Neural Networks and Deep Learning

Tian-Li Yu

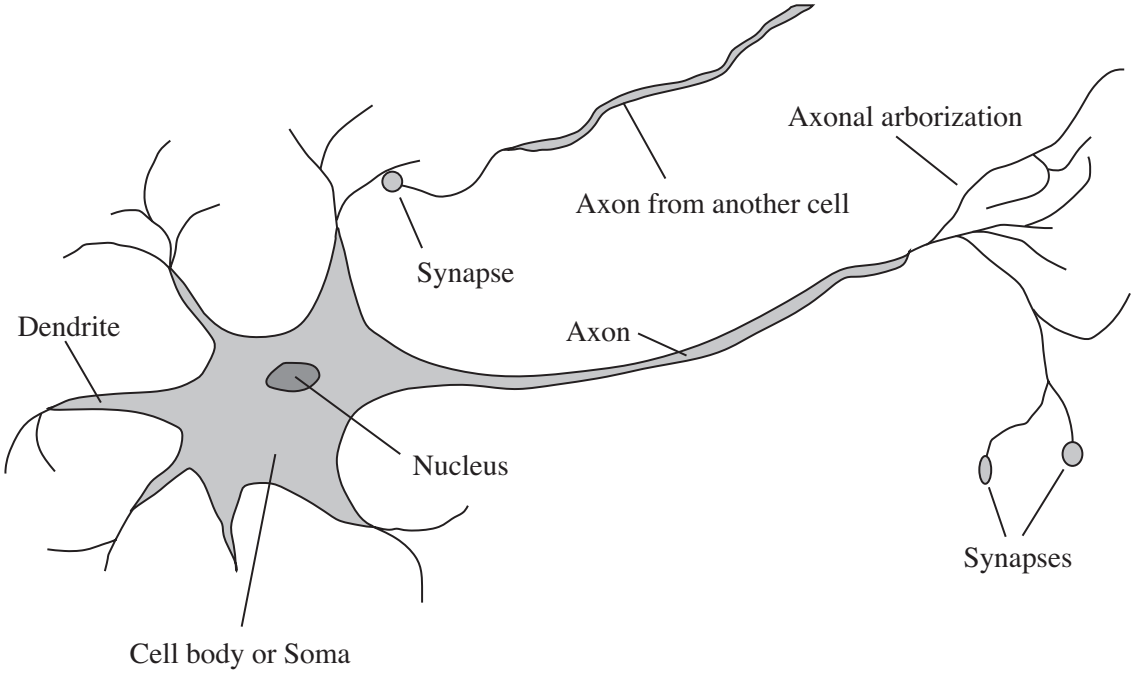
National Taiwan University
Department of Electrical Engineering
tianliyu@ntu.edu.tw

Readings: AIMA 18.7

Outline

Brains

- 10^{11} neurons of > 20 types, 10^{14} synapses, $1 \sim 10$ms cycle time.

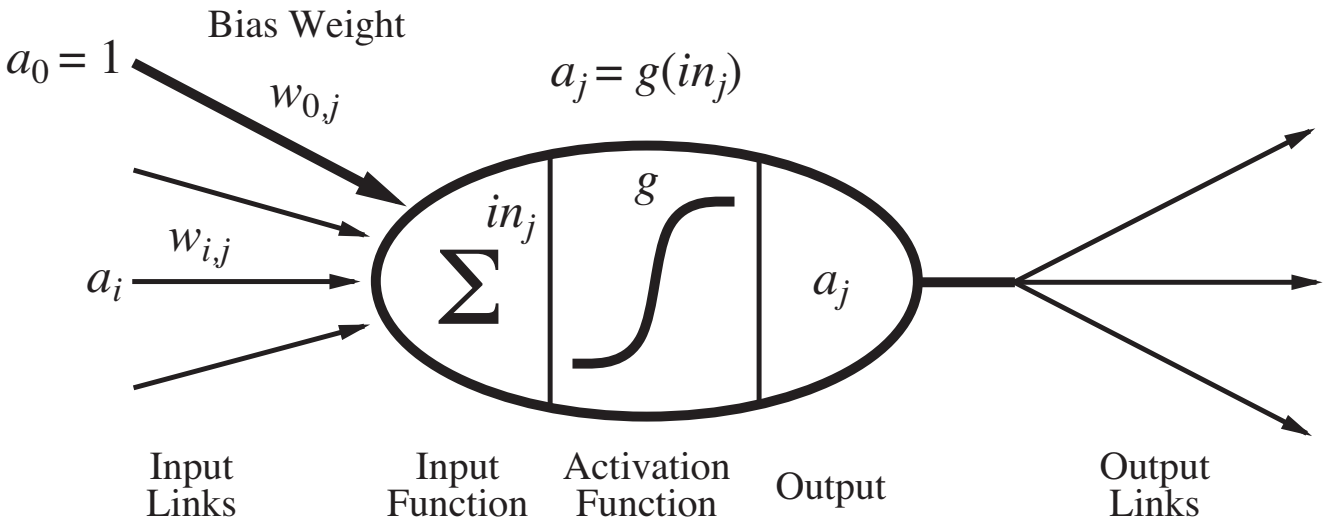


- Hypothesis: mental activity consists primarily of electrochemical activity in networks of neurons.
- Artificial neural networks, connectionism, neural computation.

McCulloch–Pitts Unit

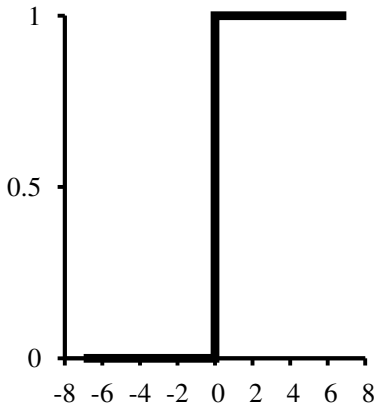
- Output is a "squashed" linear function of the inputs:

$$a_j = g(in_j) = g\left(\sum_i w_{i,j} a_i\right)$$

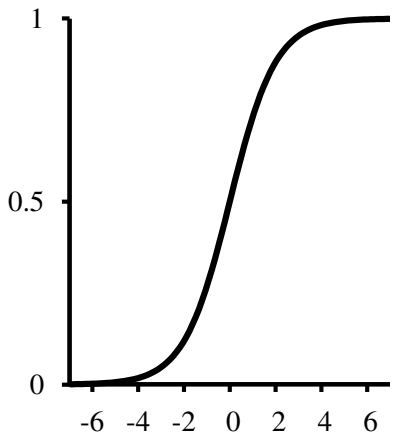


- A gross oversimplification of real neurons, but its purpose is to develop understanding of what networks of simple units can do.
- Usually, the term perceptron is used to refer a neuron or a network of neurons.

Activation Functions



(a) Step/threshold function:



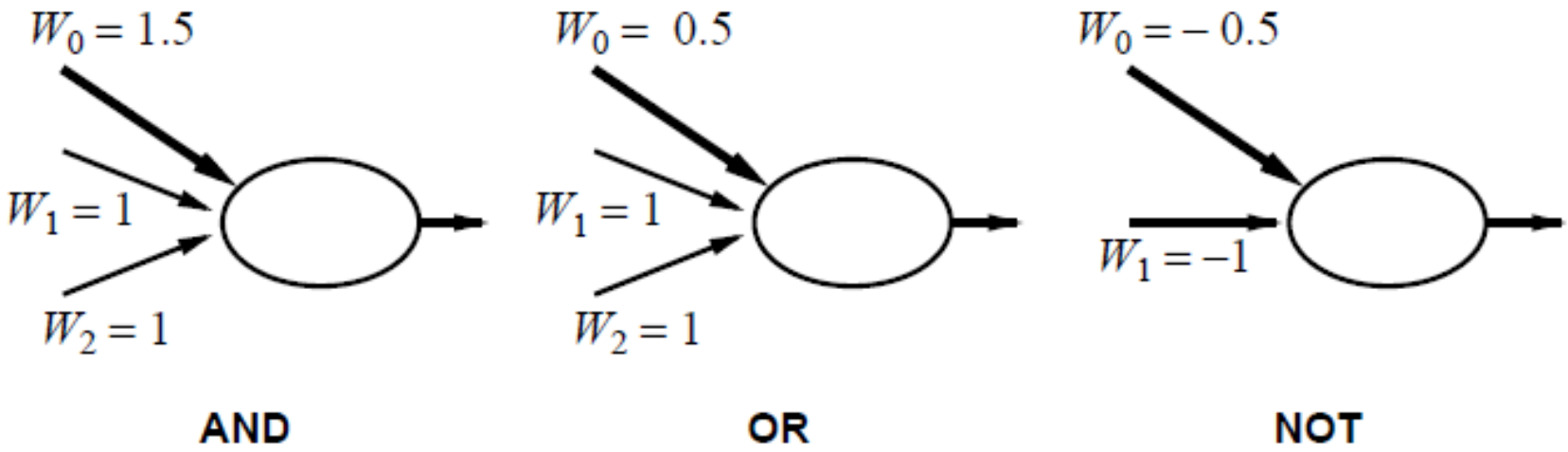
(b) Sigmoid function:

$$Threshold(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ 1, & \text{if } x > 0 \end{cases}$$

$$Logistic(x) = \frac{1}{1 + e^{-x}}$$

- Changing the bias weight $w_{0,j}$ moves the threshold location.
- $-w_{0,j}$ is also called threshold.

Implementing Logical Functions



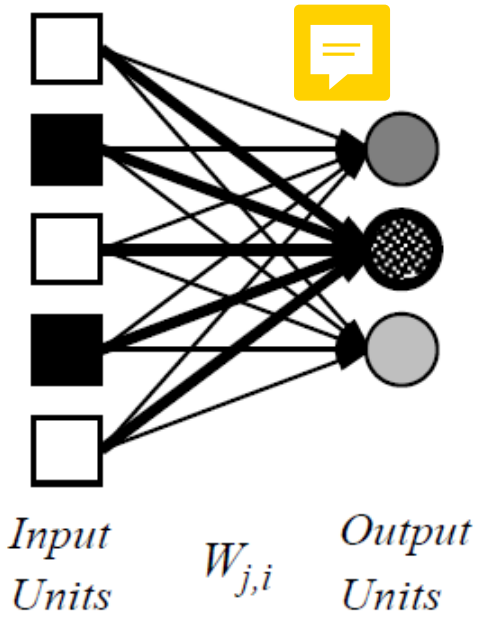
McCulloch and Pitts: every Boolean function can be implemented.

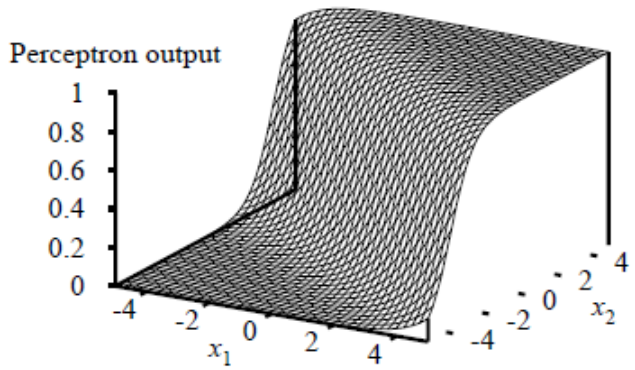
Network Structures

- Feed-forward networks
  - Directed acyclic graph (DAG).
  - Feed-forward networks implement functions, have no internal state.
  - Single-layer or multi-layer (with hidden layers and hidden units).
  - For threshold activation function, perceptron learning rule.
  - For logistic activation function, logistic regression.

- Recurrent networks
  - Hopfield networks have symmetric weights ($w_{i,j} = w_{j,i}$)
    $g(x) = \text{sign}(x) = \pm 1$; Associative memory.
  - Boltzmann machines use stochastic activation functions.
  - Recurrent neural nets have directed cycles with delays
    $\Rightarrow$ have internal state (like flip-flops), can oscillate etc.

Single-Layer Feed-Forward Neural Networks



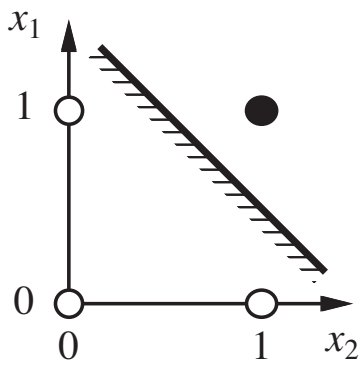


- Output units all operate separately—no shared weights.
- Adjusting weights moves the location, orientation, and steepness of cliff.

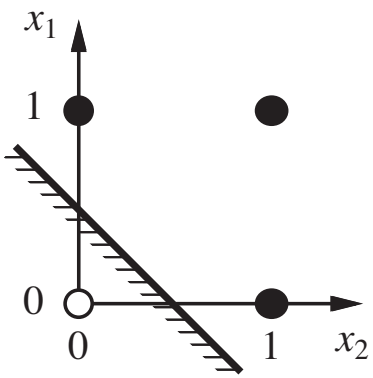
Expressiveness of Single-Layer Networks

- Consider a perceptron with g being the step function.
- Can represent AND, OR, NOT, etc., but not XOR.
- Represents a linear separator in input space:

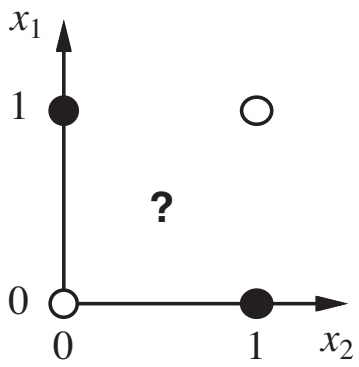
$$\sum_j w_j x_j > 0 \quad \text{or} \quad \mathbf{w} \cdot \mathbf{x} > 0$$



(a) x_1 **and** x_2



(b) x_1 **or** x_2



(c) x_1 **xor** x_2

Perceptron Learning Rule

- g is the **step** function, $o = g(\mathbf{w} \cdot \mathbf{x})$ is the output.
- Learn by adjusting weights to reduce error on training set.
- Training data: $(\mathbf{x}, y)$

$$w_i \leftarrow w_i + \alpha(y - o) \times x_i$$

- For correct output, *i.e.*, $y = g(\mathbf{w} \cdot \mathbf{x})$, weights do not change.
- If $y = 1$ but $o = 0$, then w_i increases for positive x_i and decreases for negative x_i.
- If $y = 0$ but $o = 1$, then w_i decreases for positive x_i and increases for negative x_i.

