# TensorFlow 딥러닝 실습

## 2016.9.24
## 인지과학산업협회 튜토리얼

Eun-Sol Kim

Biointelligence Laboratory
Department of Computer Science and Engineering
Seoul National Univertisy

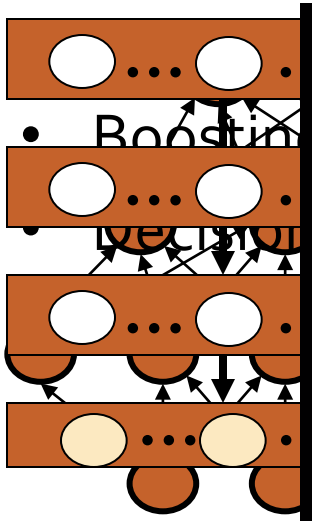
http://bi.snu.ac.kr

# 개요

- 간단한 딥러닝 소개
- 다양한 딥러닝 프레임워크 비교 설명
- TensorFlow의 특징 및 문법
- TensorFlow 실습
  - TensorFlow 기본 문법
  - 간단한 NN 구현
- Recurrent Neural Network 소개
- Recurrent Neural Network 구현

# Neural Network & Convolutional Neural Network

# History of Neural Network Research



| Rank | Name | Error rate | Description |
|---|---|---|---|
| 1 | **U. Toronto** | 0.15315 | Deep Conv Net |
| 2 | U. Tokyo | 0.26172 | Hand-crafted features and learning models. Bottleneck. |
| 3 | U. Oxford | 0.26979 | |
| 4 | Xerox/INRIA | 0.27058 | |

## Deep Networks Advance State of Art in Speech

Deep Learning leads to breakthrough in speech recognition at MSR.

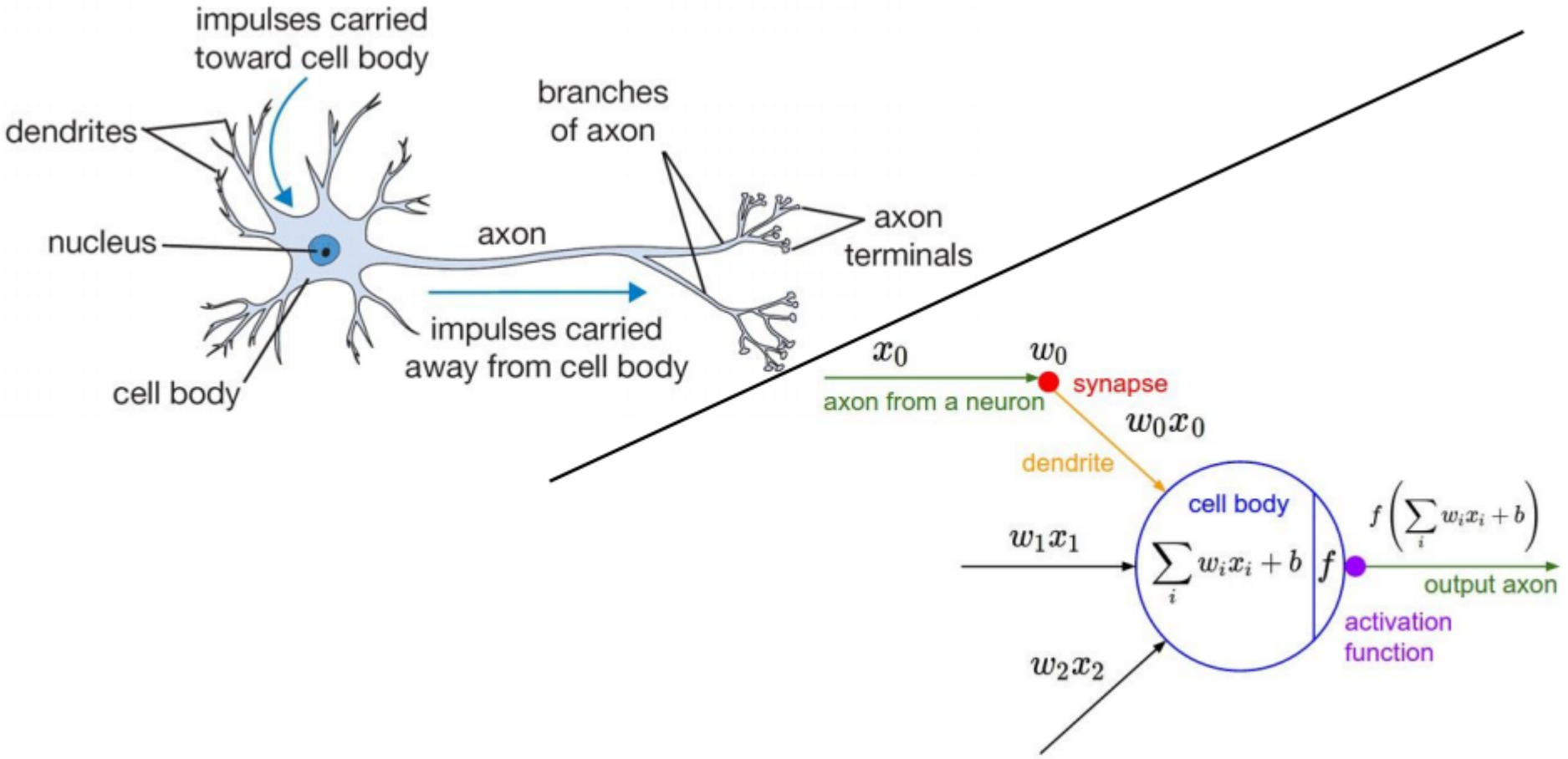
# Classification Problem

- 데이터 x가 주어졌을 때 해당되는 레이블 y를 찾는 문제
  - ex1) x: 사람의 얼굴 이미지, y: 사람의 이름
  - ex2) x: 혈당 수치, 혈압 수치, 심박수, y: 당뇨병 여부
  - ex3) x: 사람의 목소리, y: 목소리에 해당하는 문장

- x: D차원 벡터, y: 정수 (Discrete)

- 대표적인 패턴 인식 알고리즘
  - Support Vector Machine
  - Decision Tree
  - K-Nearest Neighbor
  - Multi-Layer Perceptron (Artificial Neural Network; 인공신경망)

