2019 CS420 Machine Learning, Lecture 4

Neural Networks

Weinan Zhang

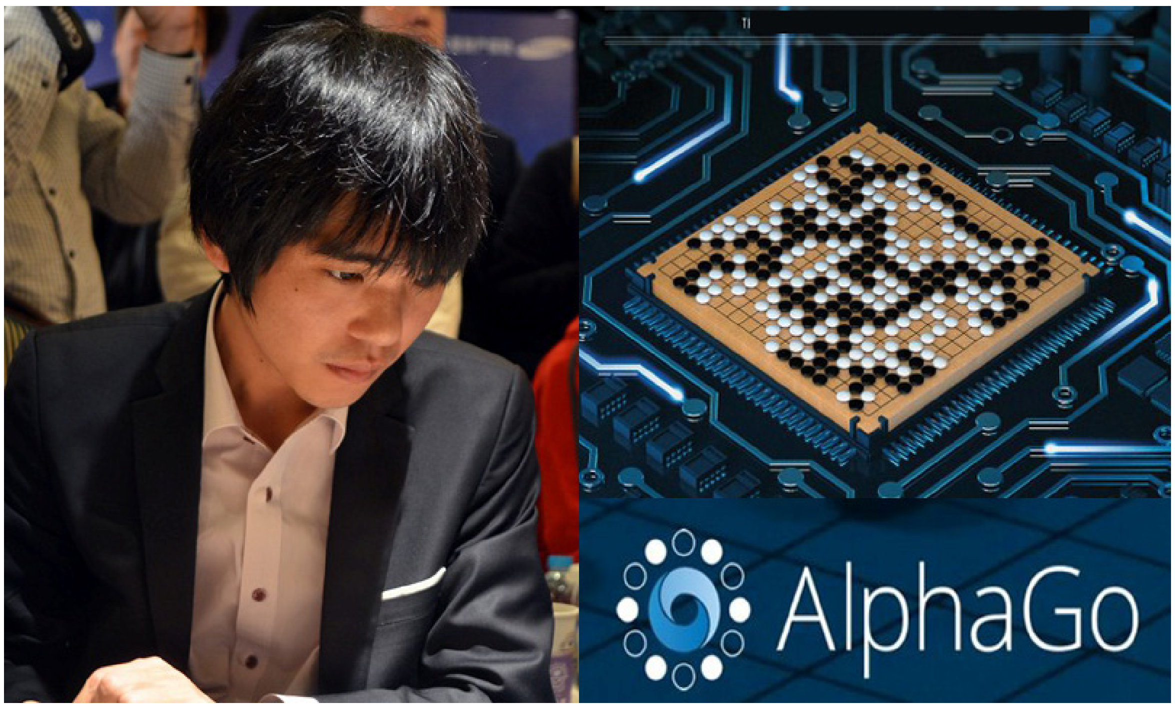
Shanghai Jiao Tong University

http://wnzhang.net

http://wnzhang.net/teaching/cs420/index.html

Breaking News of AI in 2016

- AlphaGo wins Lee Sedol (4-1)



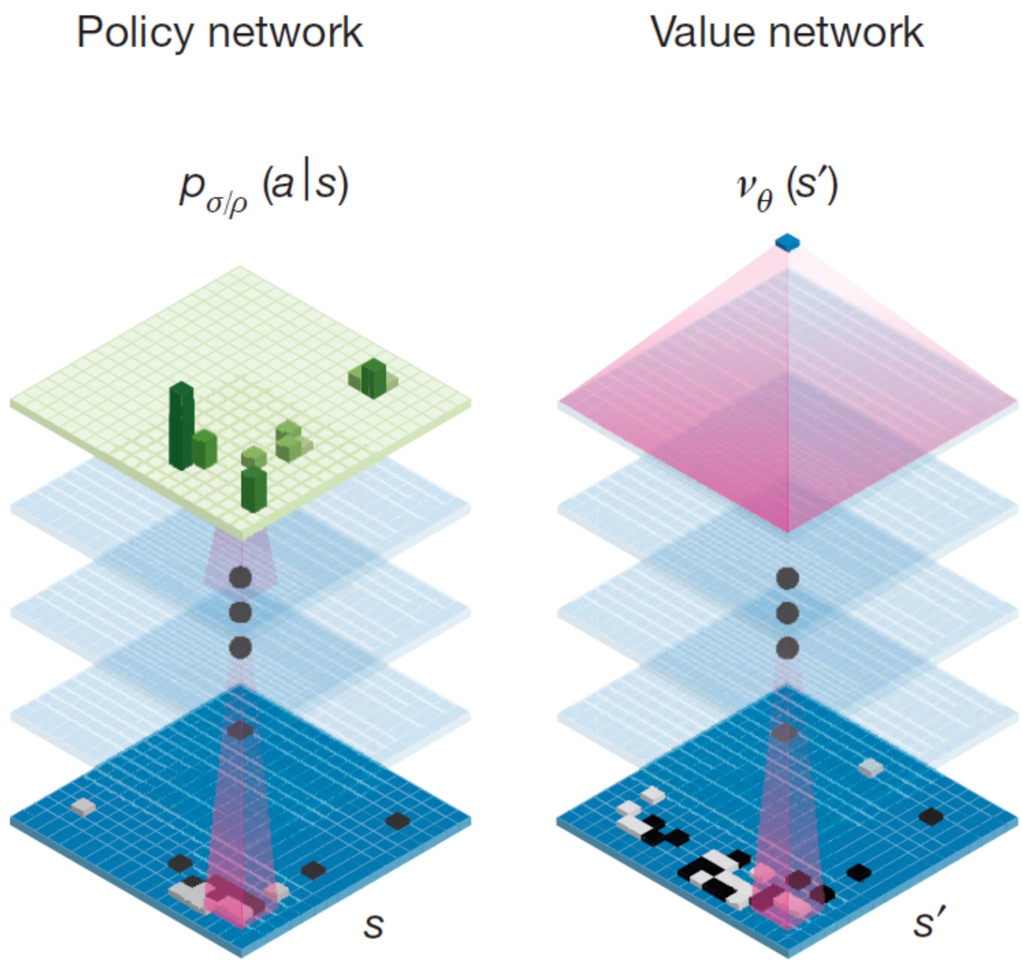
https://deepmind.com/research/alphago/

Rank	Name	↑ ↓	Flag	Elo
1	Ke Jie	↑		3628
2	AlphaGo			3598
3	Park Junghwan	↑		3585
4	Tuo Jiaxi	↑		3535
5	Mi Yuting	↑		3534
6	Iyama Yuta	↑		3525
7	Shi Yue	↑		3522
8	Lee Sedol	↑		3521
9	Zhou Ruiyang	↑		3517
10	Shin Jinseo	↑		3503
11	Chen Yaoye	↑		3495
12	Lian Xiao	↑		3493
13	Tan Xiao	↑		3489
14	Kim Jiseok	↑		3489
15	Choi Cheolhan	↑		3482
16	Park Yeonghun	↑		3482
17	Gu Zihao	↑		3468
18	Fan Yunruo	↑		3468
19	Huang Yunsong	↑		3467
20	Li Qincheng	↑		3465
21	Tang Weixing	↑		3461
22	Lee Donghoon	↑		3460
23	Lee Yeongkyu	↑		3459
24	Fan Tingyu	↑		3459
25	Tong Mengcheng	↑		3447
26	Kang Dongyun	↑		3442
27	Wang Xi	↑		3439
28	Weon Seongjin	↑		3439
29	Yang Dingxin	↑		3439
30	Gu Li	↑		3436

https://www.goratings.org/

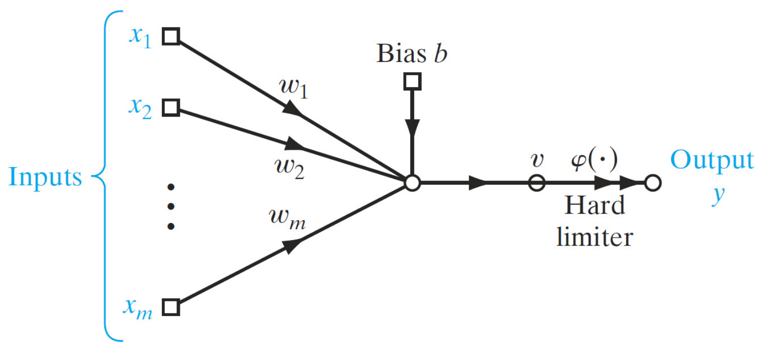
Machine Learning in AlphaGo

- Policy Network
  - Supervised Learning
    - Predict what is the best next human move
  - Reinforcement Learning
    - Learning to select the next move to maximize the winning rate
- Value Network
  - Expectation of winning given the board state
- Implemented by (deep) neural networks

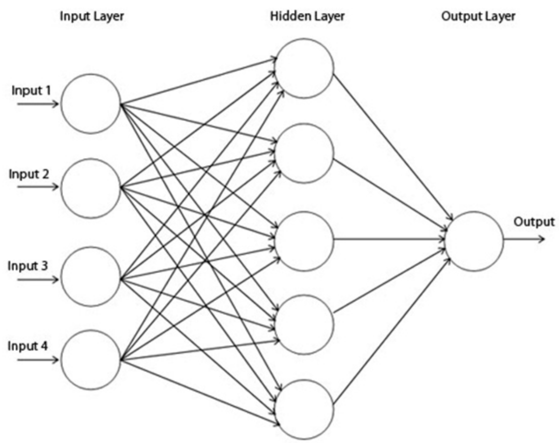


Neural Networks

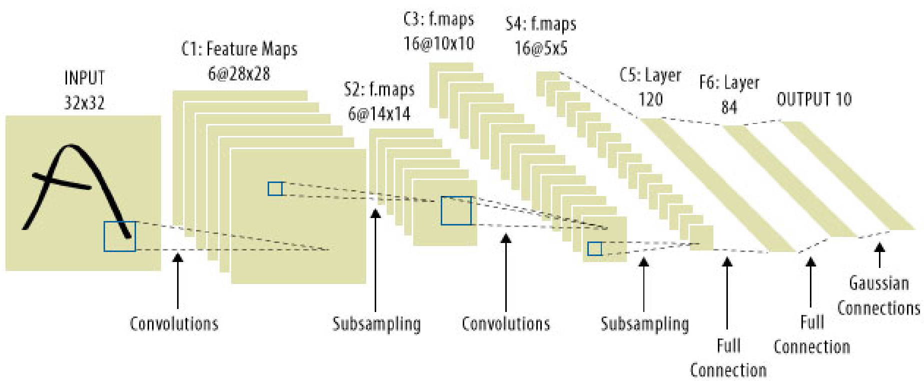
- Neural networks are the basis of deep learning



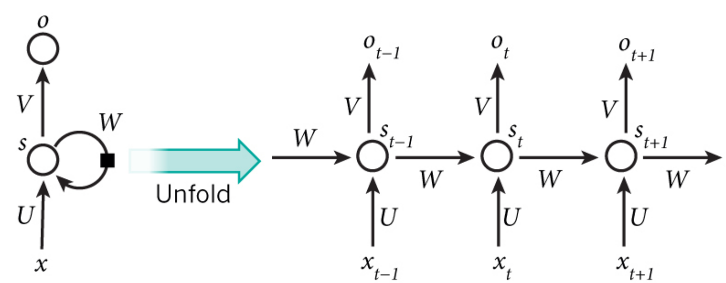
Perceptron



Multi-layer Perceptron



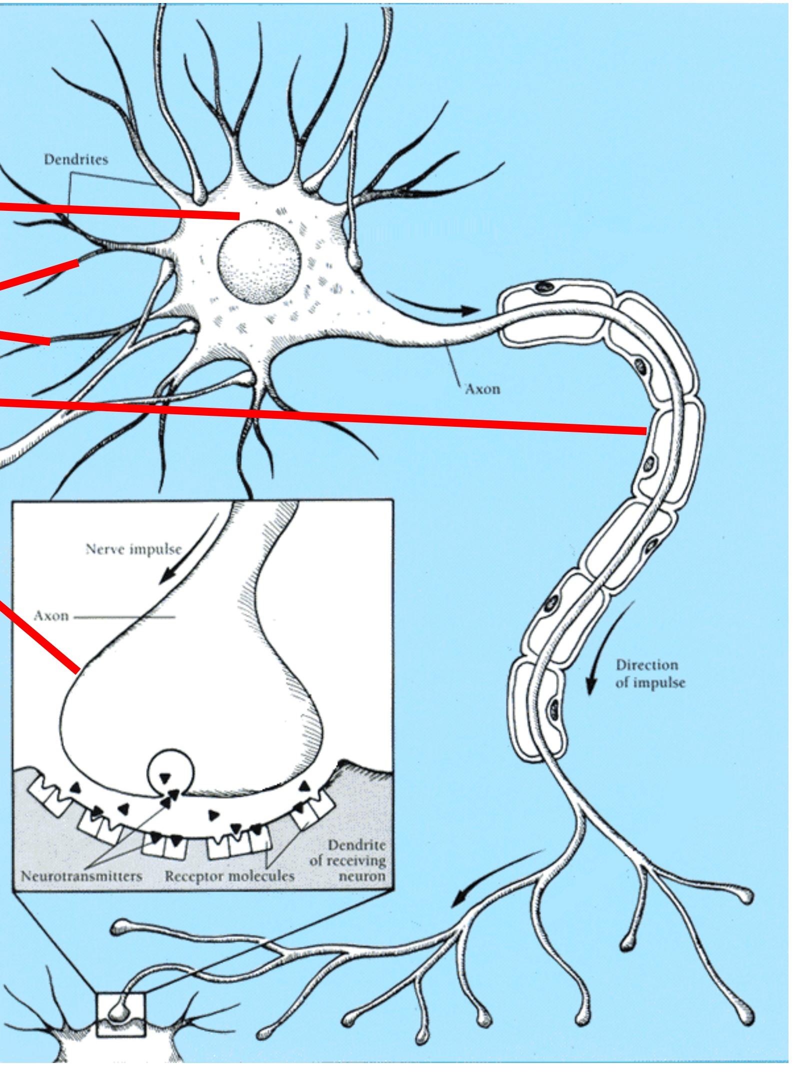
Convolutional Neural Network



Recurrent Neural Network

Real Neurons

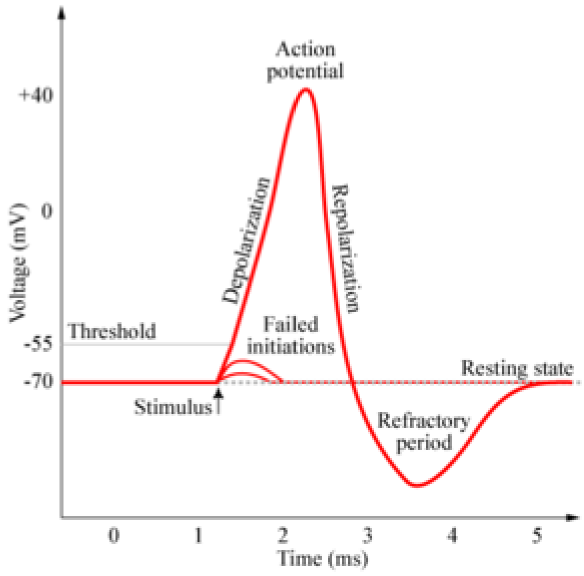
- Cell structures
  - Cell body
  - Dendrites
  - Axon
  - Synaptic terminals



Slides credit: Ray Mooney

Neural Communication

- Electrical potential across cell membrane exhibits spikes called action potentials.
- Spike originates in cell body, travels down axon, and causes synaptic terminals to release neurotransmitters.
- Chemical diffuses across synapse to dendrites of other neurons.
- Neurotransmitters can be excitatory or inhibitory.



- If net input of neurotransmitters to a neuron from other neurons is excitatory and exceeds some threshold, it fires an action potential.

Slides credit: Ray Mooney

Real Neural Learning

- Synapses change size and strength with experience.
- Hebbian learning: When two connected neurons are firing at the same time, the strength of the synapse between them increases.
- “Neurons that fire together, wire together.”
- These motivate the research of artificial neural nets

Slides credit: Ray Mooney

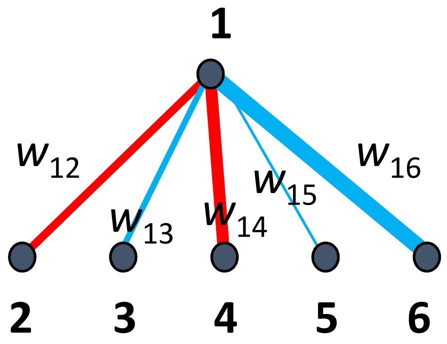
Brief History of Artificial Neural Nets

- The First wave
  - 1943 McCulloch and Pitts proposed the McCulloch-Pitts neuron model
  - 1958 Rosenblatt introduced the simple single layer networks now called Perceptrons.
  - 1969 Minsky and Papert's book Perceptrons demonstrated the limitation of single layer perceptrons, and almost the whole field went into hibernation.
- The Second wave
  - 1986 The Back-Propagation learning algorithm for Multi-Layer Perceptrons was rediscovered and the whole field took off again.
- The Third wave
  - 2006 Deep (neural networks) Learning gains popularity and
  - 2012 made significant break-through in many applications.

Slides credit: Jun Wang

Artificial Neuron Model

- Model network as a graph with cells as nodes and synaptic connections as weighted edges from node *i* to node *j*, w_{ji}



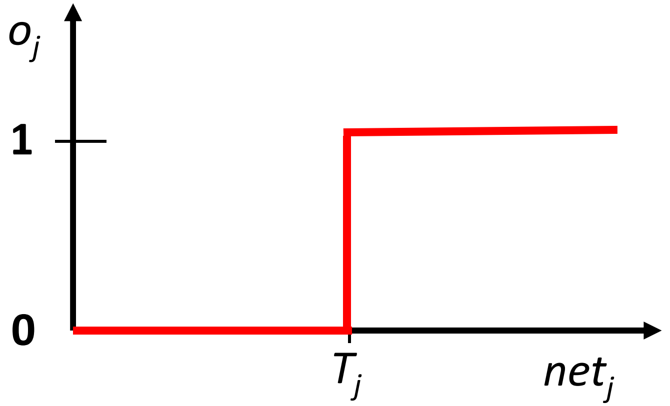
- Model net input to cell as

$$\text{net}_j = \sum_i w_{ji} o_i$$

- Cell output is

$$o_j = \begin{cases} 0 & \text{if net}_j < T_j \\ 1 & \text{if net}_j \geq T_j \end{cases}$$

(T_j is threshold for unit *j*)

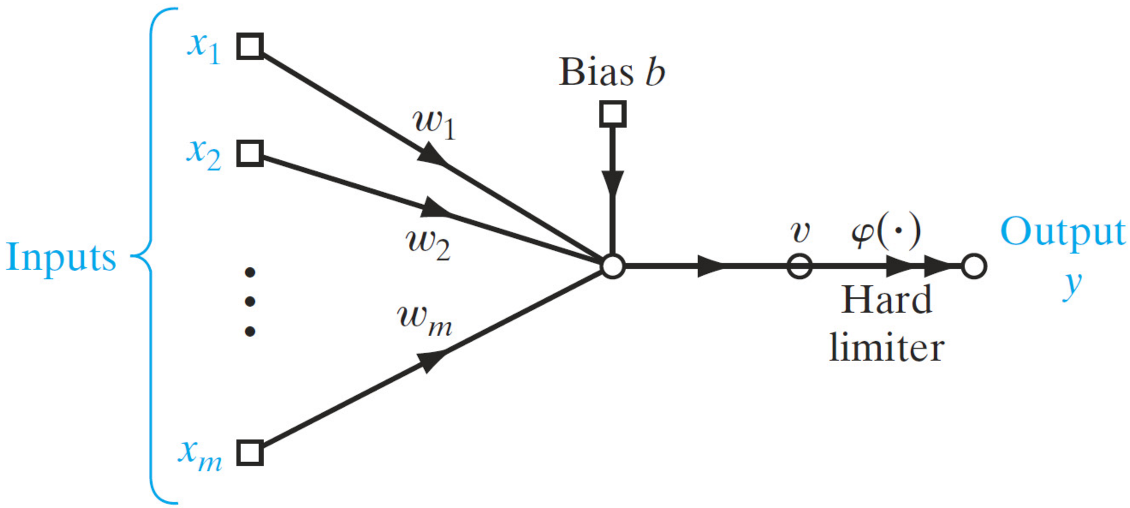


McCulloch and Pitts [1943]

Slides credit: Ray Mooney

Perceptron Model

- Rosenblatt's single layer perceptron [1958]



- Rosenblatt [1958] further proposed the *perceptron* as the first model for learning with a teacher (i.e., supervised learning)
- Focused on how to find appropriate weights w_m for two-class classification task
  - y = 1: class one
  - y = -1: class two

- Prediction

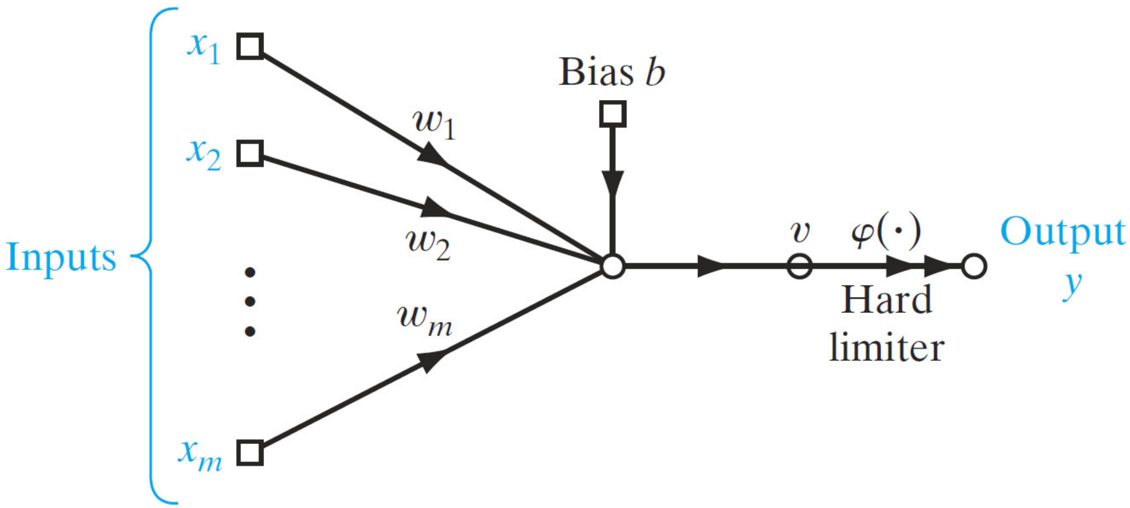
$$\hat{y} = \varphi\Big(\sum_{i=1}^{m} w_i x_i + b\Big)$$

- Activation function

$$\varphi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Training Perceptron

- Rosenblatt's single layer perceptron [1958]



- Training

$$w_i = w_i + \eta(y - \hat{y})x_i$$

$$b = b + \eta(y - \hat{y})$$

- Equivalent to rules:
  - If output is correct, do nothing
  - If output is high, lower weights on active inputs
  - If output is low, increase weights on active inputs

