Introduction to Neural Networks: DNN / CNN / RNN

AI602: Recent Advances in Deep Learning

Lecture 1

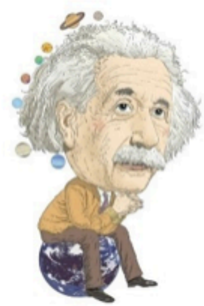
Slide made by

Hyungwon Choi and Yunhun Jang

KAIST EE

What is Machine/Deep Learning?

- Human Learning

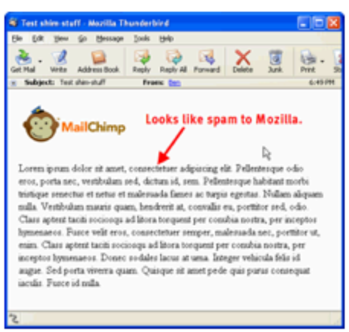


Train using sample exams

Take an exam
(e.g. $E=MC^2$)

Report the exam score

- Machine Learning = Build an algorithm from data
  - Deep learning is a special type of algorithms in machine learning



Big Data

Learning perceptions

Train using sample (question, answer) pairs
"supervised learning"

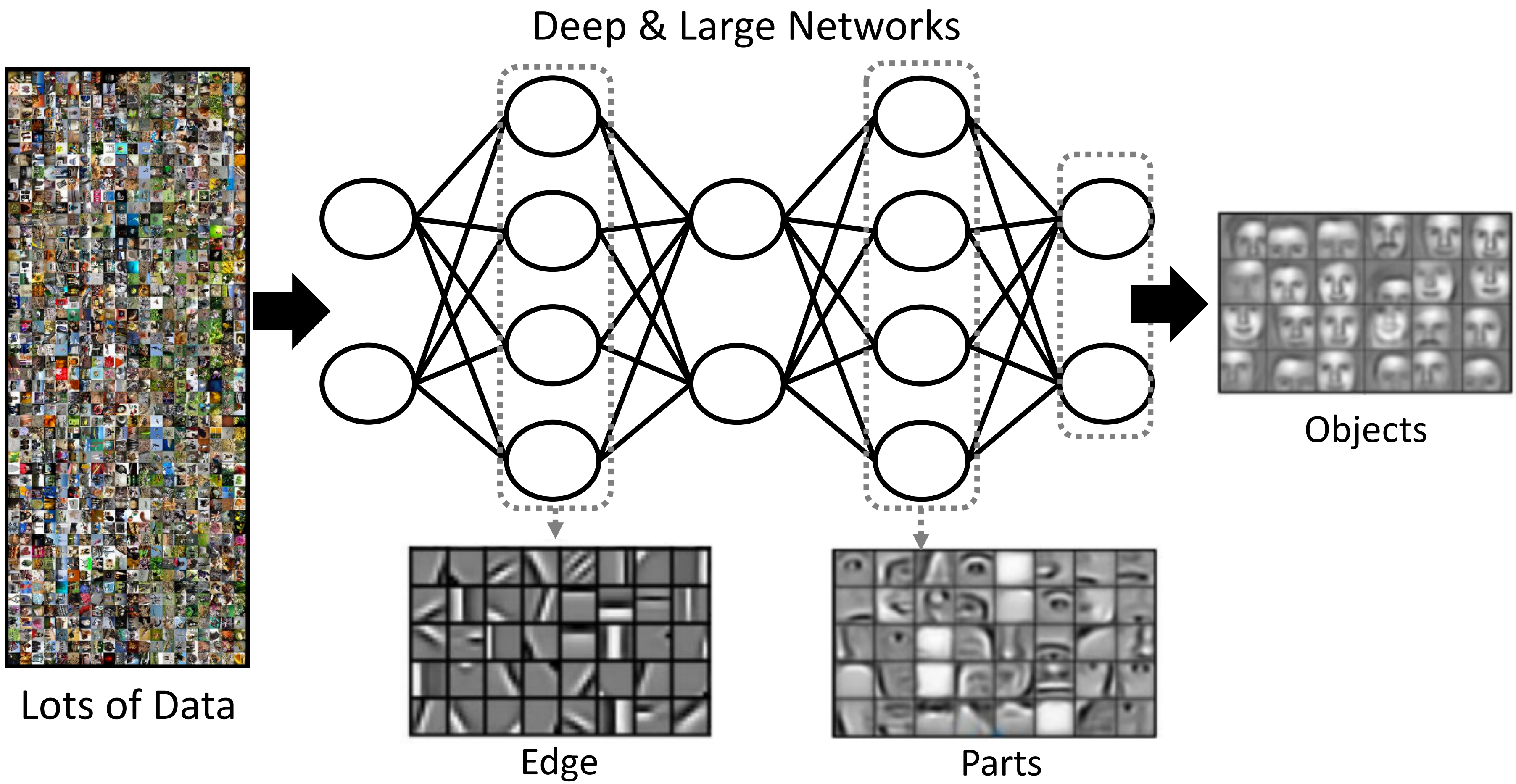
Answer a question
(e.g. spam filtering)

Report how good the answer is
"reinforcement learning"

Learning interactions

Definition of Deep Learning

- An algorithm that learns multiple levels of abstractions in data

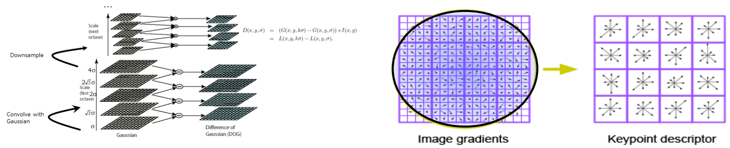


Multi-layer Data Representations (feature hierarchy)

Deep Learning = Feature Learning

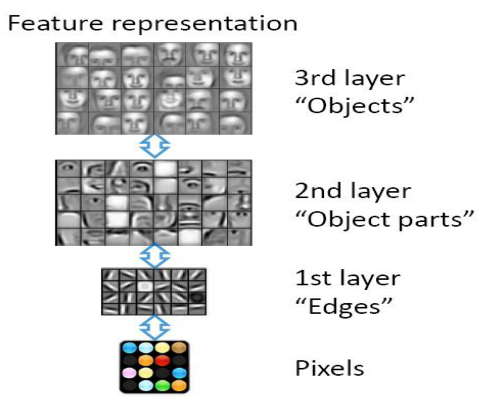
- Why deep learning outperforms other machine learning (ML) approaches for vision, speech, language?

Input → Feature Extraction → Other ML → Output



SIFT

Input → Deep Network → Output



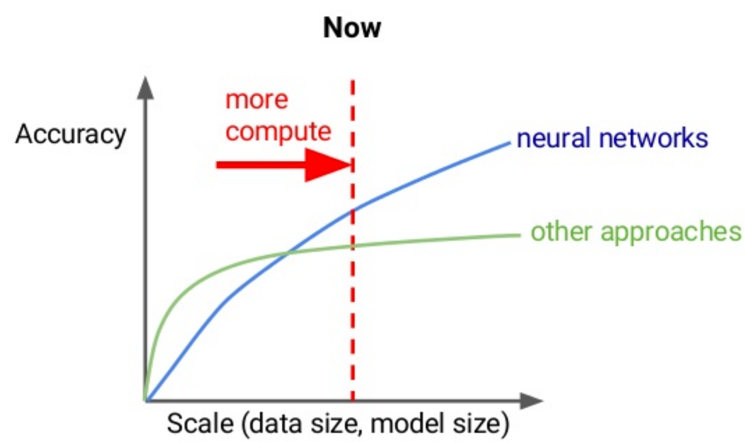
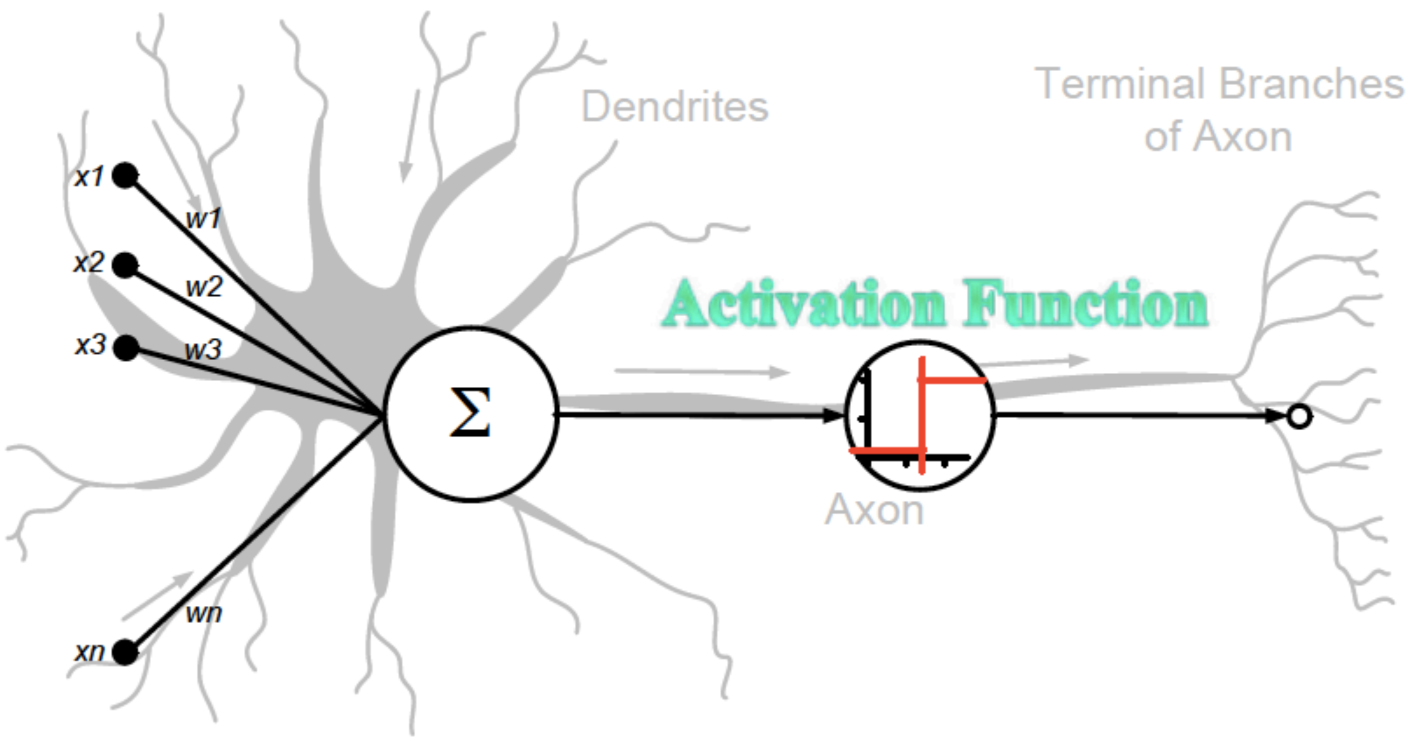


Table of Contents

Table of Contents

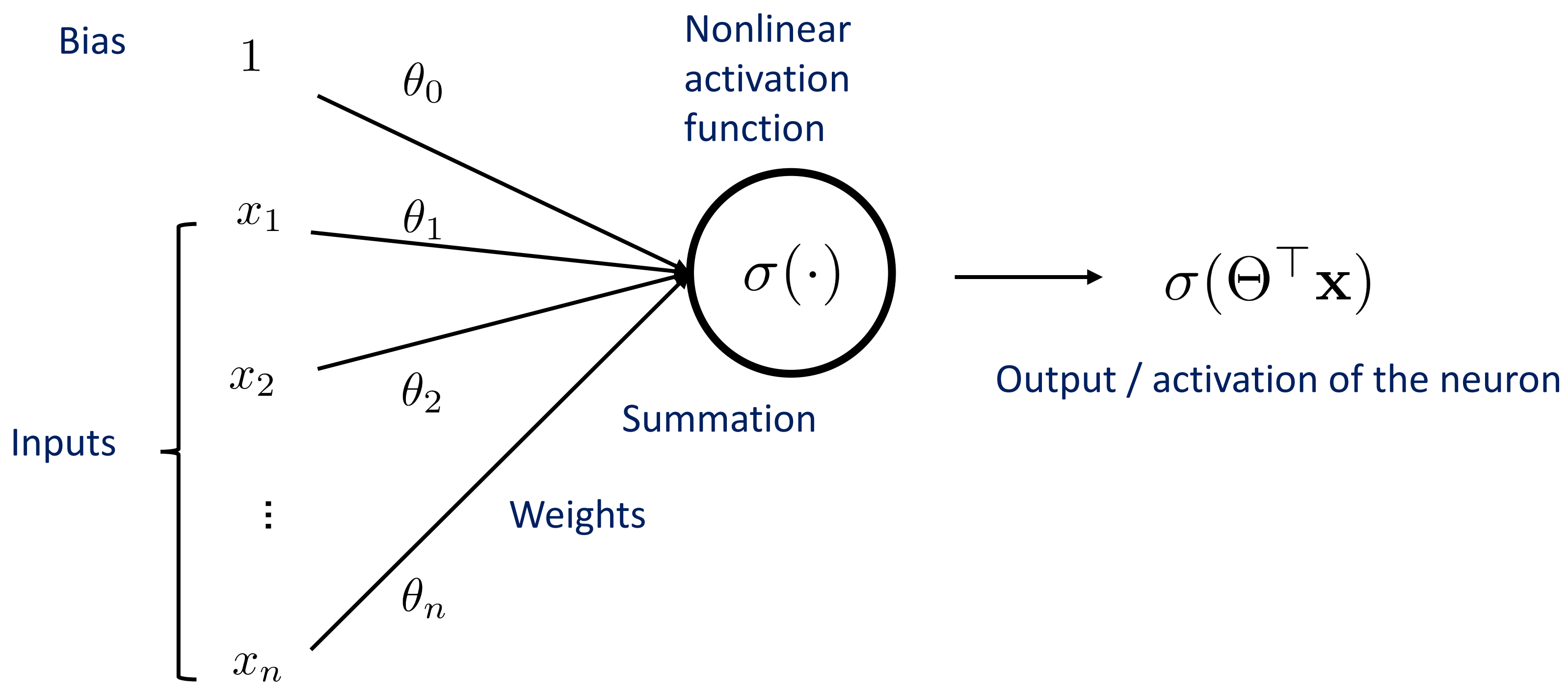
DNN: Neurons in the Brain

- Human brain is made up of 100 billion **neurons**
  - Neurons **receive** electric signals at the dendrites and **send** them to the axon
  - Dendrites can perform complex **non-linear** computations
  - Synapses are not a single weight but a **complex** non-linear dynamical system



*source : https://pt.slideshare.net/hammawan/deep-neural-networks

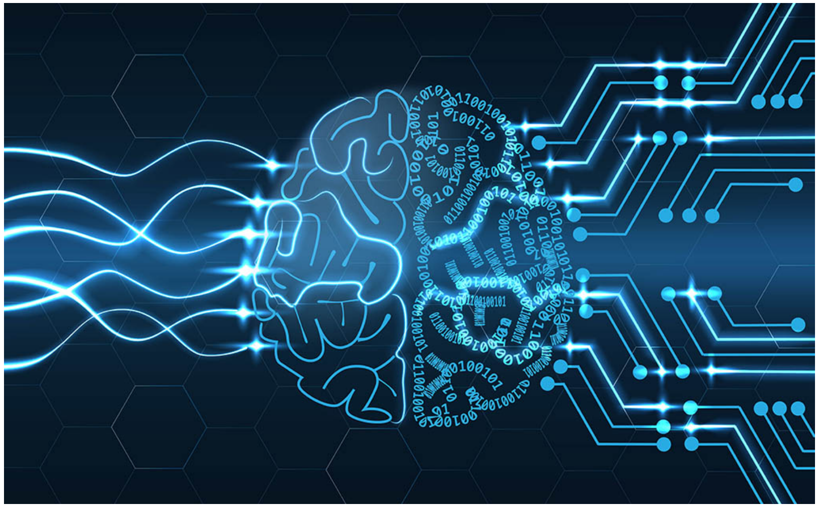
- **Artificial neural networks**
  - A **simplified** version of biological neural network



DNN: The Brain vs. Artificial Neural Networks

- **Similarities**
  - Consists of neurons & connections between neurons
  - Learning process = Update of **connections**
  - Massive **parallel** processing

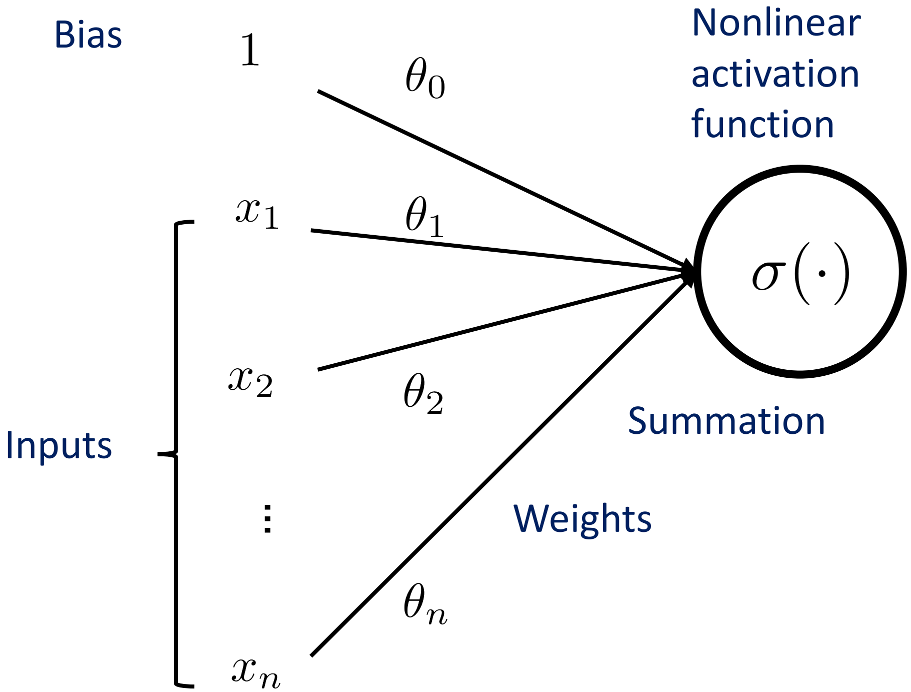
- **Differences**
  - Computation within neuron vastly **simplified**
  - **Discrete** time steps
  - Typically some of **supervised** learning with massive number of stimuli

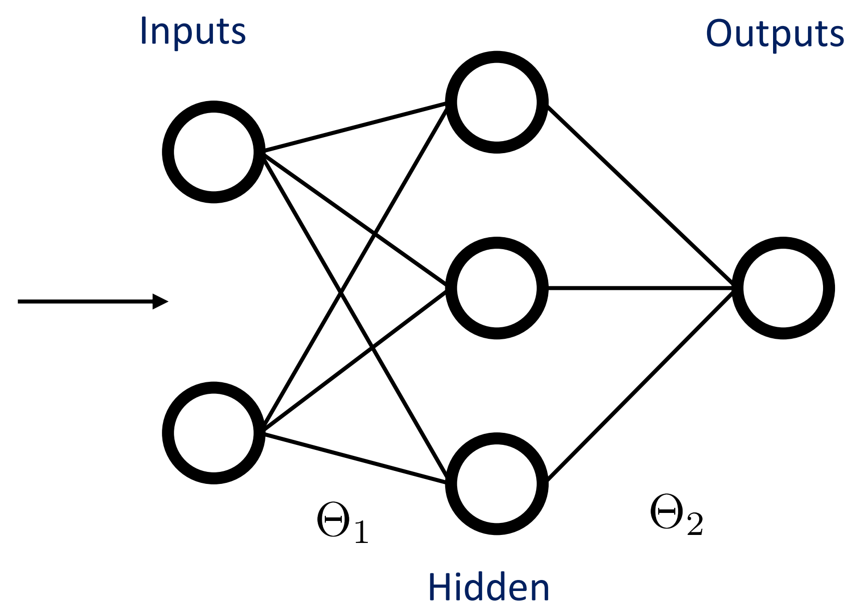


*source : http://mt-class.org/jhu/slides/lecture-nn-intro.pdf

DNN: Basics

- **Deep neural networks**
  - Neural network with more than 2 layers
  - Can model more **complex** functions



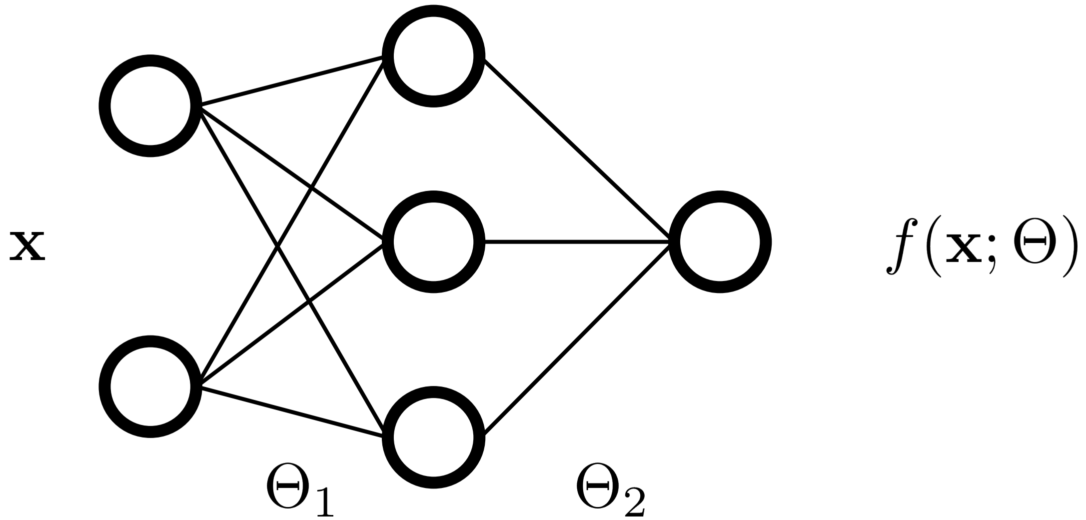


"2-layer Neural Net"
"1-hidden-layer Neural Net"

DNN: Notation

- Training dataset $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$
  - $\mathbf{x}_i$: i^{th} input data
  - y_i : i^{th} target data (or label for classification)

