Neural Network and Deep Learning

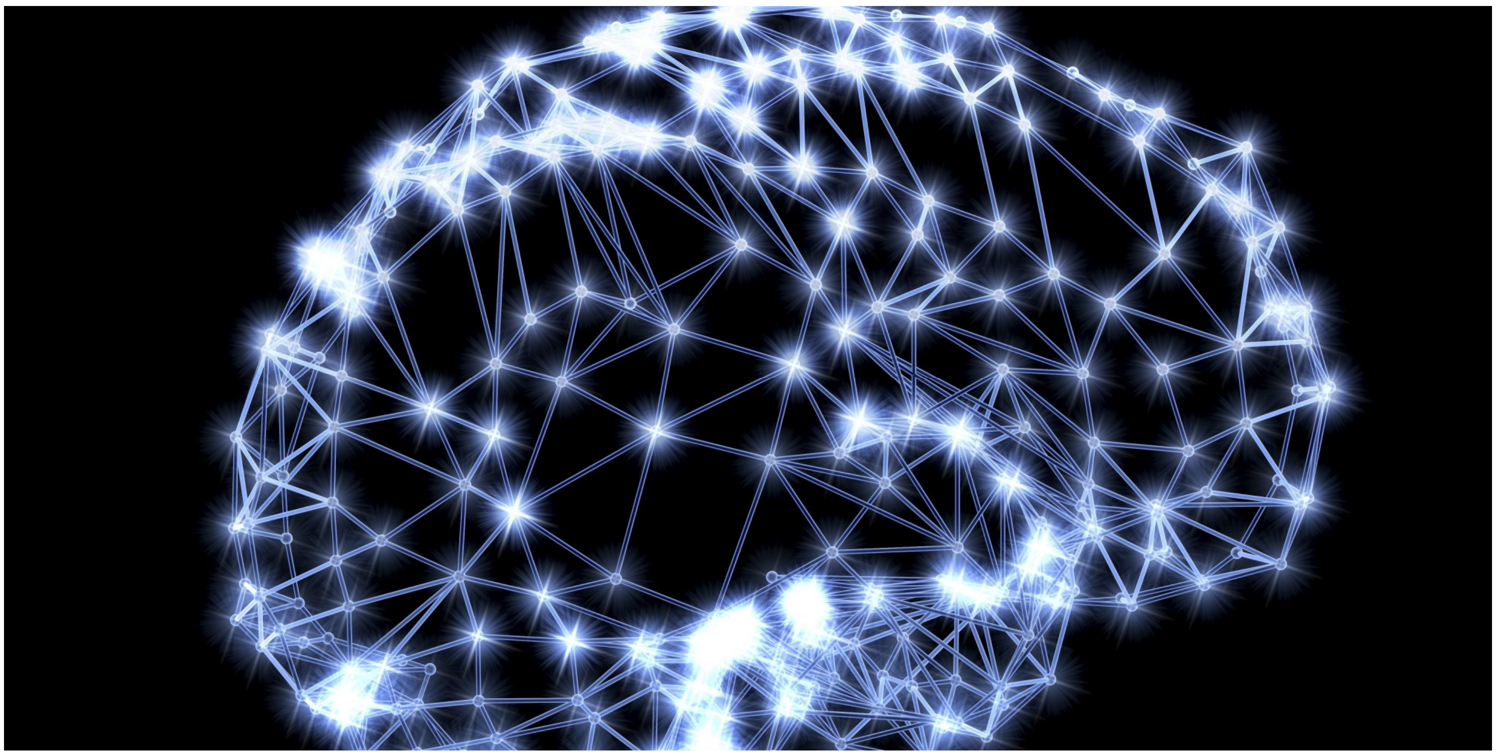
Md Shad Akhtar

Research Scholar

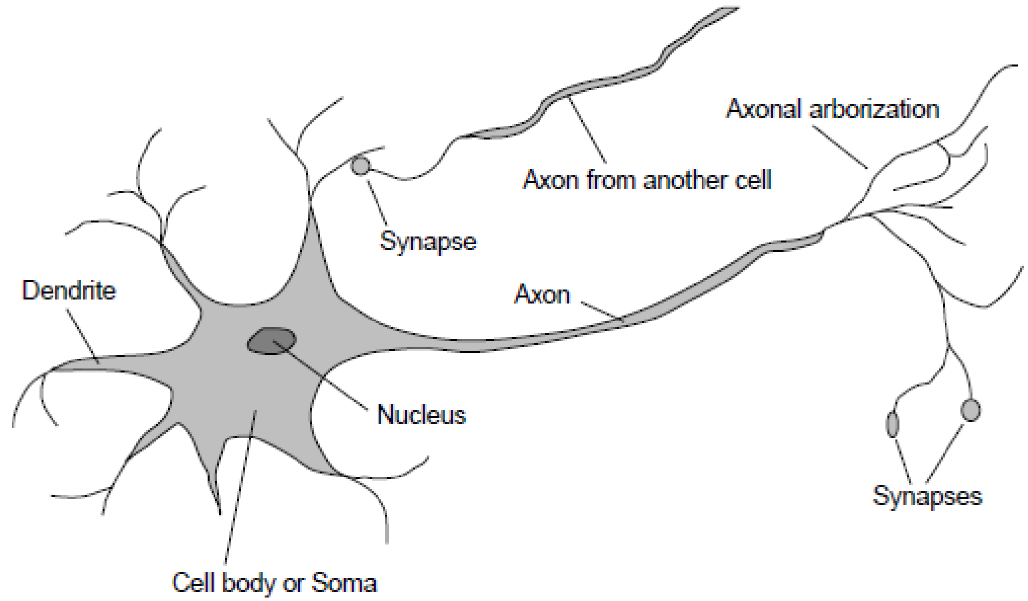
IIT Patna

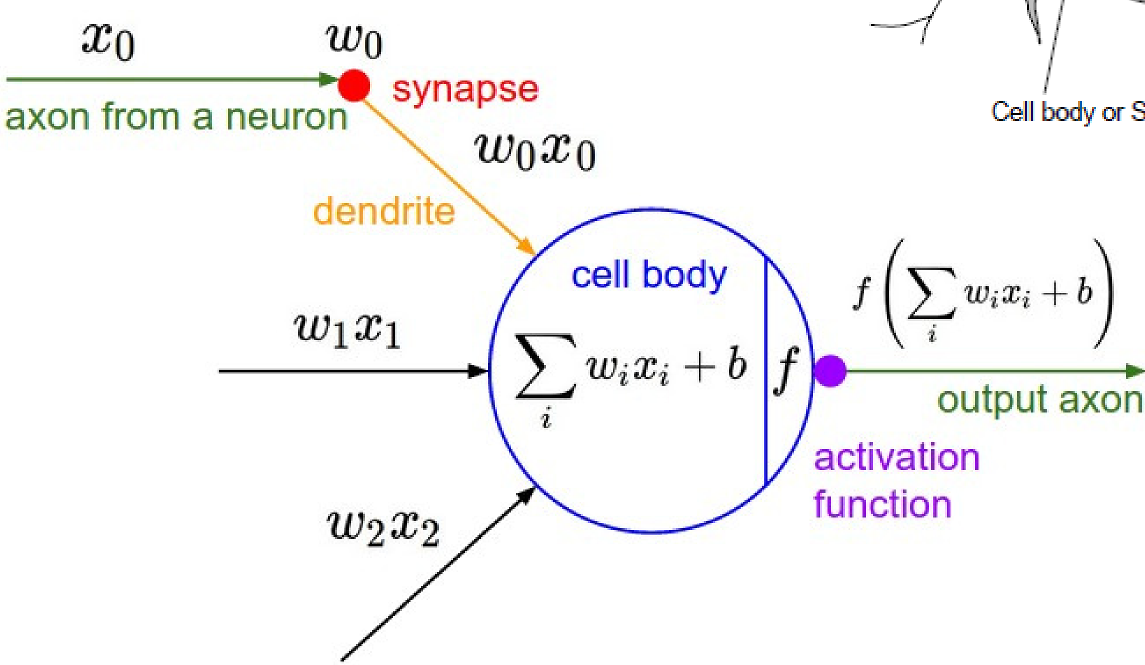
Neural Network

- *Mimics the functionality of a brain.*
- *A neural network is a graph with neurons (nodes, units etc.) connected by links.*



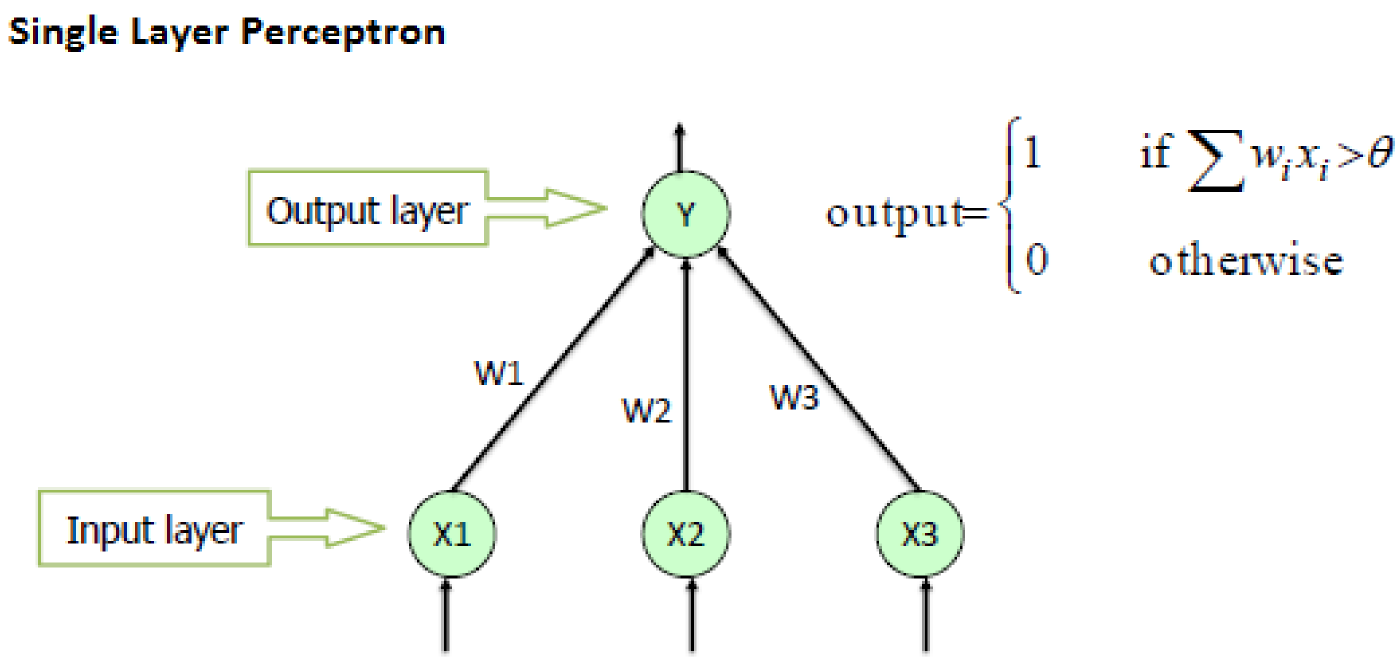
Neural Network: Neuron



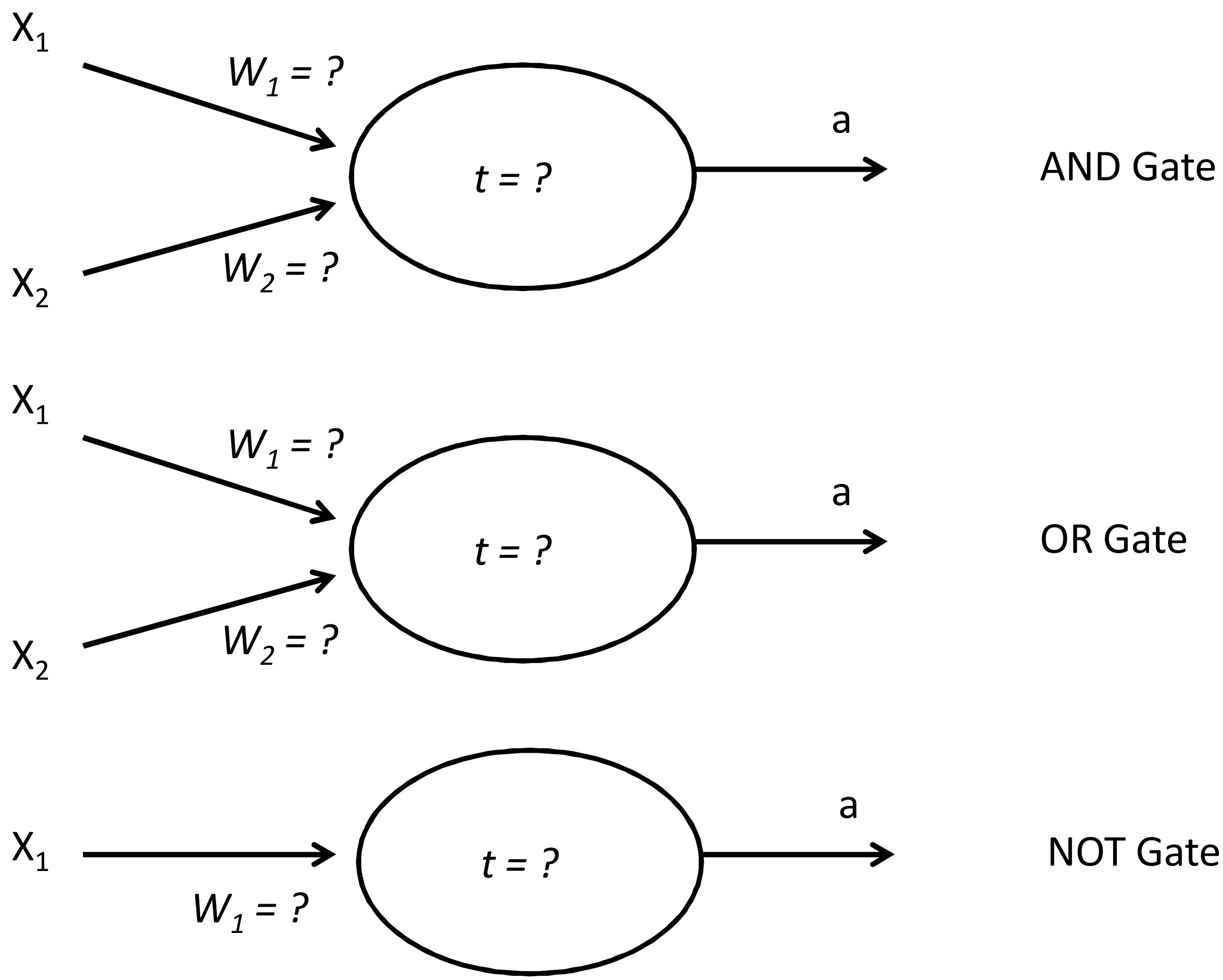


Neural Network: Perceptron

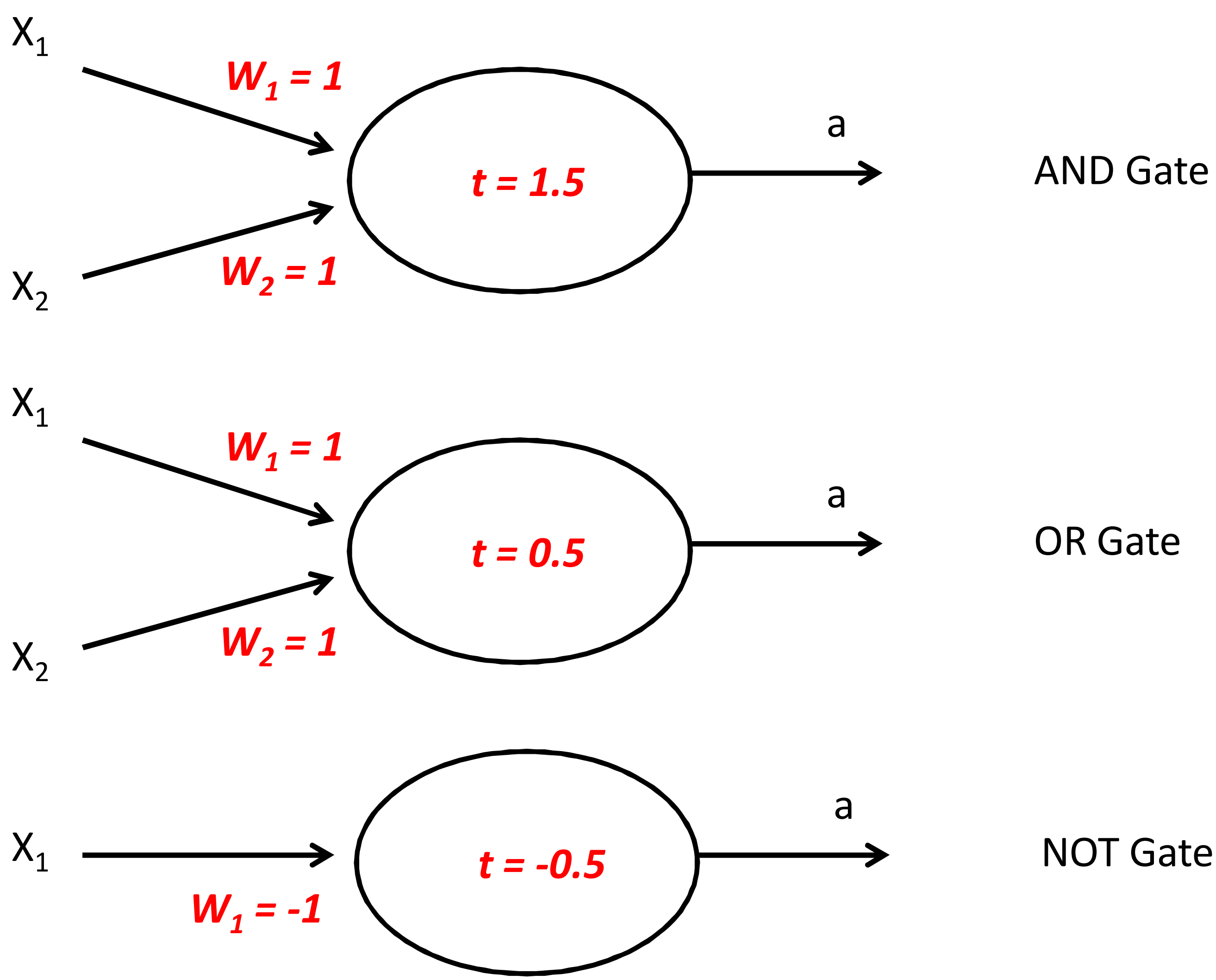
- Network with only single layer.
- No hidden layers