- Prediction

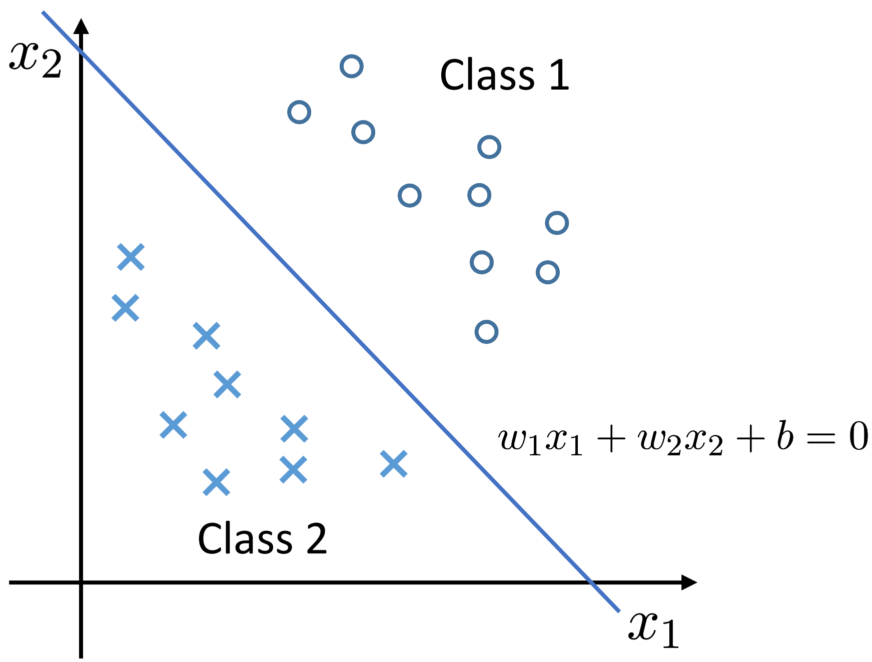
$$\hat{y} = \varphi\left(\sum_{i=1}^{m} w_i x_i + b\right)$$

- Activation function

$$\varphi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Properties of Perceptron

- Rosenblatt's single layer perceptron [1958]

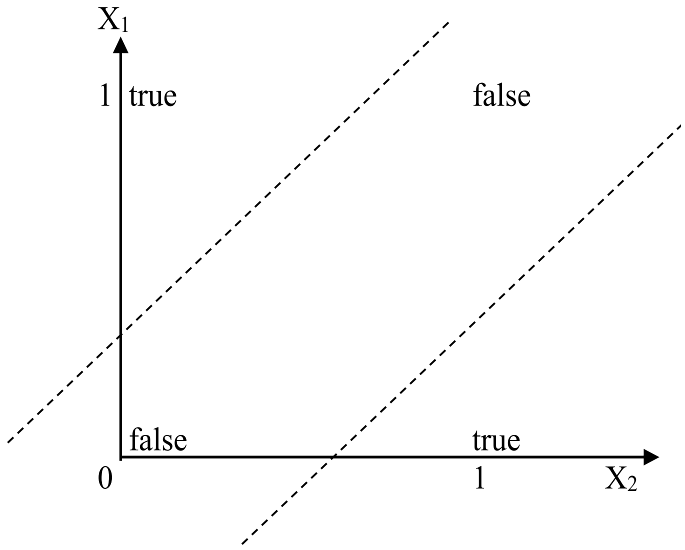


- Rosenblatt proved the convergence of a learning algorithm if two classes said to be linearly separable (i.e., patterns that lie on opposite sides of a hyperplane)
- Many people hoped that such a machine could be the basis for artificial intelligence

Properties of Perceptron

- The XOR problem

Input x		Output y
X_1	X_2	X_1 XOR X_2
0	0	0
0	1	1
1	0	1
1	1	0

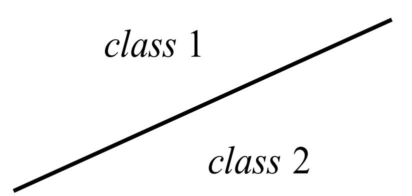


XOR is non linearly separable: These two classes (true and false) cannot be separated using a line.

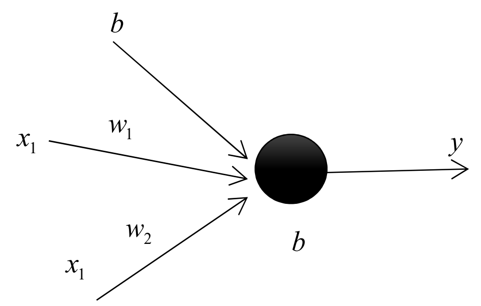
- However, Minsky and Papert [1969] showed that some rather elementary computations, such as *XOR* problem, could not be done by Rosenblatt's one-layer perceptron
- However Rosenblatt believed the limitations could be overcome if more layers of units to be added, but no learning algorithm known to obtain the weights yet
- Due to the lack of learning algorithms people left the neural network paradigm for almost 20 years

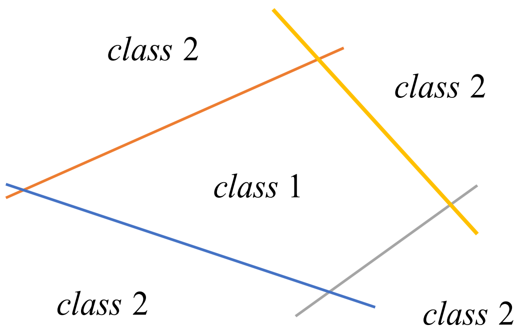
Hidden Layers and Backpropagation (1986~)

- Adding hidden layer(s) (internal presentation) allows to learn a mapping that is not constrained by linearly separable

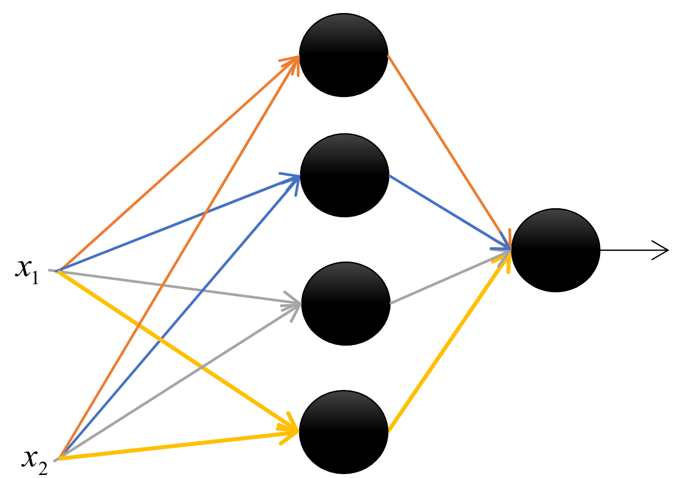


decision boundary: $x_1 w_1 + x_2 w_2 + b = 0$





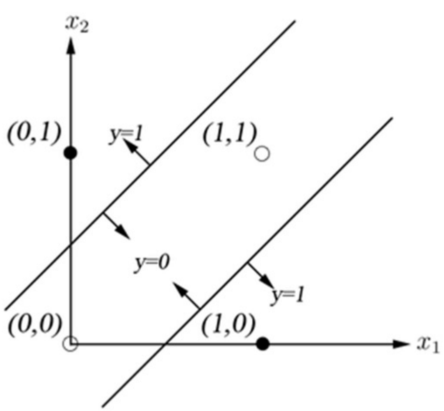
Each hidden node realizes one of the lines bounding the convex region



Hidden Layers and Backpropagation (1986~)

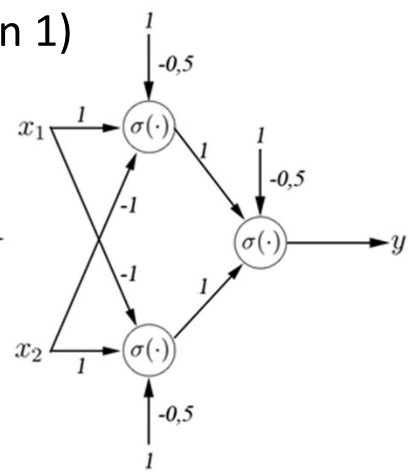
- But the solution is quite often not unique

Input x		Output y
X_1	X_2	X_1 XOR X_2
0	0	0
0	1	1
1	0	1
1	1	0



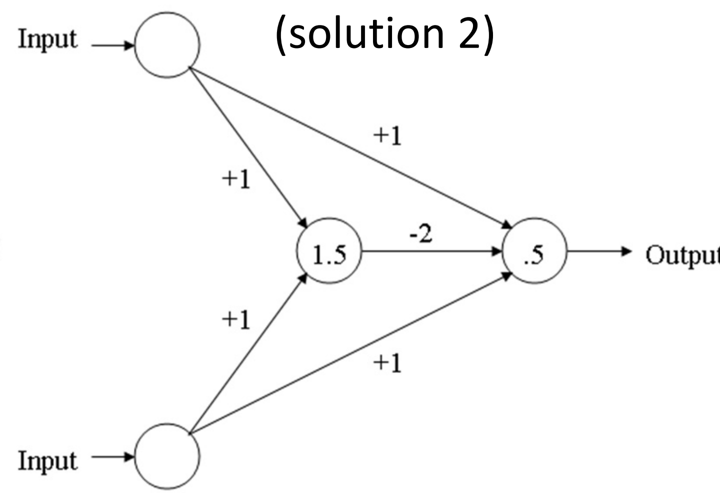
Two lines are necessary to divide the sample space accordingly

(solution 1)



Sign activation function

(solution 2)

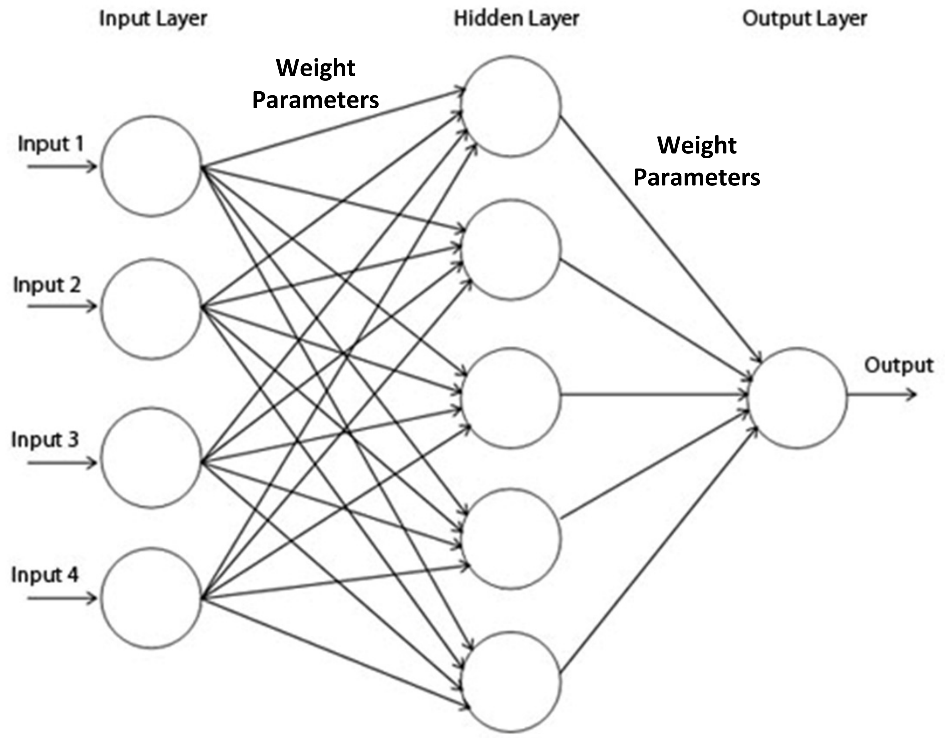


The number in the circle is a threshold

http://www.cs.stir.ac.uk/research/publications/techreps/pdf/TR148.pdf
http://recognize-speech.com/basics/introduction-to-artificial-neural-networks

Hidden Layers and Backpropagation (1986~)

- Feedforward: massages move forward from the input nodes, through the hidden nodes (if any), and to the output nodes. There are no cycles or loops in the network

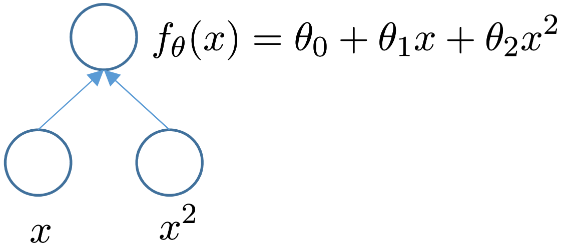


Two-layer feedforward neural network

Single / Multiple Layers of Calculation

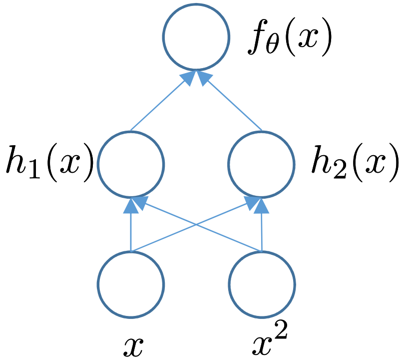
- Single layer function

$$f_\theta(x) = \sigma(\theta_0 + \theta_1 x + \theta_2 x^2)$$



- Multiple layer function

$$h_1(x) = \tanh(\theta_0 + \theta_1 x + \theta_2 x^2)$$
$$h_2(x) = \tanh(\theta_3 + \theta_4 x + \theta_5 x^2)$$
$$f_\theta(x) = f_\theta(h_1(x), h_2(x)) = \sigma(\theta_6 + \theta_7 h_1 + \theta_8 h_2)$$



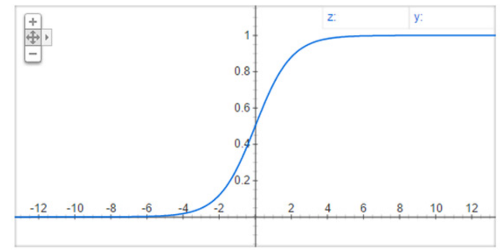
  - With non-linear activation function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad \tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

Non-linear Activation Functions

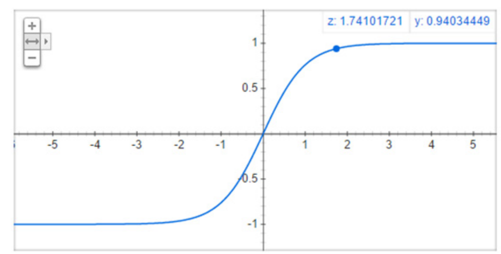
- Sigmoid

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



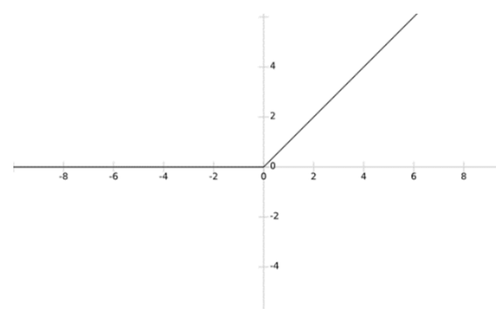
- Tanh

$$\tanh(z) = \frac{1 - e^{-2z}}{1 + e^{-2z}}$$



- Rectified Linear Unit (ReLU)

$$\text{ReLU}(z) = \max(0, z)$$



Universal Approximation Theorem

- A feed-forward network with a single hidden layer containing a finite number of neurons (i.e., a multilayer perceptron), can approximate continuous functions

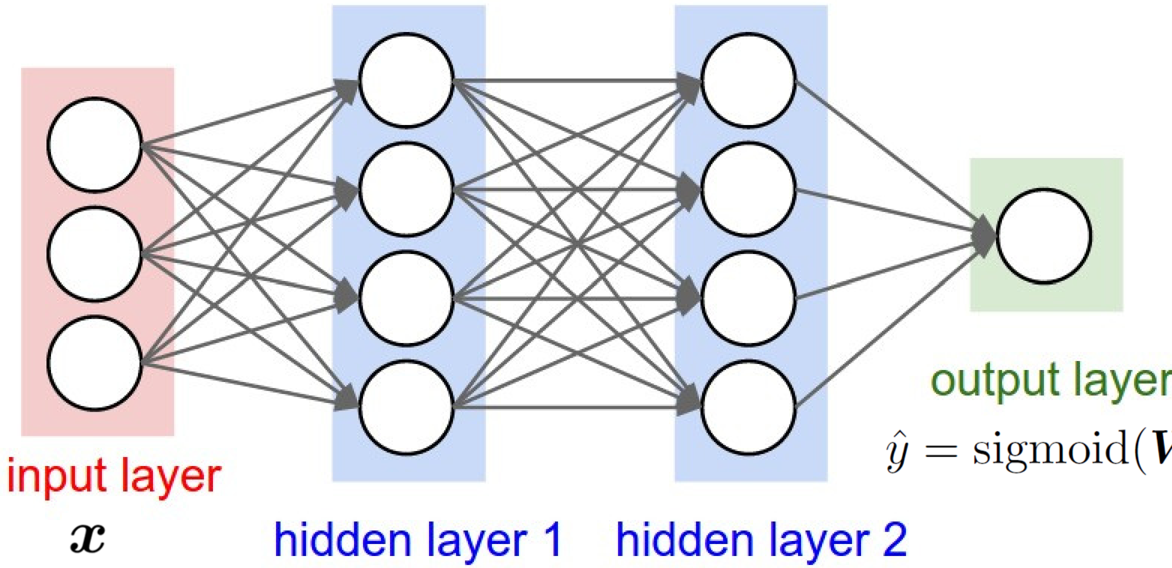
  - on compact subsets of $\mathbb{R}^n$

  - under mild assumptions on the activation function
    - Such as Sigmoid, Tanh and ReLU

[Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." *Neural networks* 2.5 (1989): 359-366.]

Universal Approximation

- Multi-layer perceptron approximate any continuous functions on compact subset of $\mathbb{R}^n$



$$\boldsymbol{x}$$

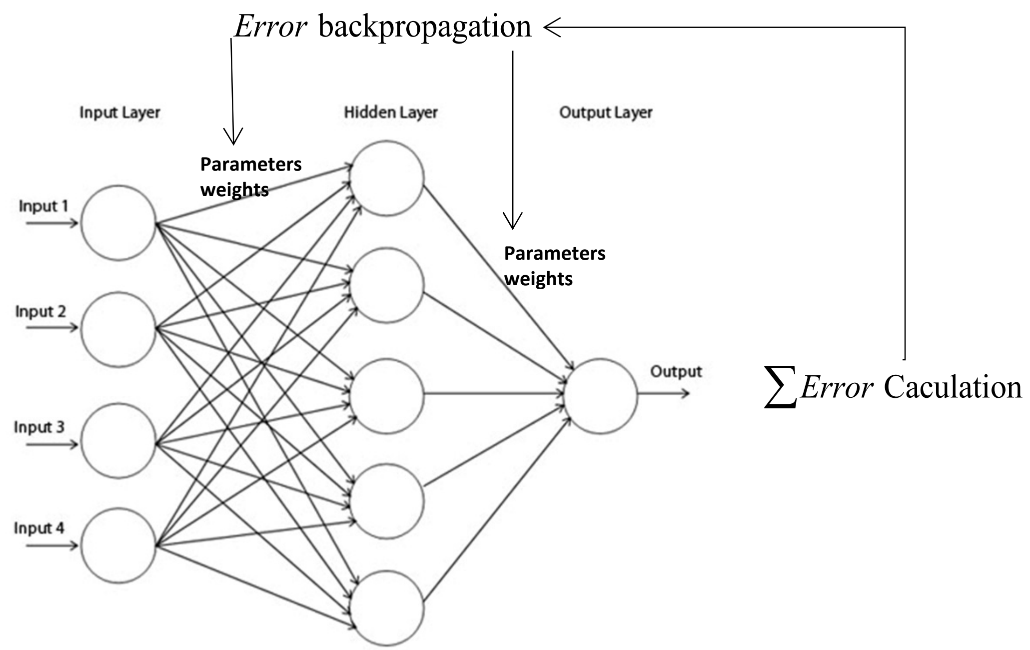
$$\hat{y} = \mathrm{sigmoid}(\boldsymbol{W}_3 \boldsymbol{l}_2 + b_3)$$

$$\boldsymbol{l}_1 = \tanh(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) \quad \boldsymbol{l}_2 = \tanh(\boldsymbol{W}_2 \boldsymbol{l}_1 + \boldsymbol{b}_2)$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad \tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

Hidden Layers and Backpropagation (1986~)

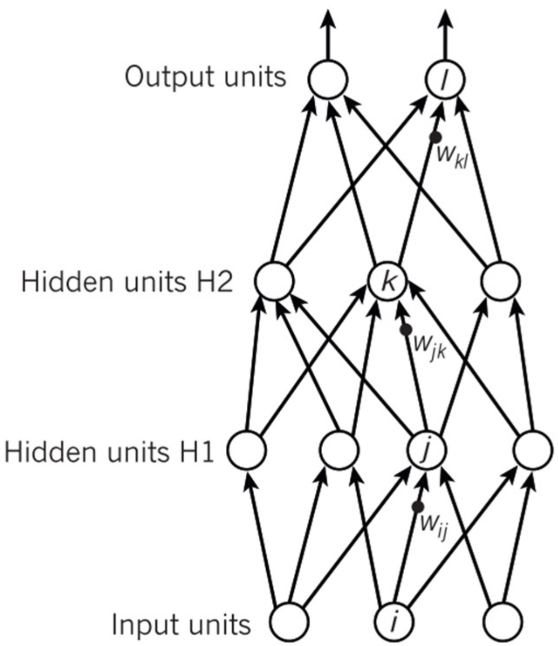
- One of the efficient algorithms for multi-layer neural networks is the *Backpropagation* algorithm
- It was re-introduced in 1986 and Neural Networks regained the popularity



Note: *backpropagation* appears to be found by Werbos [1974]; and then independently rediscovered around 1985 by Rumelhart, Hinton, and Williams [1986] and by Parker [1985]

Learning NN by Back-Propagation

Compare outputs with correct answer to get error



$$y_l = f(z_l)$$

$$z_l = \sum_{k \, \varepsilon \, \text{H2}} w_{kl} \, y_k$$

$$y_k = f(z_k)$$

$$z_k = \sum_{j \, \varepsilon \, \text{H1}} w_{jk} \, y_j$$

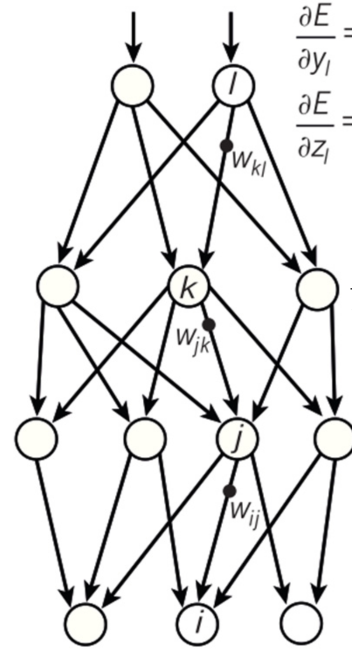
$$y_j = f(z_j)$$

$$z_j = \sum_{i \, \varepsilon \, \text{Input}} w_{ij} \, x_i$$

$$\frac{\partial E}{\partial y_k} = \sum_{l \, \varepsilon \, \text{out}} w_{kl} \frac{\partial E}{\partial z_l}$$

$$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial z_k}$$

Compare outputs with correct answer to get error derivatives



$$\frac{\partial E}{\partial y_l} = y_l - t_l$$

$$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l} \frac{\partial y_l}{\partial z_l}$$