- Neural network $f(\mathbf{x}; \Theta) \in \mathbb{R}$ parameterized by Θ



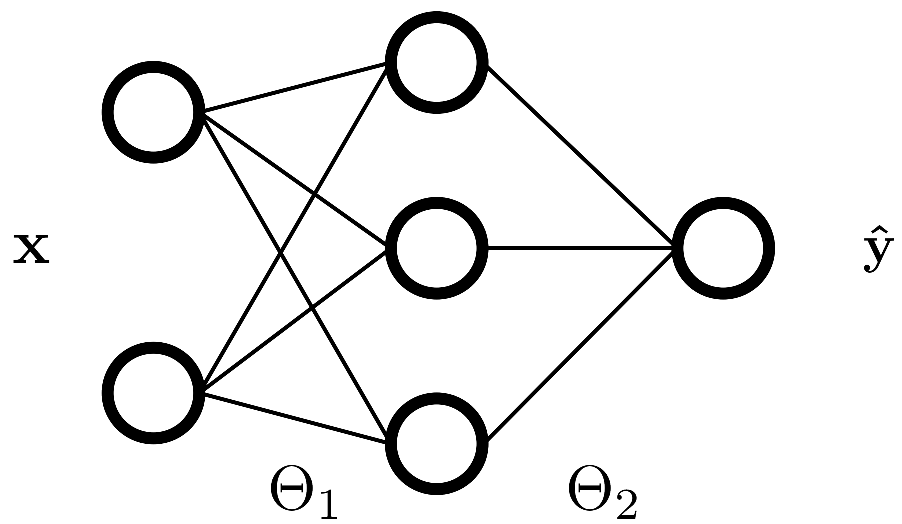
Next, forward propagation

DNN: Forward Propagation

- **Forward propagation**: calculate the output $\hat{\mathbf{y}}$ of the neural network

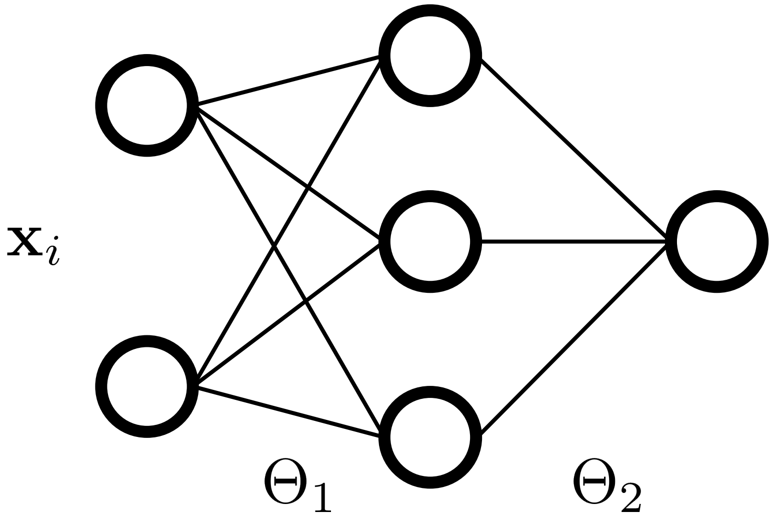
$$\hat{\mathbf{y}} = \sigma\left(\Theta_k^\top \sigma\left(\Theta_{k-1}^\top \sigma(\cdots \sigma(\Theta_1^\top \mathbf{x}))\right)\right)$$

where $\sigma(\cdot)$ is activation function (e.g., sigmoid function) and k is number of layers



DNN: Forward Propagation (Example)

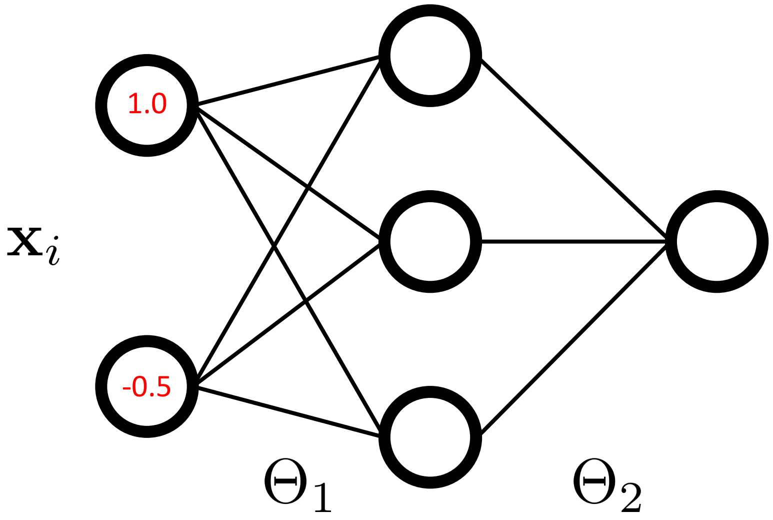
$$\mathbf{x}_i = \begin{pmatrix} 1.0 \\ -0.5 \end{pmatrix} \qquad \Theta_1 = \begin{pmatrix} 1.2 & 2.1 & 1.5 \\ -0.3 & -0.7 & 0.3 \end{pmatrix} \quad \Theta_2 = \begin{pmatrix} -0.2 \\ 0.5 \\ 1.3 \end{pmatrix}$$



DNN: Forward Propagation (Example)

$$\mathbf{x}_i = \begin{pmatrix} 1.0 \\ -0.5 \end{pmatrix} \qquad \Theta_1 = \begin{pmatrix} 1.2 & 2.1 & 1.5 \\ -0.3 & -0.7 & 0.3 \end{pmatrix} \quad \Theta_2 = \begin{pmatrix} -0.2 \\ 0.5 \\ 1.3 \end{pmatrix}$$

- Input data $\mathbf{x}_i$



DNN: Forward Propagation (Example)

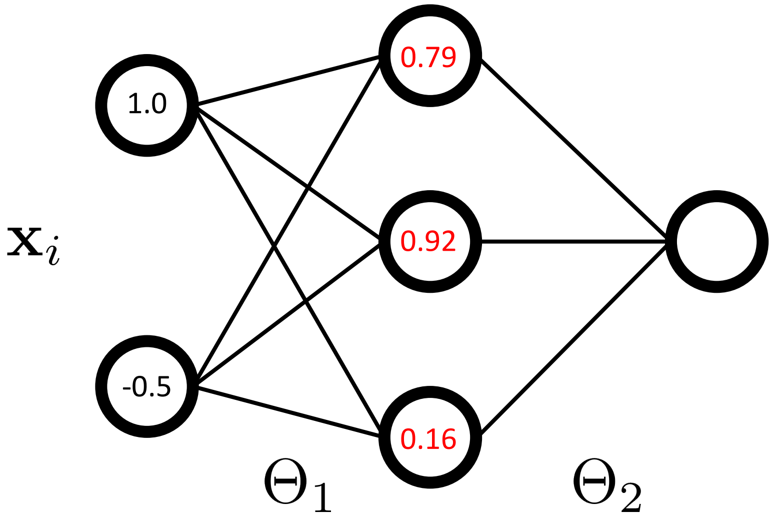
$$\mathbf{x}_i = \begin{pmatrix} 1.0 \\ -0.5 \end{pmatrix} \qquad \Theta_1 = \begin{pmatrix} 1.2 & 2.1 & 1.5 \\ -0.3 & -0.7 & 0.3 \end{pmatrix} \quad \Theta_2 = \begin{pmatrix} -0.2 \\ 0.5 \\ 1.3 \end{pmatrix}$$

- Compute hidden units $\mathbf{h}_1$

$$\Theta_1^\top \mathbf{x}_i = \begin{pmatrix} 1.2 & -0.3 \\ 2.1 & -0.7 \\ -1.5 & 0.3 \end{pmatrix} \begin{pmatrix} 1.0 \\ -0.5 \end{pmatrix} = \begin{pmatrix} 1.35 \\ 2.45 \\ -1.65 \end{pmatrix}$$

$$\mathbf{h}_1 = \sigma(\Theta_1^\top \mathbf{x}_i) = \begin{pmatrix} \sigma(1.35) \\ \sigma(2.45) \\ \sigma(-1.65) \end{pmatrix} = \begin{pmatrix} 0.79 \\ 0.92 \\ 0.16 \end{pmatrix}$$

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



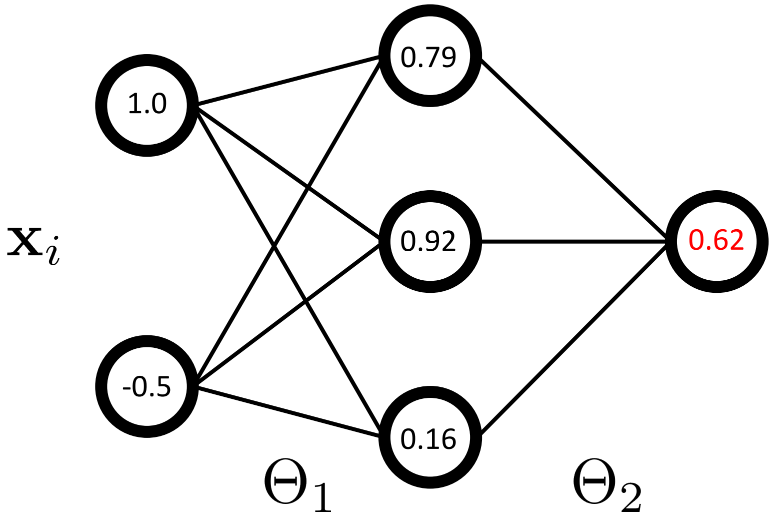
DNN: Forward Propagation (Example)

$$\mathbf{x}_i = \begin{pmatrix} 1.0 \\ -0.5 \end{pmatrix} \qquad \Theta_1 = \begin{pmatrix} 1.2 & 2.1 & 1.5 \\ -0.3 & -0.7 & 0.3 \end{pmatrix} \quad \Theta_2 = \begin{pmatrix} -0.2 \\ 0.5 \\ 1.3 \end{pmatrix}$$

- Compute output $\hat{y}_i$

$$\Theta_2^\top \mathbf{h}_1 = \begin{pmatrix} -0.2 & 0.5 & 1.3 \end{pmatrix} \begin{pmatrix} 0.79 \\ 0.92 \\ 0.16 \end{pmatrix} = 0.51$$

$$\hat{y}_i = \sigma(\Theta_2^\top \mathbf{h}_1) = \sigma(0.51) = 0.62$$

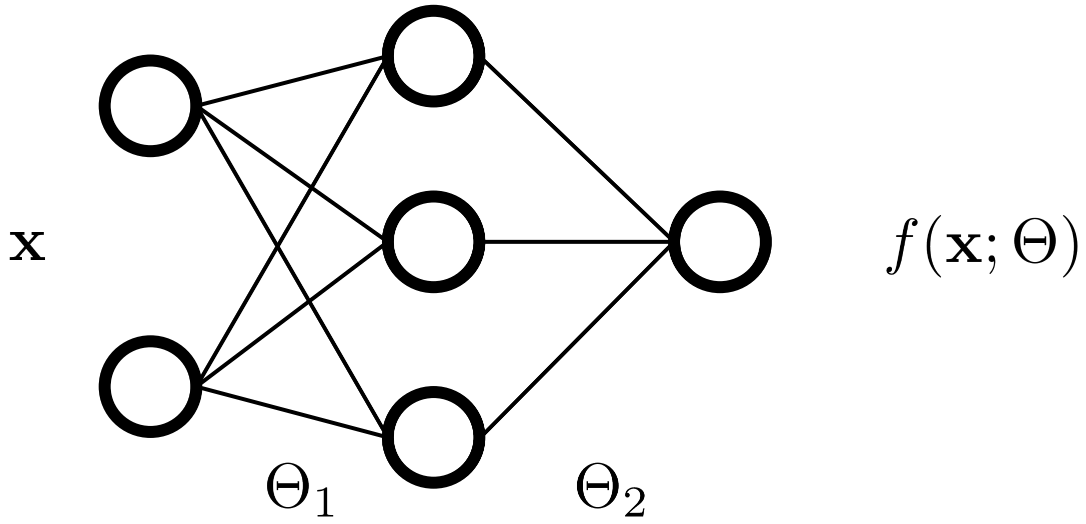


Next, training objective

- **Objective:** Find a parameter that minimizes the error (or empirical risk)

$$\min_{\Theta} \frac{1}{n} \sum_{i=1}^{n} \ell(f(\mathbf{x}_i; \Theta), y_i) := L(\Theta)$$

where $\ell(\cdot, \cdot)$ is a loss function e.g., MSE(Mean square error) or cross entropy



Next, how to optimize $L(\Theta)$?

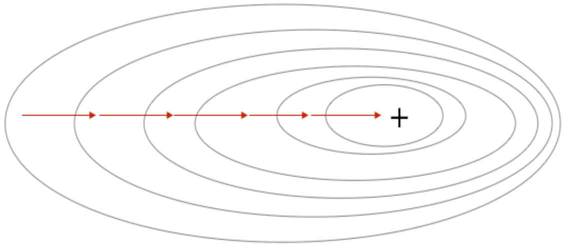
- **Gradient descent (GD)** updates parameters iteratively to the gradient direction.

parameters

loss function

$$\Theta^{(t+1)} = \Theta^{(t)} - \gamma \underbrace{\nabla L\left(\Theta^{(t)}\right)}_{:= \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(\mathbf{x}_i, y_i; \Theta^{(t)})}$$

learning rate

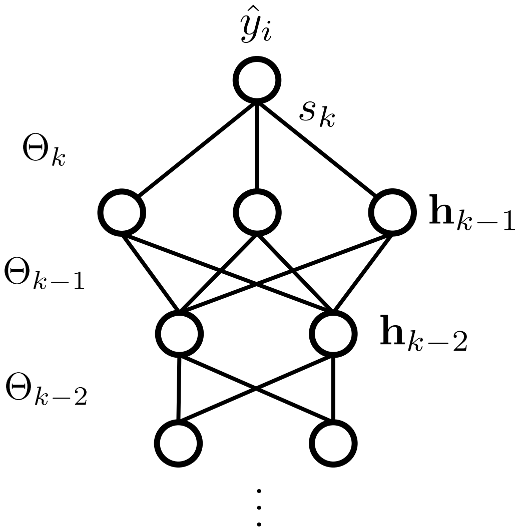


- **Backpropagation**
  - First adjust the **last layer** weights Θ_k
  - Propagate error back to each previous layers
  - Adjust **previous layer** weights $\Theta_{k-1}, \Theta_{k-2}, \ldots \Theta_1$

Next, backpropagation in details

DNN: Backpropagation

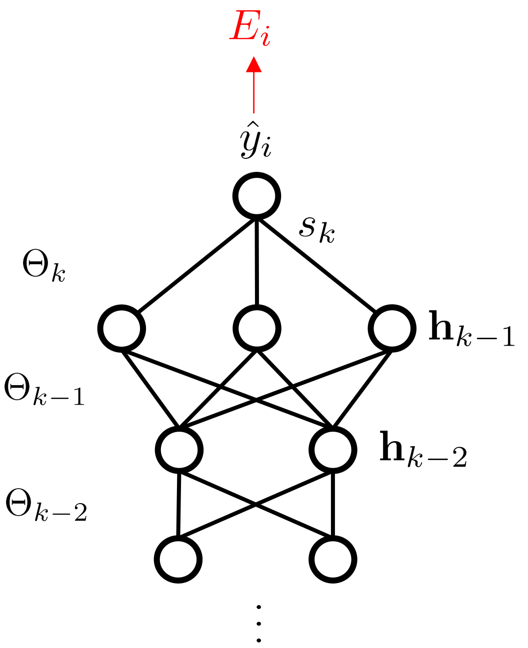
- Consider the input $(\mathbf{x}_i, y_i)$
- Forward propagation to compute output $\hat{y}_i = f(\mathbf{x}_i; \Theta)$
- i^{th} layer intermediate output $s_i = \Theta_i^\top \mathbf{h}_{i-1}$



DNN: Backpropagation

- Consider the input $(\mathbf{x}_i, y_i)$
- Forward propagation to compute output $\hat{y}_i = f(\mathbf{x}_i; \Theta)$
- i^{th} layer intermediate output $s_i = \Theta_i^\top \mathbf{h}_{i-1}$
- **Compute error** $\ell(\hat{y}_i, y_i)$ (where $\ell(\cdot, \cdot)$ is MSE loss)

$$\ell(\hat{y}_i, y_i) = \frac{1}{2}(y_i - \hat{y}_i)^2 := E_i$$



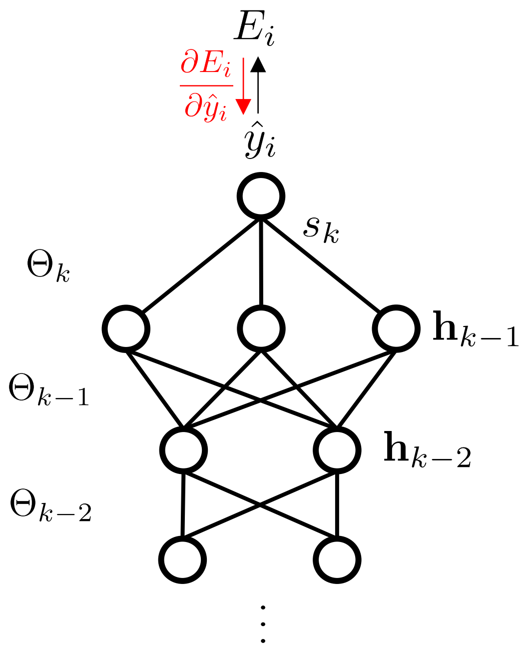
DNN: Backpropagation

- Consider the input $(\mathbf{x}_i, y_i)$
- Forward propagation to compute output $\hat{y}_i = f(\mathbf{x}_i; \Theta)$
- i^{th} layer intermediate output $s_i = \Theta_i^\top \mathbf{h}_{i-1}$
- Compute error $\ell(\hat{y}_i, y_i)$ (where $\ell(\cdot,\cdot)$ is MSE loss)

$$\ell(\hat{y}_i, y_i) = \frac{1}{2}(y_i - \hat{y}_i)^2 := E_i$$

- **Compute derivative of E_i with respect to $\hat{y}_i$**

$$\frac{\partial E_i}{\partial \hat{y}_i} = \frac{\partial}{\partial \hat{y}_i}\frac{1}{2}(y_i - \hat{y}_i)^2 = -(y_i - \hat{y}_i)$$



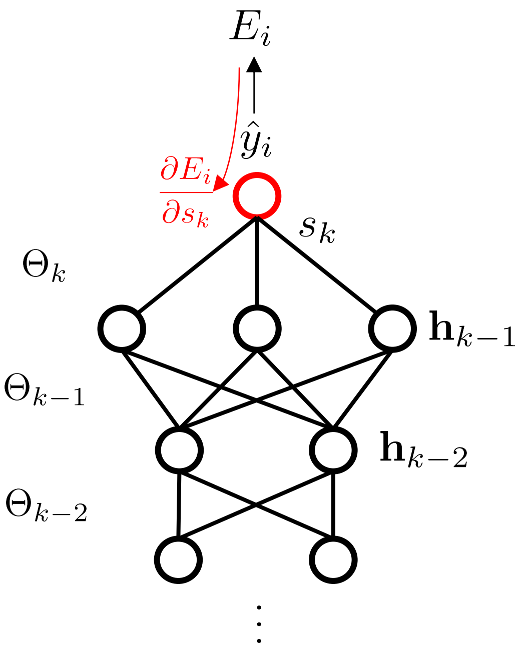
DNN: Backpropagation

- Consider the input $(\mathbf{x}_i, y_i)$
- Forward propagation to compute output $\hat{y}_i = f(\mathbf{x}_i; \Theta)$
- i^{th} layer intermediate output $s_i = \Theta_i^\top \mathbf{h}_{i-1}$
- Compute error $\ell(\hat{y}_i, y_i)$ (where $\ell(\cdot,\cdot)$ is MSE loss)

$$\ell(\hat{y}_i, y_i) = \frac{1}{2}(y_i - \hat{y}_i)^2 := E_i$$

- **Compute derivative of** E_i **with respect to** s_k

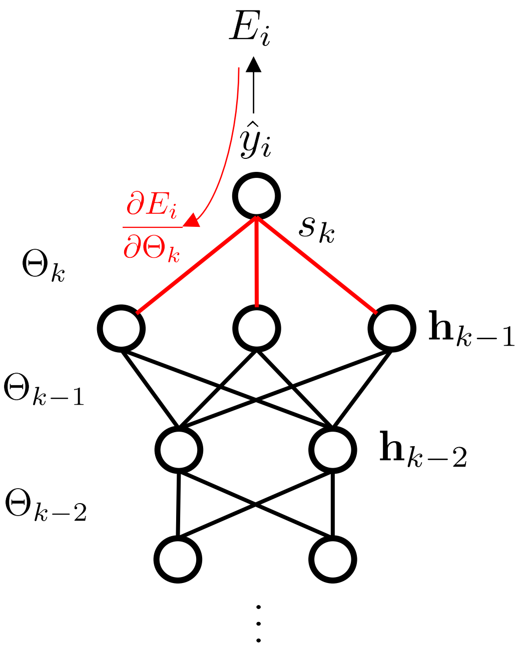
$$\frac{\partial E_i}{\partial s_k} = \frac{\partial E_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s_k} = \frac{\partial E_i}{\partial \hat{y}_i}\frac{\partial}{\partial s_k}\sigma(s_k) = (\hat{y}_i - y_i)\sigma'(s_k)$$



DNN: Backpropagation

- Consider the input $(\mathbf{x}_i, y_i)$
- Forward propagation to compute output $\hat{y}_i = f(\mathbf{x}_i; \Theta)$
- i^{th} layer intermediate output $s_i = \Theta_i^\top \mathbf{h}_{i-1}$
- Compute error $\ell(\hat{y}_i, y_i)$ (where $\ell(\cdot,\cdot)$ is MSE loss)

$$\ell(\hat{y}_i, y_i) = \frac{1}{2}(y_i - \hat{y}_i)^2 := E_i$$



- **Compute derivative of E_i with respect to Θ_k**

$$\frac{\partial E_i}{\partial \Theta_k} = \frac{\partial E_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s_k}\frac{\partial s_k}{\partial \Theta_k} = \frac{\partial E_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s_k}\frac{\partial}{\partial \Theta_k}(\Theta_k^\top \mathbf{h}_{k-1}) = (\hat{y}_i - y_i)\sigma'(s_k)\mathbf{h}_{k-1}$$

- Parameter update rule

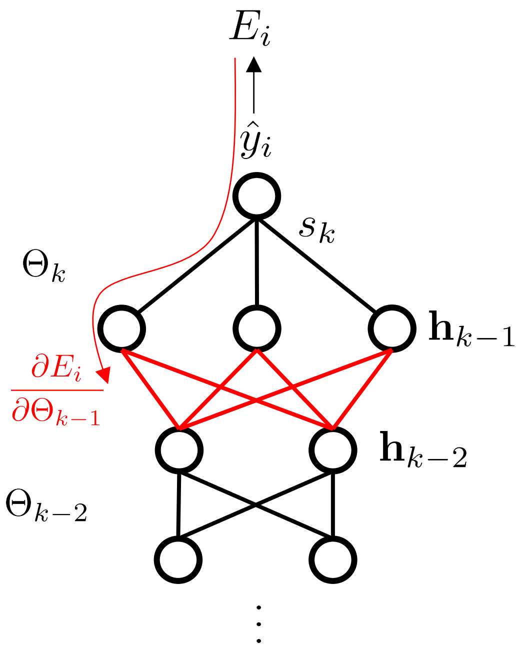
learning rate

$$\Theta_k \leftarrow \Theta_k - \gamma \frac{\partial E_i}{\partial \Theta_k}$$

DNN: Backpropagation

- Consider the input $(\mathbf{x}_i, y_i)$
- Forward propagation to compute output $\hat{y}_i = f(\mathbf{x}_i; \Theta)$
- i^{th} layer intermediate output $s_i = \Theta_i^\top \mathbf{h}_{i-1}$
- Compute error $\ell(\hat{y}_i, y_i)$ (where $\ell(\cdot,\cdot)$ is MSE loss)

$$\ell(\hat{y}_i, y_i) = \frac{1}{2}(y_i - \hat{y}_i)^2 := E_i$$



- **Compute derivative of E_i with respect to Θ_{k-1}**

$$\frac{\partial E_i}{\partial \Theta_{k-1}} = \frac{\partial E_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s_k}\frac{\partial s_k}{\partial \mathbf{h}_{k-1}}\frac{\partial \mathbf{h}_{k-1}}{\partial \mathbf{s}_{k-1}}\frac{\partial \mathbf{s}_{k-1}}{\partial \Theta_{k-1}} = \frac{\partial E_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s_k}\frac{\partial s_k}{\partial \mathbf{h}_{k-1}}\frac{\partial \mathbf{h}_{k-1}}{\partial \mathbf{s}_{k-1}}\frac{\partial}{\partial \Theta_{k-1}}(\Theta_{k-1}^\top \mathbf{h}_{k-2})$$