# Perceptron (1/3)

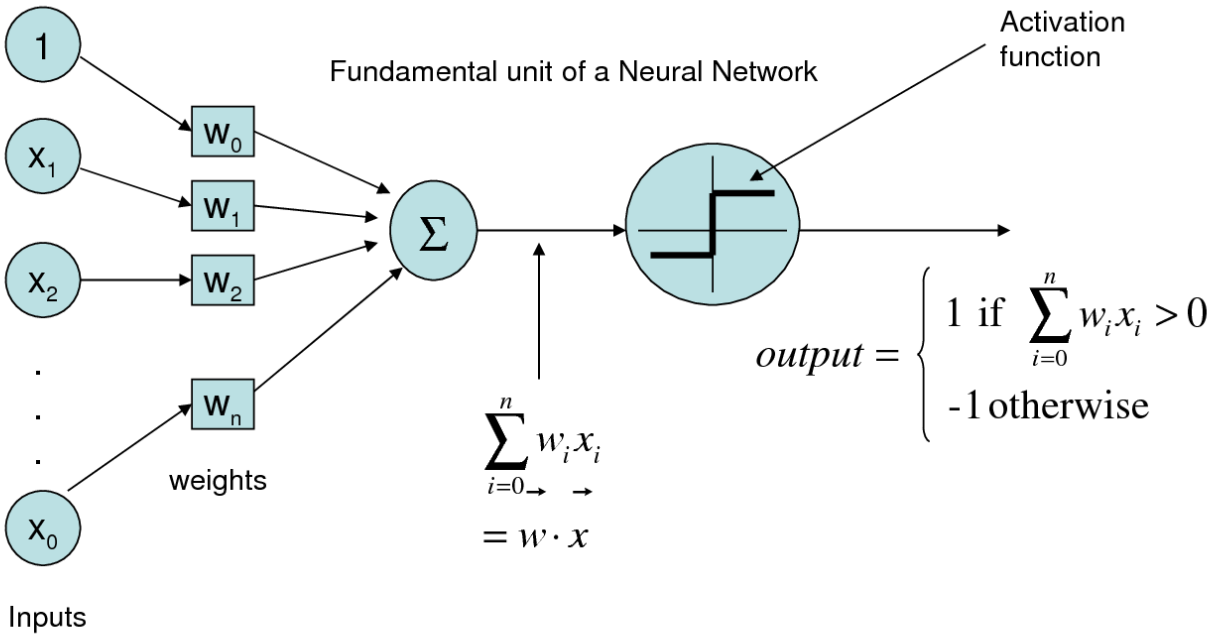



http://cs231n.stanford.edu/slides/winter1516_lecture4.pdf

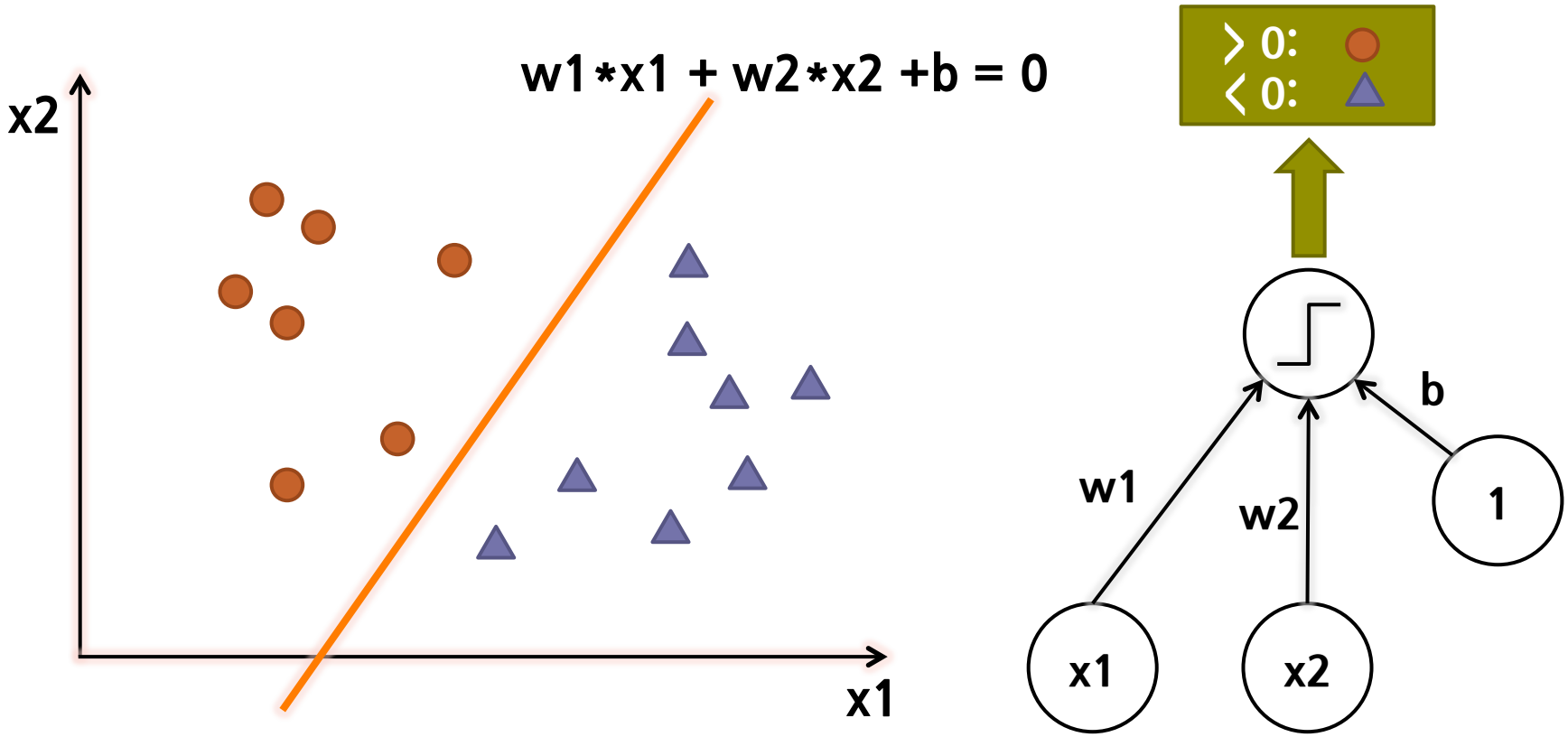
# Perceptron (2/3)

***Artificial Neural Networks***

***The Perceptron***

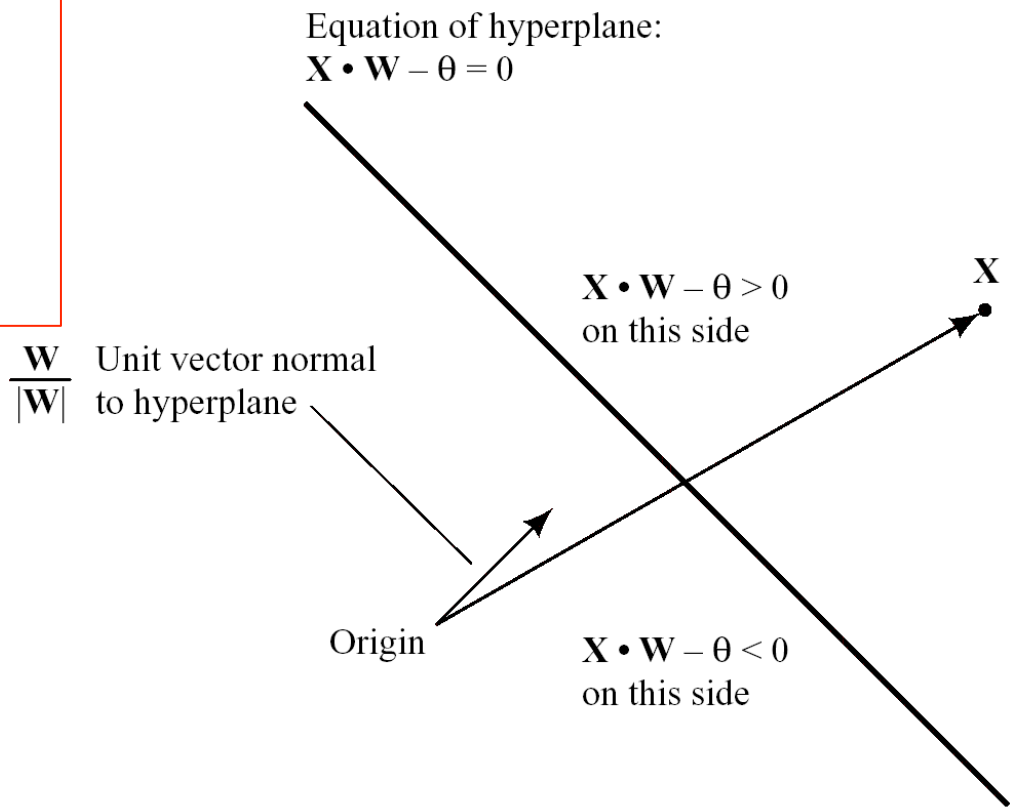


# Perceptron (3/3)

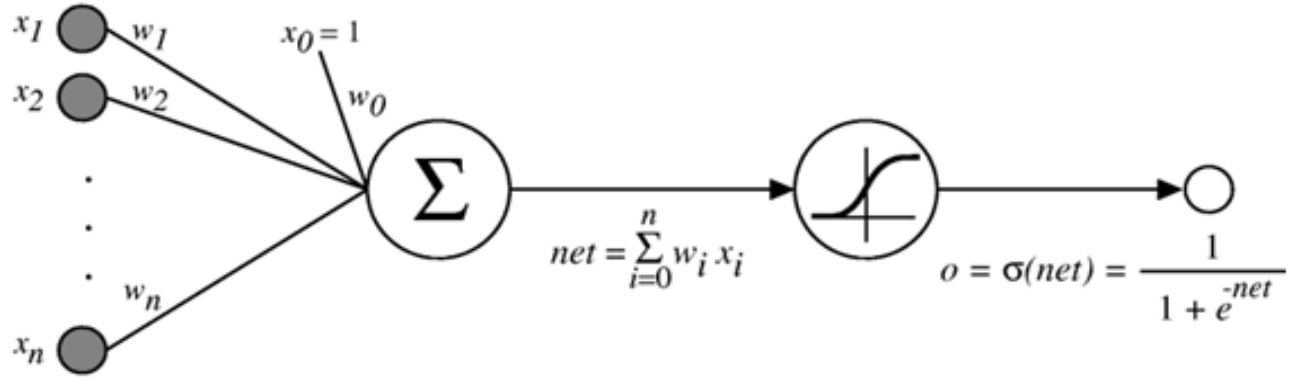


# Parameter Learning in Perceptron

**start**:
The weight vector w is generated randomly
**test**:
A vector $x \in P \cup N$ is selected randomly,
If $x \in P$ and $w \cdot x > 0$ goto test,
If $x \in P$ and $w \cdot x \leq 0$ goto add,
If $x \in N$ and $w \cdot x < 0$ go to test,
If $x \in N$ and $w \cdot x \geq 0$ go to subtract.
**add**:
Set w = w+x, goto test
**subtract**:
Set w = w-x, goto test