Neural Network: Perceptron

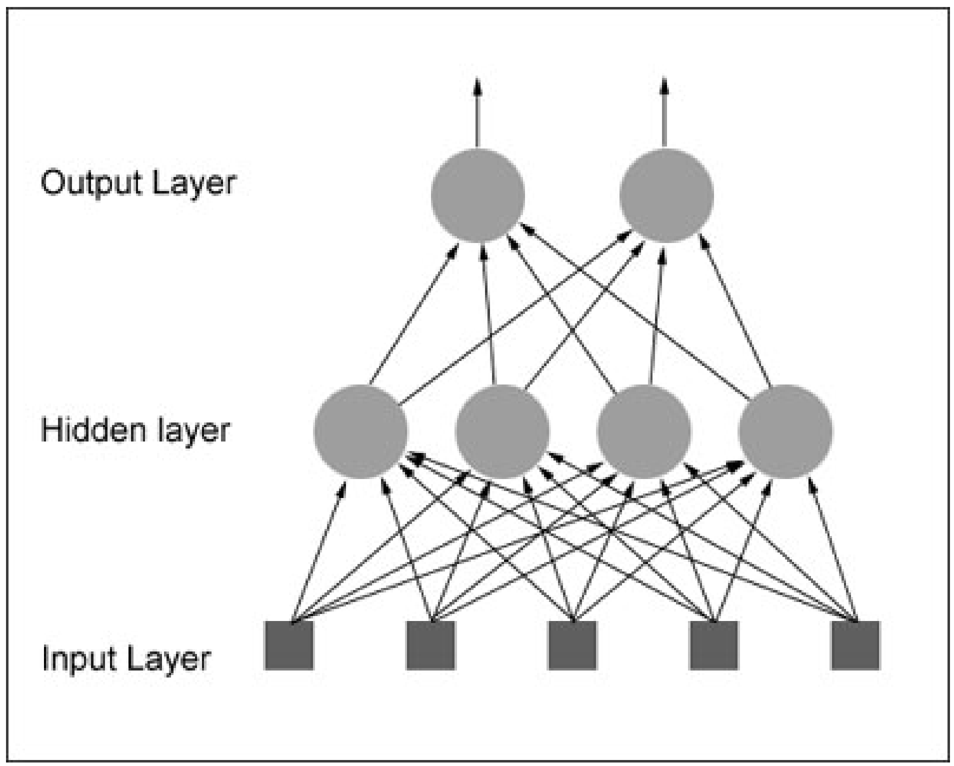


Neural Network: Perceptron



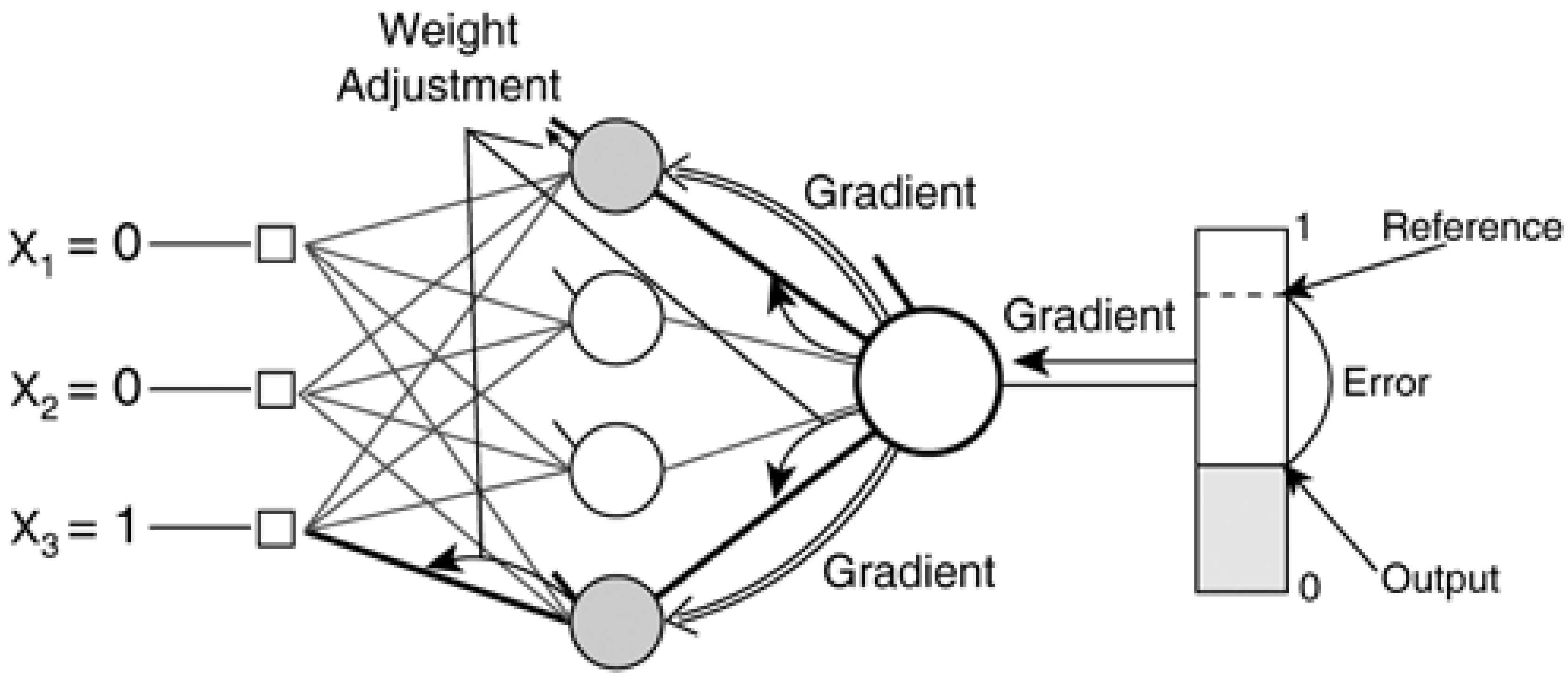
Neural Network: Multi Layer Perceptron (MLP) or Feed-Forward Network (FNN)