- Parameter update rule

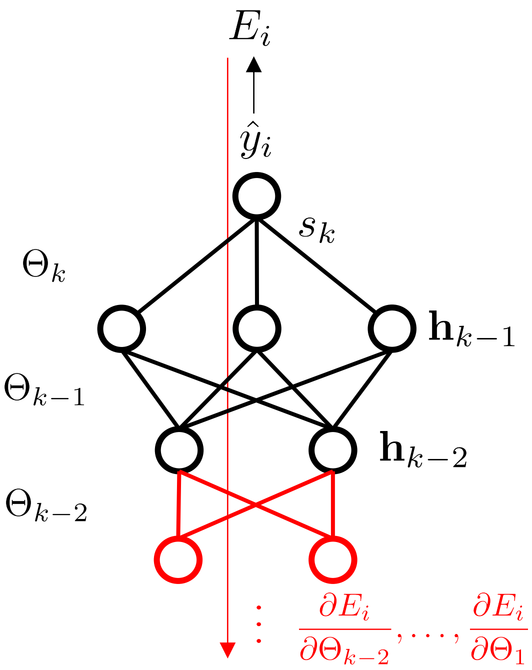
learning rate

$$\Theta_{k-1} \leftarrow \Theta_{k-1} - \gamma \frac{\partial E_i}{\partial \Theta_{k-1}}$$

DNN: Backpropagation

- Consider the input $(\mathbf{x}_i, y_i)$
- Forward propagation to compute output $\hat{y}_i = f(\mathbf{x}_i; \Theta)$
- i^{th} layer intermediate output $s_i = \Theta_i^\top \mathbf{h}_{i-1}$
- Compute error $\ell(\hat{y}_i, y_i)$ (where $\ell(\cdot,\cdot)$ is MSE loss)

$$\ell(\hat{y}_i, y_i) = \frac{1}{2}(y_i - \hat{y}_i)^2 := E_i$$



- Similarly, we can compute gradients with respect to $\Theta_{k-2}, \ldots, \Theta_1$
  - And update using the same update rule

DNN: Backpropagation (Example)

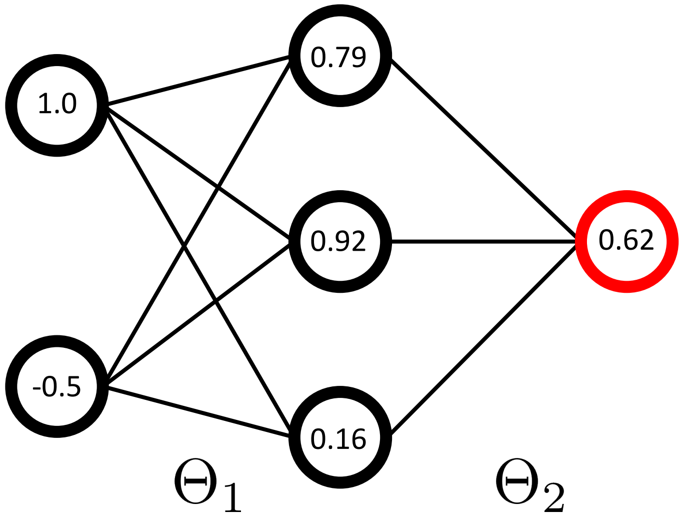
$$\mathbf{x}_i = \begin{pmatrix} 1.0 \\ -0.5 \end{pmatrix} \quad y_i = (1.0) \quad \Theta_1 = \begin{pmatrix} 1.2 & 2.1 & 1.5 \\ -0.3 & -0.7 & 0.3 \end{pmatrix} \quad \Theta_2 = \begin{pmatrix} -0.2 \\ 0.5 \\ 1.3 \end{pmatrix}$$

- Compute the error $\ell(\hat{y}_i, y_i)$

$$\ell(\hat{y}_i, y_i) = \frac{1}{2}(y_i - \hat{y}_i)^2 = 0.072$$

- Compute $\dfrac{\partial E_i}{\partial \hat{y}_i}$

$$\frac{\partial E_i}{\partial \hat{y}_i} = (\hat{y}_i - y_i) = -0.38$$



DNN: Backpropagation (Example)

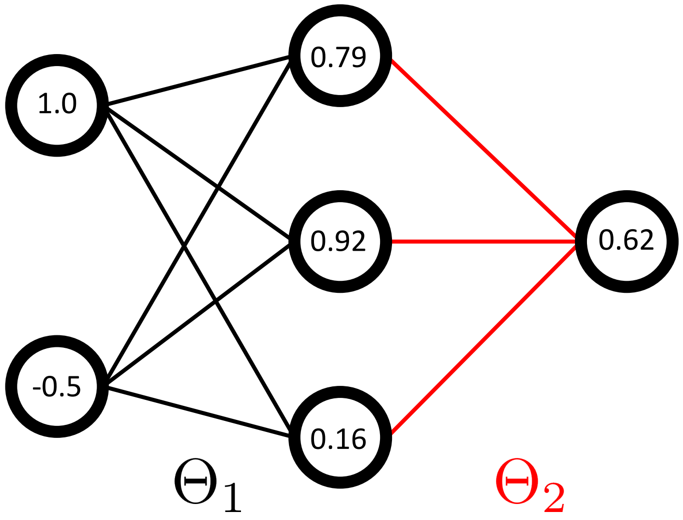
$$\mathbf{x}_i = \begin{pmatrix} 1.0 \\ -0.5 \end{pmatrix} \quad y_i = (1.0) \quad \Theta_1 = \begin{pmatrix} 1.2 & 2.1 & 1.5 \\ -0.3 & -0.7 & 0.3 \end{pmatrix} \quad \Theta_2 = \begin{pmatrix} -0.2 \\ 0.5 \\ 1.3 \end{pmatrix}$$

- Compute $\dfrac{\partial E_i}{\partial \Theta_2}$

$$\frac{\partial E_i}{\partial \Theta_2} = (\hat{y}_i - y_i)\sigma'(s_2)\mathbf{h}_1 = \begin{pmatrix} 0.02 \\ -0.05 \\ -0.12 \end{pmatrix}$$

- Update Θ_2 with $\gamma = 1$

$$\Theta_2 = \begin{pmatrix} -0.2 \\ 0.5 \\ 1.3 \end{pmatrix} - 1 \begin{pmatrix} 0.02 \\ -0.05 \\ -0.12 \end{pmatrix} = \begin{pmatrix} -0.22 \\ 0.55 \\ 1.42 \end{pmatrix}$$

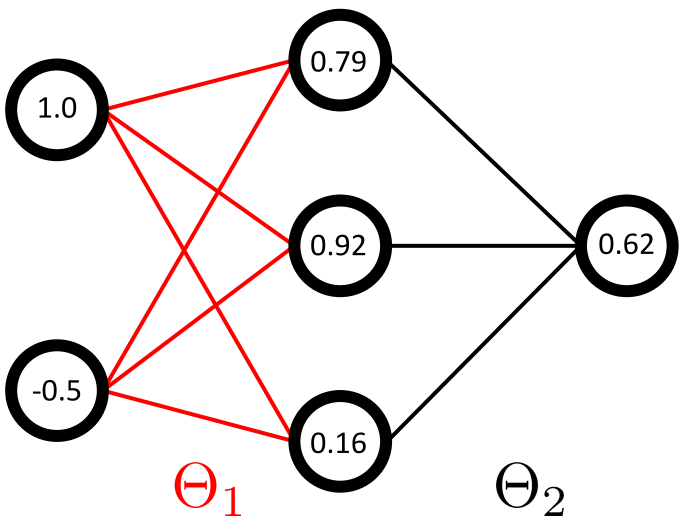


DNN: Backpropagation (Example)

$$\mathbf{x}_i = \begin{pmatrix} 1.0 \\ -0.5 \end{pmatrix} \quad y_i = (1.0) \quad \Theta_1 = \begin{pmatrix} 1.2 & 2.1 & 1.5 \\ -0.3 & -0.7 & 0.3 \end{pmatrix} \quad \Theta_2 = \begin{pmatrix} -0.2 \\ 0.5 \\ 1.3 \end{pmatrix}$$

- Compute $\frac{\partial E_i}{\partial \Theta_2}$

$$\frac{\partial E_i}{\partial \Theta_2} = (\hat{y}_i - y_i)\sigma'(s_2)\mathbf{h}_1 = \begin{pmatrix} 0.02 \\ -0.05 \\ -0.12 \end{pmatrix}$$



- Update Θ_2 with $\gamma = 1$

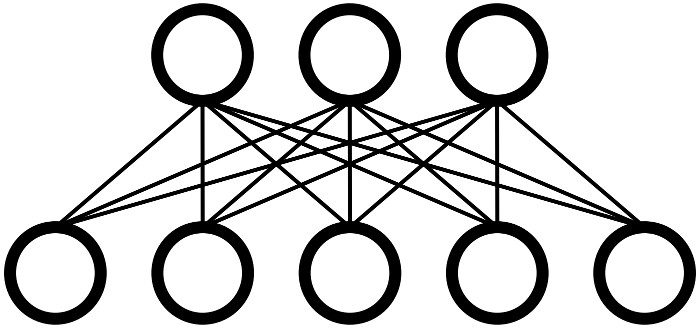
$$\Theta_2 = \begin{pmatrix} -0.2 \\ 0.5 \\ 1.3 \end{pmatrix} - 1 \begin{pmatrix} 0.02 \\ -0.05 \\ -0.12 \end{pmatrix} = \begin{pmatrix} -0.22 \\ 0.55 \\ 1.42 \end{pmatrix}$$

- Similarly, we can update Θ_1

Table of Contents

CNN: Drawbacks of Fully-Connected DNN

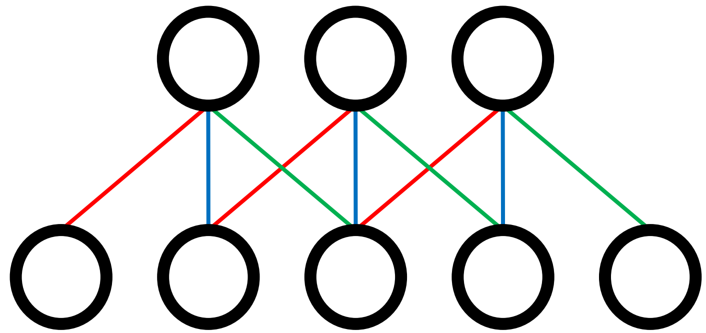
- Previous DNNs use fully-connected layers
    - Connect **all** the neurons between the layers



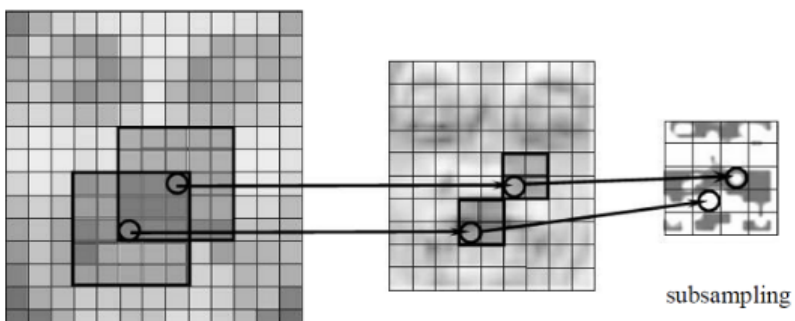
- Drawbacks
    - (-) **Large number of parameters**
        - Easy to be over-fitted
        - Large memory consumption
    - (-) Does not enforce any structure, e.g., ***local*** **information**
        - In many applications, local features are important, e.g., images, language, etc.

CNN: Basics

- **Weight sharing** and **local connectivity** (convolution)
  - Use multiple filters convolve over inputs
  - (+) **Reduce** the number of **parameters** (less over-fitting)
  - (+) Learn **local** features
  - (+) **Translation invariance**



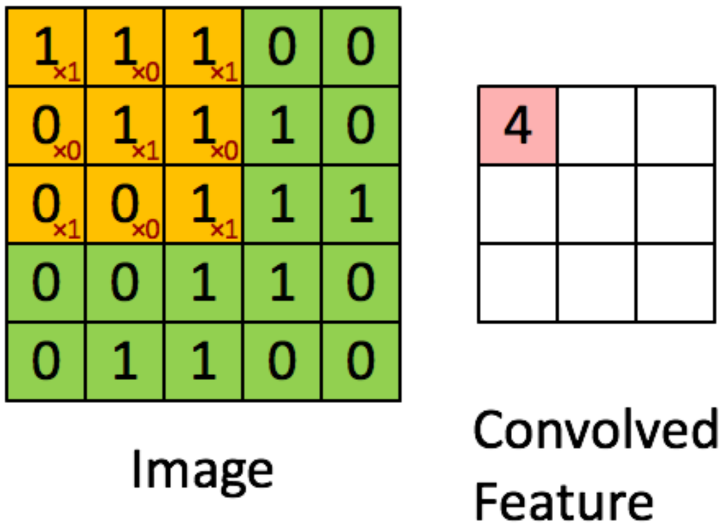
- **Pooling** (or subsampling)
  - Make the **representations smaller**
  - (+) Reduce number of parameters and computation



*source : http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.1794&rep=rep1&type=pdf

CNN: Weight Sharing and Translation Invariance

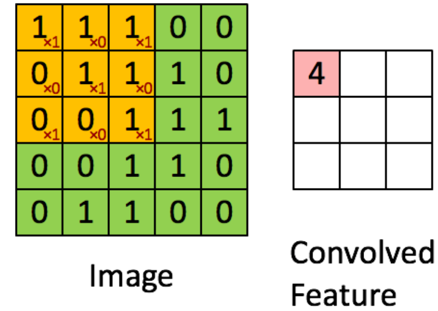
- **Weight sharing**
  - Apply same weights over the different spatial regions
  - One can achieve **translation invariance** (not perfect though)



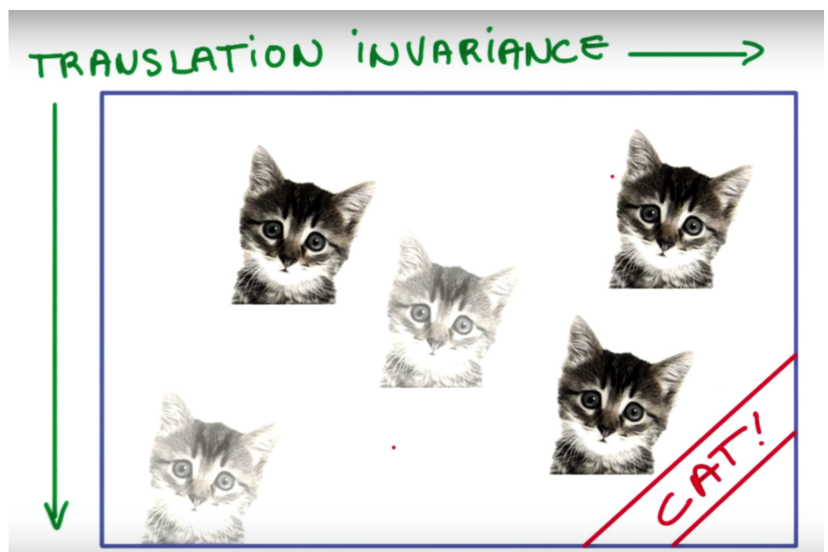
*source : https://www.cc.gatech.edu/~san37/post/dlhc-cnn/

CNN: Weight Sharing and Translation Invariance

- **Weight sharing**
  - Apply same weights over the different spatial regions
  - One can achieve **translation invariance**



- **Translation invariance**
  - When input is changed spatially (translated or shifted), the corresponding output to recognize the object should not be changed
  - CNN can produce the same output even though the input image is shifted due to weight sharing



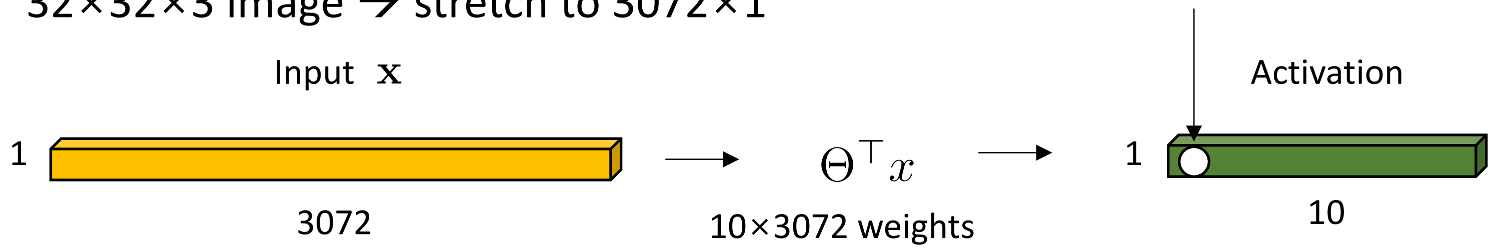
*source : https://www.cc.gatech.edu/~san37/post/dlhc-cnn/

CNN: Convolution

Fully-connected layer

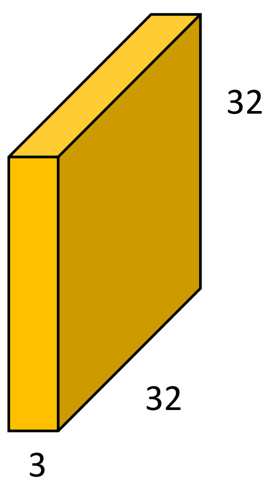
- $32 \times 32 \times 3$ image → stretch to 3072×1

The result of taking a dot product between a row of $\Theta^\top$ and the input



Convolution layer

$32 \times 32 \times 3$ image



$5 \times 5 \times 3$ filter

(equivalent to 1×75 weights for FC layer)

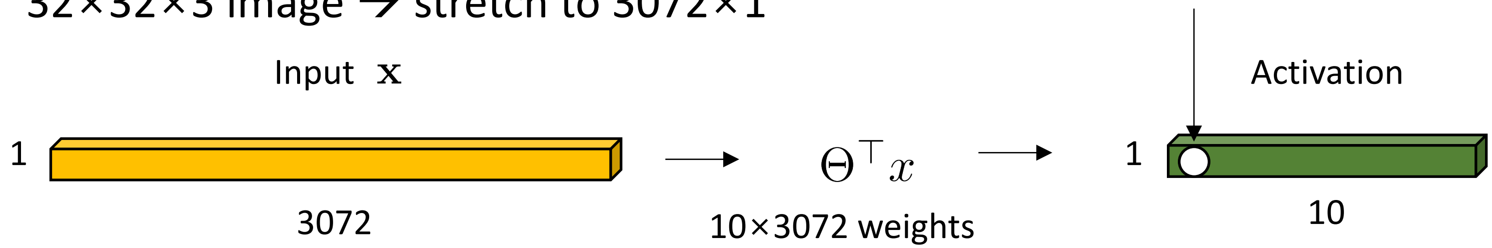
Convolve the filter with the image i.e., "slide over the image spatially, computing dot products"

*reference : http://cs231n.stanford.edu/2017/

Fully-connected layer

- 32×32×3 image → stretch to 3072×1

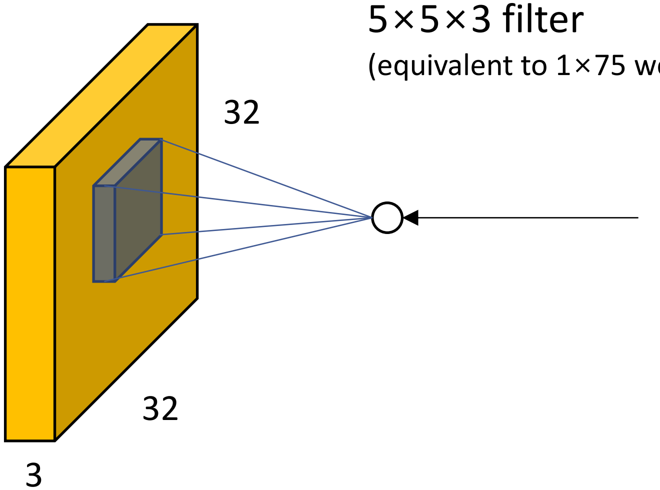
The result of taking a dot product between a row of $\Theta^\top$ and the input



Convolution layer

32×32×3 image

5×5×3 filter
(equivalent to 1×75 weights for FC layer)