Logistic Regression

- g is the **logistic** function, $o = g(\mathbf{w} \cdot \mathbf{x})$ is the output.
- The squared error for an example with input x and true output y is

$$E \equiv \frac{1}{2} Err^2 \equiv \frac{1}{2}(y - o)^2$$

- Recall Widrow-Hoff rule (delta rule):

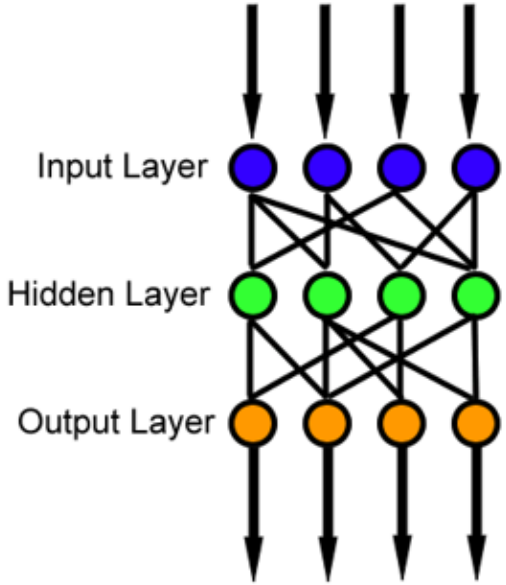
$$w_i \leftarrow w_i - \alpha \frac{\partial E}{\partial w_i}$$

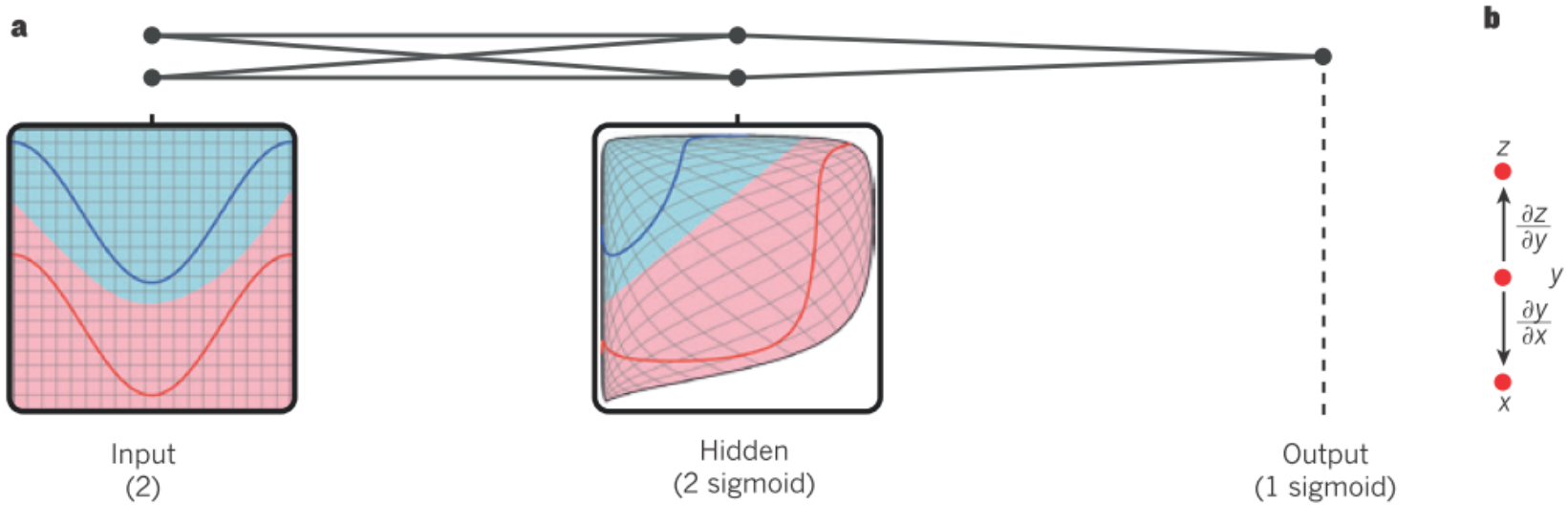
- $\frac{\partial E}{\partial w_i} = Err \cdot \frac{\partial Err}{\partial w_i} = Err \cdot \frac{\partial}{\partial w_i}\left(y - g\left(\sum_i w_i x_i\right)\right) = -Err \cdot g'(\mathbf{w} \cdot \mathbf{x}) \cdot x_i$
- For the logistic function, $g'(z) = g(z)(1 - g(z))$.

$$w_i \leftarrow w_i + \alpha \cdot (y - o) \cdot o \cdot (1 - o) \cdot x_i$$

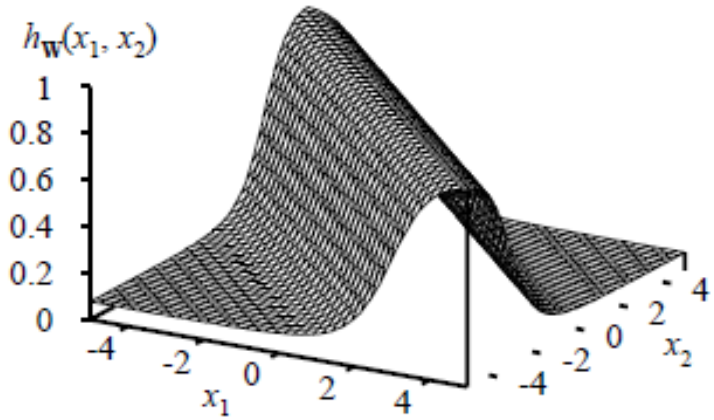
Multi-layer Perceptron (MLP)

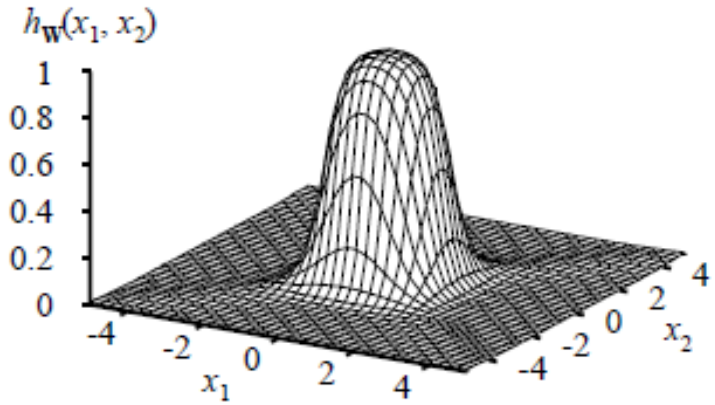
- Layers are usually fully connected.
- Numbers of hidden units typically chosen by hand.
- 3 layers were most popular.





Expressiveness of MLP





- Combine two opposite-facing threshold functions to make a ridge.
- Combine two ridges to make a bump.
- Good news: Add bumps of various sizes and locations to fit any surface.
- Bad news: May requires exponentially many hidden units.

Back-Propagation Learning

- **Output layer:** same as for single-layer:

$$w_{j,k} \leftarrow w_{j,k} + \alpha \cdot a_j \cdot \Delta_k, \text{ where } \Delta_k = Err_k \cdot g'(in_k).$$

- **Hidden layer:** back-propagate the error from the output layer:

$$\Delta_j = g'(in_j) \cdot \sum_k w_{j,k} \Delta_k$$

  Update rule for weights in hidden layer (same as output layer):

$$w_{i,j} \leftarrow w_{i,j} + \alpha \cdot a_i \cdot \Delta_j$$

- Most neuroscientists deny that back-propagation occurs in the brain.

Back-Propagation Derivation

- **Output layer:** The gradient for $E = \sum_k E_k = \sum_k \frac{1}{2}(y_k - a_k)^2$:

$$
\begin{aligned}
\frac{\partial E}{\partial w_{j,k}} = \frac{\partial E_k}{\partial w_{j,k}} &= -(y_k - a_k)\frac{\partial a_k}{\partial w_{j,k}} \\
&= -(y_k - a_k)\frac{\partial g(in_k)}{\partial w_{j,k}} \\
&= -(y_k - a_k)g'(in_k)\frac{\partial in_k}{\partial w_{j,k}} \\
&= -(y_k - a_k)g'(in_k)\frac{\partial}{\partial w_{j,k}}\left(\sum_j w_{j,k}a_j\right) \\
&= -(y_k - a_k)g'(in_k)a_j \\
&= -a_j\Delta_k
\end{aligned}
$$

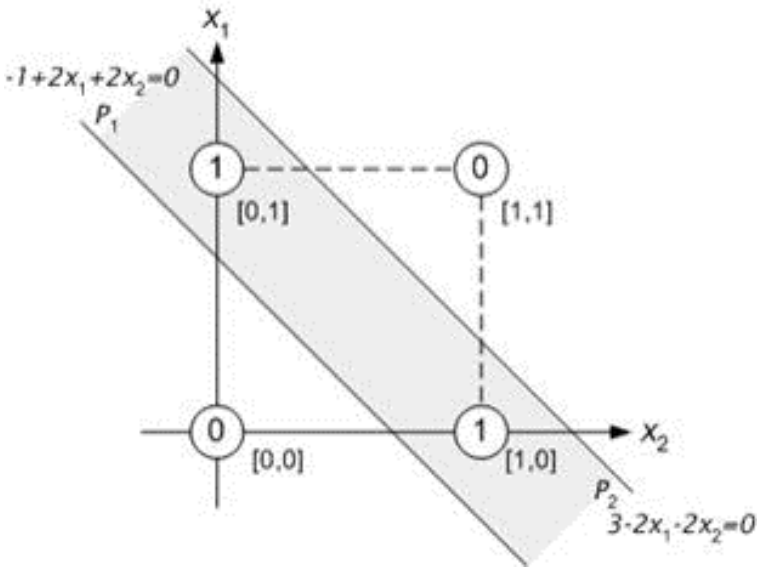
Back-Propagation Derivation

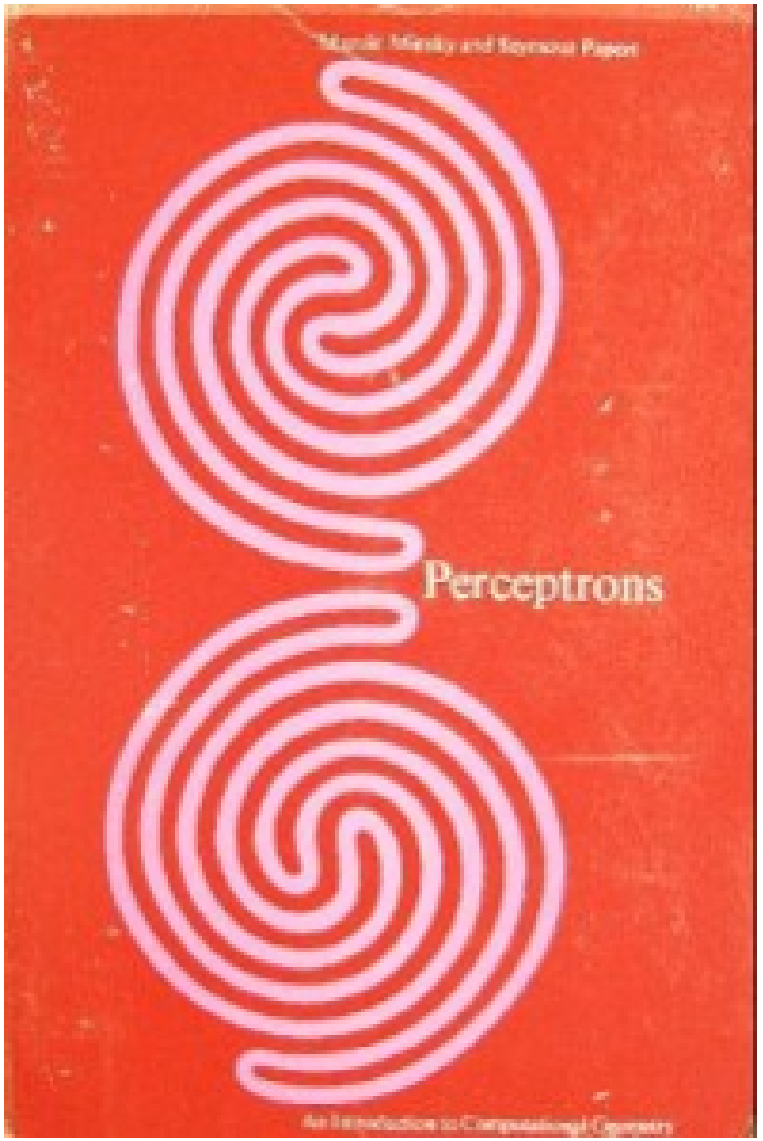
- **Hidden layer:**

$$
\begin{aligned}
\frac{\partial E_k}{\partial w_{i,j}} &= -(y_k - a_k)\frac{\partial a_k}{\partial w_{i,j}} = -(y_k - a_k)\frac{\partial g(in_k)}{\partial w_{i,j}} \\
&= -(y_k - a_k)g'(in_k)\frac{\partial in_k}{\partial w_{i,j}} = -\Delta_k \frac{\partial}{\partial w_{i,j}}\left(\sum_j w_{j,k}a_j\right) \\
&= -\Delta_k w_{j,k}\frac{\partial a_j}{\partial w_{i,j}} = -\Delta_k w_{j,k}\frac{\partial g(in_j)}{\partial w_{i,j}} = -\Delta_k w_{j,k}g'(in_j)\frac{\partial in_j}{\partial w_{i,j}} \\
&= -\Delta_k w_{j,k}g'(in_j)\frac{\partial}{\partial w_{i,j}}\left(\sum_i w_{i,j}a_i\right) \\
&= -\Delta_k w_{j,k}g'(in_j)a_i \\
\frac{\partial E}{\partial w_{j,k}} &= \sum_k \frac{\partial E_k}{\partial w_{i,j}} = -\sum_k \Delta_k w_{j,k}g'(in_j)a_i = -a_i\Delta_j
\end{aligned}
$$

Decline of ANN

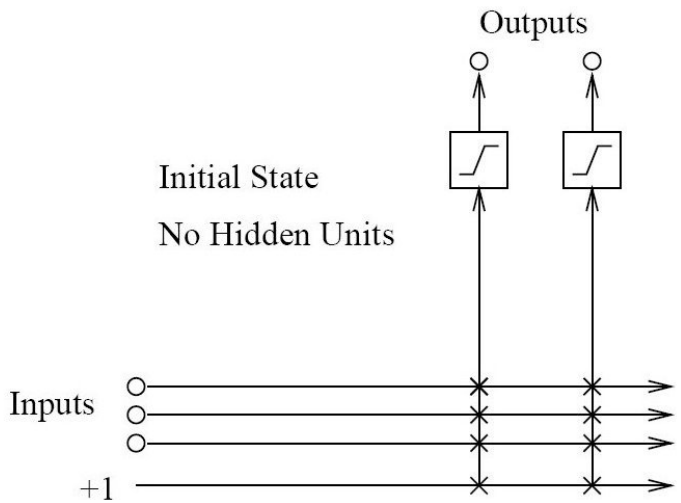
- Minsky and Papert, MIT (1969)
- XOR problem.
- Connection problem.