- Network with n+1 layers
- One output and n hidden layers.
- Typically n = 1

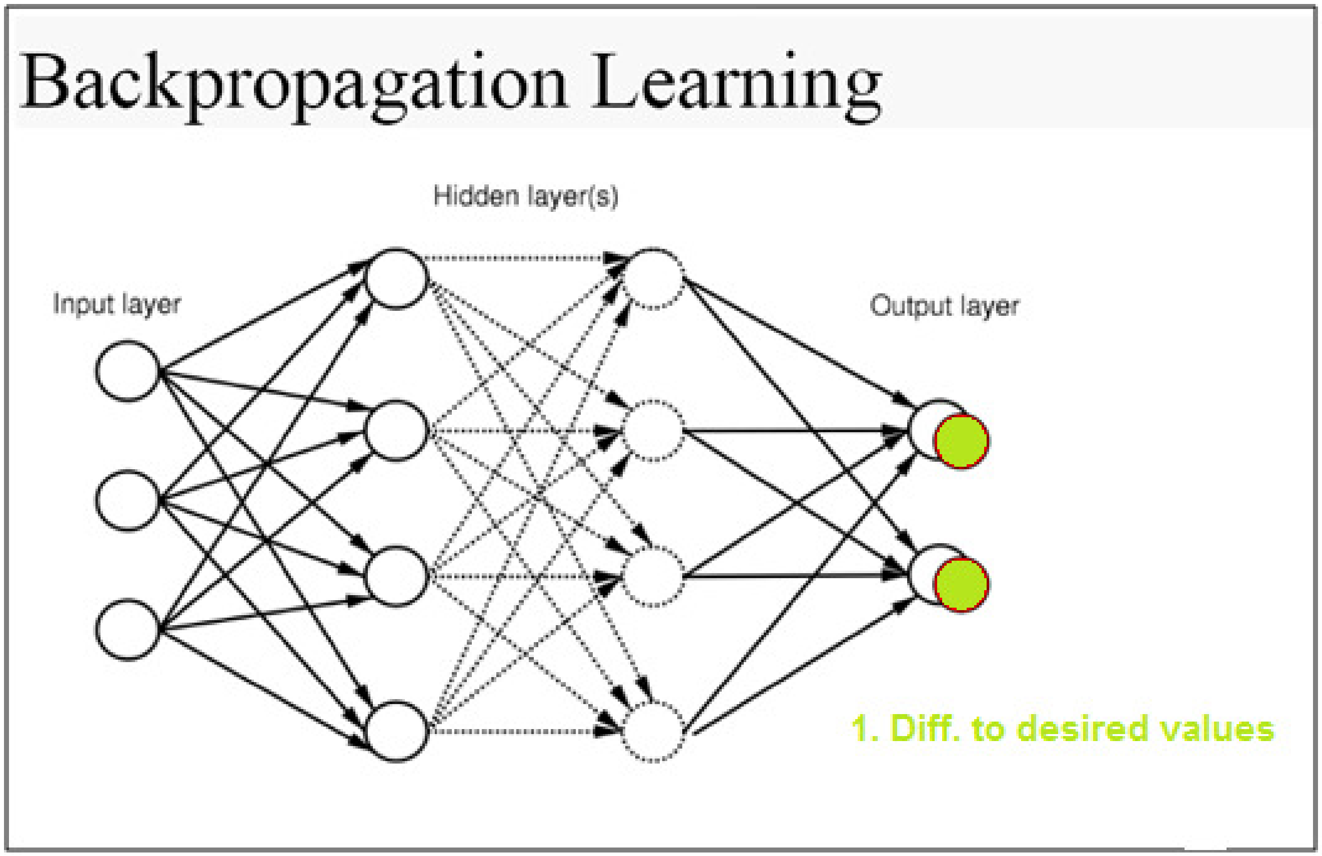


Training: Back propagation algorithm

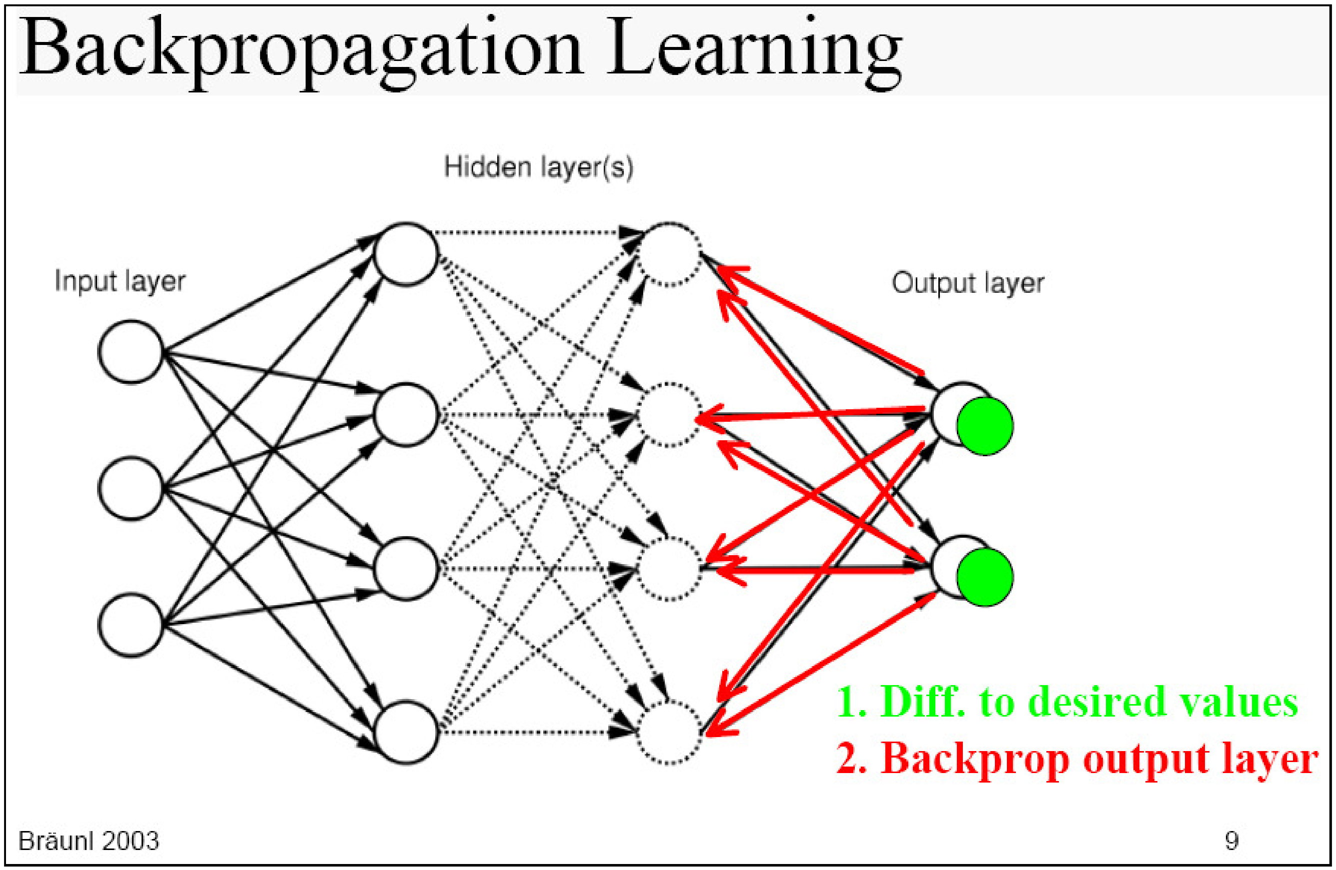
- Gradient decent algorithm



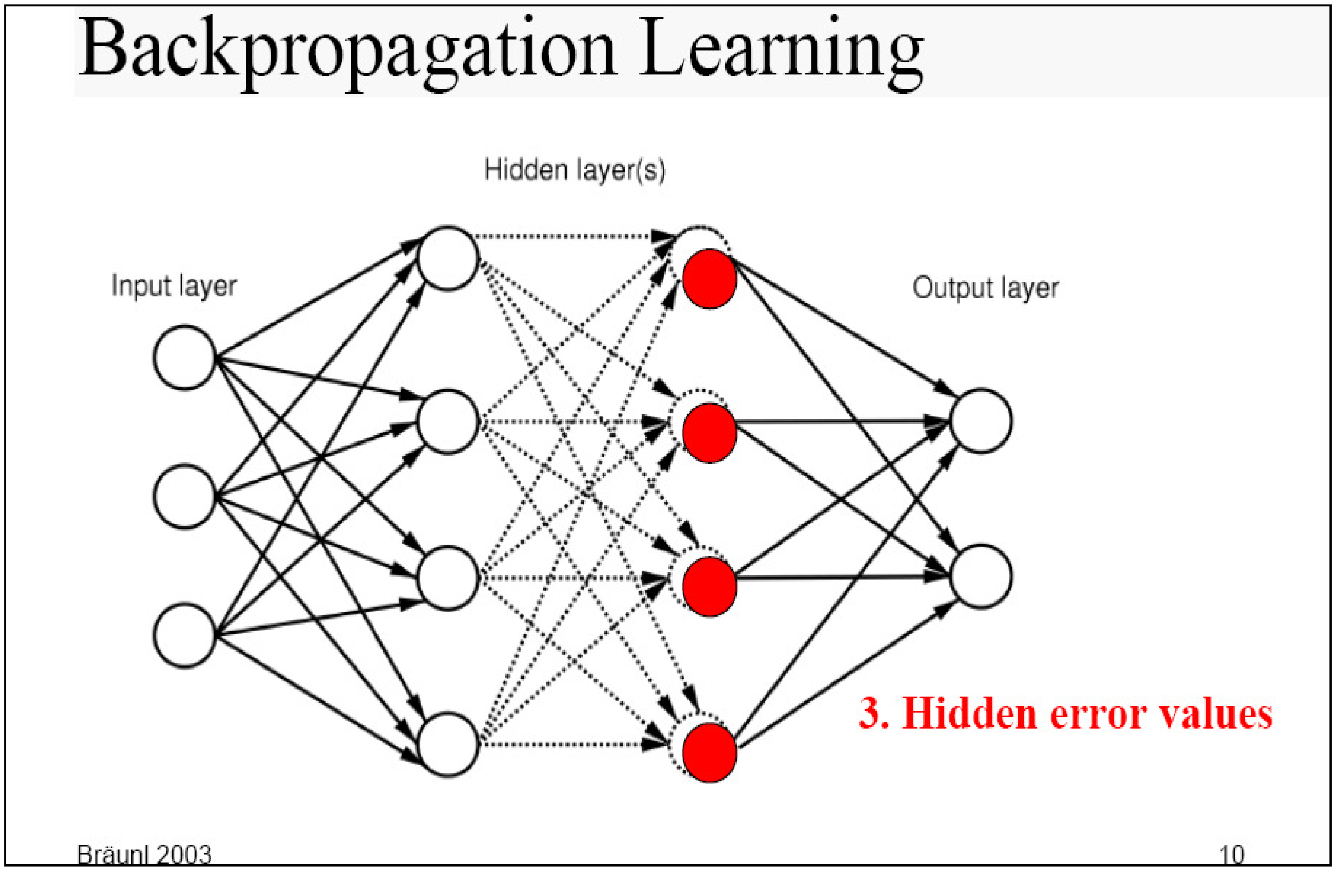
Training: Back propagation algorithm



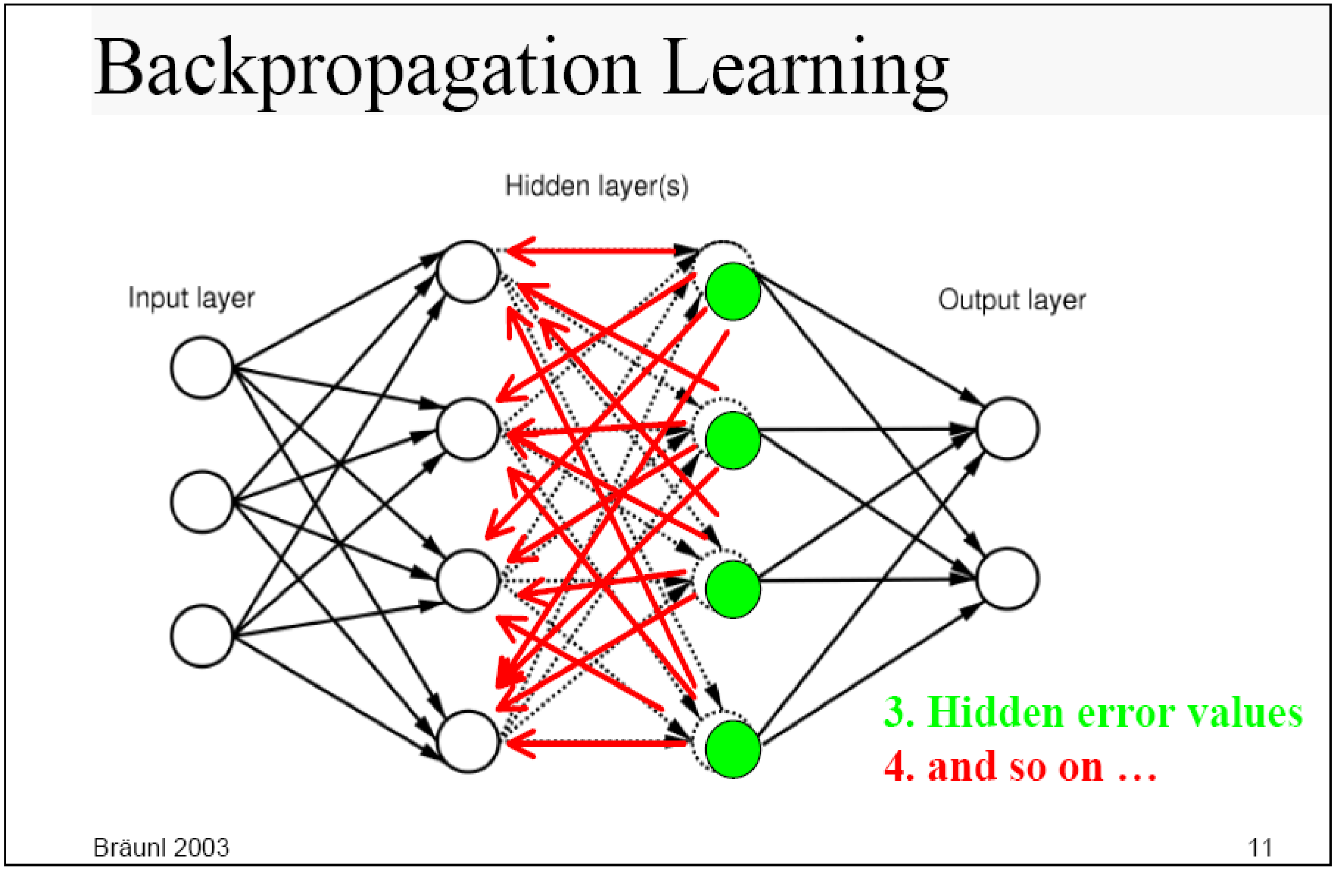
Training: Back propagation algorithm



Training: Back propagation algorithm



Training: Back propagation algorithm



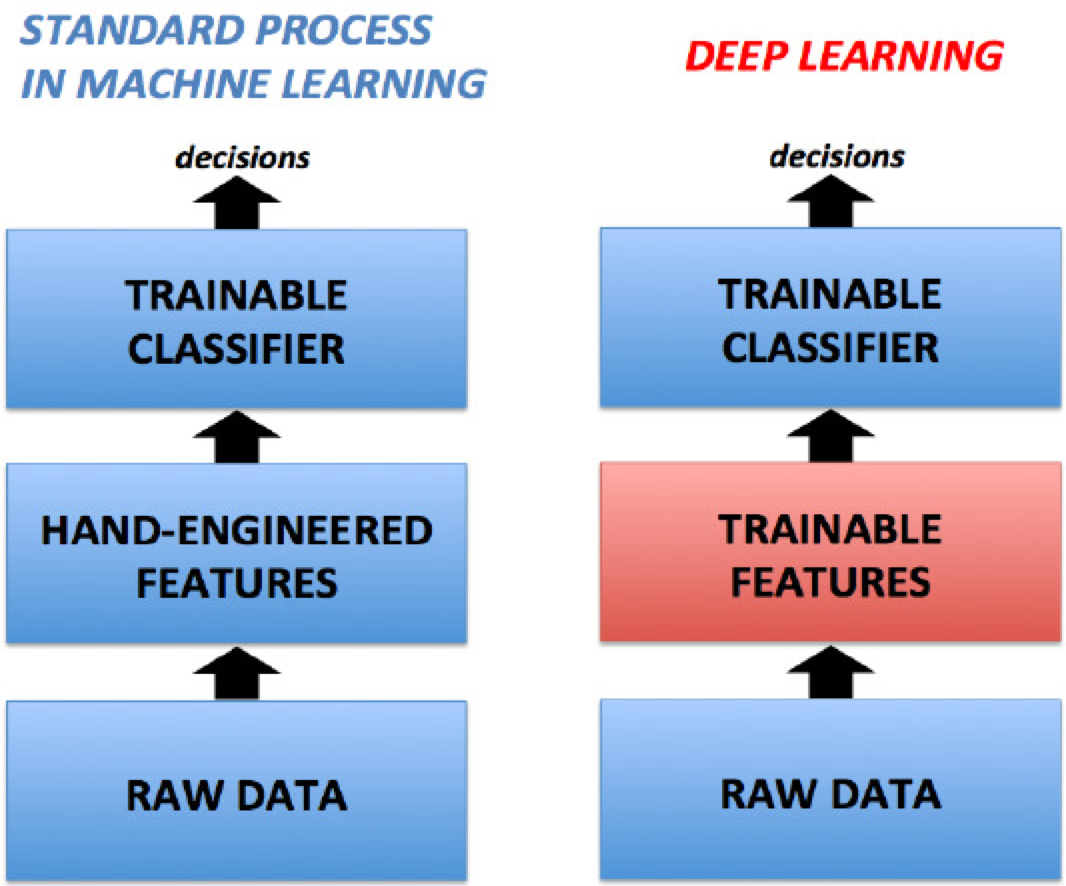
Training: Back propagation algorithm

1. Initialize network with random weights
2. For all training cases (called examples):
   a. Present training inputs to network and calculate output
   b. For all layers (starting with output layer, back to input layer):
      i. Compare network output with correct output (error function)
      ii. Adapt weights in current layer

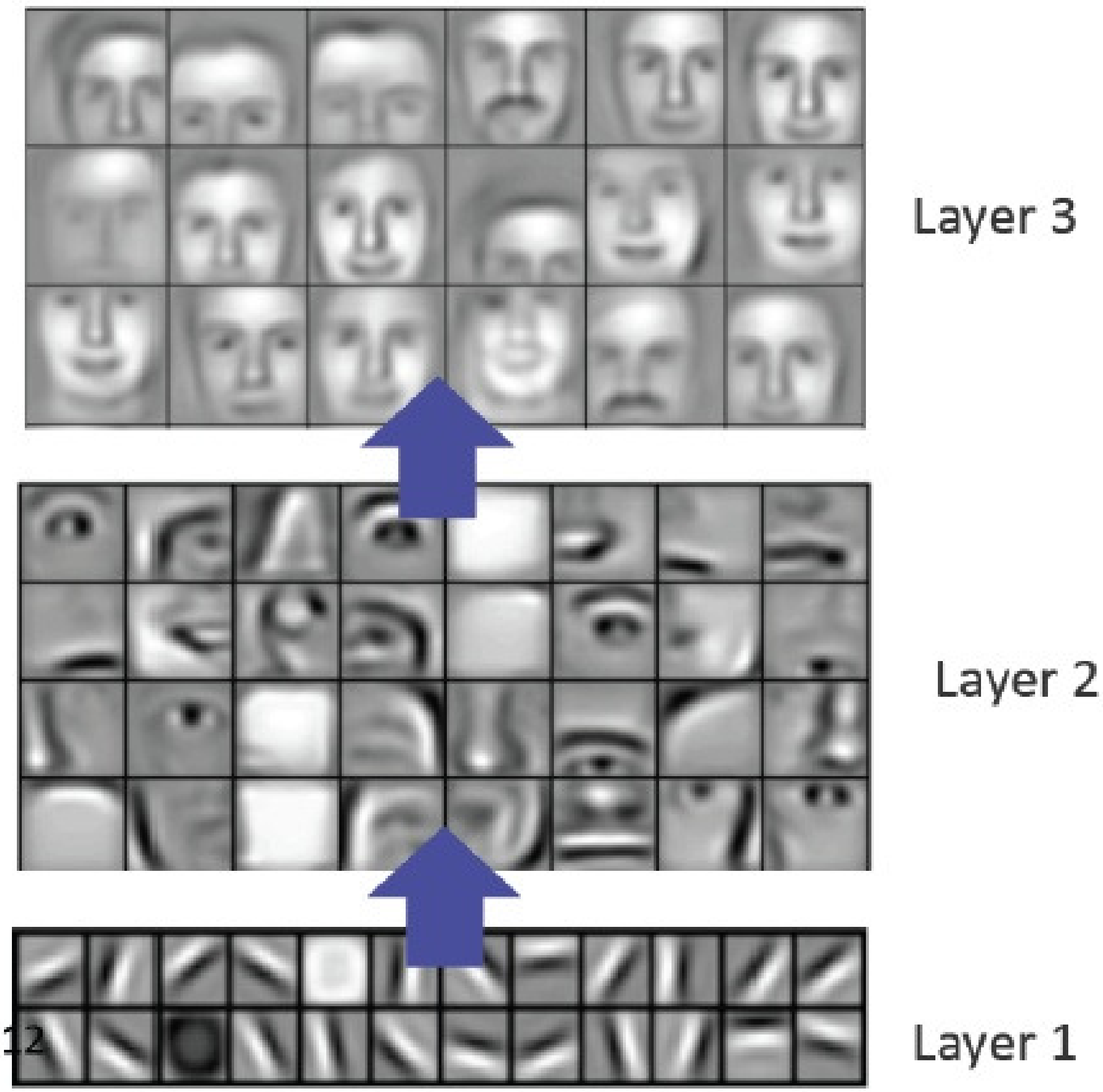
Deep Learning

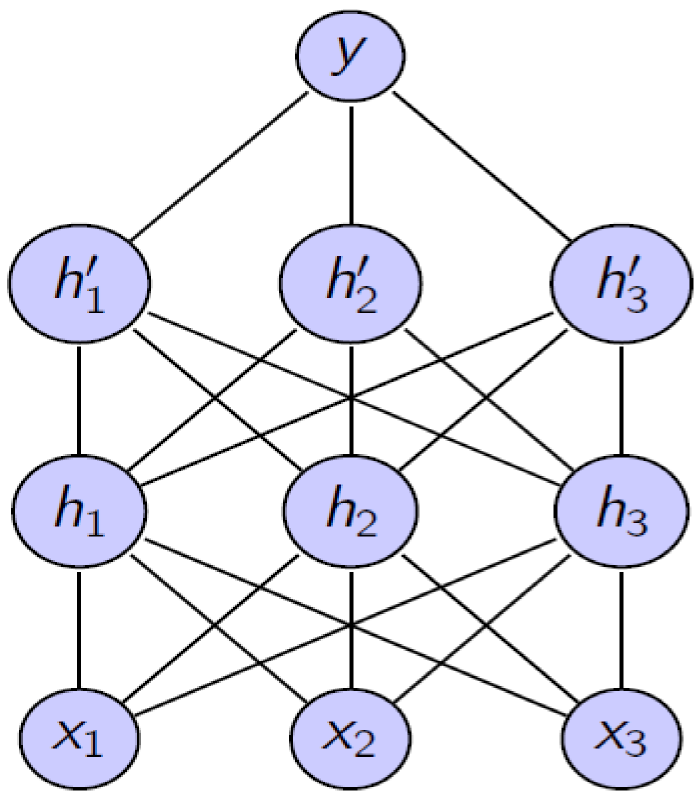
What is Deep Learning?