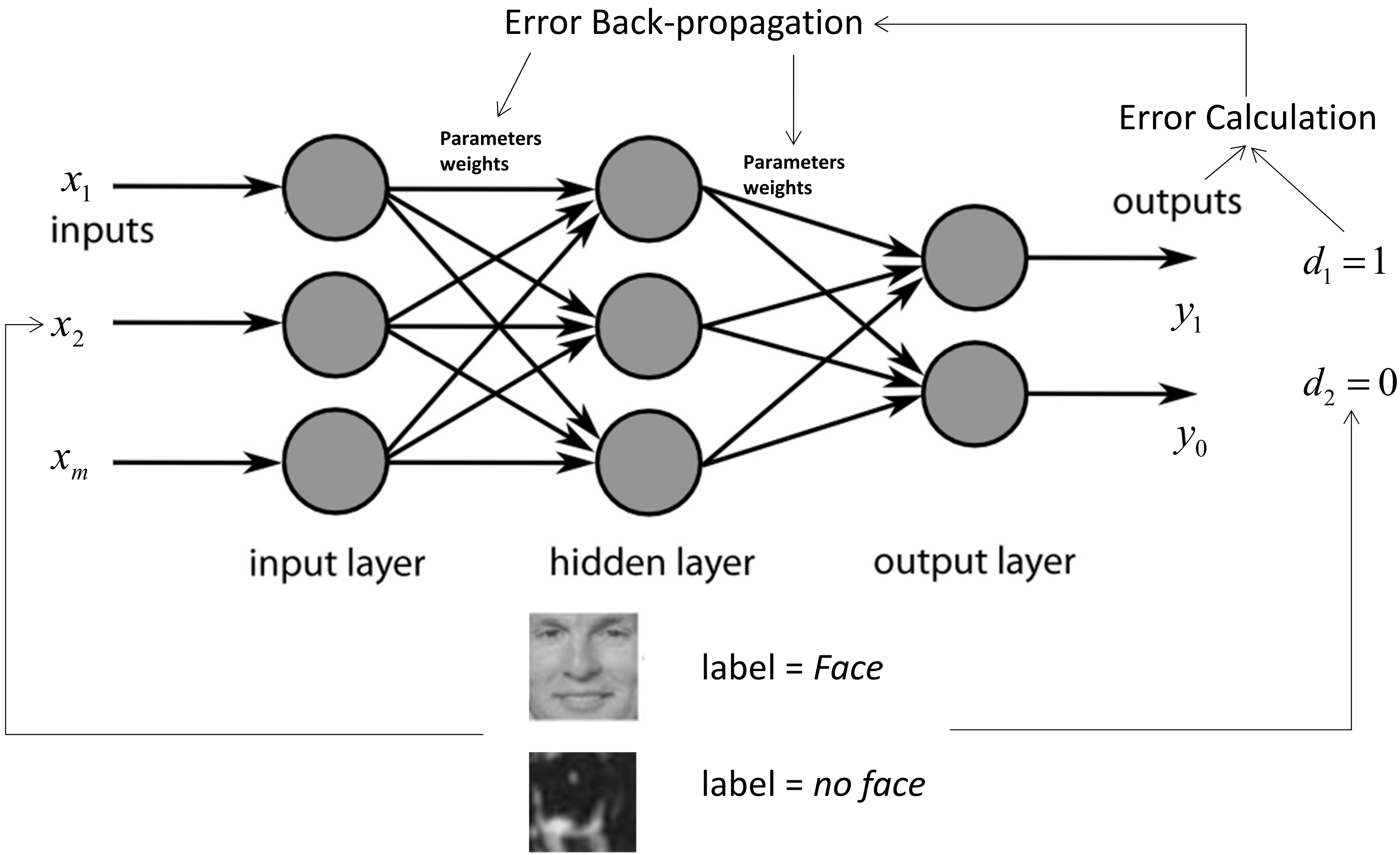
$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial w_{jk}} = \frac{\partial E}{\partial z_k} y_j$$

$$\frac{\partial E}{\partial y_j} = \sum_{k \, \varepsilon \, \text{H2}} w_{jk} \frac{\partial E}{\partial z_k}$$

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_j}$$

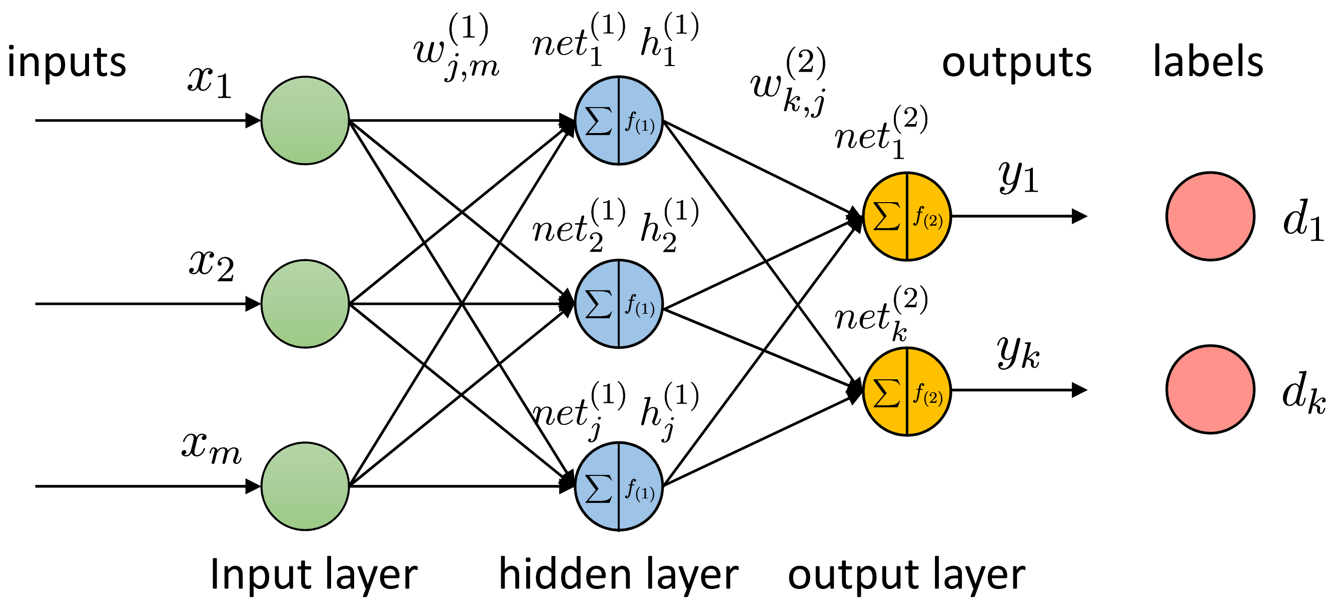
[LeCun, Bengio and Hinton. Deep Learning. Nature 2015.]

Learning NN by Back-Propagation



Training instances…

Make a Prediction



Two-layer feedforward neural network

Feed-forward prediction:

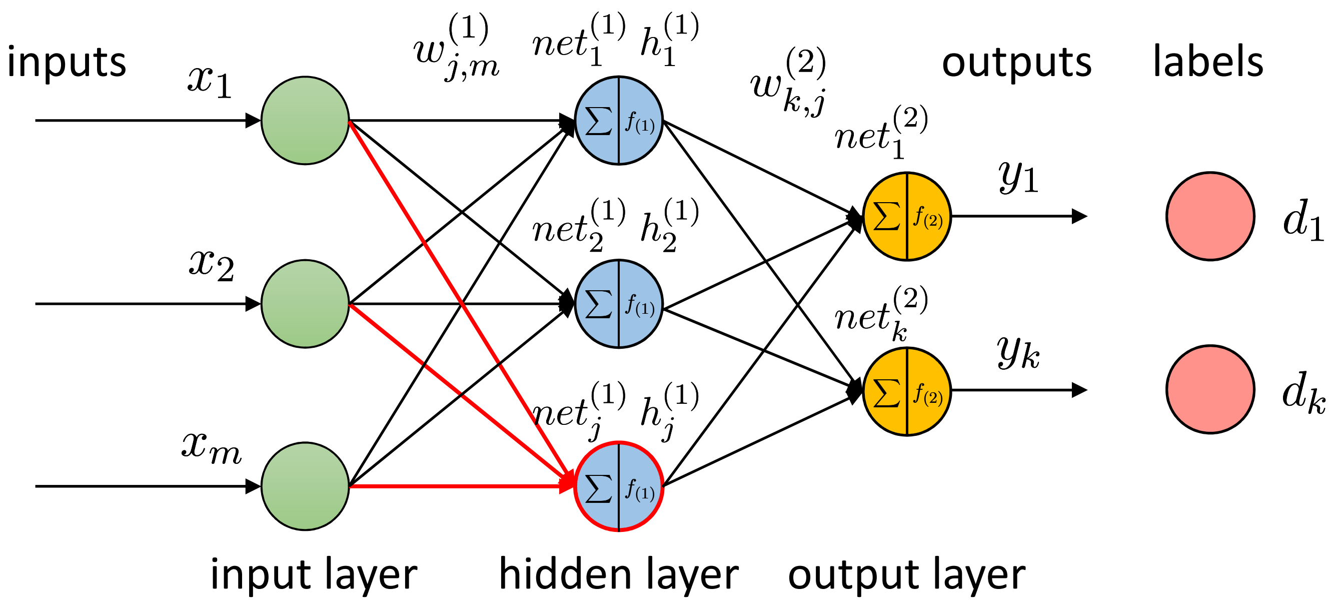
$$h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}(\sum_m w_{j,m}^{(1)} x_m) \quad y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}(\sum_j w_{k,j}^{(1)} h_j^{(1)})$$

$$x = (x_1, \ldots, x_m) \longrightarrow h_j^{(1)} \longrightarrow y_k$$

where

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m \qquad net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$$

Make a Prediction



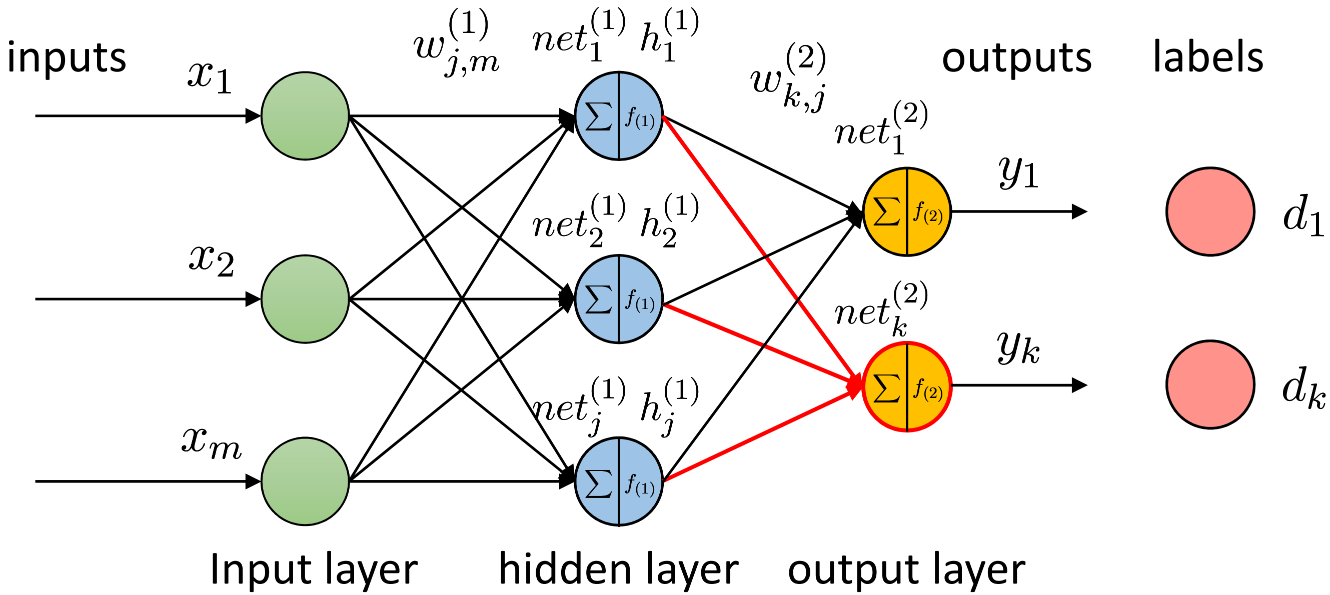
Two-layer feedforward neural network

Feed-forward prediction:

$$x = (x_1, \ldots, x_m) \xrightarrow{h^{(1)}_j = f_{(1)}(net^{(1)}_j) = f_{(1)}(\sum_m w^{(1)}_{j,m} x_m)} h^{(1)}_j \xrightarrow{y_k = f_{(2)}(net^{(2)}_k) = f_{(2)}(\sum_j w^{(1)}_{k,j} h^{(1)}_j)} y_k$$

where $net^{(1)}_j = \sum_m w^{(1)}_{j,m} x_m$ $\qquad$ $net^{(2)}_k = \sum_j w^{(2)}_{k,j} h^{(1)}_j$

Make a Prediction



Two-layer feedforward neural network

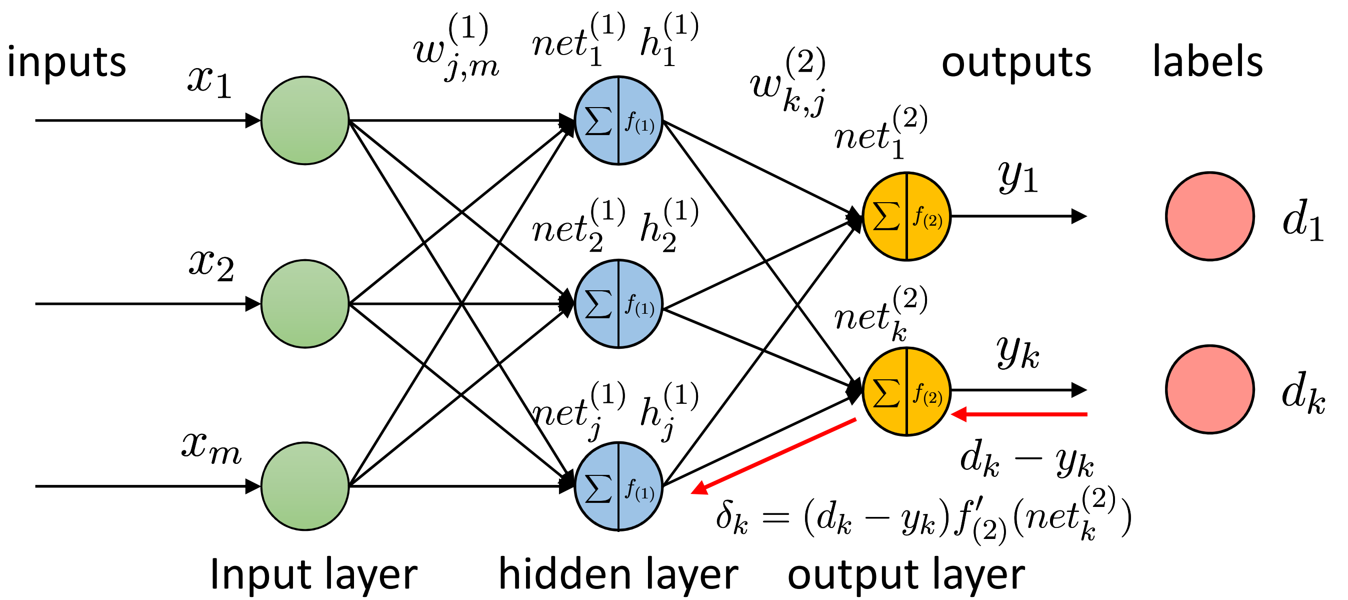
Feed-forward prediction:

$$h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}(\sum_m w_{j,m}^{(1)} x_m) \qquad y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}(\sum_j w_{k,j}^{(1)} h_j^{(1)})$$

$$x = (x_1, \ldots, x_m) \longrightarrow h_j^{(1)} \longrightarrow y_k$$

where $net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$ $\qquad net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$

When Backprop/Learn Parameters



Two-layer feedforward neural network

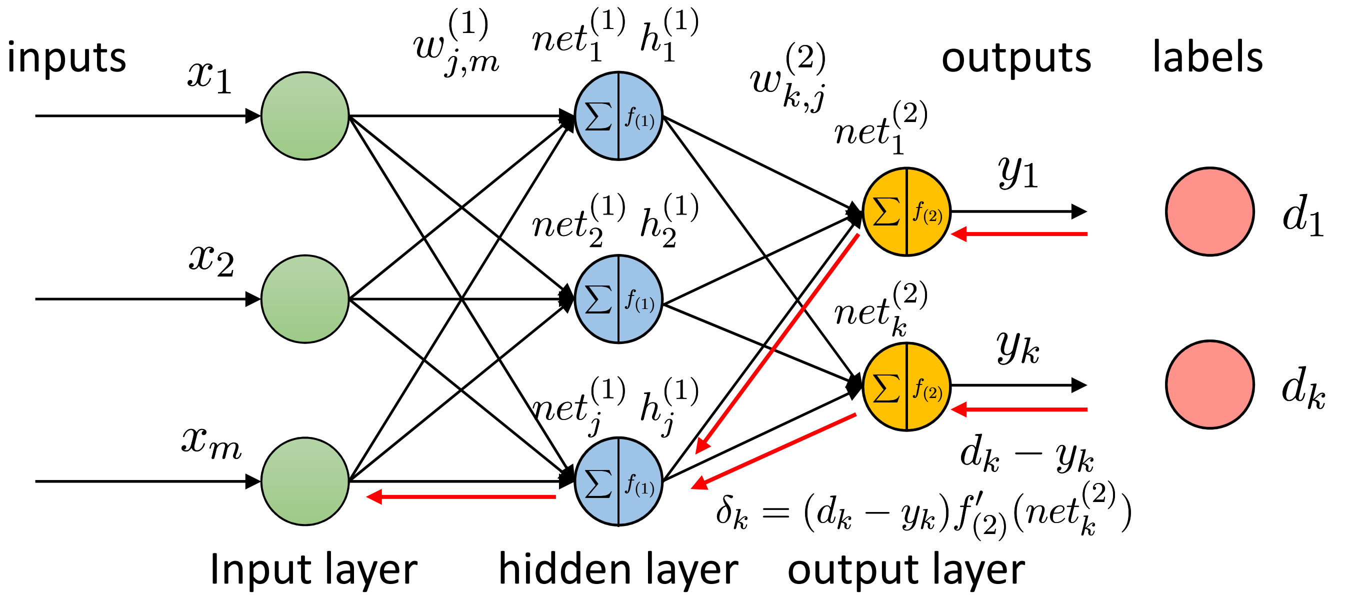
Notations: $net^{(1)}_j = \sum_m w^{(1)}_{j,m} x_m$ $\quad net^{(2)}_k = \sum_j w^{(2)}_{k,j} h_j$

Backprop to learn the parameters

$$w^{(2)}_{k,j} = w^{(2)}_{k,j} + \Delta w^{(2)}_{k,j} \quad \leftarrow \quad \Delta w^{(2)}_{k,j} = \eta Error_k Output_j = \eta \delta_k h^{(1)}_j \quad E(W) = \frac{1}{2}\sum_k (y_k - d_k)^2$$

$$\Delta w^{(2)}_{k,j} = -\eta \frac{\partial E(W)}{\partial w^{(2)}_{k,j}} = -\eta (y_k - d_k) \frac{\partial y_k}{\partial net^{(2)}_k} \frac{\partial net^{(2)}_k}{\partial w^{(2)}_{k,j}} = \eta (d_k - y_k) f'_{(2)}(net^{(2)}_k) h^{(1)}_j = \eta \delta_k h^{(1)}_j$$

When Backprop/Learn Parameters



Two-layer feedforward neural network

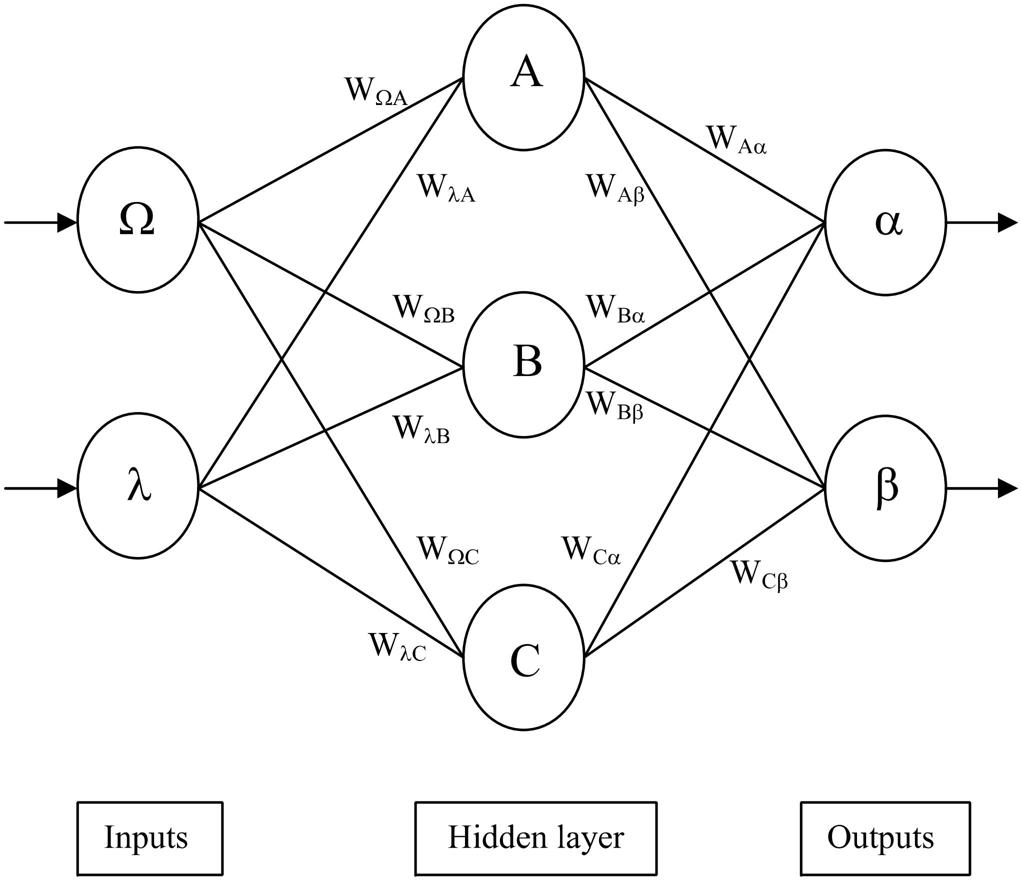
Notations: $net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$ $\qquad net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j$