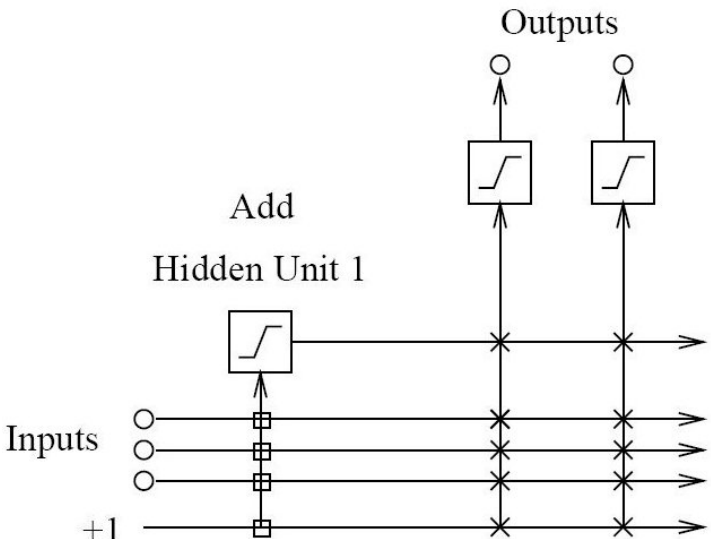
Cascade Correlation Neural Networks

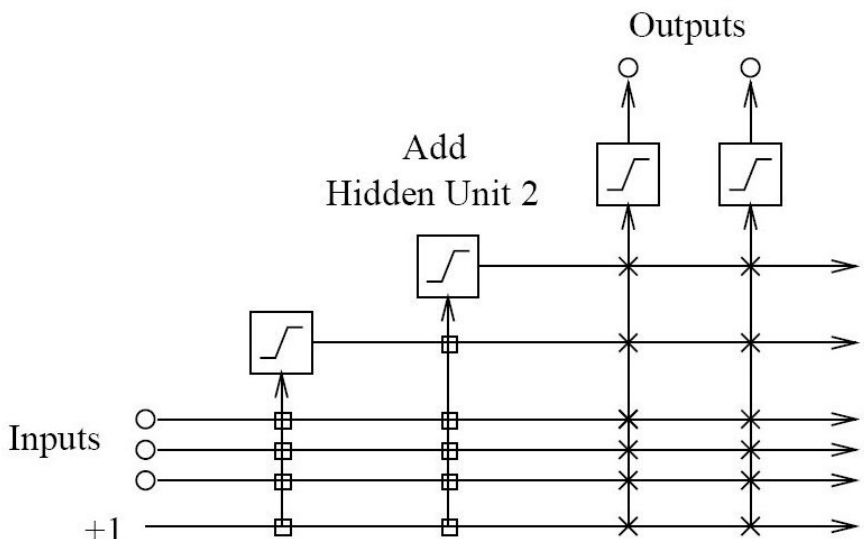
- Back propagation works fine in the most cases.
- It still suffers from (1) slow convergence and (2) the moving target problem.
  1. Many weights to train at the same time.
  2. Each unit is trying the evolve into a feature detector, but the task is complicated given all units are moving at the same time.
- Cascade correlation neural networks are self-organized and train only one unit at one time.



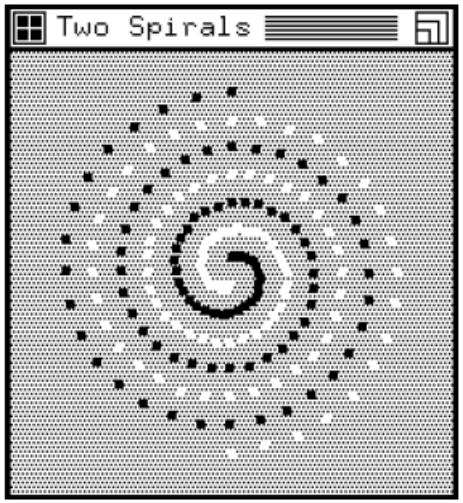
Cascade Correlation Neural Networks

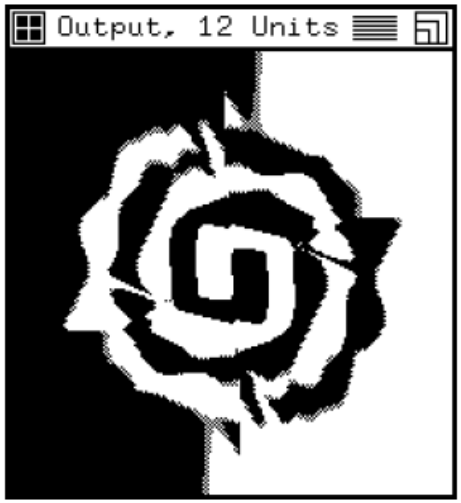
- It adds only one unit at a time (except the first stage).
- All inputs and all previous hidden units are inputs.
- When training a newly added unit, all previous trained weights are fixed.





Cascade Correlation Neural Networks





- For the two-spiral problem, 3-layer NN can not solve the problem.
- 2-5-5-5-1 network is able to solve the problem with 20,000 epochs.
- With QuickProp (a modified version of back propagation), 2-5-5-5-1 needs 8,000 epochs; 2-5-5-1 needs 60,000 epochs.
- CCNN with average 15 units solve the problem with 1700 epochs.

Optimal Brain Damage

- Find a set of parameters whose deletion will cause the least increase of MSE.
- Assume the Hessian matrix is diagonal.
- Assume cost function is nearly quadratic.

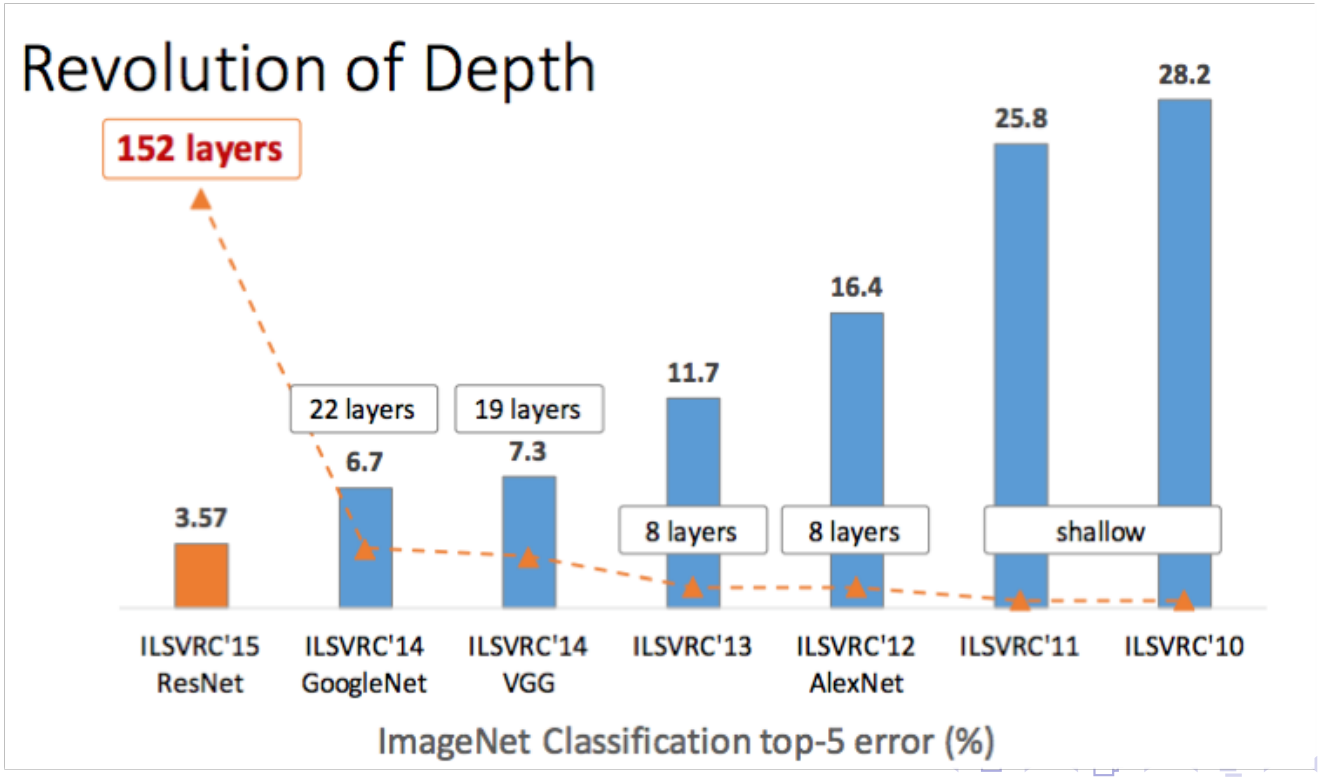
$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

$$\partial E = \frac{1}{2} \sum_i h_{ii} \partial u_i^2, \text{ where } U = < u_i > \text{ is a perturbation.}$$

1. Choose a reasonable network architecture (usually a fully connected network).
2. Train the network until local minimum is obtained.
3. Compute the second derivatives (h_{ii}) for each parameter.
4. Compute the saliencies.
5. Delete the low-saliency parameters, and then repeat 1 to 5.

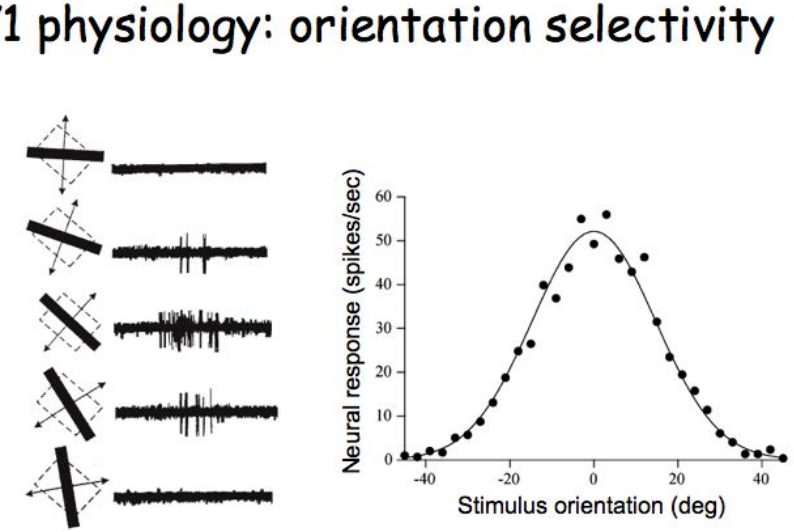
Deep vs. Shallow Structures

- It's all about representation
  - $12345 \times 54321 = ?$
  - $(3 \times 5 \times 823) + (3 \times 19 \times 953) = ?$
- Shallow structures perform well on simpler, concrete concepts.
- Deep structures seem to perform well on more abstract concepts.



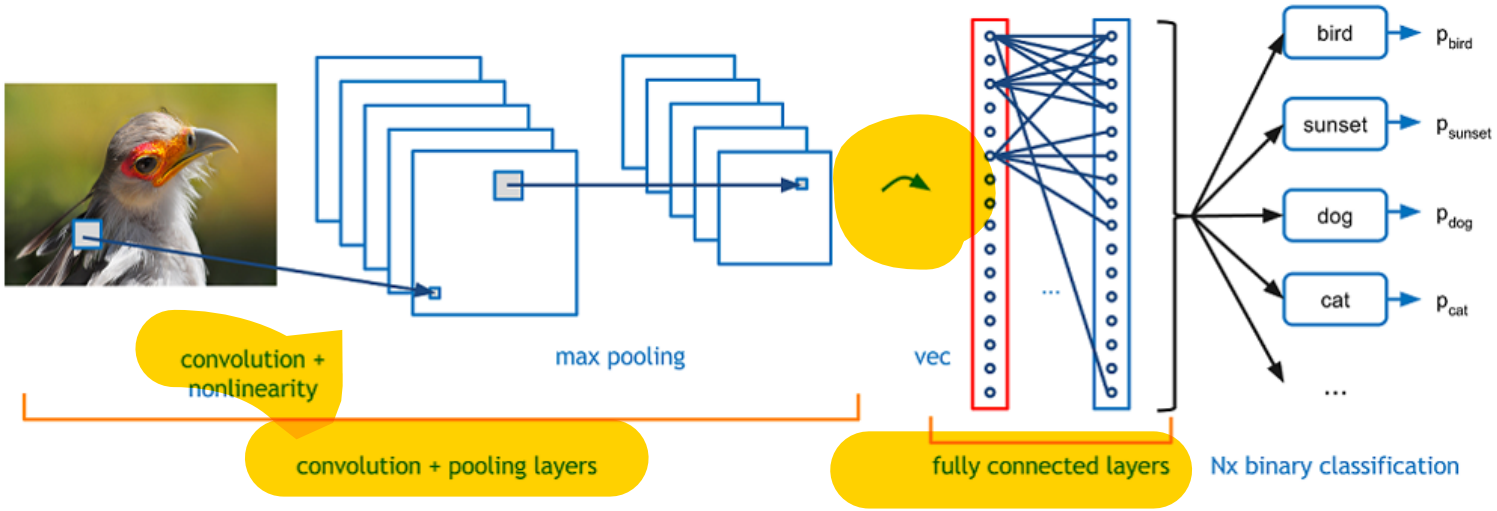
Deep Convolutional Neural Networks (DCNN)

- Inspired from Hubel and Wiesel's early work on the cats visual cortex in 1968.
- First convolutional neural network (CNN) is done by LeCun in 1989 (Later LeNet5 in 1998).
- First breakthrough in DCNN is by Hiton *et al.* in 2006.
- Gained much attention since 2012's AlexNet by Krizhevsky.
- Success due to big data, GPU, and algorithmic improvements.