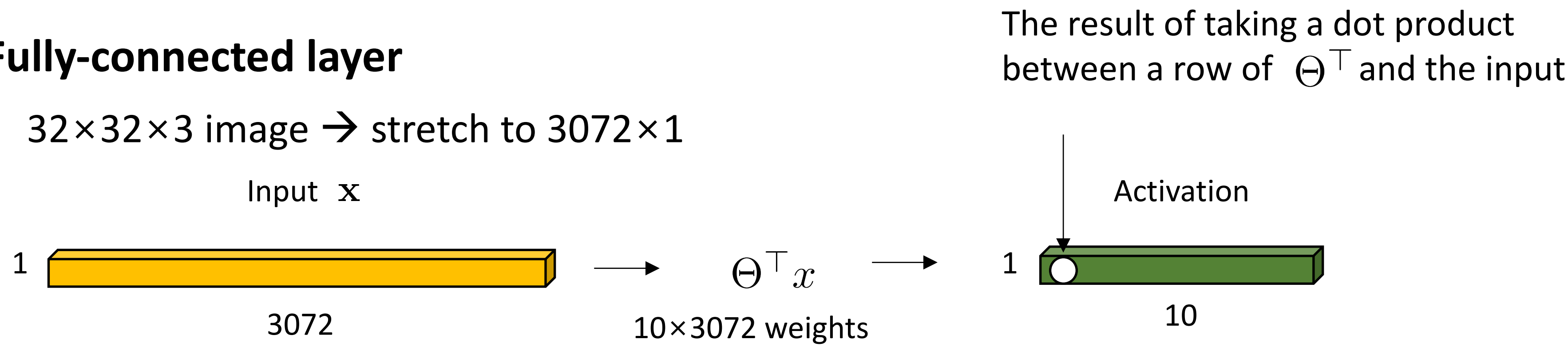
The result of taking a dot product between the filter and a small 5×5×3 chunk of the image
(i.e., 5×5×3 = 75-dimensional dot product + bias)

$$\Theta^\top \mathbf{x} + \mathbf{b}$$

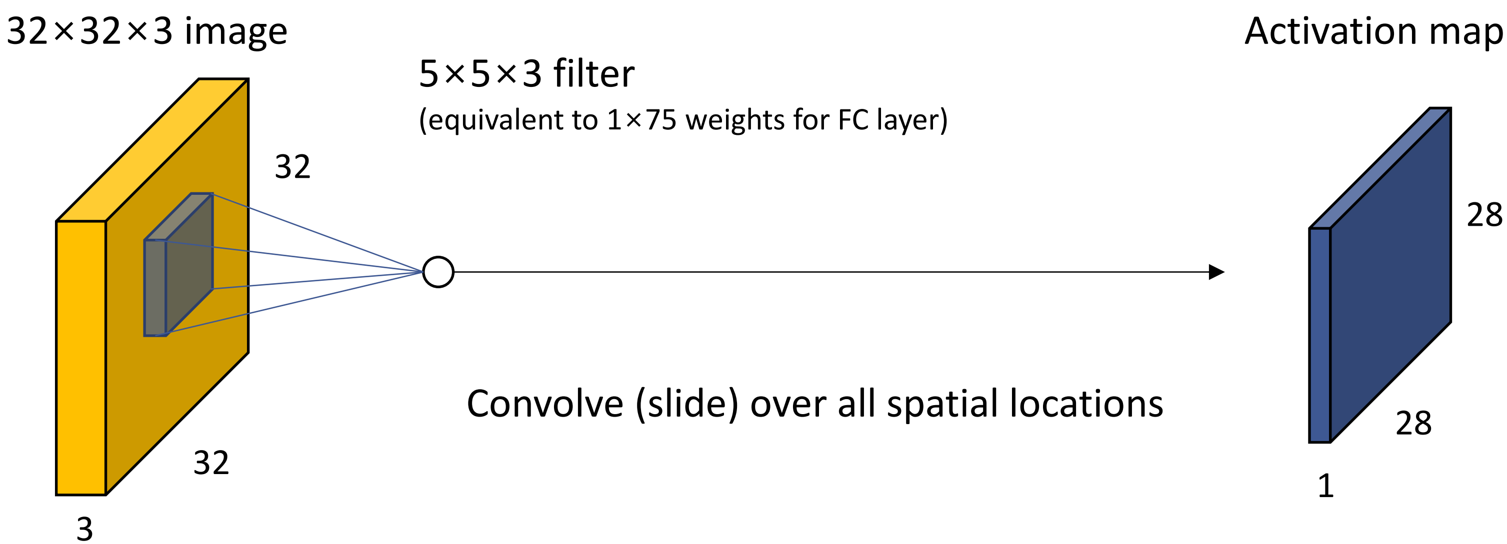
*reference : http://cs231n.stanford.edu/2017/

Fully-connected layer

- 32×32×3 image → stretch to 3072×1



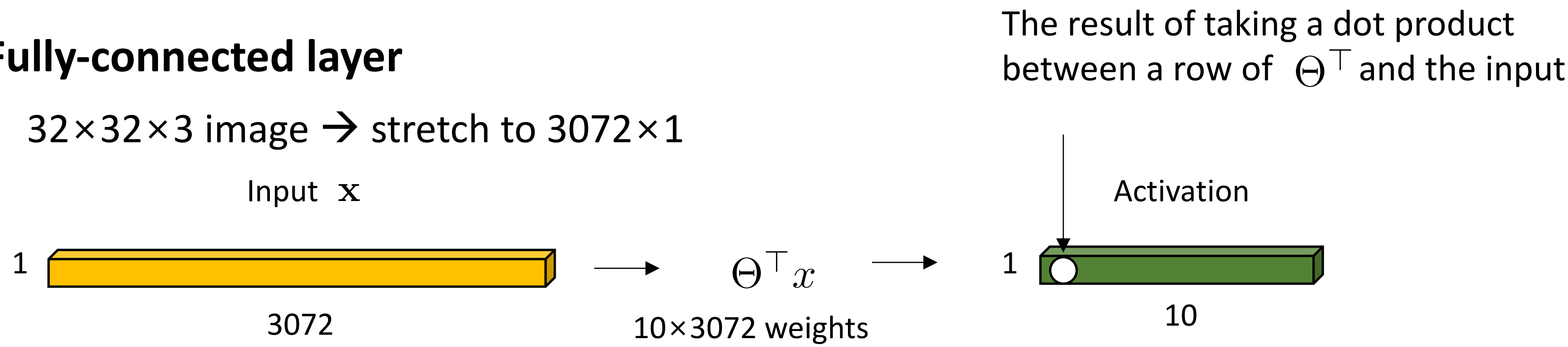
Convolution layer



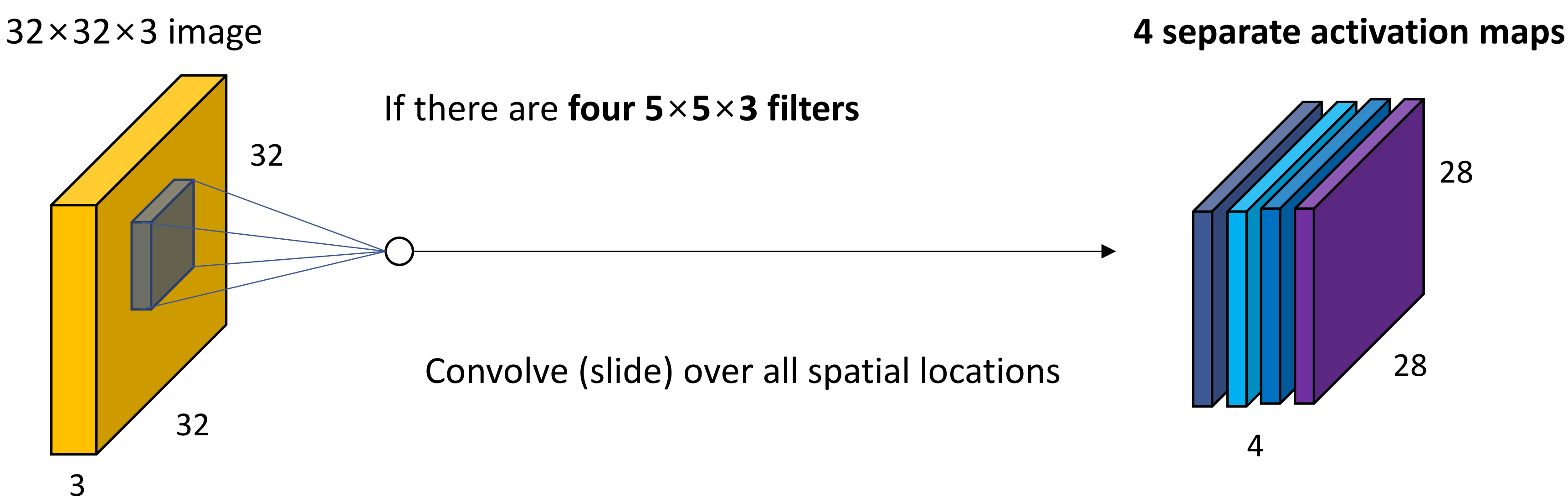
*reference : http://cs231n.stanford.edu/2017/

Fully-connected layer

- 32×32×3 image → stretch to 3072×1



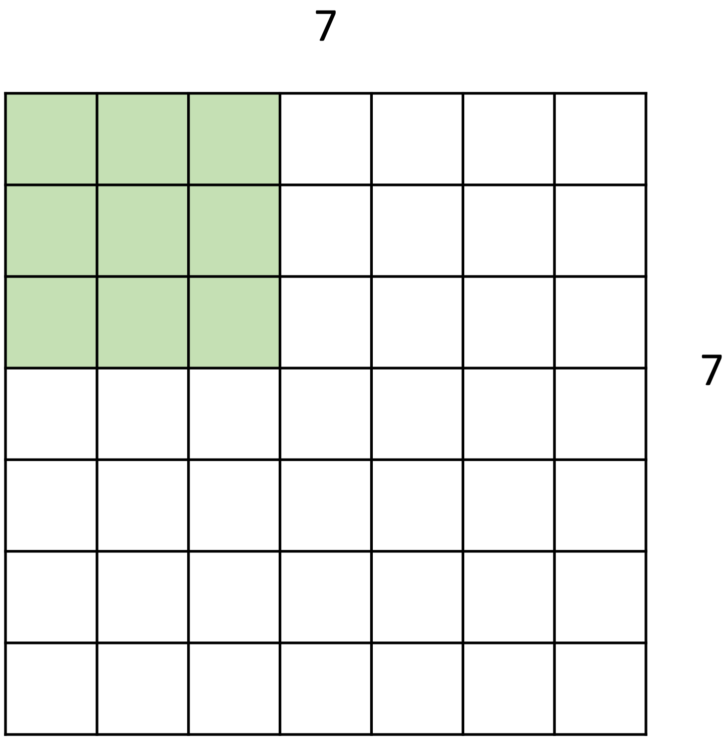
Convolution layer



*reference : http://cs231n.stanford.edu/2017/

CNN: An Example

- Closer look at spatial dimensions

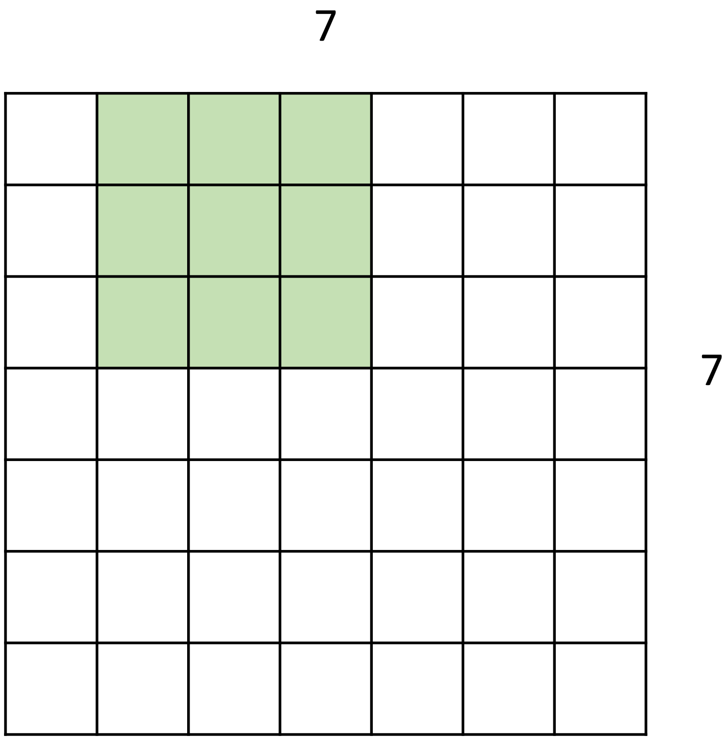


7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 1**

*reference : http://cs231n.stanford.edu/2017/

CNN: An Example

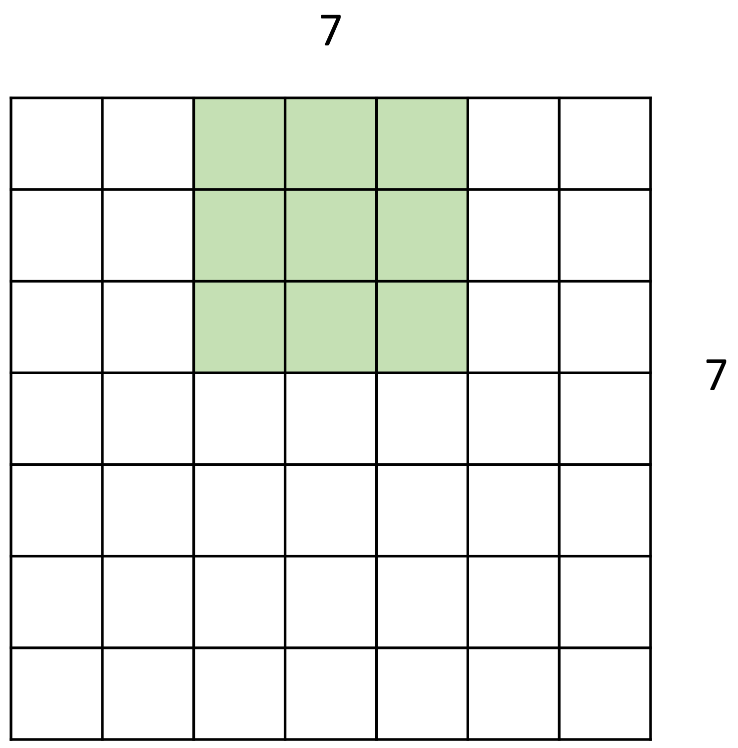
- Closer look at spatial dimensions



7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 1**

*reference : http://cs231n.stanford.edu/2017/

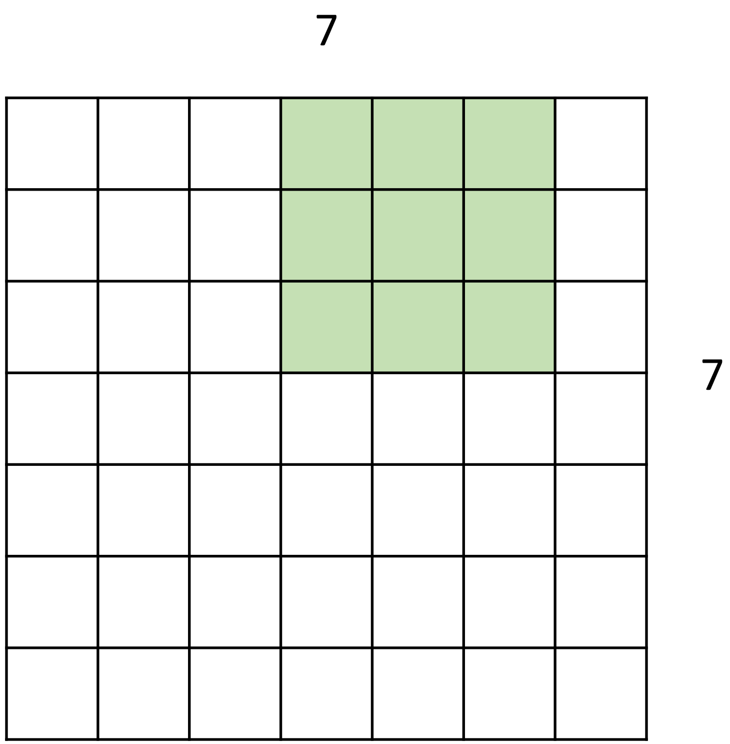
- Closer look at spatial dimensions



7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 1**

*reference : http://cs231n.stanford.edu/2017/

- Closer look at spatial dimensions

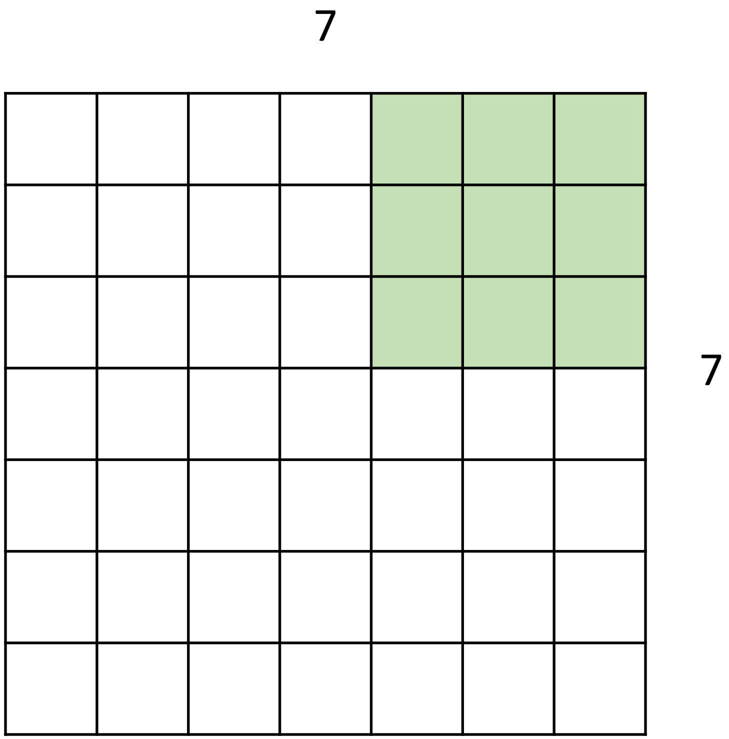


7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 1**

*reference : http://cs231n.stanford.edu/2017/

CNN: An Example

- Closer look at spatial dimensions



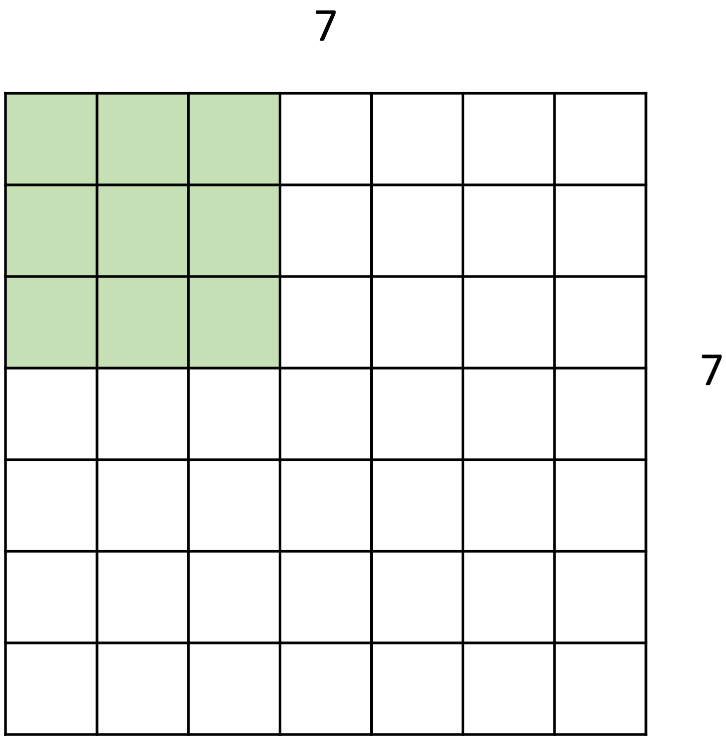
7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 1**

→ **5×5 output**

*reference : http://cs231n.stanford.edu/2017/

CNN: An Example

- Closer look at spatial dimensions

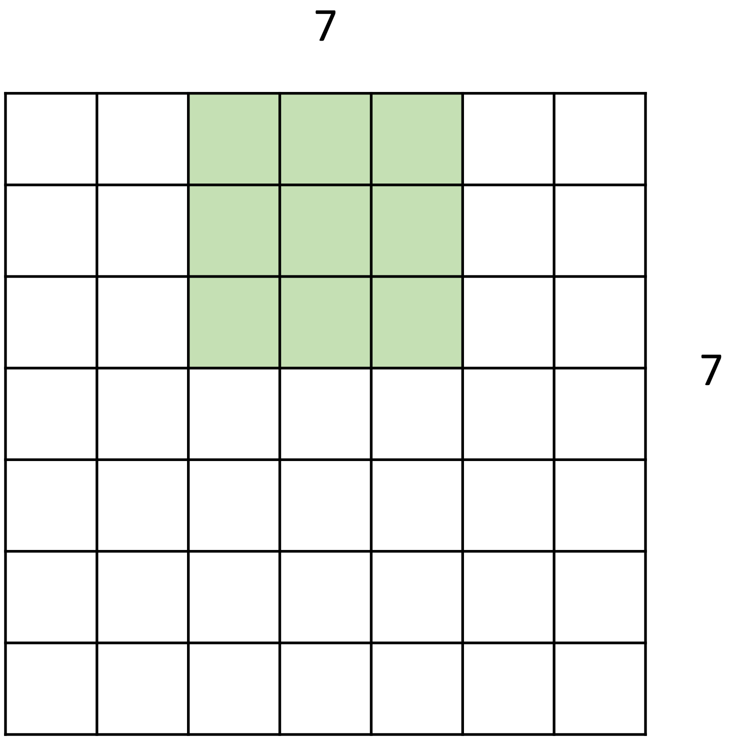


7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 2**

*reference : http://cs231n.stanford.edu/2017/

CNN: An Example

- Closer look at spatial dimensions

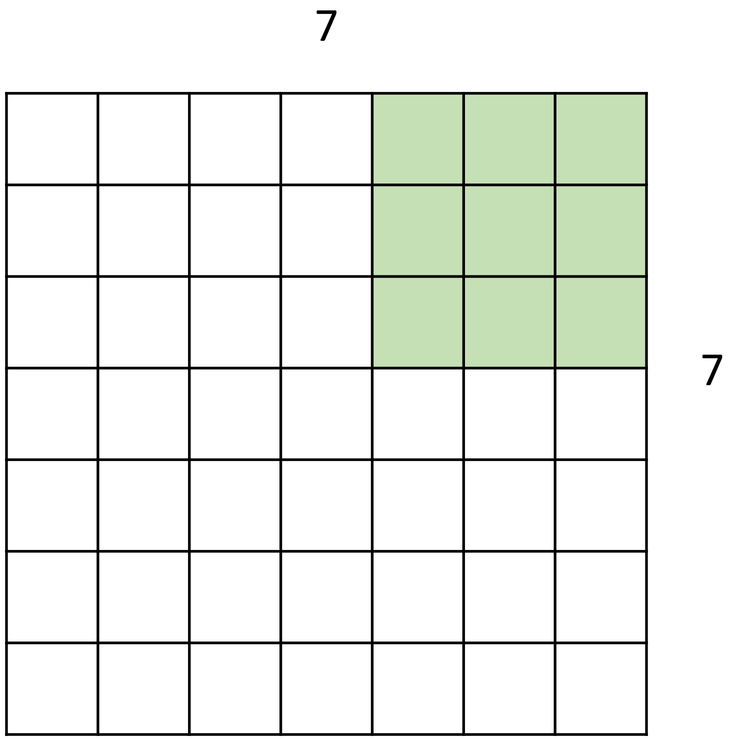


7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 2**

*reference : http://cs231n.stanford.edu/2017/

CNN: An Example

- Closer look at spatial dimensions



7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 2**

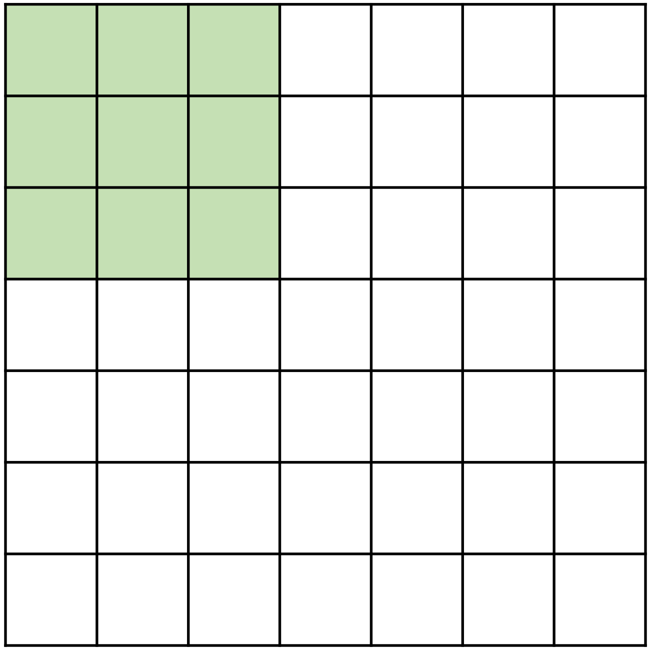
→ **3×3 output**

*reference : http://cs231n.stanford.edu/2017/

CNN: An Example

- Closer look at spatial dimensions

7



7

7×7 input (spatially)
Assume 3×3 filter
Applied with **stride 3 ?**

Doesn't fit!
Cannot apply 3×3 filter on
7×7 input with stride 3

*reference : http://cs231n.stanford.edu/2017/

CNN: An Example

- In practice: Common to **zero pad** the border
  - Used to control the output filter size