Backprop to learn the parameters

$$\boxed{w_{j,m}^{(1)} = w_{j,m}^{(1)} + \Delta w_{j,m}^{(1)}} \longleftarrow \Delta w_{k,j}^{(2)} = \eta Error_j Output_m = \eta \delta_j x_m \qquad E(W) = \frac{1}{2}\sum_k (y_k - d_k)^2$$

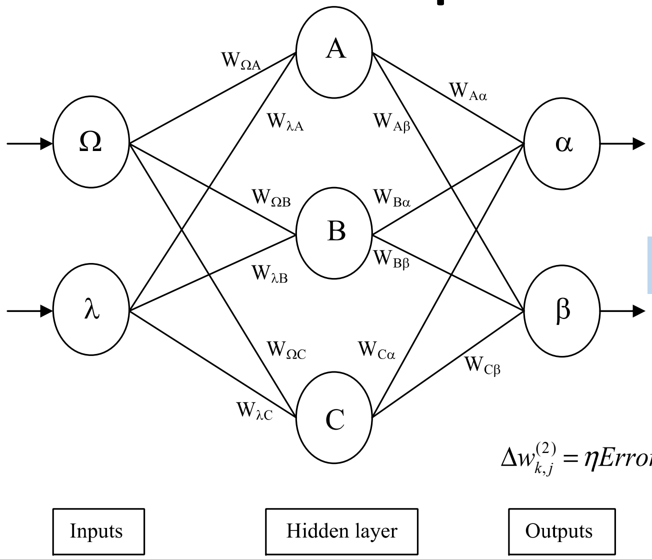
$$\Delta w_{j,m}^{(1)} = -\eta \frac{\partial E(W)}{\partial w_{j,m}^{(1)}} = -\eta \frac{\partial E(W)}{\partial h_j^{(1)}} \frac{\partial h_j^{(1)}}{\partial w_{j,m}^{(1)}} = \eta \sum_k (d_k - y_k) f'_{(2)}(net_k^{(2)}) w_{k,j}^{(2)} x_m f'_{(1)}(net_j^{(1)}) = \eta \delta_j x_m$$

An example for Backprop



https://www4.rgu.ac.uk/files/chapter3%20-%20bp.pdf

An example for Backprop



1. Calculate errors of output neurons

$$\delta_k = (d_k - y_k)\ f_{(2)}'(net_k^{(2)})$$

$$\delta_\alpha = out_\alpha\ (1 - out_\alpha)\ (Target_\alpha - out_\alpha)$$
$$\delta_\beta = out_\beta\ (1 - out_\beta)\ (Target_\beta - out_\beta)$$

2. Change output layer weights

$$\Delta w_{k,j}^{(2)} = \eta Error_k Output_j = \eta \delta_k h_j^{(1)}$$

$$W^+_{A\alpha} = W_{A\alpha} + \eta\delta_\alpha\, out_A \qquad W^+_{A\beta} = W_{A\beta} + \eta\delta_\beta\, out_A$$
$$W^+_{B\alpha} = W_{B\alpha} + \eta\delta_\alpha\, out_B \qquad W^+_{B\beta} = W_{B\beta} + \eta\delta_\beta\, out_B$$
$$W^+_{C\alpha} = W_{C\alpha} + \eta\delta_\alpha\, out_C \qquad W^+_{C\beta} = W_{C\beta} + \eta\delta_\beta\, out_C$$

3. Calculate (back-propagate) hidden layer errors

$$\delta_j = f_{(1)}'(net_j^{(1)}) \sum_k \delta_k w_{k,j}^{(2)}$$

$$\delta_A = out_A\ (1 - out_A)\ (\delta_\alpha W_{A\alpha} + \delta_\beta W_{A\beta})$$
$$\delta_B = out_B\ (1 - out_B)\ (\delta_\alpha W_{B\alpha} + \delta_\beta W_{B\beta})$$
$$\delta_C = out_C\ (1 - out_C)\ (\delta_\alpha W_{C\alpha} + \delta_\beta W_{C\beta})$$

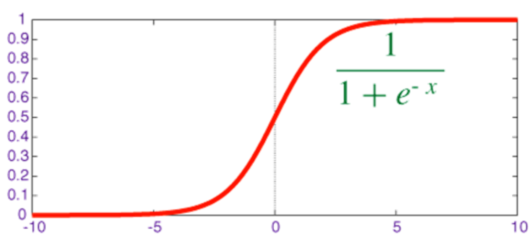
4. Change hidden layer weights

$$\Delta w_{j,m}^{(1)} = \eta Error_j Output_m = \eta \delta_j x_m$$

$$W^+_{\lambda A} = W_{\lambda A} + \eta\delta_A\, in_\lambda \qquad W^+_{\Omega A} = W^+_{\Omega A} + \eta\delta_A\, in_\Omega$$
$$W^+_{\lambda B} = W_{\lambda B} + \eta\delta_B\, in_\lambda \qquad W^+_{\Omega B} = W^+_{\Omega B} + \eta\delta_B\, in_\Omega$$
$$W^+_{\lambda C} = W_{\lambda C} + \eta\delta_C\, in_\lambda \qquad W^+_{\Omega C} = W^+_{\Omega C} + \eta\delta_C\, in_\Omega$$

Consider sigmoid activation function

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

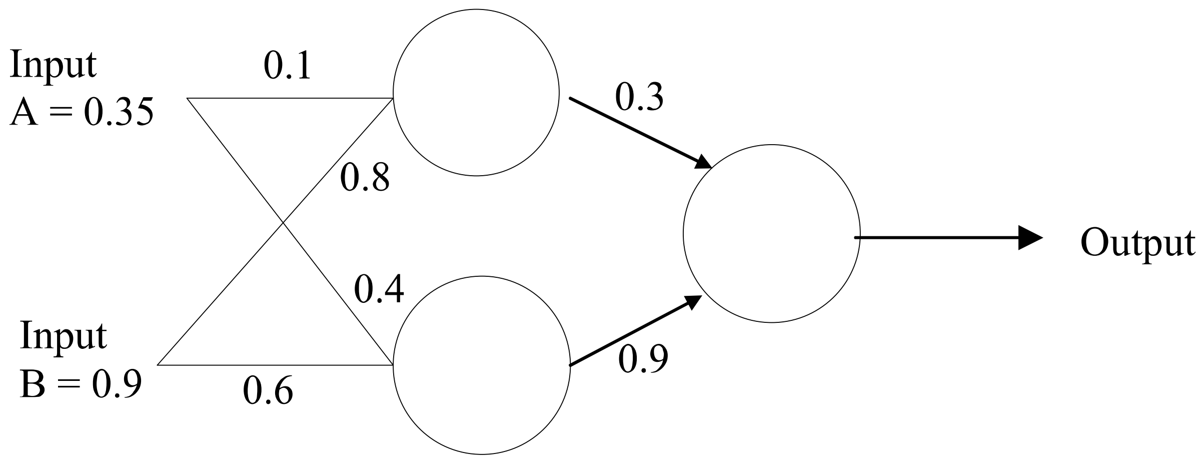


$$f'_{Sigmoid}(x) = f_{Sigmoid}(x)(1 - f_{Sigmoid}(x))$$

https://www4.rgu.ac.uk/files/chapter3%20-%20bp.pdf

Let us do some calculation

Consider the simple network below:



Assume that the neurons have a Sigmoid activation function and

1. Perform a forward pass on the network
2. Perform a reverse pass (training) once (target = 0.5)
3. Perform a further forward pass and comment on the result

https://www4.rgu.ac.uk/files/chapter3%20-%20bp.pdf

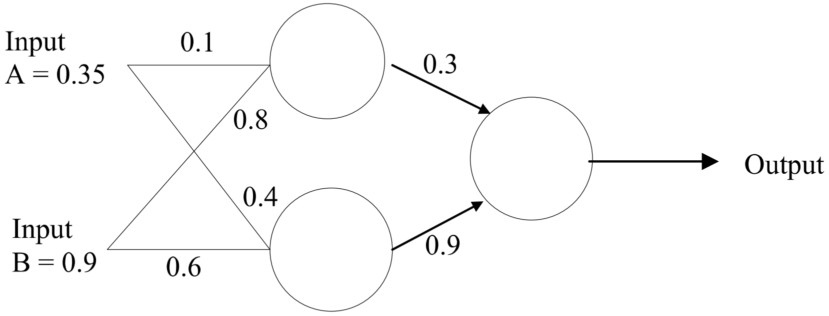
Let us do some calculation

Answer:
(i)
Input to top neuron = (0.35x0.1)+(0.9x0.8)=0.755. Out = 0.68.
Input to bottom neuron = (0.9x0.6)+(0.35x0.4) = 0.68. Out = 0.6637.
Input to final neuron = (0.3x0.68)+(0.9x0.6637) = 0.80133. Out = 0.69.

(ii)
Output error δ=(t-o)(1-o)o = (0.5-0.69)(1-0.69)0.69 = -0.0406.

New weights for output layer
$w1^+$ = w1+(δ x input) = 0.3 + (-0.0406x0.68) = 0.272392.
$w2^+$ = w2+(δ x input) = 0.9 + (-0.0406x0.6637) = 0.87305.

Errors for hidden layers:
$\delta 1 = \delta$ x w1 = -0.0406 x 0.272392 x (1-o)o = -2.406×10^{-3}
$\delta 2 = \delta$ x w2 = -0.0406 x 0.87305 x (1-o)o = -7.916×10^{-3}

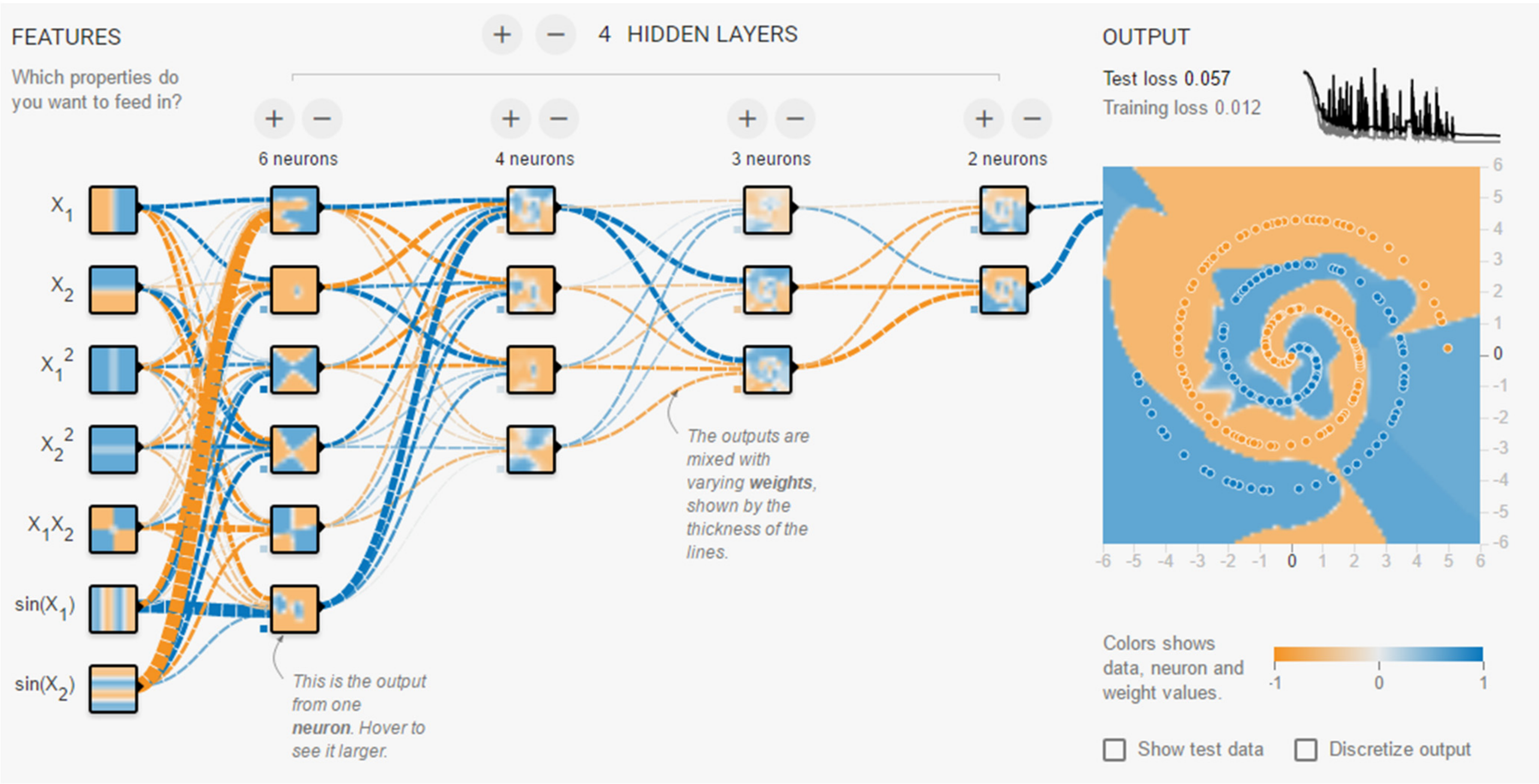


New hidden layer weights:
$w3^+$=0.1 + (-2.406 x 10^{-3} x 0.35) = 0.09916.
$w4^+$ = 0.8 + (-2.406 x 10^{-3} x 0.9) = 0.7978.
$w5^+$ = 0.4 + (-7.916 x 10^{-3} x 0.35) = 0.3972.
$w6^+$ = 0.6 + (-7.916 x 10^{-3} x 0.9) = 0.5928.

(iii)
Old error was -0.19. New error is -0.18205. Therefore error has reduced.

https://www4.rgu.ac.uk/files/chapter3%20-%20bp.pdf

A demo from Google

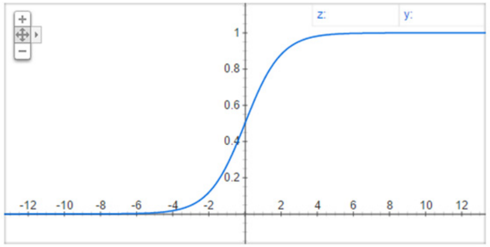


http://playground.tensorflow.org/

Non-linear Activation Functions

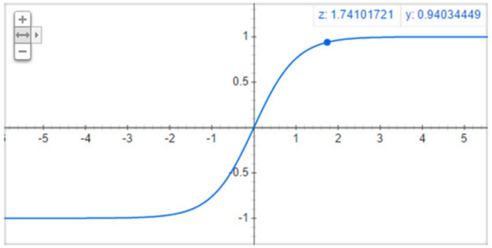
- Sigmoid

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



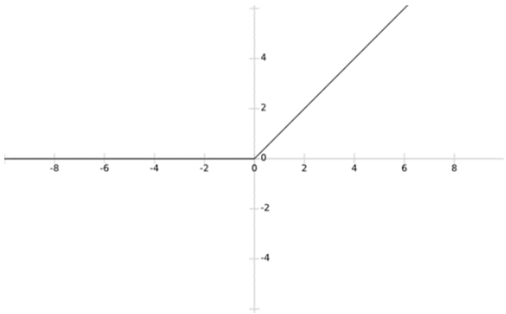
- Tanh

$$\tanh(z) = \frac{1 - e^{-2z}}{1 + e^{-2z}}$$

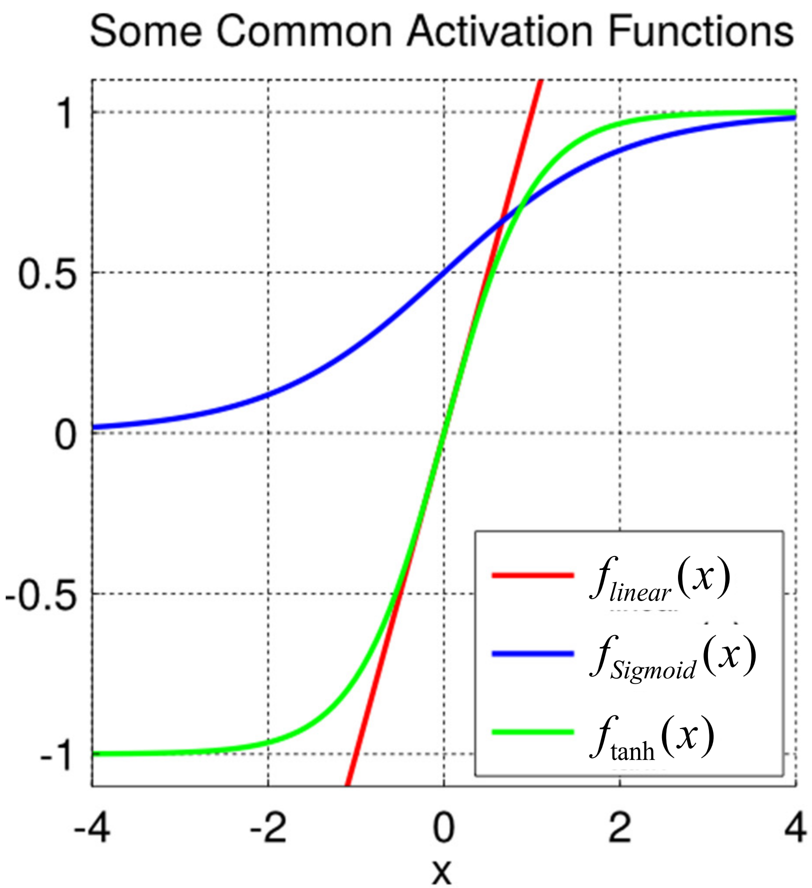


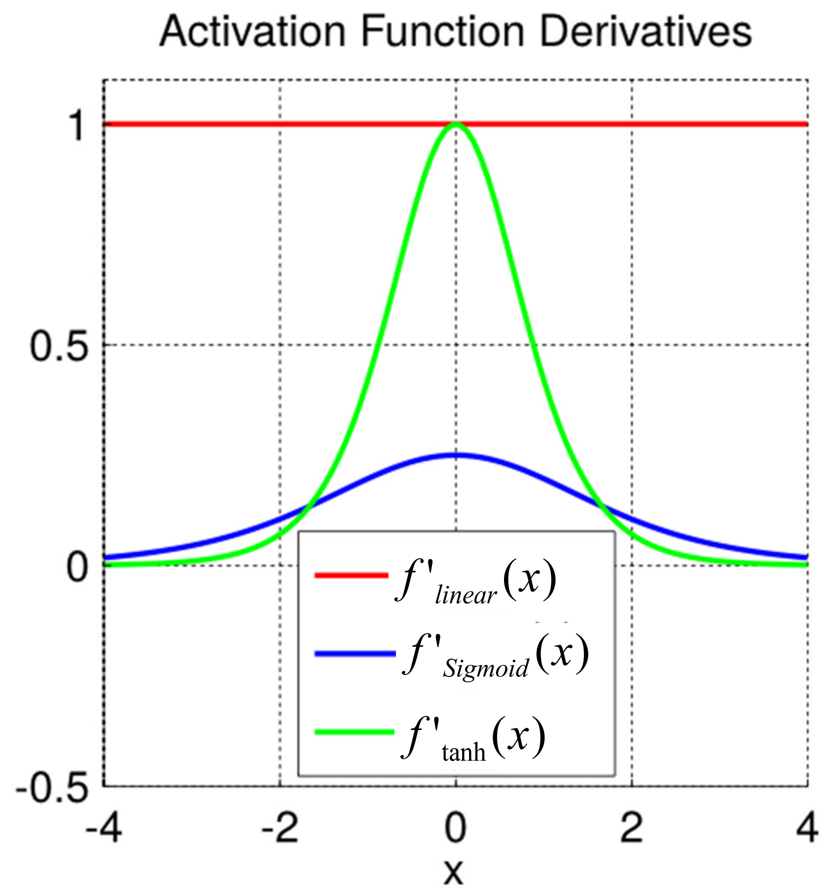
- Rectified Linear Unit (ReLU)

$$\text{ReLU}(z) = \max(0, z)$$



Active functions





https://theclevermachine.wordpress.com/tag/tanh-function/

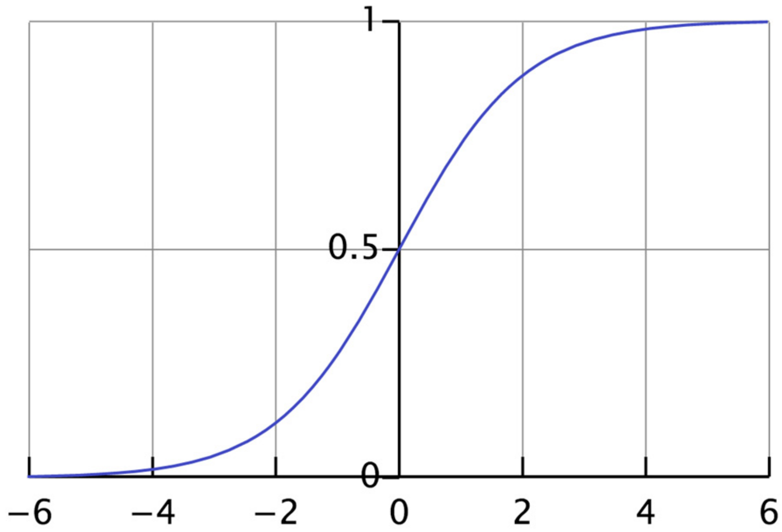
Activation functions

- **Logistic Sigmoid:**

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

Its derivative:

$$f'_{Sigmoid}(x) = f_{Sigmoid}(x)(1 - f_{Sigmoid}(x))$$

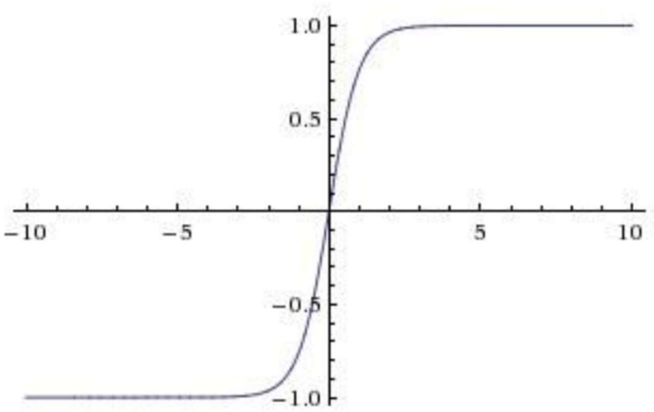


- Output range [0,1]
- Motivated by biological neurons and can be interpreted as the probability of an artificial neuron "firing" given its inputs
- However, saturated neurons make gradients vanished **(why?)**