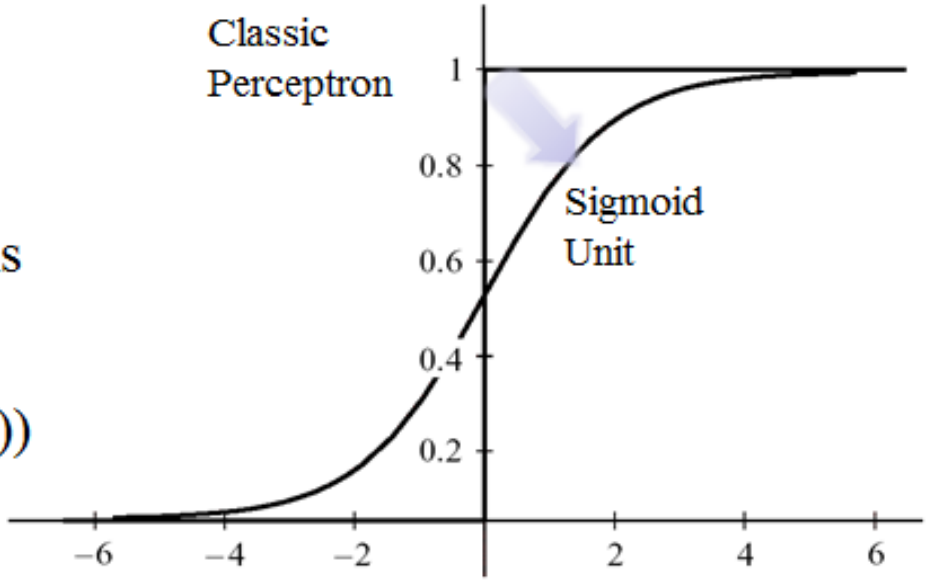


# Sigmoid Unit




Sigmoid function is **Differentiable**

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$$



# Learning Algorithm of Sigmoid Unit

- Loss Function

Target, Unit Output

$$\varepsilon = (d - f)^2$$

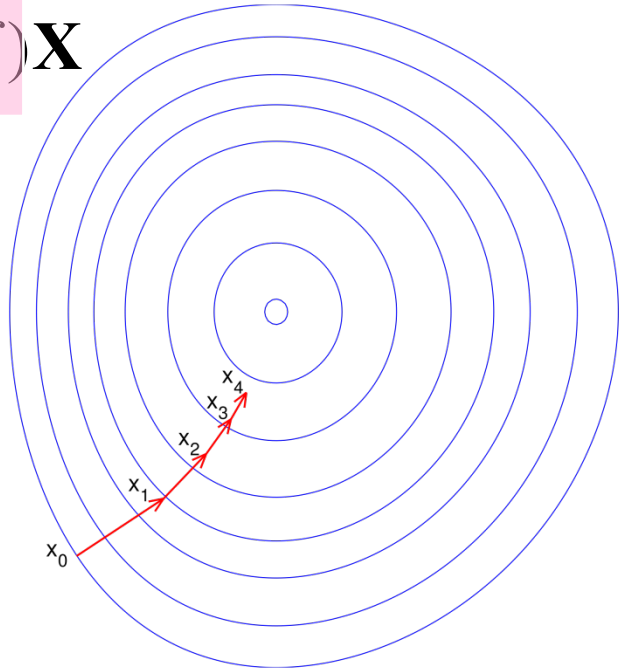
- Gradient Descent Update

$$\frac{\partial \varepsilon}{\partial \mathbf{W}} = -2(d - f)\frac{\partial f}{\partial s}\mathbf{X} = -2(d - f)f(1 - f)\mathbf{X}$$

$$f(s) = 1/(1 + e^{-s})$$

$$f'(s) = f(s)(1 - f(s))$$

$$\mathbf{W} \leftarrow \mathbf{W} + c(d - f)f(1 - f)\mathbf{X}$$



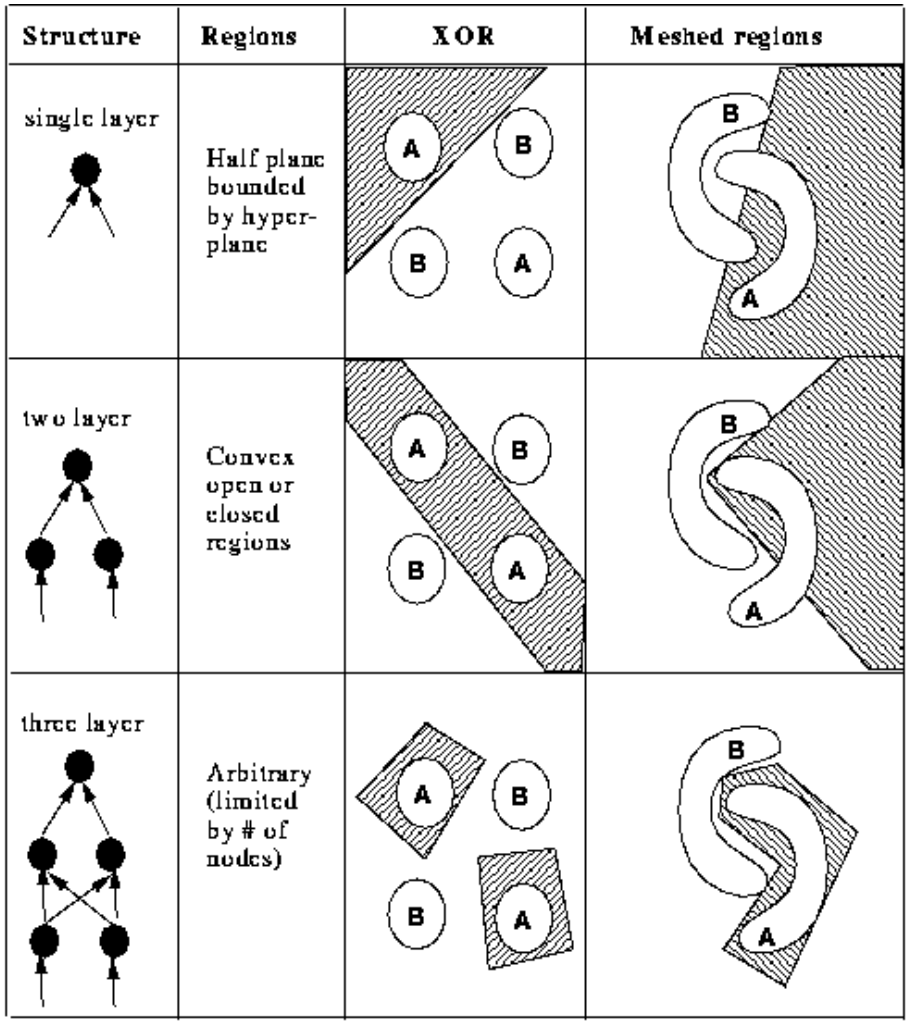
# Need for Multiple Units and Multiple Layers

- Multiple boundaries are needed (e.g. XOR problem)

→ Multiple Units

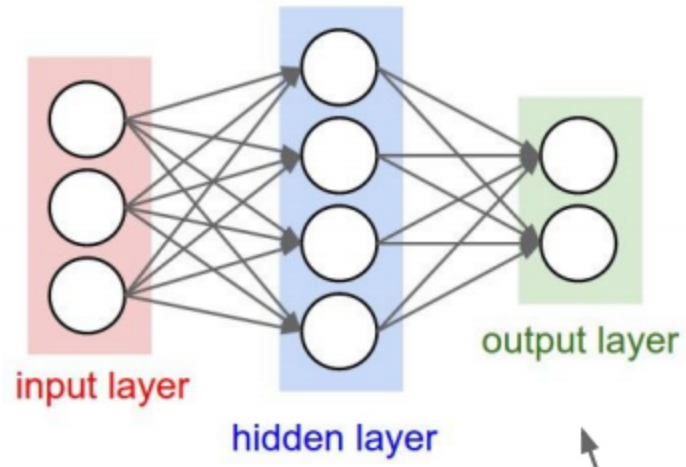
- More complex regions are needed (e.g. Polygons)

