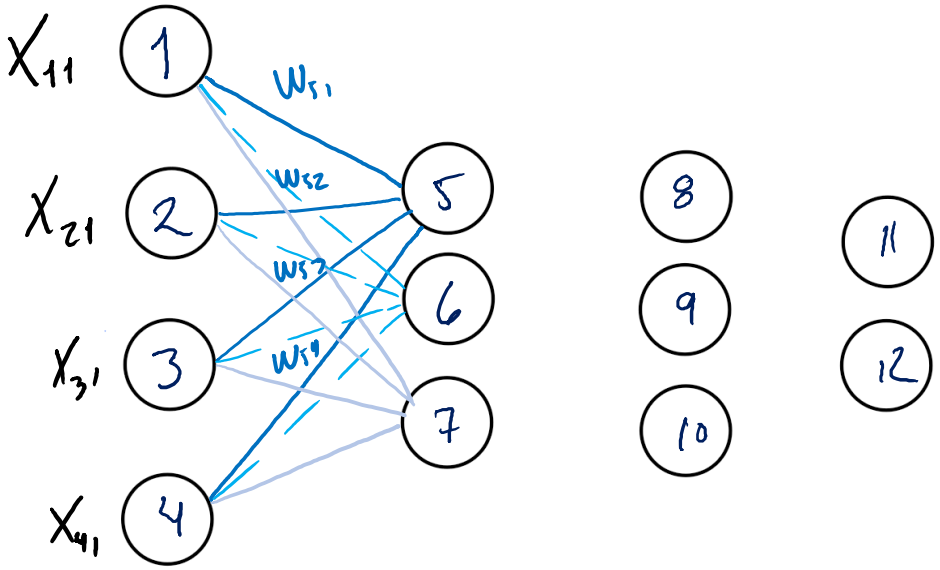
- Disentangling NNs

Neural networks – recap. (from perceptrons to DNNs)



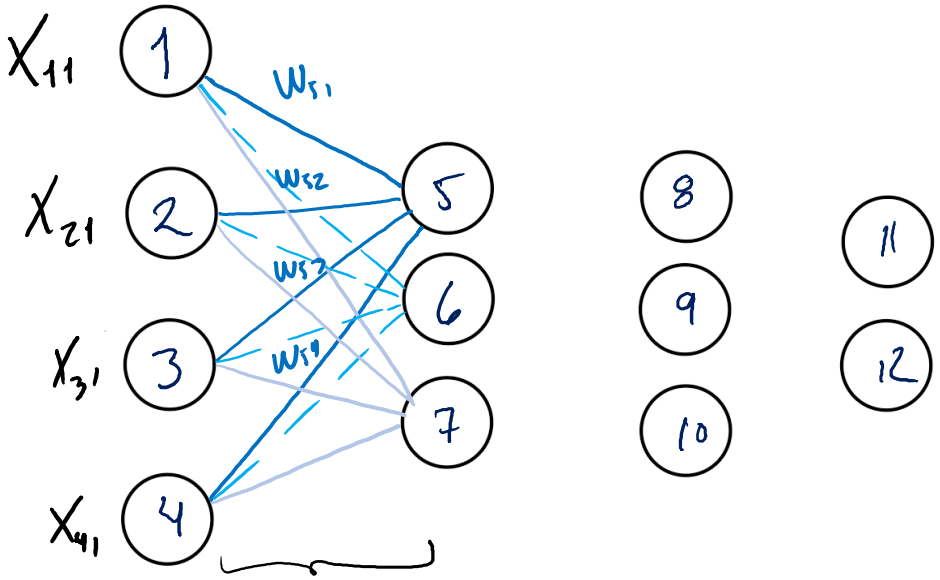
- Disentangling NNs

Neural networks – recap. (from perceptrons to DNNs)



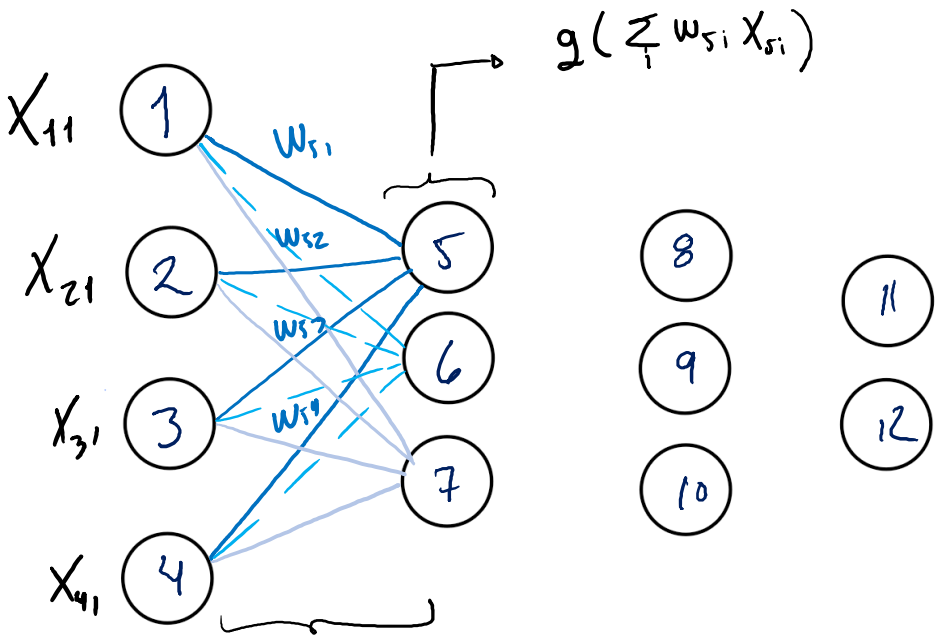
- Disentangling NNs

Neural networks – recap. (from perceptrons to DNNs)



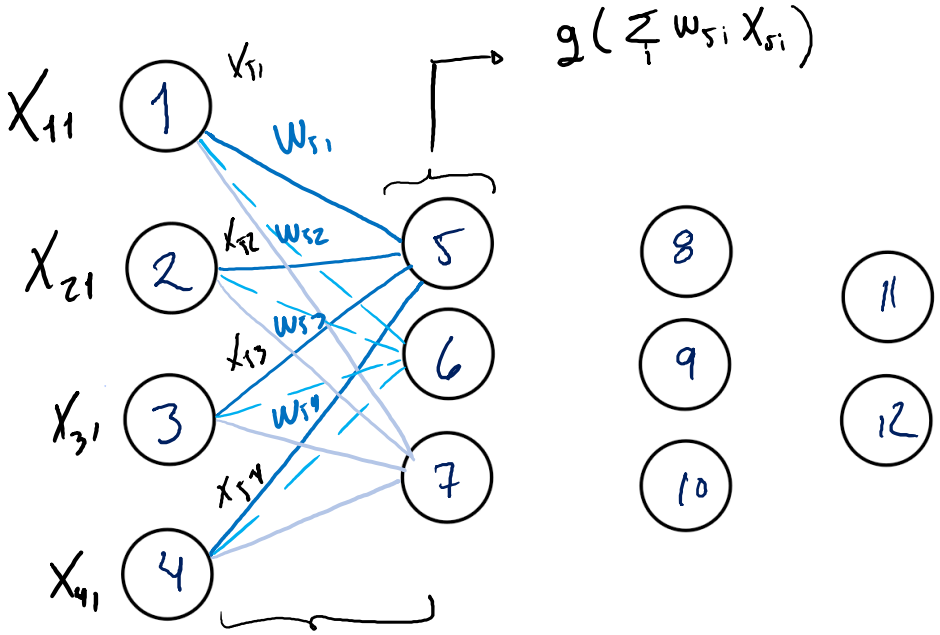
Weight matrix (4x3)

Neural networks – recap. (from perceptrons to DNNs)



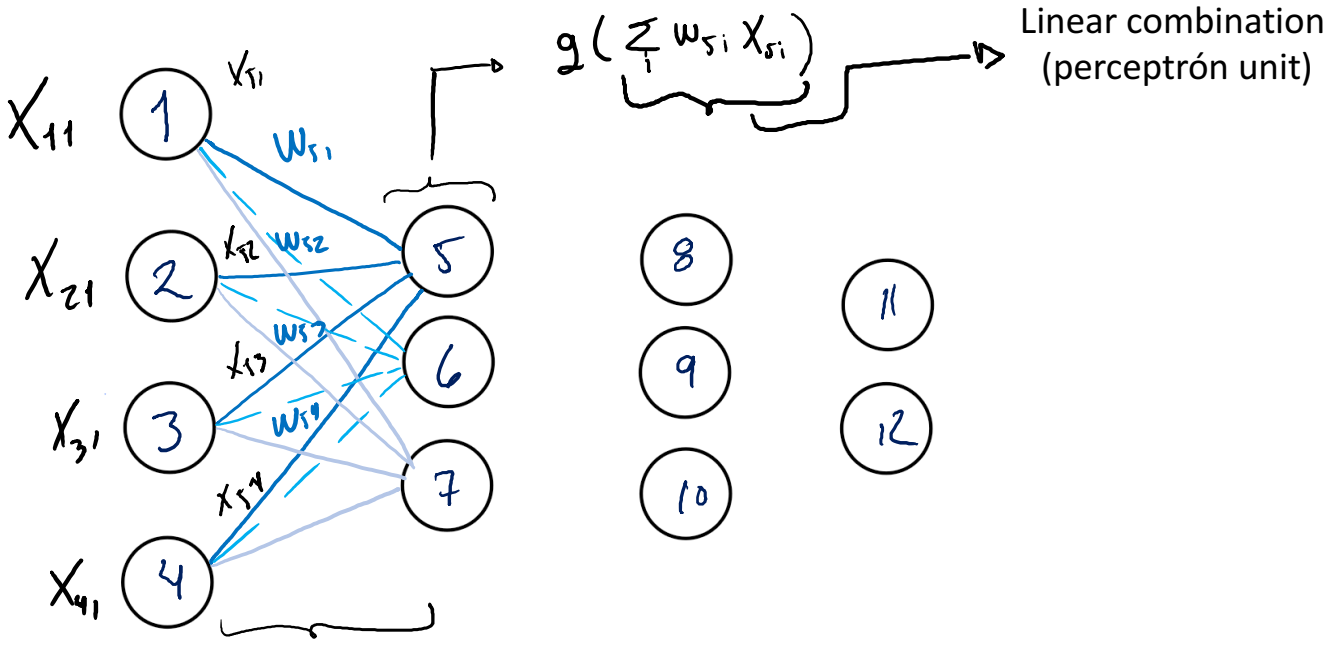
Weight matrix (4x3)

Neural networks – recap. (from perceptrons to DNNs)

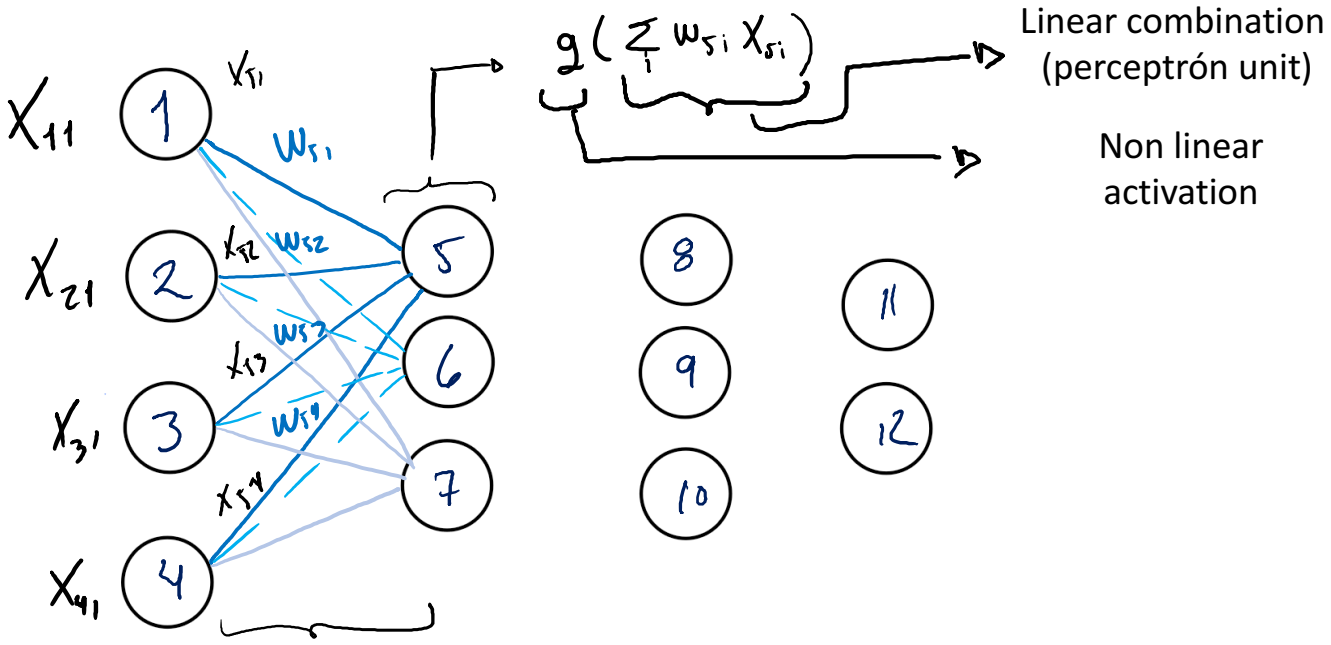


Weight matrix (4x3)

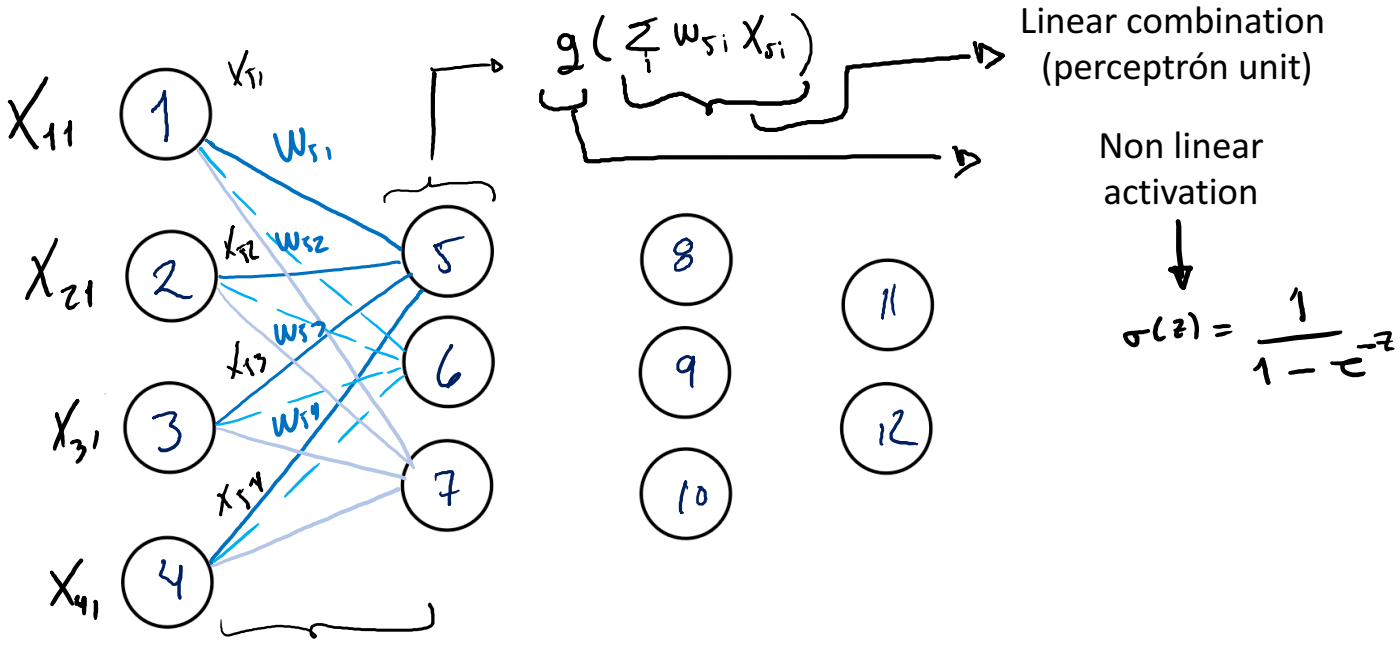
Neural networks – recap. (from perceptrons to DNNs)



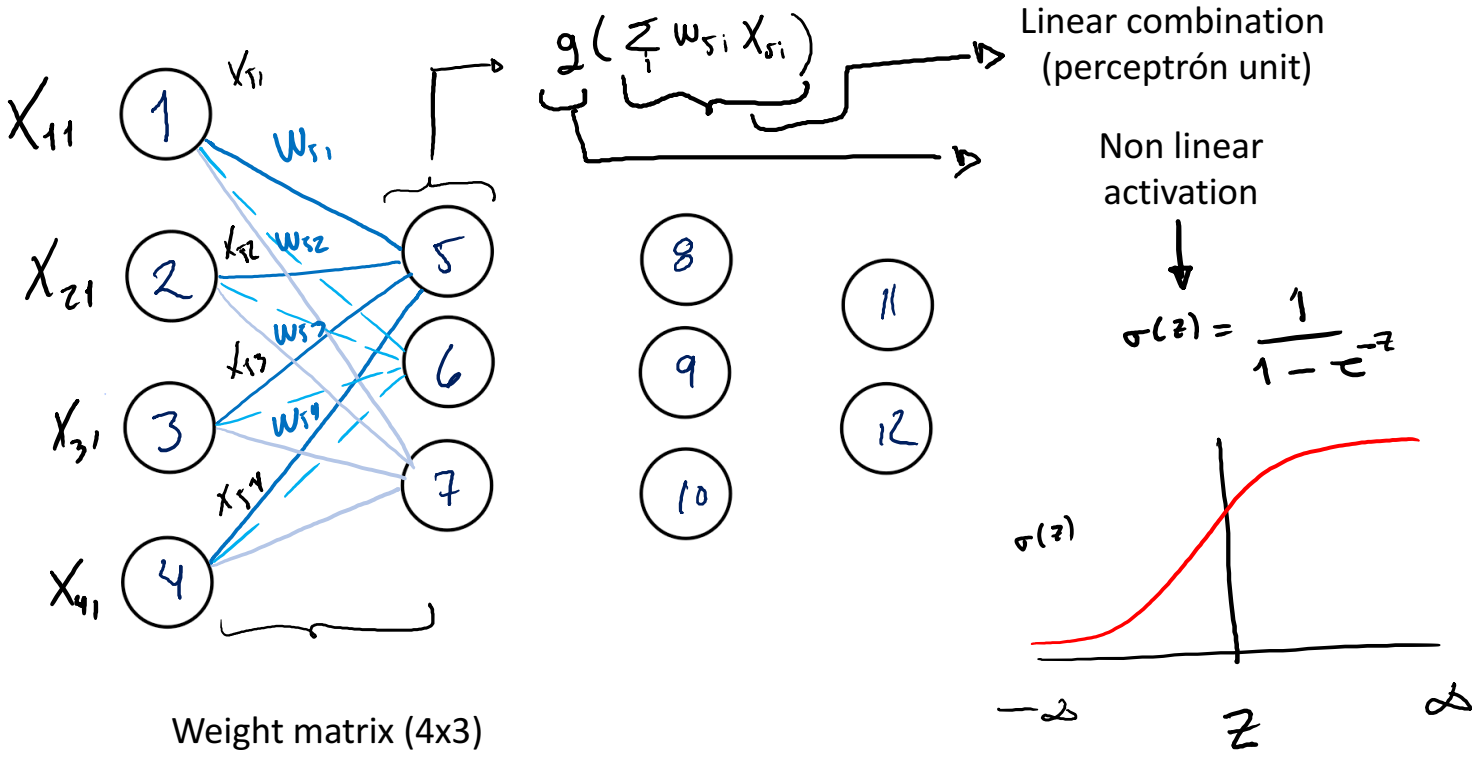
Neural networks – recap. (from perceptrons to DNNs)



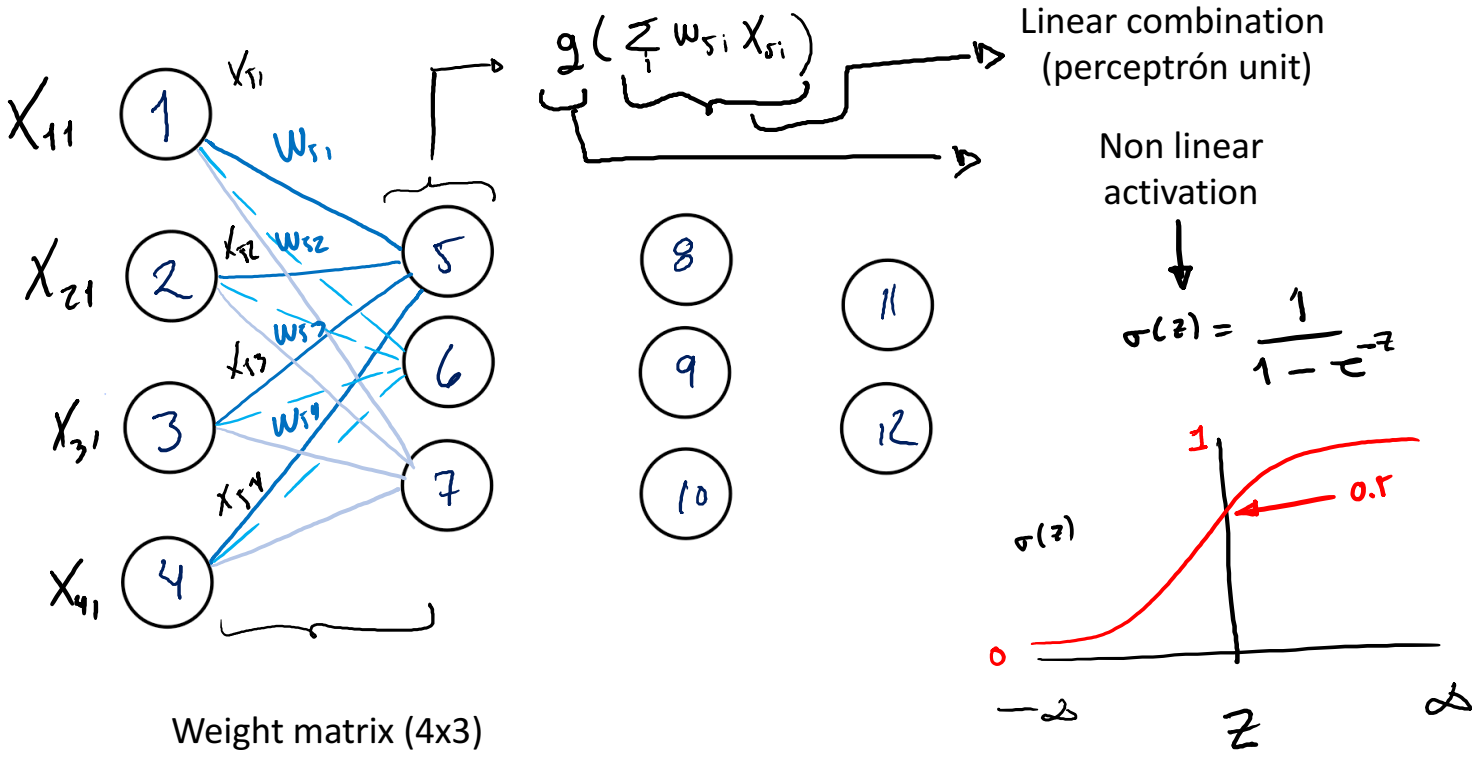
Neural networks – recap. (from perceptrons to DNNs)



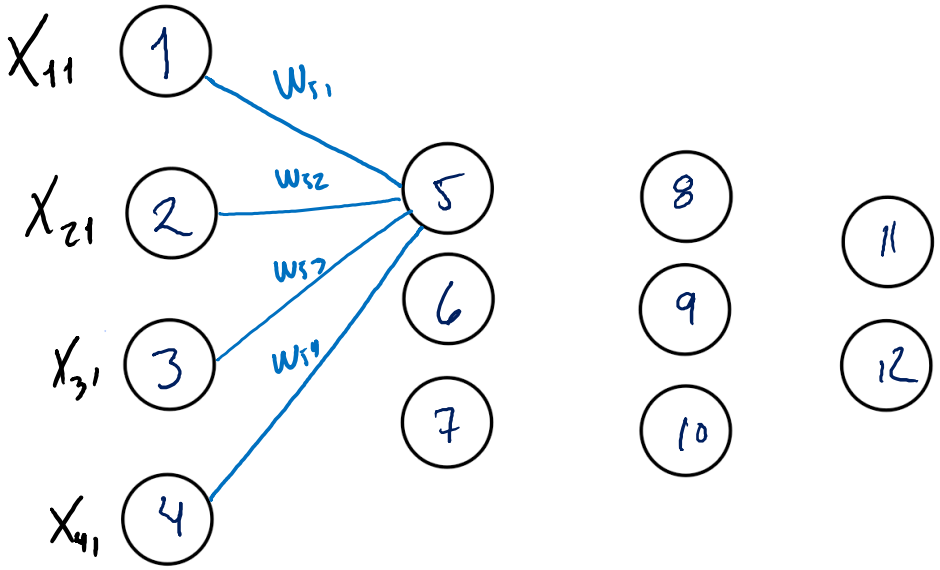
Neural networks – recap. (from perceptrons to DNNs)