Activation functions

- Tanh function

$$f_{\tanh}(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



Its gradient:

$$f_{\tanh}(x) = 1 - f_{\tanh}(x)^2$$

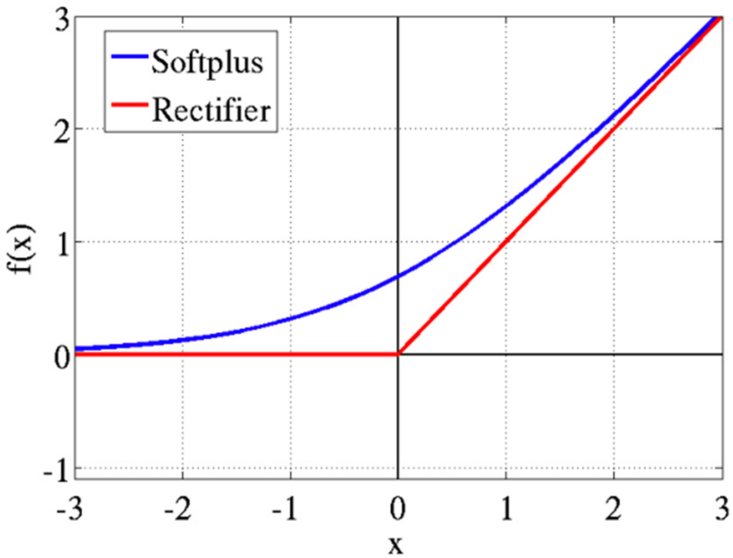
- Output range [-1,1]
- Thus strongly negative inputs to the tanh will map to negative outputs.
- Only zero-valued inputs are mapped to near-zero outputs
- These properties make the network less likely to get “stuck” during training

https://theclevermachine.wordpress.com/tag/tanh-function/

Active Functions

- ReLU (rectified linear unit)

$$f_{\text{ReLU}}(x) = \max(0, x)$$



http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/40811.pdf

- The derivative:

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

- Another version is Noise ReLU:

$$f_{\text{NoisyReLU}}(x) = \max(0, x + N(0, \delta(x)))$$

- ReLU can be approximated by softplus function

$$f_{\text{Softplus}}(x) = \log(1 + e^x)$$

- ReLU gradient doesn't vanish as we increase x
- It can be used to model positive number
- It is fast as no need for computing the exponential function
- It eliminates the necessity to have a "*pretraining*" phase

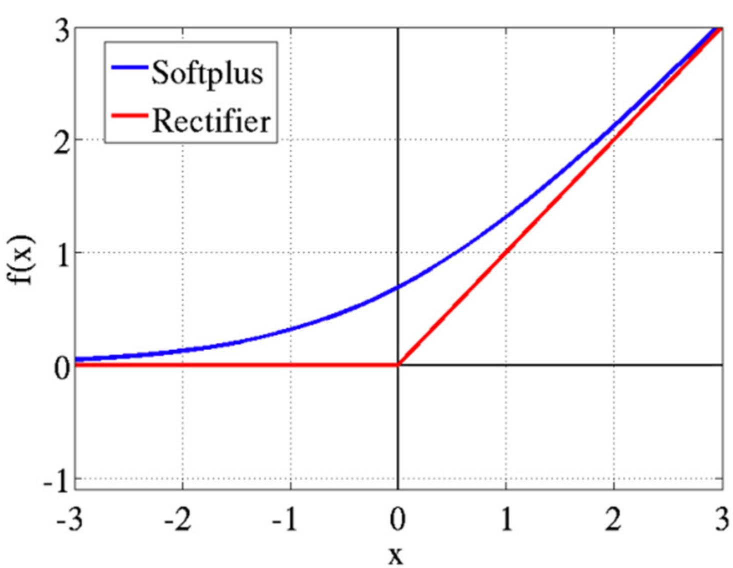
Active Functions

- ReLU (rectified linear unit)

$$f_{\mathrm{ReLU}}(x) = \max(0, x)$$

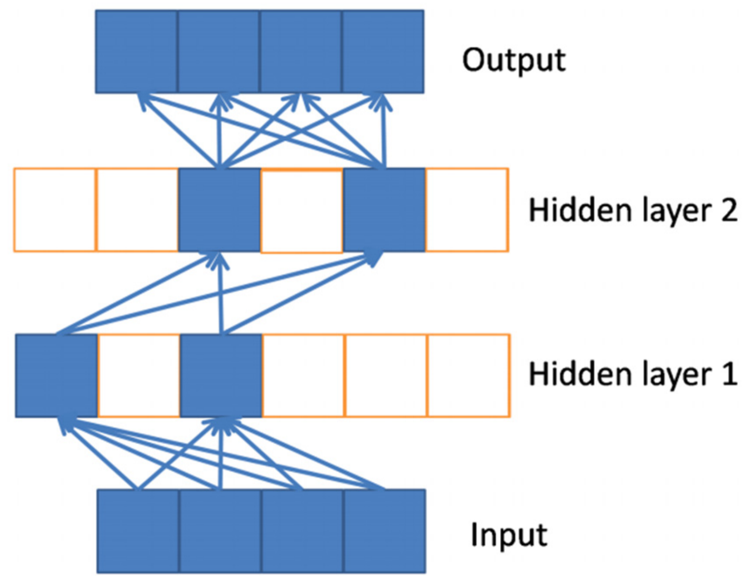
ReLU can be approximated by softplus function

$$f_{\mathrm{Softplus}}(x) = \log(1 + e^x)$$



Additional active functions:
Leaky ReLU, Exponential LU, Maxout etc

- The only non-linearity comes from the path selection with individual neurons being active or not
- It allows sparse representations:
  - for a given input only a subset of neurons are active



Sparse propagation of activations and gradients

http://www.jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf

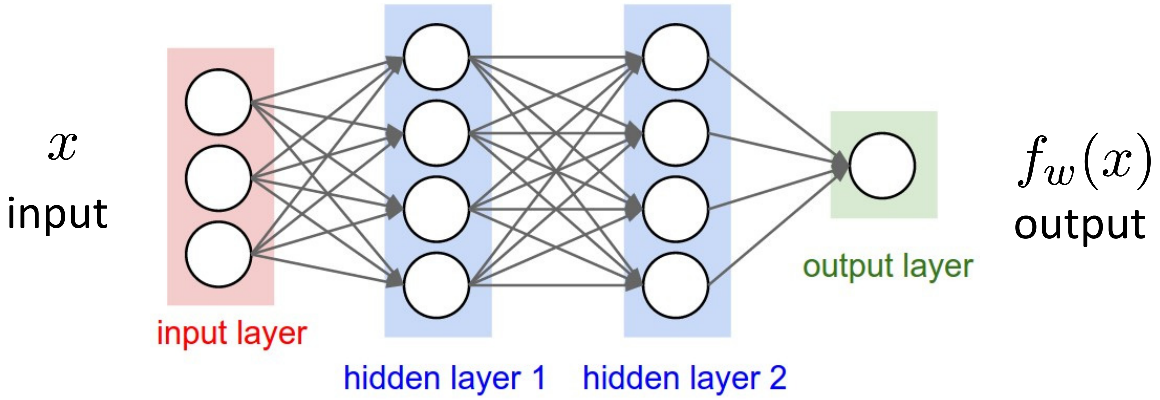
Error/Loss function

- Recall stochastic gradient descent
  - Update from a randomly picked example (but in practice do a batch update)

$$w = w - \eta \frac{\partial \mathcal{L}(w)}{\partial w}$$

- Squared error loss for one binary output:

$$\mathcal{L}(w) = \frac{1}{2}(y - f_w(x))^2$$



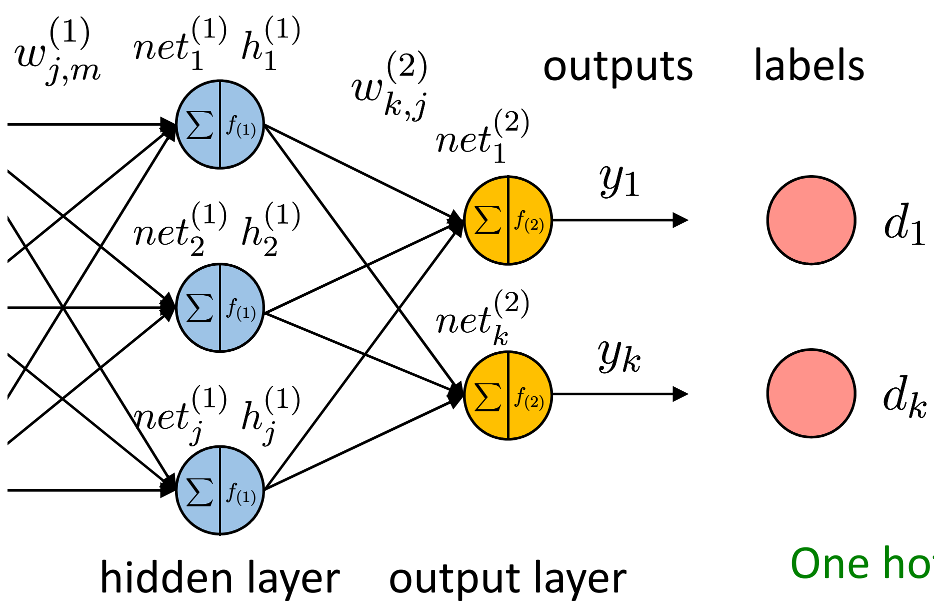
Error/Loss function

- Softmax (cross-entropy loss) for multiple classes

(Class labels follow multinomial distribution)

$$\mathcal{L}(w) = -\sum_k \left(d_k \log \hat{y}_k + (1 - d_k) \log(1 - y_k)\right)$$

where $$\hat{y}_k = \frac{\exp\left(\sum_j w_{k,j}^{(2)} h_j^{(1)}\right)}{\sum_{k'} \exp\left(\sum_j w_{k',j}^{(2)} h_j^{(1)}\right)}$$



One hot encoded class labels

Advanced Topic of this Lecture

Deep Learning

As a prologue of the DL Course in the next semester

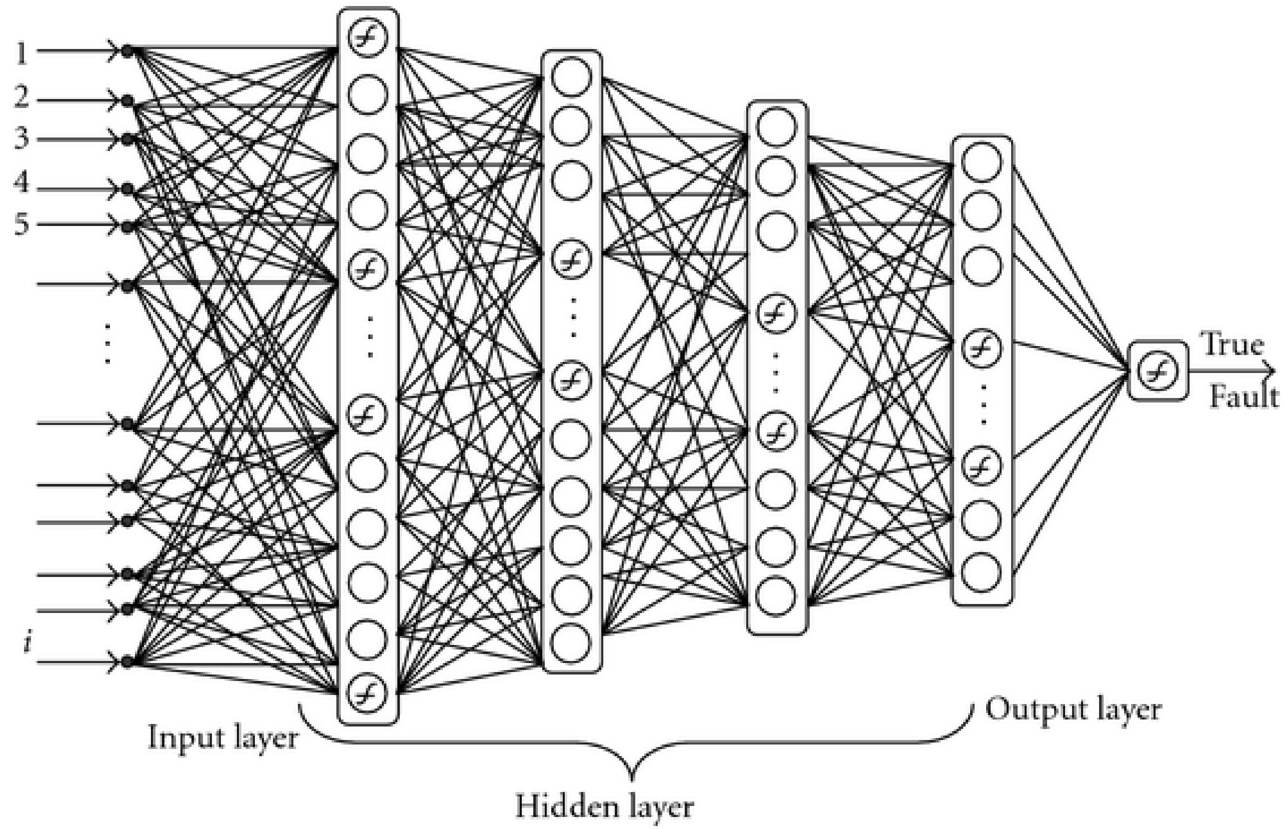
What is Deep Learning

- Deep learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level.

- Mostly implemented via neural networks

[LeCun, Bengio and Hinton. Deep Learning. Nature 2015.]

Deep Neural Network (DNN)



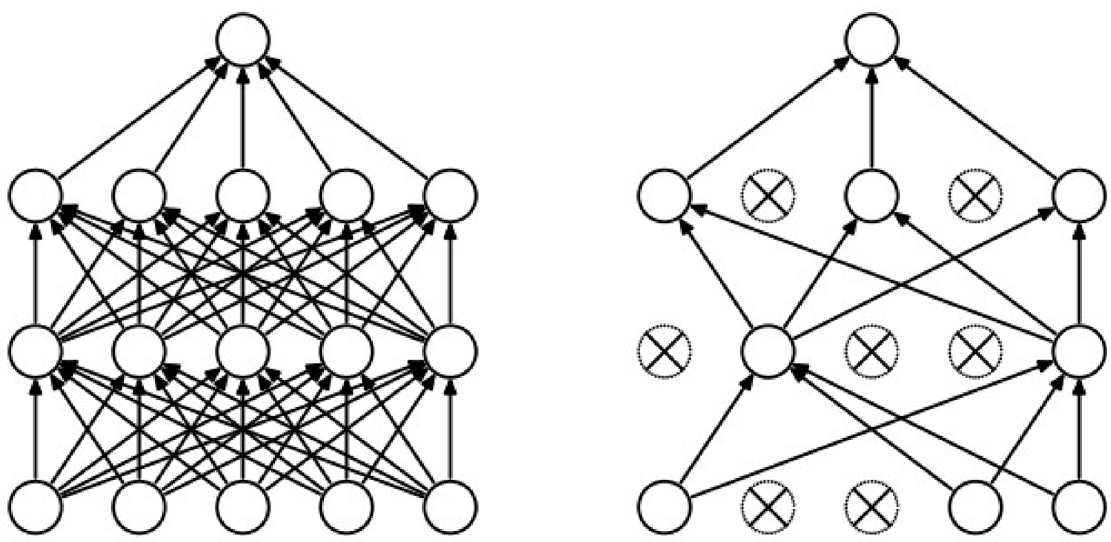
- Multi-layer perceptron with many hidden layers

Difficulty of Training Deep Nets

- Lack of big data
  - Now we have a lot of big data
- Lack of computational resources
  - Now we have GPUs and HPCs
- Easy to get into a (bad) local minimum
  - Now we use pre-training techniques & various optimization algorithms
- Gradient vanishing
  - Now we use ReLU
- Regularization
  - Now we use Dropout

Dropout

- Dropout randomly 'drops' units from a layer on each training step, creating 'sub-architectures' within the model.
- It can be viewed as a type of sampling of a smaller network within a larger network
- Prevent neural networks from overfitting

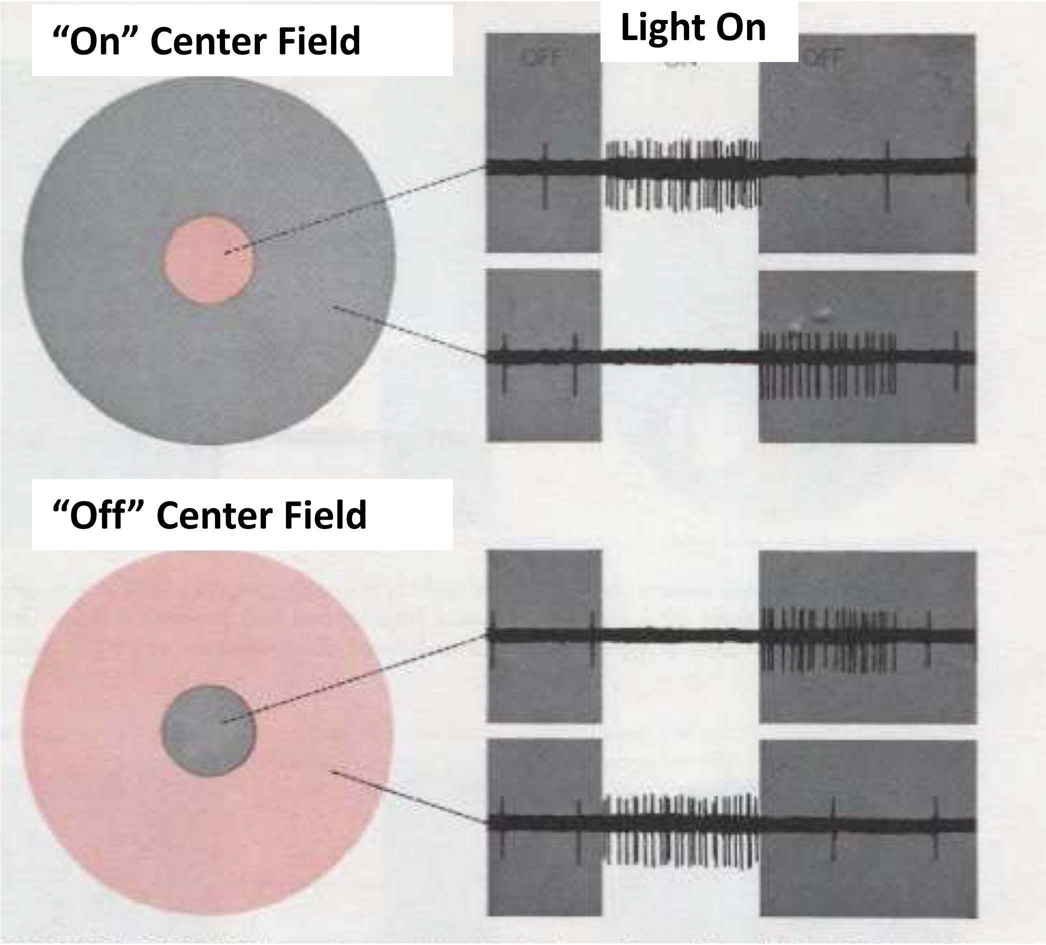


(a) Standard Neural Net (b) After applying dropout.

Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

Convolutional neural networks: Receptive field

- Receptive field: Neurons in the retina respond to light stimulus in restricted regions of the visual field
- Animal experiments on receptive fields of two retinal ganglion cells
  - Fields are circular areas of the retina
  - The cell (upper part) responds when the center is illuminated and the surround is darkened.
  - The cell (lower part) responds when the center is darkened and the surround is illuminated.
  - Both cells give on- and off- responses when both center and surround are illuminated, but neither response is as strong as when only center or surround is illuminated



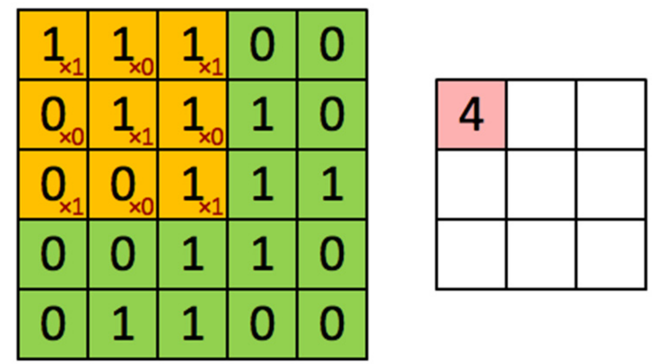
Contributed by Hubel and Wiesel for the studies of the visual system of a cat

Hubel D.H. : The Visual Cortex of the Brain Sci Amer 209:54-62, 1963

Convolutional neural networks

- Sparse connectivity by local correlation
  - Filter: the input of a hidden unit in layer *m* are from a subset of units in layer *m-1* that have spatially connected receptive fields
- Shared weights
  - each filter is replicated across the entire visual field. These replicated units share the same weights and form a feature map.