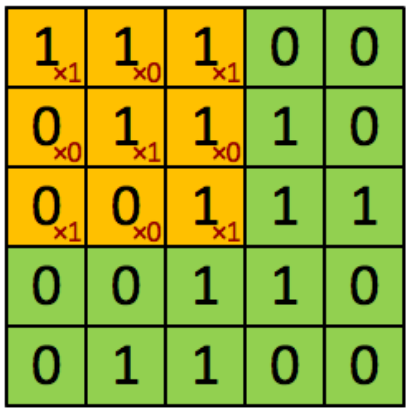
Basic Structure of CNN

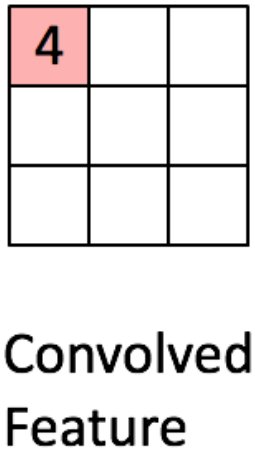
- Convolution + pooling for automatic features.
- MLP (or other techniques such as SVM) for classification.



Convolutional Layer

- Sliding filter (also called neuron or kernel)
- Example
  - Input: $32 \times 32 \times 3$
  - Filter: $5 \times 5 \times 1$
  - Output: $28 \times 28 \times 3$



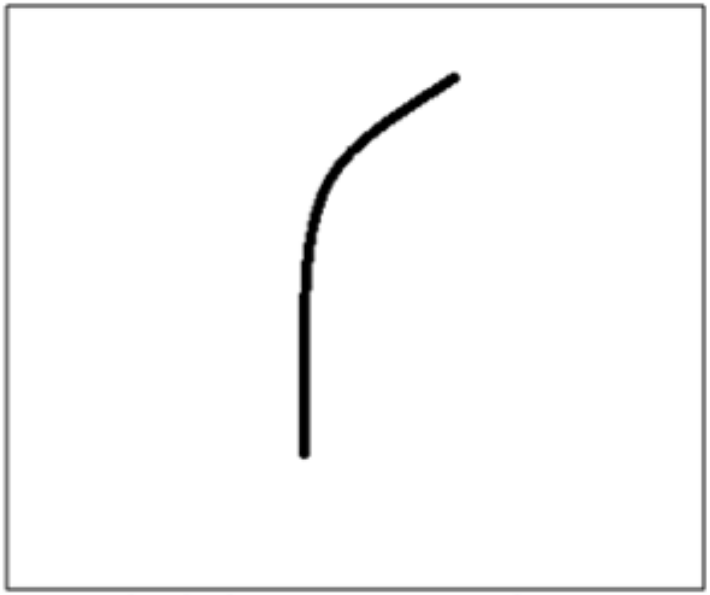


High-Level Perspective

- Filter: feature identifier

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



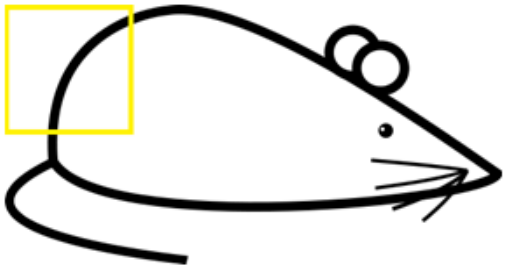
Visualization of a curve detector filter

Pictures from "A Beginner's Guide To Understanding Convolutional Neural Networks" (Adit Deshpande's Blog)

Convolution

- Sliding inner product

Original image



Visualization of the filter on the image

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

$*$

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Pictures from "A Beginner's Guide To Understanding Convolutional Neural Networks" (Adit Deshpande's Blog)

Convolution

- A filter detects a specific shape.

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

$*$

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

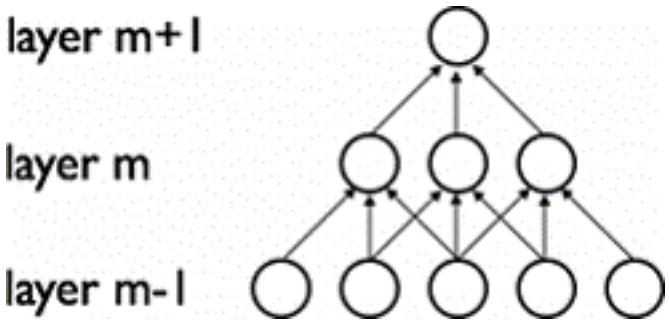
Pixel representation of filter

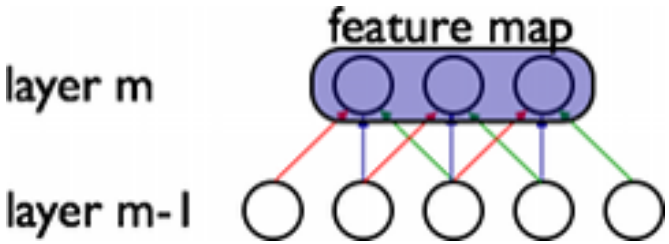
Multiplication and Summation = 0

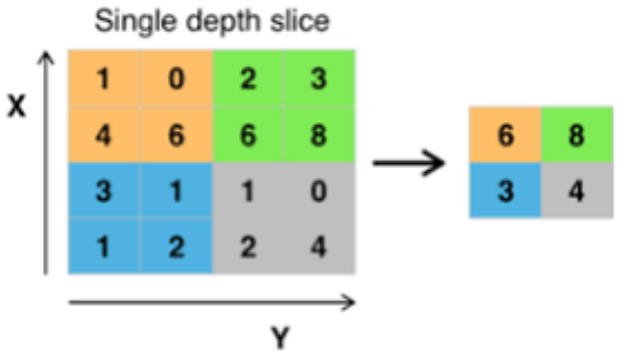
Pictures from "A Beginner's Guide To Understanding Convolutional Neural Networks" (Adit Deshpande's Blog)

Keys to CNN

- Sparse connectivity
- Shared weights
- Pooling (max or weighted average)

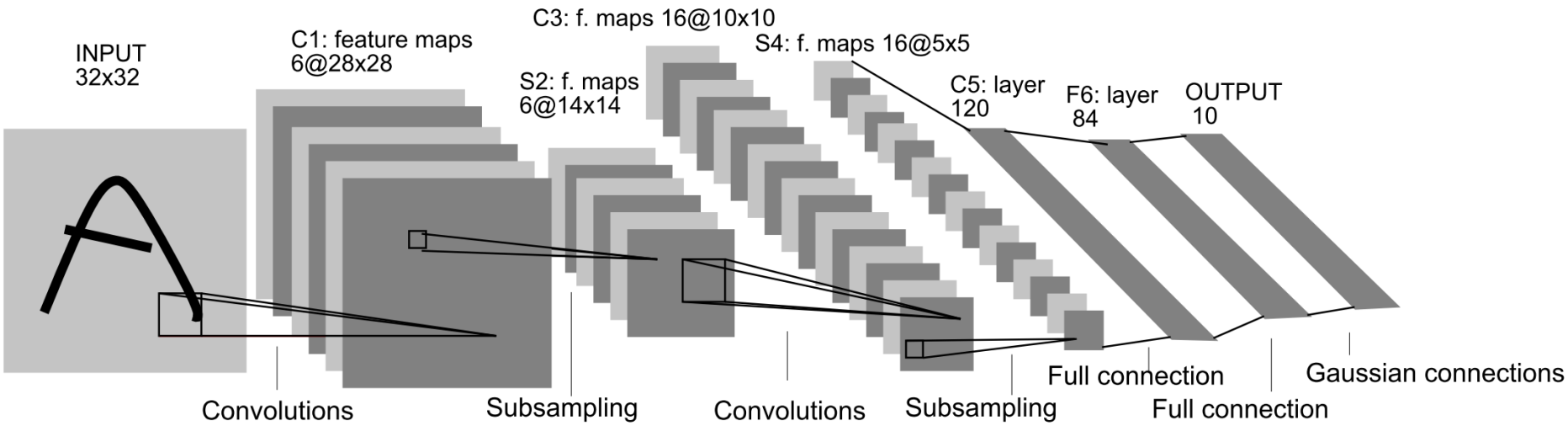






LeNet 5

- LeCun, 1998



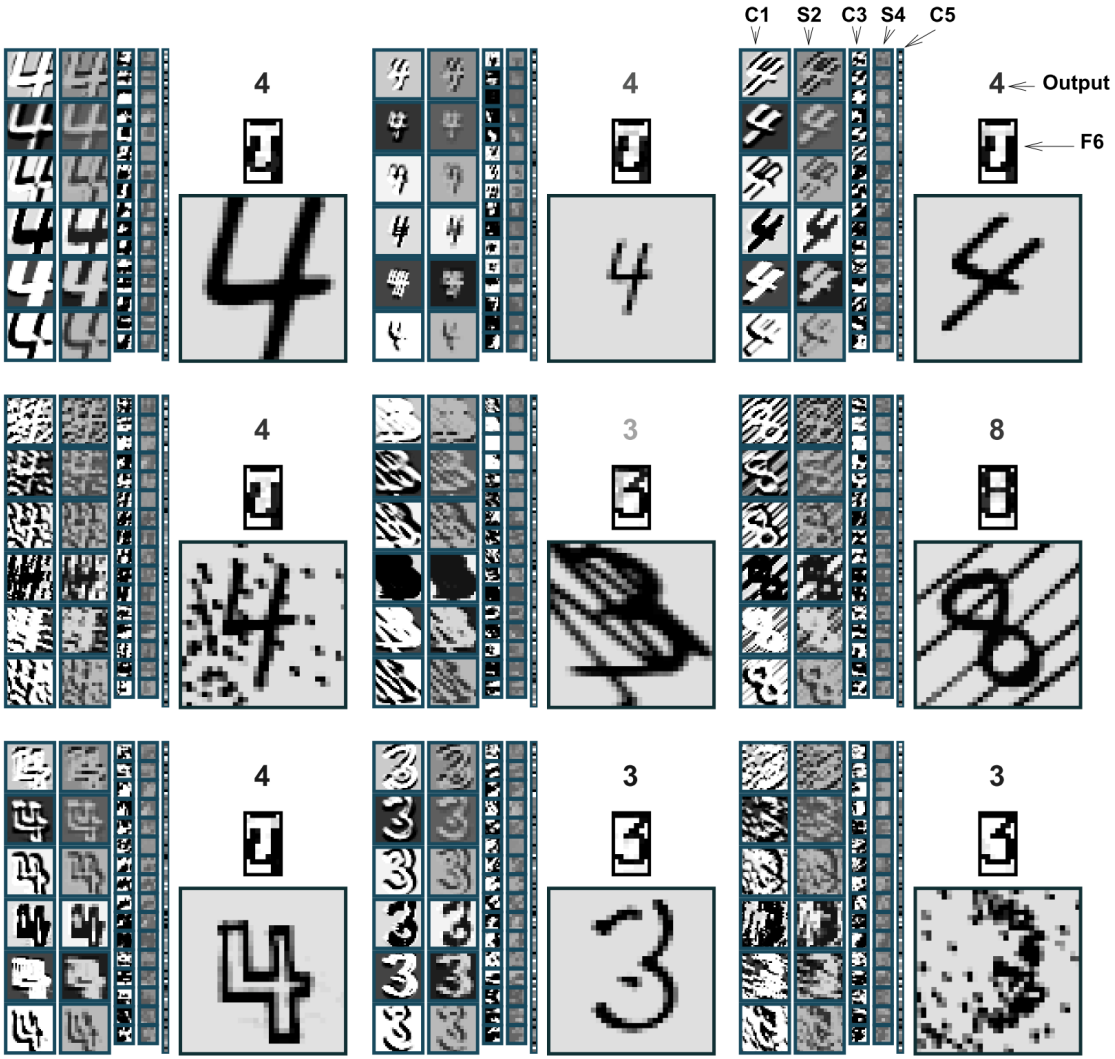
Details of LeNet 5

- C1: $5 \times 5 \times 6$ filter: $(25+1) \times 6 = 156$ parameters.
- S2: $2 \times 2 \times 1$ subsampling filter ($a * sum + b$): $6 \times (1+1) = 12$ parameters.
- C3: 5×5 filter: $6 \times (3 \times 25 + 1) + 9 \times (4 \times 25 + 1) + (25 \times 6 + 1) = 1516$ parameters

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

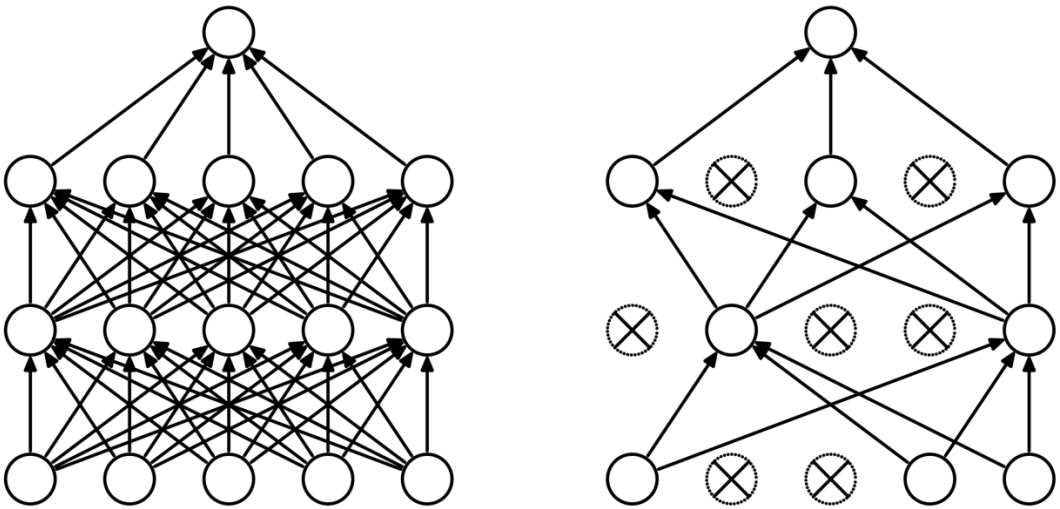
- S4: $16 \times (1+1) = 32$ parameters.
- C5: $5 \times 5 \times 1$ filter: $120 \times (16 \times 25 + 1) = 48120$ parameters.
- F6: fully connected ($8 \times 15 \rightarrow 7 \times 12$): $(120+1) * 84 = 10164$ parameters.
- Final step: $y_i = \sum_j (x_j - w_{ij})^2$

Results of LeNet 5



Dropout

- With a probability, neurons are deactivated during training.
- All neurons are activated during testing (and for real use).
- Similar effect to ensemble (the embedding way).