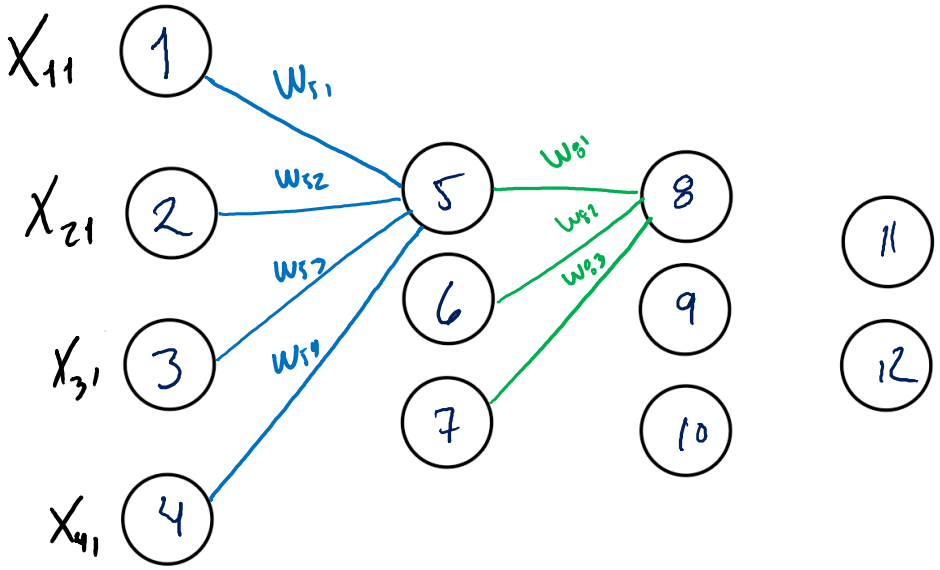
Neural networks – recap. (from perceptrons to DNNs)



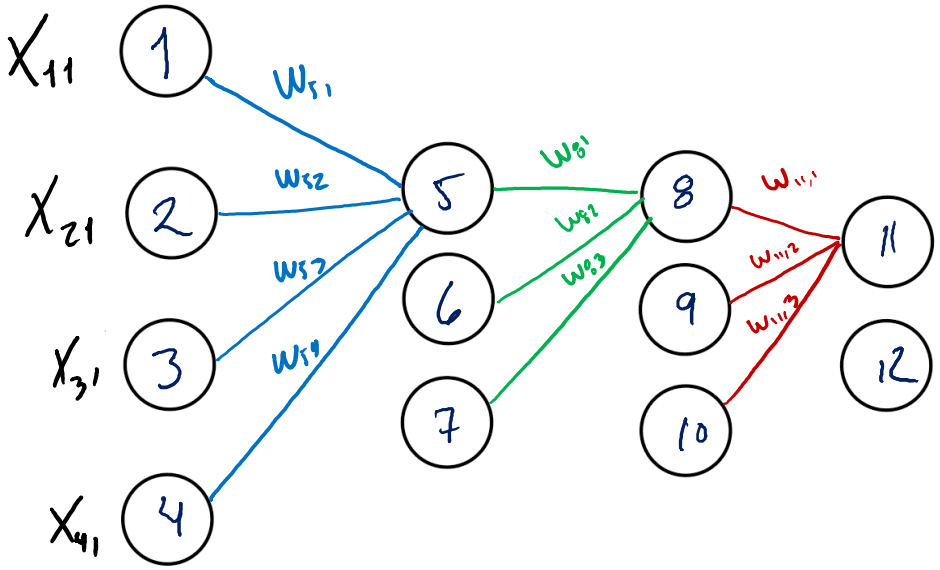
Neural networks – recap. (from perceptrons to DNNs)



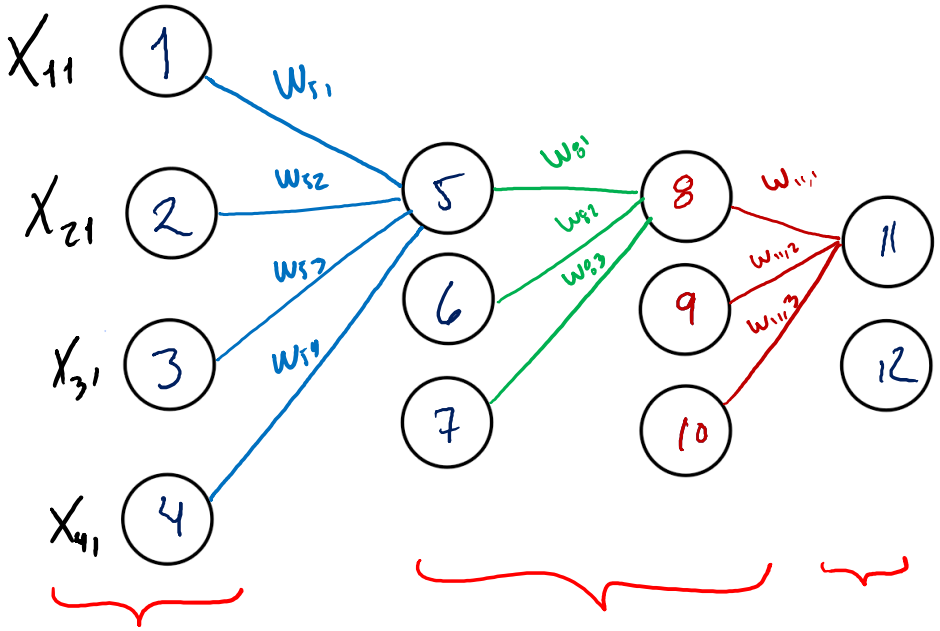
Neural networks – recap. (from perceptrons to DNNs)



Neural networks – recap. (from perceptrons to DNNs)



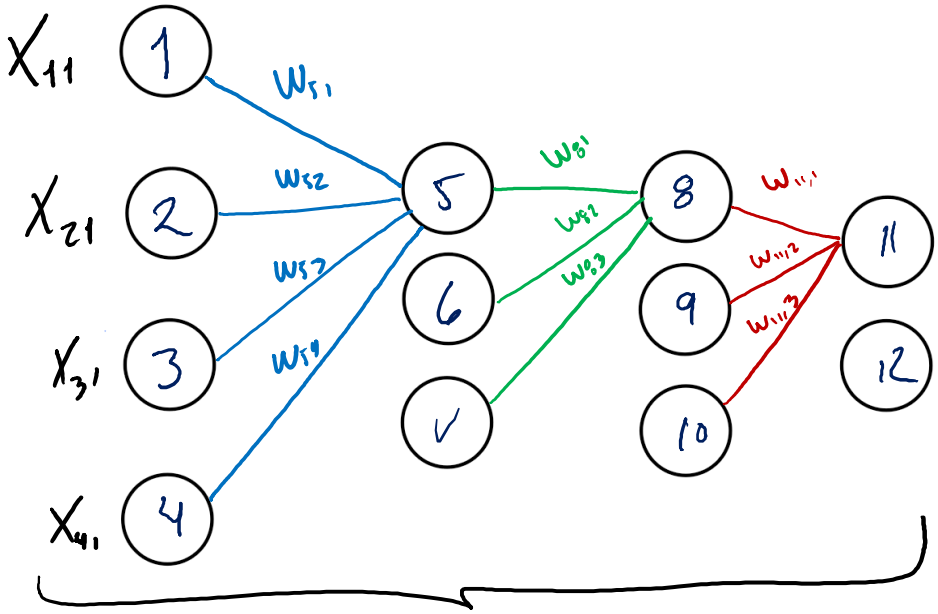
Neural networks – recap. (from perceptrons to DNNs)



Neural networks – recap. (from perceptrons to DNNs)

- As other learning algorithms, NNs aim to learn a function mapping inputs to outputs
- Training a NN reduces to learning the weights in the network that minimize an error estimate
  - How many parameters?
  - How to adjust/determine their values?
  - What criterion to adopt?

Neural networks – recap. (from perceptrons to DNNs)



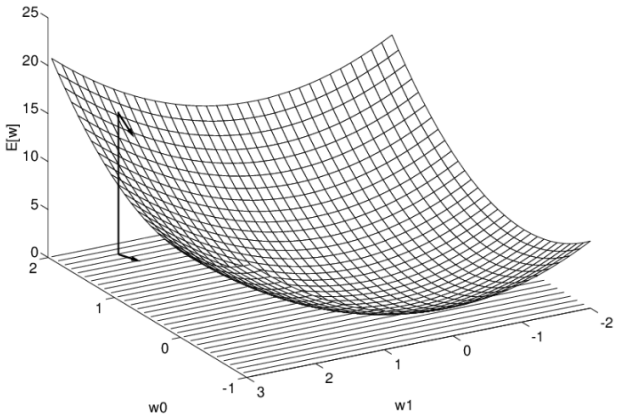
Neural networks – recap. (from perceptrons to DNNs)

- The de facto algorithm for training NNs is backpropagation + gradient descent

- Key idea:
  - use gradient descent to learn the weights that best fit the data
  - smartly using the chain rules of calculus, to dedicate the gradient of error with respect to weights

Neural networks – recap. (from perceptrons to DNNs)

- The backprop + SGD algorithm for training a MLP



T. Mitchell. **Machine Learning**, McGrawHill 1997,

BACKPROPAGATION($training\_examples, \eta, n_{in}, n_{out}, n_{hidden}$)

Each training example is a pair of the form $\langle \vec{x}, \vec{t} \rangle$, *where* $\vec{x}$ *is the vector of network input values, and* $\vec{t}$ *is the vector of target network output values.*

η *is the learning rate (e.g., .05).* n_{in} *is the number of network inputs,* n_{hidden} *the number of units in the hidden layer, and* n_{out} *the number of output units.*

The input from unit i *into unit* j *is denoted* x_{ji}, *and the weight from unit* i *to unit* j *is denoted* w_{ji}.

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers (e.g., between $-.05$ and .05).
- Until the termination condition is met, Do
  - For each $\langle \vec{x}, \vec{t} \rangle$ in $training\_examples$, Do

Propagate the input forward through the network:

1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \qquad \text{(T4.3)}$$

3. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k \qquad \text{(T4.4)}$$

4. Update each network weight w_{ji}

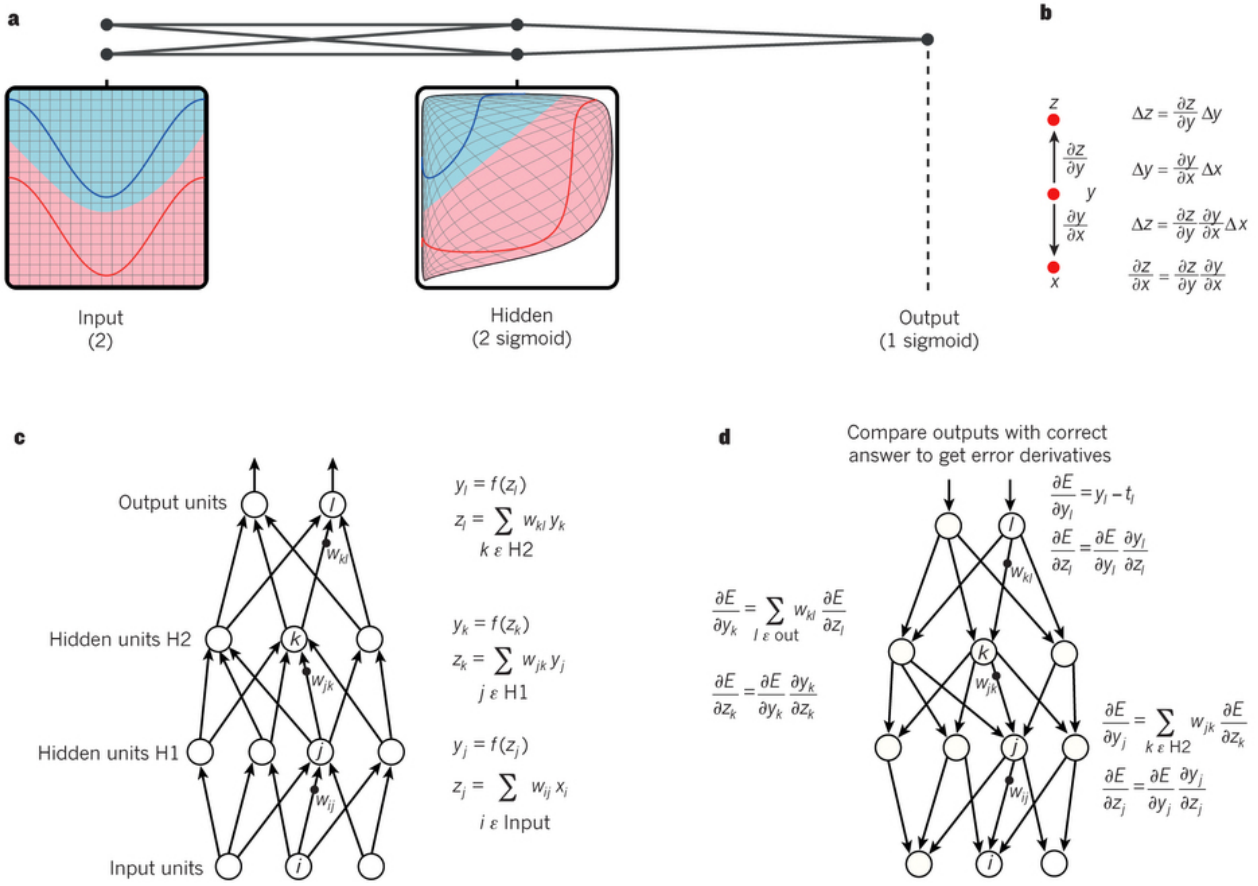
$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

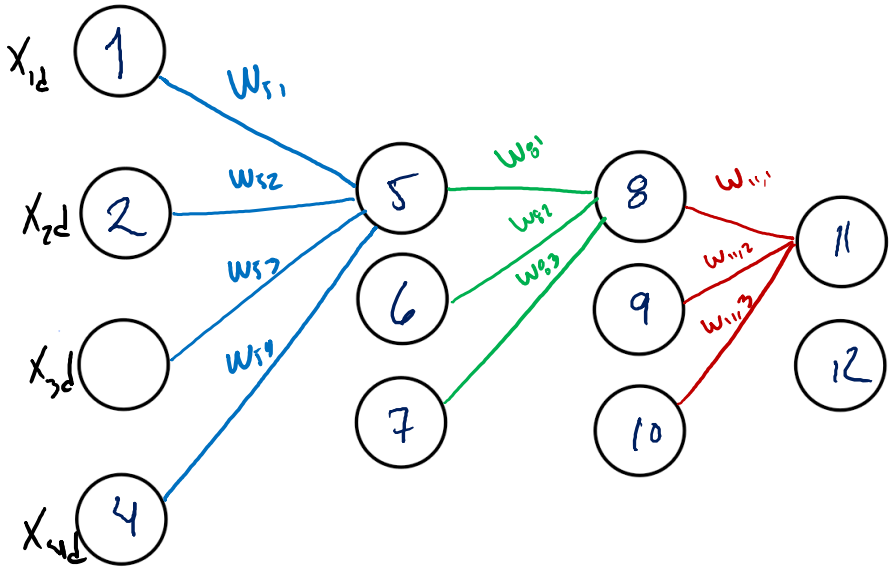
$$\Delta w_{ji} = \eta\, \delta_j\, x_{ji} \qquad \text{(T4.5)}$$

Neural networks – recap. (from perceptrons to DNNs)

- The backprop + SGD algorithm for training a MLP



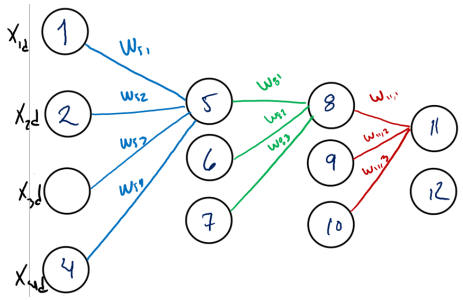
Neural networks – recap. (from perceptrons to DNNs)



Neural networks – recap. (from perceptrons to DNNs)

- We want to obtain the set of weights that minimize

$$E_D(w) \equiv \sum_{d \in D} \sum_{k \in outs} \frac{1}{2} (t_{d,k} - o_{d,k})^2$$



- For a specific instance d

$$E_d(w) \equiv \frac{1}{2} \sum_{k \in outs} (t_k - o_k)^2$$

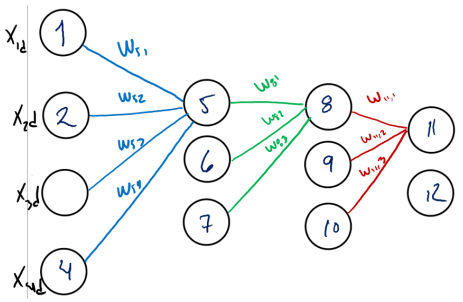
- Using stochastic gradient descent, we aim to update the weights with each instance d as follows

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji} \quad , \text{ with } \quad \Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

Neural networks – recap. (from perceptrons to DNNs)

- We aim to estimate $\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} \quad \forall w_{ji}$
- Since every weight influence the model only through $\sum_i w_{ji} x_{ji}$ } We call this net_j
- We have:

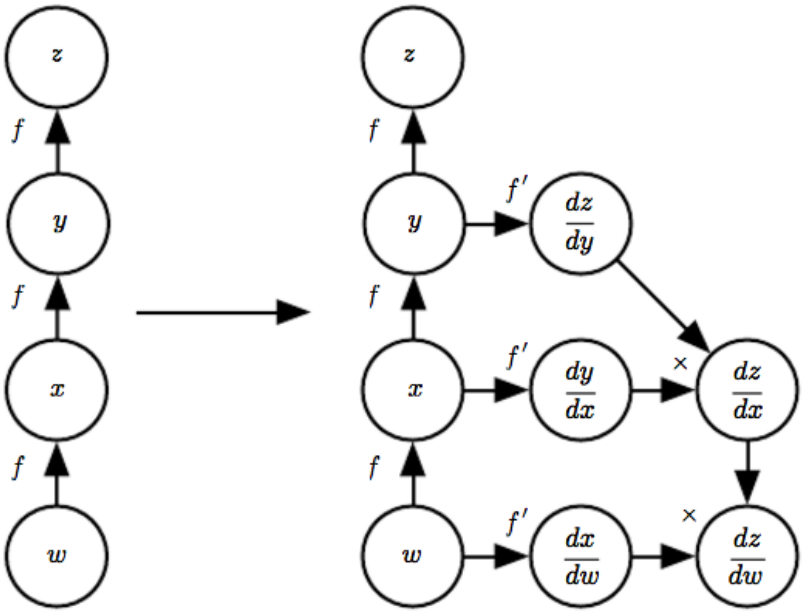
$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} x_{ji}$$



The chain rule of calculus

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$

Neural networks – recap. (from perceptrons to DNNs)

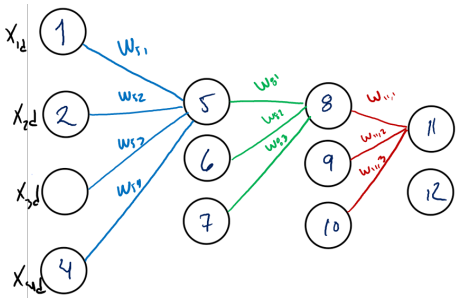


Neural networks – recap. (from perceptrons to DNNs)

- We aim to estimate $\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} \quad \forall w_{ji}$
- Since every weight influence the model only through $\sum_i w_{ji} x_{ji}$ } We call this net_j
- We have:

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} x_{ji}$$

→ Only unknown

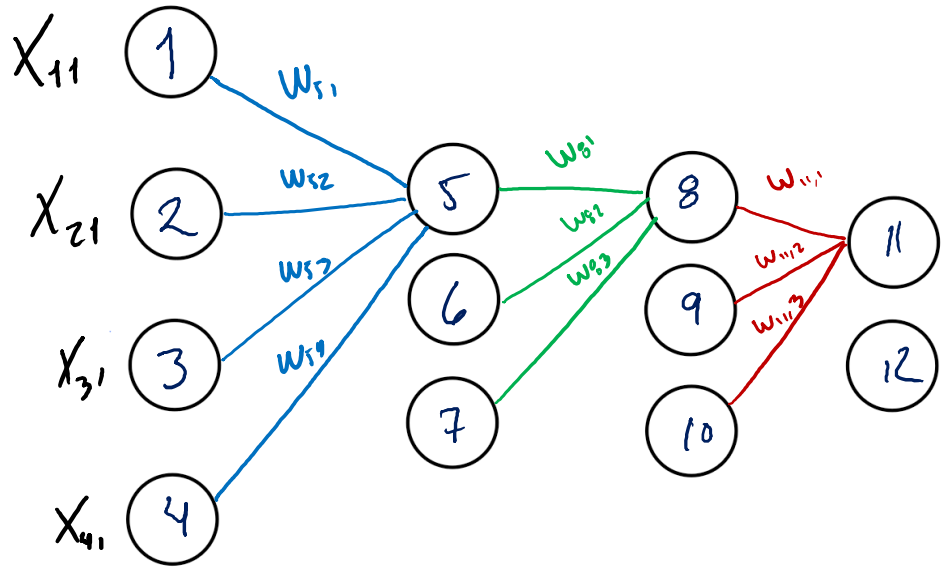


The chain rule of calculus

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$

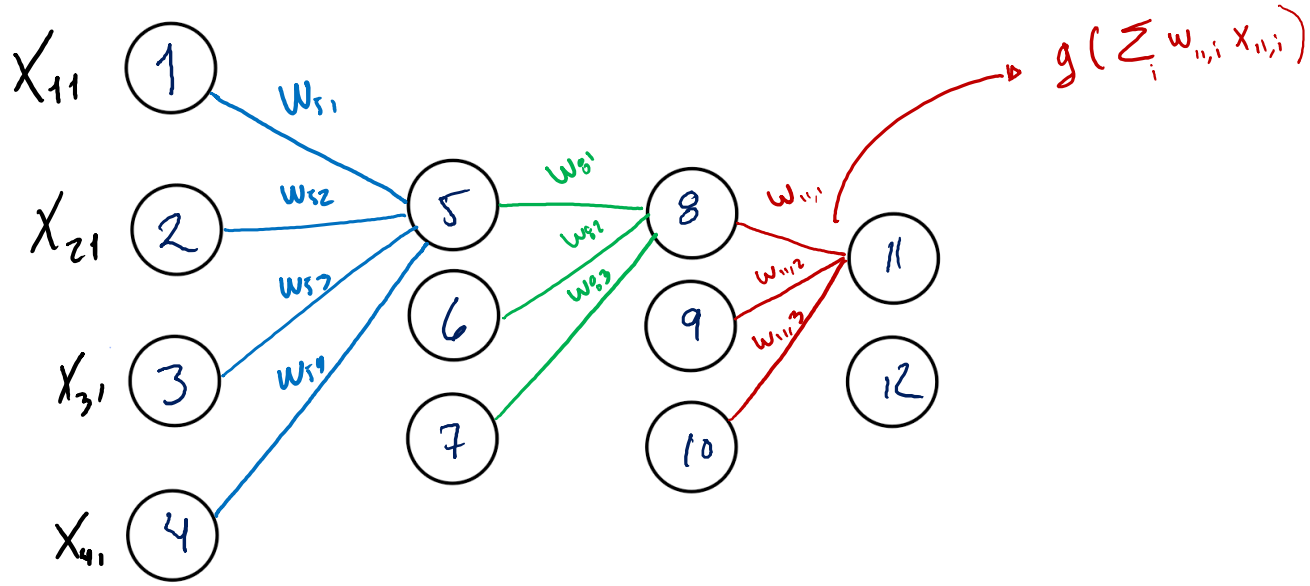
Neural networks – recap. (from perceptrons to DNNs)

- Since there are two types of units, we have to estimate the gradient for both cases:



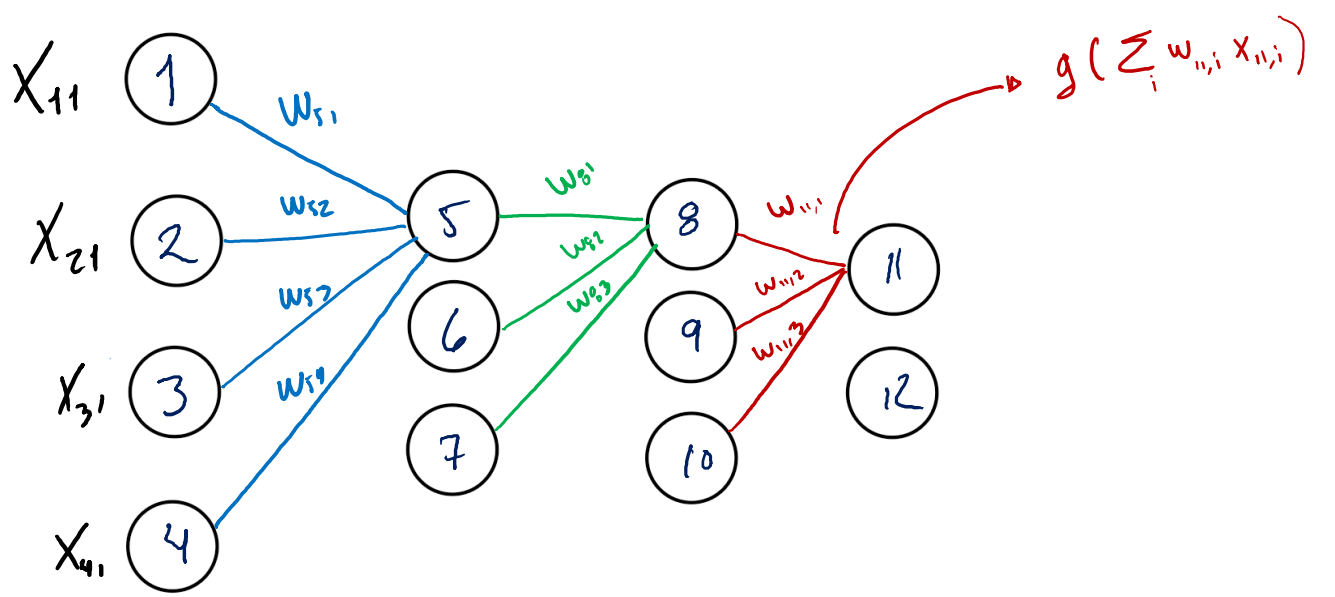
Neural networks – recap. (from perceptrons to DNNs)

- Since there are two types of units, we have to estimate the gradient for both cases:



Neural networks – recap. (from perceptrons to DNNs)

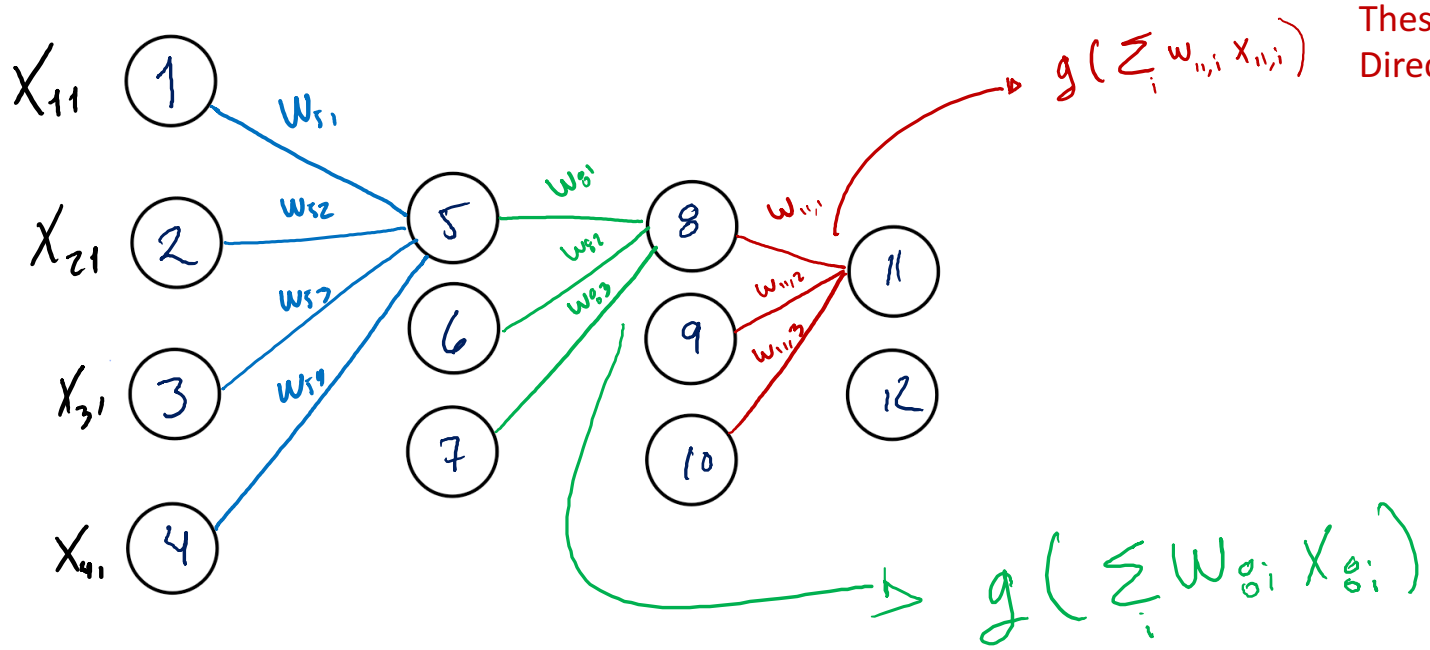
- Since there are two types of units, we have to estimate the gradient for both cases:



These can be compared
Directly with the target

Neural networks – recap. (from perceptrons to DNNs)

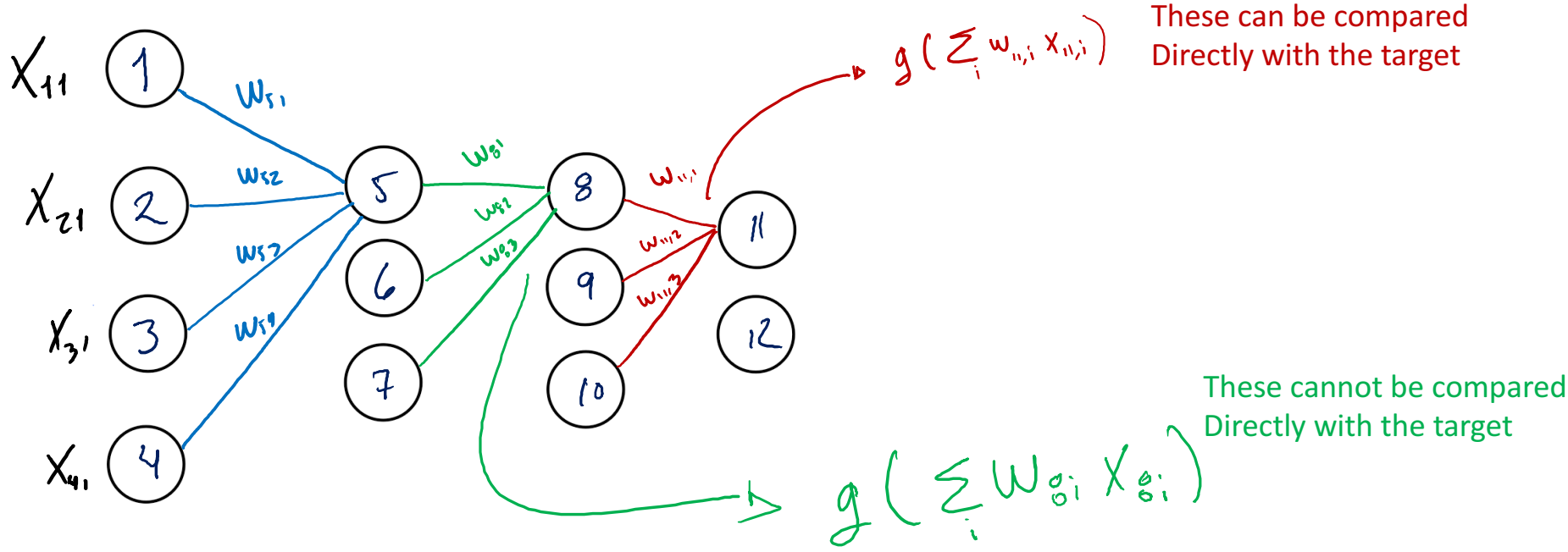
- Since there are two types of units, we have to estimate the gradient for both cases:



These can be compared
Directly with the target

Neural networks – recap. (from perceptrons to DNNs)

- Since there are two types of units, we have to estimate the gradient for both cases:



Neural networks – recap. (from perceptrons to DNNs)

- Output units, we aim to estimate: $\frac{\partial E_d}{\partial net_j}$ with $net_j = \sum_i w_{ji} x_{ji}$

- We notice net_j only influences the model via the outputs of the model

$$o_j = g(net_j)$$

then

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j}$$

Neural networks – recap. (from perceptrons to DNNs)

- Output units, we aim to estimate: $\frac{\partial E_d}{\partial net_j}$ with $net_j = \sum_i w_{ji} x_{ji}$

- We notice net_j only influences the model via the outputs of the model

$o_j = g(net_j)$

then

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in outs} (t_k - o_k)^2$$

$$= \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2$$

$$= \frac{1}{2} 2(t_j - o_j) \frac{\partial (t_j - o_j)}{\partial o_j}$$

$$= (t_j - o_j) - 1$$

Neural networks – recap. (from perceptrons to DNNs)

- Output units, we aim to estimate: $\frac{\partial E_d}{\partial net_j}$ with $net_j = \sum_i w_{ji} x_{ji}$

- We notice net_j only influences the model via the outputs of the model

$o_j = g(net_j)$

then

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j}$$

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial\, \sigma(net_j)}{\partial net_j} = \sigma(net_j)(1 - \sigma(net_j)) = o_j(1 - o_j)$$

Neural networks – recap. (from perceptrons to DNNs)

- Output units, we aim to estimate: $\frac{\partial E_d}{\partial net_j}$ with $net_j = \sum_i w_{ji} x_{ji}$

- We notice net_j only influences the model via the outputs of the model

$o_j = g(net_j)$

then

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j}$$

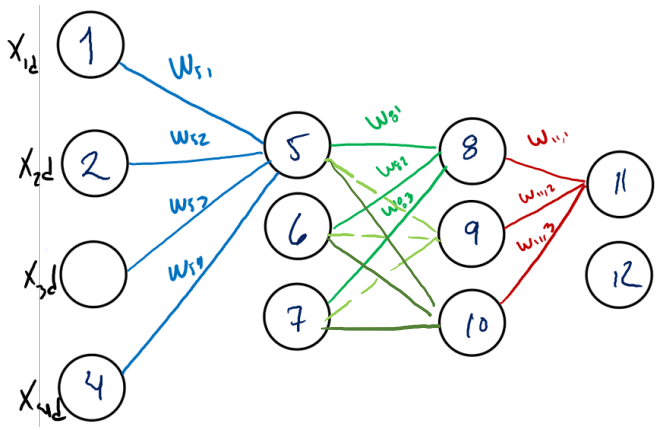
Substituting:

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j) o_j (1 - o_j)$$

$$\Rightarrow \Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta (t_j - o_j) o_j (1 - o_j) x_{ji}$$

Neural networks – recap. (from perceptrons to DNNs)

- For hidden units, we aim to estimate $\frac{\partial E_d}{\partial net_j}$
- We notice that hidden units only influence the model trough the units that have as input the output of the hidden unit

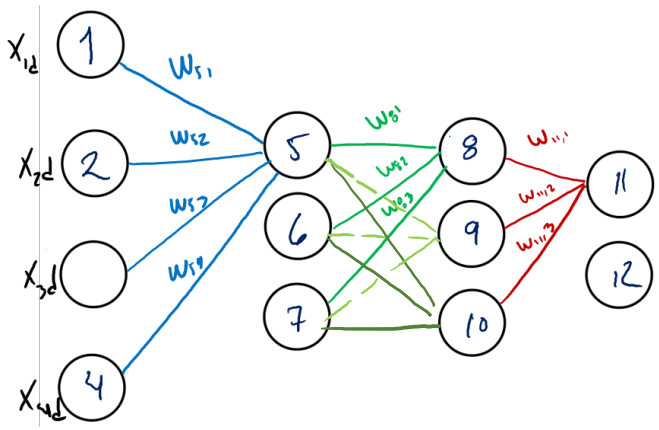


Let down(j) denote the set of units affected by j

Neural networks – recap. (from perceptrons to DNNs)

- For hidden units, we aim to estimate $\frac{\partial E_d}{\partial net_j}$

- We notice that hidden units only influence the model trough the units that have as input the output of the hidden unit



Let down(j) denote the set of units affected by j

$\rightarrow$ $down(5) = \{8, 9, 10\}$

Neural networks – recap. (from perceptrons to DNNs)

- For hidden units, we aim to estimate $\frac{\partial E_d}{\partial net_j}$
- We notice that hidden units only influence the model trough the units that have as input the output of the hidden unit

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in downs(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j}$$

Neural networks – recap. (from perceptrons to DNNs)

- For hidden units, we aim to estimate $\frac{\partial E_d}{\partial net_j}$
- We notice that hidden units only influence the model trough the units that have as input the output of the hidden unit

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in down(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j} \quad \text{s-a} \quad \delta_k = -\frac{\partial E_d}{\partial net_k}$$

Neural networks – recap. (from perceptrons to DNNs)

- For hidden units, we aim to estimate $\frac{\partial E_d}{\partial net_j}$

- We notice that hidden units only influence the model trough the units that have as input the output of the hidden unit

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in down(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j} \qquad \text{s-a} \quad \delta_k = -\frac{\partial E_d}{\partial net_k}$$

$$= \sum_{k \in down(j)} -\delta_k \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in down(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

Neural networks – recap. (from perceptrons to DNNs)

- For hidden units, we aim to estimate $\frac{\partial E_d}{\partial net_j}$

- We notice that hidden units only influence the model trough the units that have as input the output of the hidden unit

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in down(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in down(j)} -\delta_k \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in down(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

s.t. $\delta_k = -\frac{\partial E_d}{\partial net_k}$

$$= \sum_{k \in down(j)} -\delta_k w_{kj} \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in down(j)} -\delta_k w_{kj} o_j (1 - o_j)$$

Neural networks – recap. (from perceptrons to DNNs)

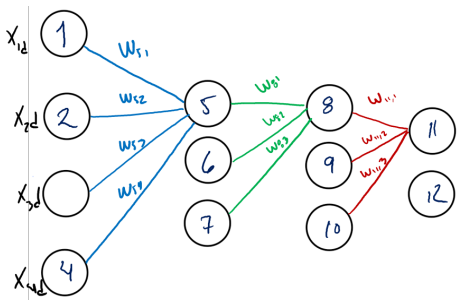
- For hidden units, we aim to estimate $\frac{\partial E_d}{\partial net_j}$
- We notice that hidden units only influence the model trough the units that have as input the output of the hidden unit

$$\delta_j = \frac{\partial E_d}{\partial net_j} = o_j(1-o_j)\sum_{k \in down(j)} \delta_k w_{kj}$$

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

Neural networks – recap. (from perceptrons to DNNs)

- The backprop + SGD algorithm for training a MLP



T. Mitchell. **Machine Learning**, McGrawHill 1997,

BACKPROPAGATION($training\_examples, \eta, n_{in}, n_{out}, n_{hidden}$)

Each training example is a pair of the form $\langle \vec{x}, \vec{t} \rangle$, *where* $\vec{x}$ *is the vector of network input values, and* $\vec{t}$ *is the vector of target network output values.*

η *is the learning rate (e.g., .05).* n_{in} *is the number of network inputs,* n_{hidden} *the number of units in the hidden layer, and* n_{out} *the number of output units.*

The input from unit i *into unit* j *is denoted* x_{ji}, *and the weight from unit* i *to unit* j *is denoted* w_{ji}.

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers (e.g., between $-.05$ and .05).
- Until the termination condition is met, Do
  - For each $\langle \vec{x}, \vec{t} \rangle$ in $training\_examples$, Do

Propagate the input forward through the network:

1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \tag{T4.3}$$

3. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k \tag{T4.4}$$

4. Update each network weight w_{ji}

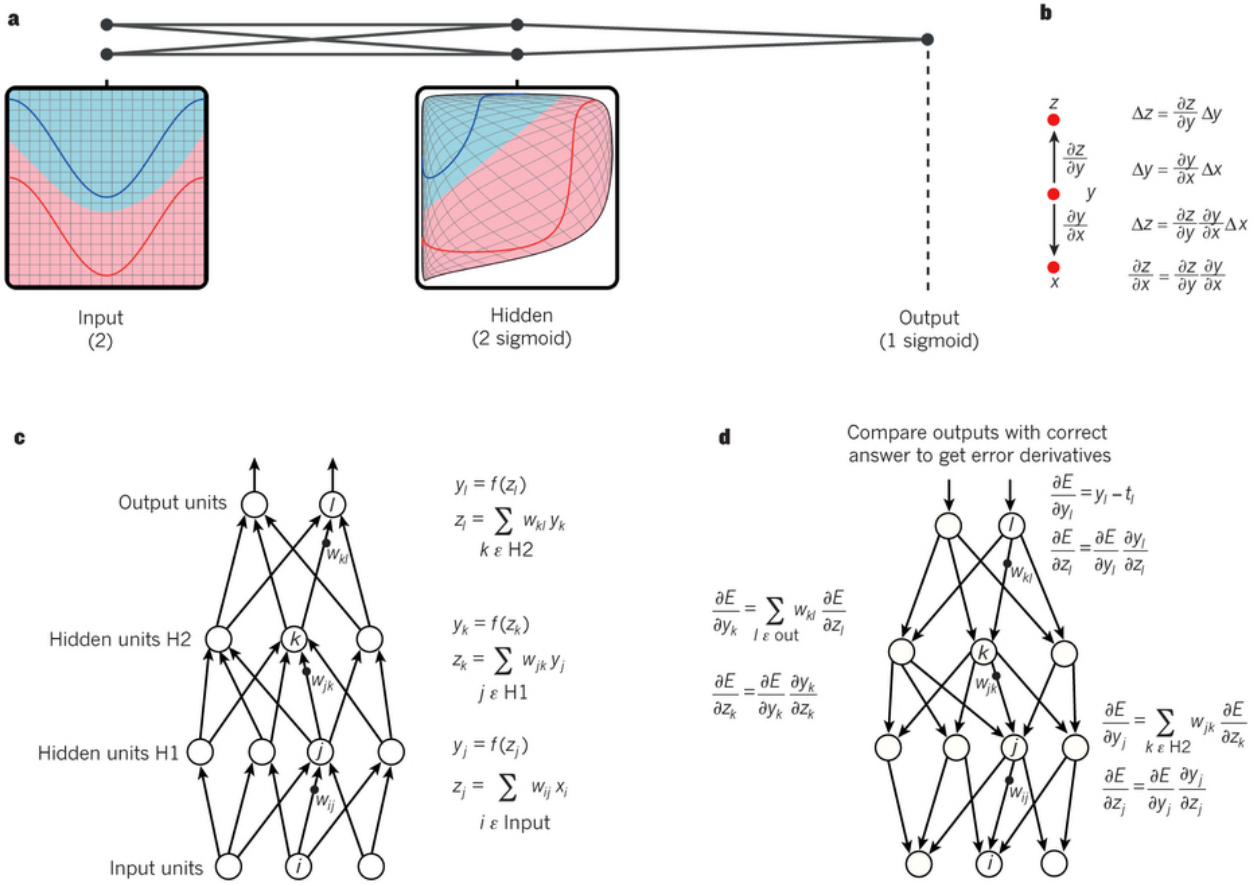
$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = \eta\, \delta_j\, x_{ji} \tag{T4.5}$$

Neural networks – recap. (from perceptrons to DNNs)

- The backprop + SGD algorithm for training a MLP

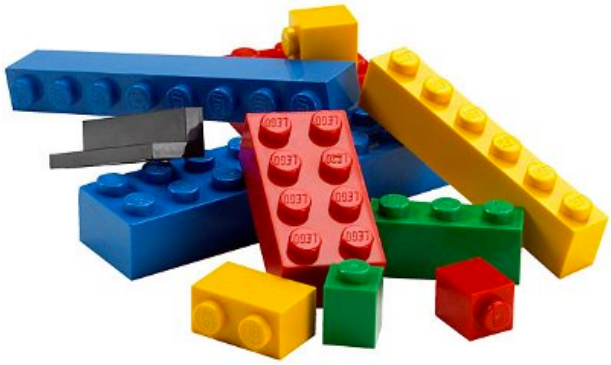


Deep neural networks

- DNNs are nothing but neural network models with several hidden layers (the number of layers indicating the depth of the model)
- They can be seen as composed functions of the form:
  - $f^n(f^{n-1}(f^{n-2}(\dots f^1(\mathbf{x}))))$
- The output layer is usually the direct link with the goal of the task
- Hidden/intermediate layers capture/learn particularities of data indirectly related to the goal
- They usually have many (millions) of parameters to learn/optimize, still backpropagation +SGD (++ a bunch of hints) make learning feasible

Deep neural networks

- Components of a DNN / design choices
  - Architecture
  - Output units
  - Hidden units
  - Learning strategy
  - Cost function



DNNs – output units

- The form of output units depends on the goal the DNN has to solve (Output units can serve as hidden units as well)

- We will study three types of output units
  - Linear units for Gaussian output distributions
  - Sigmoid units for Bernoulli output distributions
  - Softmax units for Multinomial output distributions

- Assuming their goal is to take as input the output of features from a hidden unit and transform them to complete the task associated to the DNN
  - $\mathbf{h} = f(\mathbf{x}; \Theta)$

DNNs – output units

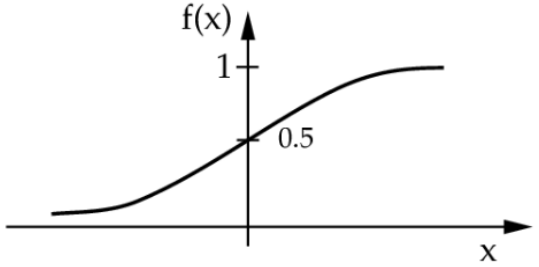
- Linear units for Gaussian output distributions

- Perform an affine transformation with no linearity
  - $\mathbf{y}^* = \mathbf{W}\mathbf{h} + b$

- They are commonly used to produce the mean of a conditional Gaussian distribution
  - $p(\mathbf{y}|\mathbf{x}) = \mathrm{N}(\mathbf{y}; \mathbf{y}^*, \mathrm{I})$

DNNs – output units

- Sigmoid units for Bernoulli output distributions
  - $y^* = \sigma(w\boldsymbol{h} + b)$
  - With: $\sigma(\mathrm{z}) = \frac{1}{1+exp^{-z}}$



- Ideal units when one wants to predict the value of a binary variable y (e.g., binary classification problems)

DNNs – output units

- Softmax units for Multinomial output distributions

- Used to represent a probability distribution over a discrete variable with k possible values (e.g., the output of a multiclass classifier)

- First, a linear layer predicts un-normalized log probabilities
  - $\boldsymbol{z} = \mathbf{W}\mathbf{h} + b, \quad where \quad z_i = \log P^{\sim}(y = i|\mathbf{x})$
  - Then: $softmax(\boldsymbol{z})_i = \frac{\exp(z_j)}{\sum_j \exp(z_j)}$
  - So, we have as output a vector of k *probabilities*

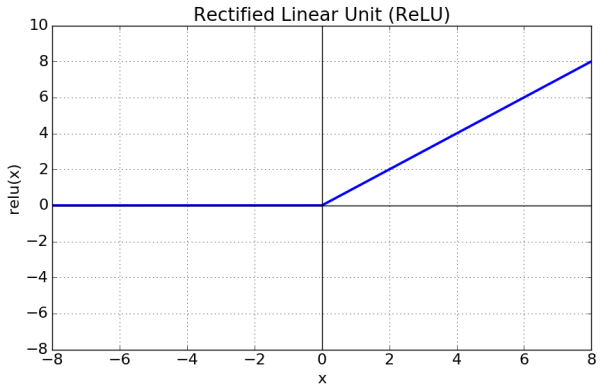
DNNs – hidden units

- The selection of hidden units is not straightforward, yet, we will review the most common ones
  - ReLU – Rectified linear units
  - Logistic and hyperbolic tangent

- We assume hidden units take as input a vector of inputs **x** and perform an affine transformation and a nonlinear transformation that is used as input by another unit
  - $\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$

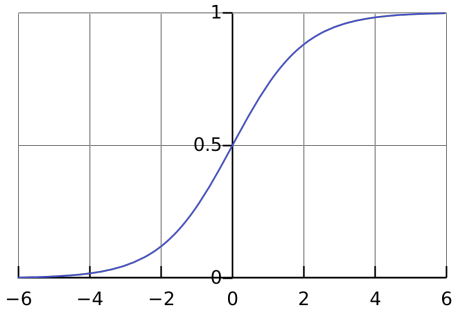
DNNs – hidden units

- ReLU – Rectified linear units (the de facto hidden unit in the renaissance era)
  - $g(z) = \max\{0, z\}, \quad so \quad \mathbf{h} = \max\{0, (\mathbf{W}\mathbf{x} + \mathbf{b})\}$
- Very similar to linear units (except it outputs 0 across half its domain)
- Derivatives are large, whenever it is active
- Several variants/improvements (yet these yield little improvements)