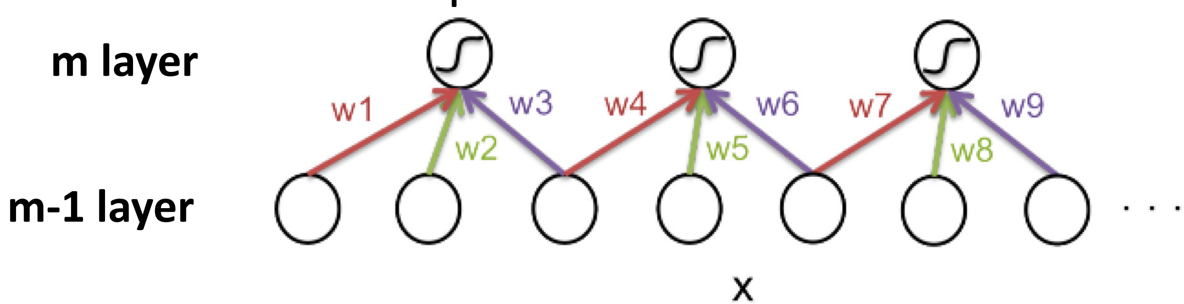
2-d case (subscripts are weights)



***m*-1 layer**

***one filter at m* layer**

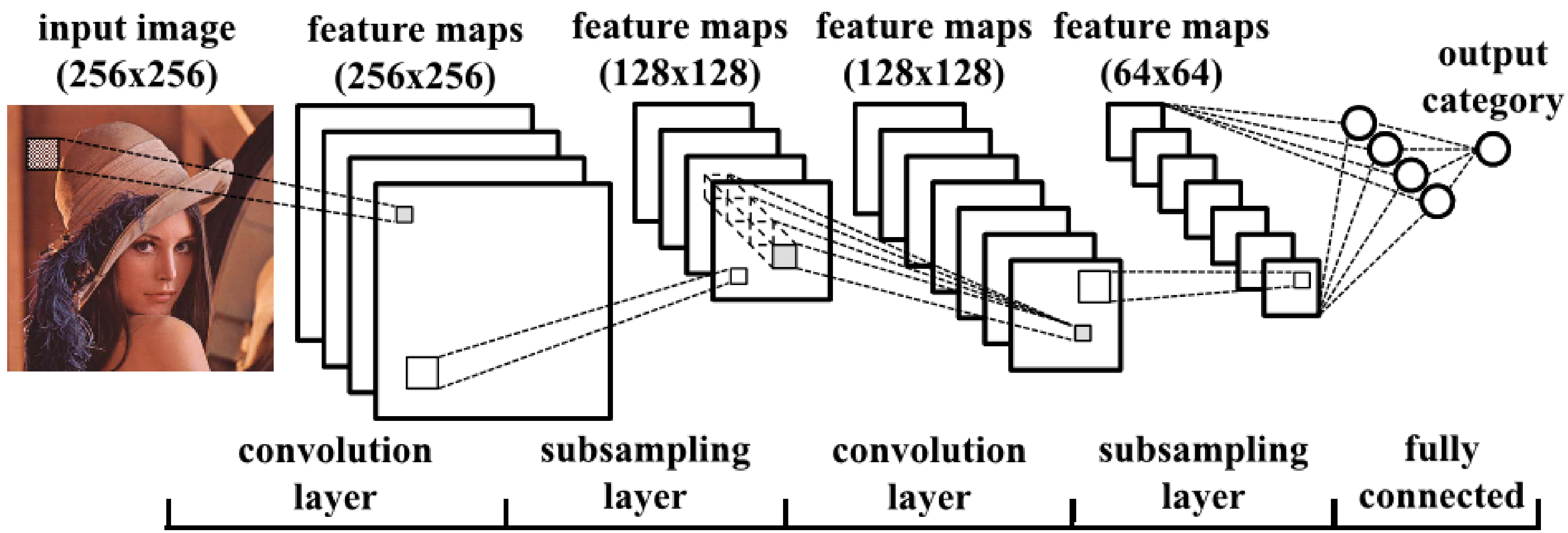
1-d case



edges that have the same color have the same weight

http://deeplearning.net/tutorial/lenet.html

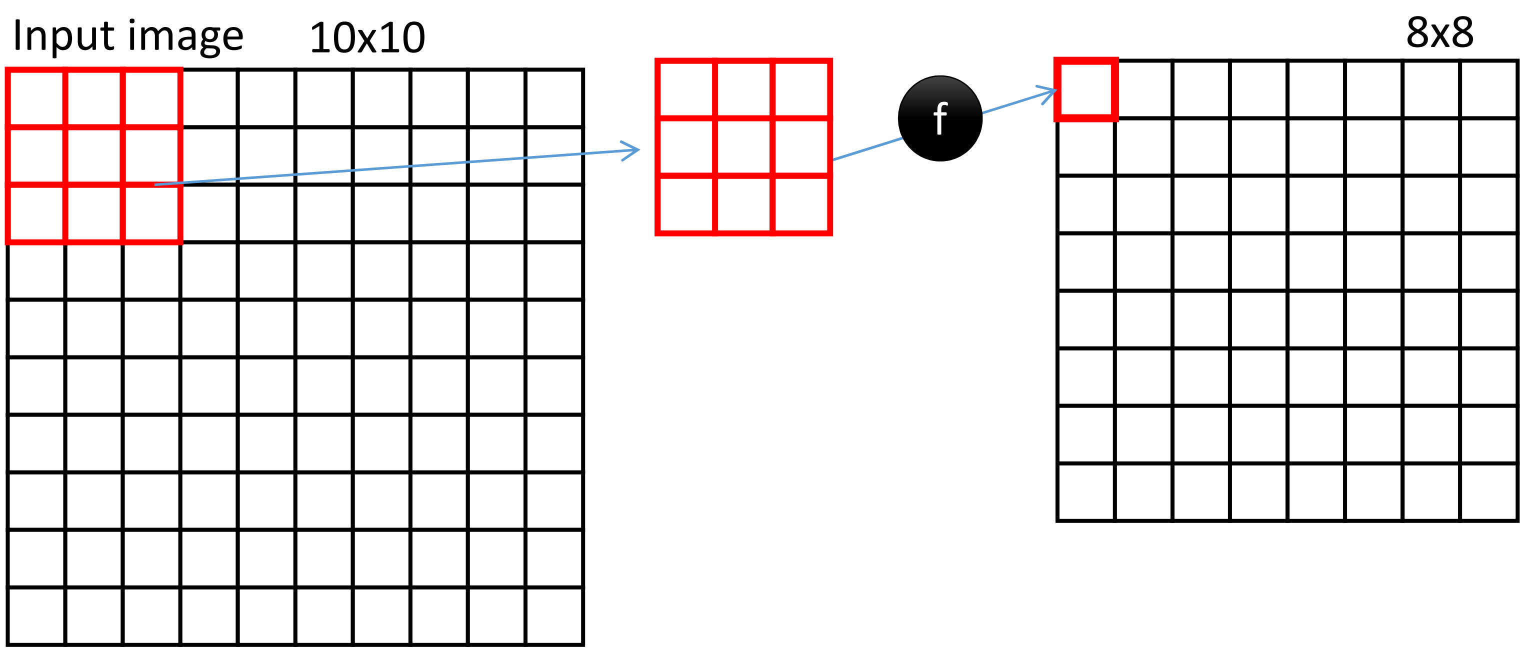
Convolutional Neural Network (CNN)

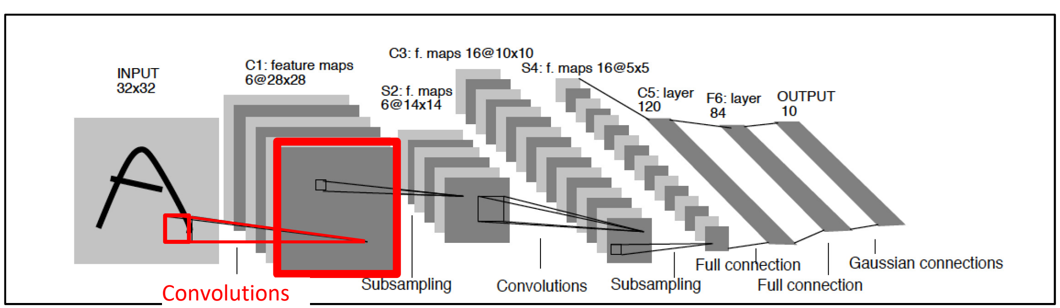


[Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11) 1998]

Convolution Layer

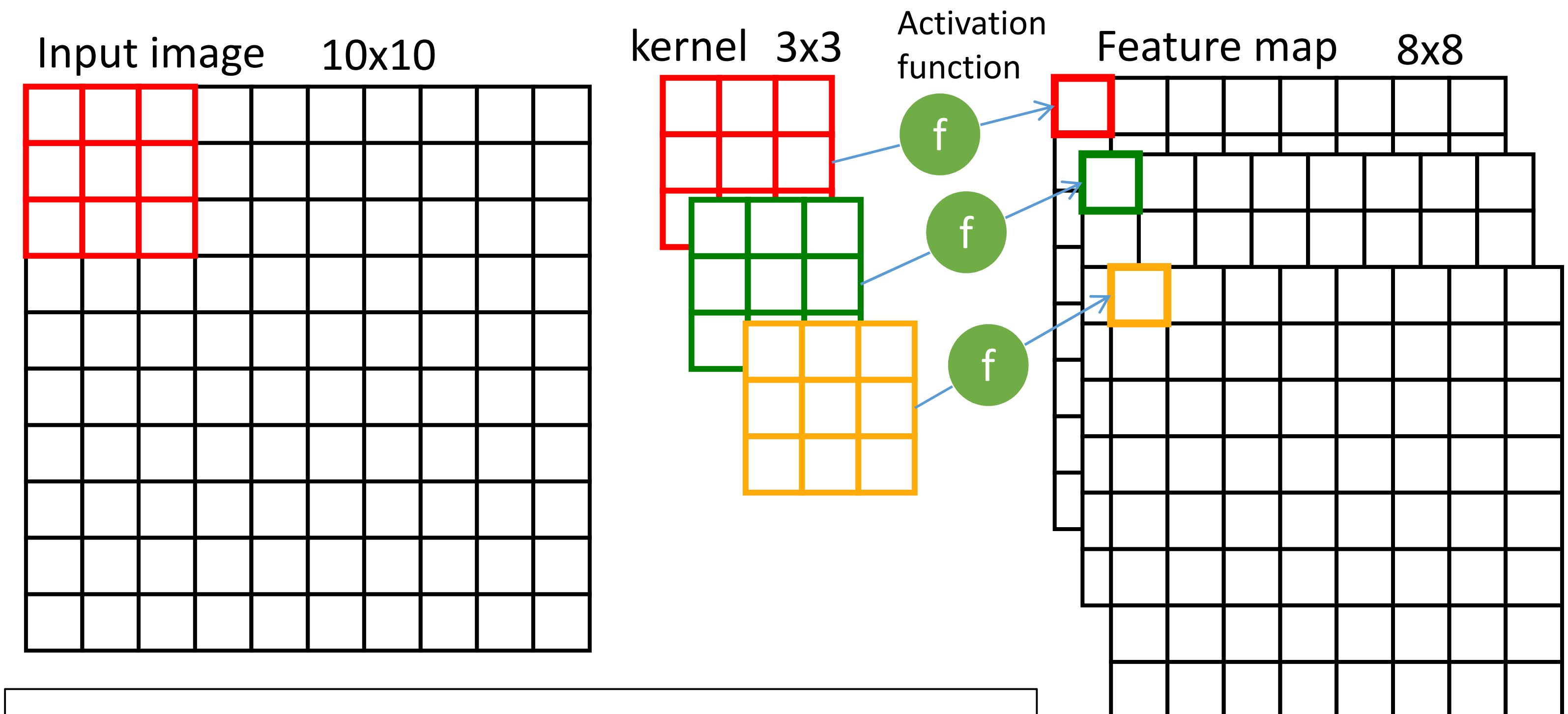
Example: a 10x10 input image with a 3x3 filter result in an 8x8 output image

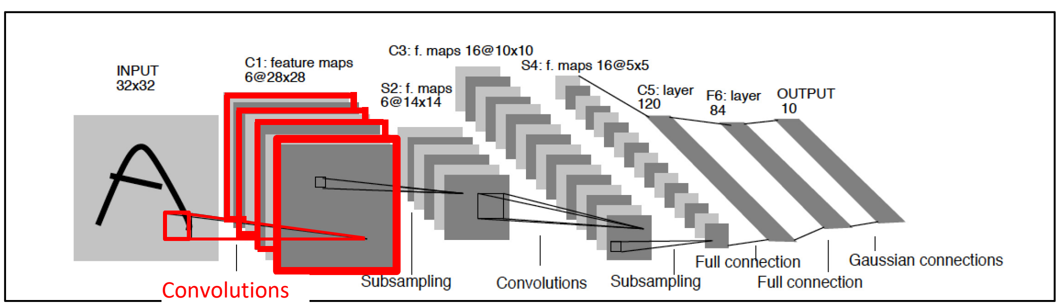




Convolution Layer

- Example: a 10x10 input image with a 3x3 filter result in an 8x8 output image
- 3 different filters (weights are different) lead to 3 8x8 out images



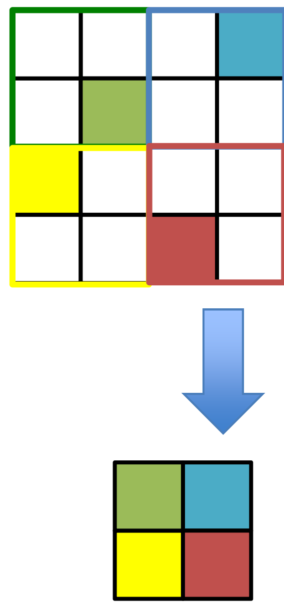


Pooling Subsampling Layer

- Pooling: partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum or average value.

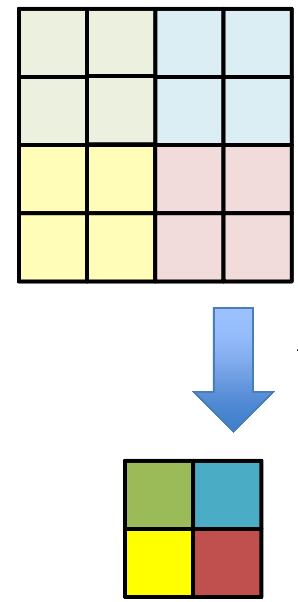
Max pooling
- reduces computation and
- is a way of taking the most responsive node of the given interest region,
- but may result in loss of accurate spatial information

Max pooling

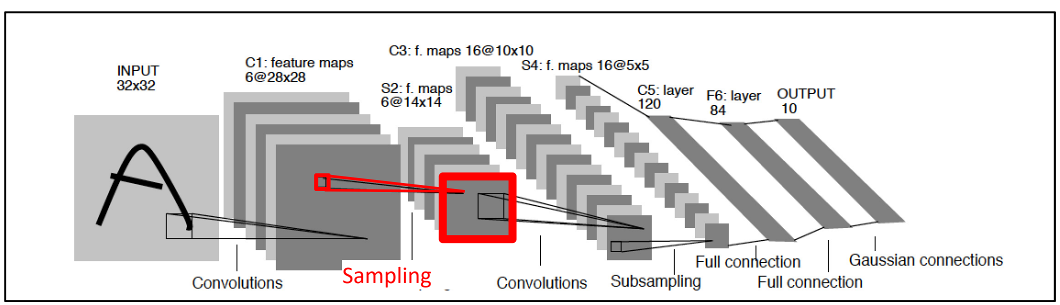


Max in a 2x2 filter

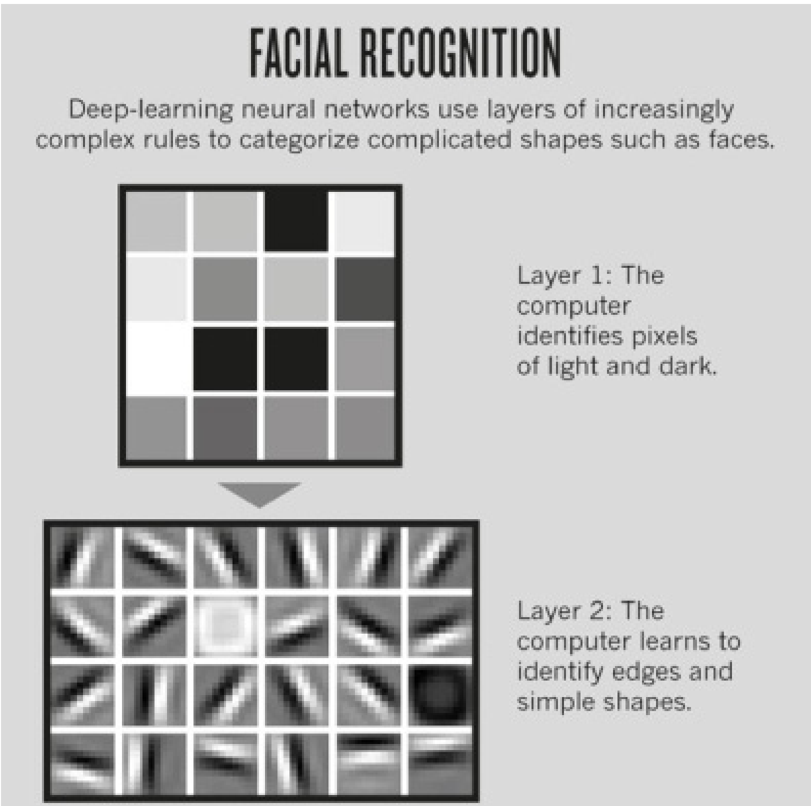
Average pooling

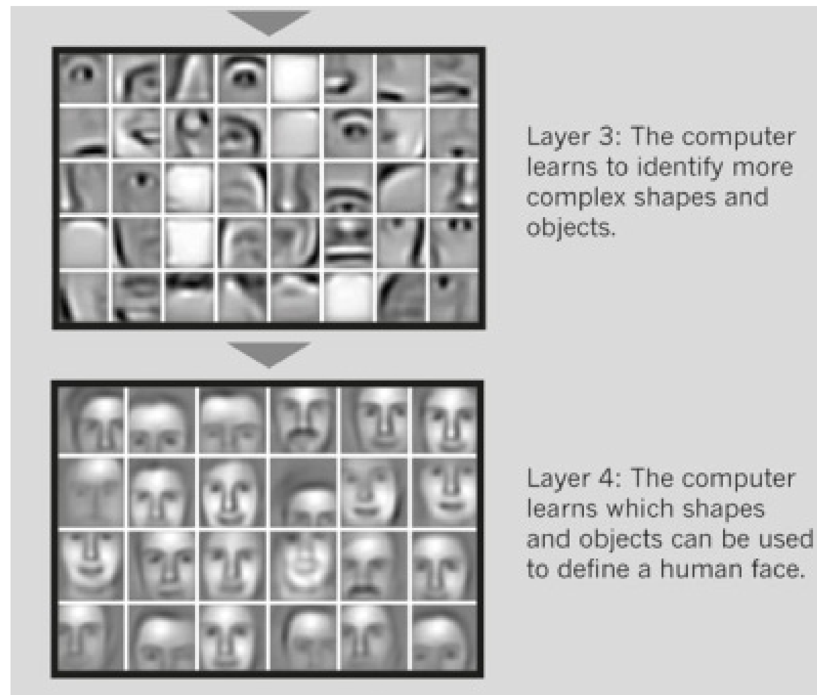


Average in a 2x2 filter



Use Case: Face Recognition



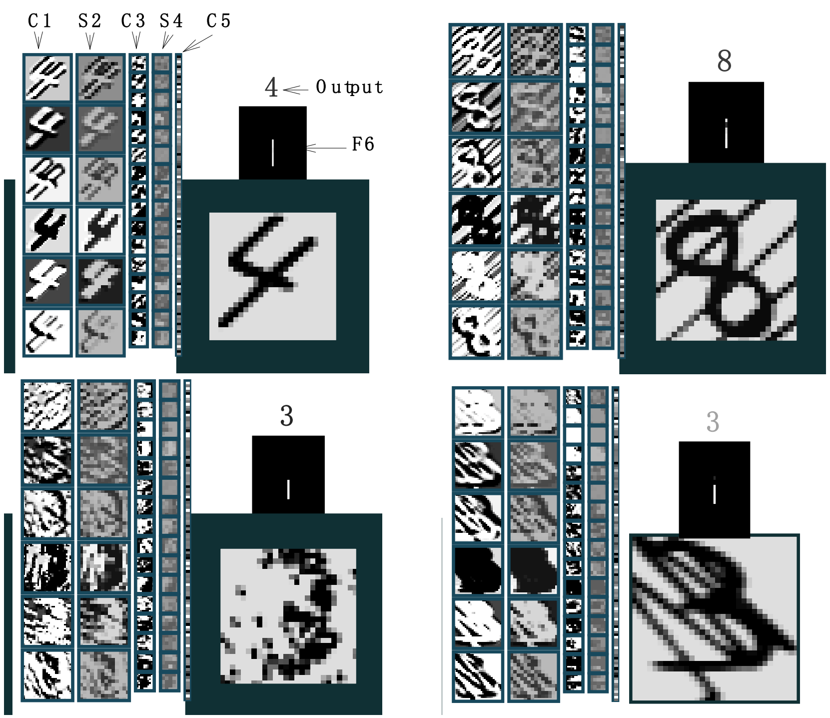


Use Case: Digits Recognition

- MNIST (handwritten digits) Dataset:

  http://yann.lecun.com/exdb/mnist/

  - 60k training and 10k test examples
- Test error rate 0.95%

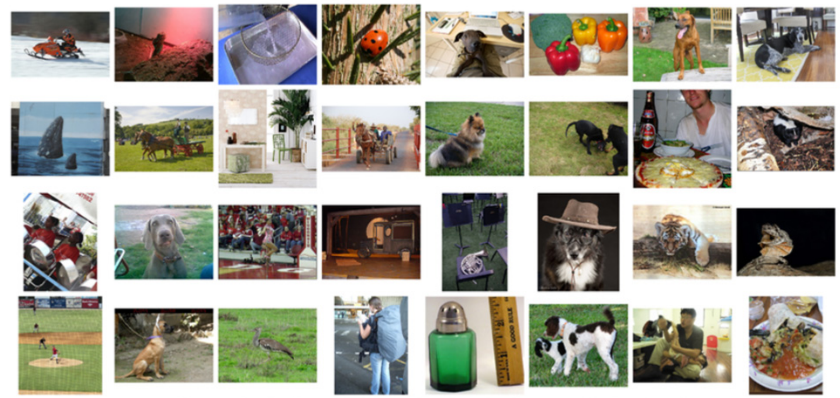


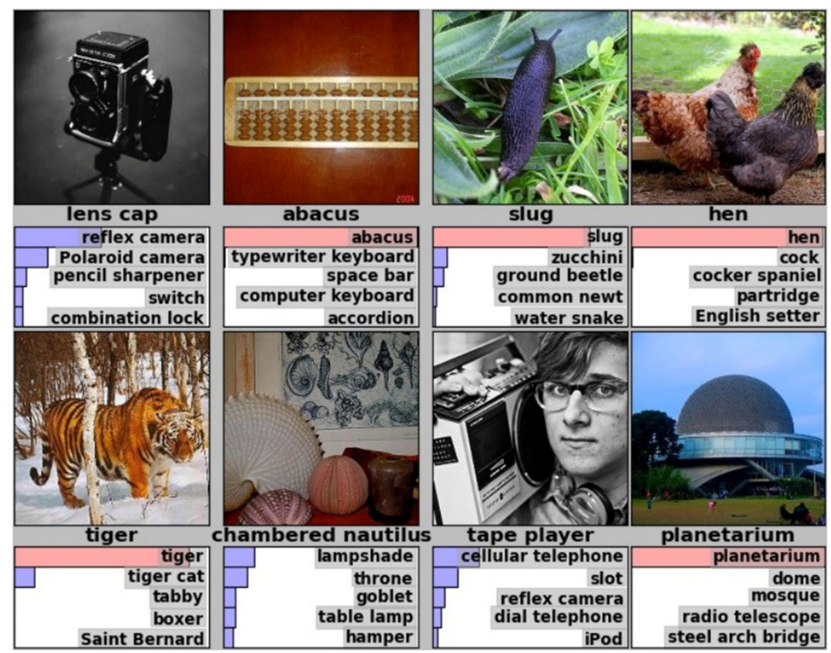
Total only 82 errors from LeNet-5. correct answer left and right is the machine answer.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11):2278-2324, November 1998

More General Image Recognition

- ImageNet
  - Over 15M labeled high resolution images
  - Roughly 22K categories
  - Collected from web and labeled by Amazon Mechanical Turk
- The Image/scene classification challenge
  - Image/scene classification
  - Metric: Hit@5 error rate - make 5 guesses about the image label





http://cognitiveseo.com/blog/6511/will-google-read-rank-images-near-future/

Russakovsky O, Deng J, Su H, et al. Imagenet large scale visual recognition challenge[J]. International Journal of Computer Vision, 2015, 115(3): 211-252.