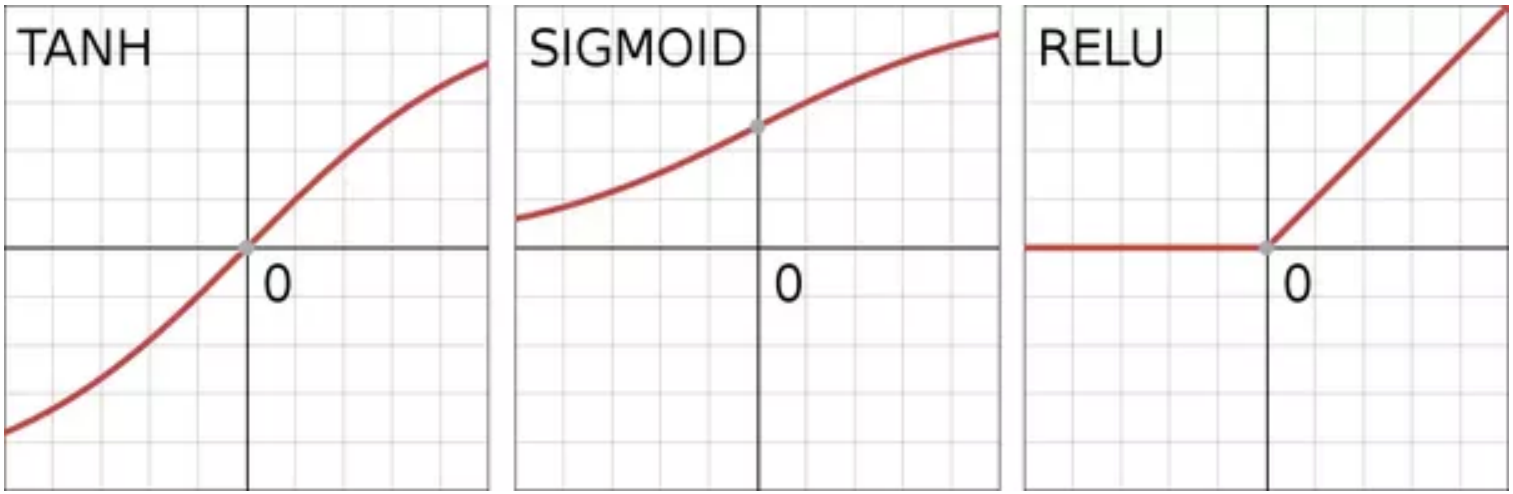


(a) Standard Neural Net

(b) After applying dropout.

Activation Functions

- tanh $(-1, 1)$: $\frac{e^x - e^{-x}}{e^x + e^{-x}}$
- sigmoid $(0, 1)$: $\frac{1}{1+e^{-x}}$
- ReLU (rectified linear unit) $(0, \infty)$: $\max(0, x)$



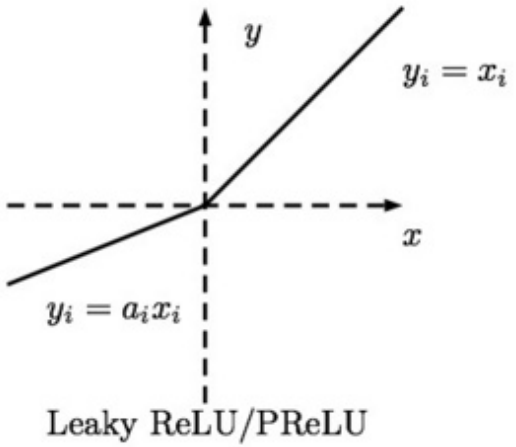
ReLU

- Pros
  - Sparsity
  - Preventing the vanishing gradient problem
  - Fast computation
- Cons
  - Dead ReLU
  - Numerical infeasibility

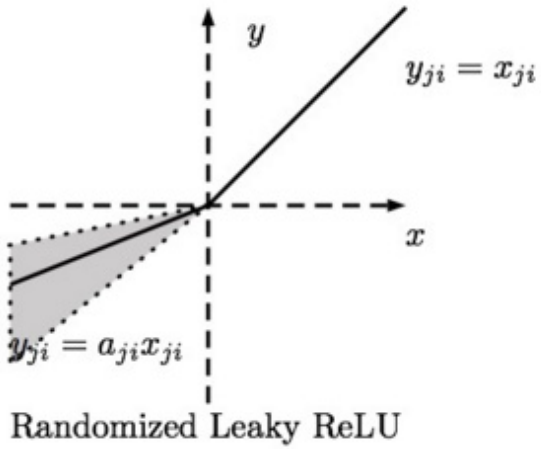
ReLU Variants

- Leaky ReLU, parametric ReLU, randomized leaky ReLU

$$f(x) = \begin{cases} x, & x > 0 \\ ax, & x \leq 0 \end{cases}$$



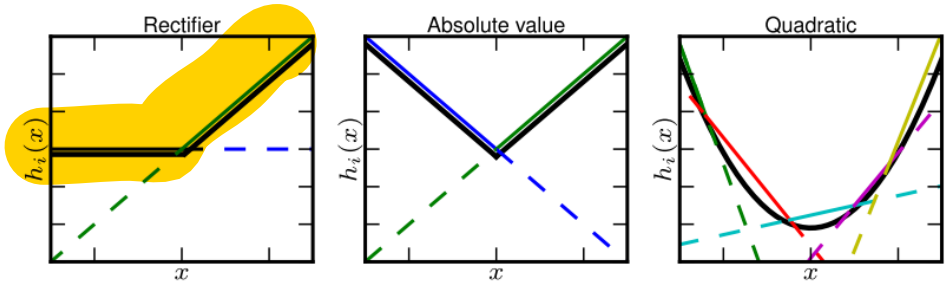
Leaky ReLU/PReLU



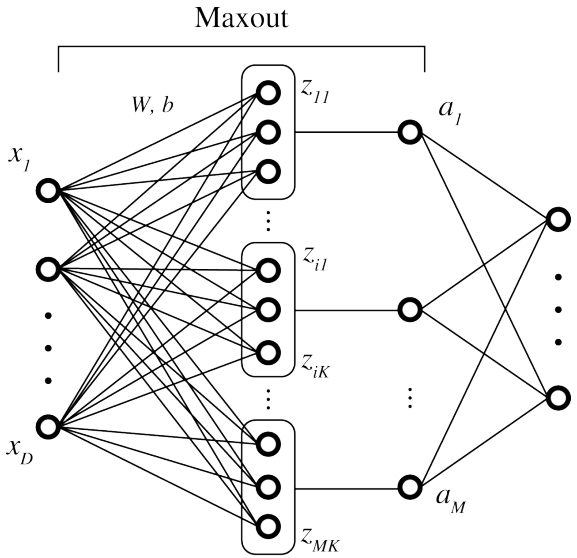
Randomized Leaky ReLU

Maxout

- Goodfellow *et al.*, 2013
- Not dropout, not max pooling.

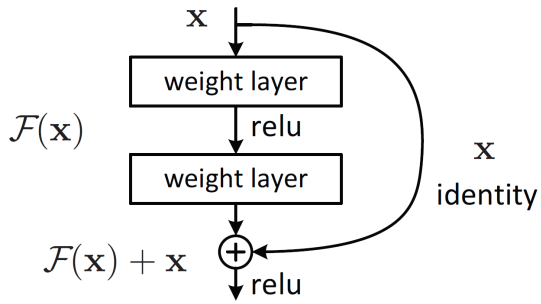


$$h_i(x) = \max_{j \in [1,k]} z_{ij}$$

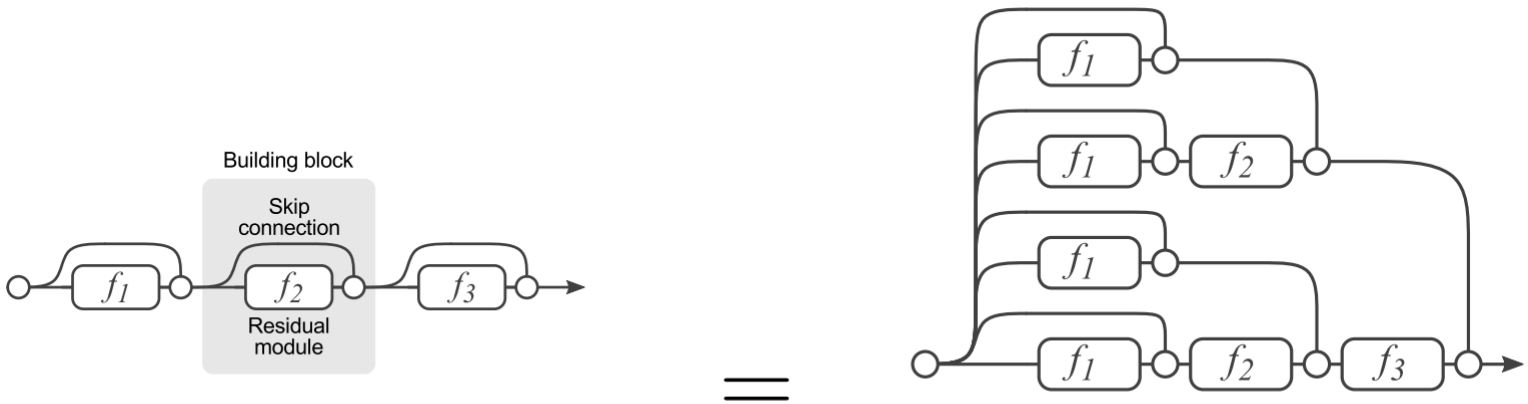


Residual Net

- He *et al.*, 2015 (Microsoft Research).
- 2015 ILSVRC winner, 152 layers.



- Focusing on learning other things than self.
- Similar idea as LSTM.
- Can be viewed as another ensemble.

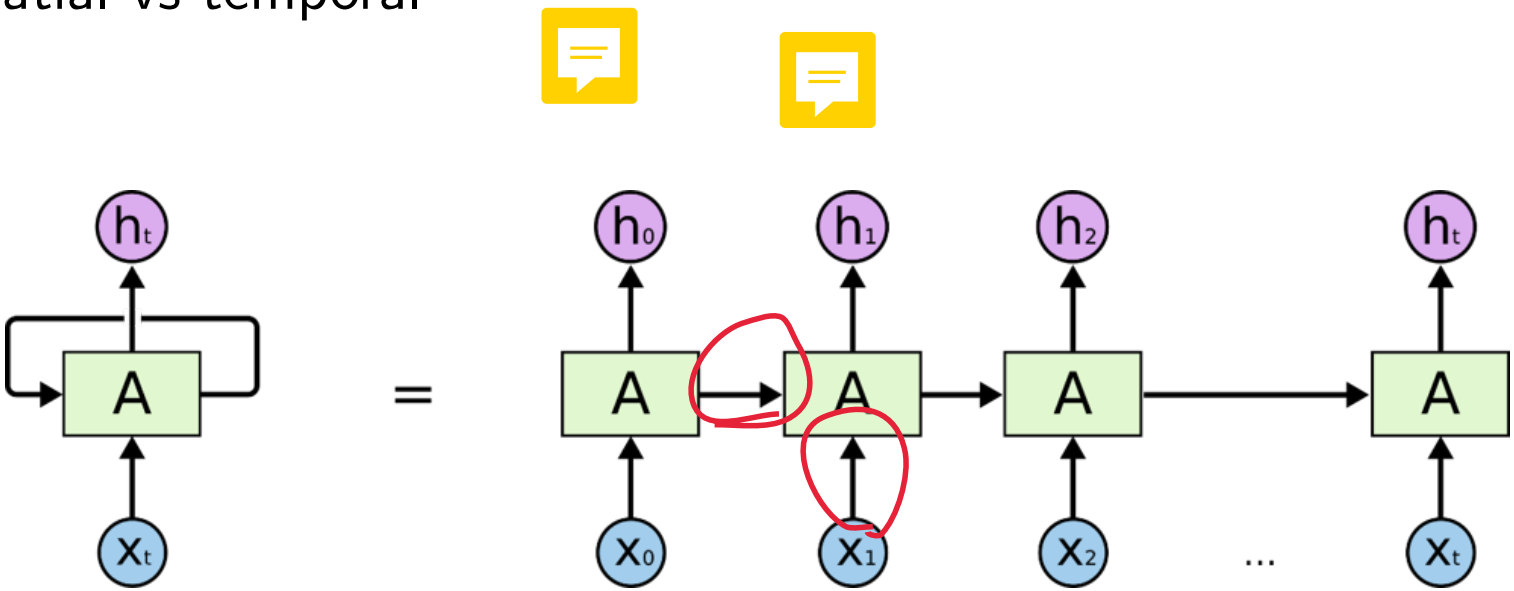


(a) Conventional 3-block residual network

(b) Unraveled view of (a)

Recurrent Networks

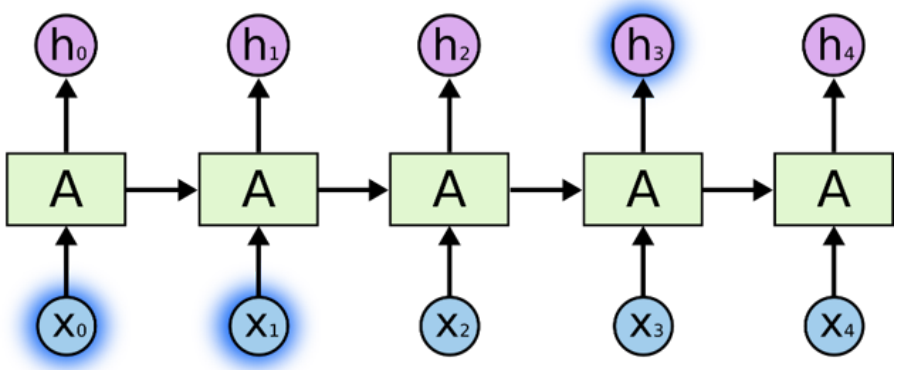
- Not a function anymore
- Internal state (memory)
- Spatial vs temporal



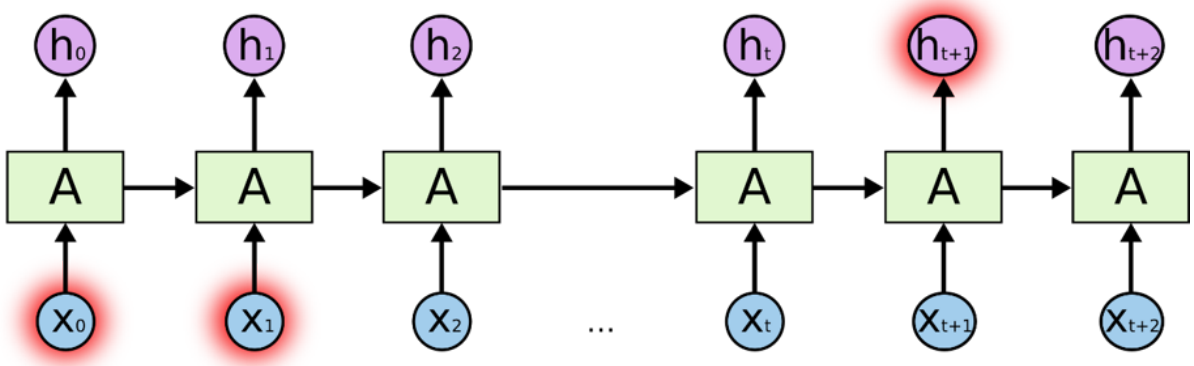
Pictures from "Understanding LSTM Networks" (Christopher Olah's Blog)

Long-Term Dependency

- Short: the clouds are in the **sky**.