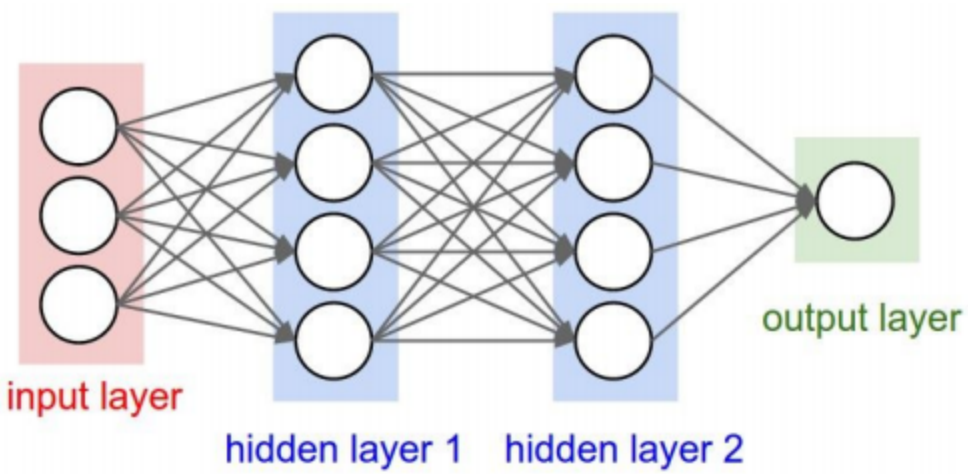
→ Multiple Layers




http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

# Structure of Multilayer Perceptron



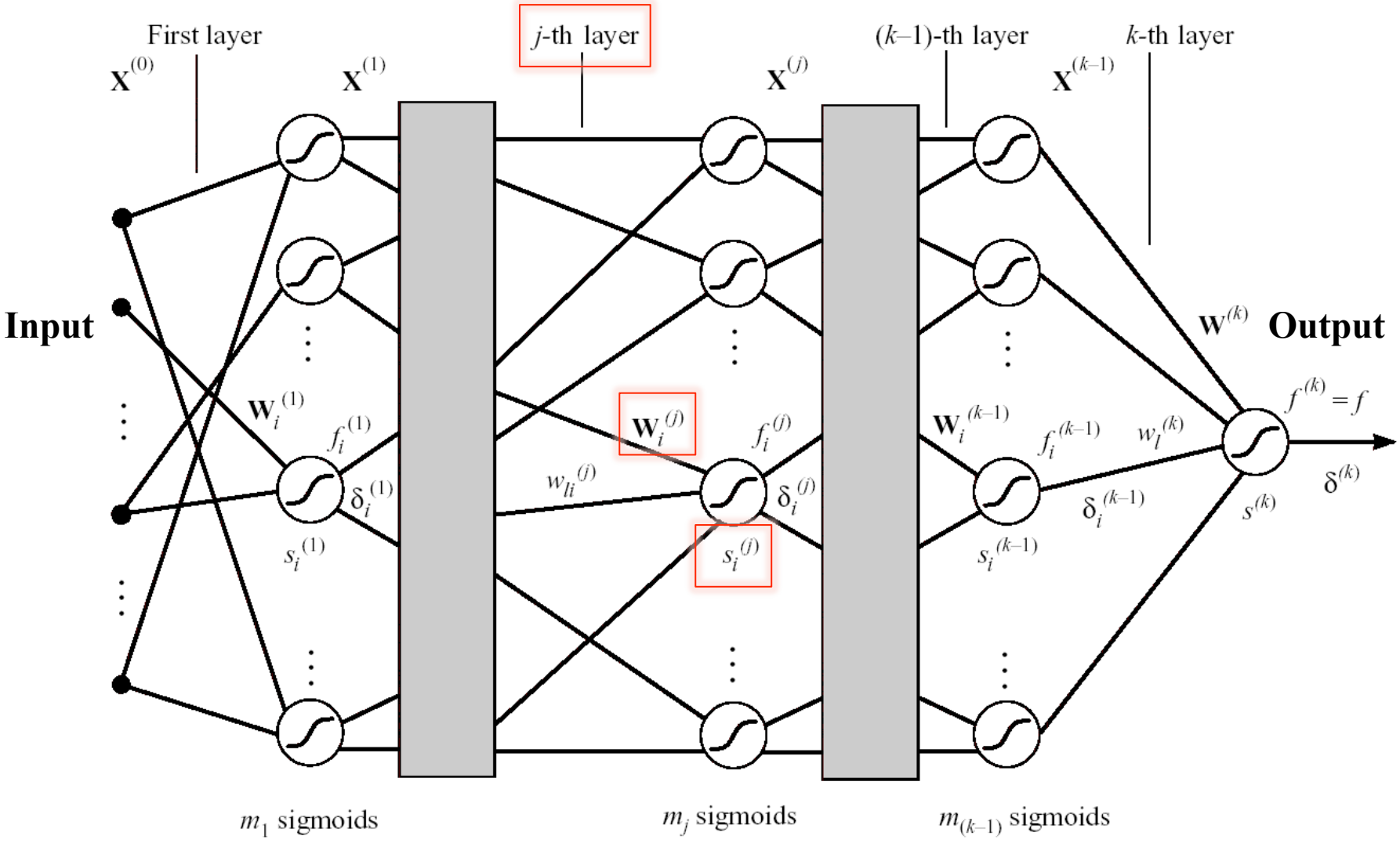

"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"Fully-connected" layers

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

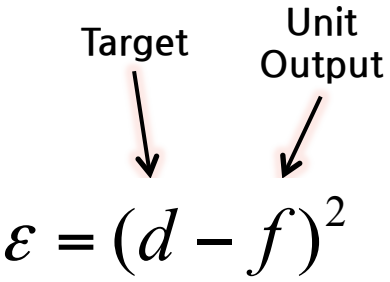
http://cs231n.stanford.edu/slides/winter1516_lecture4.pdf

# Structure of Multilayer Perceptron (MLP; Artificial Neural Network)

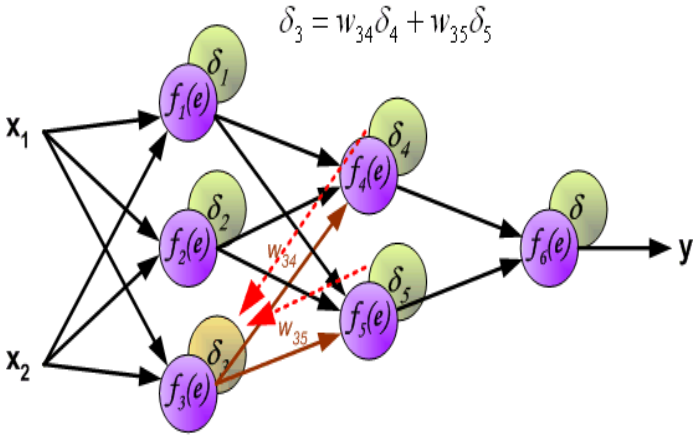


# Learning Parameters of MLP

- Loss Function
  - We have the same Loss Function
  - But the # of parameters are now much more (Weight for **each layer** and **each unit**)
  - To use Gradient Descent, we need to calculate the **gradient for all the parameters**

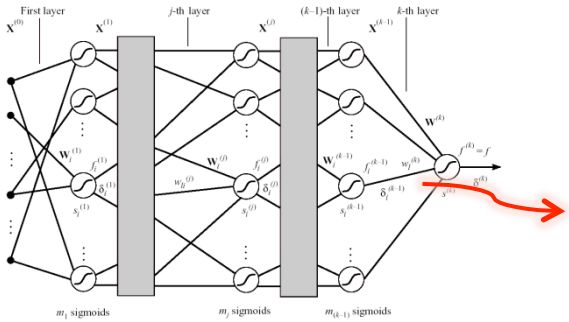



- Recursive Computation of Gradients
  - Computation of loss-gradient of **the top-layer** weights is **the same** as before
  - Using the **chain rule**, we can compute the loss-gradient of lower-layer weights **recursively (Back Propagation)**



# Back Propagation Learning Algorithm (1/3)

- Gradients of top-layer weights and update rule




$$\varepsilon = (d - f)^2$$

$$\frac{\partial \varepsilon}{\partial \mathbf{W}} = -2(d - f)\frac{\partial f}{\partial s}\mathbf{X} = -2(d - f)f(1 - f)\mathbf{X}$$

**Gradient Descent update rule**

$$\mathbf{W} \leftarrow \mathbf{W} + c(d - f)f(1 - f)\mathbf{X}$$

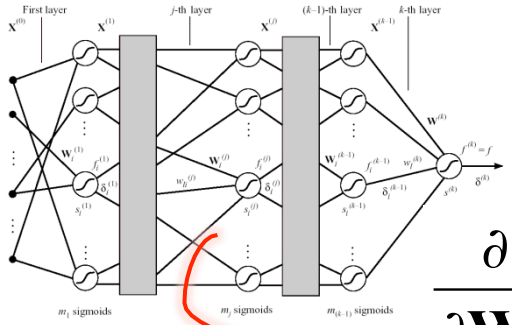
- Store intermediate value delta for later use of chain rule

$$\delta^{(k)} = \frac{\partial \varepsilon}{\partial s_i^{(j)}} = (d - f)\frac{\partial f}{\partial s_i^{(j)}}$$

$$= (d - f)f(1 - f)$$

# Back Propagation Learning Algorithm (2/3)

- Gradients of lower-layer weights




Weighted sum

$$s_i^{(j)} = \mathbf{X}^{(j-1)} \cdot \mathbf{W}_i^{(j)}$$

$$\frac{\partial \varepsilon}{\partial \mathbf{W}_i^{(j)}} = \frac{\partial \varepsilon}{\partial s_i^{(j)}} \frac{\partial s_i^{(j)}}{\partial \mathbf{W}_i^{(j)}} = \frac{\partial \varepsilon}{\partial s_i^{(j)}} \mathbf{X}^{(j-1)}$$

$$= -2(d - f)\frac{\partial f}{\partial s_i^{(j)}} \mathbf{X}^{(j-1)} = -2\delta_i^{(j)} \mathbf{X}^{(j-1)}$$

Local gradient

$$\frac{\partial \varepsilon}{\partial s_i^{(j)}} = \frac{\partial (d - f)^2}{\partial s_i^{(j)}} = -2(d - f)\frac{\partial f}{\partial s_i^{(j)}}$$

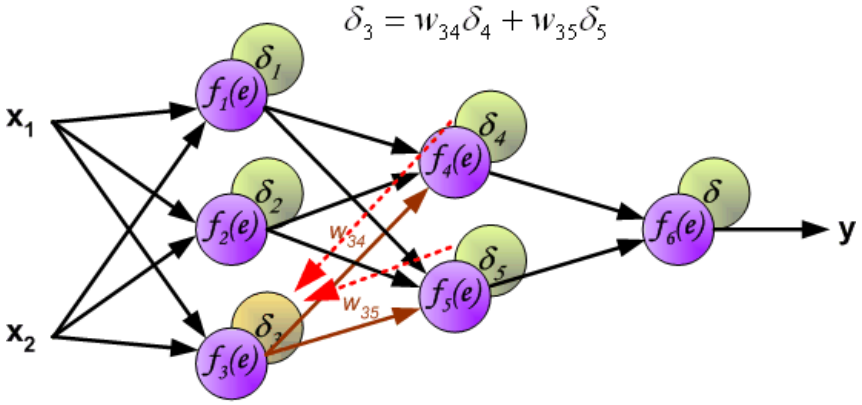
**Gradient Descent Update rule for lower-layer weights**

$$\mathbf{W}_i^{(j)} \leftarrow \mathbf{W}_i^{(j)} + c_i^{(j)} \delta_i^{(j)} \mathbf{X}^{(j-1)}$$

# Back Propagation Learning Algorithm (3/3)

- Applying chain rule, recursive relation between delta's

$$\delta_i^{(j)} = f_i^{(j)}(1 - f_i^{(j)}) \sum_{l=1}^{m_{j+1}} \delta_i^{(j+1)} w_{il}^{(j+1)}$$