Leadertable (ImageNet image classification)

2015 ResNet (ILSVRC'15) 3.57

Year	Codename	Error (percent)	99.9% Conf Int
2014	**GoogLeNet**	**6.66**	**6.40 - 6.92**
2014	VGG	7.32	7.05 - 7.60
2014	MSRA	8.06	7.78 - 8.34
2014	AHoward	8.11	7.83 - 8.39
2014	DeeperVision	9.51	9.21 - 9.82
2013	Clarifai†	11.20	10.87 - 11.53
2014	CASIAWS†	11.36	11.03 - 11.69
2014	Trimps†	11.46	11.13 - 11.80
2014	Adobe†	11.58	11.25 - 11.91
2013	**Clarifai**	**11.74**	**11.41 - 12.08**
2013	NUS	12.95	12.60 - 13.30
2013	ZF	13.51	13.14 - 13.87
2013	AHoward	13.55	13.20 - 13.91
2013	OverFeat	14.18	13.83 - 14.54
2014	Orange†	14.80	14.43 - 15.17
2012	SuperVision†	15.32	14.94 - 15.69
2012	**SuperVision**	**16.42**	**16.04 - 16.80**
2012	ISI	26.17	25.71 - 26.65
2012	VGG	26.98	26.53 - 27.43
2012	XRCE	27.06	26.60 - 27.52
2012	UvA	29.58	29.09 - 30.04

Microsoft ResNet, a 152 layers network

GoogLeNet, 22 layers network

U. of Toronto, SuperVision, a 7 layers network

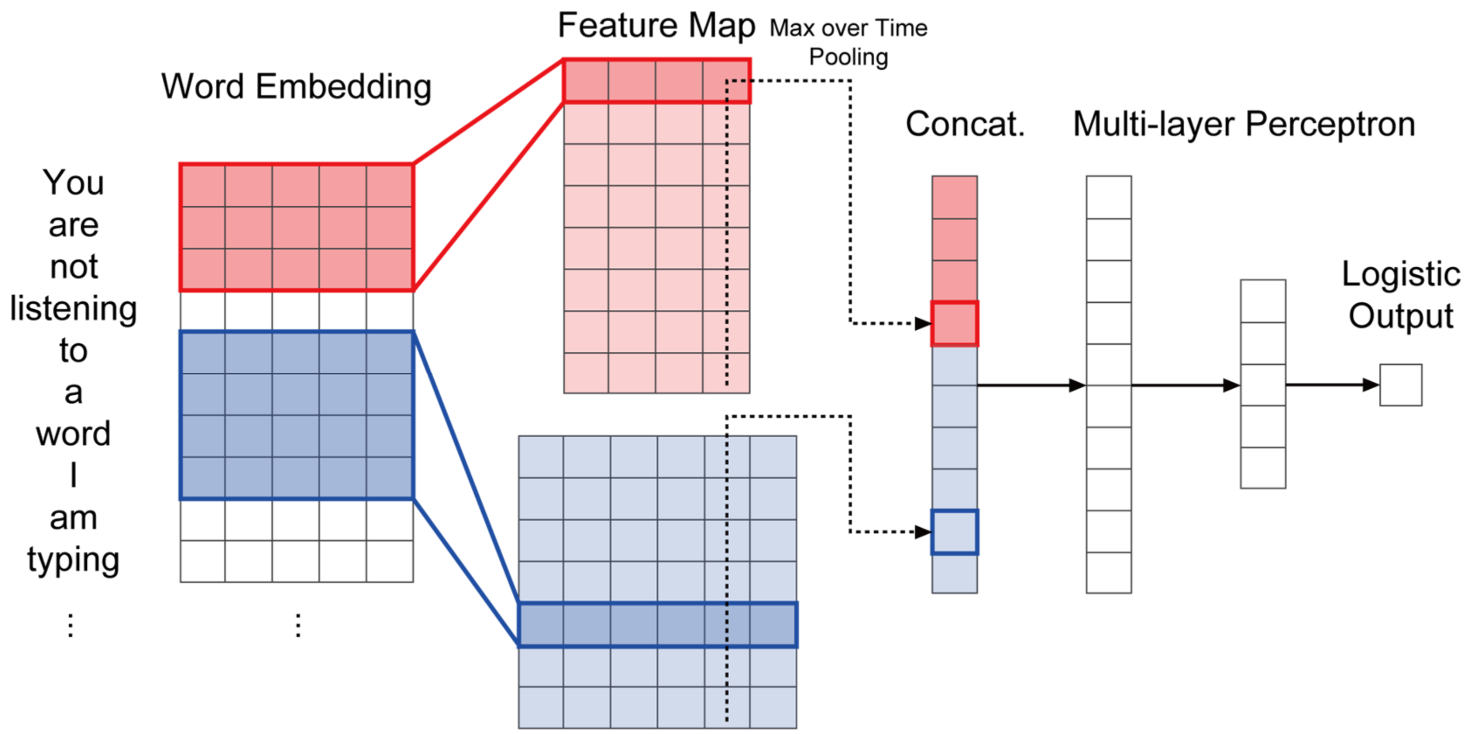
Unofficial human error is around *5.1%* on a subset

Why human error still? When labeling, human raters judged whether it belongs to a class (binary classification); the challenge is a 1000-class classification problem.

http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/)

Russakovsky O, Deng J, Su H, et al. Imagenet large scale visual recognition challenge[J]. International Journal of Computer Vision, 2015, 115(3): 211-252.

Use Case: Text Classification



- Word embedding: map each word to a *k*-dimensional dense vector
- CNN kernel: *n* x *k* matrix to explore the neighbor *k* words' patterns
- Max-over-time pooling: find the most salient pattern from the text for each kernel
- MLP: further feature interaction and distill high-level patterns

[Kim, Y. 2014. Convolutional neural networks for sentence classification. EMNLP 2014.]

Recurrent Neural Network (RNN)

- To model sequential data
  - Text
  - Time series
- Trained by Back-Propagation Through Time (BPTT)

x : input vector, o : output vector,

s : hidden state vector,

$\mathbf{U}$: layer 1 param. matrix,

$\mathbf{V}$: layer 2 param. matrix,

f: tanh or ReLU

$o = f(s\mathbf{V})$

$s = f(x\mathbf{U})$

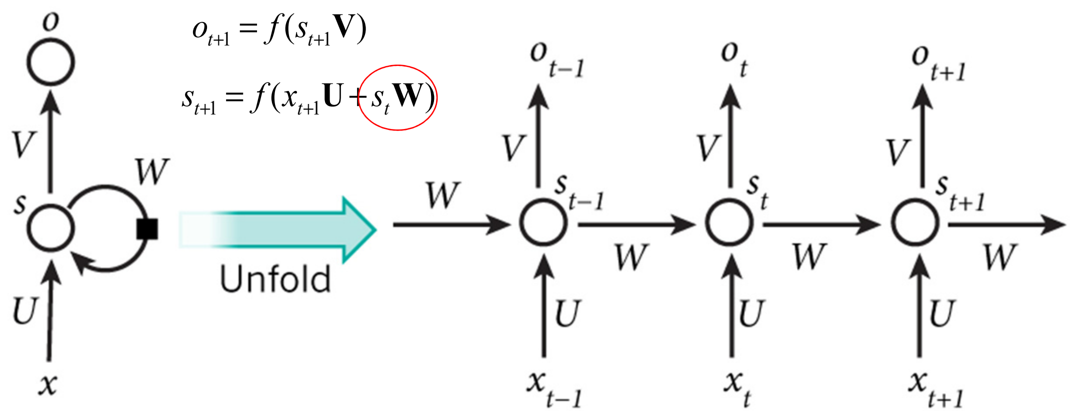
Two-layer feedforward network

Add time-dependency of the hidden state *s*

W : State transition param. matrix

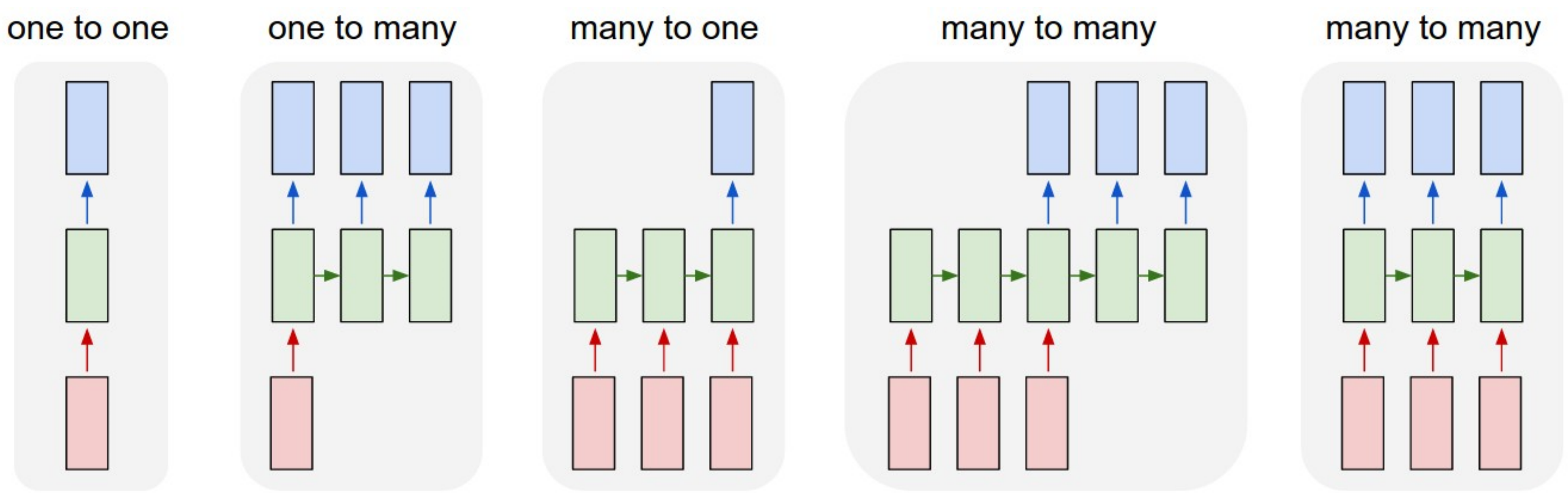
$o_{t+1} = f(s_{t+1}\mathbf{V})$

$s_{t+1} = f(x_{t+1}\mathbf{U} + s_t\mathbf{W})$



[http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/]

Different RNNs



Vanilla NN

Image captioning
Text generation

Text classification
Sentiment analysis

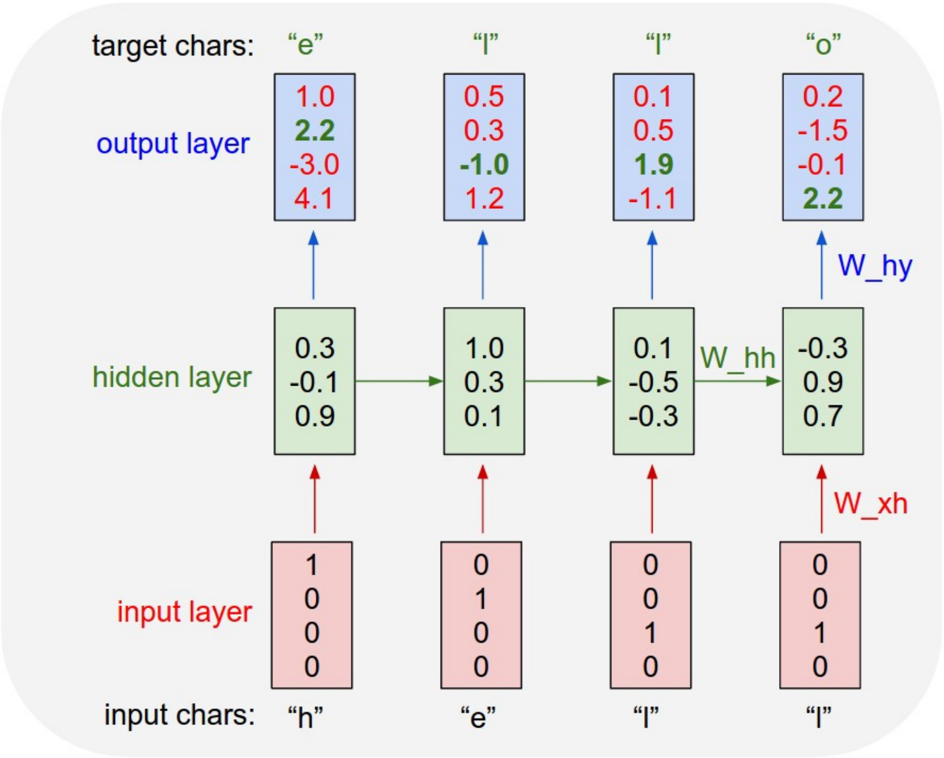
Machine translation
Dialogue system

Stock price estimation
Video frame classification

- Different architecture for various tasks
- Strongly recommend Andrej Karpathy's blog
  - http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Use Case: Language Model

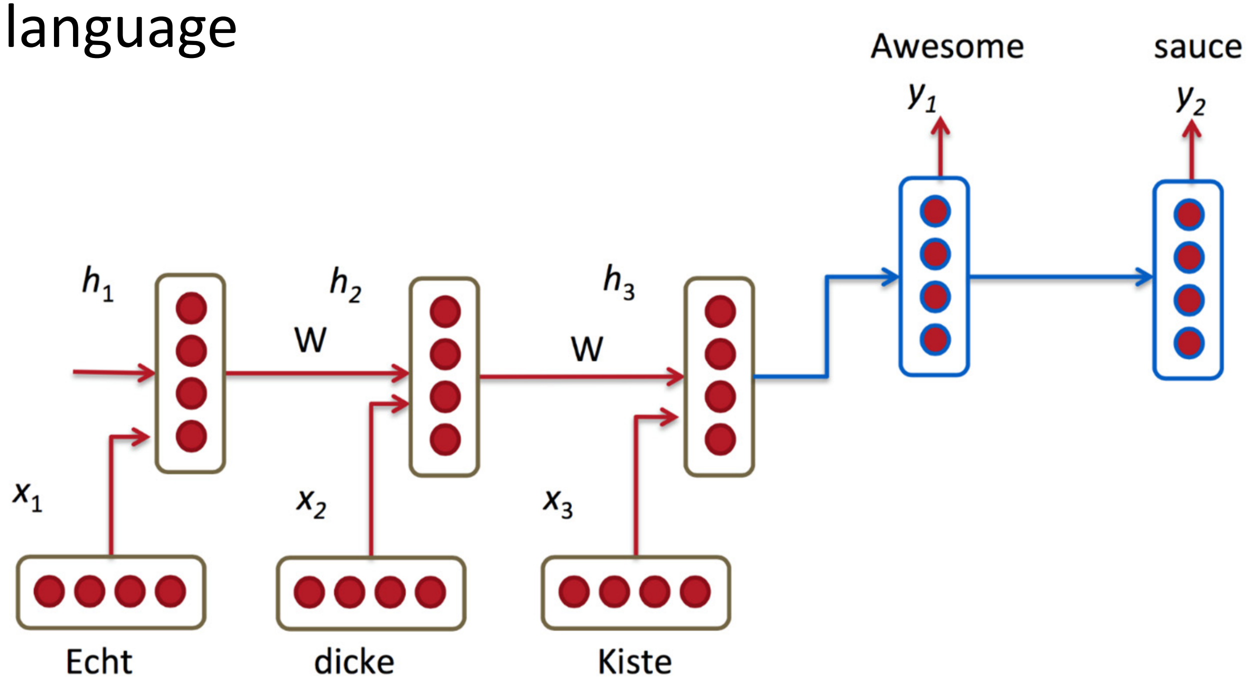
- Word-level or even character-level language model
  - Given previous words/characters, predict the next



[http://karpathy.github.io/2015/05/21/rnn-effectiveness/]

Use Case: Machine Translation

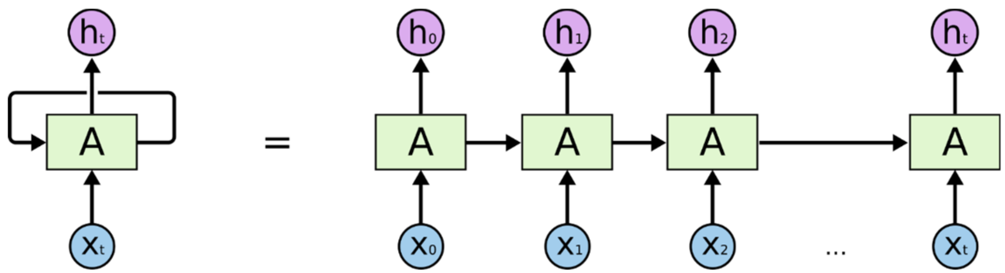
- Encode/decode RNN
  - First, encode the input sentence (into a vector e.g. h_3)
  - Then decode the vector into the sentence in another language



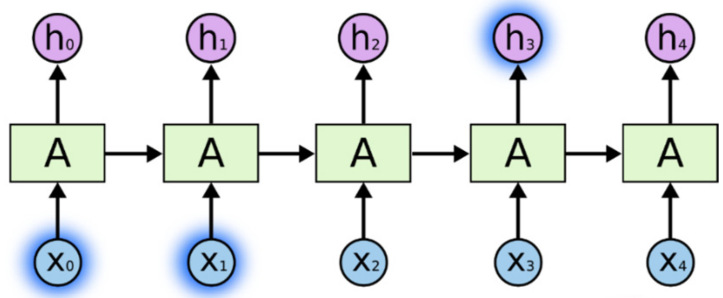
[http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/]

Problem of RNN

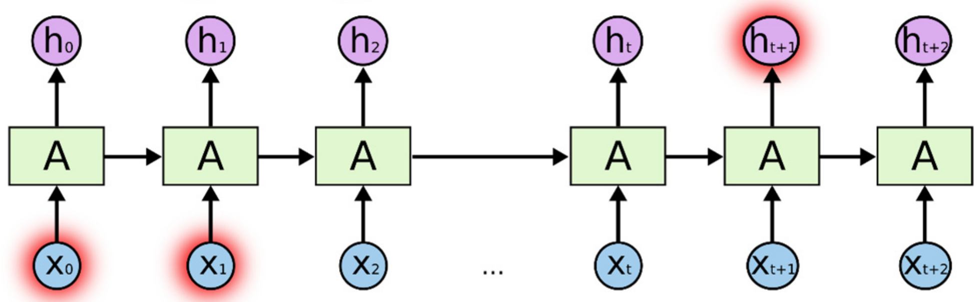
- Problem: RNN cannot nicely leverage the early information



Gap dependency

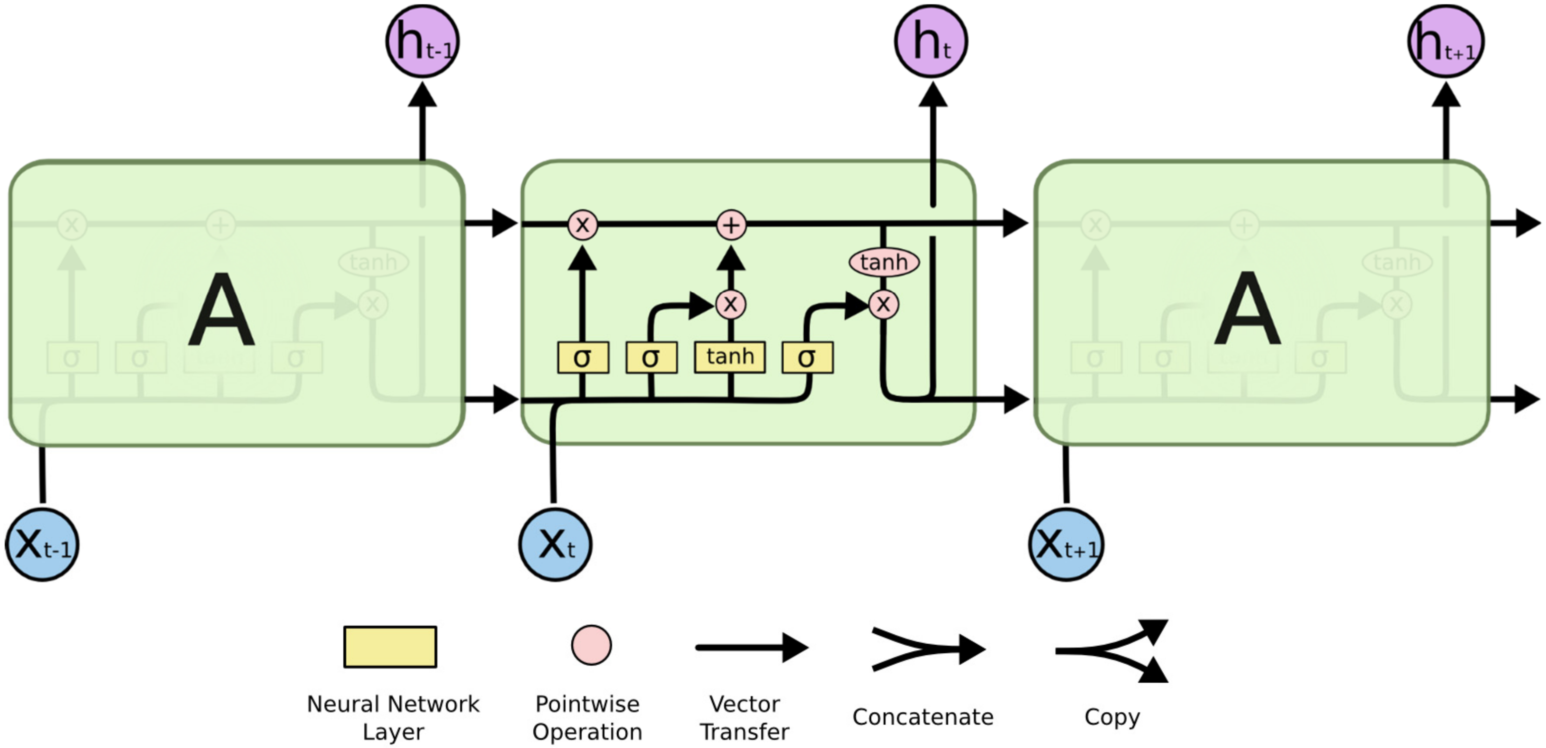


Long-term dependency



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

Long Short-Term Memory (LSTM)

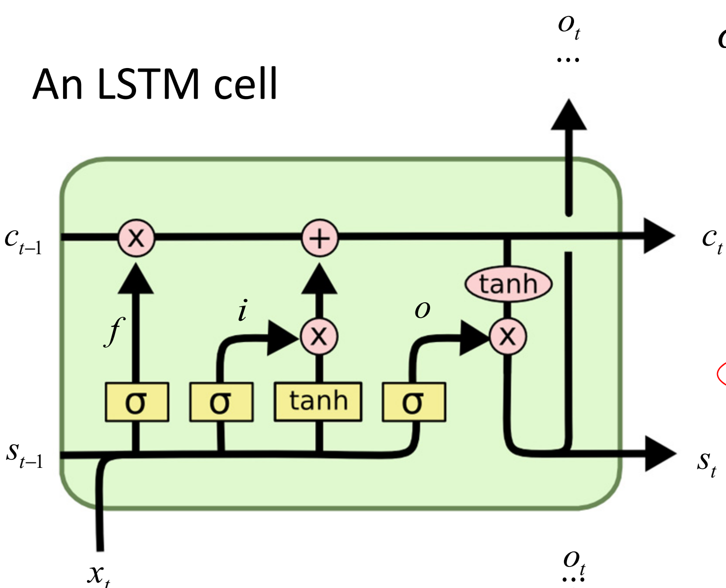


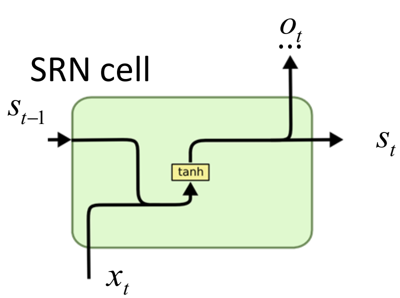
[http://colah.github.io/posts/2015-08-Understanding-LSTMs/]

[Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.]