- A family of methods that uses deep architectures to learn high-level feature representations

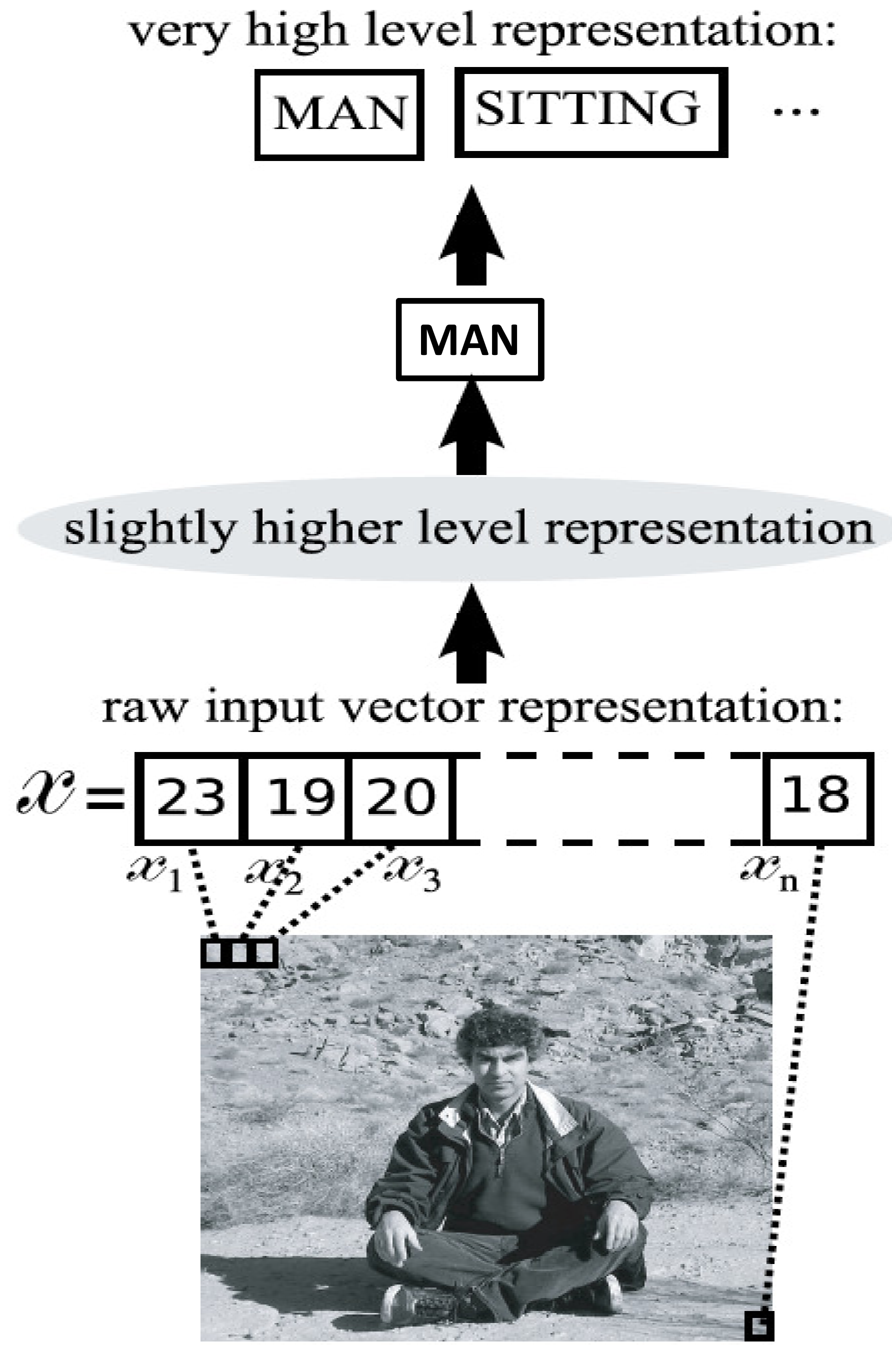


Example 1





y
h'1
h'2
h'3
h1
h2
h3
x1
x2
x3



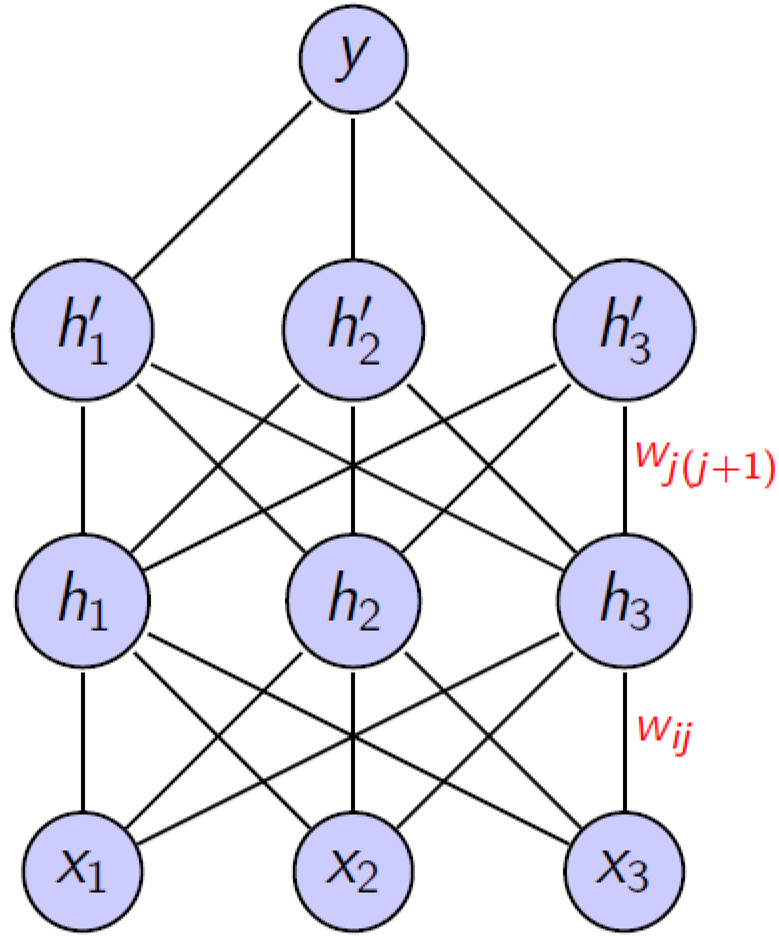
very high level representation:
MAN
SITTING
...
MAN
slightly higher level representation
raw input vector representation:
x =
23
19
20
18
x1
x2
x3
xn

Example 2

Why are Deep Architectures hard to train?

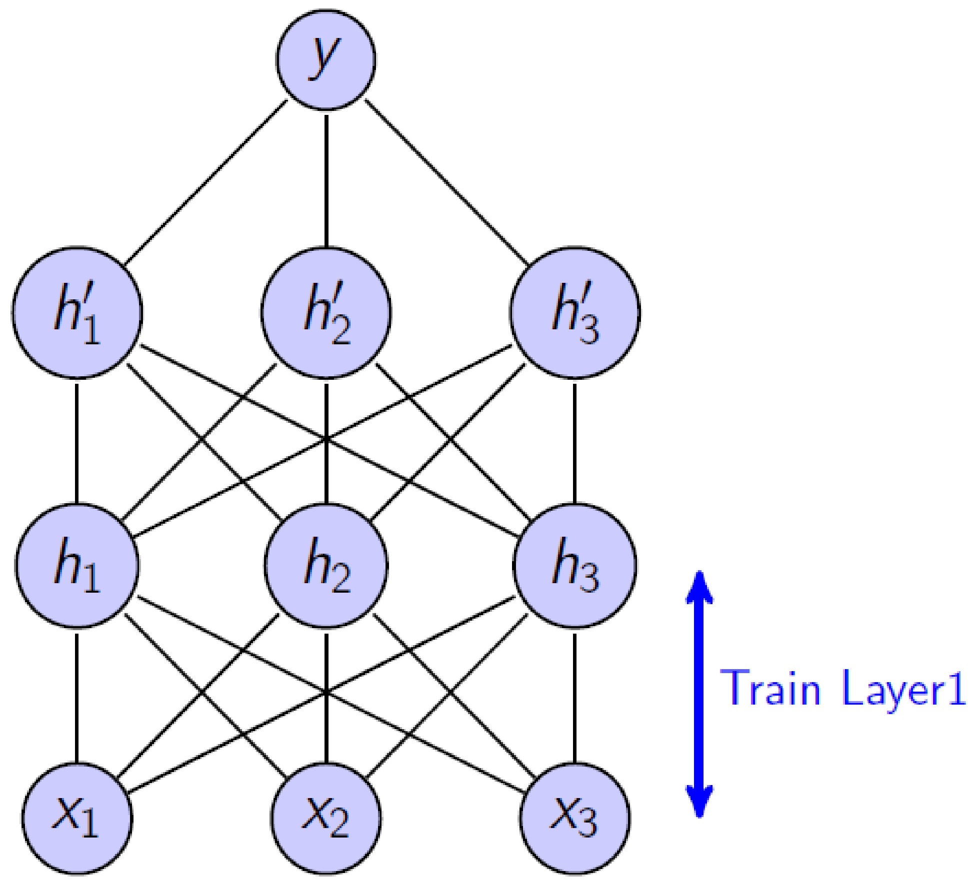
- Vanishing gradient problem in Back Propagation

- $\frac{\partial Loss}{\partial w_{ij}} = \frac{\partial Loss}{\partial in_j} \frac{\partial in_j}{\partial w_{ij}} = \delta_j x_i$
- $\delta_j = \left[\sum_{j+1} \delta_{j+1} w_{j(j+1)}\right] \sigma'(in_j)$
- δ_j may vanish after repeated multiplication



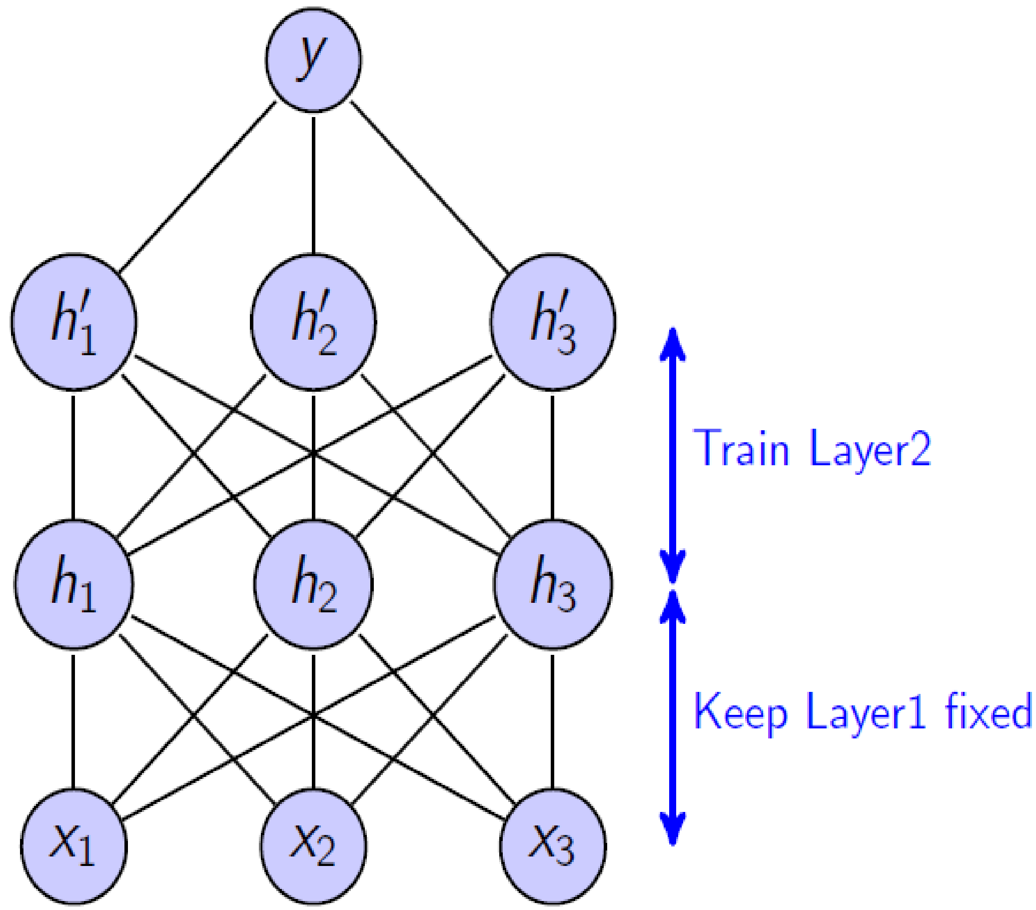
Layer-wise Pre-training

- First, train one layer at a time, optimizing data-likelihood objective P(x)



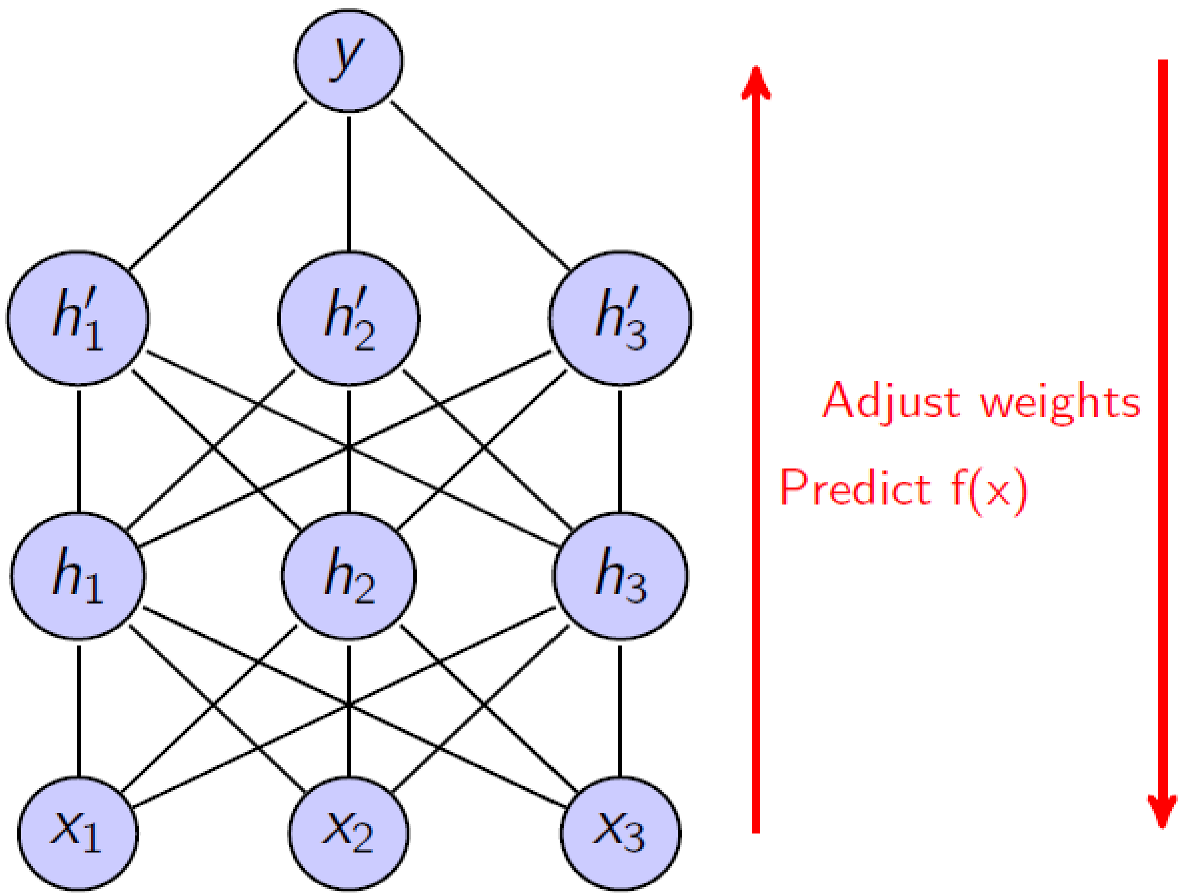
Layer-wise Pre-training

- Then, train second layer next, optimizing data-likelihood objective P(h)



Layer-wise Pre-training

- Finally, fine-tune labelled objective P(y|x) by Backpropagation



Deep Belief Nets

- Uses Restricted Boltzmann Machines (RBMs)
- Hinton et al. (2006), *A fast learning algorithm for deep belief nets*.