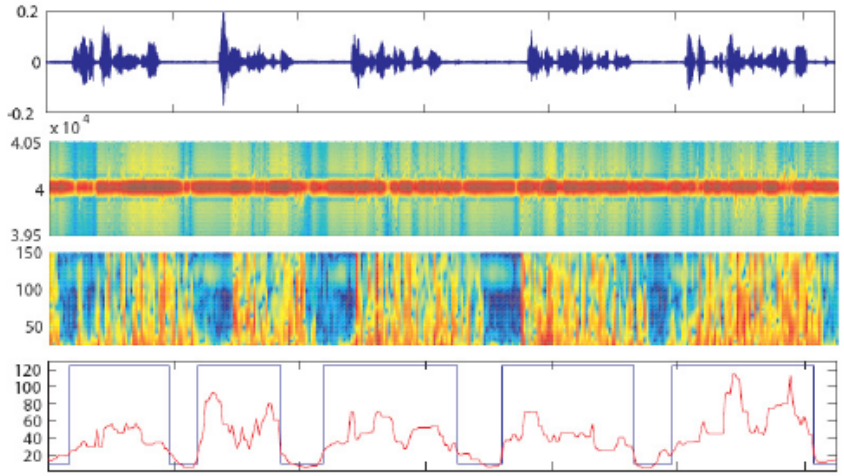



**Algorithm: Back Propagation**

1. **Randomly Initialize weight parameters**
2. **Calculate the activations of all units (with input data)**
3. **Calculate top-layer delta**
4. **Back-propagate delta from top to the bottom**
5. **Calculate actual gradient of all units using delta's**
6. **Update weights using Gradient Descent rule**
7. **Repeat 2~6 until converge**

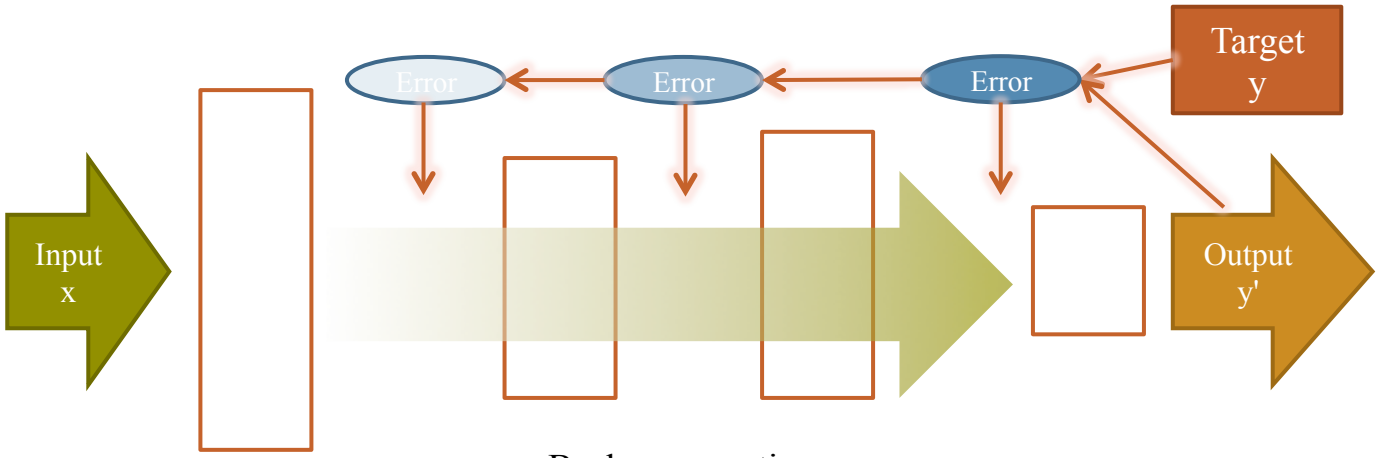
# Applications

- Almost All Classification Problems
  - Face Recognition
  - Object Recognition
  - Voice Recognition
  - Spam mail Detection
  - Disease Detection
  - etc.



# Limitations and Breakthrough

- Limitations
  - Back Propagation barely changes lower-layer parameters (Vanishing Gradient)
  - Therefore, Deep Networks cannot be fully (effectively) trained with Back Propagation

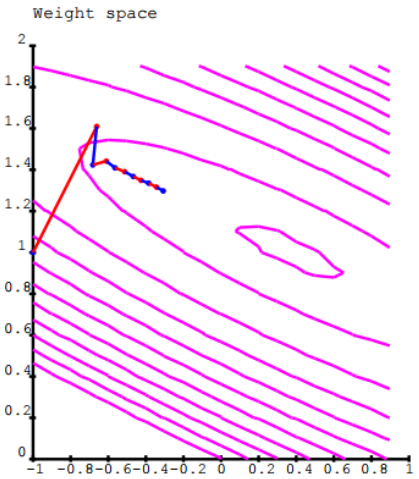


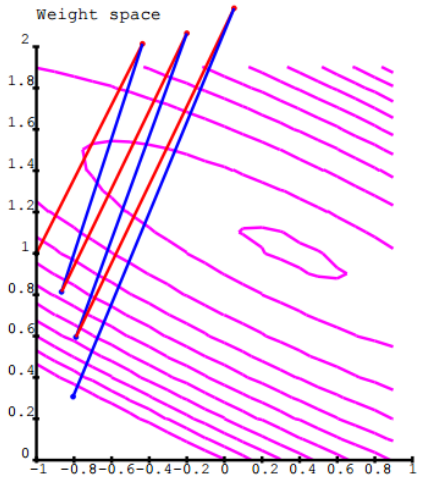

Back-propagation

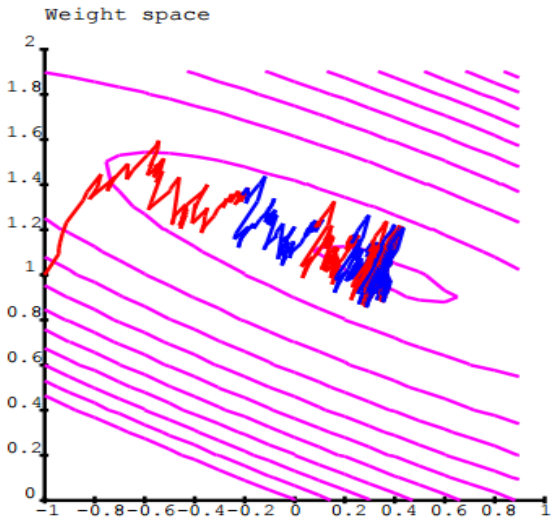
- Breakthrough
  - Deep Belief Networks (Unsupervised Pre-training)
  - Convolutional Neural Networks (Reducing Redundant Parameters)
  - Rectified Linear Unit (Constant Gradient Propagation)

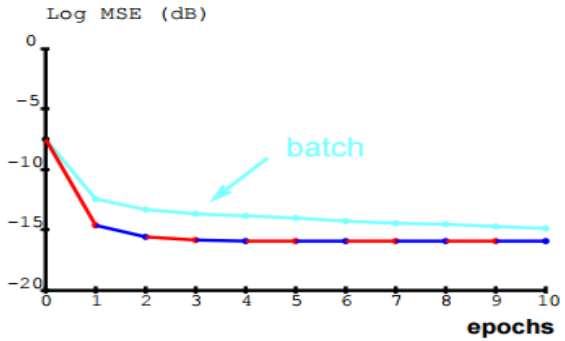
# Some Issues (1/3)