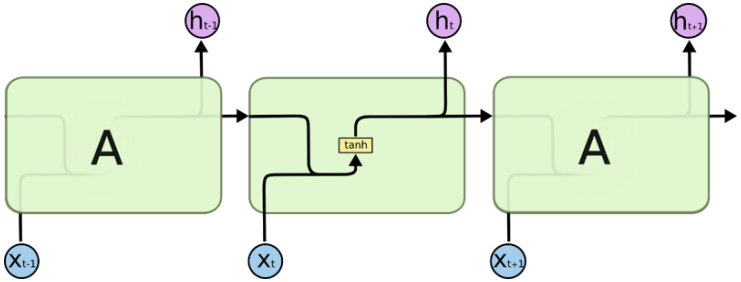
- Long: I grew up in France I speak fluent **French**.



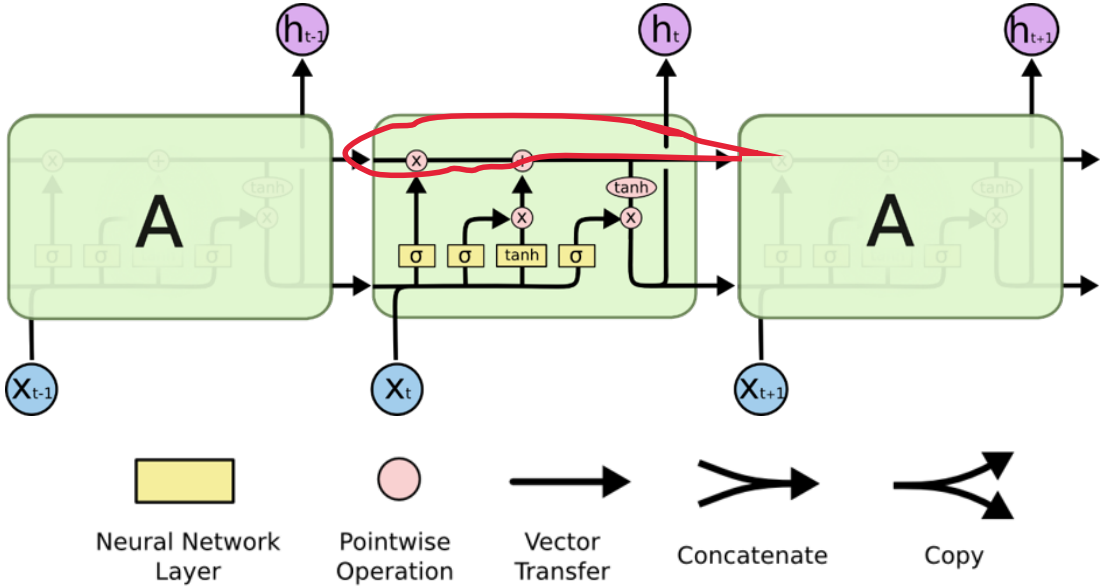
Pictures from "Understanding LSTM Networks" (Christopher Olah's Blog)

Long Short-Term Memory (LSTM)

- Single layer RNN



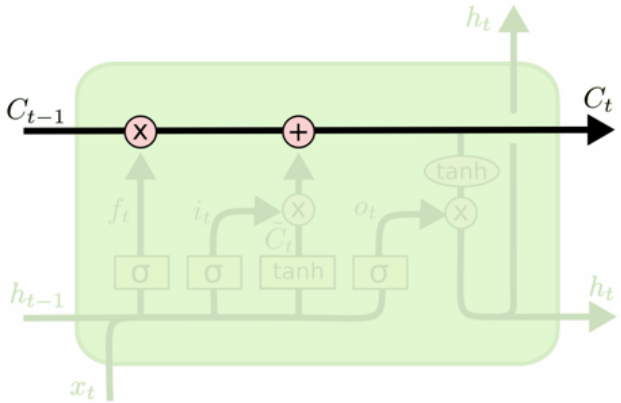
- LSTM



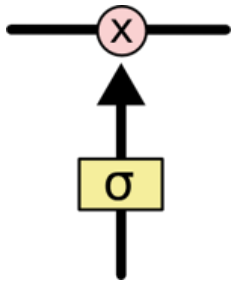
Pictures from "Understanding LSTM Networks" (Christopher Olah's Blog)

Core Idea

- Core state unchanged.



- Control

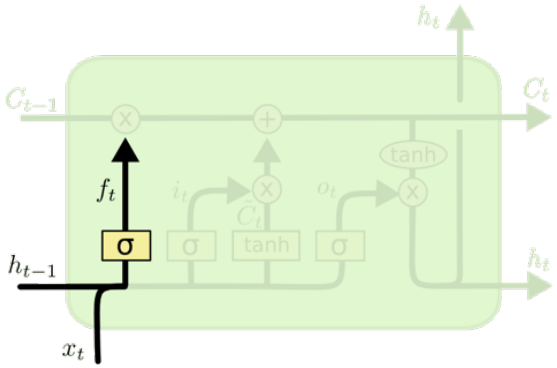


Pictures from "Understanding LSTM Networks" (Christopher Olah's Blog)

- 0 $\rightarrow$ let nothing through
- 1 $\rightarrow$ let everything through

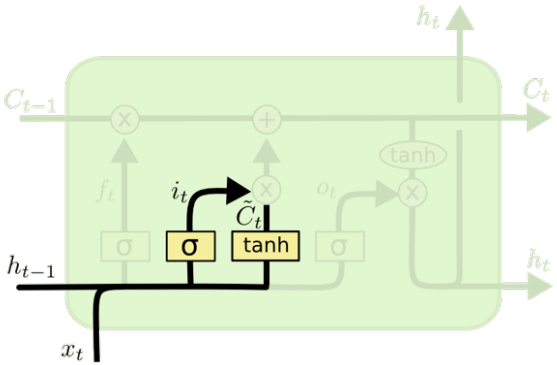
Forget Gate & New Info

- Previous output
- New subject



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

- New information

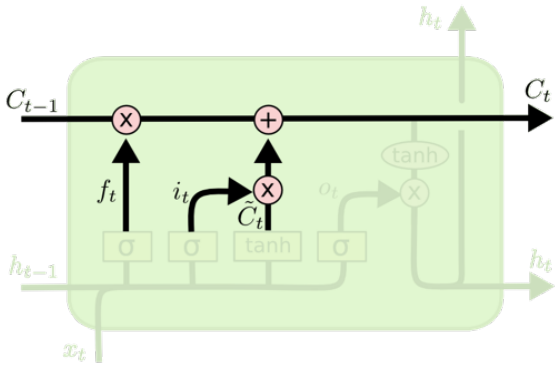


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Pictures from "Understanding LSTM Networks" (Christopher Olah's Blog)

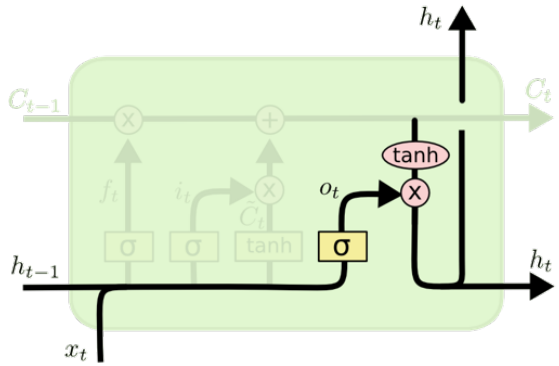
New Cell State & Output

- Linear combination of previous state and new info



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

- Cell state filtered by previous output and new subject

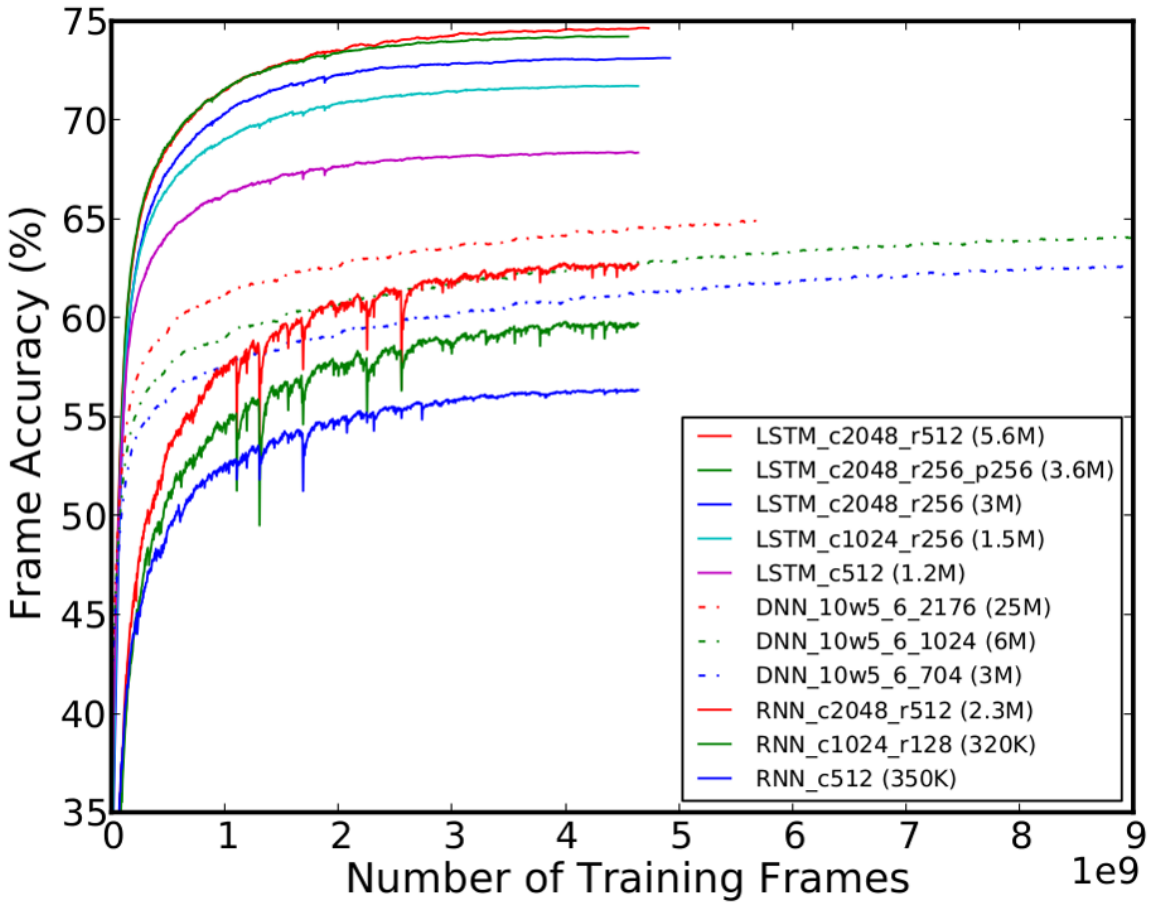


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Pictures from "Understanding LSTM Networks" (Christopher Olah's Blog)