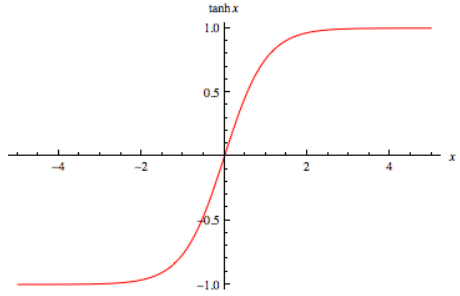


DNNs – hidden units

- Logistic and hyperbolic tangent units: prior to ReLU, most NNs used these units
  - $g(z) = \sigma(\mathrm{z}) = \frac{1}{1+exp^{-z}}$
  - $g(z) = thanh(z)$



- Their usage is discouraged nowadays, because they saturate with extreme values



DNNs – further aspects / considerations

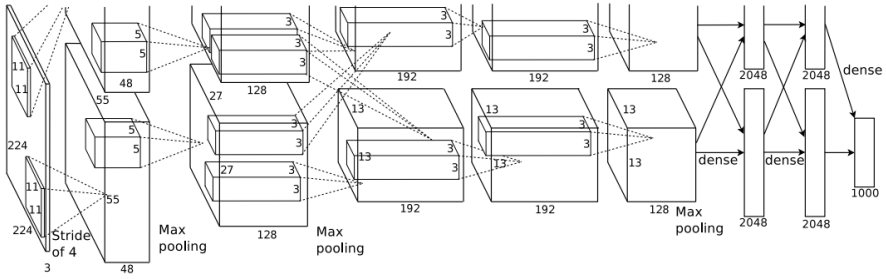
- **Architecture design:** an art! (stacked layers, reducing the number of units each layer is standard, the deeper the better, usually, to some extend)
- **Training DNNs:** Backpropagation with SGD is the common choice
- **Regularization:** Dropout, adversarial training, early stopping,
- **Implementation:** GPU – computing is necessary

Deep learning variants

- Main DL models:
  - Deep neural networks (DNNs, MLPs)
  - Convolutional neural networks (CNNs)
  - LSTM
  - Restricted Boltzman machines
  - Deep belief networks
  - Autoencoders
- New paradigms
  - Residual DNNs
  - Gated recurrent NNs
  - Generative adversarial networks

Convolutional neural networks

- Type of neural network for processing data having grid-like topology
  - Time series (1D grid)
  - Images (2D grid)
  - Video (3D grid)
- Components: convolutional layers, activation of units, pooling layers,
- Weights are learned with backpropagation



Convolutional neural networks

- These networks employ the mathematical operation called convolution
  - CNNs are simply NNs that use convolution in place of matrix multiplication in at least one of their layers

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau$$
$$= \int_{-\infty}^{\infty} f(t-\tau)\, g(\tau)\, d\tau.$$

$$s(t) = \int x(a) w(t-a) da$$

$$s(t) = (x * w)(t)$$

Convolutional neural networks

- Convolution in NN terminology:
  - x – the input
  - w – the kernel
  - s – the feature map

$$s(t) = (x * w)(t)$$

$$s(t) = \int x(a)w(t-a)da$$

- Discrete convolution:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

Convolutional neural networks

- 2D convolution

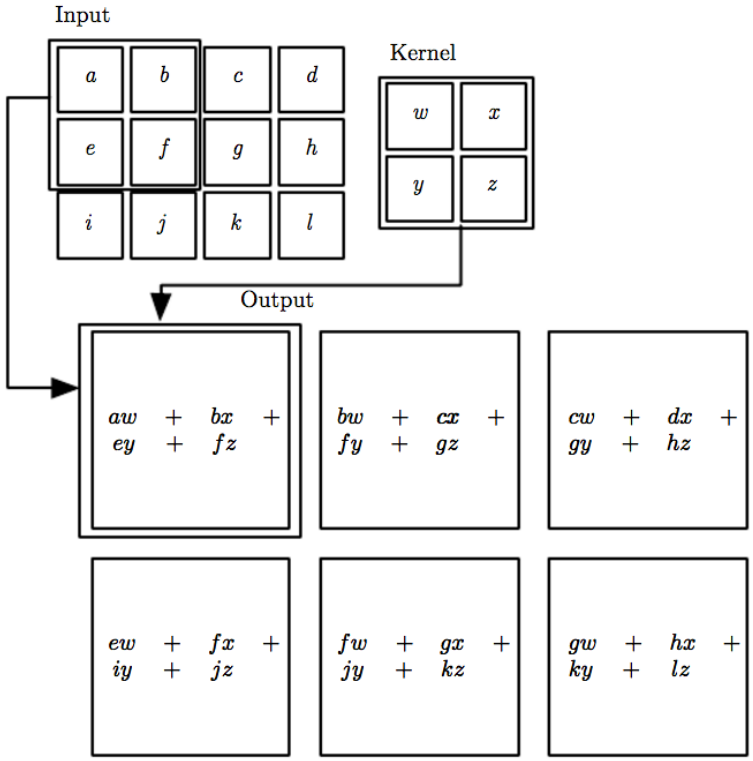
$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).$$

- Convolution is commutative

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n).$$

Convolutional neural networks

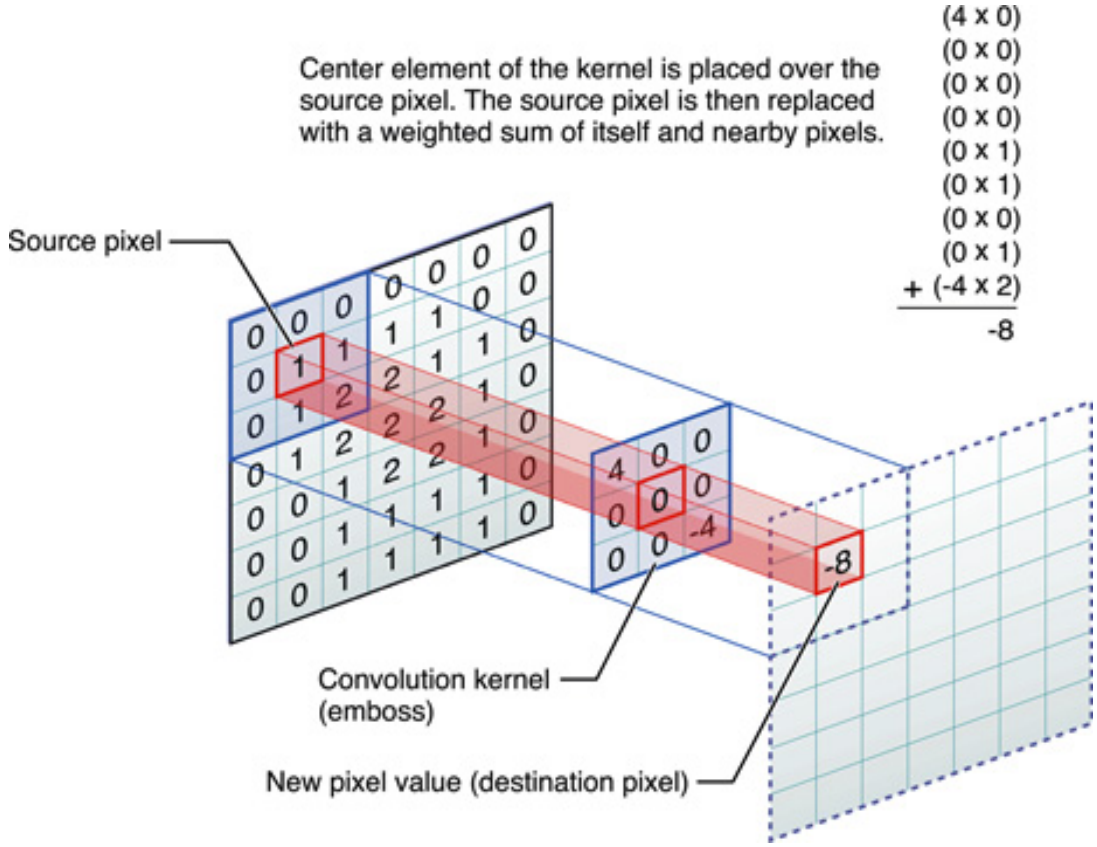
- 2D convolution



Convolutional neural networks

- 2D convolution

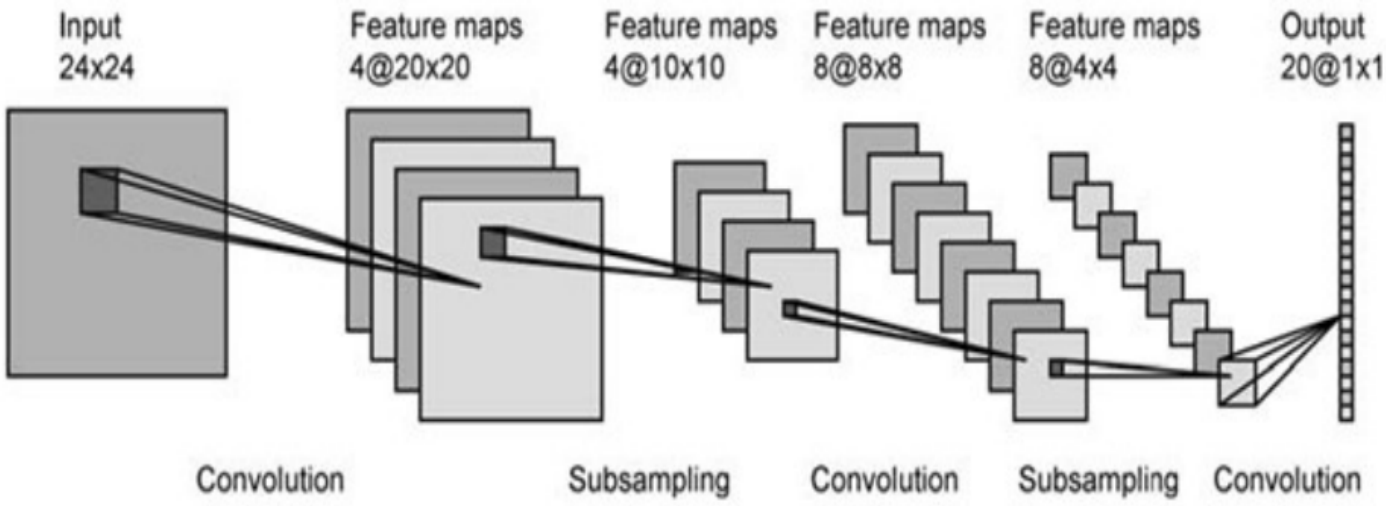
-2	-2	0
-2	6	0
0	0	0



https://developer.apple.com/library/content/documentation/Performance/Conceptual/vImage/ConvolutionOperations/ConvolutionOperations.html

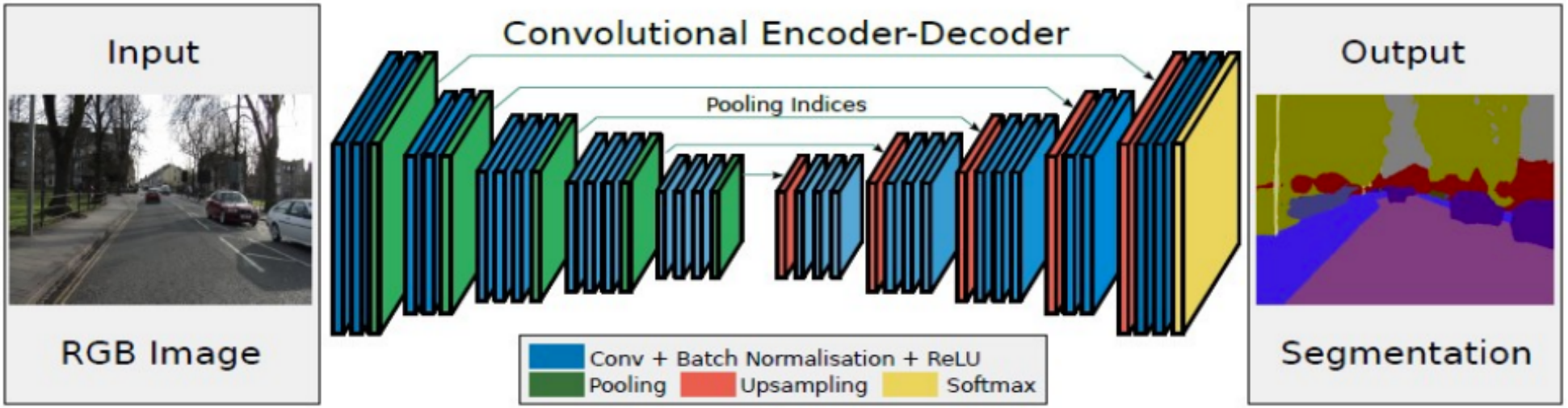
Convolutional neural networks

- Typical architecture I



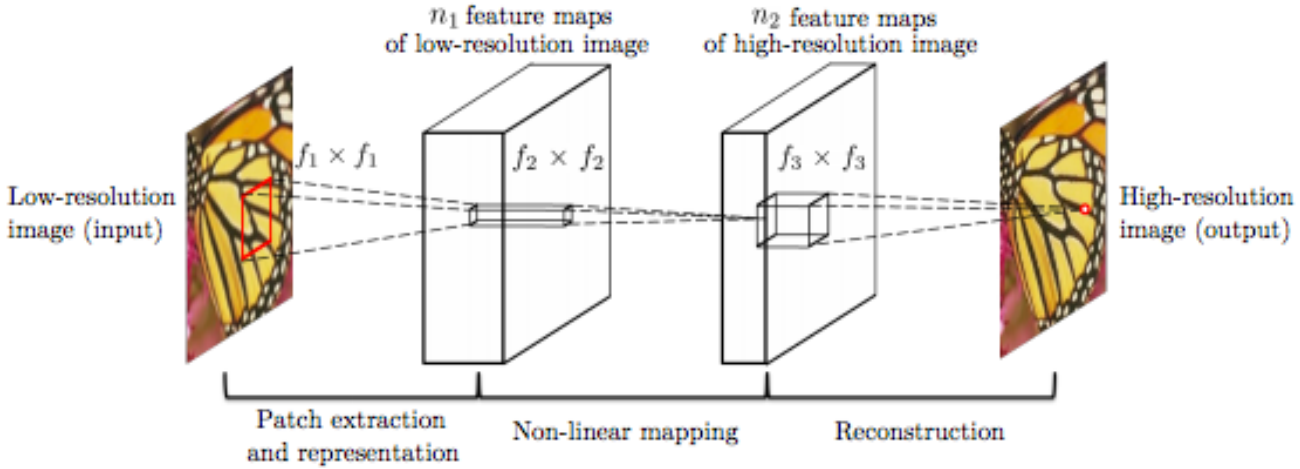
Convolutional neural networks

- Typical architecture II



Convolutional neural networks

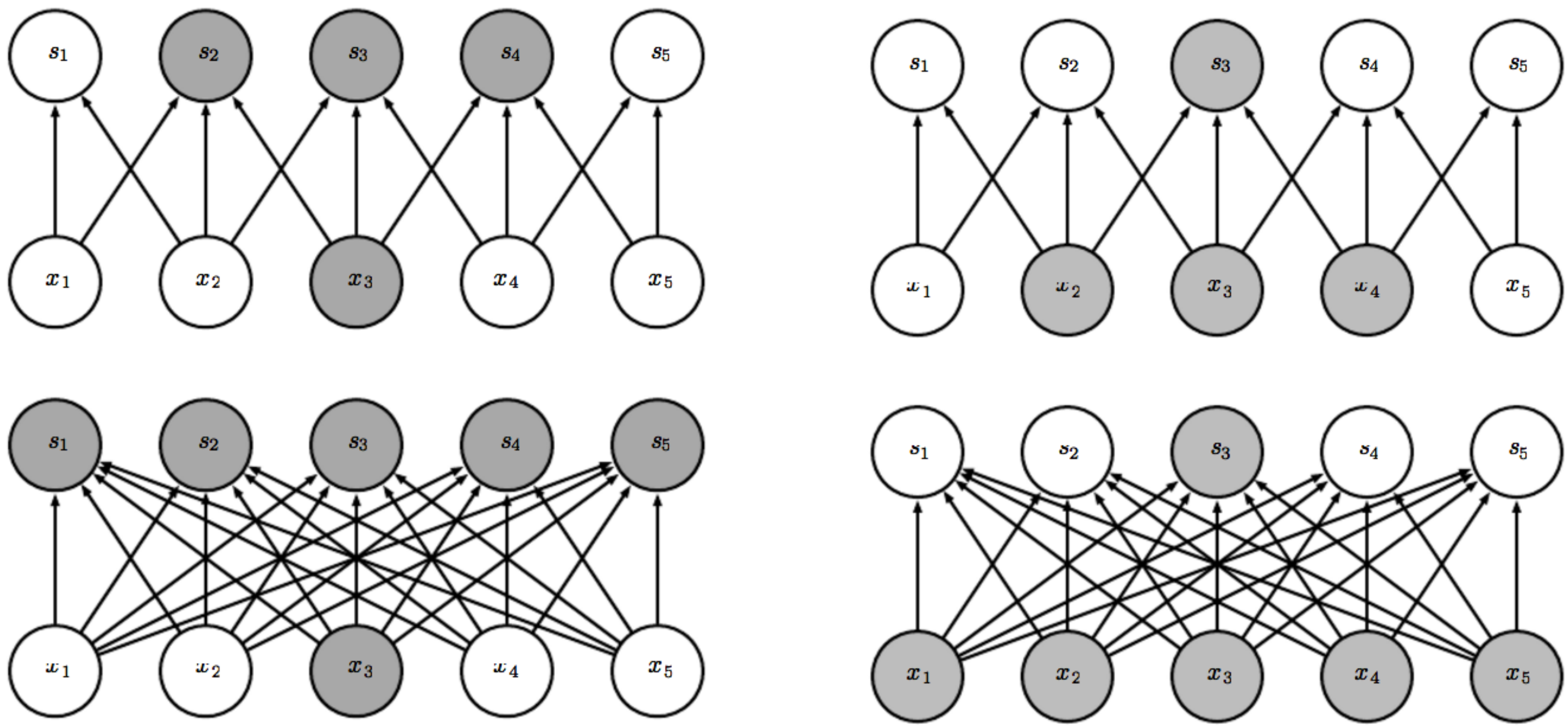
- Typical architecture II



C. Dong, C. C. Loy, K. He, and X. Tang, "**Image super-resolution using deep convolutional networks**," CoRR, vol. abs/1501.00092, 2015.

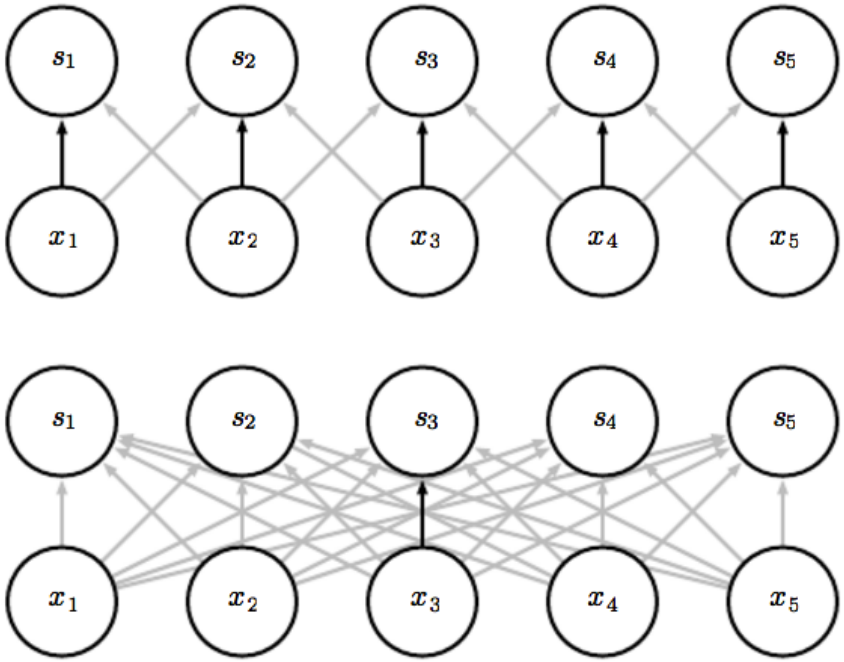
Convolutional neural networks

- Why convolution layers?
  - Sparse connectivity: less weights/parameters



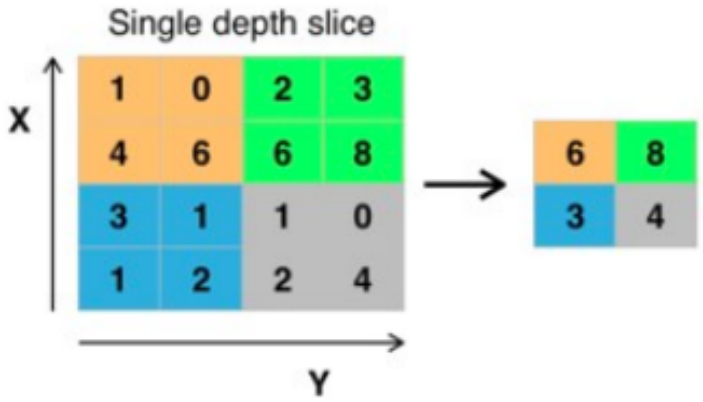
Convolutional neural networks

- Why convolution layers?
  - Parameter sharing: less weights/parameters



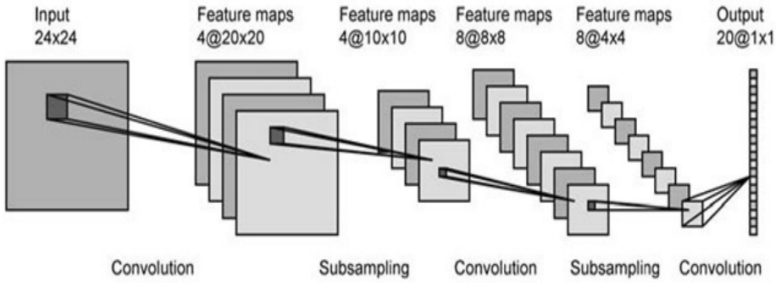
Convolutional neural networks

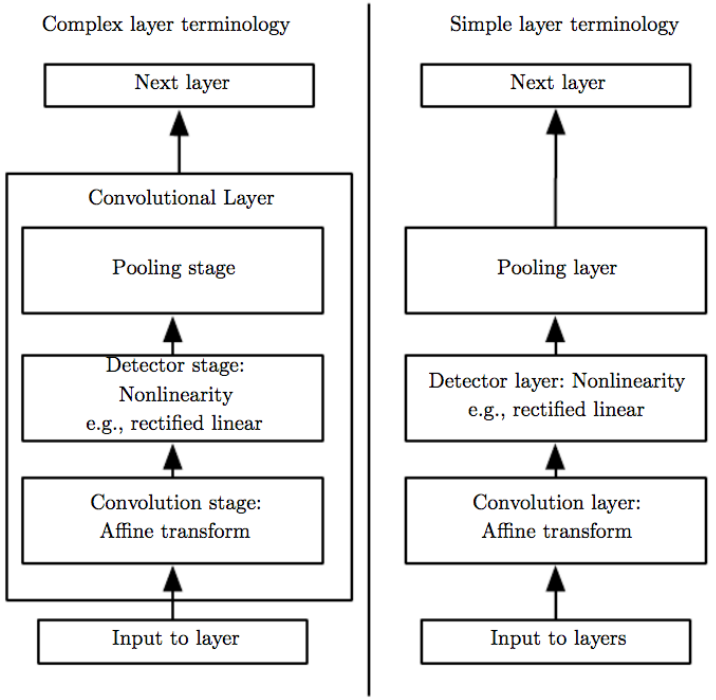
- **Pooling:** replaces the output of a net at a certain location with a summary statistic of nearby outputs
  - Makes aprox. Invariant to translation
  - Variants include: max, sum, avg. poolings
  - Usually, stride is considered