- Stochastic Gradient Descent



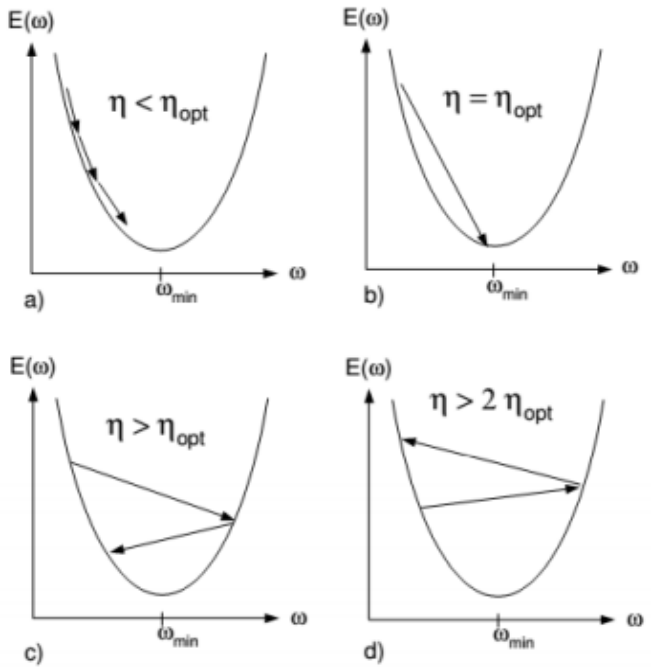



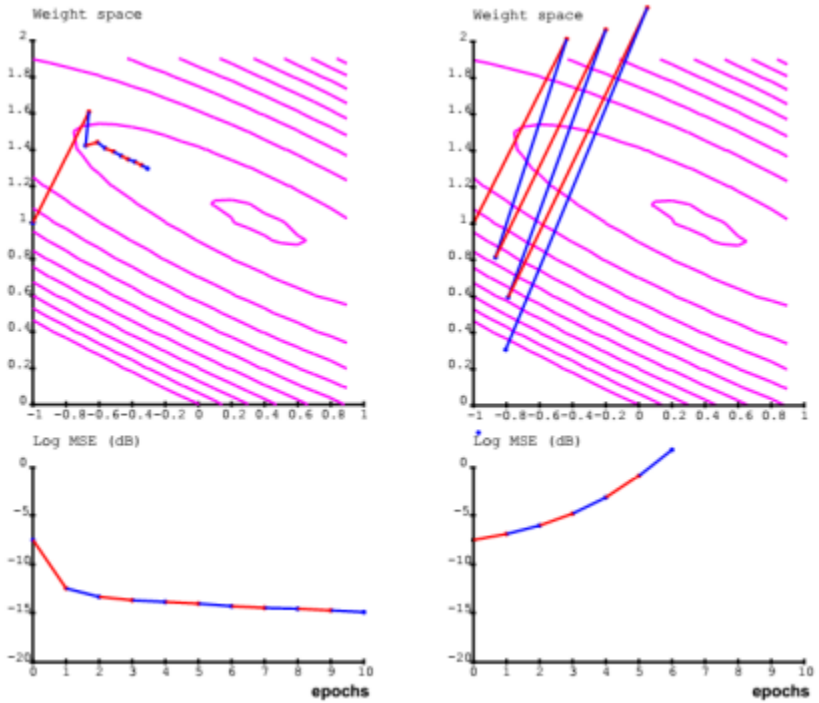




http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

# Some Issues (2/3)

- Learning Rate Adaptation
- Momentum
- Weight Decay



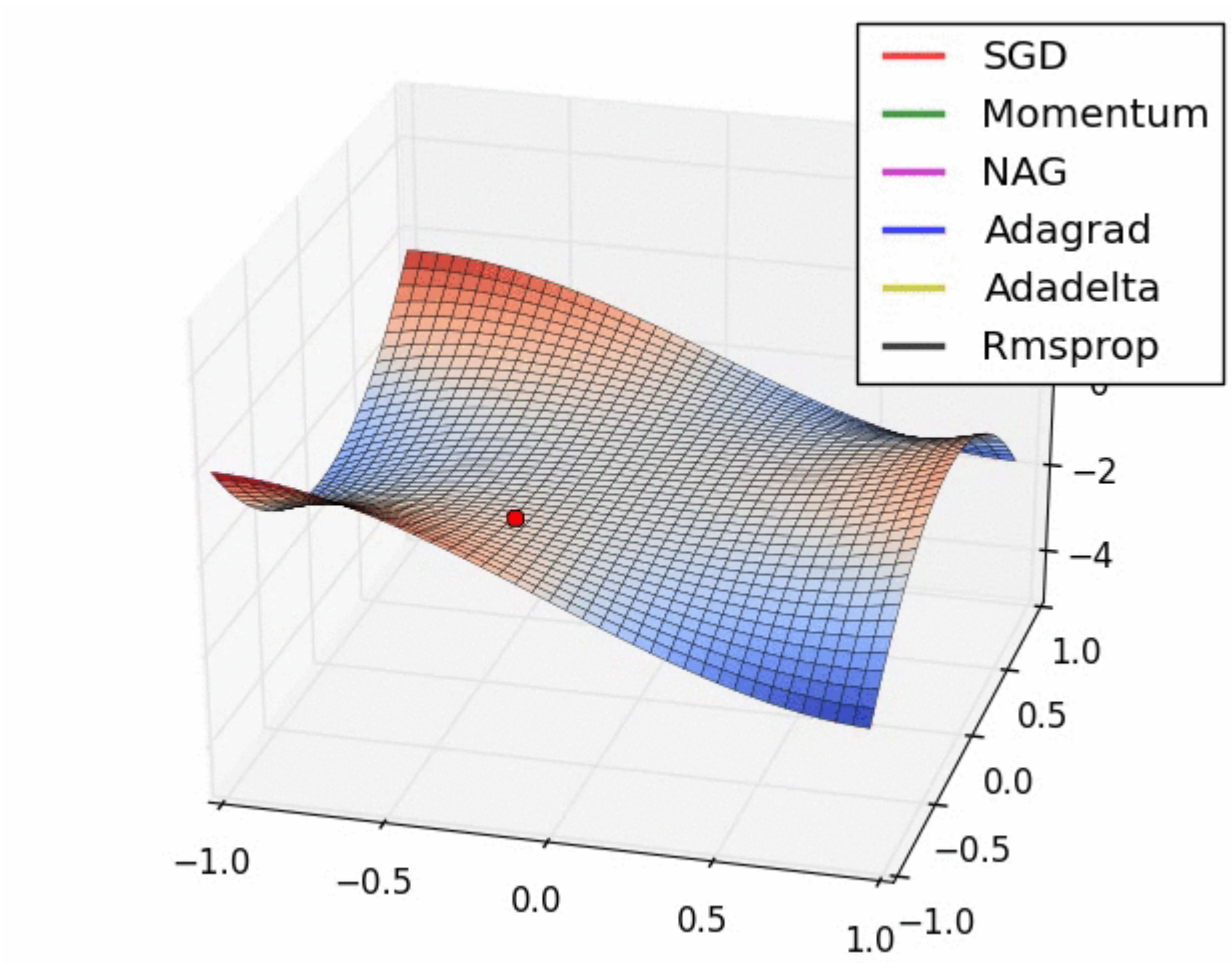




http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

http://courses.cs.tau.ac.il/Caffe_workshop/Bootcamp/pdf_lectures/Lecture%204%20CNN%20-%20optimization.pdf

# Some Issues (3/3)

- State-of-the-art optimization techniques on NN



# Applications (Image Classification) (1/4)

## Image Net Competition Ranking
(1000-class, 1 million images)

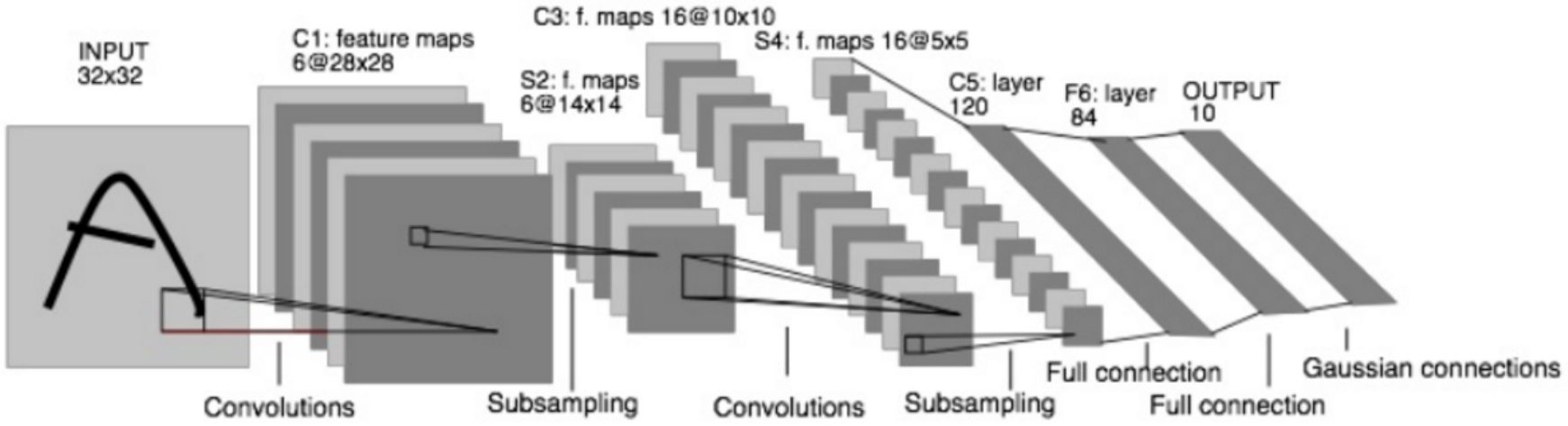
2. NUS: **Deep Convolutional Neural Networks**
3. ZF: **Deep Convolutional Neural Networks**
4. Andrew Howard: **Deep Convolutional Neural Networks**
5. OverFeat: **Deep Convolutional Neural Networks**
6. UvA-Euvision: **Deep Convolutional Neural Networks**
7. Adobe: **Deep Convolutional Neural Networks**
8. VGG: **Deep Convolutional Neural Networks**
9. CognitiveVision: **Deep Convolutional Neural Networks**
10. decaf: **Deep Convolutional Neural Networks**
11. IBM Multimedia Team: **Deep Convolutional Neural Networks**
12. Deep Punx (0.209): **Deep Convolutional Neural Networks**
13. *MIL (0.244): Local image descriptors + FV + linear classifier (Hidaka et al.)*
14. Minerva-MSRA: **Deep Convolutional Neural Networks**
15. Orange: **Deep Convolutional Neural Networks**

ALL CNN!!