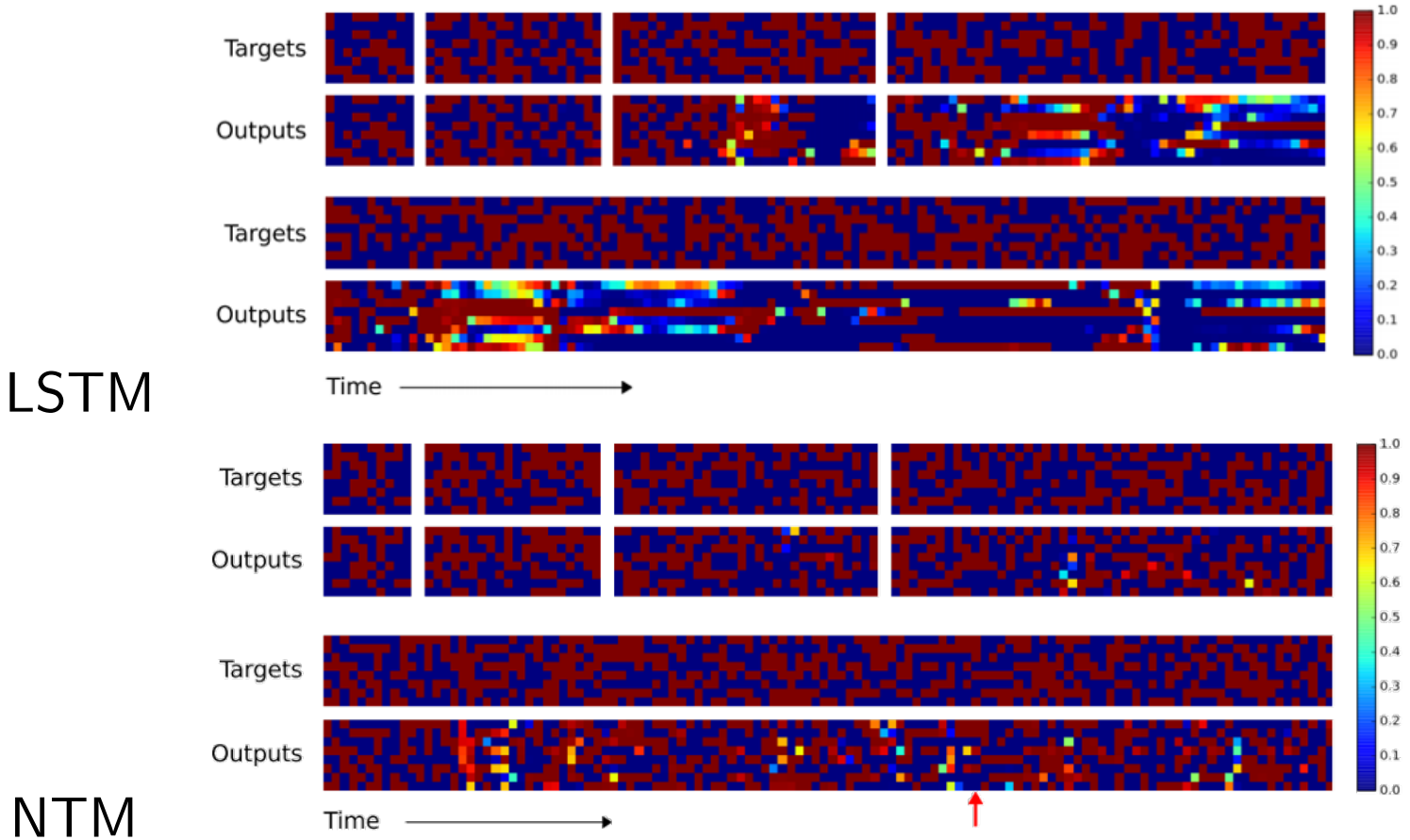
Performance of LSTM

- Sak *et al.*, 2014



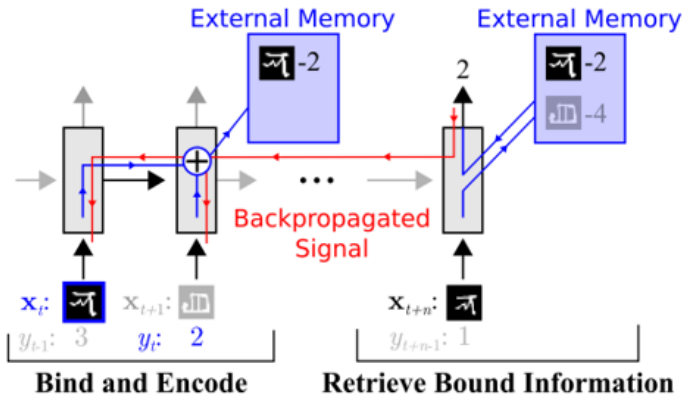
Neural Turing Machine

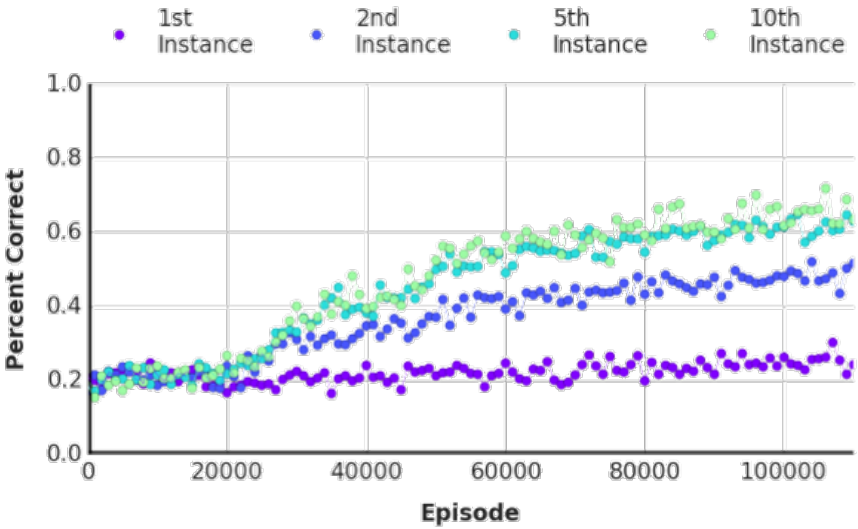
- Graves *et al.*, 2014

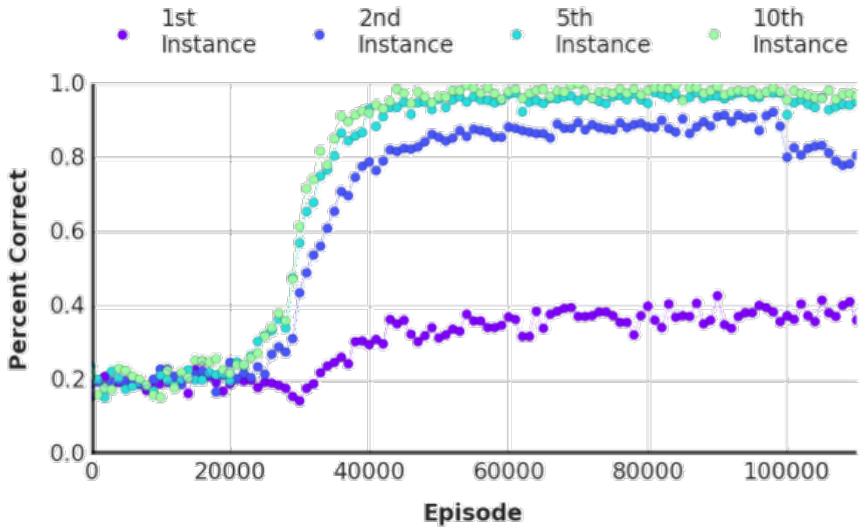


Memory-Augmented Neural Networks (MANN)

- Santoro *et al.*, 2016 (DeepMind)

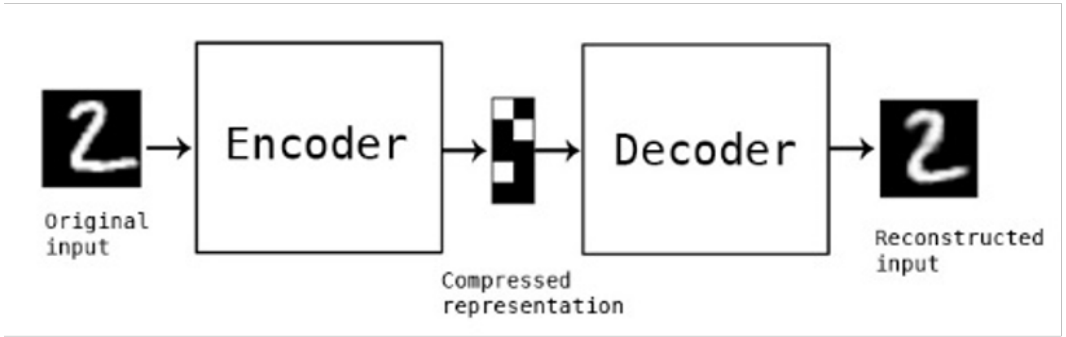


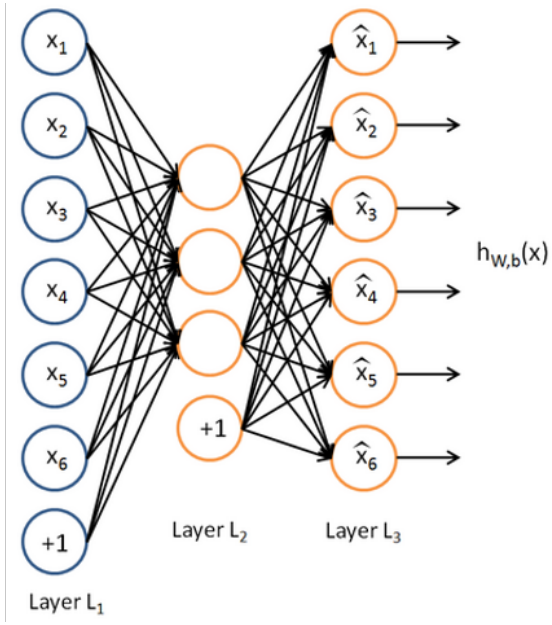




Autoencoder

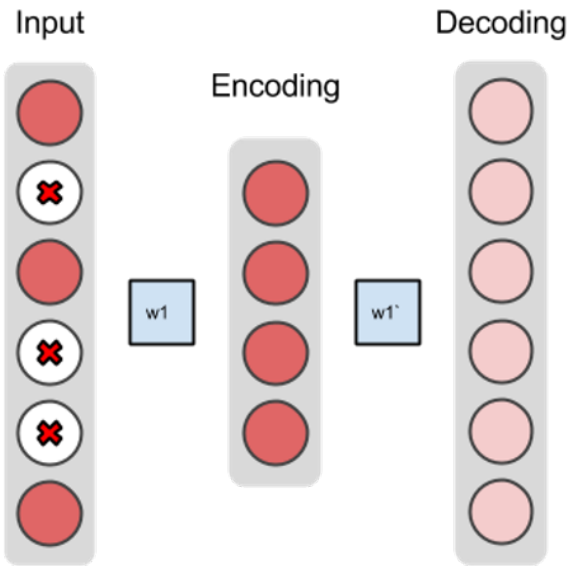
- Unsupervised framework
- Dimension reduction
  - Without nonlinearity activation functions, it is PCA.

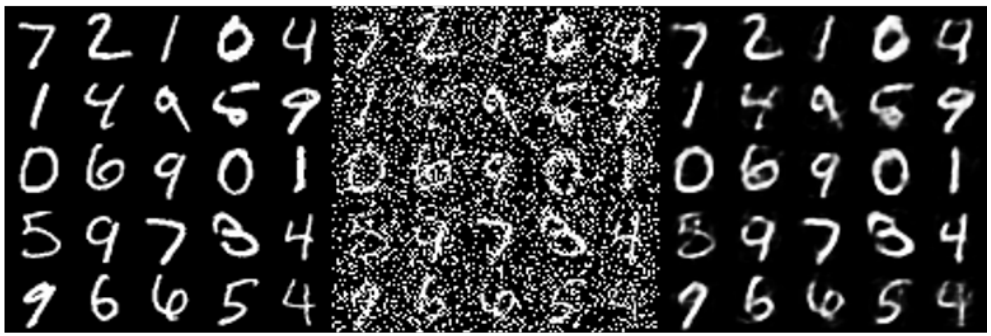




Denoising Autoencoder

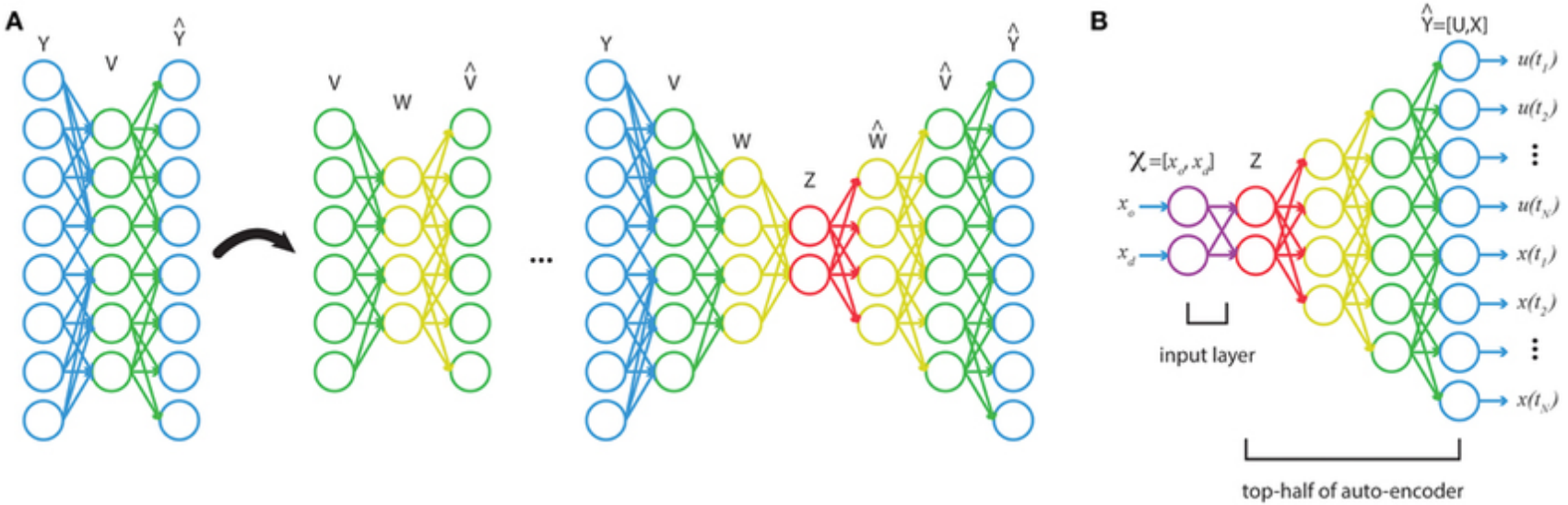
- Similar idea to dropout by adding noise to input.





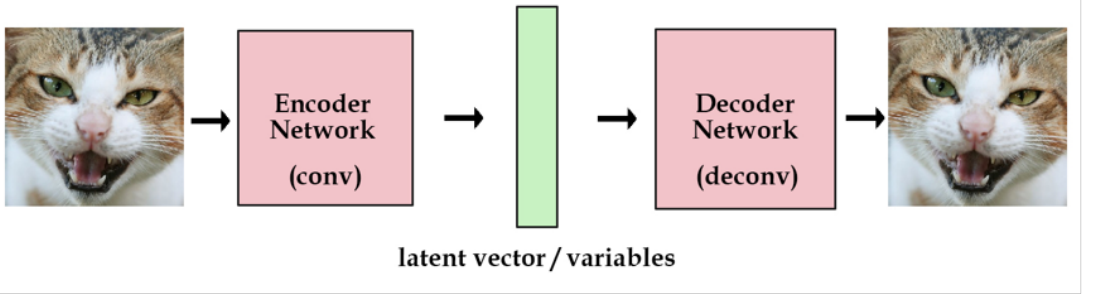
Stacked Autoencoder

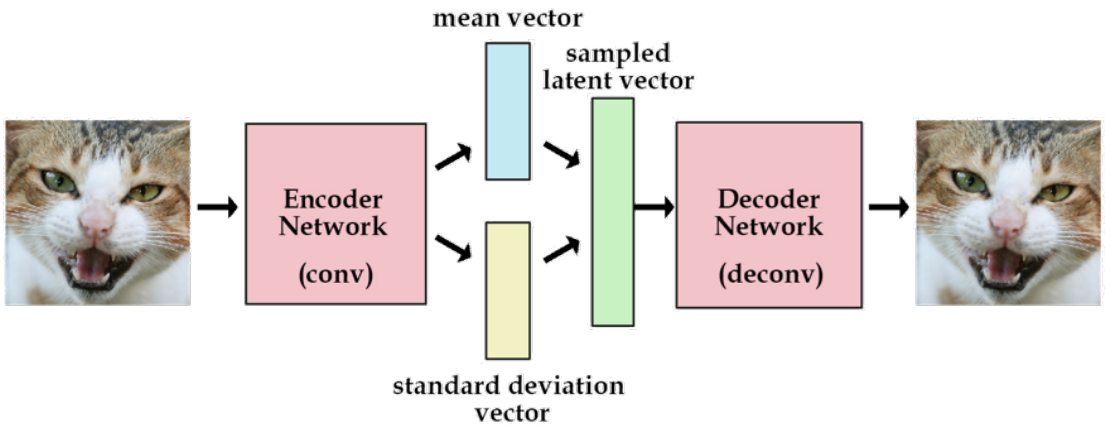
- Learning one layer at a time.
- High-level features.