Restricted Boltzmann Machine (RBM)

- RBM is a simple energy-based model:

$$p(x, h) = \frac{1}{Z_\theta} \exp\left(-E_\theta(x, h)\right)$$

where

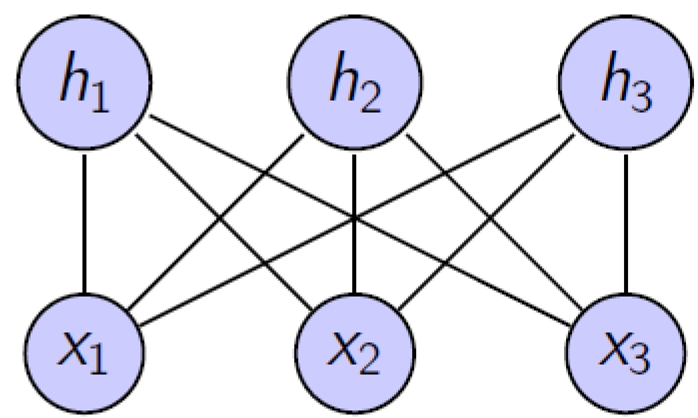
$$E_\theta(x, h) = -x^T Wh - b^T x - d^T h$$

$$Z_\theta = \sum_{(x,h)} \exp(-E_\theta(x, h))$$

Example:

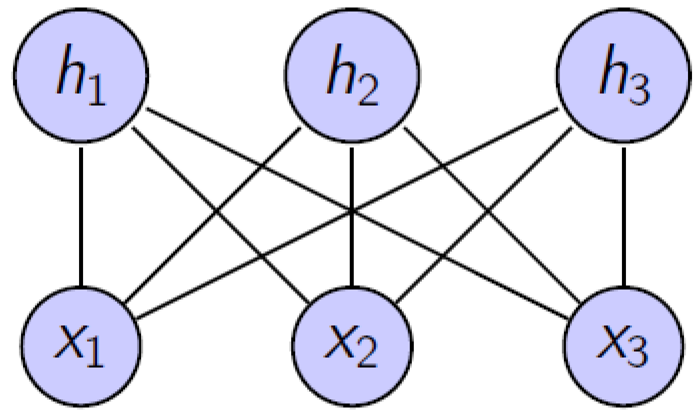
- Let weights (h_1; x_1), (h_1; x_3) be positive, others be zero, b = d = 0.
- Calculate $p(x,h)$?

- Ans: p(x1 = 1; x2 = 0; x3 = 1; h1 = 1; h2 = 0; h3 = 0)



Restricted Boltzmann Machine (RBM)

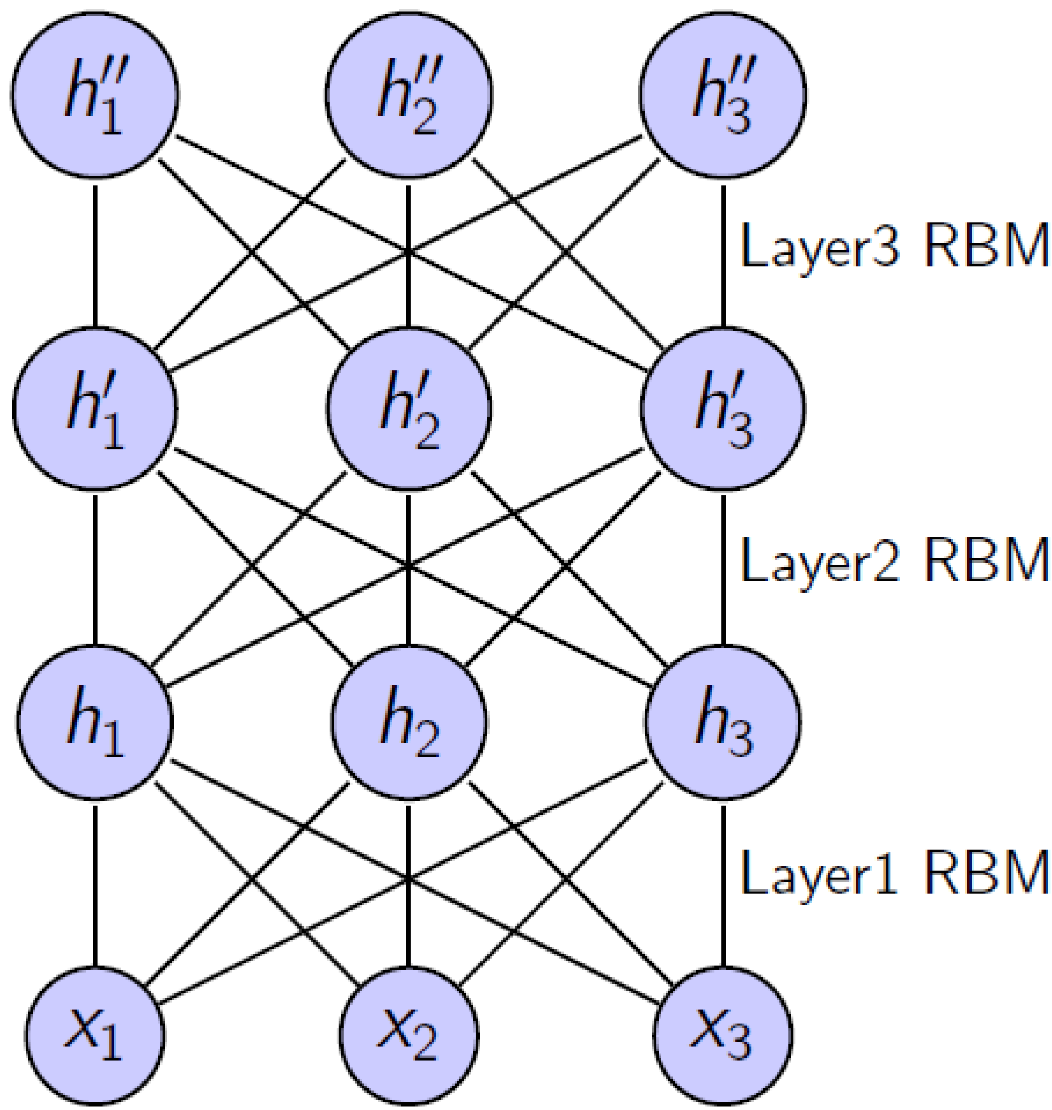
- *P(x, h) = P(h|x) P(x)*
- *P(h|x): easy to compute*
- *P(x): hard if datasets are larg*



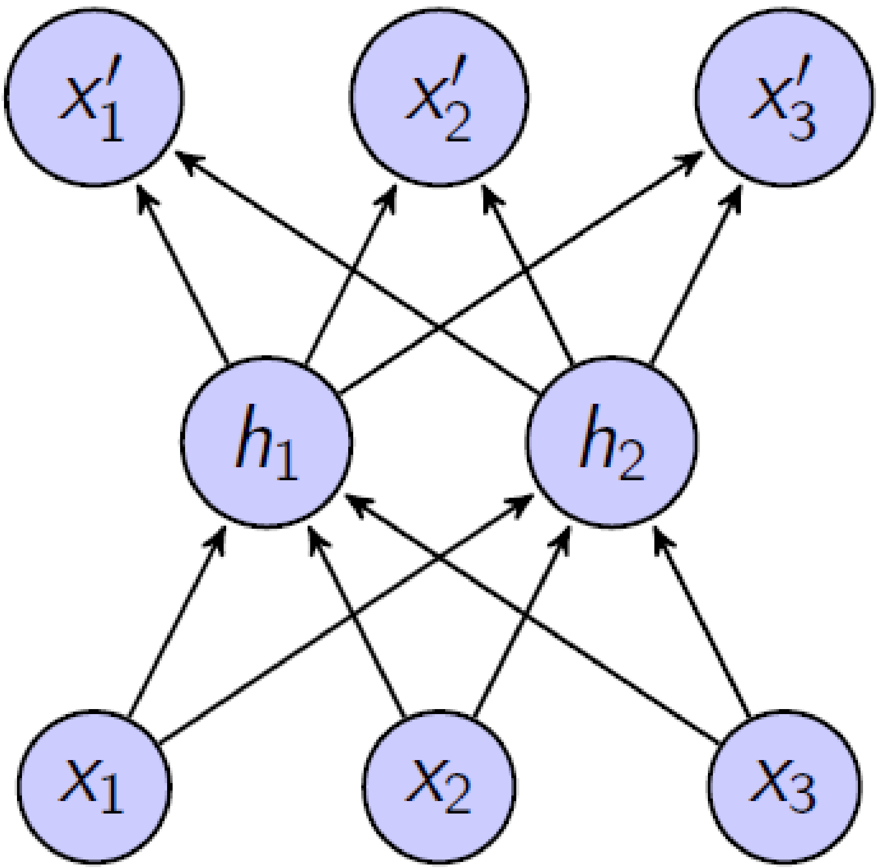
Contrastive Divergence:

1. Let $x^{(m)}$ be training point, $W = [w_{ij}]$ be current model weights
2. Sample $\hat{h}_j \in \{0, 1\}$ from $p(h_j | x = x^{(m)}) = \sigma(\sum_i w_{ij} x_i^{(m)} + d_j)\ \forall j$.
3. Sample $\tilde{x}_i \in \{0, 1\}$ from $p(x_i | h = \hat{h}) = \sigma(\sum_j w_{ij} \hat{h}_j + b_i)\ \forall i$.
4. Sample $\tilde{h}_j \in \{0, 1\}$ from $p(h_j | x = \tilde{x}) = \sigma(\sum_i w_{ij} \tilde{x}_i + d_j)\ \forall j$.
5. $w_{ij} \leftarrow w_{ij} + \gamma(x_i^{(m)} \cdot \hat{h}_j - \tilde{x}_i \cdot \tilde{h}_j)$

Deep Belief Nets (DBN) = Stacked RBM



Auto-Encoders: Simpler alternative to RBMs



Decoder: $x' = \sigma(W'h + d)$

Encoder: $h = \sigma(Wx + b)$

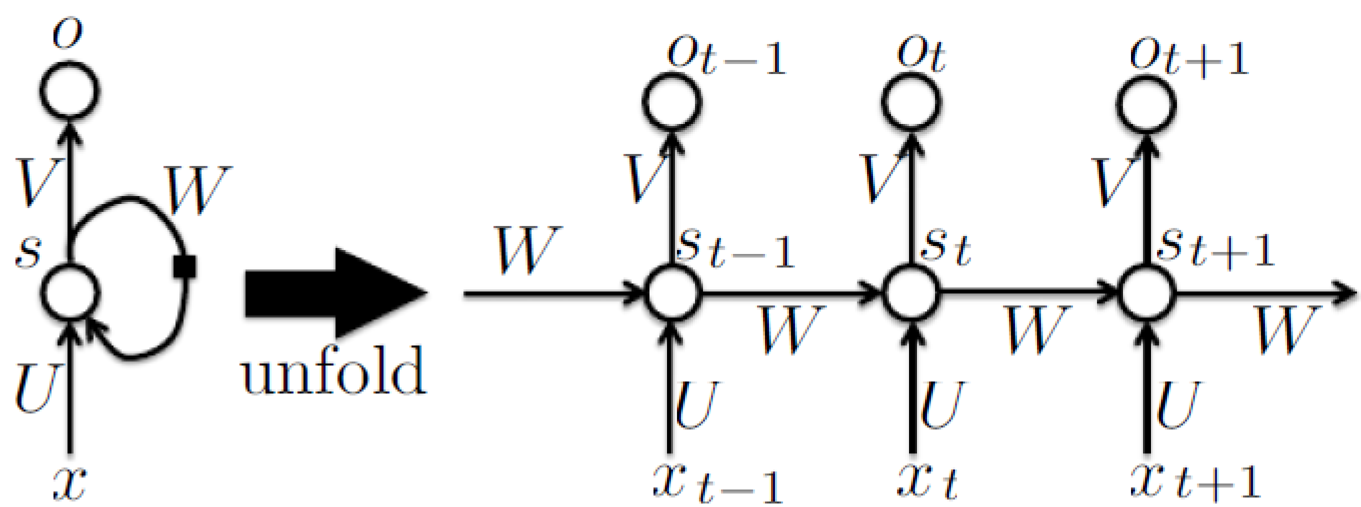
Deep Learning - Architecture

- Recurrent Neural Network (RNN)
- Convolution Neural Network (CNN)

Recurrent Neural Network (RNN)

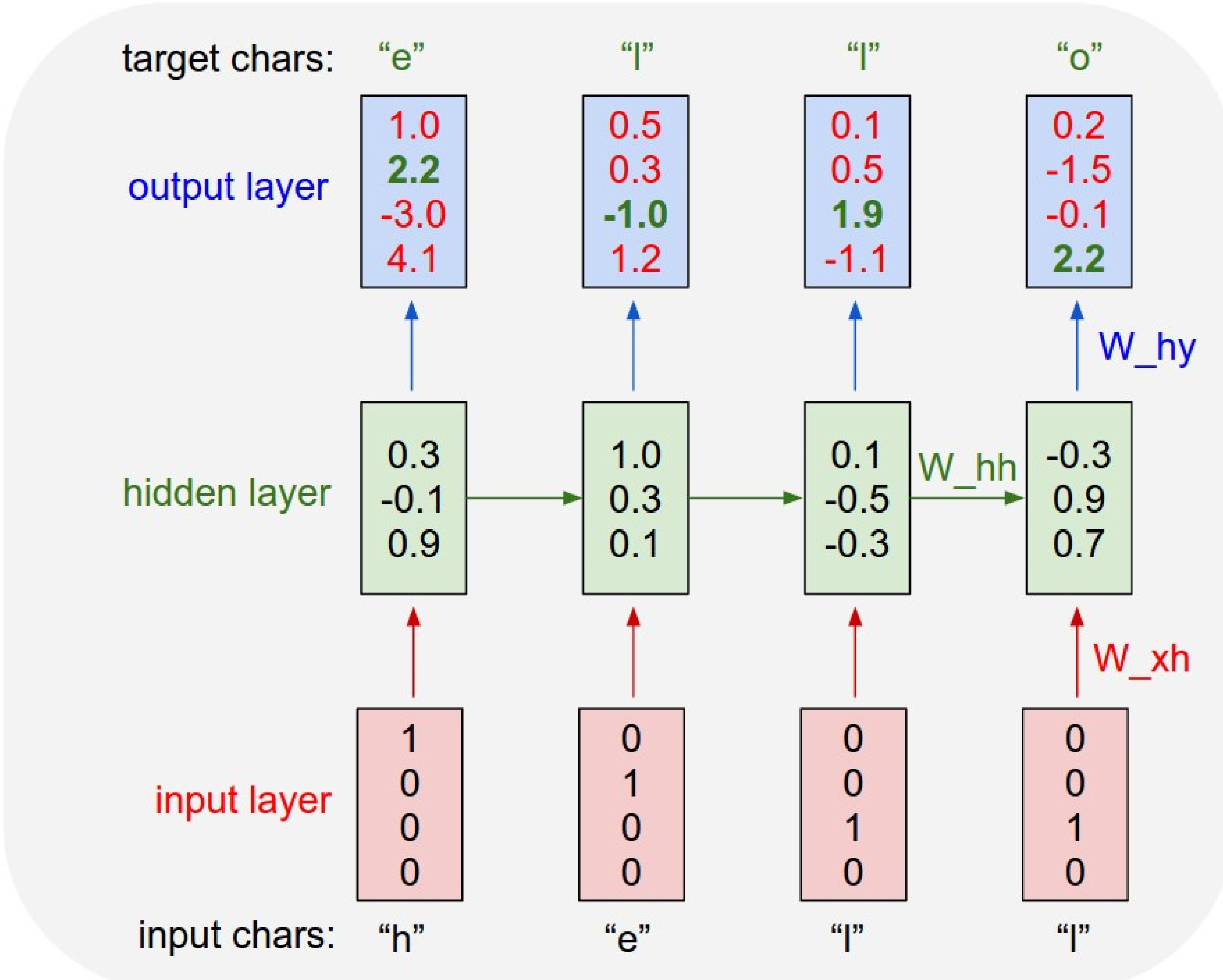
Recurrent Neural Network (RNN)