Convolutional neural networks

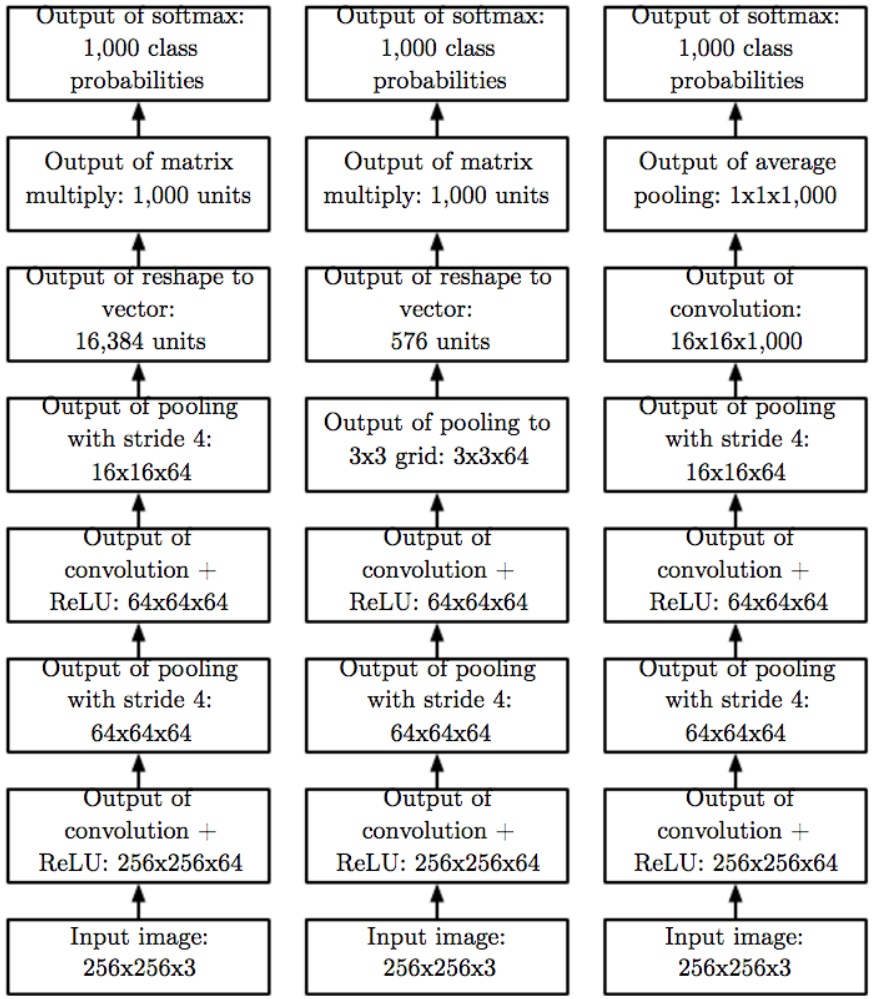
- Typical architecture of a layer



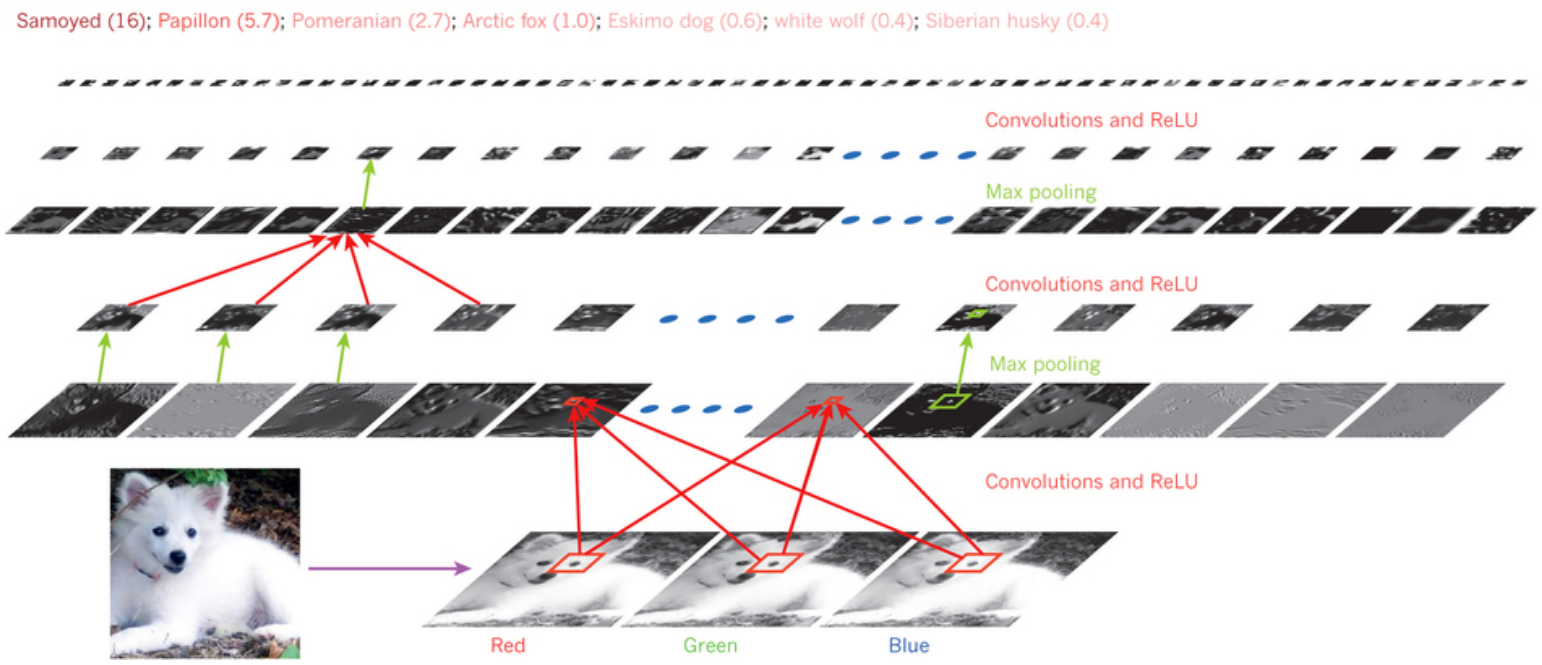


Convolutional neural networks

- Common architectures

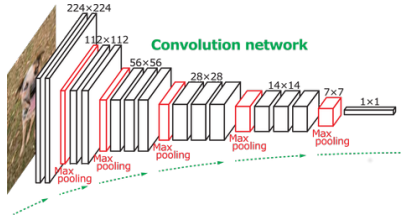


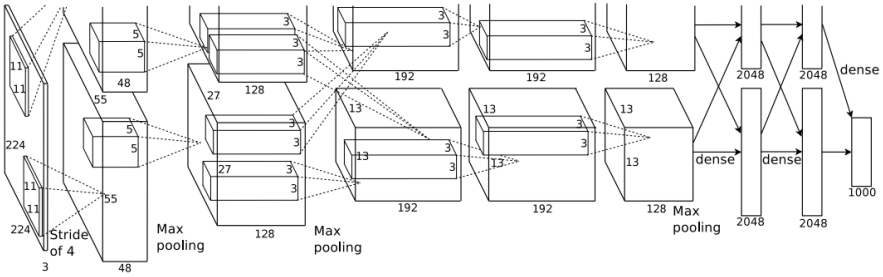
Convolutional neural networks

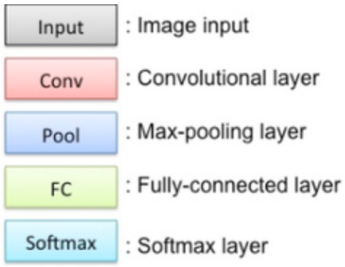


Deep learning in a nutshell

- Going deeper (CNNs)

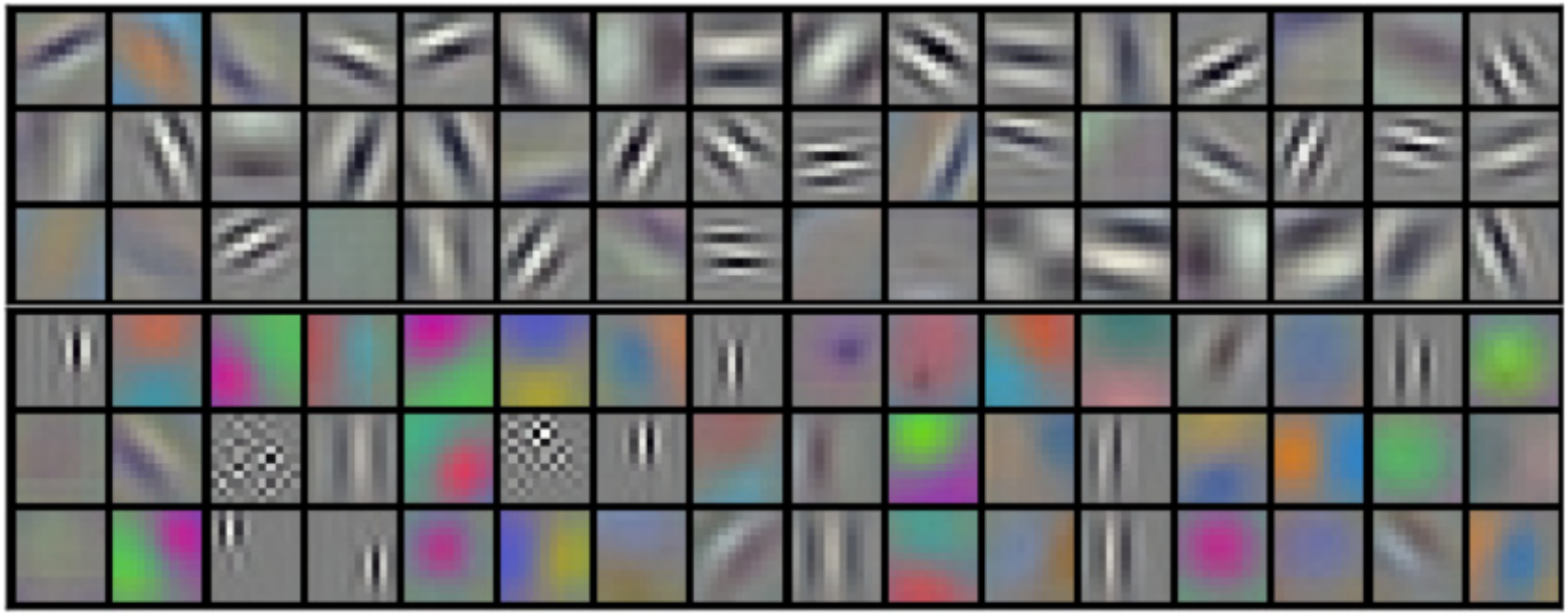






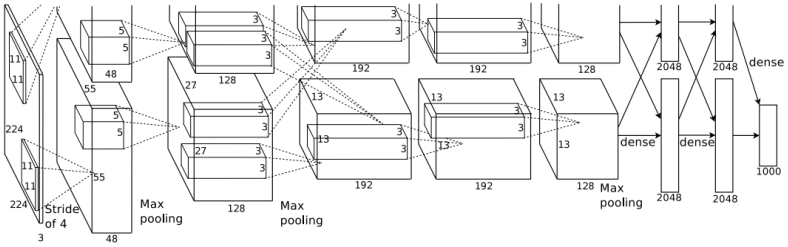
Convolutional neural networks

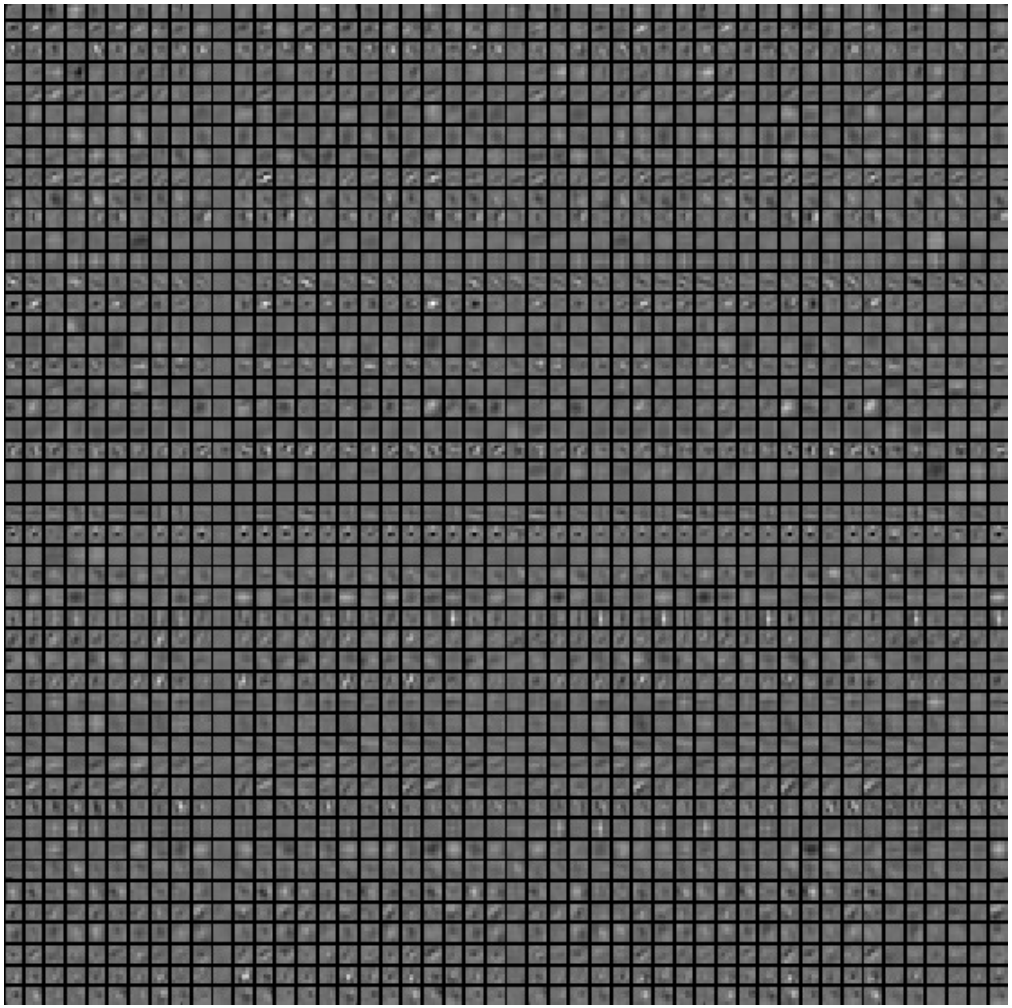
- How do the filters look like?



Convolutional neural networks

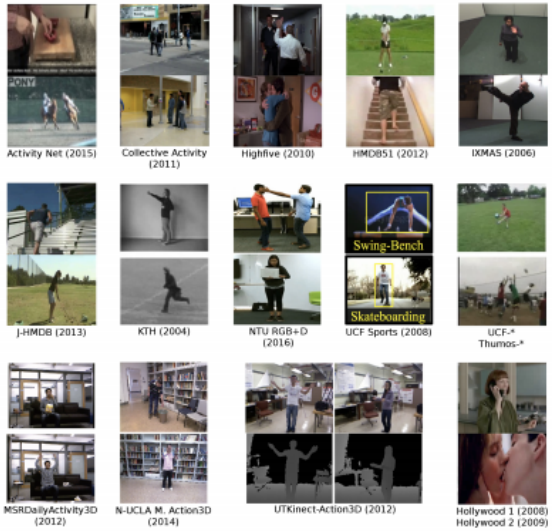
- How do the filters look like?

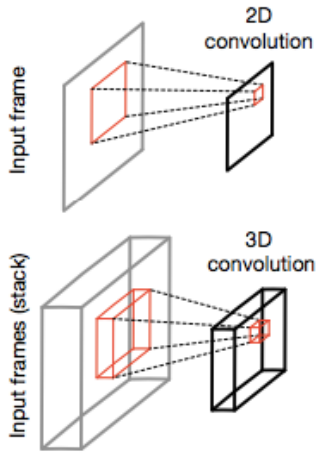


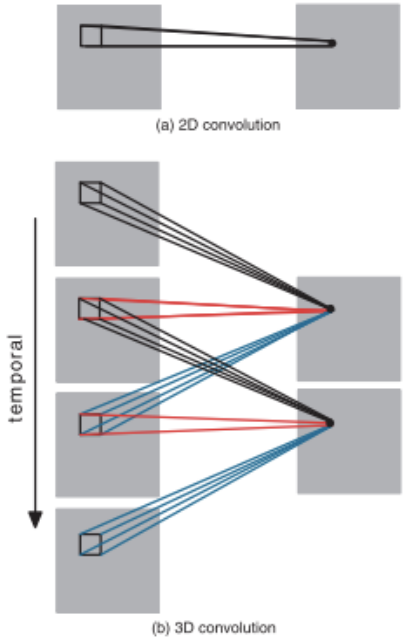


Convolutional neural networks

- What about 3D?



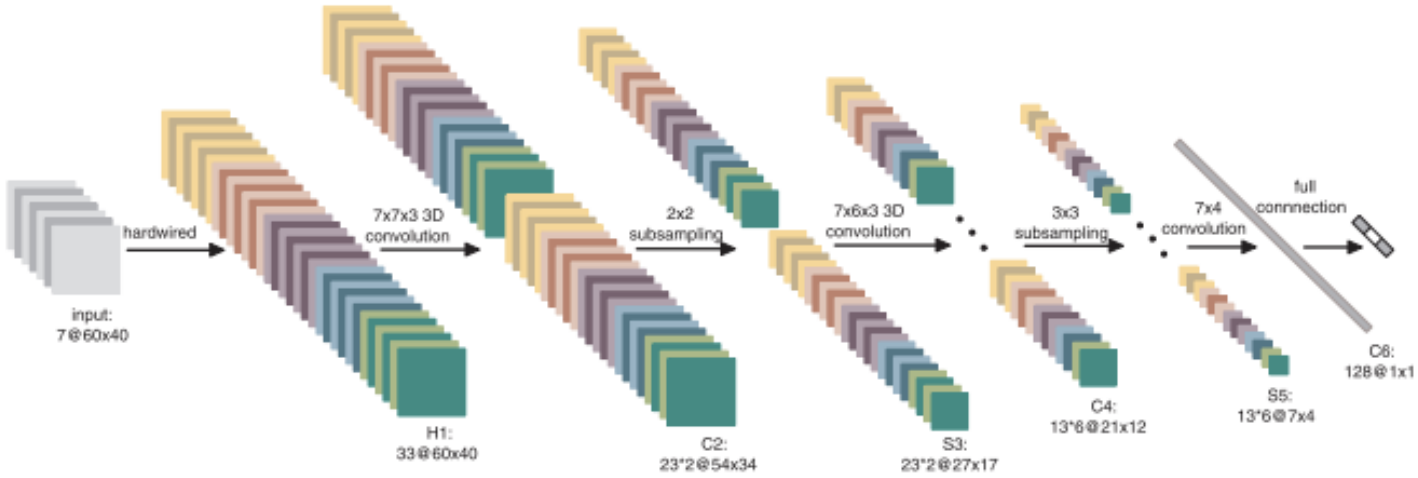




Maryam Asadi-Aghbolaghi, Albert Clapés, Marco Bellantonio, Hugo Jair Escalante, Víctor Ponce-López, Xavier Baró, Isabelle Guyon, Shohreh Kasaei, Sergio Escalera. **A survey on deep learning based approaches for action and gesture recognition in image sequences.** Proceedings of the 12th IEEE Conference on Automatic Face and Gesture Recognition, 2017

Convolutional neural networks

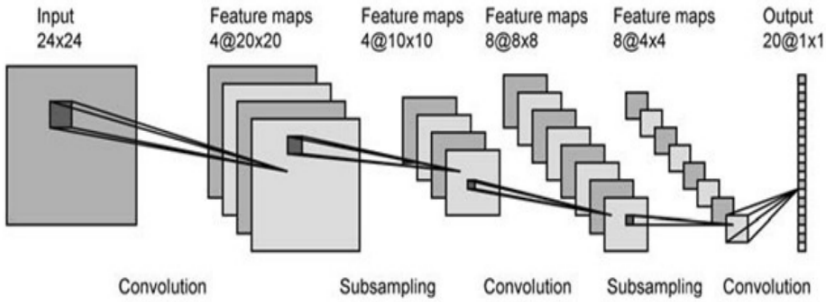
- What about 3D?

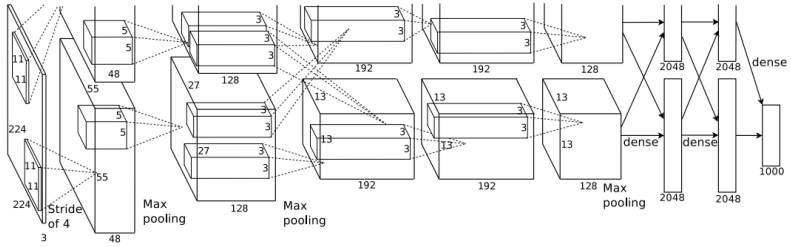


Shuiwang Ji, Wei Xu, Ming Yang, Kai Yu. **3D Convolutional Neural Networks for Human Action Recognition.** TPAMI Vol. 35(1):221-231, 2013

Convolutional neural networks

- **Transfer learning:** it is common to re-use pretrained architectures (the weights) trained with millions of images
  - Direct use: use them as feature extractors
  - Tailored models: re-use some layers and perform a fine tuning for layers of interest

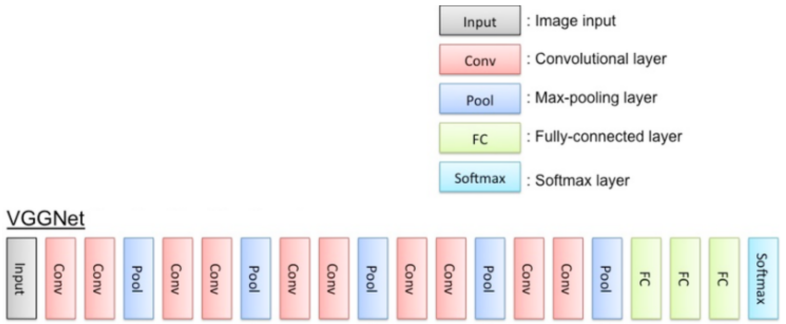


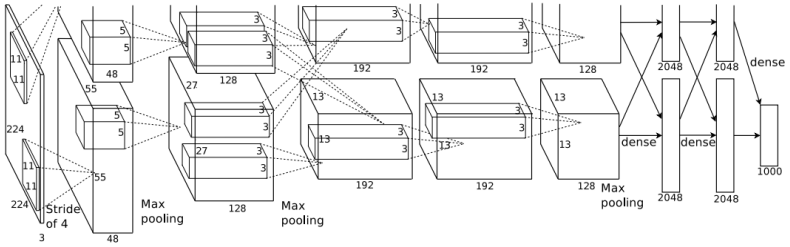


Yosinski J, Clune J, Bengio Y, and Lipson H. **How transferable are features in deep neural networks?** In Advances in Neural Information Processing Systems 27 (NIPS '14), NIPS Foundation, 2014

Convolutional neural networks

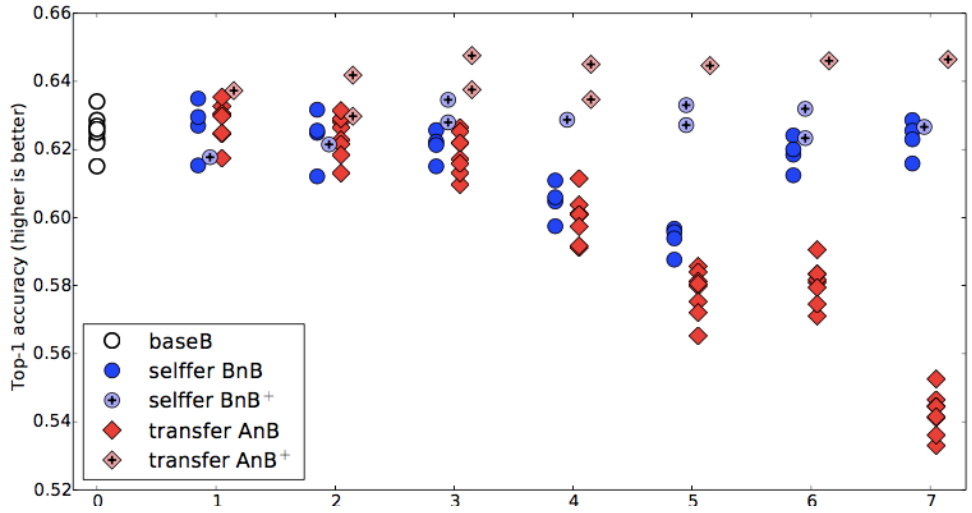
- Commonly used pretrained-CNNs
  - AlexNet
  - VGG
  - GoogleNet
  - PlacesNet
  - FaceNet





Convolutional neural networks

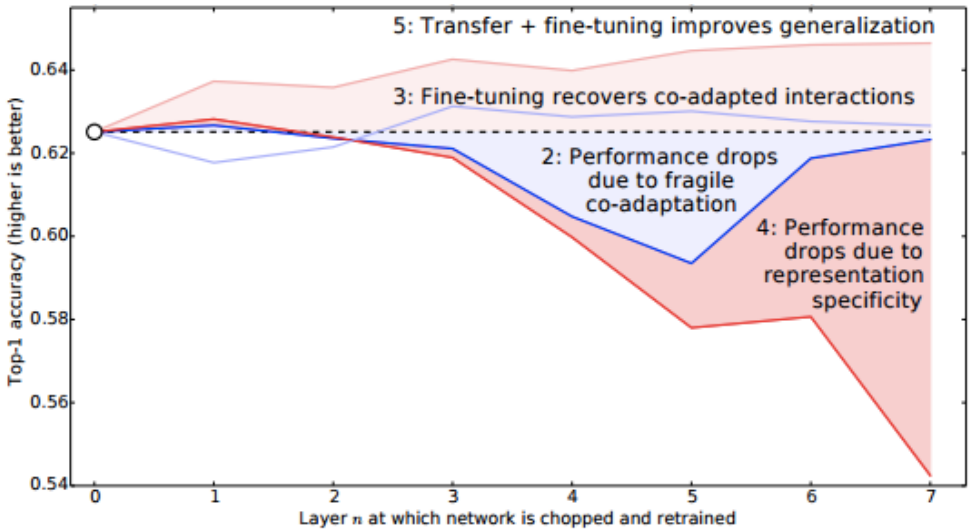
- **Transfer learning:** it is common to re-use pretrained architectures (the weights) trained with millions of images



Yosinski J, Clune J, Bengio Y, and Lipson H. **How transferable are features in deep neural networks?** In Advances in Neural Information Processing Systems 27 (NIPS '14), NIPS Foundation, 2014

Convolutional neural networks

- **Transfer learning:** it is common to re-use pretrained architectures (the weights) trained with millions of images



Yosinski J, Clune J, Bengio Y, and Lipson H. **How transferable are features in deep neural networks?** In Advances in Neural Information Processing Systems 27 (NIPS '14), NIPS Foundation, 2014

Convolutional neural networks

- The ruling models for CV&PR
  - (only since 2012, even when Lecun sucessfully used CNNs for digit recognition in the 80s)
- Outstanding results in a number of tasks, domains, data sets
- Designing a CNN is an art, lots of tricks, improvements, modifications can be performed (deep learning engineering)
- Models become obsolete extremely fast, it is difficult to be aware of SOTA
- Conferences on CNNs: CVPR, ICCV, ECCV, NIPS….