LSTM Cell

- An LSTM cell learn to decide which to remember/forget





σ : sigmoid (control signal between 0 and 1); $\circ$: elementwise multiplication

$$i = \sigma(x_t U^i + s_{t-1} W^i)$$ input gate

$$f = \sigma(x_t U^f + s_{t-1} W^f)$$ forget gate

$$o = \sigma(x_t U^o + s_{t-1} W^o)$$ output gate

$$g = \ tanh(x_t U^g + s_{t-1} W^g)$$ “candidate” hidden state:

$$c_t = c_{t-1} \circ f + g \circ i$$ Cell internal memory

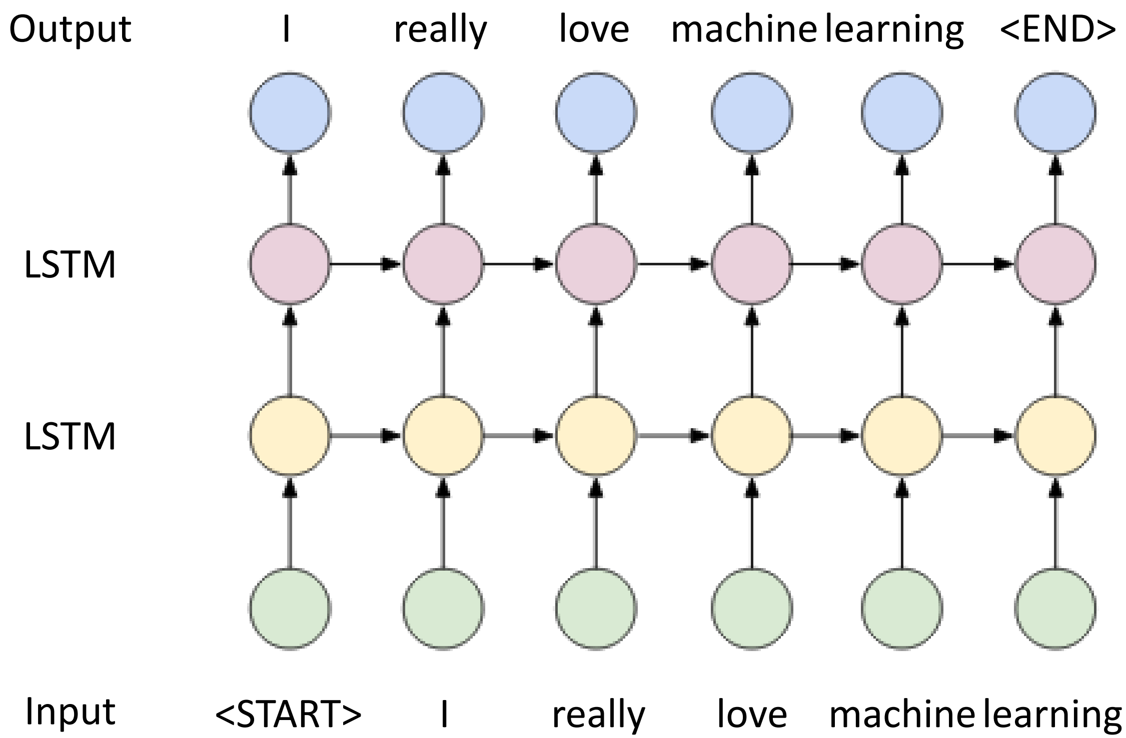
$$s_t = \tanh(c_t) \circ o$$ Hidden state

$$s_t = \tanh(x_t \mathbf{U} + s_{t-1} \mathbf{W})$$

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

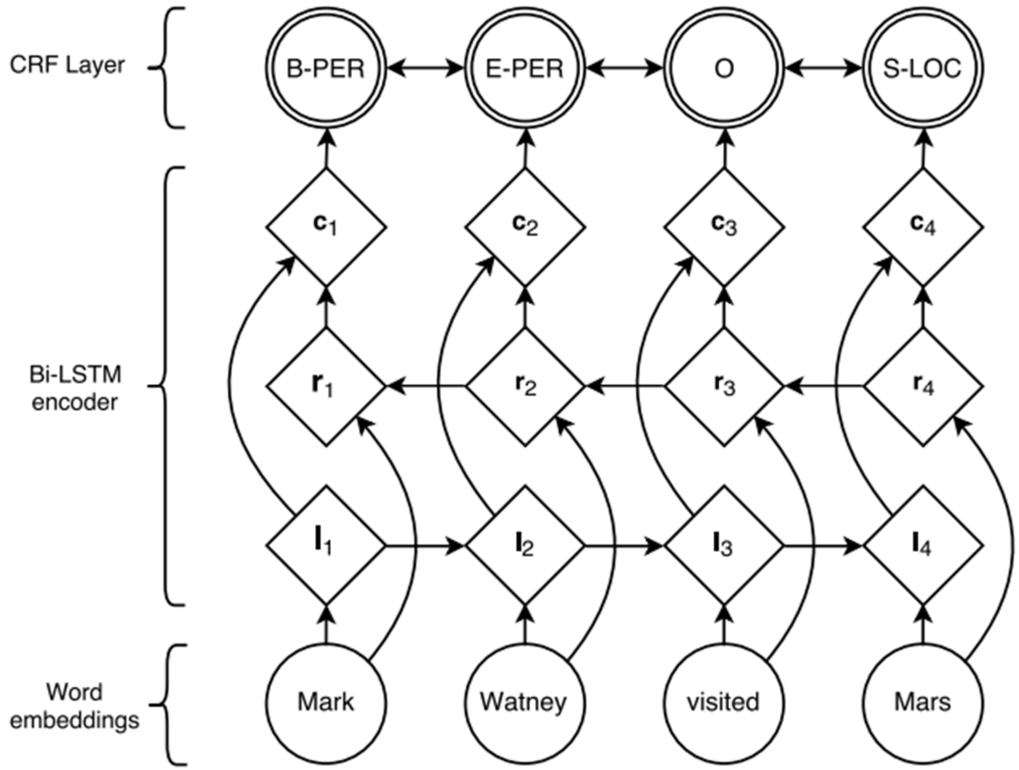
[Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.]

Use Case: Text Generation



- A demo on character-level text generation
  - http://cs.stanford.edu/people/karpathy/recurrentjs/

Use Case: Named Entity Recognition



[Guillaume Lample et al. Neural Architectures for Named Entity Recognition. NAACL-HLT]

Word embedding

- From bag of words to word embedding
  - Use a real- valued vector in R^m to represent a word (concept)

  v("cat")=(0.2, -0.4, 0.7, ...)

  v("mat")=(0.0, 0.6, -0.1, ...)

- Continuous bag of word (CBOW) model (word2vec)
  - Input/output words x/y are one-hot encoded
  - Hidden layer is shared for all input words

***N*-dim Vector representation of a word**

Hidden nodes:

$$\mathbf{h} = \frac{1}{C}\mathbf{W}\cdot(\mathbf{x}_1+\mathbf{x}_2+\cdots+\mathbf{x}_C) = \frac{1}{C}\cdot(\mathbf{v}_{w_1}+\mathbf{v}_{w_2}+\cdots+\mathbf{v}_{w_C})$$

The cross-entropy loss:

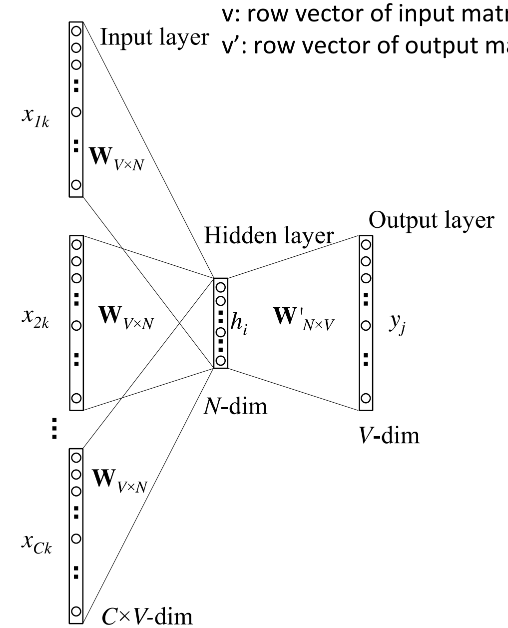
$$E = -\log p(w_O|w_{I,1},\cdots,w_{I,C}) = -{\mathbf{v}'_{w_O}}^T\cdot\mathbf{h}+\log\sum_{j'=1}^{V}\exp({\mathbf{v}'_{w_j}}^T\cdot\mathbf{h})$$

The gradient updates:

$${\mathbf{v}'_{w_j}}^{(\text{new})} = {\mathbf{v}'_{w_j}}^{(\text{old})} - \eta\cdot e_j\cdot\mathbf{h} \qquad \text{for } j=1,2,\cdots,V.$$

$$\mathbf{v}^{(\text{new})}_{w_{I,c}} = \mathbf{v}^{(\text{old})}_{w_{I,c}} - \frac{1}{C}\cdot\eta\cdot\text{EH} \qquad \text{for } c=1,2,\cdots,C.$$

V: vocabulary size;
C: num. input words;
v: row vector of input matrix W;
v': row vector of output matrix W'



Continuous bag of word (CBOW) model

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V}\frac{\partial E}{\partial u_j}\cdot\frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{V} e_j\cdot w'_{ij} := \text{EH}_i$$

Rong, Xin. "word2vec parameter learning explained." arXiv preprint arXiv:1411.2738 (2014).

Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).

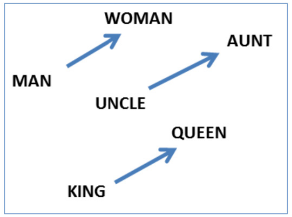
Remarkable properties from Word embedding

- Simple algebraic operations with the word vectors

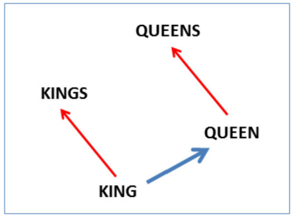
Using X = v("biggest") − v("big") + v("small") as query and searching for the nearest word based on cosine distance results in v("smallest")

v("woman")−v("man") ≃ v("aunt")−v("uncle")

v("woman")−v("man") ≃ v("queen")−v("king")



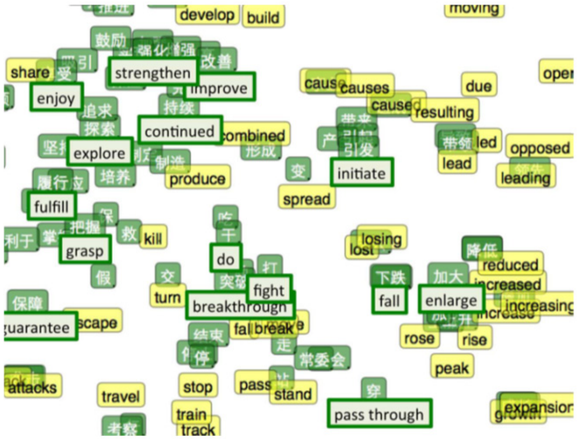
Vector offsets for gender relation



The singular/plural relation for two words

Word the relationship is defined by subtracting two word vectors, and the result is added to another word. Thus for example, **Paris - France + Italy = Rome**.

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza



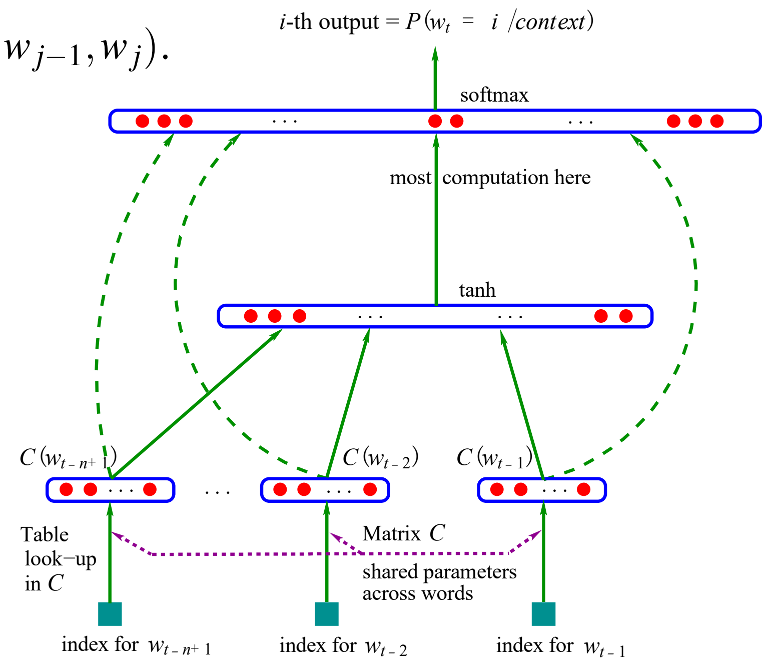
Mikolov, Tomas, Wen-tau Yih, and Geoffrey Zweig. "Linguistic Regularities in Continuous Space Word Representations." HLT-NAACL. 2013.
Zou, Will Y., et al. "Bilingual Word Embeddings for Phrase-Based Machine Translation." EMNLP. 2013.

Neural Language models

- *n-gram* model
  - Construct conditional probabilities for the next word, given combinations of the last *n-1* words (*contexts*)

$$\hat{P}(w_t|w_1^{t-1}) \approx \hat{P}(w_t|w_{t-n+1}^{t-1}) \quad \text{where} \quad w_i^j = (w_i, w_{i+1}, \cdots, w_{j-1}, w_j).$$

- Neural language model
  - associate with each word a *distributed word feature vector* for word embedding,
  - express the joint *probability function* of word sequences using those vectors, and
  - learn simultaneously the *word feature vectors* and the parameters of that *probability function*.



Bengio, Yoshua, et al. "Neural probabilistic language models." Innovations in Machine Learning. Springer Berlin Heidelberg, 2006. 137-186.

RNN based Language models

- The limitation of the feedforward network approach:
  - it has to fix the length context
- Recurrent network solves the issue
  - by keeping a (hidden) context and updating over time

x(t) is the input vector:
It is formed by **concatenating** vector *w(t)* representing current word, and hidden state s at time t − 1. w(t) is one hot encoder of a word

$$x(t) = [w(t), s(t-1)]$$

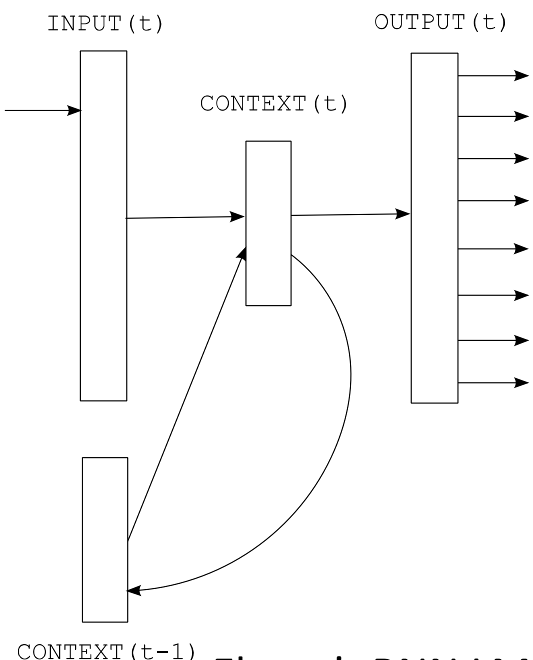
s(t) is state of the network (the hidden layer):

$$s_j(t) = f\left(\sum_i x_i(t) u_{ji}\right)$$

output is denoted as y(t):

$$y_k(t) = g\left(\sum_j s_j(t) v_{kj}\right)$$

Sigmoid for hidden layer $f(z) = \frac{1}{1+e^{-z}}$ **Softmax for output layer** $g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}$

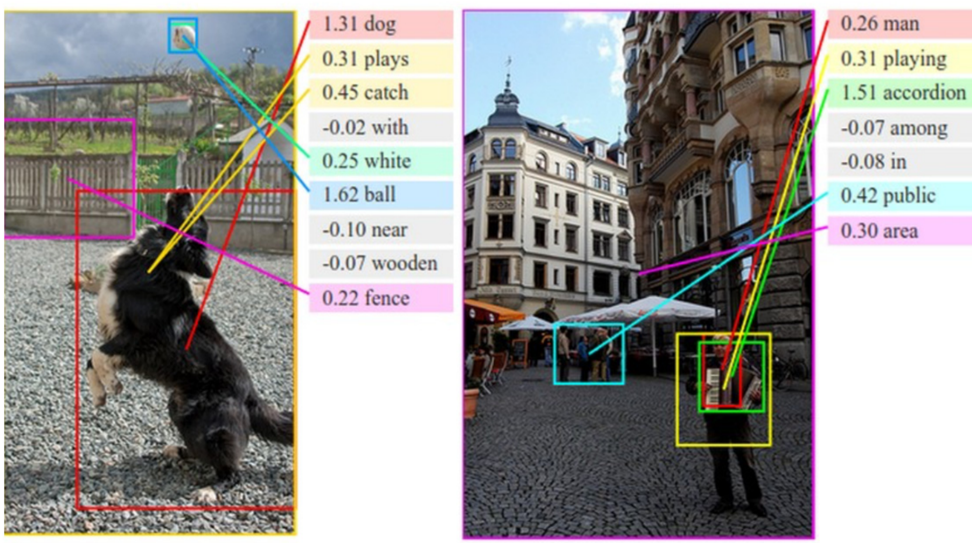


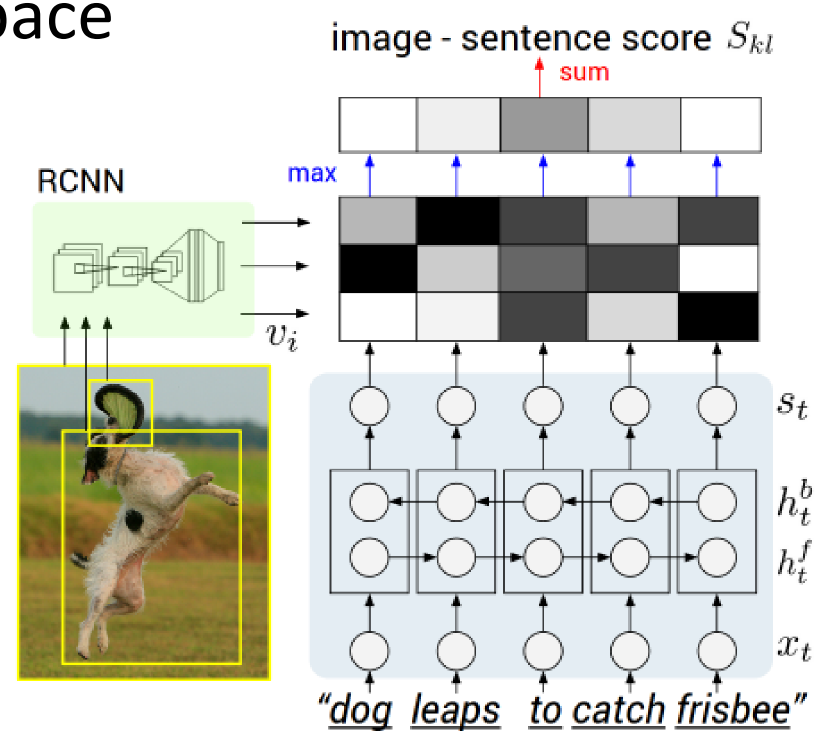
Elman's RNN LM

Mikolov, Tomas, et al. "Recurrent neural network based language model." INTERSPEECH. Vol. 2. 2010.
Elman J L. Finding structure in time[J]. Cognitive science, 1990, 14(2): 179-211.

Learning to align visual and language data

- Regional CNN + Bi-directional RNN
  - associates the two modalities through a common, multimodal embedding space



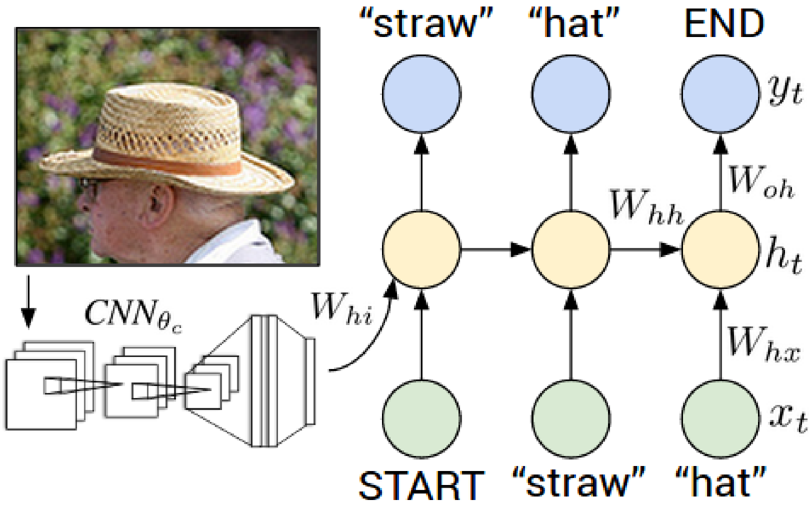


Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

Learning to generate image descriptions

- Trained CNN on images + RNN with sentence
  - The RNN takes a word, the previous context and defines a distribution over the next word
  - The RNN is conditioned on the image information at the first time step
  - START and END are special tokens.

"two young girls are playing with lego toy."



Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

Summary

- Universal Approximation: two-layer neural networks can approximate any functions
- Backpropagation is the most important training scheme for multi-layer neural networks so far
- Deep learning, i.e. deep architecture of NN trained with big data, works incredibly well
- Neural works built with other machine learning models achieve further success

Reference Materials

- Prof. Geoffery Hinton's Coursera course
  - https://www.coursera.org/learn/neural-networks
- Prof. Jun Wang's DL tutorial in UCL (special thanks)
  - http://www.slideshare.net/JunWang5/deep-learning-61493694
- Prof. Fei-fei Li's CS231n in Stanford
  - http://cs231n.stanford.edu/
- Prof. Kai Yu's DL Course in SJTU
  - http://speechlab.sjtu.edu.cn/~kyu/node/10
- Michael Nielsen's online DL book
  - http://neuralnetworksanddeeplearning.com/
- Research Blogs
  - Andrej Karpathy: http://karpathy.github.io/
  - Christopher Olah: http://colah.github.io/