9

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

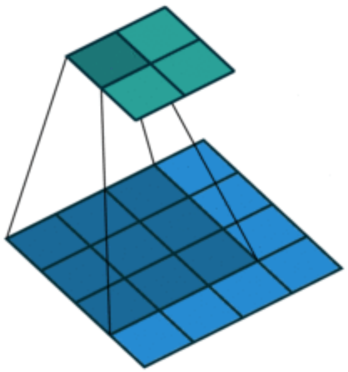
9

7×7 input (spatially)
Zero pad 1 pixel border
Assume 3×3 filter
Applied with **stride 3**

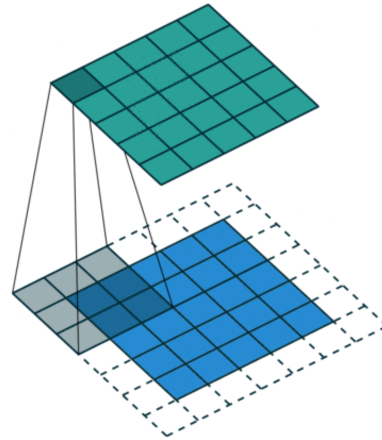
→ **3×3 output**

*reference : http://cs231n.stanford.edu/2017/

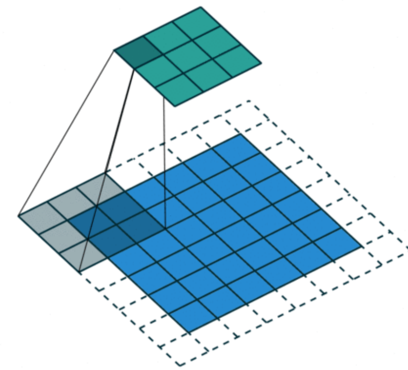
CNN: An Example (Animation)



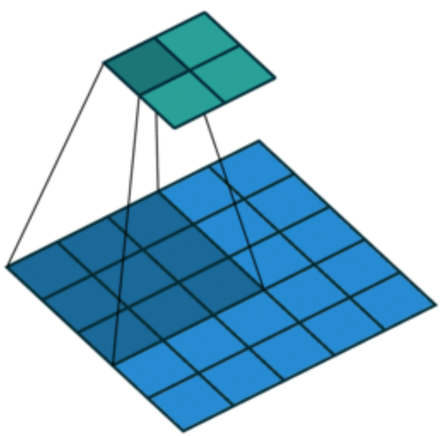
No padding, stride 1



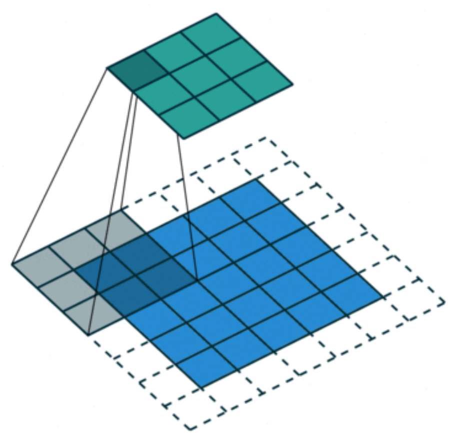
Padding 1, stride 1



Padding 1, stride 2 (odd)



No padding, stride 2

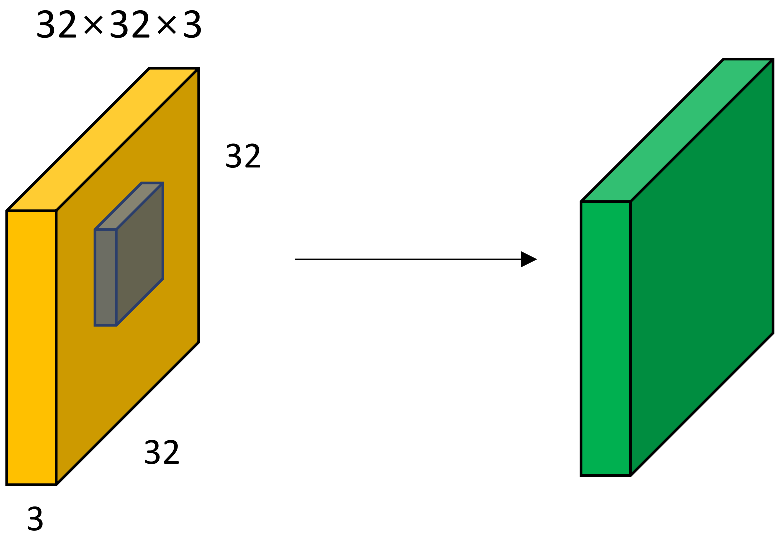


Padding 1, stride 2

*source : https://github.com/vdumoulin/conv_arithmetic

CNN: An Example

- Input volume : 32×32×3
- 10 5×5 filters with stride 1, pad 2

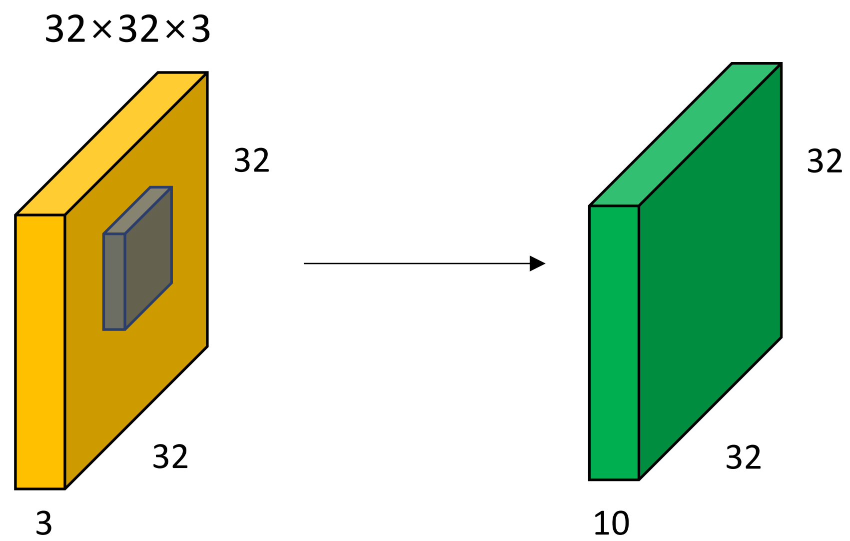


Output volume size = ?

*reference : http://cs231n.stanford.edu/2017/

CNN: An Example

- Input volume : $32 \times 32 \times 3$
- 10 5×5 filters with stride 1, pad 2



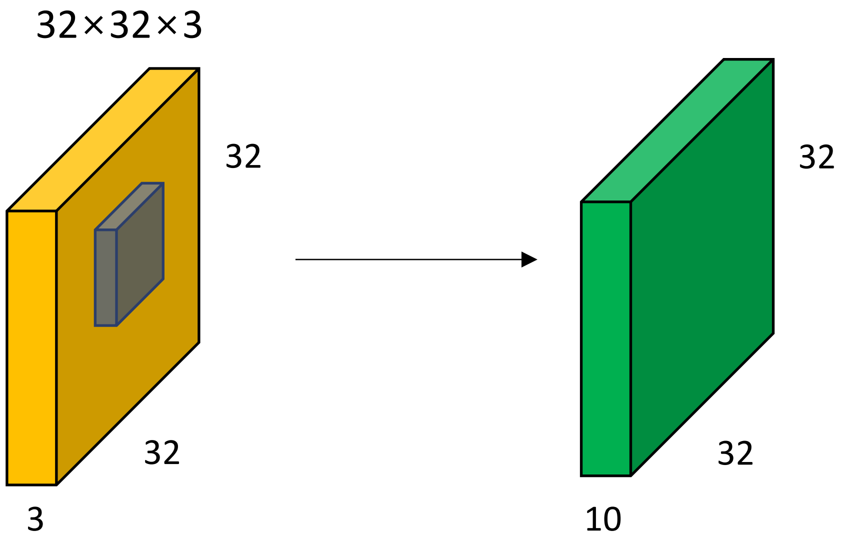
Output volume size = ?

- $(32 + 2 \times 2 - 5)/1 + 1 = 32$ spatially
- $=> 32 \times 32 \times 10$

*reference : http://cs231n.stanford.edu/2017/

CNN: An Example

- Input volume : 32×32×3
- 10 5×5 filters with stride 1, pad 2

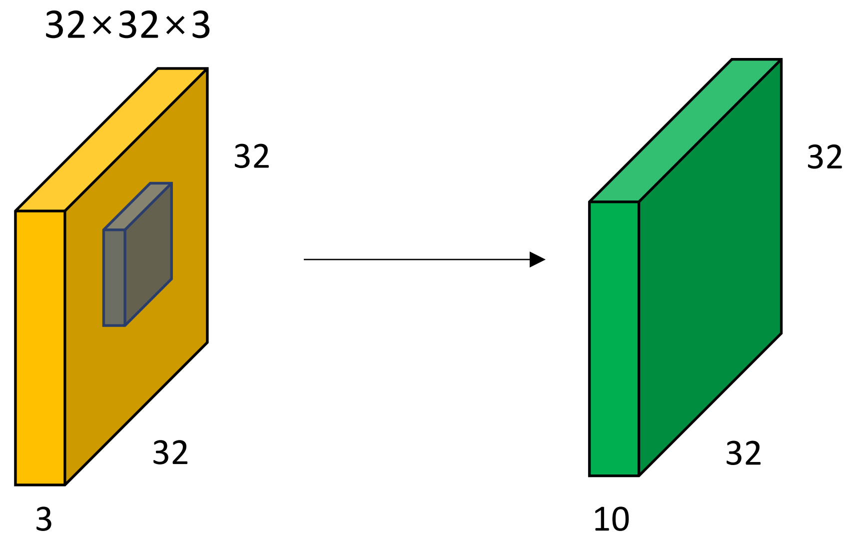


Number of parameters in this layer?

*reference : http://cs231n.stanford.edu/2017/

CNN: An Example

- Input volume : $32\times32\times3$
- 10 5×5 filters with stride 1, pad 2



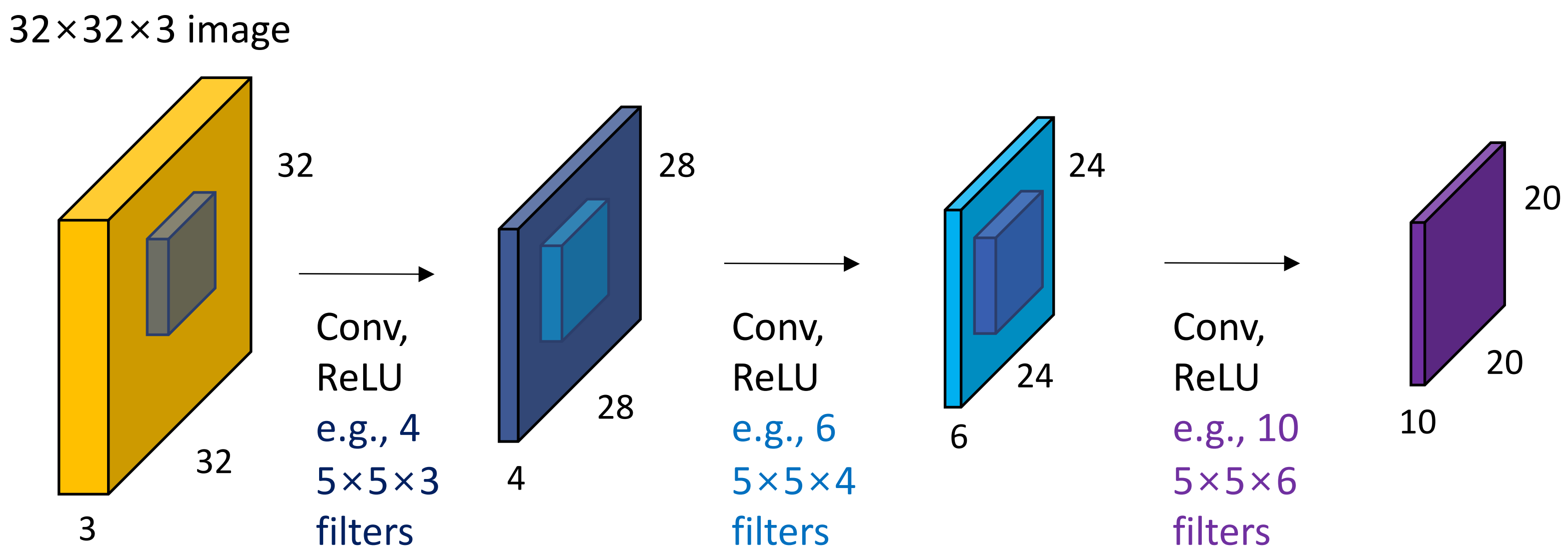
Number of parameters in this layer?

- Each filter has $5\times5\times3 + 1 = 76$ params (+1 for bias)
- = > $76\times10 =$ **760**

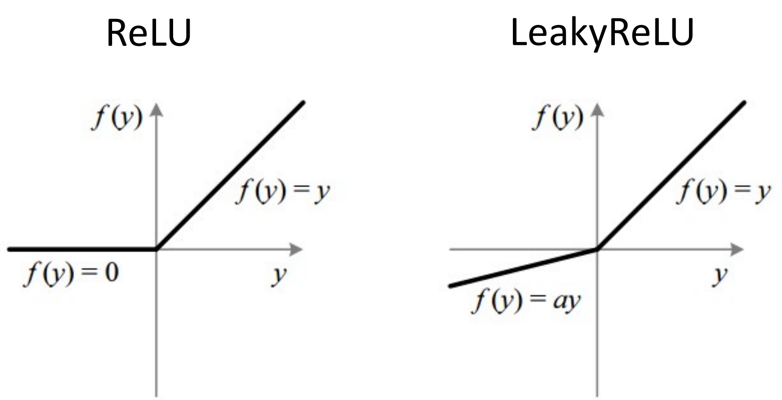
*reference : http://cs231n.stanford.edu/2017/

CNN: Convolution

- **ConvNet** is a sequence of Convolutional layers, followed by non-linearity



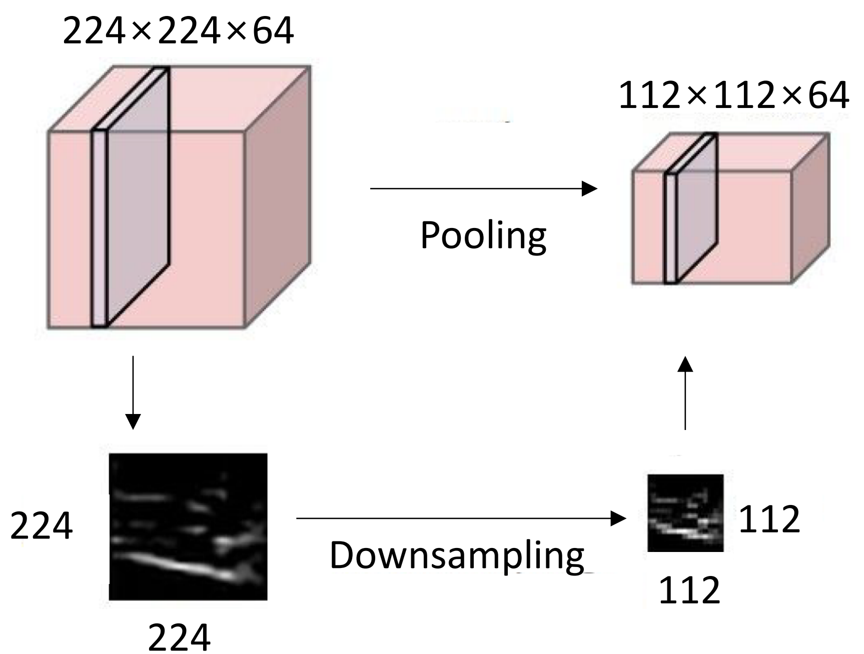
- Choices of other **non-linearity**
  - Tanh/Sigmoid
  - ReLU [Nair et al., 2010]
  - Leaky ReLU [Maas et. al., 2013]



*reference: http://cs231n.stanford.edu/2017/
*Image source: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

CNN: Pooling

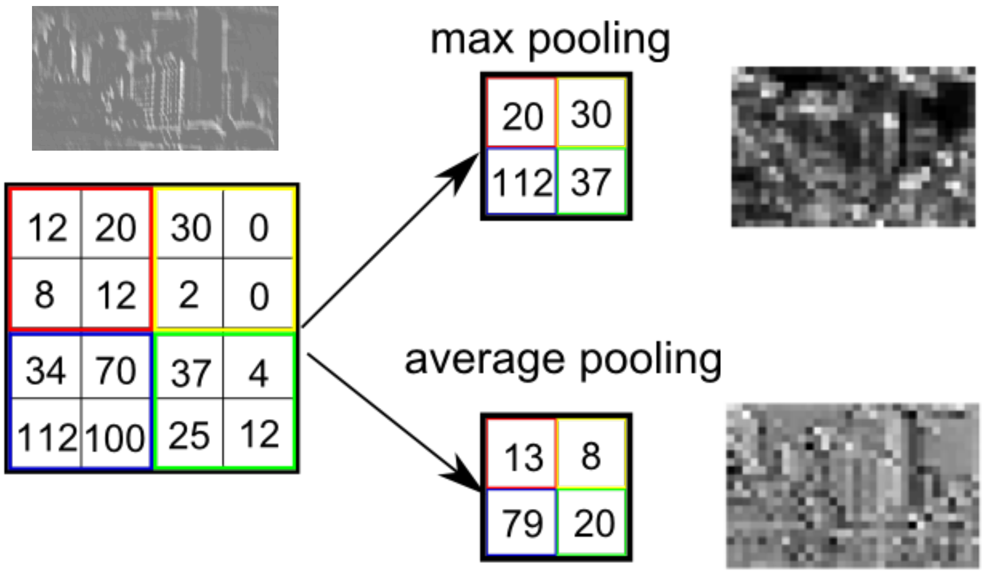
- **Pooling layer**
  - Makes the **representations smaller** and more **manageable**
  - Operates over each activation map independently
  - Enhance **translation invariance** (invariance to small transformation)
  - Larger receptive fields (see more of input)
  - Regularization effect



*reference : http://cs231n.stanford.edu/2017/

CNN: Pooling

- Max pooling and average pooling
    - With 2×2 filters and stride 2



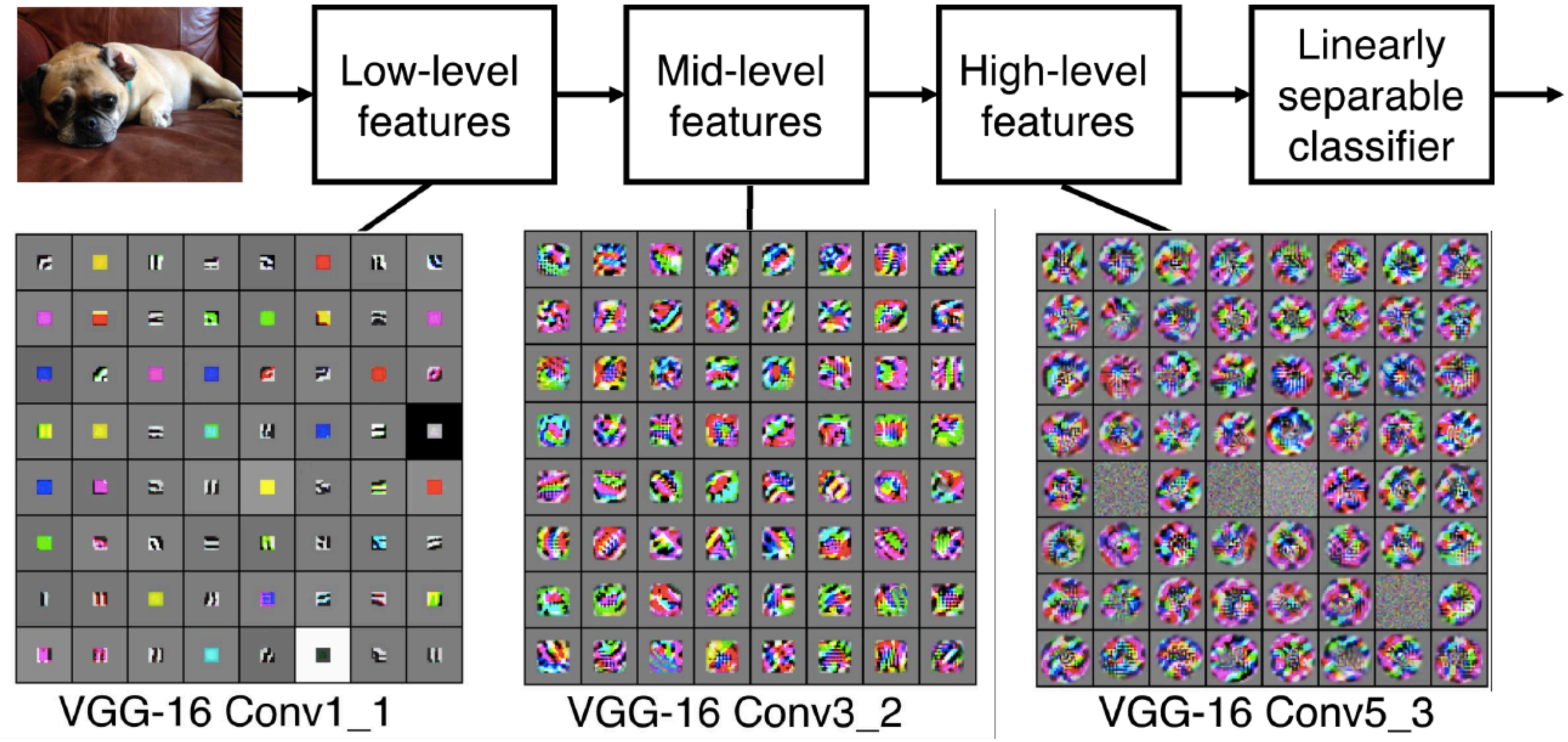
ROI pooling

- Another kind of pooling layers are also used
    - e.g. stochastic pooling, ROI pooling