Deep learning variants

- Main DL models:
  - Deep neural networks (DNNs, MLPs)
  - Convolutional neural networks (CNNs)
  - LSTM
  - Restricted Boltzman machines
  - Deep belief networks
  - Autoencoders

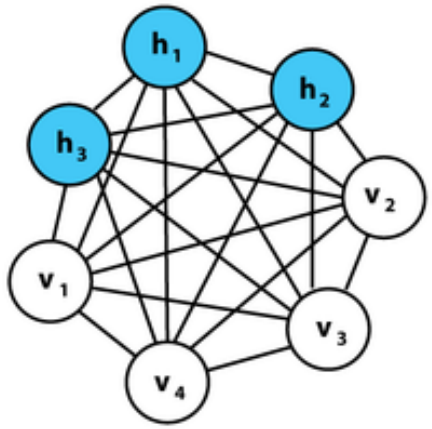
Deep generative models

- New paradigms
  - Residual DNNs
  - Gated recurrent NNs
  - Generative adversarial networks

Deep generative models

- Boltzmann machines. An energy based model defined over a d-dimensional random binary vector $\mathbf{x} \in \{0,1\}^d$
- The joint probability distribution is given by:

$$P(\boldsymbol{x}) = \frac{\exp\left(-E(\boldsymbol{x})\right)}{Z},$$



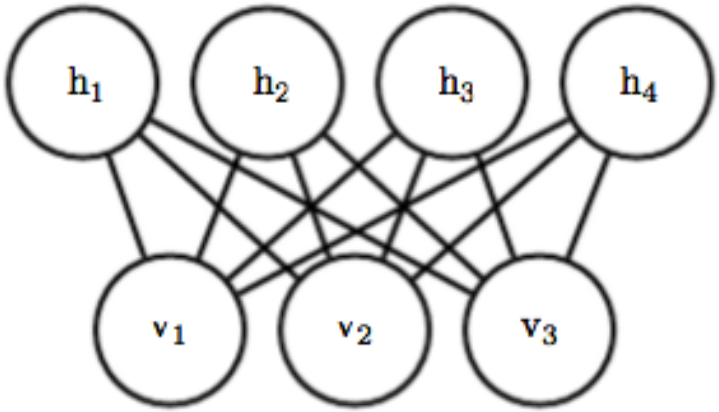
- where

Only visible units $E(\boldsymbol{x}) = -\boldsymbol{x}^\top \boldsymbol{U}\boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{x},$

Hidden and visible units $E(\boldsymbol{v}, \boldsymbol{h}) = -\boldsymbol{v}^\top \boldsymbol{R}\boldsymbol{v} - \boldsymbol{v}^\top \boldsymbol{W}\boldsymbol{h} - \boldsymbol{h}^\top \boldsymbol{S}\boldsymbol{h} - \boldsymbol{b}^\top \boldsymbol{v} - \boldsymbol{c}^\top \boldsymbol{h}.$

Deep generative models

- Restricted Boltzmann machines. A Boltzmann machine restricted to:
  - Have two layers: one of visible and other of hidden units
  - There are no connections between units in the same layer
  - Units are typically Bernoulli activation functions
- The building block for deep generative models



Deep generative models

- An RBM is an energy-based model, with the joint probability specified by its energy function as follows:

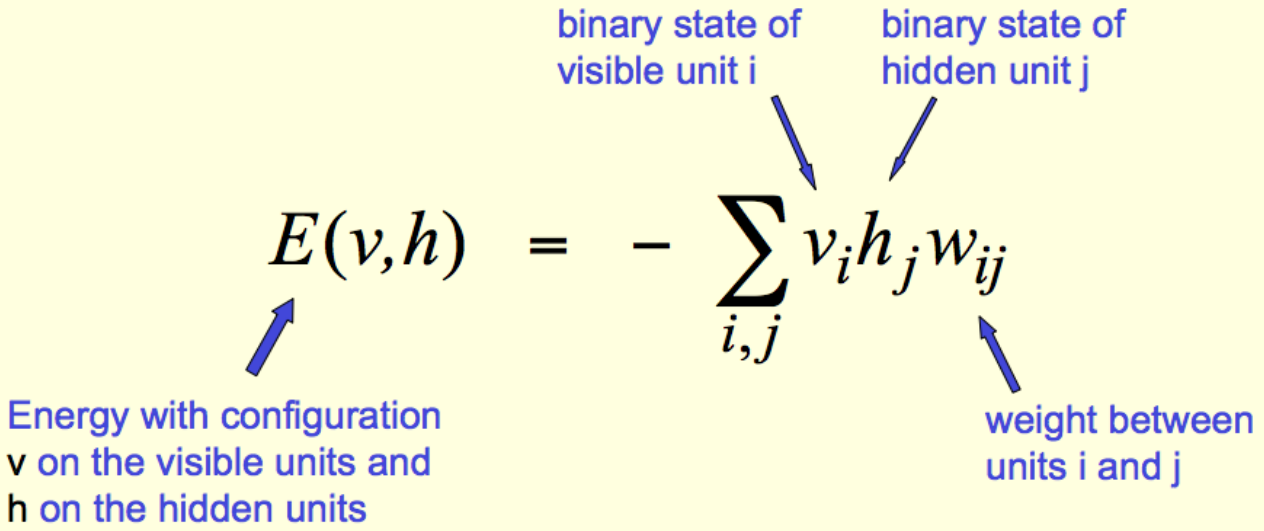
$$P(\mathbf{v} = \boldsymbol{v}, \mathbf{h} = \boldsymbol{h}) = \frac{1}{Z} \exp\left(-E(\boldsymbol{v}, \boldsymbol{h})\right).$$

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\boldsymbol{b}^\top \boldsymbol{v} - \boldsymbol{c}^\top \boldsymbol{h} - \boldsymbol{v}^\top \boldsymbol{W} \boldsymbol{h},$$

$$Z = \sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} \exp\left\{-E(\boldsymbol{v}, \boldsymbol{h})\right\}.$$

The Energy of a joint configuration

(ignoring terms to do with biases)



$$-\frac{\partial E(v,h)}{\partial w_{ij}} = v_i h_j$$

Slide from G. Hinton's tutorial at NIPS 2007

Deep generative models

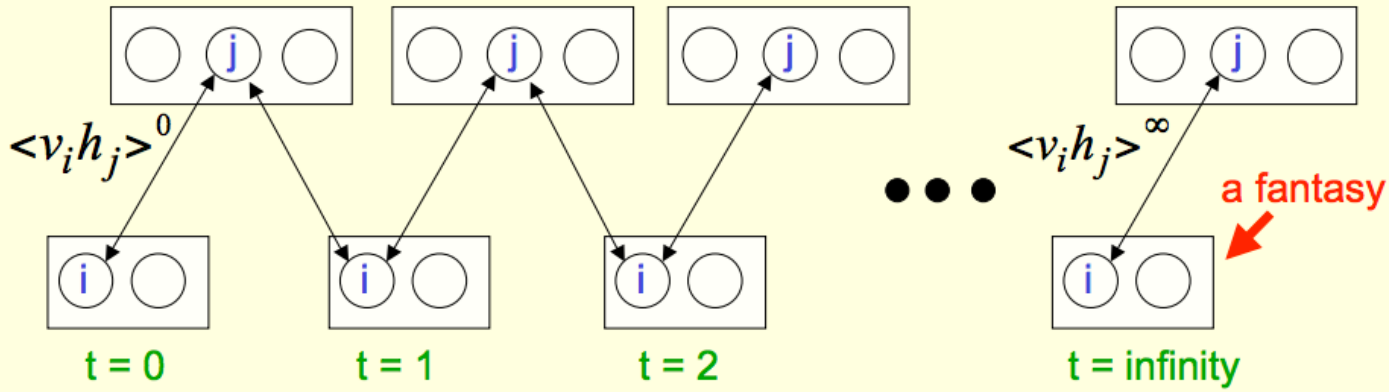
- The bipartite graph structure of the RBM has the special property that its conditional distributions $P(\mathbf{h}|\mathbf{v})$ and $P(\mathbf{v}|\mathbf{h})$ are factorial and simple to compute / sample from:

$$p(\mathbf{h} \mid \mathbf{v}) = \Pi_i p(\mathrm{h}_i \mid \mathbf{v}) \qquad p(\mathbf{v} \mid \mathbf{h}) = \Pi_i p(\mathrm{v}_i \mid \mathbf{h}).$$

$$p(h_j = 1|\mathbf{v};\theta) = \sigma\left(\sum_{i=1}^{I} w_{ij} v_i + a_j\right)$$

$$p(v_i = 1|\mathbf{h};\theta) = \sigma\left(\sum_{j=1}^{J} w_{ij} h_i + b_i\right)$$

A picture of the maximum likelihood learning algorithm for an RBM



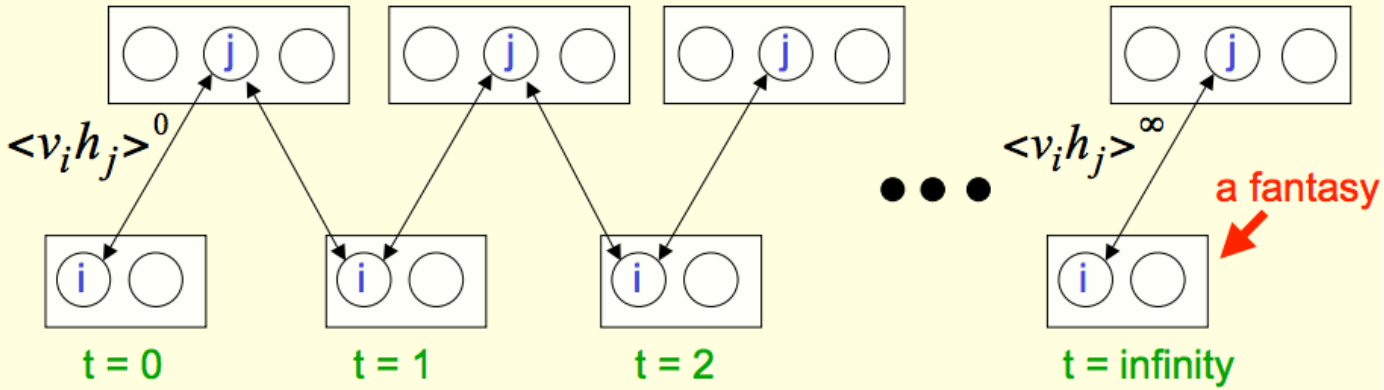
Start with a training vector on the visible units.

Then alternate between updating all the hidden units in parallel and updating all the visible units in parallel.

$$\frac{\partial \log p(v)}{\partial w_{ii}} = <v_i h_j>^0 - <v_i h_j>^\infty$$

Slide from G. Hinton's tutorial at NIPS 2007

A picture of the maximum likelihood learning algorithm for an RBM

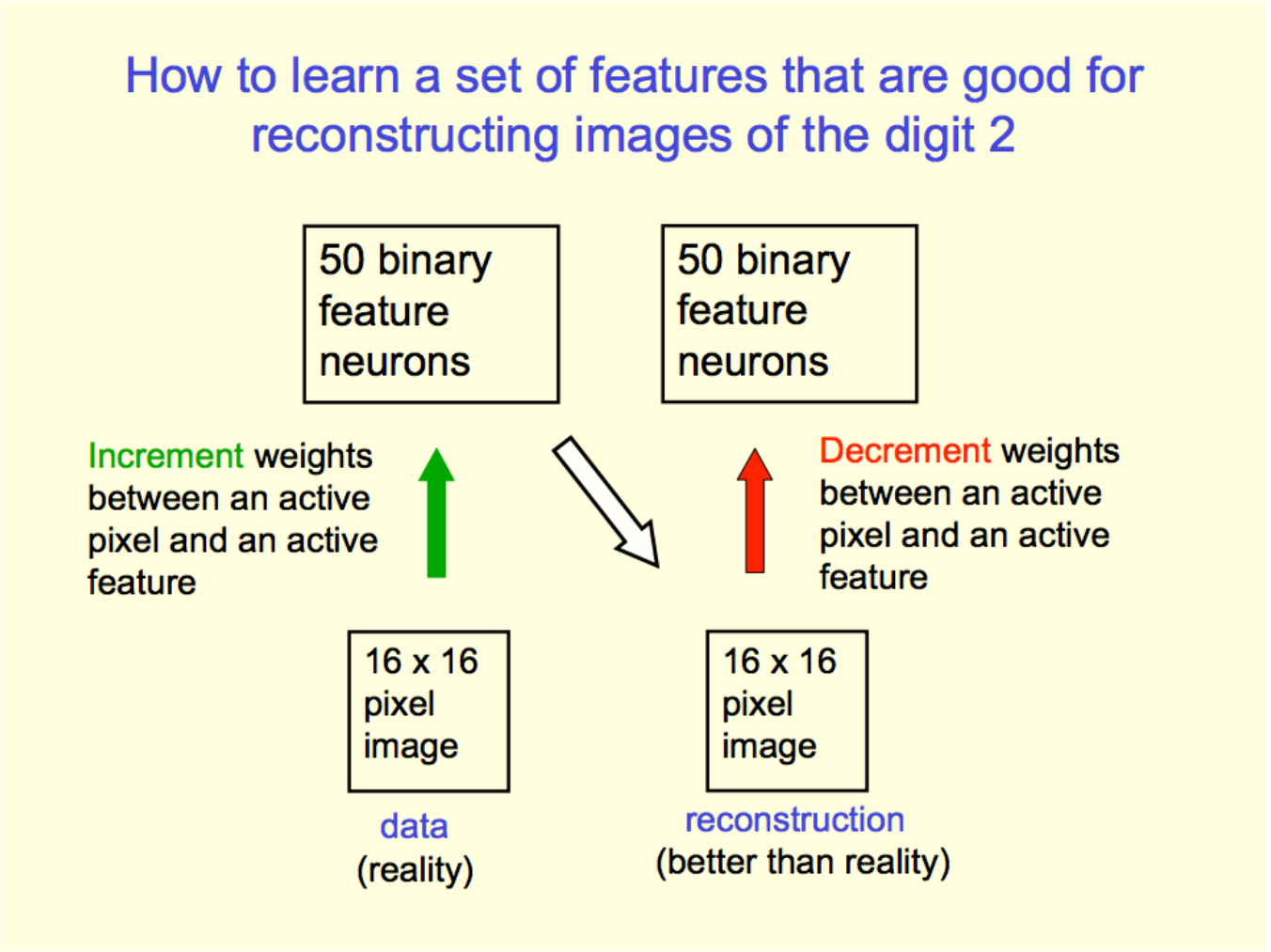


Start with a training vector on the visible units.

Then alternate between updating all the hidden units in parallel and updating all the visible units in parallel.

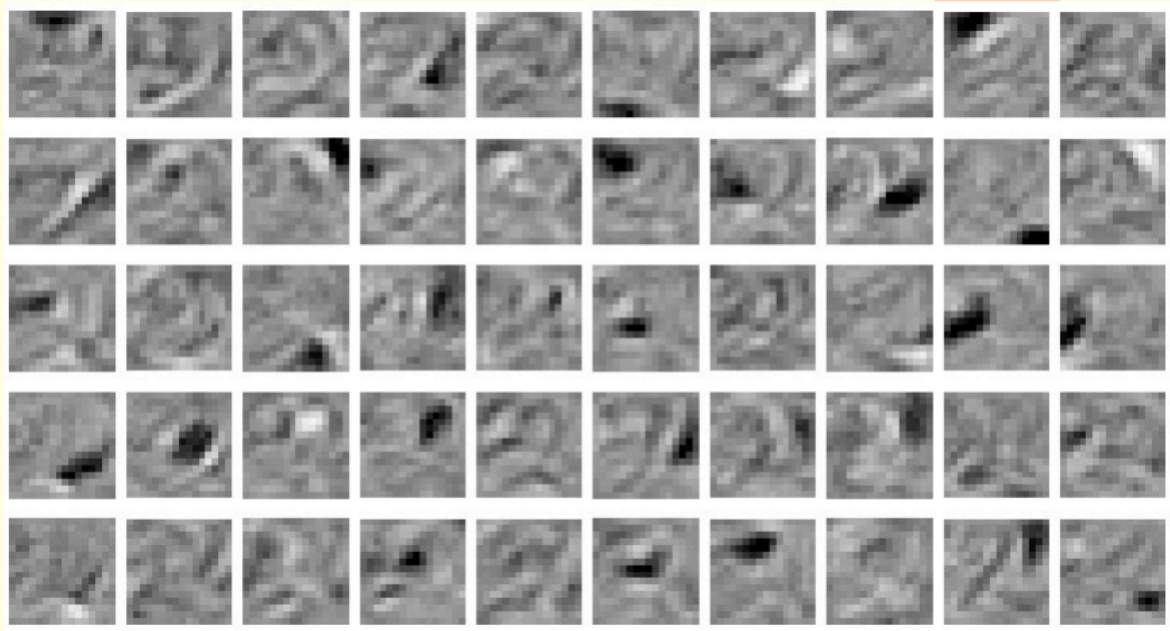
$$\frac{\partial \log p(v)}{\partial w_{ij}} = <v_i h_j>^0 - <v_i h_j>^\infty$$

Slide from G. Hinton's tutorial at NIPS 2007



Slide from G. Hinton's tutorial at NIPS 2007

The final 50 x 256 weights

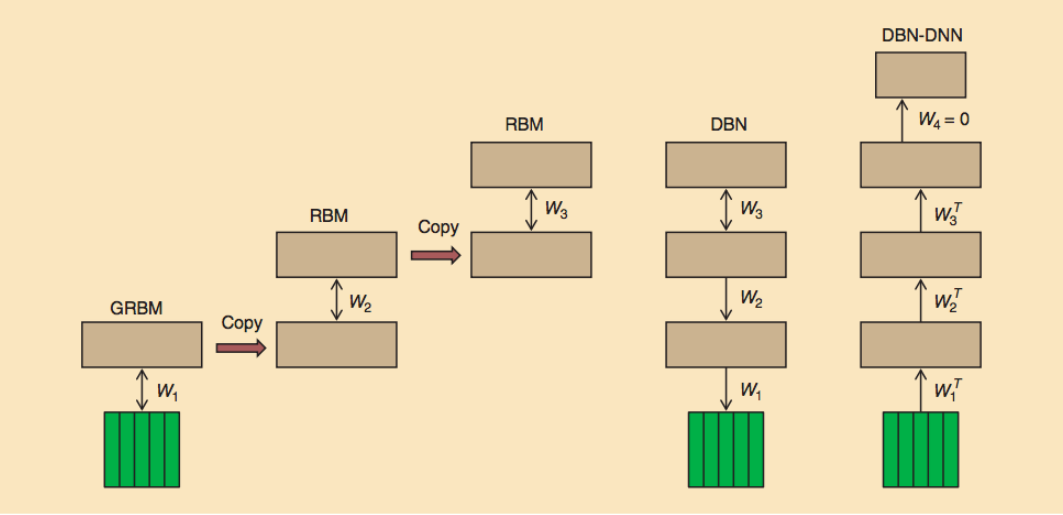


Each neuron grabs a different feature.

Slide from G. Hinton's tutorial at NIPS 2007

Deep generative models

- This training algorithm lead to the renaissance of NNs in 2016:
  - Key idea: to train deep models using nested layers of RBMs
  - Each layer is pre-trained independently
  - A final (fine tuning) stage based on backprop is commonly used



Deep generative models

- Outstanding results at that time!
- Nice movie at:
  - http://www.cs.toronto.edu/~hinton/digits.html

Version of MNIST Task	Learning Algorithm	Test Error %
Permutation invariant	Our generative model: $784 \rightarrow 500 \rightarrow 500 \leftrightarrow 2000 \leftrightarrow 10$	1.25
Permutation invariant	Support vector machine: degree 9 polynomial kernel	1.4
Permutation invariant	Backprop: $784 \rightarrow 500 \rightarrow 300 \rightarrow 10$ cross-entropy and weight-decay	1.51
Permutation invariant	Backprop: $784 \rightarrow 800 \rightarrow 10$ cross-entropy and early stopping	1.53
Permutation invariant	Backprop: $784 \rightarrow 500 \rightarrow 150 \rightarrow 10$ squared error and on-line updates	2.95
Permutation invariant	Nearest neighbor: all 60,000 examples and L3 norm	2.8
Permutation invariant	Nearest neighbor: all 60,000 examples and L2 norm	3.1
Permutation invariant	Nearest neighbor: 20,000 examples and L3 norm	4.0
Permutation invariant	Nearest neighbor: 20,000 examples and L2 norm	4.4
Unpermuted images; extra data from elastic deformations	Backprop: cross-entropy and early-stopping convolutional neural net	0.4
Unpermuted de-skewed images; extra data from 2 pixel translations	Virtual SVM: degree 9 polynomial kernel	0.56
Unpermuted images	Shape-context features: hand-coded matching	0.63
Unpermuted images; extra data from affine transformations	Backprop in LeNet5: convolutional neural net	0.8
Unpermuted images	Backprop in LeNet5: convolutional neural net	0.95

Hinton, G. E., Osindero, S. and Teh, Y. (2006) A fast learning algorithm for deep belief nets. Neural Computation, 18, pp 1527-1554.

Deep generative models

- Model the joint probability of latent and observable variables
- Efficient learning of parameters
- Are not restricted to Bernoulli units
- Not too much interest from the ML community nowadays (with respect, e.g., to CNNs)

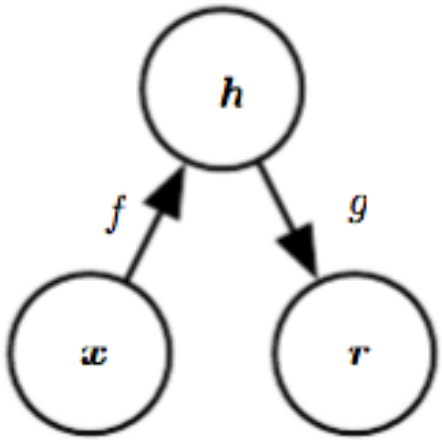
Hinton, G. E., Osindero, S. and Teh, Y. (2006) A fast learning algorithm for deep belief nets. Neural Computation, 18, pp 1527-1554.