- Enable networks to do temporal processing and learn sequences



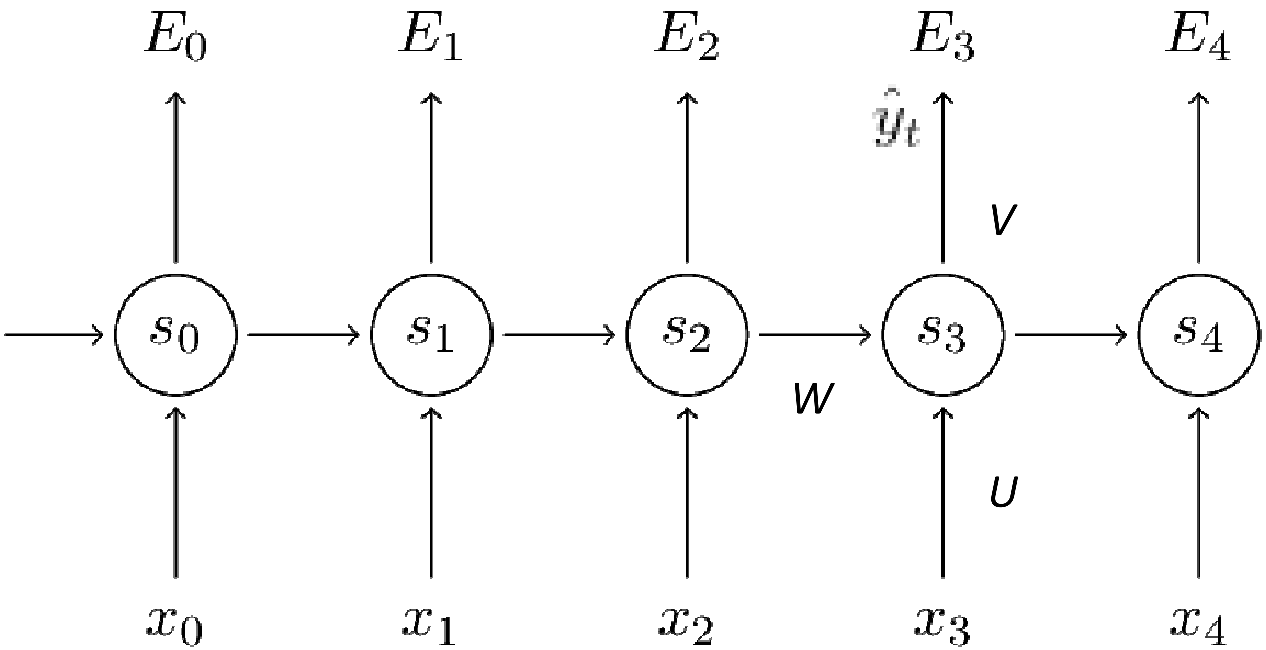
$$
\begin{aligned}
\boldsymbol{a}_t &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{s}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t \\
\boldsymbol{s}_t &= \tanh(\boldsymbol{a}_t) \\
\boldsymbol{o}_t &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{s}_t \\
\boldsymbol{p}_t &= \mathrm{softmax}(\boldsymbol{o}_t)
\end{aligned}
$$

Character level language model **Vocabulary: [h,e,l,o]**



.

Training of RNN: BPTT



$\hat{y}_t$: Predicted

y_t : Actual

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

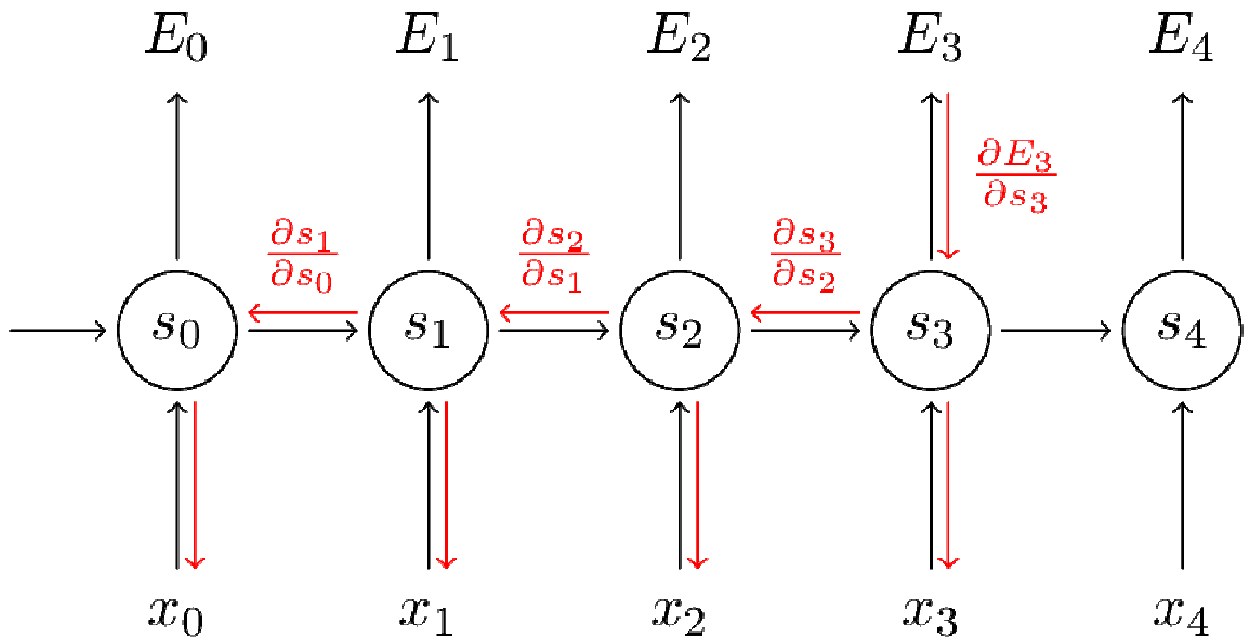
$$\begin{aligned}\frac{\partial E_3}{\partial V} &= \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial V} \\ &= \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3}\frac{\partial z_3}{\partial V} \\ &= (\hat{y}_3 - y_3) \otimes s_3\end{aligned}$$

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W}$$

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W}$$

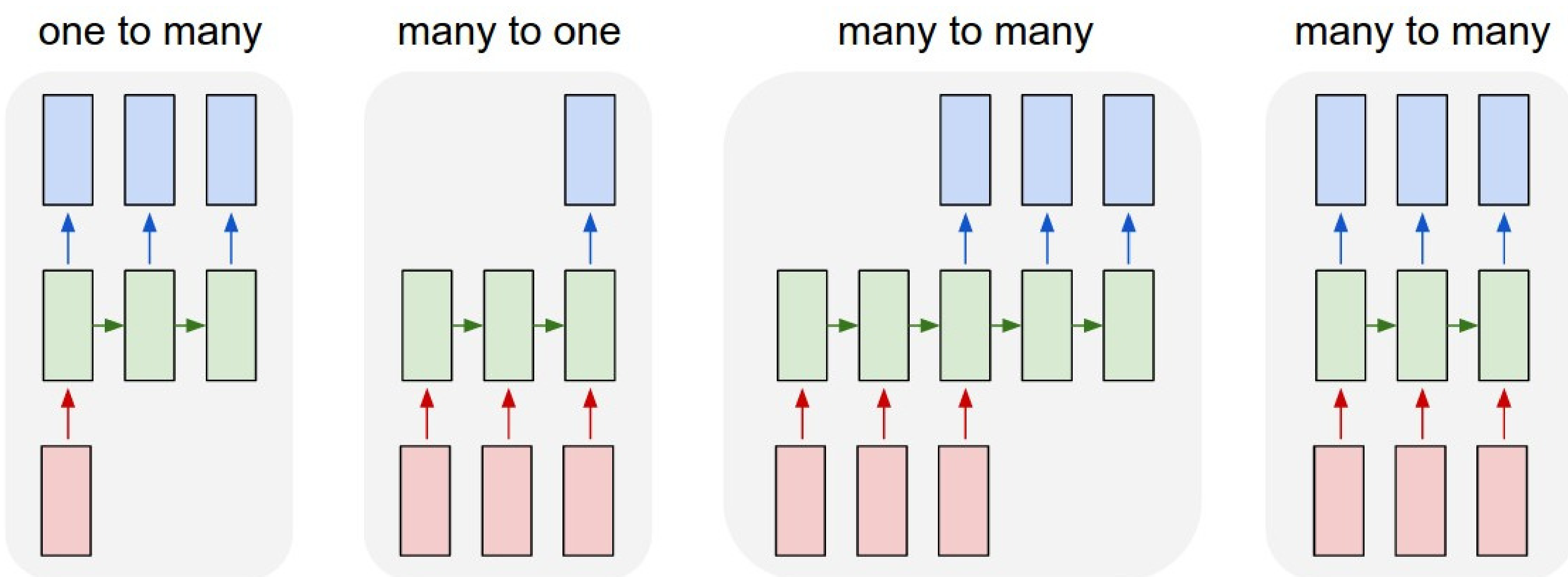
.

Training of RNN: BPTT



$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W}$$

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W}$$



One to many:

Sequence output (e.g. image captioning takes an image and outputs a sentence of words)

Many to one:

Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment)

Many to many:

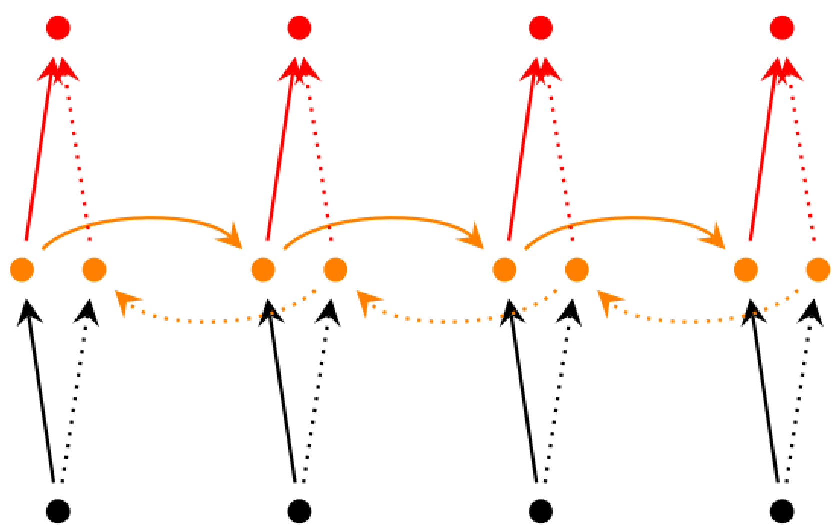
Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French)

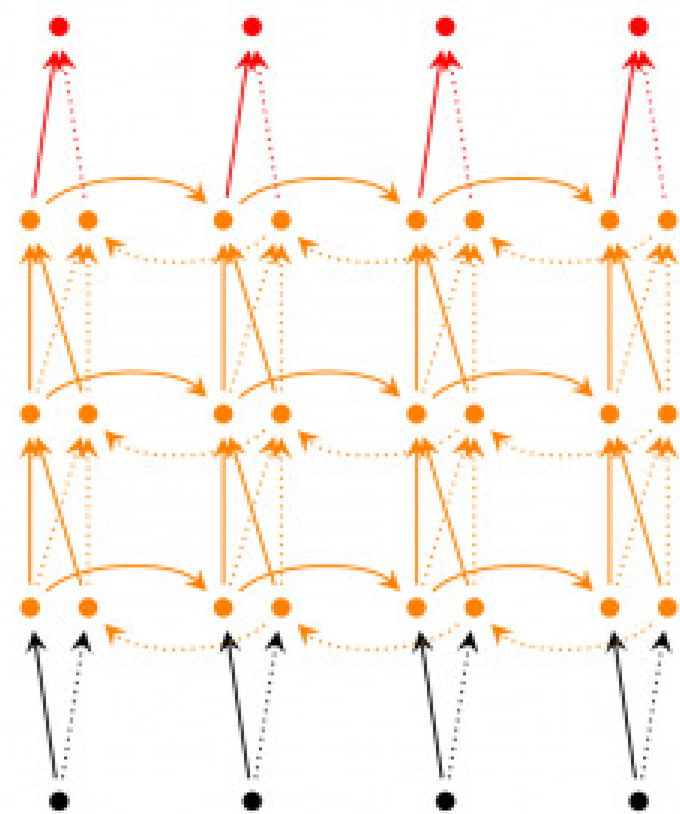
Many to many:

Synced sequence input and output (e.g. Language modelling where we wish to predict next words.

RNN Extension

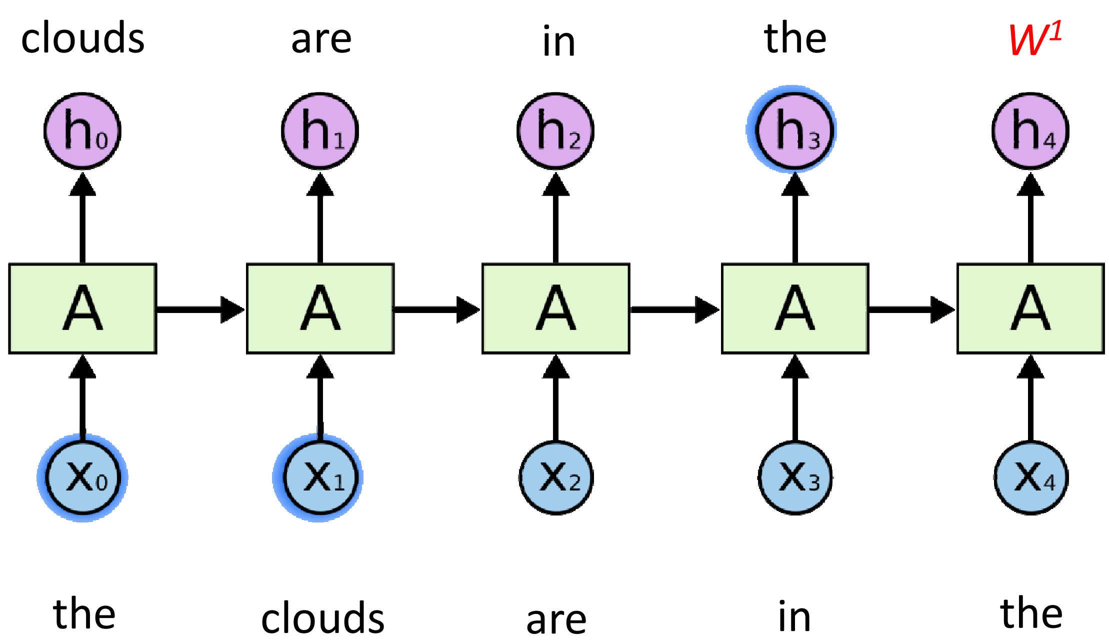
- Bidirectional RNN
- Deep (Bidirectional) RNNs





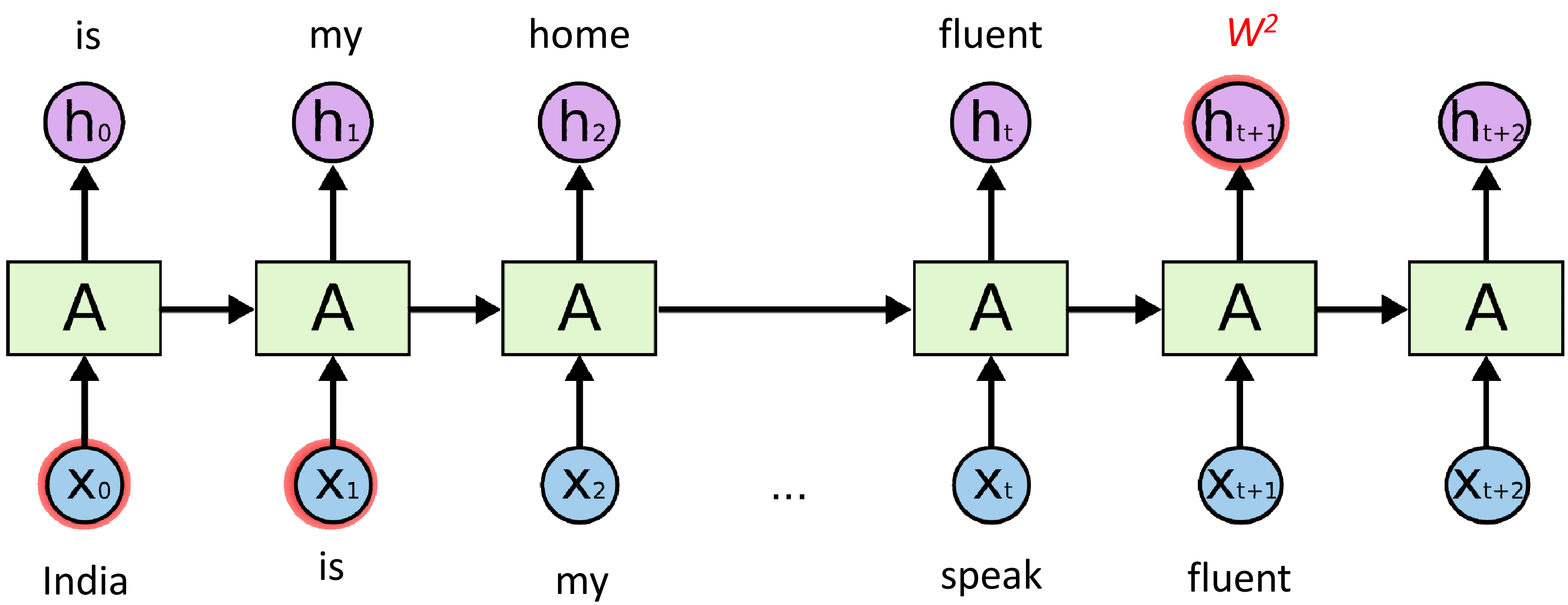
RNN (Cont..)