From Kyunghyun Cho's dnn tutorial

# Applications (Image Classification) (2/4)

- 1989, CNN for hand digit recognition, Yann LeCun

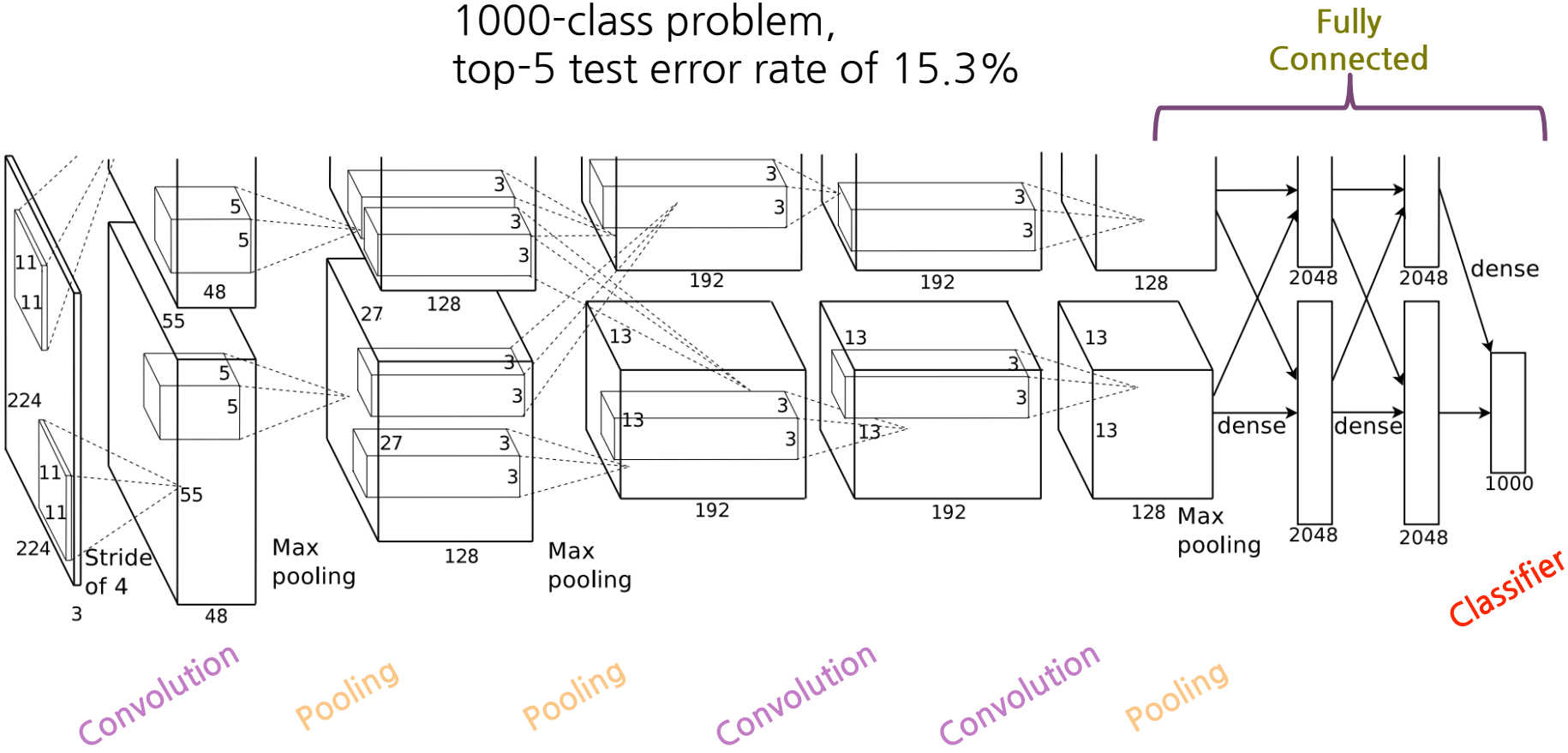



LeNet: a layered model composed of convolution and subsampling operations followed by a holistic representation and ultimately a classifier for handwritten digits. [ Yann LeCun; LeNet ]

# Applications (Image Classification) (3/4)

- Krizhevsky et al.: the winner of ImageNet 2012 Competition

1000-class problem,
top-5 test error rate of 15.3%



# Applications (Image Classification) (4/4)

- 2014 ILSVRC winner, ~6.6% Top 5 error

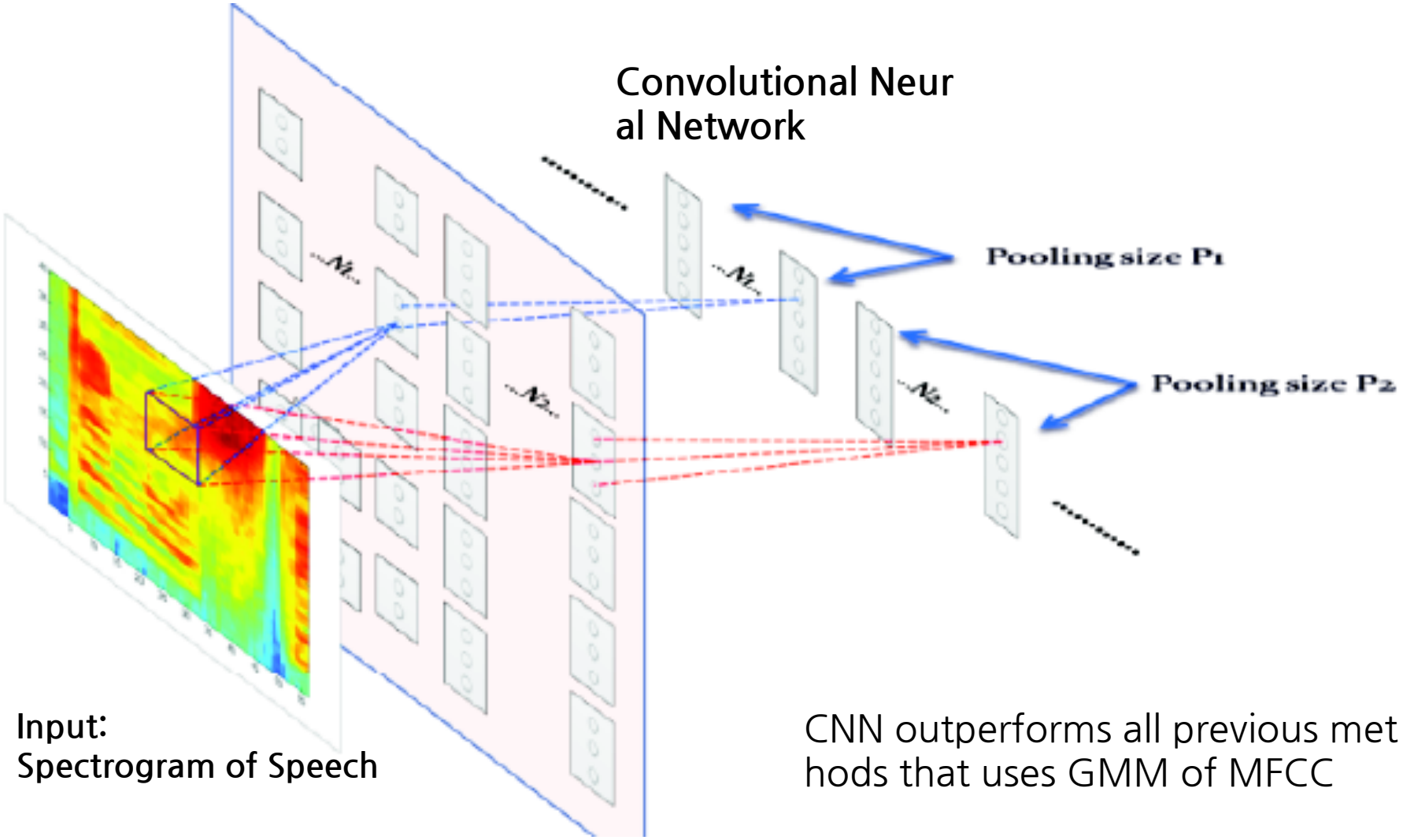
**Example: VGG**

19 layers
3x3 convolution
pad 1
stride 1



http://image-net.org/challenges/LSVRC/2014/results

# Application (Speech Recognition)




**Input:**
**Spectrogram of Speech**

CNN outperforms all previous met
hods that uses GMM of MFCC

# TensorFlow

# GPU 컴퓨팅의 필요성

- 다루는 문제의 복잡도가 증가할수록 모델이 커지고 보다 많은 연산이 요구됨
- 컨볼루션 연산, 통합 연산은 모두 행렬 연산
- CUDA를 사용하여 컨볼루션 신경망을 구현한 경우 CPU에 비해 약 10배 이상 속도 향상
  - AlexNet으로 ILSVRC 데이터 학습시 CUDA를 이용한 경우 약 4~5일 정도 소모됨

# 다양한 딥러닝 프레임워크

| | Caffe | Torch | Theano | TensorFlow |
|---|---|---|---|---|
| **Language** | C++, Python | Lua | Python | Python |
| **Pretrained** | Yes ++ | Yes ++ | Yes (Lasagne) | Inception |
| **Multi-GPU: Data parallel** | Yes | Yes cunn. DataParallelTable | Yes platoon | Yes |
| **Multi-GPU: Model parallel** | No | Yes fbcunn.ModelParallel | Experimental | Yes (best) |
| **Readable source code** | Yes (C++) | Yes (Lua) | No | No |
| **Good at RNN** | No | Mediocre | Yes | Yes (best) |

http://cs231n.stanford.edu/slides/winter1516_lecture12.pdf

# TensorFlow 소개

- 개요
  - Google에서 만든 Python 기반의 오픈소스 Package
  - Symbolic 연산 철학
- 장점
  - Symbolic 연산 철학으로 간결하고 빠르게 모델 구현 가능
  - Symbolic 미분이 가능하므로 Back-Propagation 등을 직접 구현할 필요가 없음
  - 동일한 코드를 CPU와 GPU에서 모두 사용 가능
  - Python 기반이므로, numpy, scipy, matplotlib, ipython 등 다양한 python 패키지와의 연동 용이
  - Computational Graph Abstraction
  - Theano 보다 빠른 컴파일 속도
  - TensorBoard visualization
  - 데이터, 모델 병렬화