Deep learning variants

- Main DL models:
  - Deep neural networks (DNNs, MLPs)
  - Convolutional neural networks (CNNs)
  - LSTM
  - Restricted Boltzman machines
  - Deep belief networks
  - Autoencoders
- New paradigms
  - Residual DNNs
  - Gated recurrent NNs
  - Generative adversarial networks

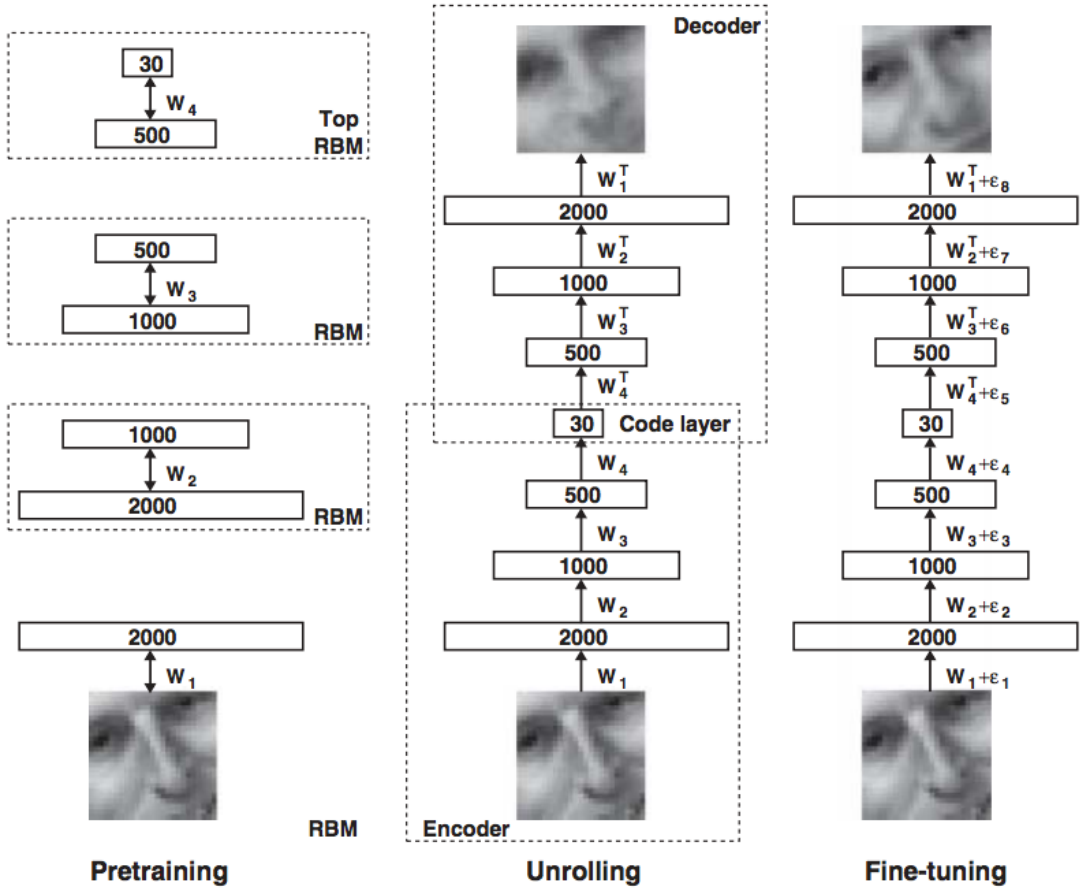
Autoencoders

- Neural networks that are trained to attempt to copy its input to its output
- A code layer is used as pivot, where there are codifying and de codifying layers of parameters
  - Encoder: $\mathbf{h} = f(\mathbf{x})$
  - Decoder: $\mathbf{r} = g(\mathbf{h})$
- Usually the dimension of h is lower than that of x (undercomplete AEs)
- They can be linear/ non linear, sparse, non sparse, and can be used for representation learning, dimensionality reduction and denoising



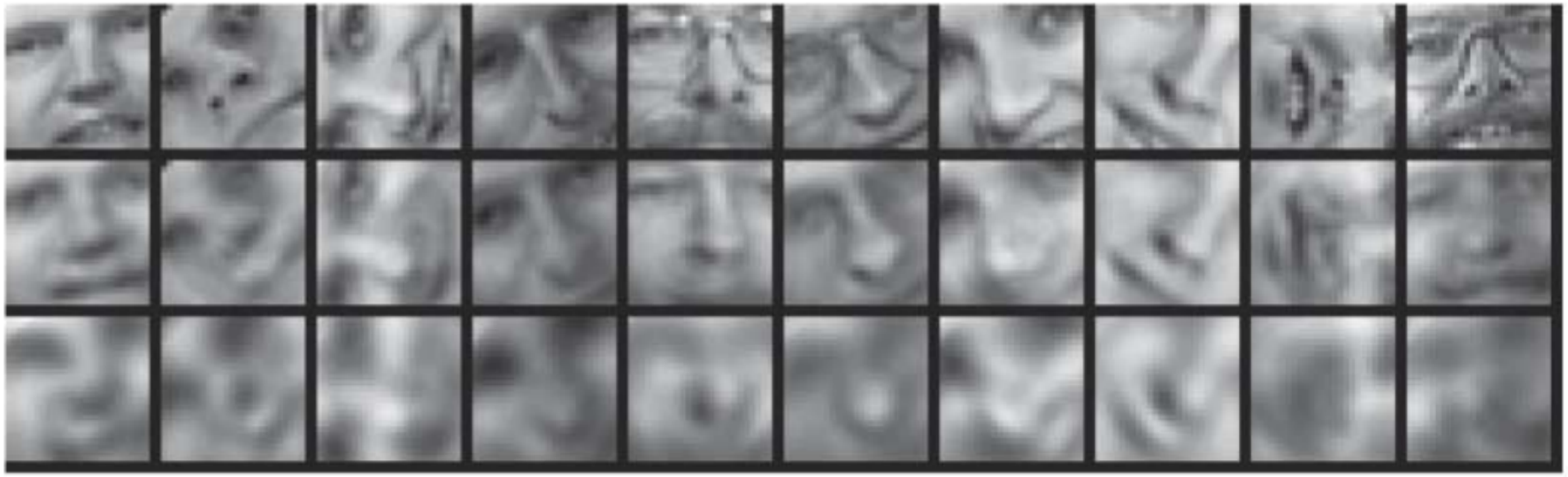
Autoencoders

- Deep autoencoders! (Hinton's science paper)
  - Pretraining of layers using RBMs
  - Unfolding
  - Fine tunning with backprop



Autoencoders

- Deep autoencoders! (Hinton's science paper)



Deep learning variants

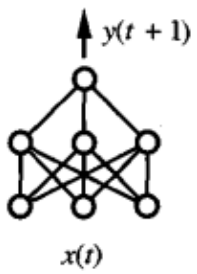
- Main DL models:
  - Deep neural networks (DNNs, MLPs)
  - Convolutional neural networks (CNNs)
  - LSTM
  - Restricted Boltzman machines
  - Deep belief networks
  - Autoencoders
- New paradigms
  - Residual DNNs
  - Gated recurrent NNs
  - Generative adversarial networks

Modeling sequential data

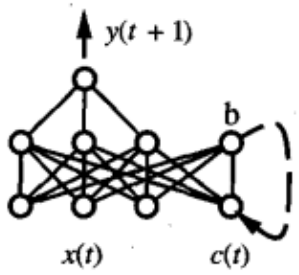
- **Recurrent neural networks.** NNs that receive as input information from their outputs

- Hidden units can be specified as

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



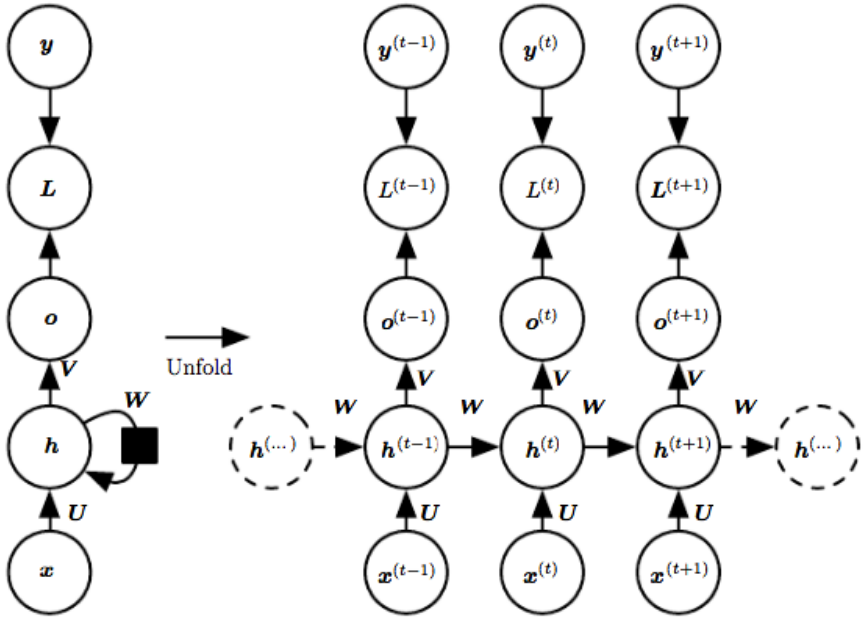
(a) Feedforward network



(b) Recurrent network

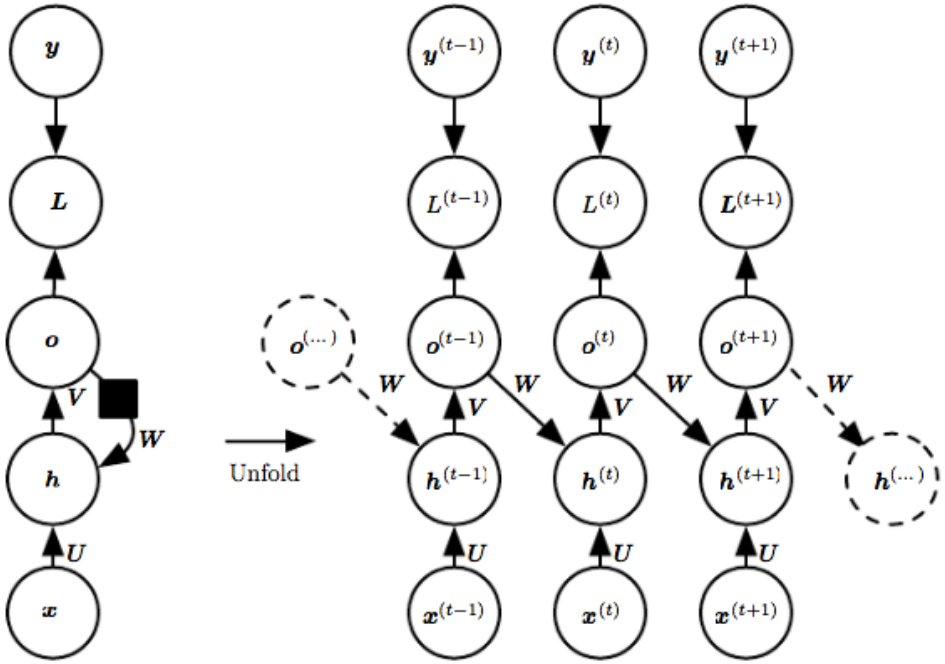
Modeling sequential data

- Types of RRNs - 1



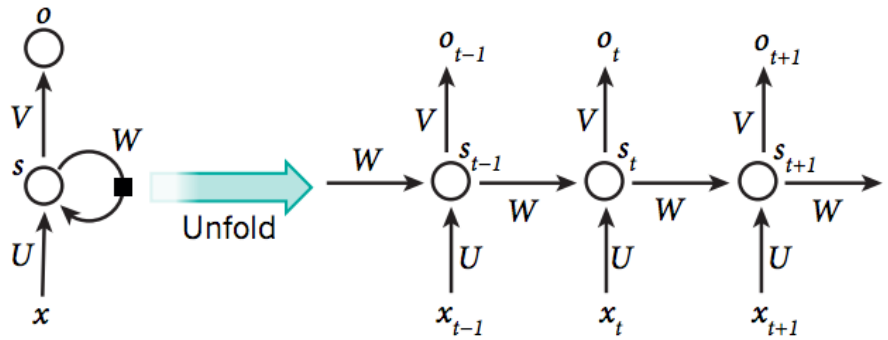
Modeling sequential data

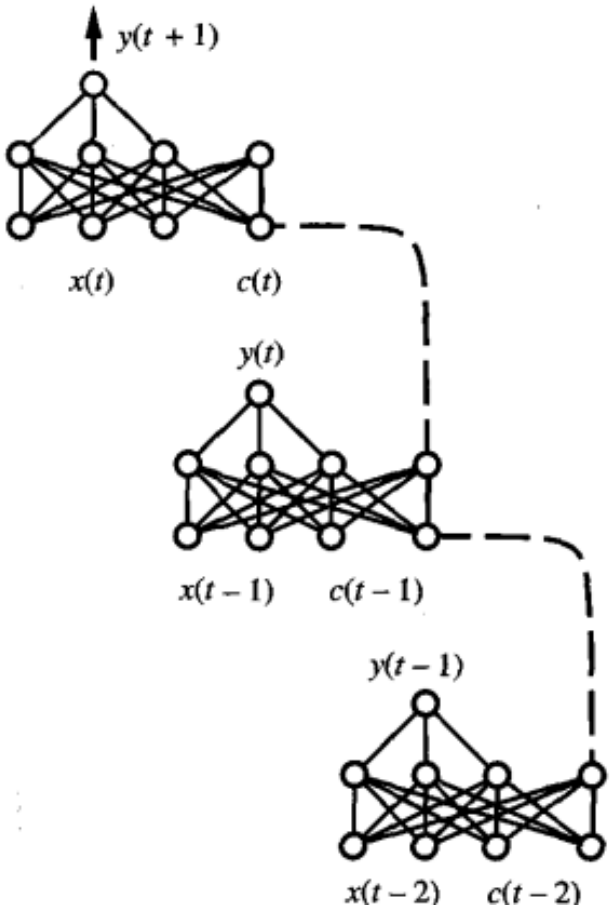
- Types of RRNs - 2



Modeling sequential data

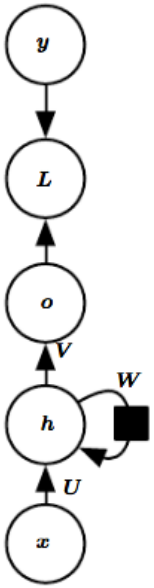
- Training RNNs
  - BPTT: Unfolding + backprop

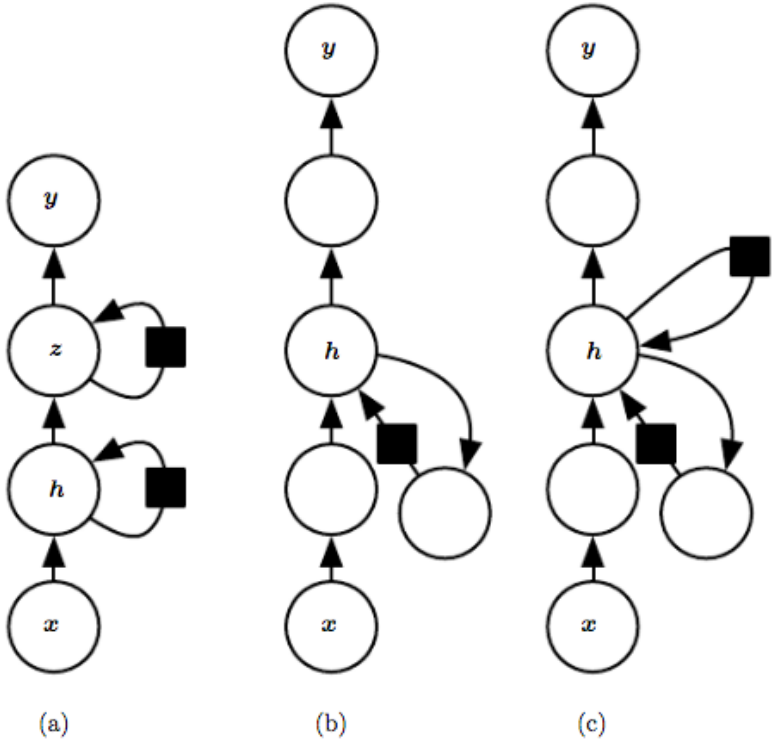




Modeling sequential data

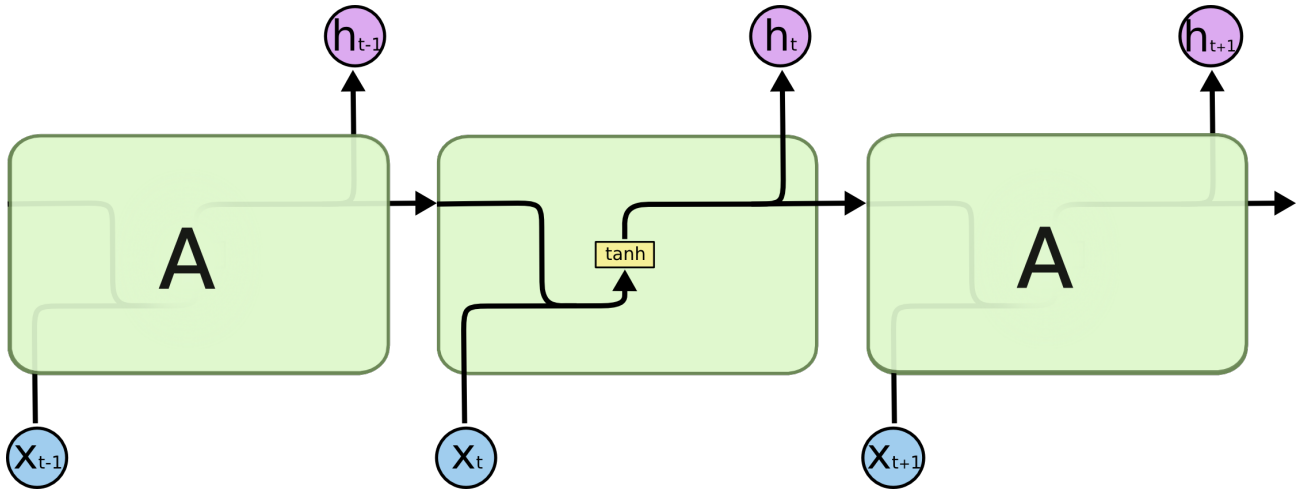
- Going deep with RNNs





Modeling sequential data

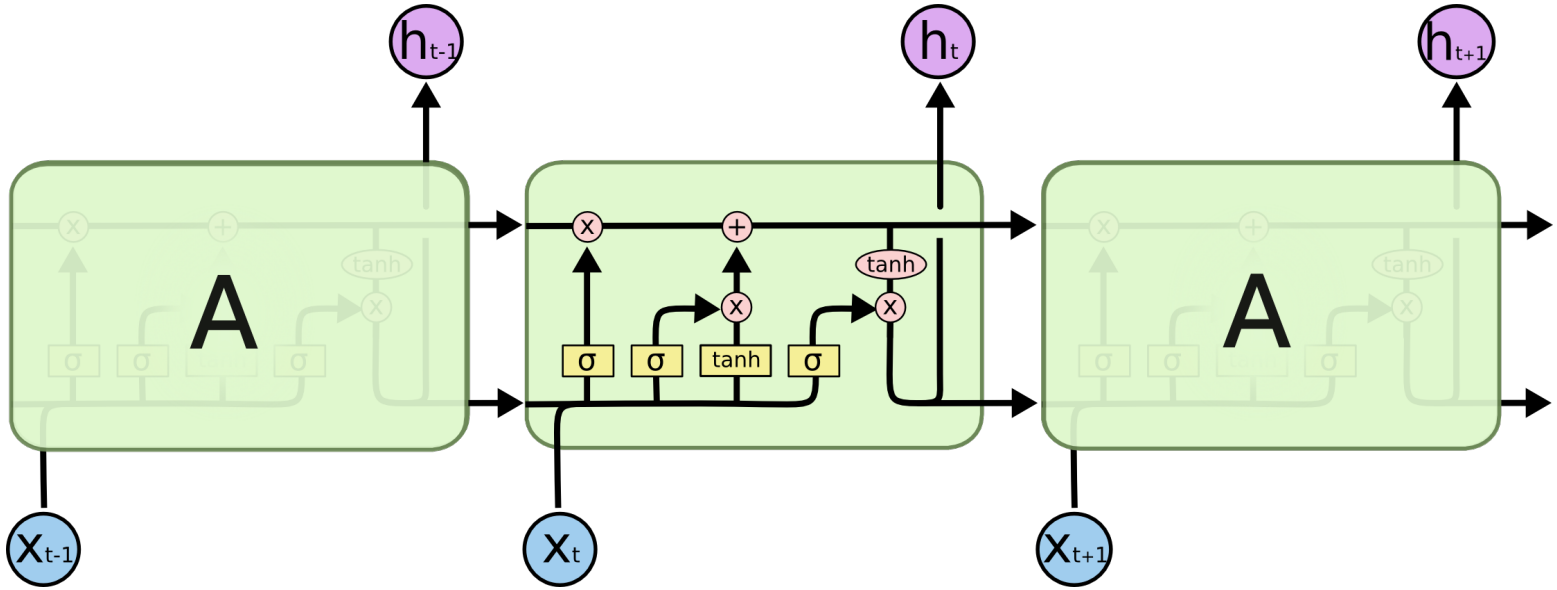
- LSTM: Long short-term memory



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Modeling sequential data

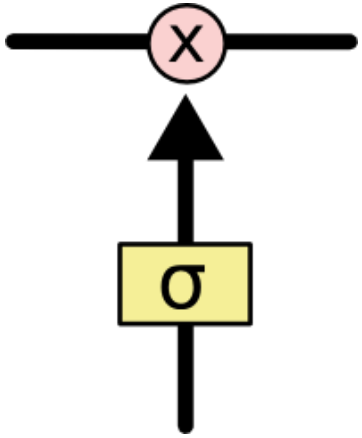
- LSTM: Long short-term memory



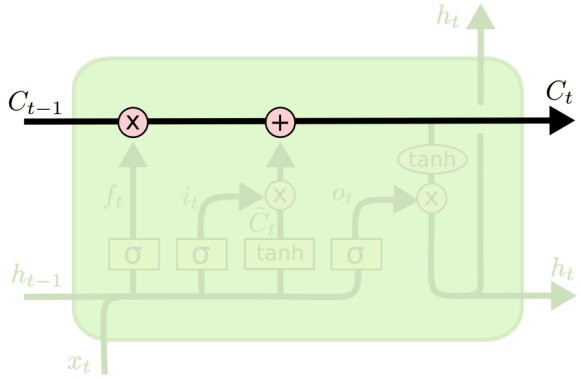
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Modeling sequential data

- LSTM: Long short-term memory



Gated units

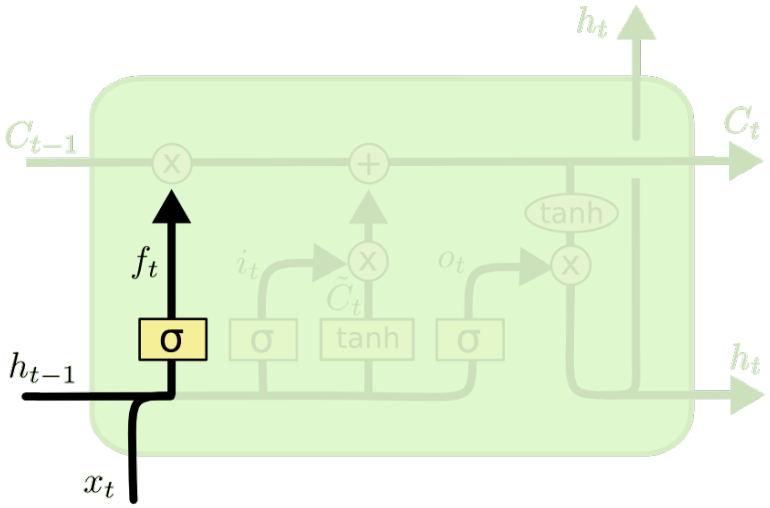


Celllstate

Modeling sequential data

- LSTM: Long short-term memory

How much to take from the previous state?