*source:
https://deepsense.ai/region-of-interest-pooling-explained/
http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf
https://vaaaaaanquish.hatenablog.com/entry/2015/01/26/060622

CNN: Visualization

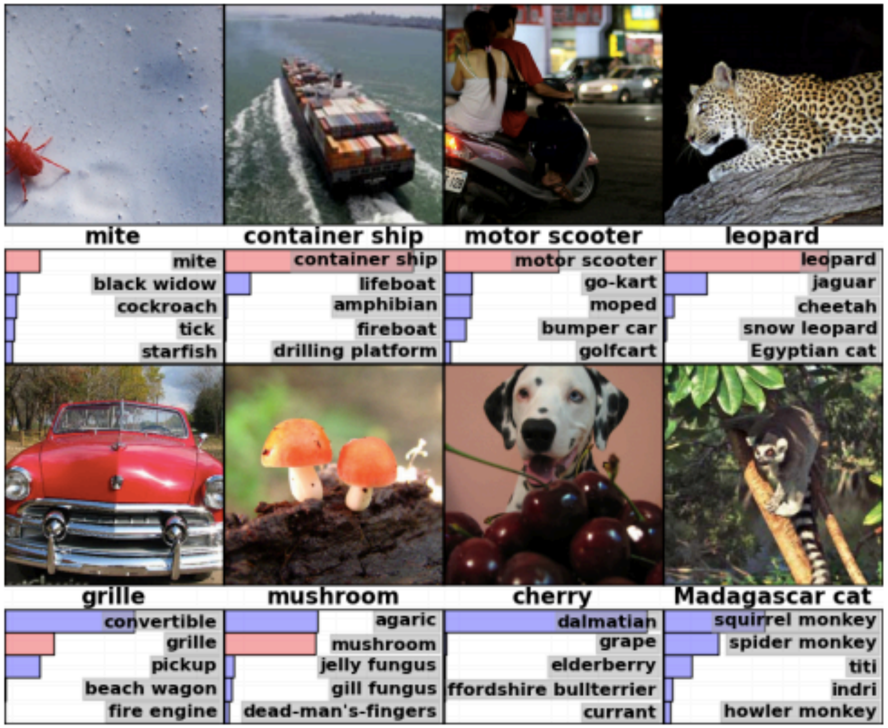
- Visualization of CNN feature representations [Zeiler et al., 2014]
  - VGG-16 [Simonyan et al., 2015]

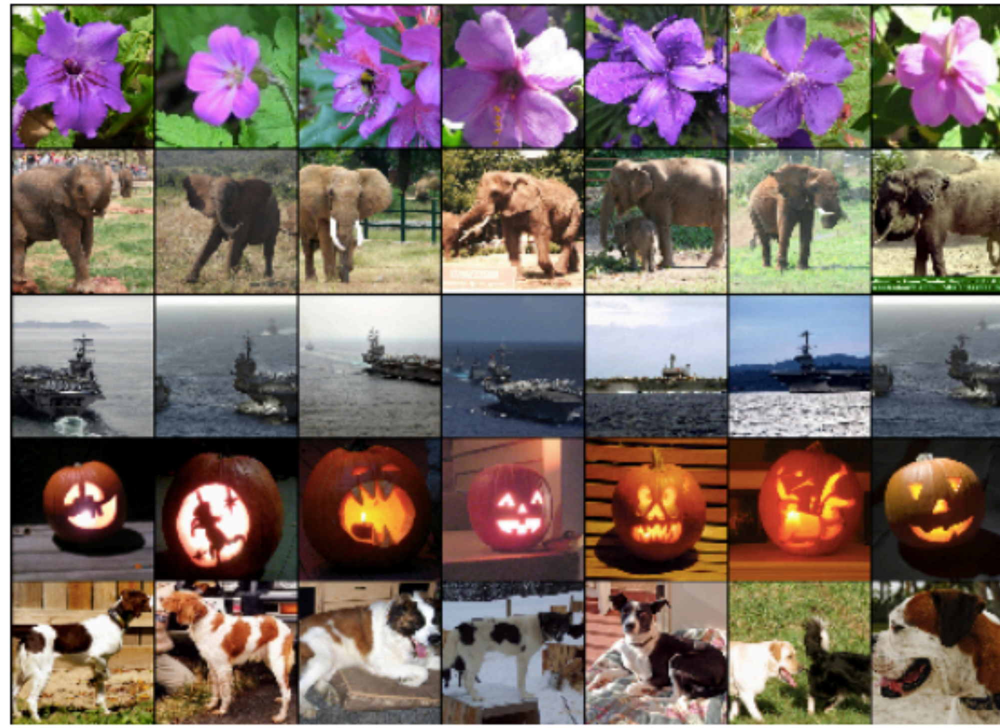


*reference : http://cs231n.stanford.edu/2017/

CNN in Computer Vision: Everywhere

Classification and retrieval [Krizhevsky et al., 2012]





CNN in Computer Vision: Everywhere

Detection [Ren et al., 2015]

Segmentation [Farabet et al., 2013]

*reference : http://cs231n.stanford.edu/2017/

CNN in Computer Vision: Everywhere

Self-driving cars

Human pose estimation [Cae et al., 2017]

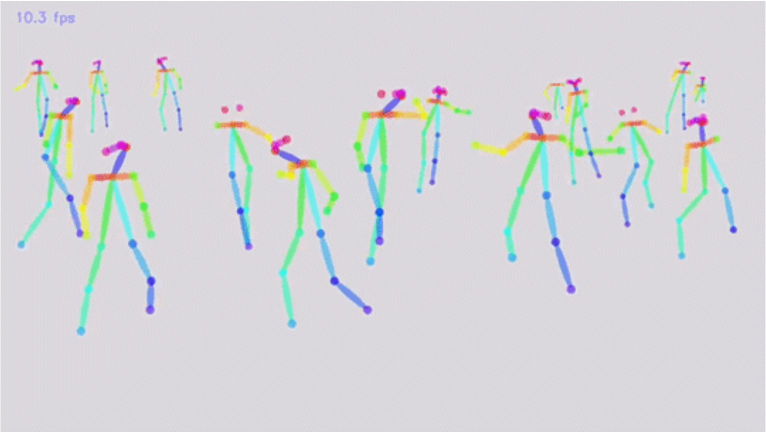


Image captioning [Vinyals et al., 2015][Karpathy et al., 2015]

No errors

A white teddy bear sitting in the grass

Minor errors

A man in a baseball uniform throwing a ball

Somewhat related

A woman is holding a cat in her hand

*reference : http://cs231n.stanford.edu/2017/

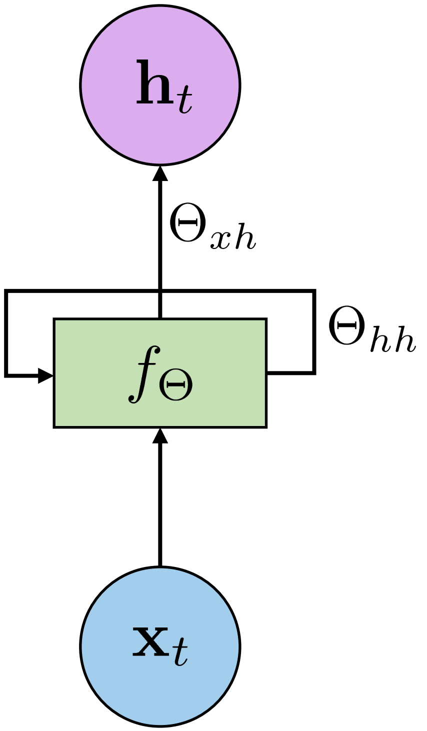
Table of Contents

RNN: Basics

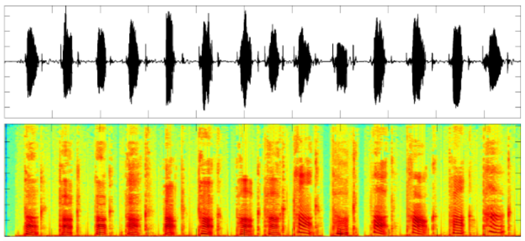
- CNN models spatial invariance information

- **Recurrent Neural Network (RNN)**
  - Models **temporal** information
  - Hidden states as a function of inputs and previous time step information

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t; \Theta)$$



- Temporal information is important in many applications
  - Language
  - Speech
  - Video



RNN: Basics

- Process a sequence of vectors by applying **recurrence formula** at every time step :

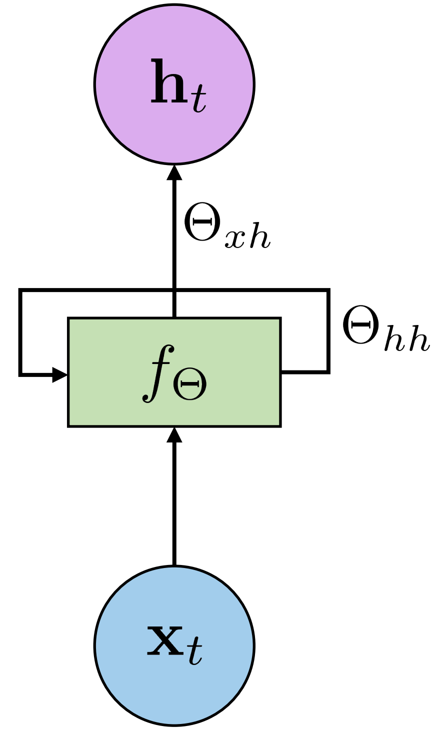
$$\boxed{\mathbf{h}_t} = \boxed{f}(\boxed{\mathbf{h}_{t-1}}, \boxed{\mathbf{x}_t}; \Theta)$$

New state

Old state

Input vector at time step t

Function parameterized by Θ e.g, DNN, CNN



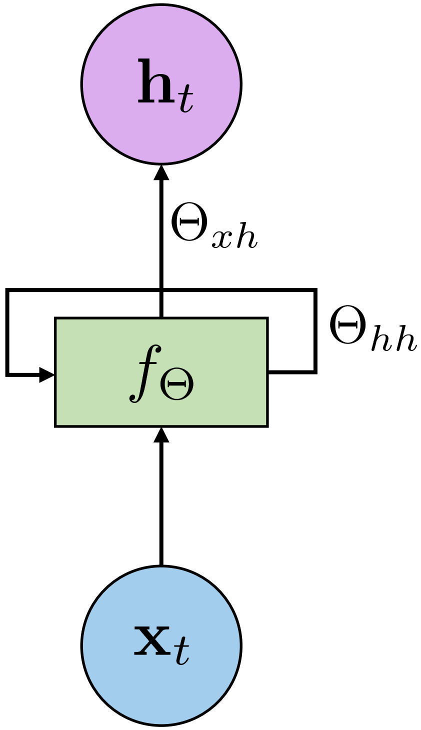
*reference : http://cs231n.stanford.edu/2017/

RNN: Basics

- Process a sequence of vectors by applying **recurrence formula** at every time step :

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t; \Theta)$$

- **Same function** and the **same set of parameters** f_Θ are used at every time step



*reference : http://cs231n.stanford.edu/2017/

RNN: Vanilla RNN

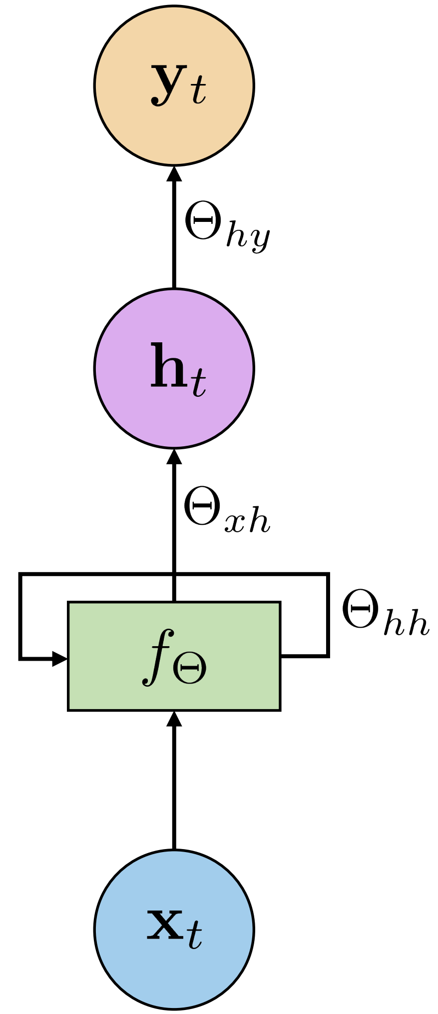
- Simple RNN
  - The state consists of a single “hidden” vector $\mathbf{h}_t$
  - Vanilla RNN (or sometimes called Elman RNN)

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t; \Theta)$$

$$\downarrow$$

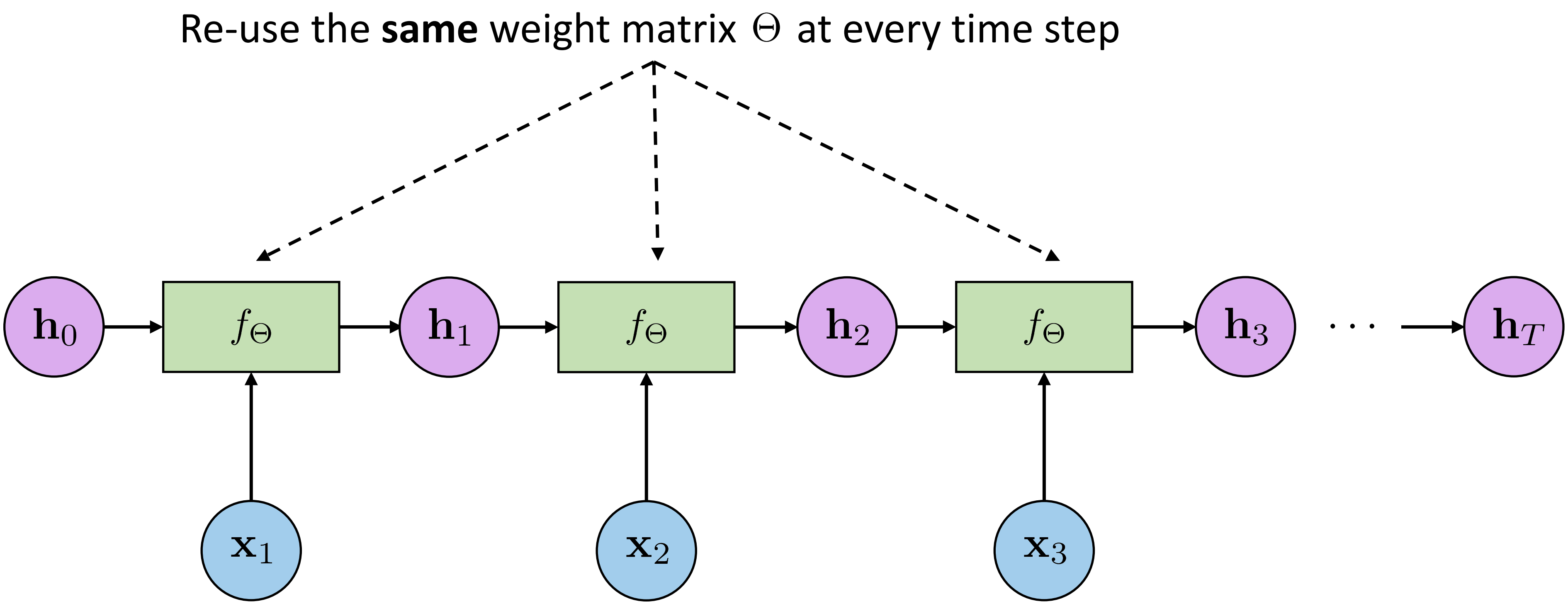
$$\mathbf{h}_t = \tanh(\Theta_{hh}\mathbf{h}_{t-1} + \Theta_{xh}\mathbf{x}_t)$$

$$\mathbf{y}_t = \Theta_{hy}\mathbf{h}_t$$



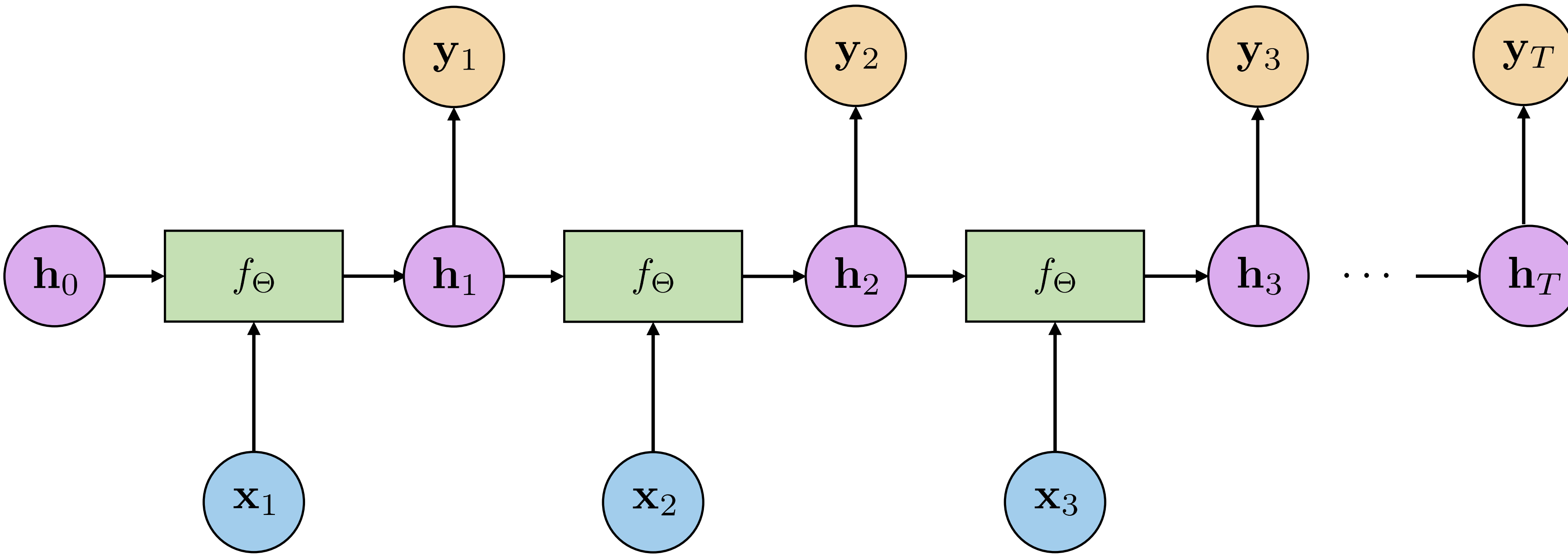
*reference : http://cs231n.stanford.edu/2017/

RNN: Computation Graph



*reference : http://cs231n.stanford.edu/2017/

RNN: Computation Graph (Many to Many)

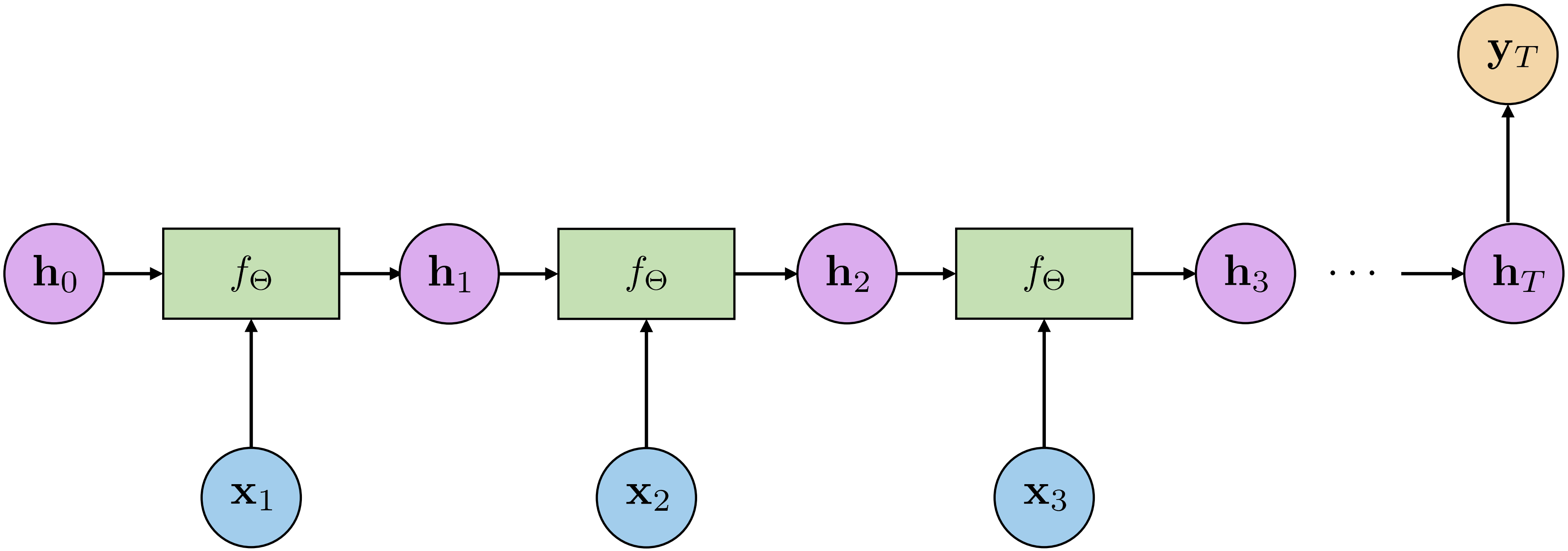


e.g., **Machine Translation**
(Sequence of words → Sequence of words)

Input sentence:	*Translation (PBMT):*	*Translation (GNMT):*	*Translation (human):*
李克強此行將啟動中加總理年度對話機制，與加拿大總理杜魯多舉行兩國總理首次年度對話。	Li Keqiang premier added this line to start the annual dialogue mechanism with the Canadian Prime Minister Trudeau two prime ministers held its first annual session.	Li Keqiang will start the annual dialogue mechanism with Prime Minister Trudeau of Canada and hold the first annual dialogue between the two premiers.	Li Keqiang will initiate the annual dialogue mechanism between premiers of China and Canada during this visit, and hold the first annual dialogue with Premier Trudeau of Canada.