# Basic Usage

- To use TensorFlow you need to understand how TensorFlow:
  - Represents computations as graphs
  - Executes graphs in the context of Sessions
  - Represents data as tensors
  - Maintains state with Variables
  - Uses feeds and fetches to get data into and out of arbitrary operations

https://www.tensorflow.org/versions/r0.10/get_started/basic_usage.html

# Session and Graph

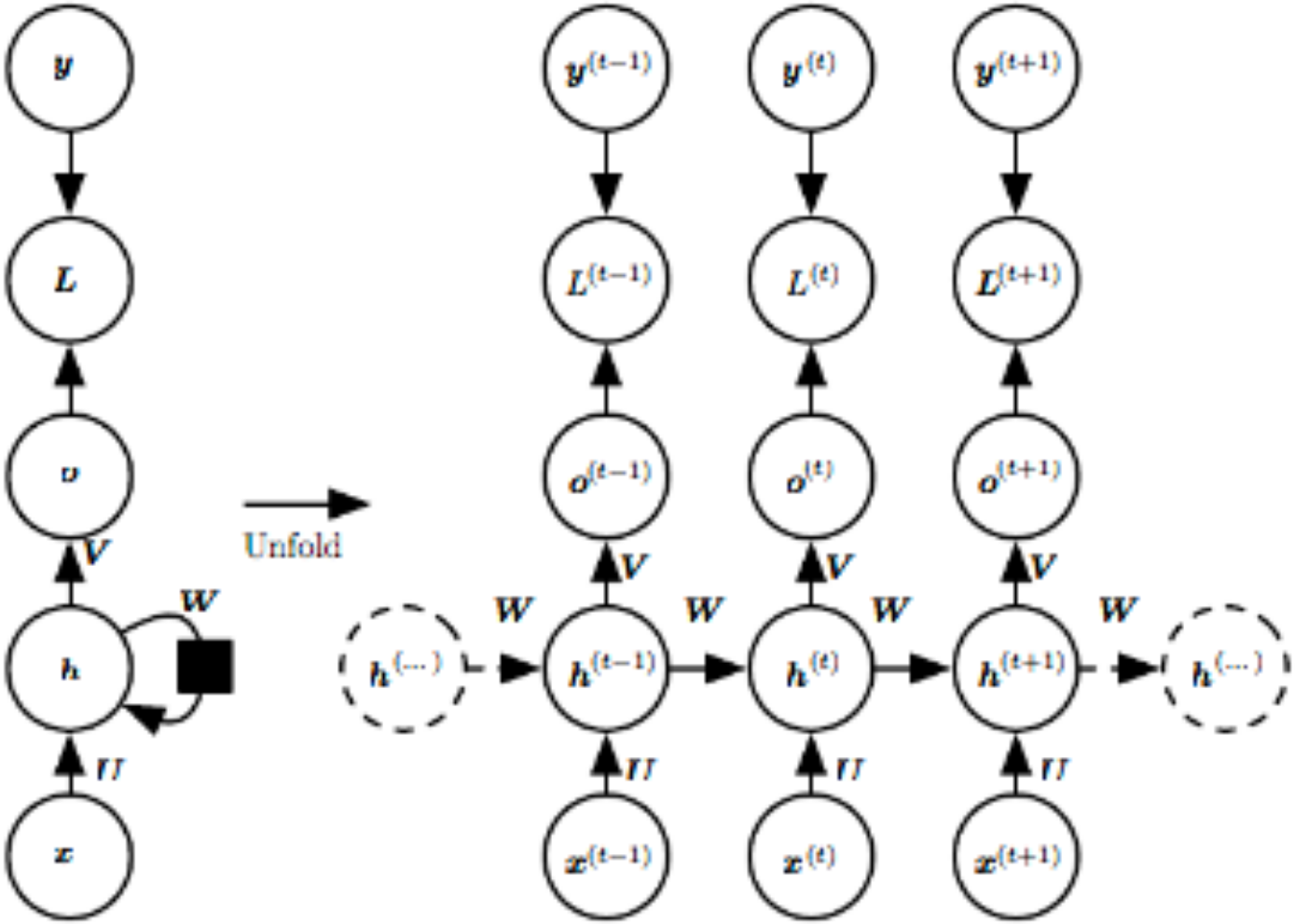
- Computational Graph
  - TensorFlow programs are structured into a construction phase, that assembles a graph, and an execution phase that uses a session to execute ops in the graph.

- Launching the graph in a session
  - Construction -> Launching
  - To launch a graph, create a Session object

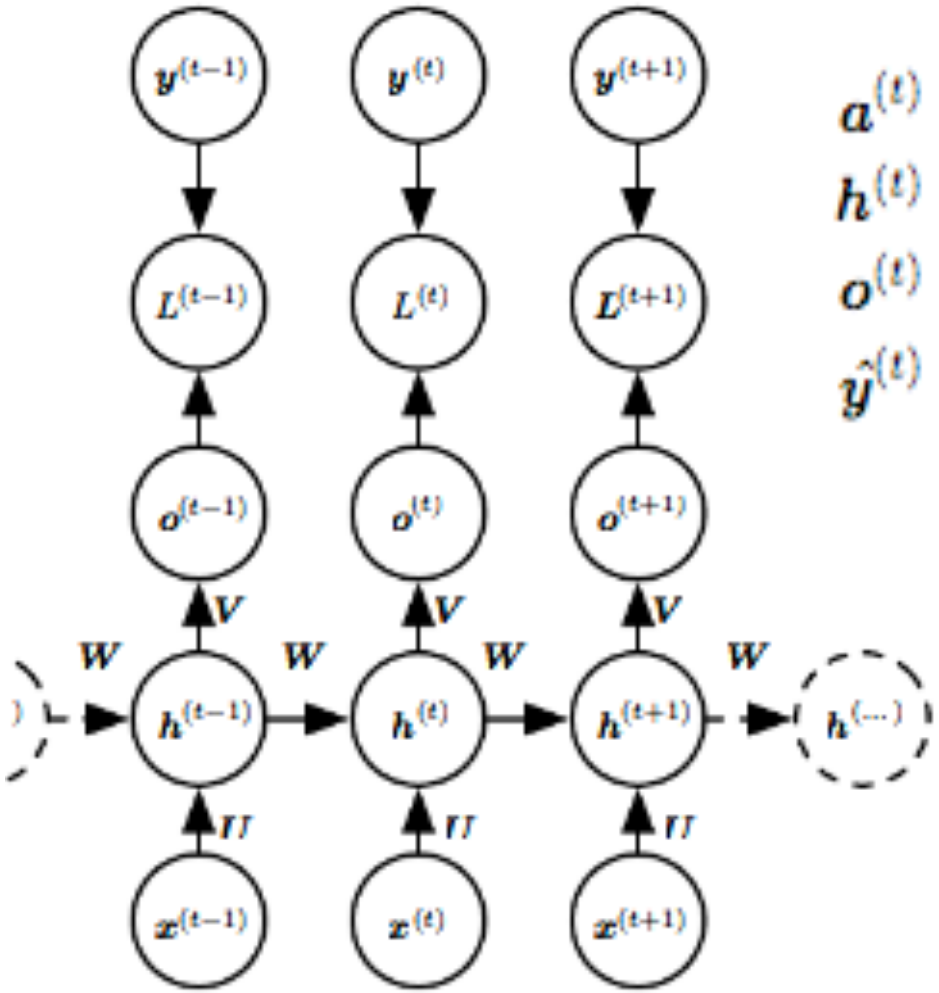
# TensorFlow Basic 실습

# Recurrent Neural Network

# Architecture

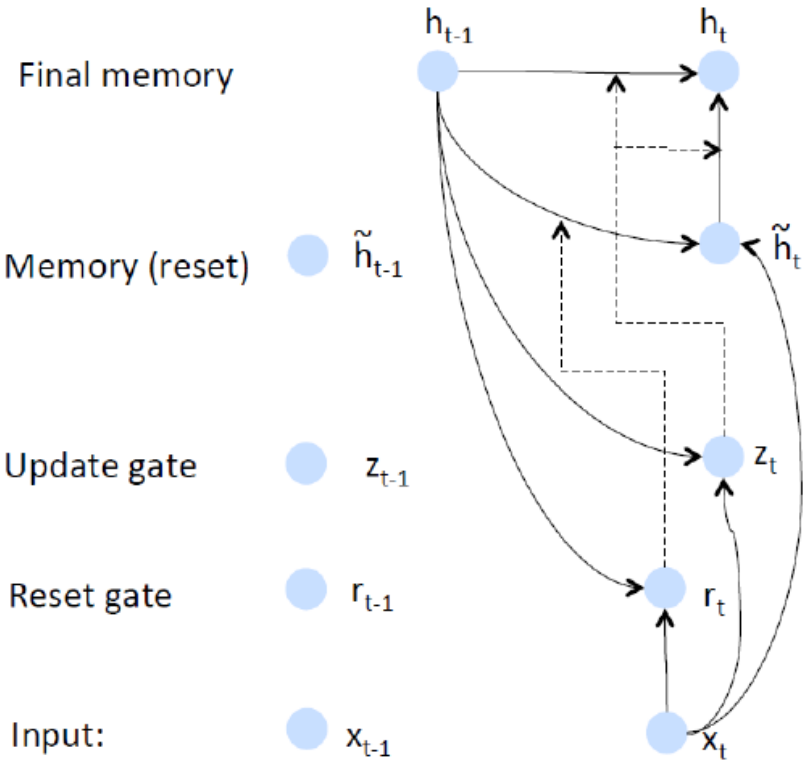


# Architecture




$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

# Gated Recurrent Unit




$$z^t = \sigma\left(W_z x^t + U_z h^{t-1}\right)$$

$$r^t = \sigma\left(W_r x^t + U_r h^{t-1}\right)$$

$$\tilde{h}^t = \tanh\left(W x^t + r^t \circ U h^{t-1}\right)$$

$$h^t = z^t \circ h^{t-1} + (1 - z^t) \circ \tilde{h}^t$$

source: Socher (2015)