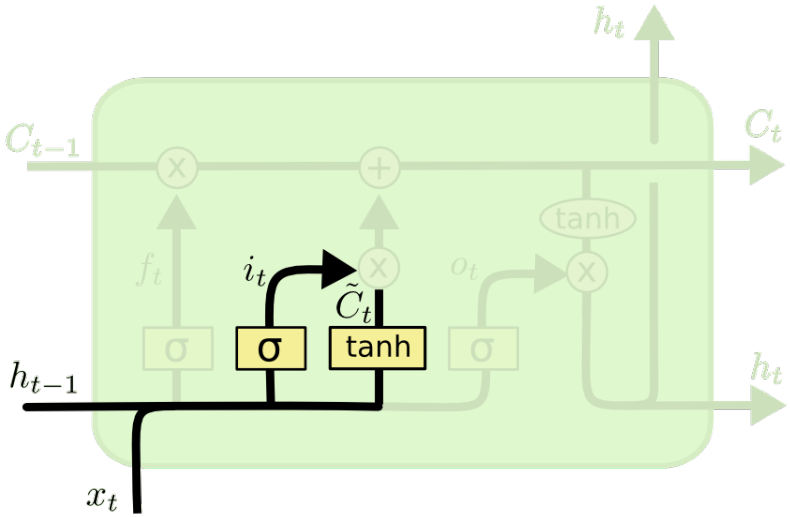


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Modeling sequential data

- LSTM: Long short-term memory

What can be added to the new state and how much



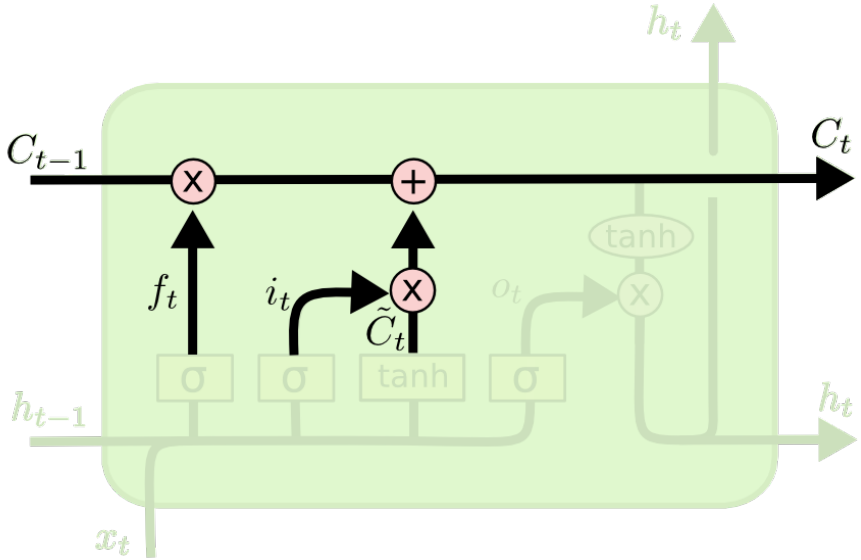
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Modeling sequential data

- LSTM: Long short-term memory

Update the state

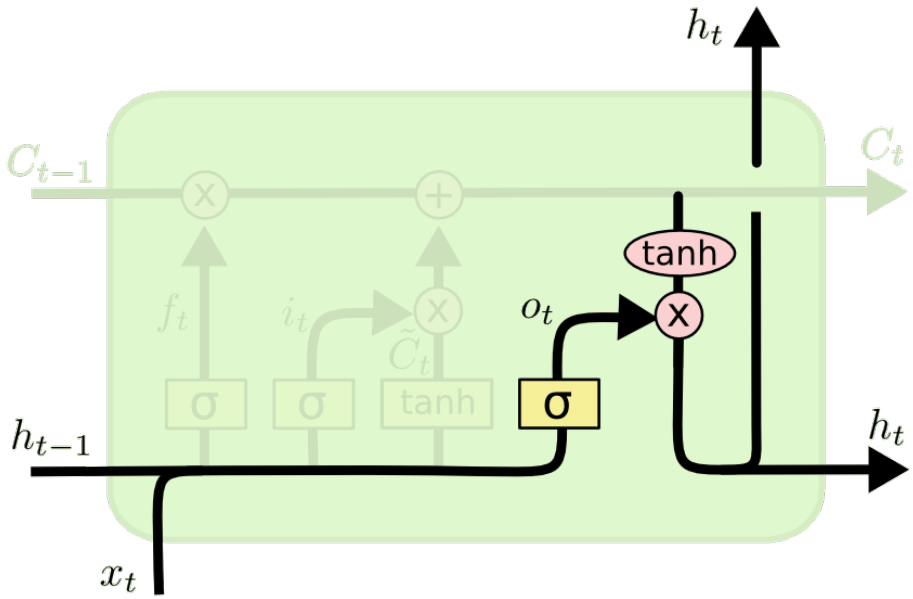


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Modeling sequential data

- LSTM: Long short-term memory

What to output?

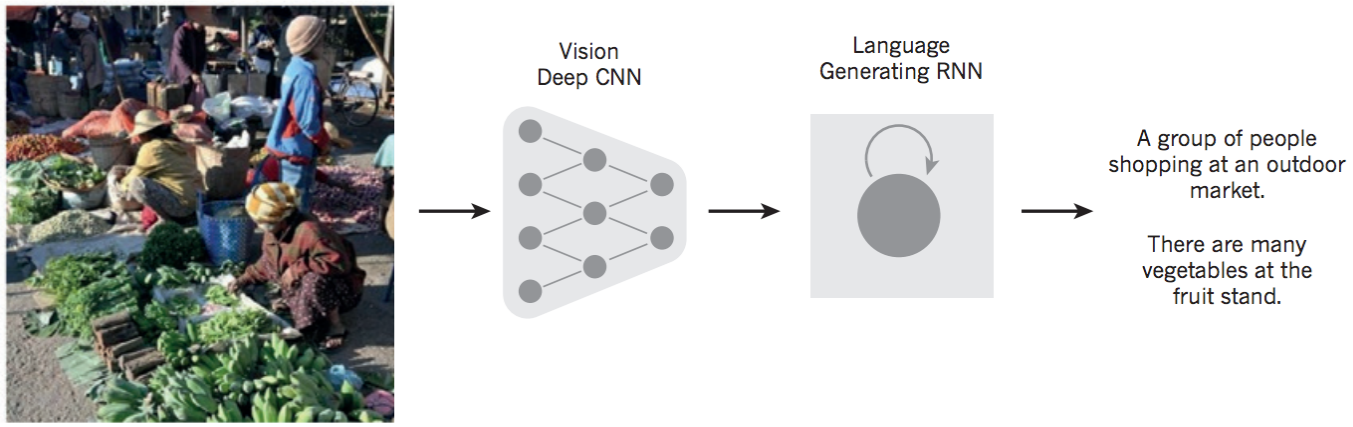


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Modeling sequential data

- LSTM: Impressive results in a number of tasks (speech processing, machine translation) and widely used nowadays:
  - Image captioning
  - Natural language processing
  - Multimodal information processing



Deep learning variants

- Main DL models:
  - Deep neural networks (DNNs, MLPs)
  - Convolutional neural networks (CNNs)
  - LSTM
  - Restricted Boltzman machines
  - Deep belief networks
  - Autoencoders

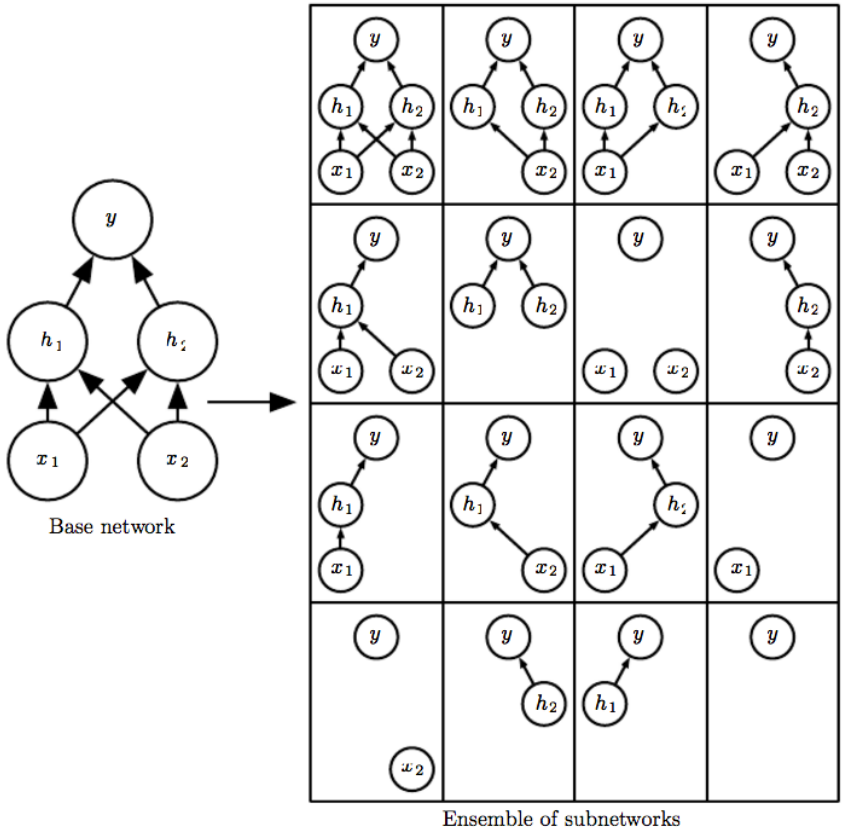
- New paradigms
  - Residual DNNs
  - Gated recurrent NNs
  - Generative adversarial networks

DL extensions, enhancements

- Dropout
- Adversarial training
- Multi task learning
- Multi stream architectures
- Inception
- …

Dropout

- Idea: to drop out (switch off) non-output units with certain probabilities;using minibatches to update the parameters of the whole DNN under the different *masks*



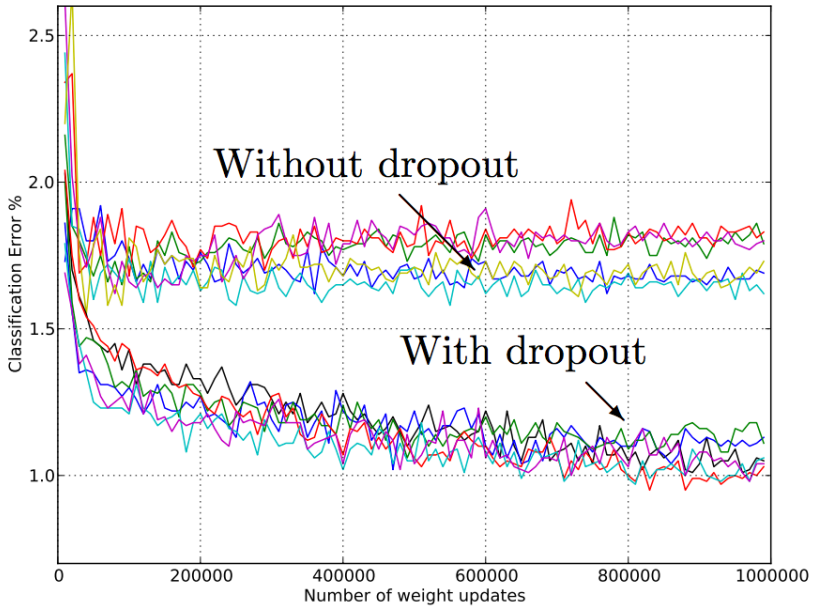
- Resembles bagging:

$$\frac{1}{k}\sum_{i=1}^{k} p^{(i)}(y \mid \boldsymbol{x}). \qquad \sum_{\boldsymbol{\mu}} p(\boldsymbol{\mu}) p(y \mid \boldsymbol{x}, \boldsymbol{\mu})$$

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. **Dropout: A Simple Way to Prevent Neural Networks from Overfitting.** 15(Jun):1929–1958, 2014.

Dropout

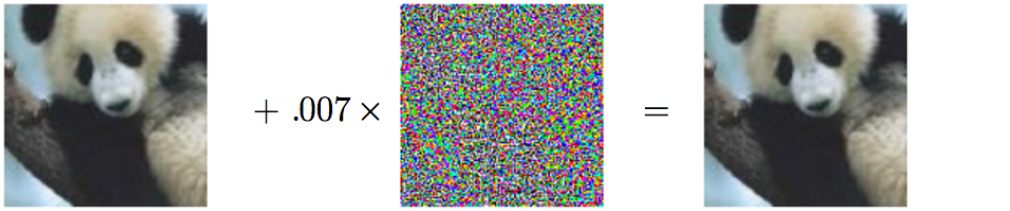
- Idea: to drop out (switch off) non-output units with certain probabilities;using minibatches to update the parameters of the whole DNN under the different *masks*



Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. **Dropout: A Simple Way to Prevent Neural Networks from Overfitting.** 15(Jun):1929−1958, 2014.

Adversarial training

- Idea: introduce *adversarial* examples during training
  - Adversarial example: an instance that after slight modifications cause the DNN to make a mistake

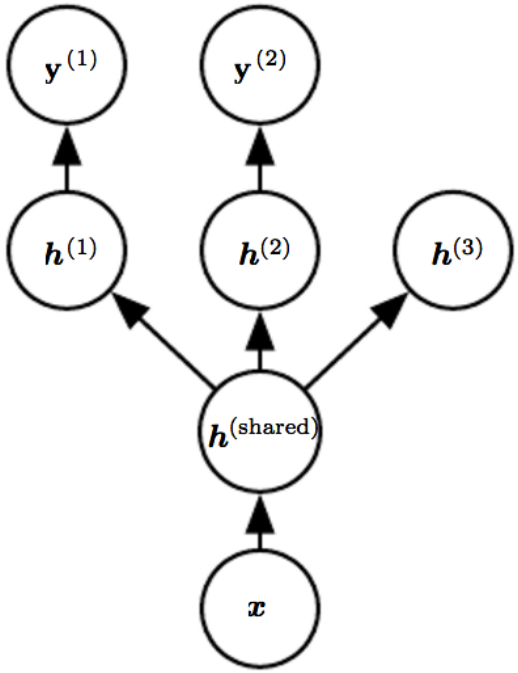


Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy. Explaining and Harnessing Adversarial Examples ArXic 1412.6572, 2014

Multitask learning

- Idea: to learn models that share generic layers and at the same time learn task specific layers

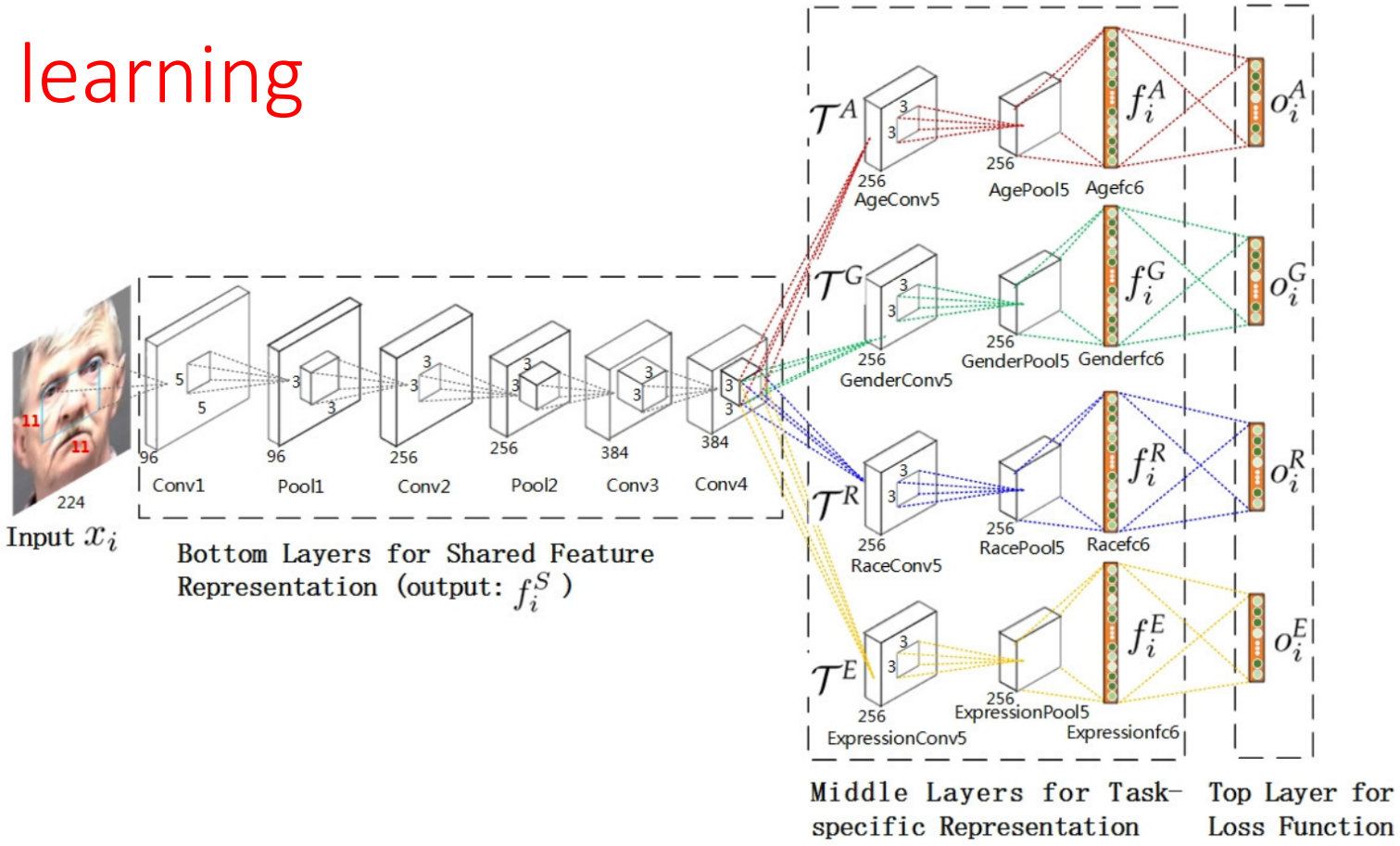
- Belief: *Among the factors that explain the variations observed in the data associated with the different tasks, some are shared across two or more tasks*



R. Caruana. **Multitask Connectionist Learning**. Proc. Connectionist models summer school, 372--379, 1993

Multitask learning

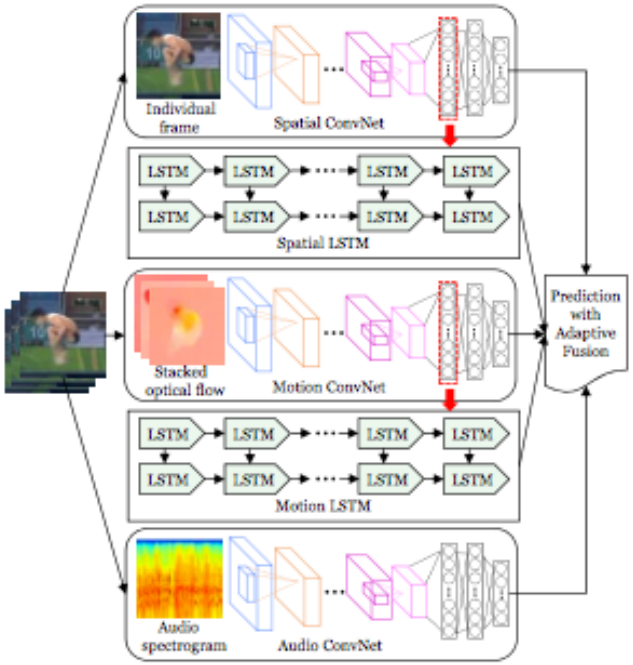
- E.g., simultaneously predict: gender, age, race, facial expression



J. Wan, S. Zhou, Z. Tan, H.J. Escalante, Y. Liang, Z. Lei, G. Guo, S.Z. Li. **Deep Nonholonomic Label Information Learning for Globally Fine-grained Face Attribute Analysis**, Submitted to TPAMI, 2017

Multi stream models

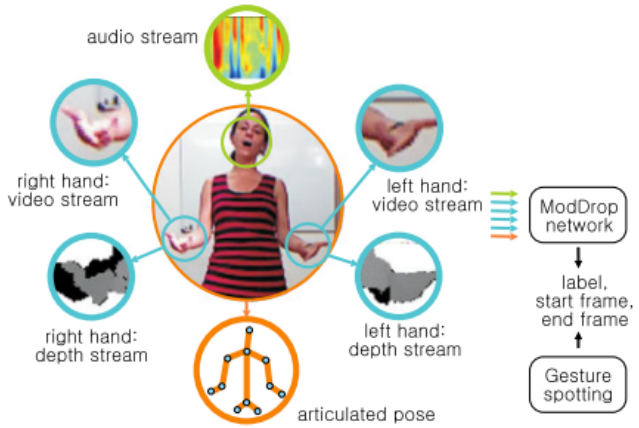
- Idea: To have different internal paths within the model, that eventually converge to the same layer (same task)

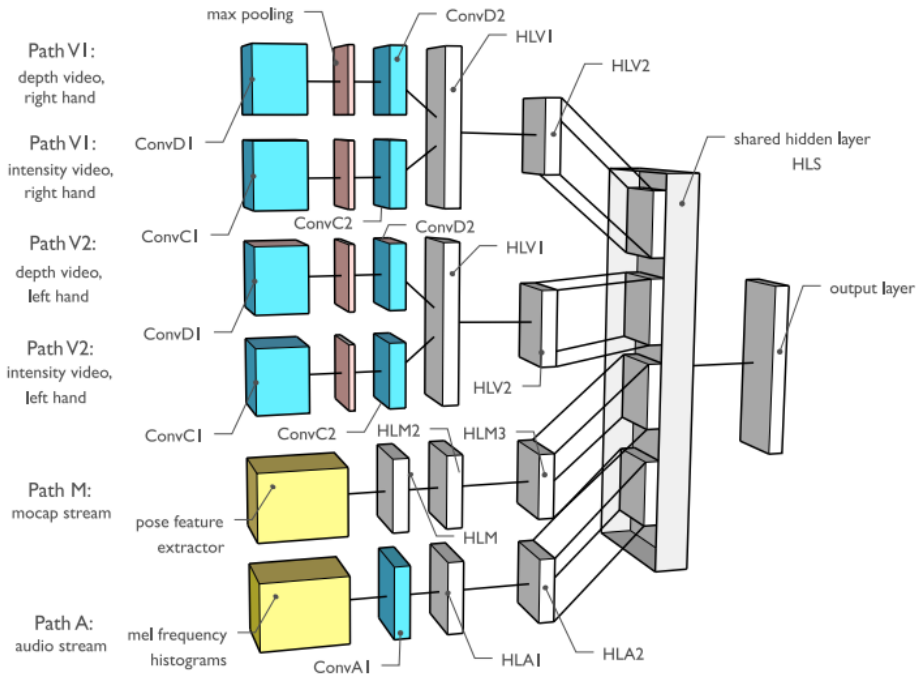


Zuxuan Wu, Yu-Gang Jiang, Xi Wang, Hao Ye, Xiangyang Xue, Jun Wang. **Fusing Multi-Stream Deep Networks for Video Classification.** arXiv:1509.06086, 2015

Multi stream models

- E.g., multimodal gesture recognition



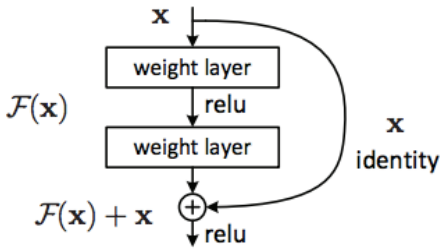


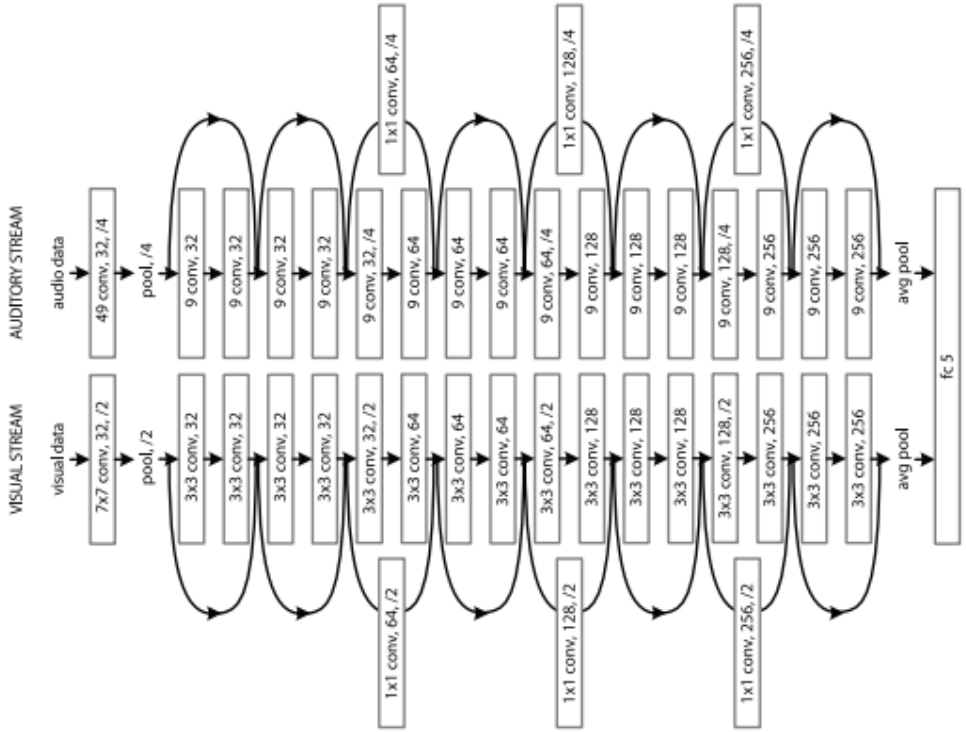
Natalia Neverova, Christian Wolf, Graham Taylor, Florian Nebout. ModDrop: adaptive multi-modal gesture recognition. TPAMI, Vol, 38(8): 1692--1706, 2016

Inception

Residual DNNs

- Idea: To introduce layers that can be used or not, that copy the output of other layers





Final remarks

- Benefits
  - Extremely good at learning representations and models from large data sets
  - Efficient training, massive parallelization capabilities
  - Outstanding generalization capabilities,

- Limitations
  - Require of extremely large data sets (big data)
  - Demanding computational resources
  - Black box models, no interpretability, explainability

Final remarks

- A very introductory tutorial on DL
- Deep models dominate the arenas of CV, PR, NLP, SP, and in shortly will be ruling over other domains
- It is difficult to track the progress in DL

References

- P. Loncomilla Deep learning: CNNS https://ccc.inaoep.mx/~pgomez/deep/presentations/2016Loncomilla.pdf