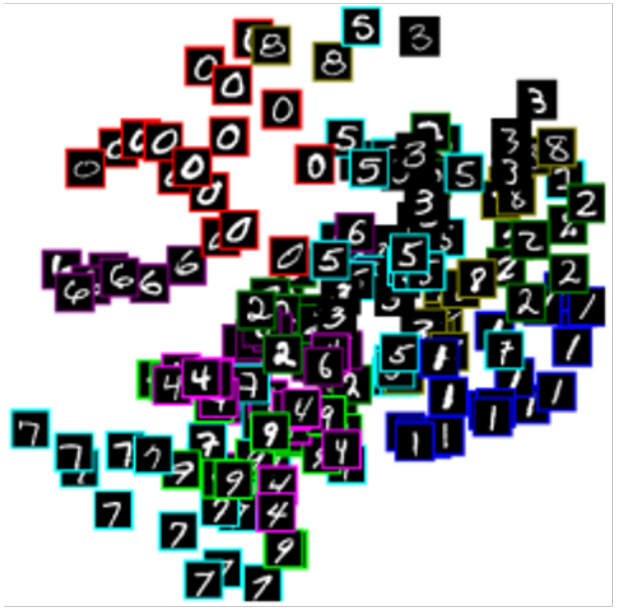
Variational Autoencoder (VAE)

- Hyper-parameters
- Pro: much faster learning
- Con: blurrier output

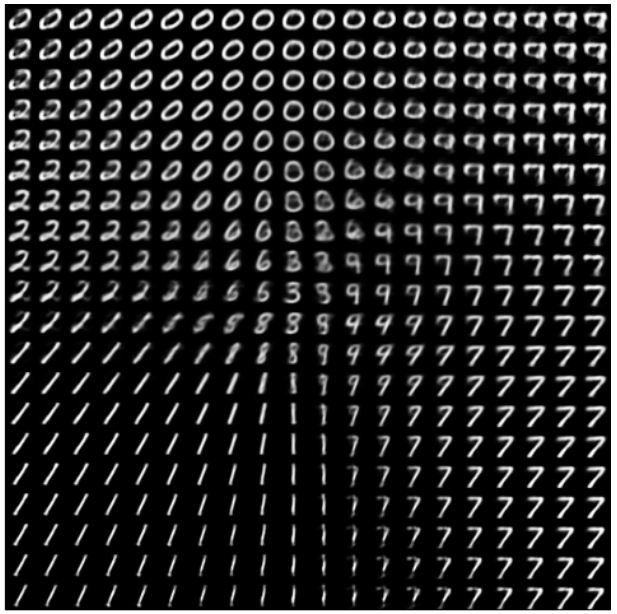




Working on MINIST



Classification

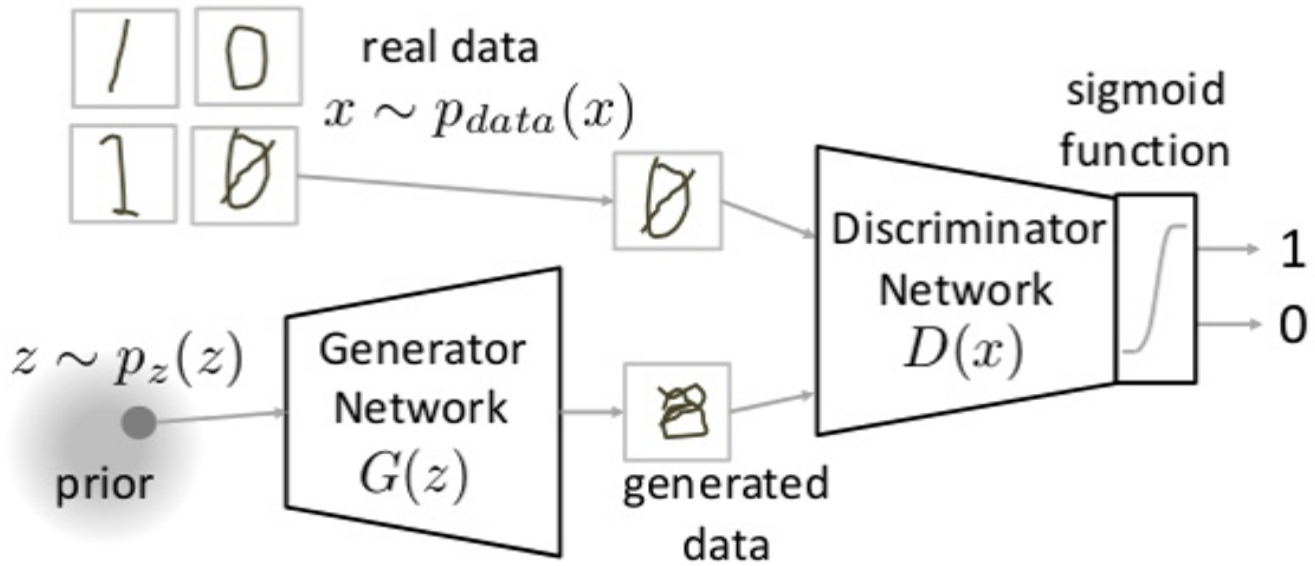


Interpolation

De-noise

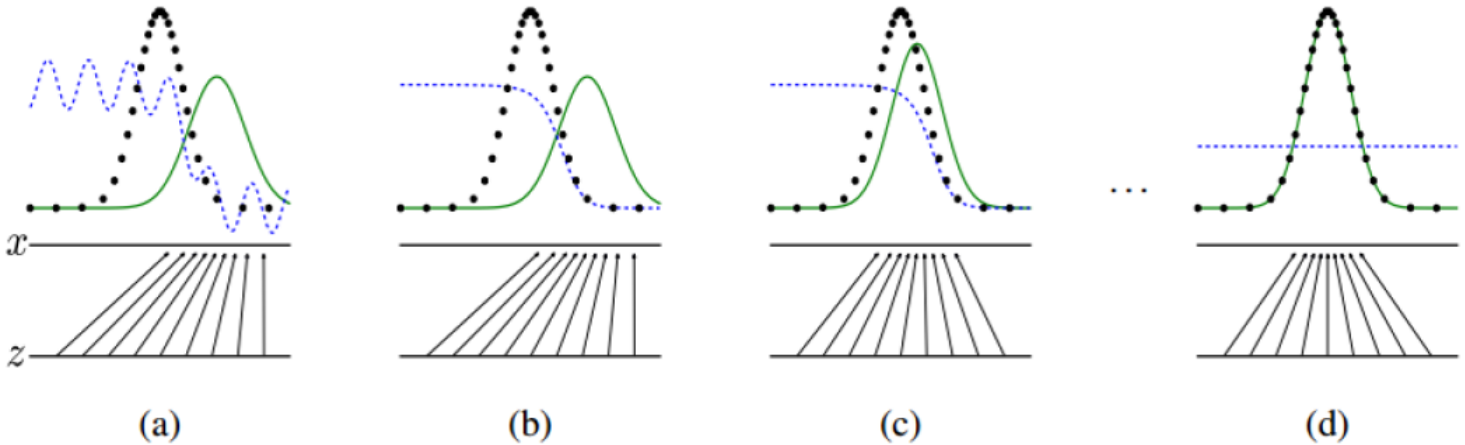
Generative Adversarial Net (GAN)

- Goodfellow *et al.*, 2014
- $G(z)$: generator
  - Random vector $\rightarrow$ sample
- $D(x)$: discriminator
  - Probability of x coming from real data



Conceptual Example

- Eventually discriminator cannot distinguish the generated data from the real data.



Training G and D

- Minimax manner

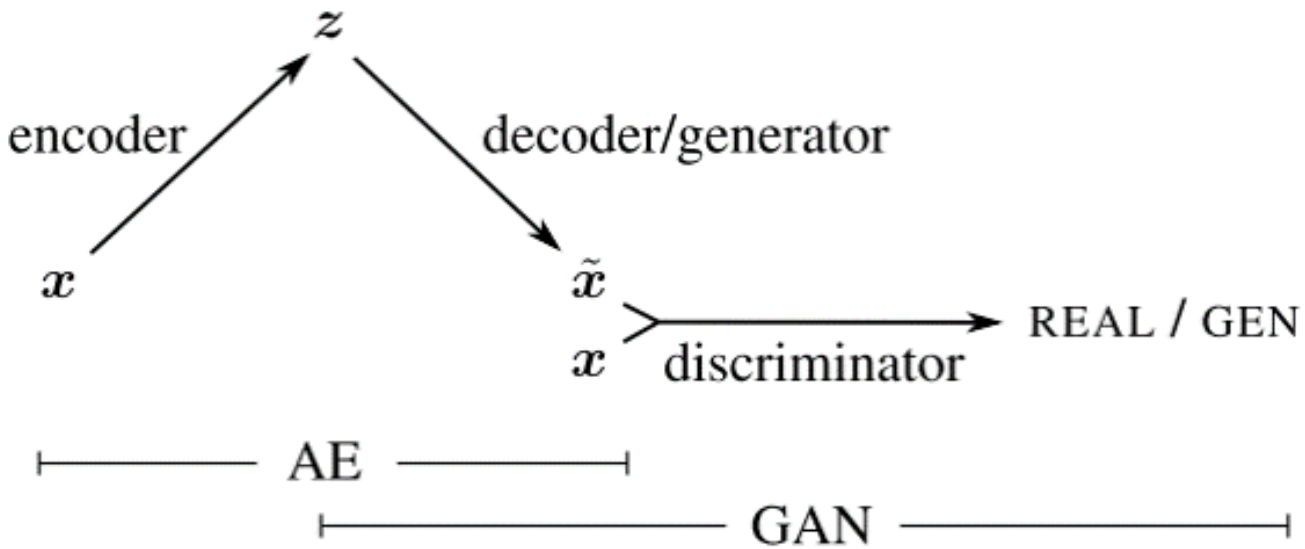
$$\min_G \max_D V(D, G)$$

$$V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

- If G and D have enough capacity, and D reaches its optimum given G, then eventually G is optimum ($p_G = p_{data}$).

AE+GAN

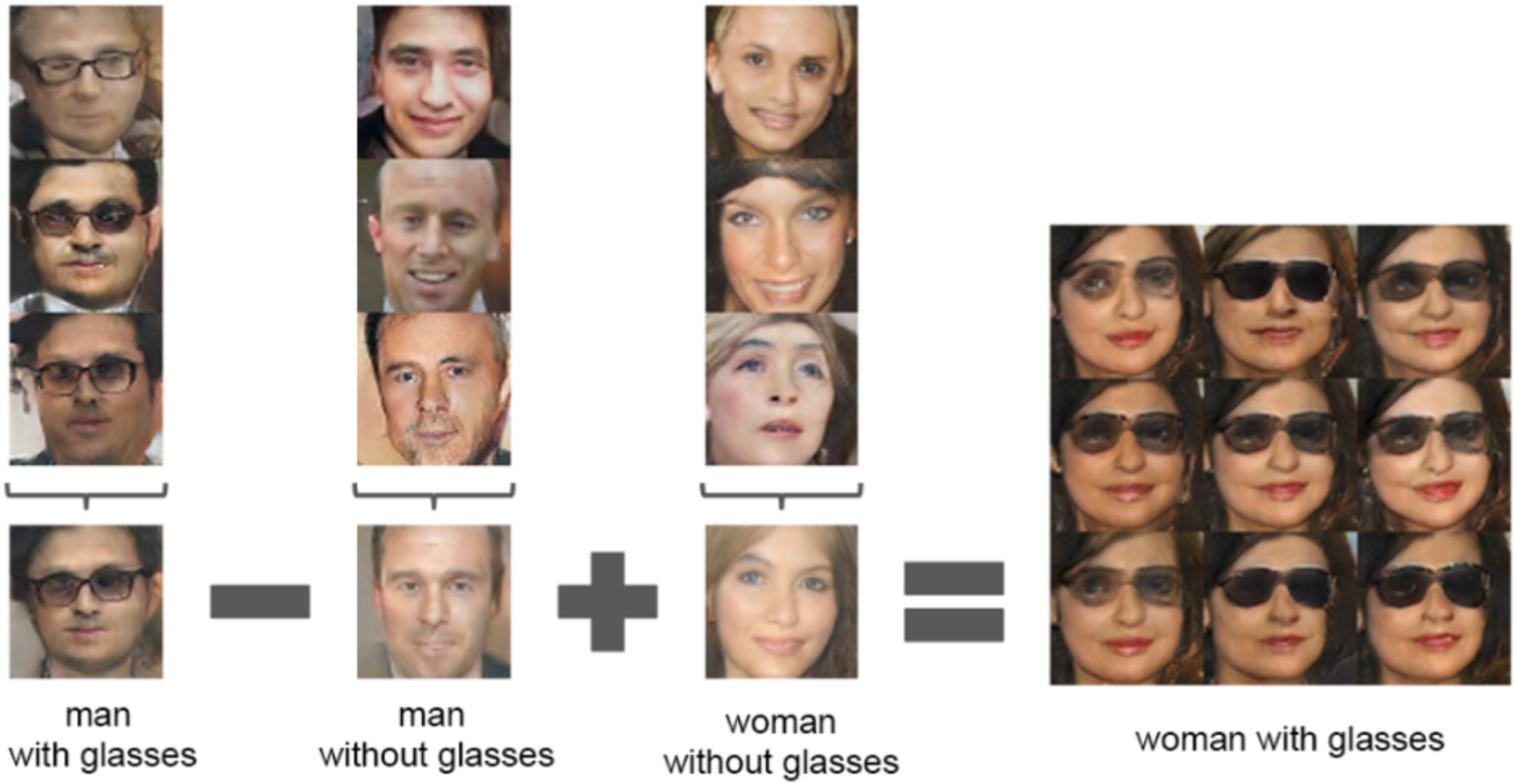
- Larsen *et al.*, 2016
- Radford *et al.*, 2016
- Reed *et al.*, 2016



AE+GAN


Input
Reconstruction
Bald
Bangs
Black hair
Blond hair
Bushy eyebrows
Eyeglasses
Gray hair
Heavy makeup
Male
Mustache
Pale skin

Arithmetic on Faces



Text to Image Synthesis

Summary

- McCulloch-Pitts units simulate neurons.
- For single-layer feed-forward NNs, use perceptron learning rule for step activation function; logistic regression for logistic activation function.
- For multi-layer feed-forward NNs, use back-propagation learning to train the NNs.
- Convolutional networks works well on image and speech recognition.
- New techniques such as pooling, ReLU, dropout, residual nets keep improving the performance.
- The combination of autoencoder and GAN yields interesting results.
- Recurrent networks remember things, and combined with automaton creates new ways of learning.