*reference : http://cs231n.stanford.edu/2017/

RNN: Computation Graph (Many to One)



e.g., **Sentiment Classification**
(Sequence of words → sentiment)

Communication and Computing Systems – Prasad et al. (Eds)
© 2017 Taylor & Francis Group, London, ISBN 978-1-138-02952-1

Real time sentiment analysis of tweets using machine learning and semantic analysis

R. Rajput & A. Solanki
School of ICT, Gautam Buddha University, Greater Noida, India

ABSTRACT: The 21st century has brought in lot of products in form of movies, software, video games, for which users make their opinion or judgments and often express it over internet through use of Social Networking Sites (SNS). This has resulted in rise of user generated content. Sentiment Analysis or Opinion Mining is the computational analysis of public opinions and sentiments towards a particular subject. The social media giants like twitter attract millions of online users sharing their opinions in form of 'tweets'. These tweets can be classified into positive or negative on the basis of their sentimental orientation. Methods used to classify the tweets are semantic and machine learning approaches. These methods are discussed and applied in this research work. A method for classification which is combination of Senti WordNet and Machine Learning algorithms is also proposed. All of these methods are then analyzed on the basis of precision, recall and F-measure. The proposed algorithm gives 86% accuracy as compared to 85% accuracy of Multinomial Naïve Bayes and gives accuracy up to 77% for real time data.

1 INTRODUCTION

The vast and ever-growing popularity of Social Networking Sites (SNS) results in overwhelming amount of information in form of user generated data as in opinions, thoughts and beliefs. Twitter is a well-known social media giant. It has millions of active users. It has become a haven for common people to express their opinion over some product, event, person or news. The reviews given by the users can be used to make important business related decisions as they give insight into product reception and quality. People generally are more likely to have a same sentiment value over a product as being held by their peers. The influence of SNS cannot be overlooked or denied. Presently due to excessive amount of data available online, it is becoming hard for an organization to monitor what sentiment is being held by the general public about some specific product or event. This brings in the urge of Sentiment Analysis automation.

2 RELATED WORK

Research in Sentiment Analysis field ranges from classification on document level (Yessenalina et al. 2010) to classifying words and phrases by learning their polarity (Hatzivassiloglou et al. 1997). It has been reported that events that happen in real life indeed have a significant effect on the public sentiment over SNS. The research work by Connor et al. 2010 predicts poll results and Bollen et al. 2011, predicted stock market using twitter trends supports this report. On the basis of such observations, some other research Mishne et al. 2006, made use of the sentiment orientations in SNS to predict movie sales. The pioneering work of figuring out application and challenges in the field of Sentiment Analysis was presented by Pang et al. 2008 and Liu 2012. They mentioned the techniques used to solve each problem in Sentiment Analysis. Ohana et al. 2009 have used SentiWordNet along with the Machine learning classifiers and Pang et al. 2002 have shown use of machine learning algorithms to classify the documents and comparing results of Naïve Bayes, Maximum Entropy, and SVM with varying n-grams. In another research work, Walia et al. 2012, have used Machine Learning based classification approaches along with the Unsupervised Semantic Orientation based algorithms for sentiment analysis of movie review texts.

3 PROPOSED METHODOLOGY

The proposed system takes in consideration two classification techniques. One being the Machine Learning approach and other being the semantic approach. In subsequent subsections, the changes in the parameters of the algorithms and reasons of doing so are discussed.

3.1 *Machine learning approach*

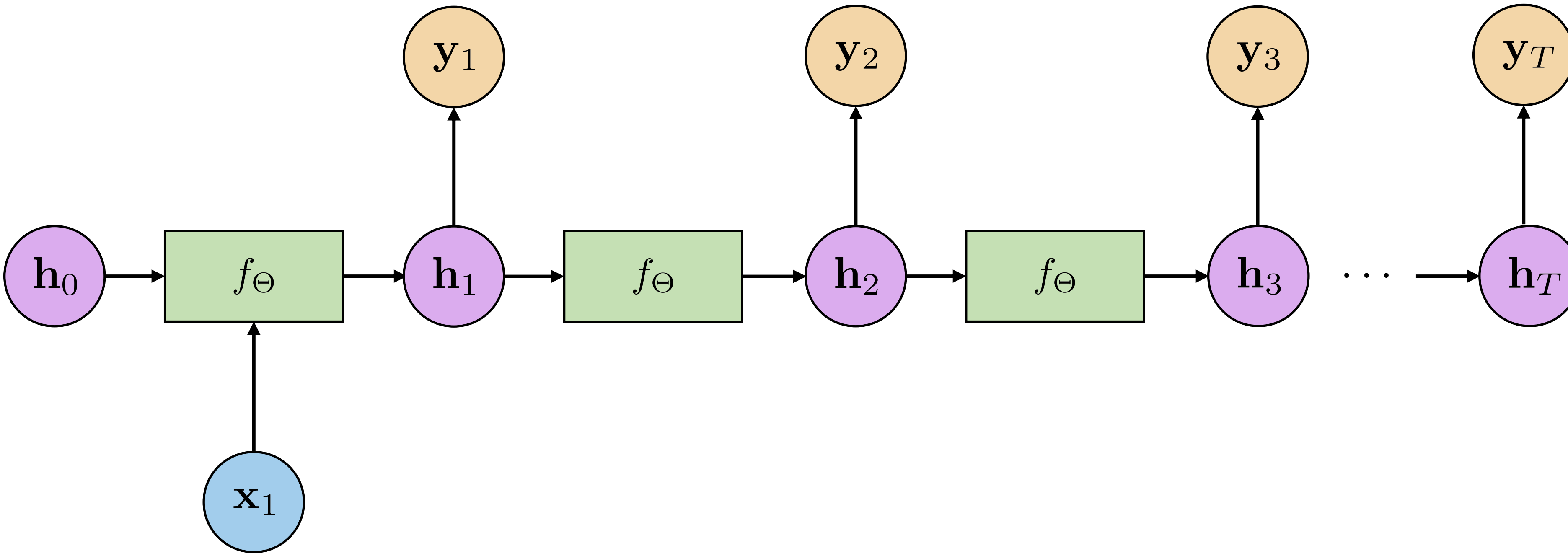
This approach relies on machine learning algorithms to solve sentiment analysis problem as text

687

→ Good paper or not?

*reference : http://cs231n.stanford.edu/2017/

RNN: Computation Graph (One to Many)



e.g., **Image Captioning**
(Image → sequence of words)

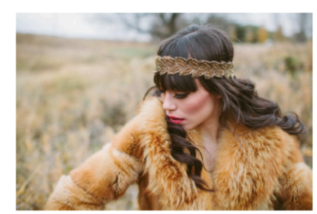
No errors

A white teddy bear sitting in the grass

Minor errors

A man in a baseball uniform throwing a ball

Somewhat related

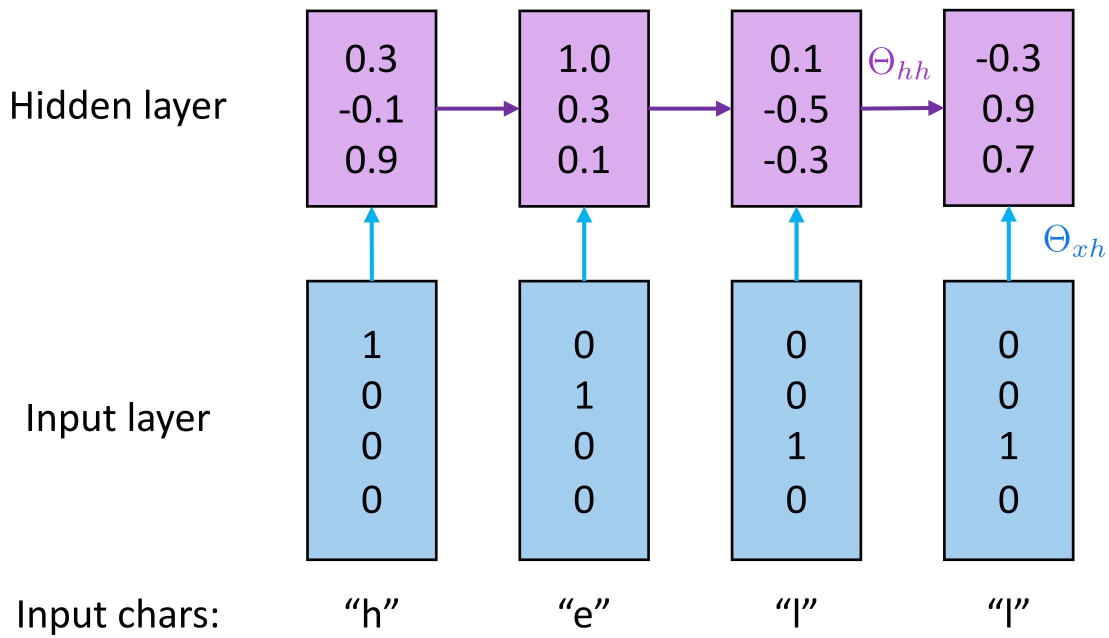


A woman is holding a cat in her hand

*reference : http://cs231n.stanford.edu/2017/

- Character-level language model
- Vocabulary : [h,e,l,o]
- Example training sequence : “hello”

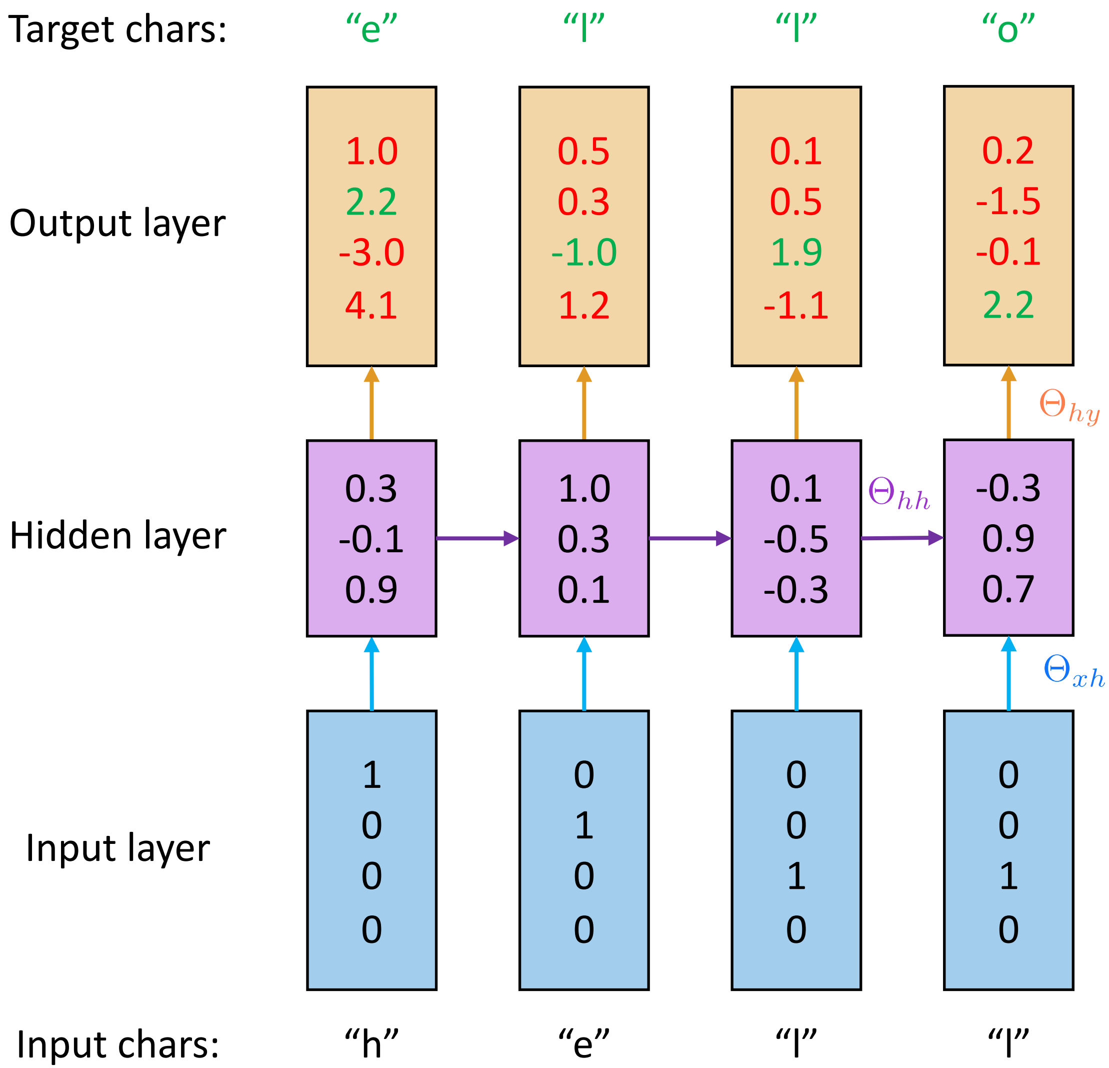
$$\mathbf{h}_t = \tanh(\Theta_{hh}\mathbf{h}_{t-1} + \Theta_{xh}\mathbf{x}_t)$$



*reference : http://cs231n.stanford.edu/2017/

RNN: An Example

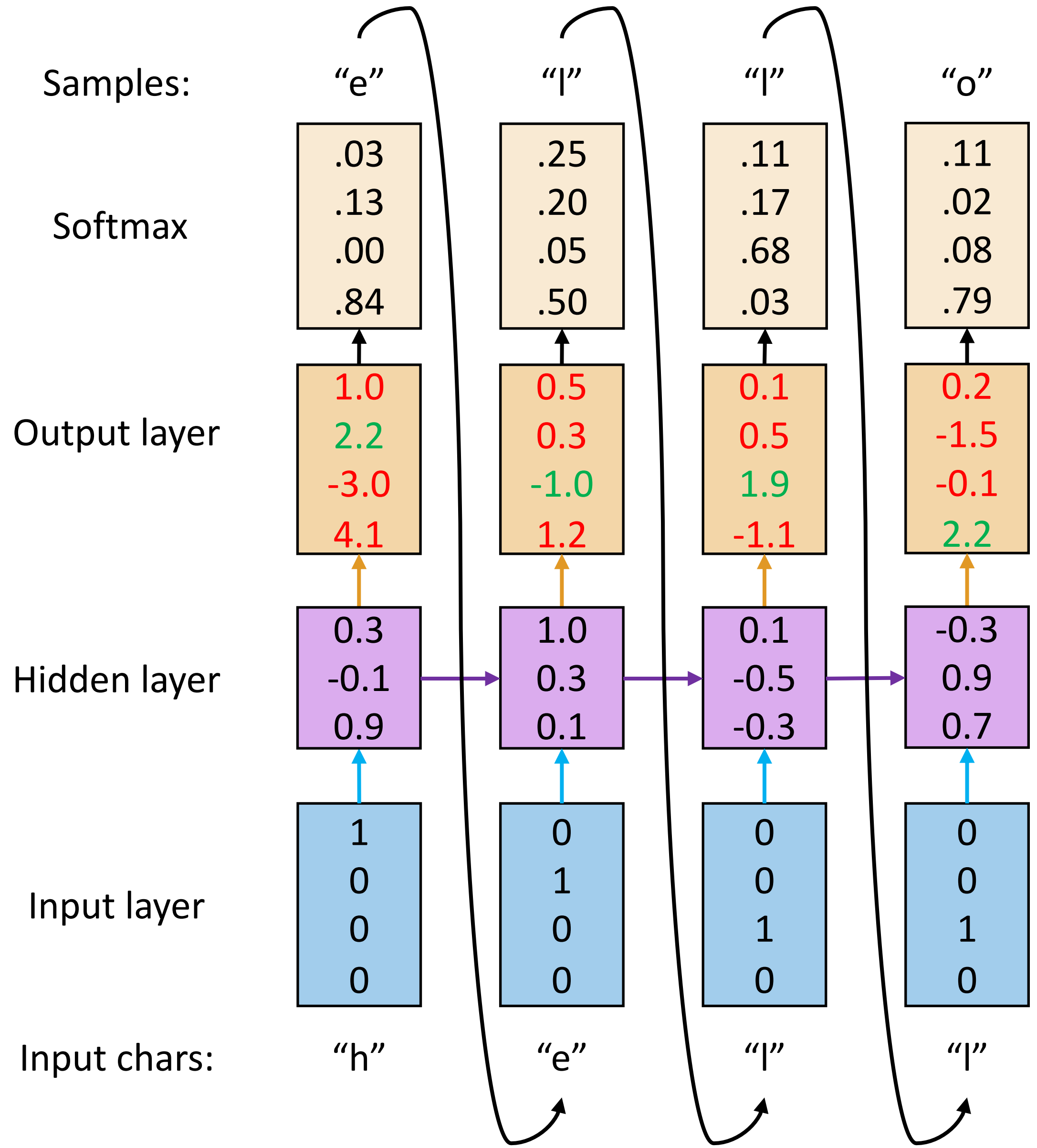
- Character-level language model
- Vocabulary : [h,e,l,o]
- Example training sequence : “hello”



*reference : http://cs231n.stanford.edu/2017/

RNN: An Example

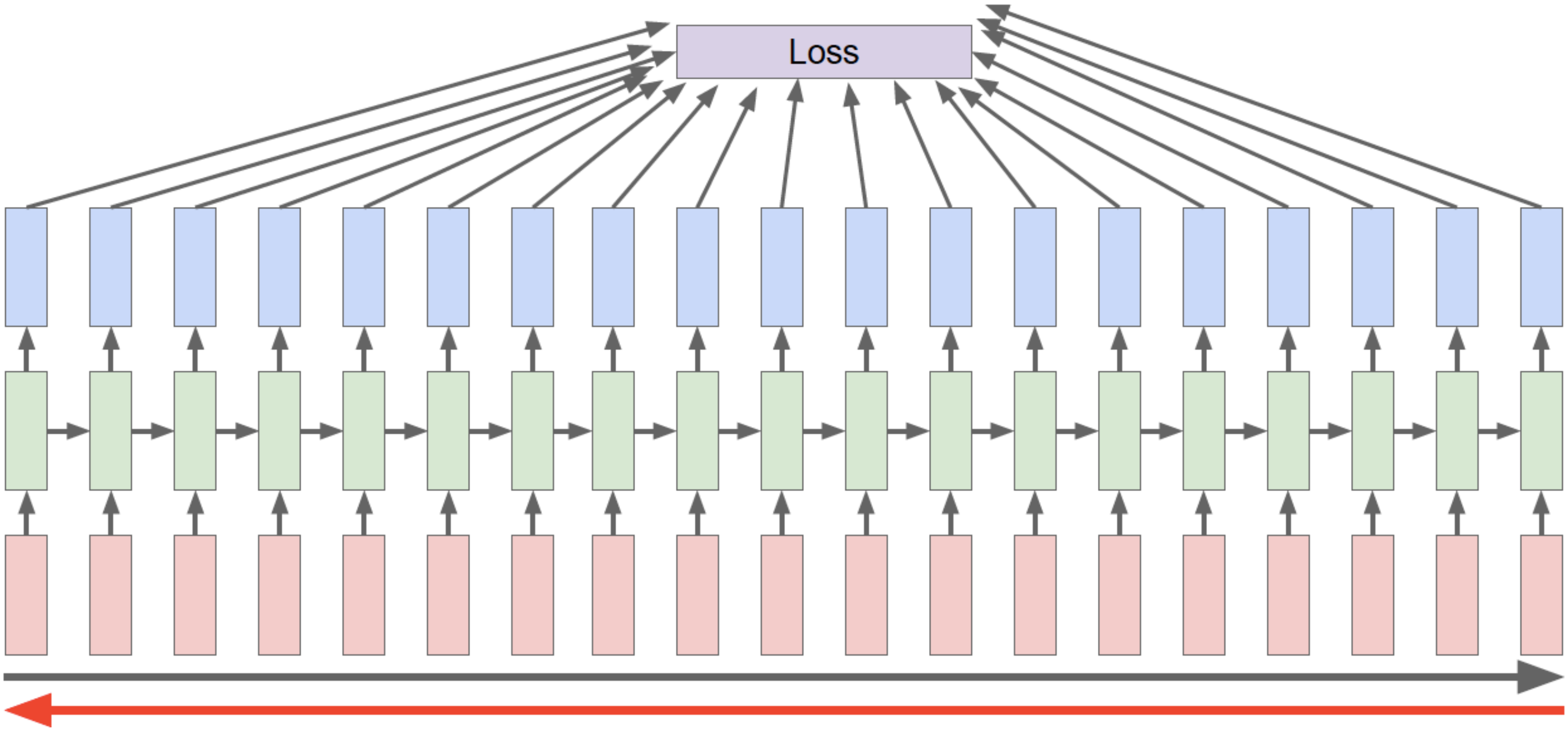
- Character-level language model
- Vocabulary : [h,e,l,o]
- At **test** time, sample character one at a time and feedback to model



*reference : http://cs231n.stanford.edu/2017/

RNN: Backpropagation Through Time (BPTT)

- Backpropagation through time (BPTT)
- Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient



*reference : http://cs231n.stanford.edu/2017/

Contents

1. **Deep Neural Networks (DNN)**
   - Basics
   - Training : Back propagation

2. **Convolutional Neural Networks (CNN)**
   - Basics
   - Convolution and Pooling
   - Some applications

3. **Recurrent Neural Networks (RNN)**
   - Basics
   - Character-level language model (example)

4. **Question**
   - Why is it difficult to train a deep neural network?

Question

- Why is it difficult to train a deep neural network?