- "the clouds are in the *sky*"



RNN (Cont..)

- "India is my home country. I can speak fluent *Hindi*."



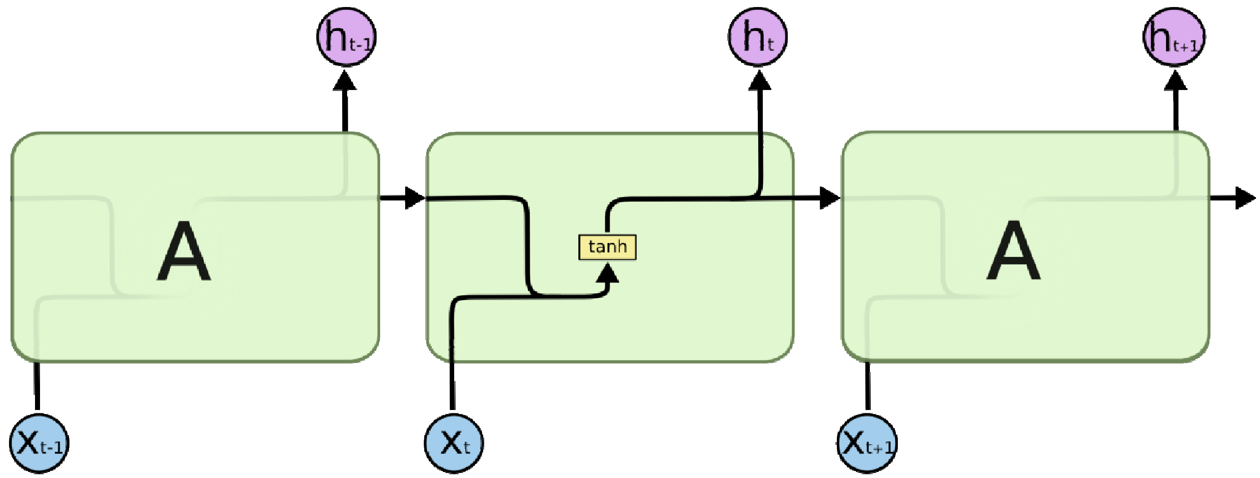
Ideally: It should

Practically: It is very hard for RNN to learn "Long Term Dependency".

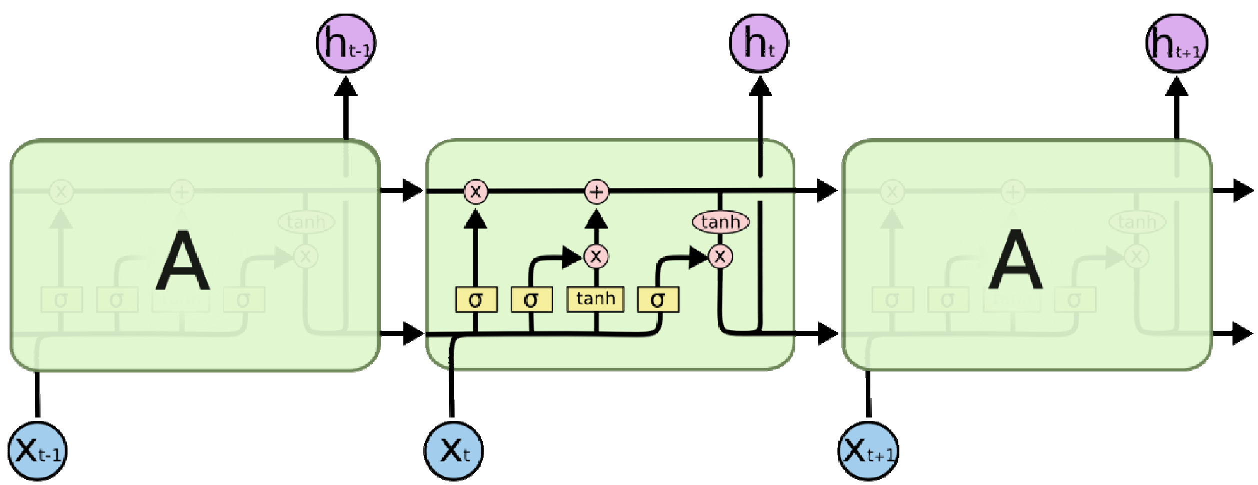
LSTM

- Capable of learning long-term dependencies.

Simple RNN



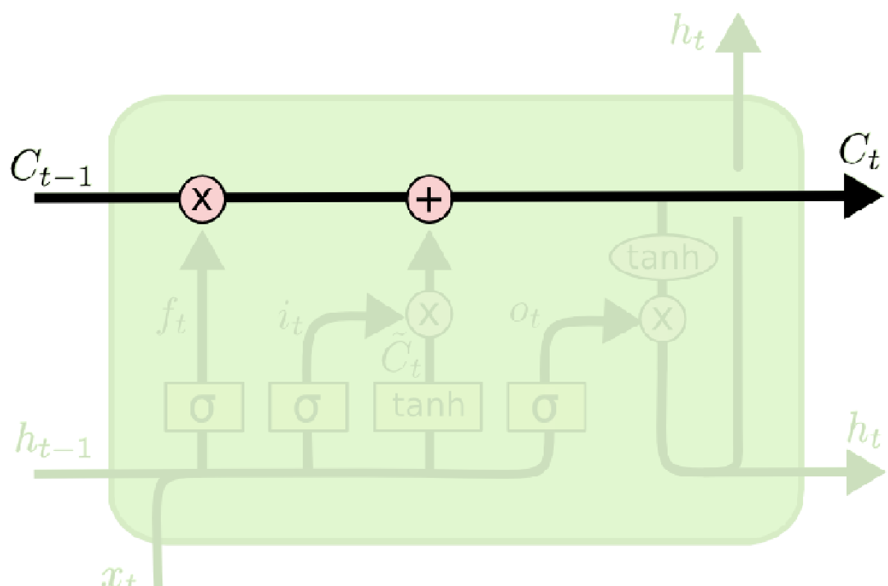
LSTM



LSTM

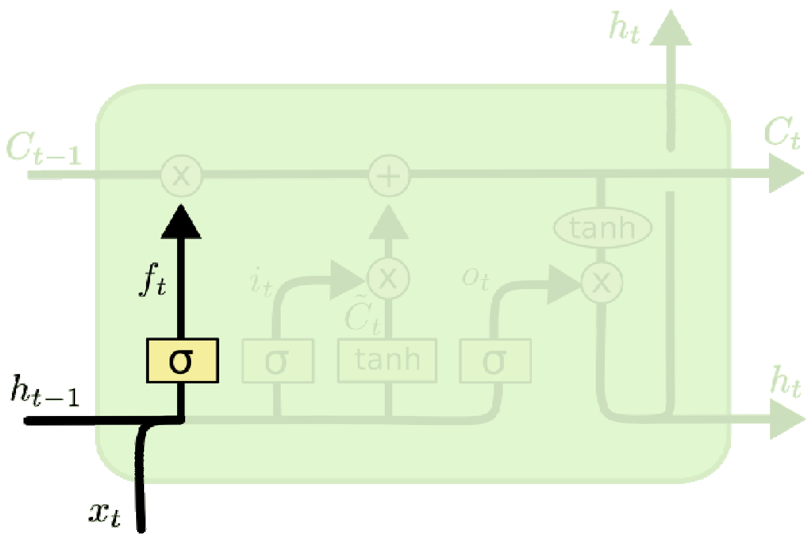
- LSTM remove or add information to the cell state, carefully regulated by structures called gates.

- Cell state: Conveyer belt of the cell



LSTM

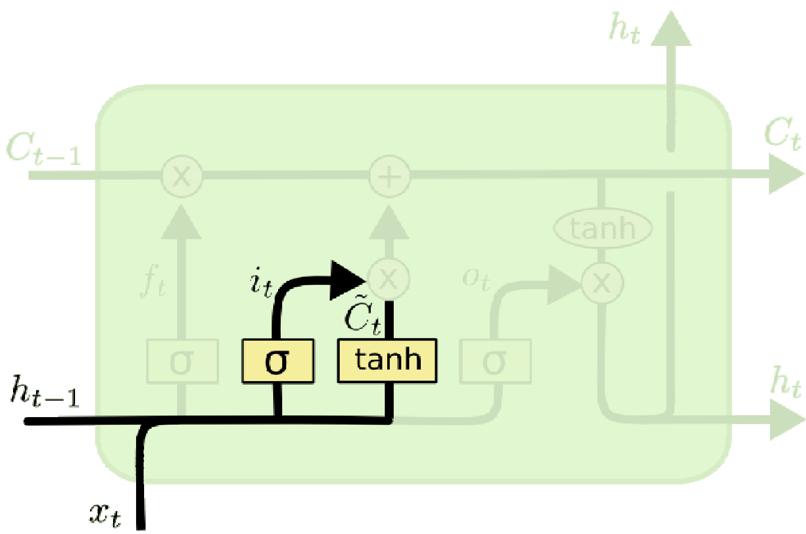
- Gates
  - Forget Gate
  - Input Gate
  - Output Gate



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

LSTM

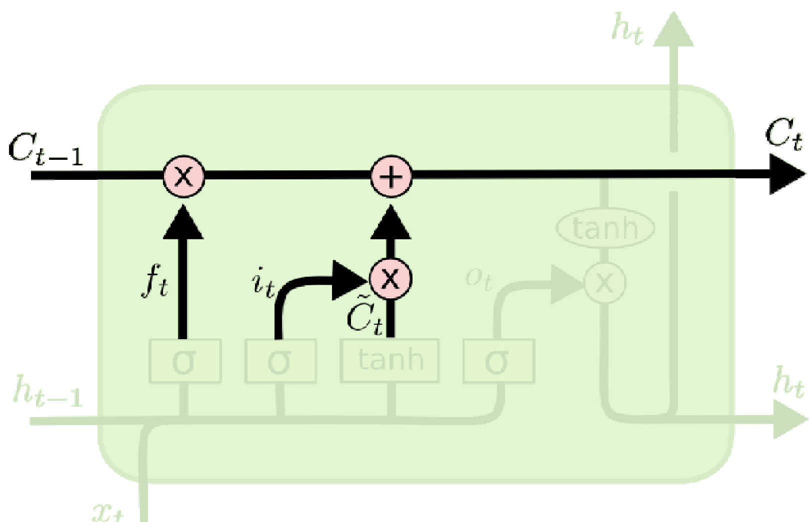
- Gates
  - Forget Gate
  - Input Gate
  - Output Gate



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

LSTM

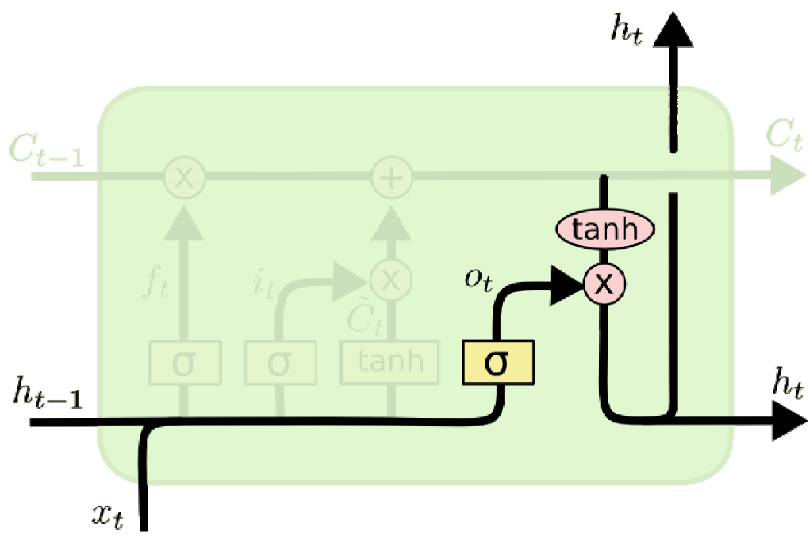
- Gates
  - Forget Gate
  - Input Gate
  - Output Gate



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

LSTM

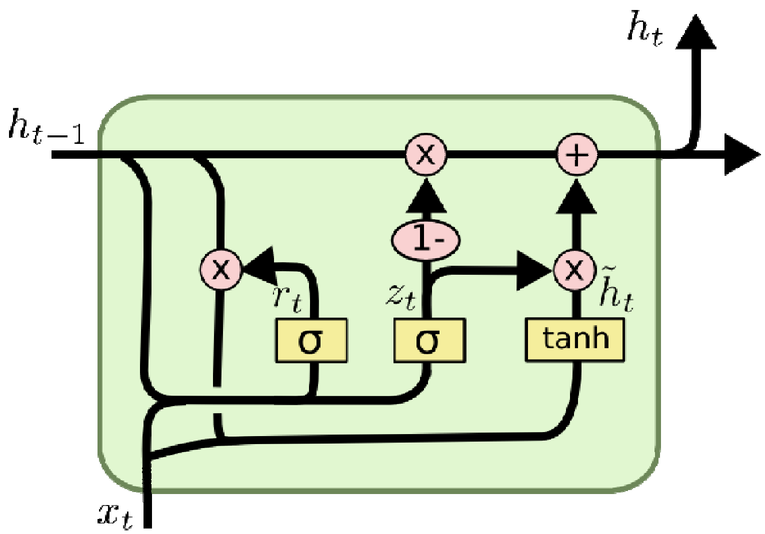
- Gates
  - Forget Gate
  - Input Gate
  - Output Gate



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

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

LSTM- Variants

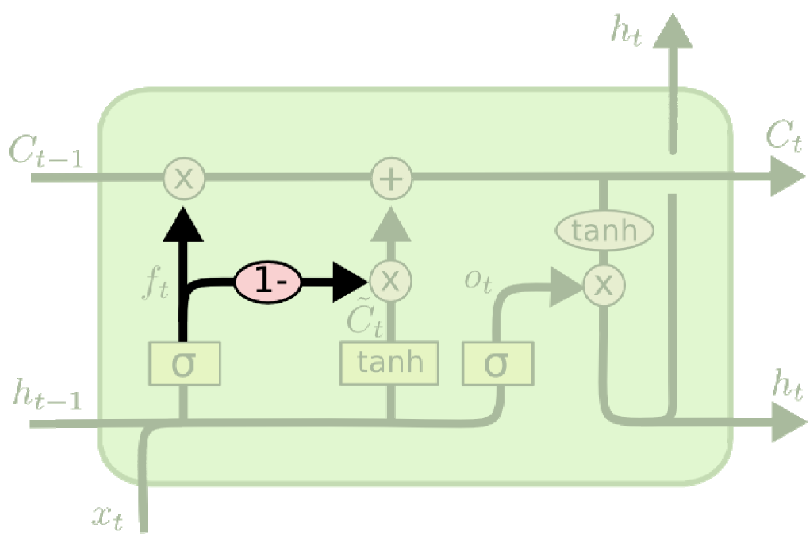


$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$



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

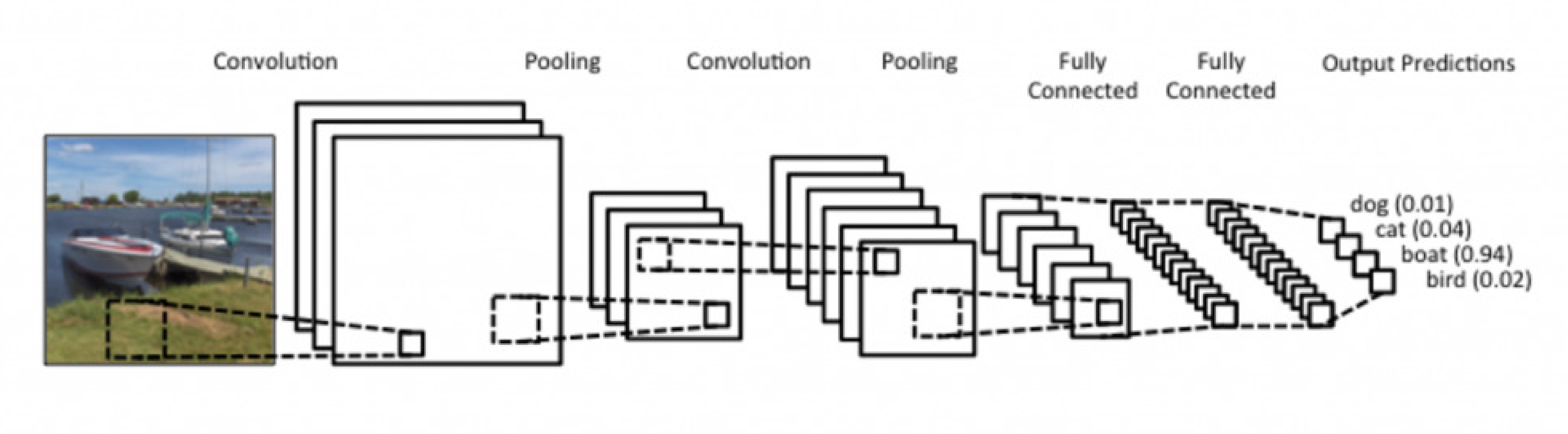
Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN)

- A special kind of multi-layer neural networks.
- Implicitly extract relevant features.
- Fully-connected network architecture does not take into account the spatial structure.
- In contrast, CNN tries to take advantage of the s patial structure.

Convolutional Neural Network (CNN)

1. Convolutional layer
2. Pooling layer
3. Fully connected layer



Convolutional Neural Network (CNN)

1. Convolutional layer

1	0	1
0	1	0
1	0	1

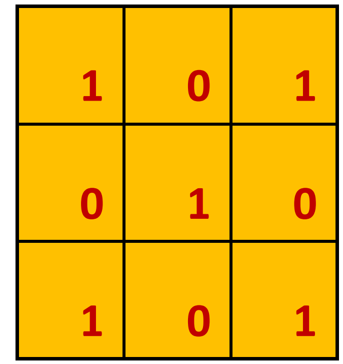
Convolution Filter

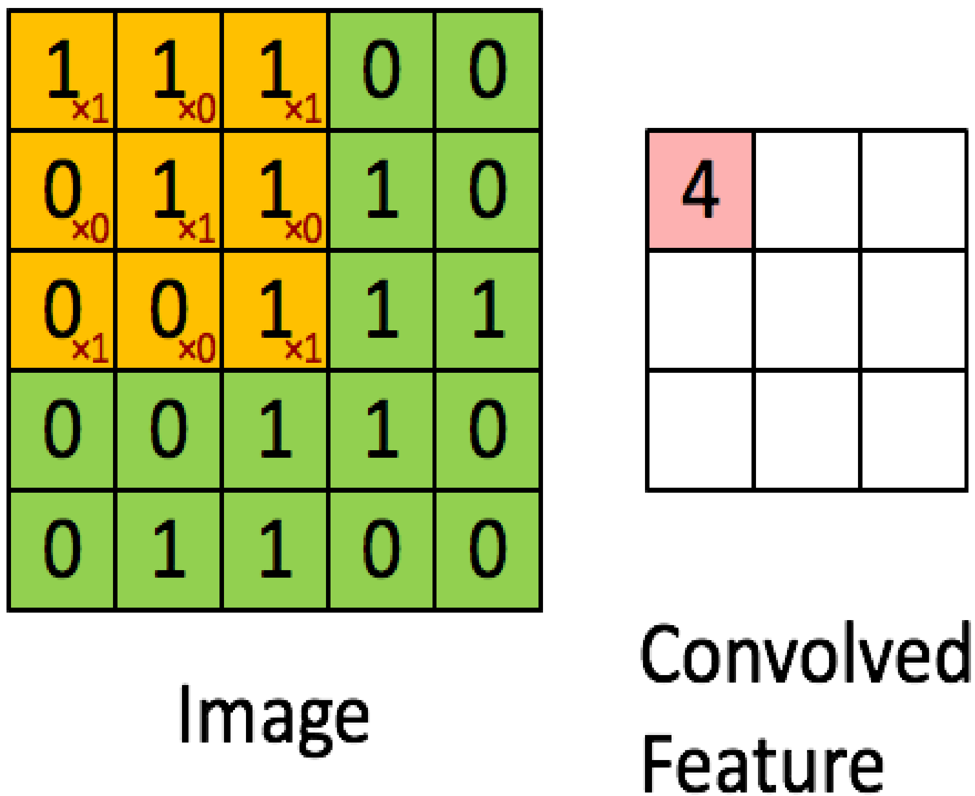
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

Convolutional Neural Network (CNN)

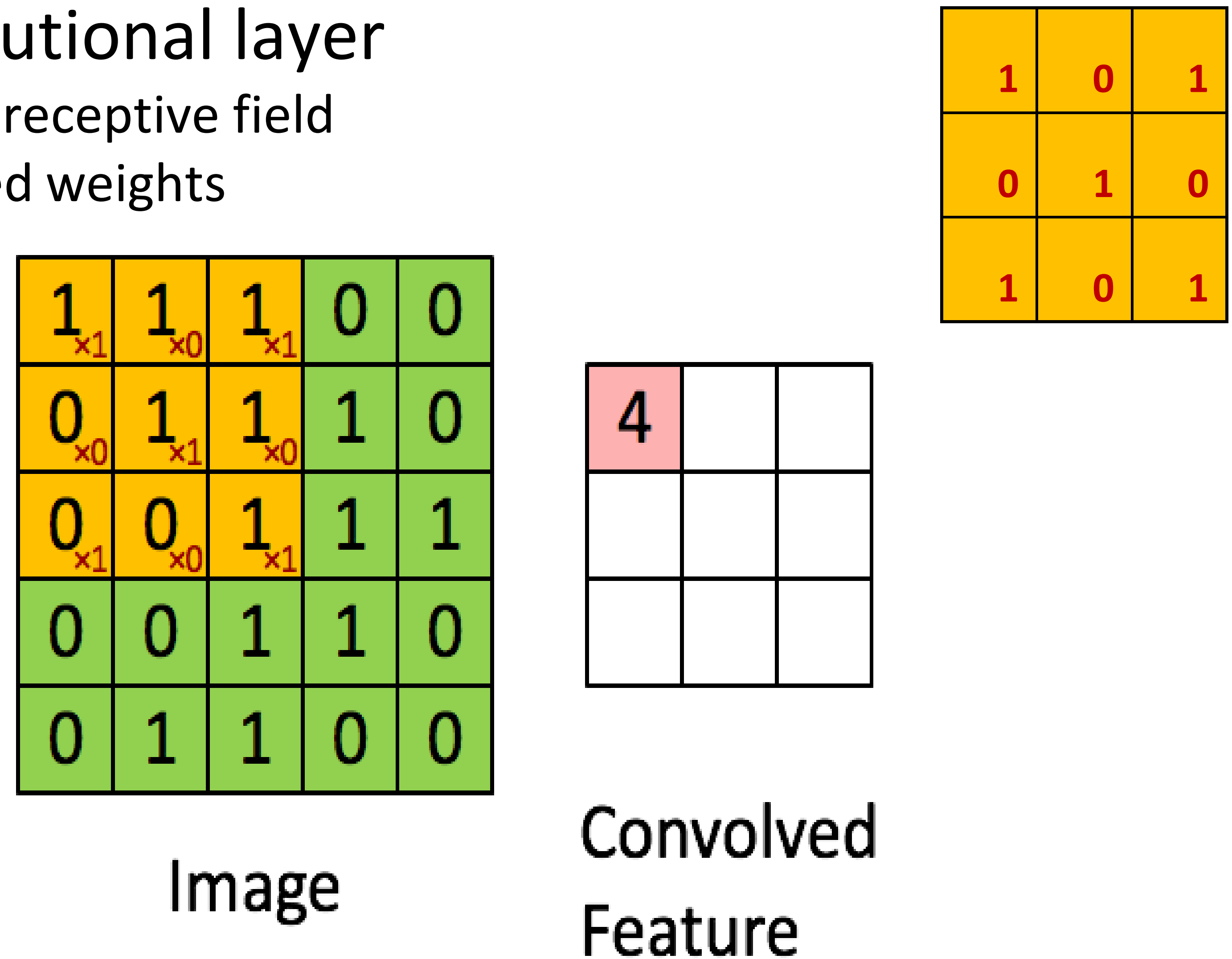
1. Convolutional layer





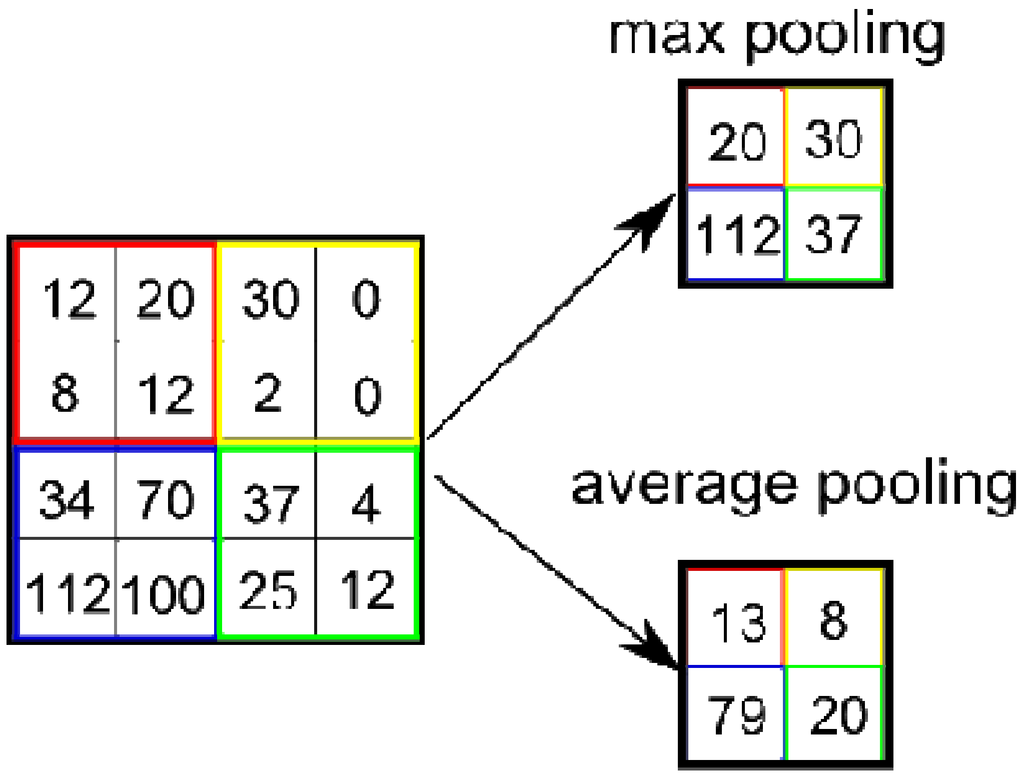
Convolutional Neural Network (CNN)

1. Convolutional layer
   - Local receptive field
   - Shared weights



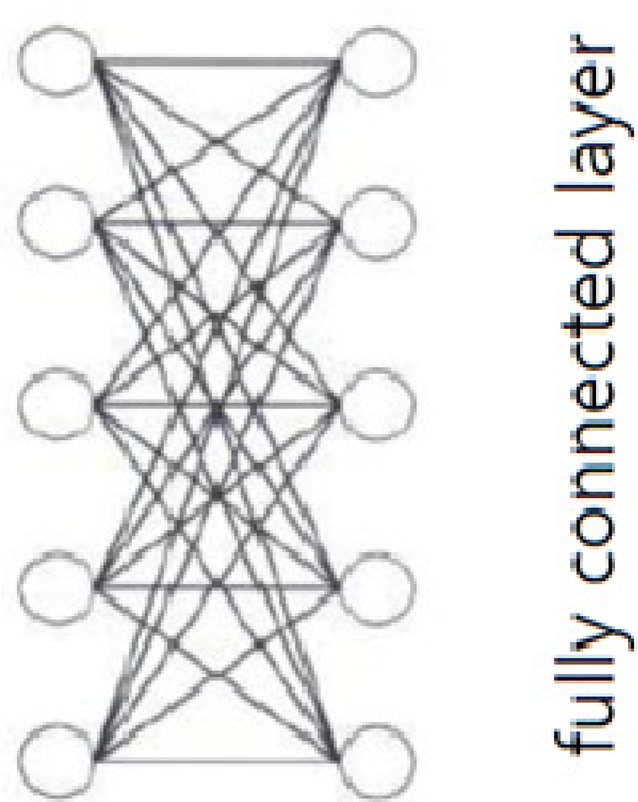
Convolutional Neural Network (CNN)

2. Pooling layer



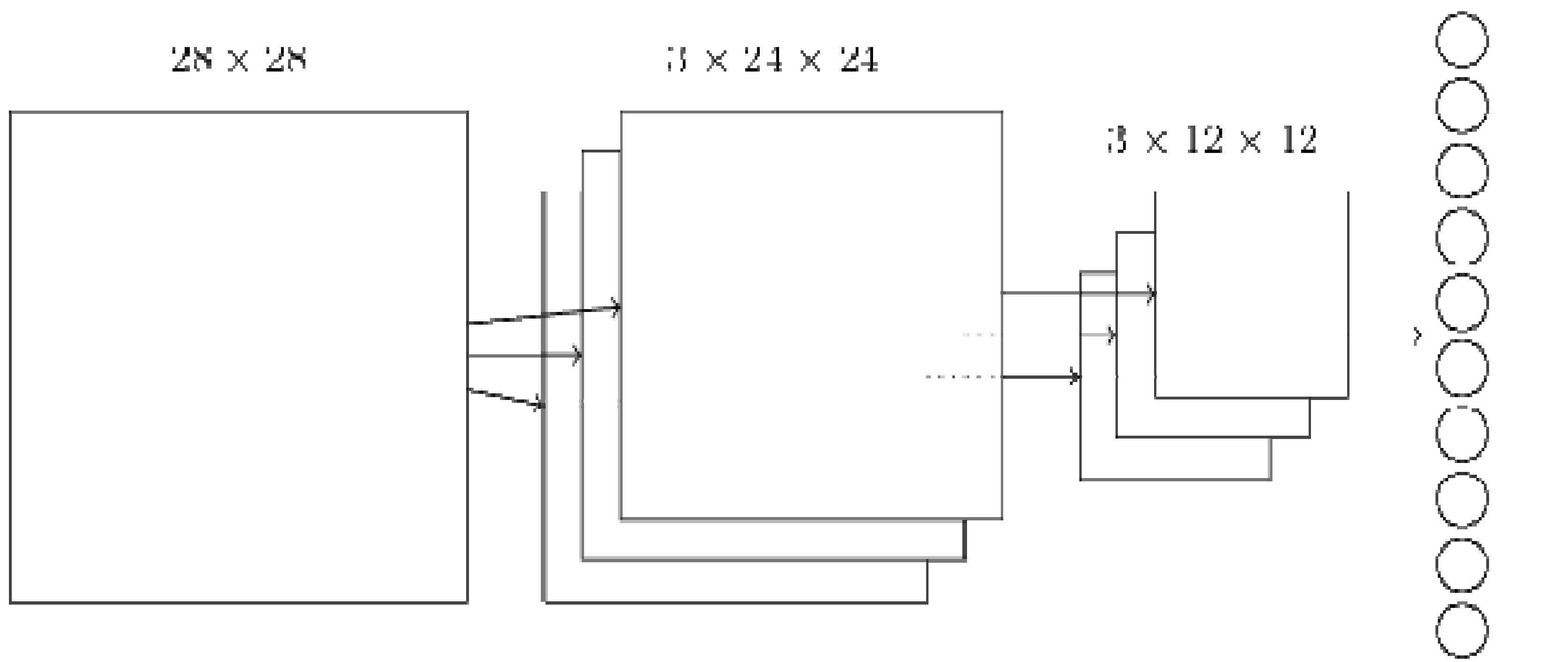
Convolutional Neural Network (CNN)

3. Fully connected layer



Convolutional Neural Network (CNN)

Putting it all together



References

- http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
- http://karpathy.github.io/2015/05/21/rnn-effectiveness/
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- http://cl.naist.jp/~kevinduh/a/deep2014/