- Can we just simply stack multiple layers and train them all?
  - Unfortunately, it does not work well
  - Even if we have infinite amount of computational resource

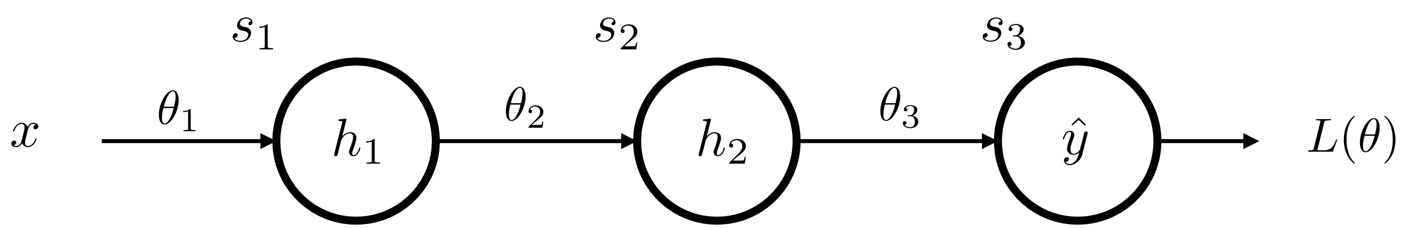
Vanishing gradient **problem :**

- The magnitude of the gradients **shrink exponentially** as we backpropagate through **many layers**
- Since typical activation functions such as sigmoid or tanh are **bounded**
- The phenomenon is called ***vanishing gradient*** problem

Vanishing Gradient Problem

- Why do gradients vanish?
- Think of a simplified 3-layer neural network

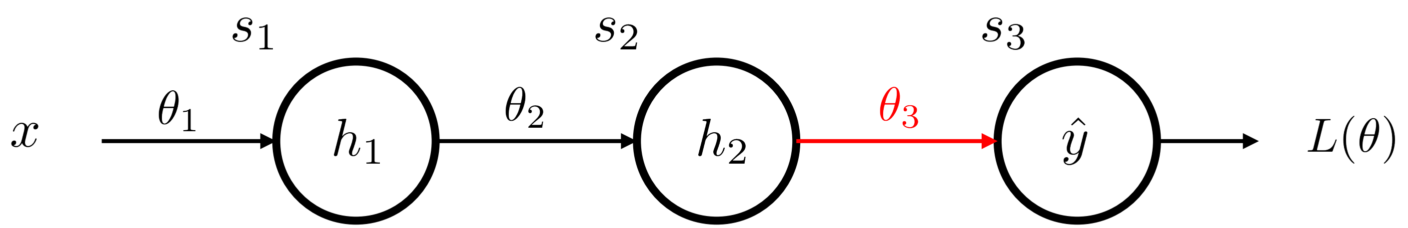
$$\hat{y} = \sigma\left(\theta_3 \sigma\left(\theta_2 \sigma(\theta_1 x)\right)\right)$$



Vanishing Gradient Problem

- Why do gradients vanish?
- Think of a simplified 3-layer neural network

$$\hat{y} = \sigma\left(\theta_3 \sigma\left(\theta_2 \sigma(\theta_1 x)\right)\right)$$



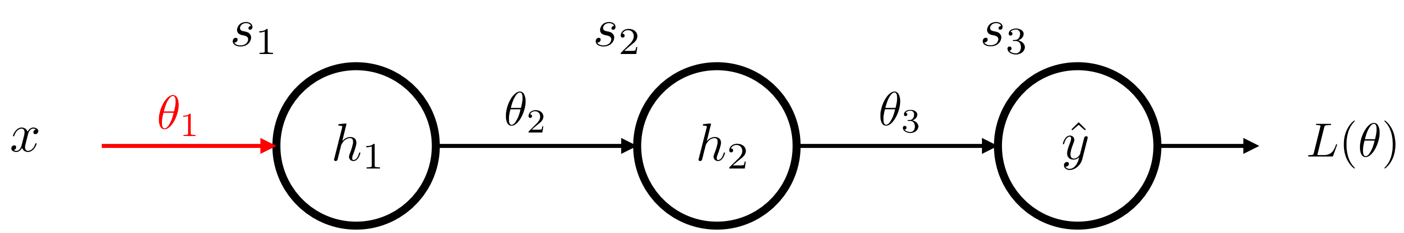
- First, let's update θ_3
  - Calculate the gradient of the loss with respect to θ_3

$$\frac{\partial L}{\partial \theta_3} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial s_3} \frac{\partial s_3}{\partial \theta_3} = \frac{\partial L}{\partial \hat{y}} \sigma'(s_3) h_2$$

Vanishing Gradient Problem

- Why do gradients vanish?
- Think of a simplified 3-layer neural network

$$\hat{y} = \sigma\left(\theta_3 \sigma\left(\theta_2 \sigma(\theta_1 x)\right)\right)$$

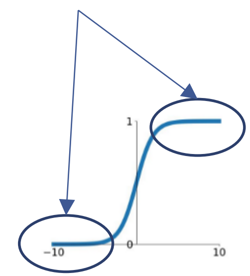


- How about θ_1 ?
  - Calculate the gradient of the loss with respect to θ_1

$$\frac{\partial L}{\partial \theta_1} = \frac{\partial L}{\partial \hat{y}} \boxed{\sigma'(s_3)} h_2 \boxed{\sigma'(s_2)} h_1 \boxed{\sigma'(s_1)} x$$

Gradients < 1

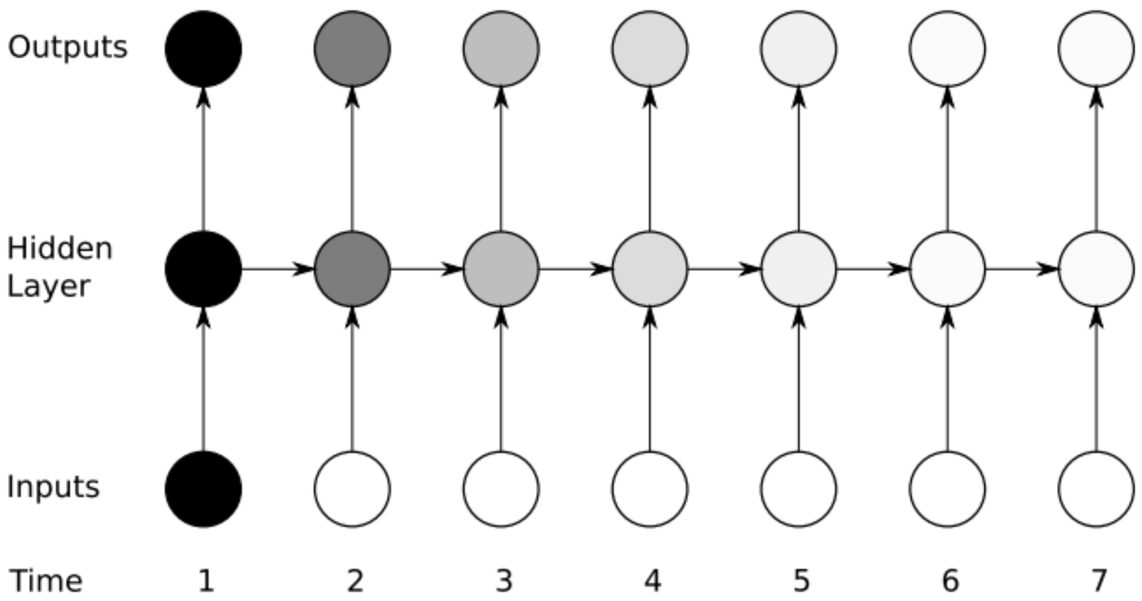
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



Keep multiplying values < 1 will decrease the amount exponentially

Vanishing Gradient Over Time

- This is more problematic in vanilla RNN (with tanh/sigmoid activation)
  - When trying to handle long temporal dependency
  - Similar to previous example, the **gradient vanishes** over time



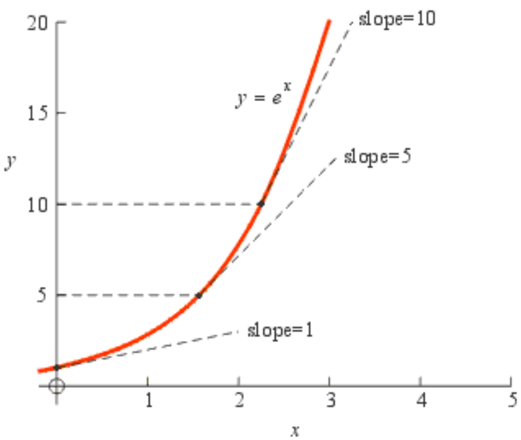
*source :https://mediatum.ub.tum.de/doc/673554/file.pdf

Quiz

- Vanishing gradient problem is critical in training neural network
- Q: Can we just use activation function that has gradients > 1?

Answer for Quiz

- Not really. It will cause another problem so called **exploding gradients.**
- Let's consider if we use exponential activation function:
  - The magnitude of gradient is always larger than 1 when input > 0
  - If output of the networks are positive, then the gradients to update θ_1 will explode



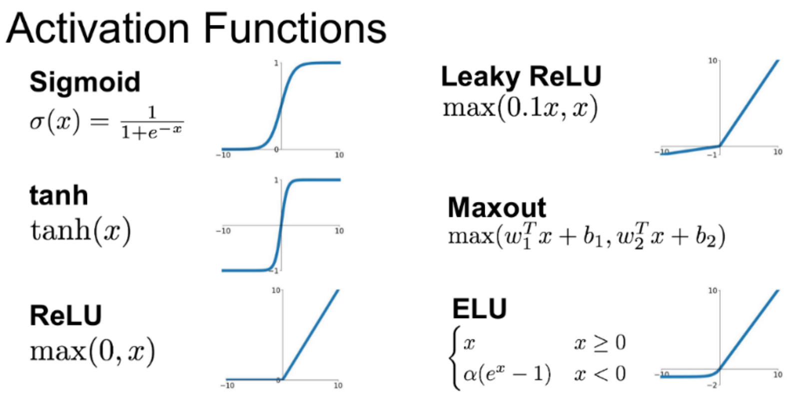
Gradients > 1

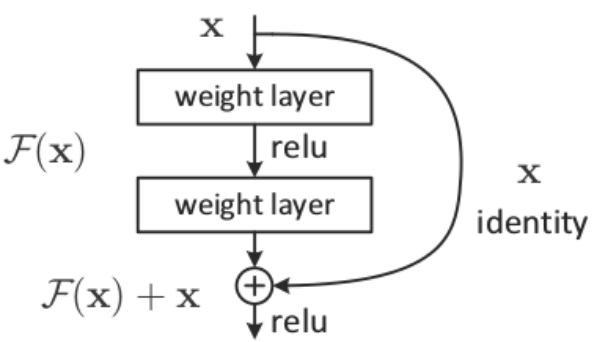
$$\frac{\partial L}{\partial \theta_1} = \frac{\partial L}{\partial \hat{y}} \boxed{\sigma'(s_3)} h_2 \boxed{\sigma'(s_2)} h_1 \boxed{\sigma'(s_1)} x$$

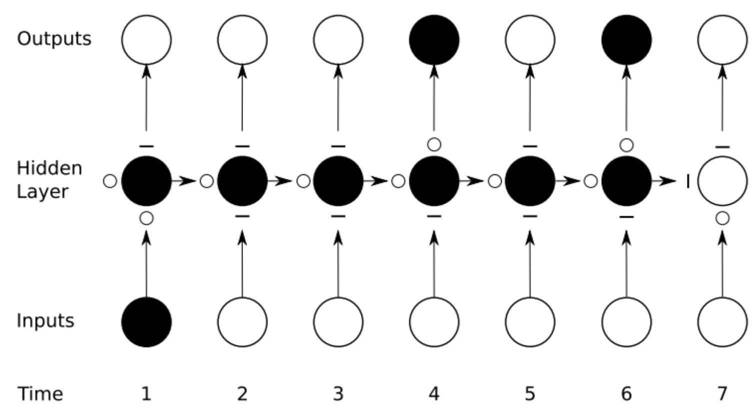
- This will cause the training very **unstable**
  - The weights will be updated in very large amount, resulting in NaN values
  - Very critical problem in training neural networks

How Can We Overcome Vanishing Gradient Problems?

- Possible solutions
  - Activation functions
  - CNN: Residual networks [He et al., 2016]
  - RNN: LSTM (Long Short-Term Memory)







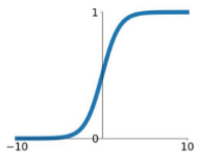
LSTM (Long Short-Term Memory)

*source
https://mediatum.ub.tum.de/doc/673554/file.pdf
https://medium.com/@shrutijadon10104776/survey-on-activation-functions-for-deep-learning-9689331ba092

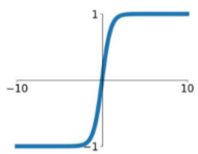
Solving Vanishing Gradient: Activation Functions

- Use different activation functions that are not bounded:
  - Recent works largely use **ReLU** or their variants
  - No saturation, easy to optimize

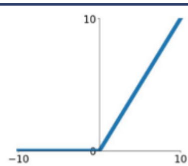
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



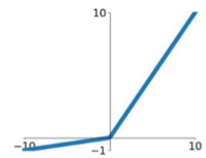
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

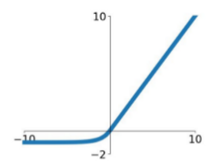


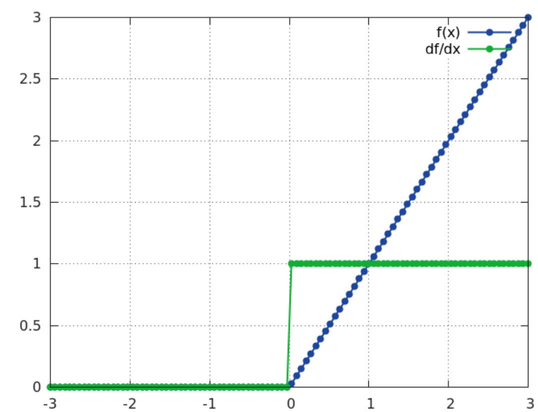
Leaky ReLU
$\max(0.1x, x)$



Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$





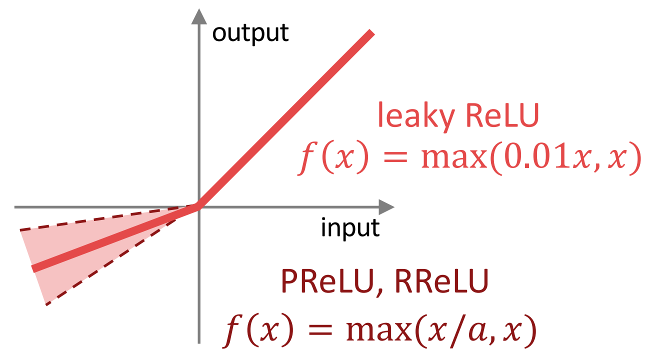
*source: https://medium.com/@shrutijadon10104776/survey-on-activation-functions-for-deep-learning-9689331ba092

Solving Vanishing Gradient: Activation Functions

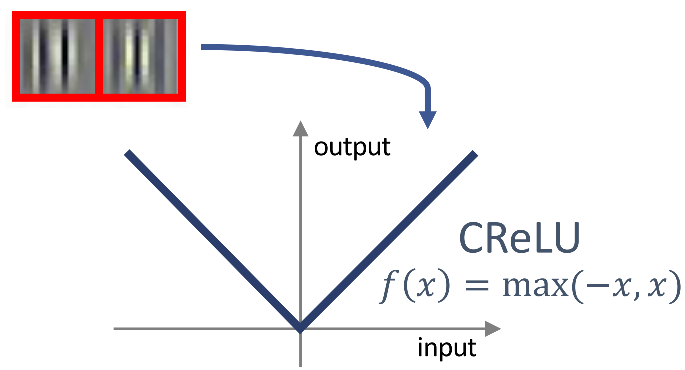
- Several generalizations of ReLU
  - Leaky ReLU [Maas et. al., 2013]: Introducing non-zero gradient for ‘dying ReLUs’
  - Parameteric ReLU (PReLU) [He et al., 2015]: Additional learnable parameter a on leaky ReLUs.
  - Randomized ReLU (RReLU) [Xu et al., 2015]: Samples parameter a from uniform distribution.

Activation	Training Error	Test Error
ReLU	0.00318	0.1245
Leaky ReLU, $a = 100$	0.0031	0.1266
Leaky ReLU, $a = 5.5$	0.00362	**0.1120**
PReLU	0.00178	0.1179
RReLU ($y_{ji} = x_{ji}/\frac{l+u}{2}$)	0.00550	**0.1119**

Table 3. Error rate of CIFAR-10 Network in Network with different activation function

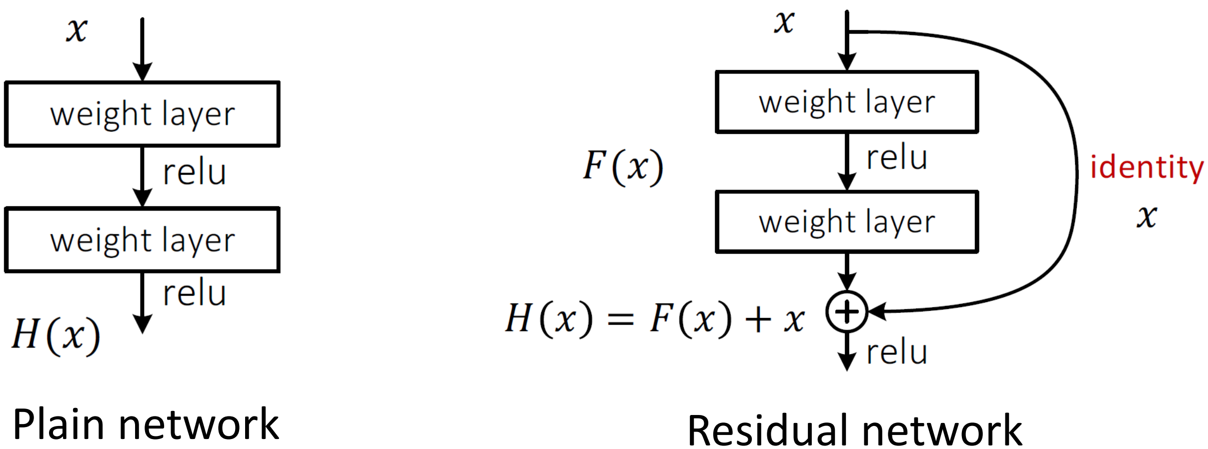


- Concatenated ReLU (CReLU) [Shang et al., 2016]
  - ‘Opposite pairs’ of filters found in CNN
    - Needs to learn twice the information
  - Two-sided ReLU



Solving Vanishing Gradient: Residual Networks

- Residual networks (ResNet [He et al., 2016])
  - Feed-forward NN with "**shortcut connections**"
  - Can preserve gradient flow throughout the entire depth of the network
  - Possible to train **more than 100 layers** by simply stacking residual blocks

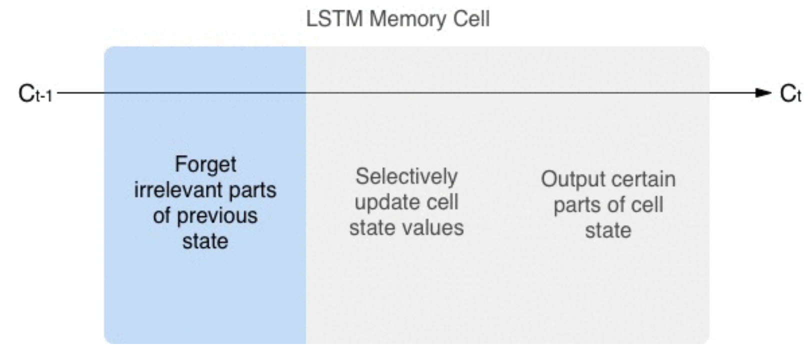


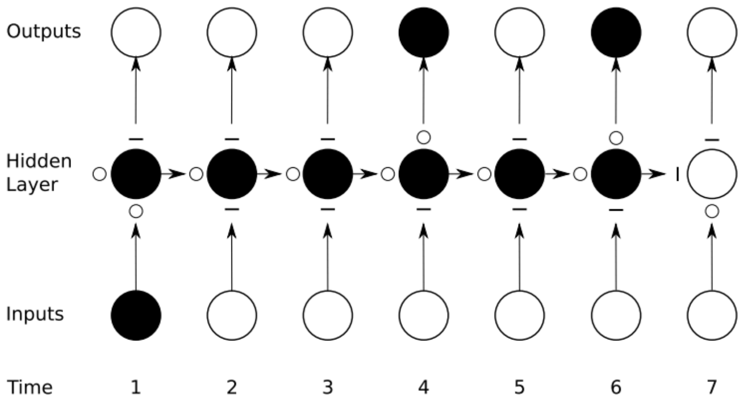
Solving Vanishing Gradient: LSTM and GRU

- LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Units)
  - Specially designed RNN which can **remember** information for much **longer period**

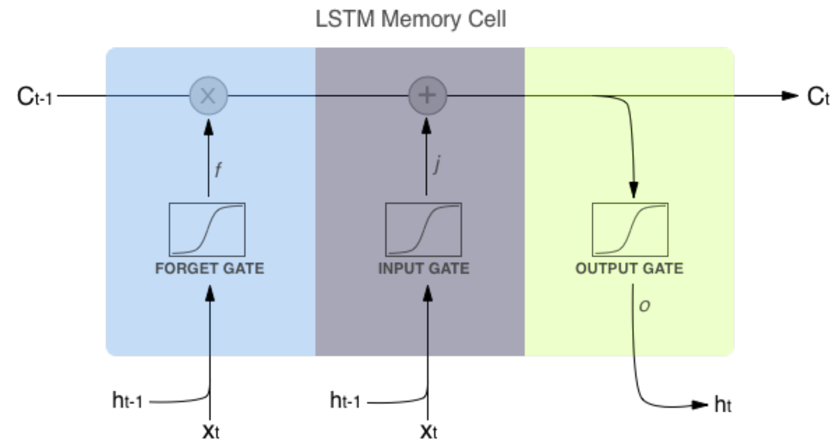
3 main steps:

- **Forget irrelevant parts** of previous state
- **Selectively update** the cell state based on the new input
- **Selectively decide** what part of the cell state to output as the new hidden state





Preservation of gradient information in LSTM



*source :
http://harinisuresh.com/2016/10/09/lstms/
https://mediatum.ub.tum.de/doc/673554/file.pdf

References

- [Nair et al., 2010] "Rectified linear units improve restricted boltzmann machines." ICML 2010.
  link : https://dl.acm.org/citation.cfm?id=3104425
- [Krizhevsky et al., 2012] "Imagenet classification with deep convolutional neural networks." NIPS 2012
  link : https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
- [Maas et al., 2013] "Rectifier nonlinearities improve neural network acoustic models." ICML 2013.
  link : https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf
- [Farabet et al., 2013] "Learning hierarchical features for scene labeling." IEEE transactions on PAMI 2013
  link : https://www.ncbi.nlm.nih.gov/pubmed/23787344
- [Zeiler et al., 2014] "Visualizing and understanding convolutional networks." ECCV 2014.
  link : https://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf
- [Simonyan et al., 2015] "Very deep convolutional networks for large-scale image recognition." ICLR 2015
  link : https://arxiv.org/abs/1409.1556
- [Ren et al., 2015] "Faster r-cnn: Towards real-time object detection with region proposal networks." NIPS 2015
  link : https://arxiv.org/abs/1506.01497
- [Vinyals et al., 2015] "Show and tell: A neural image caption generator." CVPR 2015.
  link : https://arxiv.org/abs/1411.4555
- [Karpathy et al., 2015] "Deep visual-semantic alignments for generating image descriptions." CVPR 2015
  link : https://cs.stanford.edu/people/karpathy/cvpr2015.pdf
- [He et al., 2015] "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." ICCV 2015.
  link : https://arxiv.org/abs/1502.01852

References

- [Xu et al., 2015] "Empirical evaluation of rectified activations in convolutional network." *arXiv preprint,* 2015.
  link : https://arxiv.org/abs/1505.00853
- [Shang et al., 2016] "Understanding and improving convolutional neural networks via concatenated rectified linear units." ICML 2016.
  link : https://arxiv.org/abs/1603.05201
- [He et al., 2016] "Deep residual learning for image recognition." CVPR 2016
  link : https://arxiv.org/abs/1512.03385
- [Cae et al., 2017] "Realtime multi-person 2d pose estimation using part affinity fields.", CVPR 2017
  link : https://arxiv.org/abs/1611.08050
- [Fei-Fei and Karpathy, 2017] "CS231n: Convolutional Neural Networks for Visual Recognition"*, 2017*. (Stanford University)
  link : http://cs231n.stanford.edu/2017/