



DEPARTAMENTO DE SEÑALES, SISTEMAS Y RADIOCOMUNICACIONES



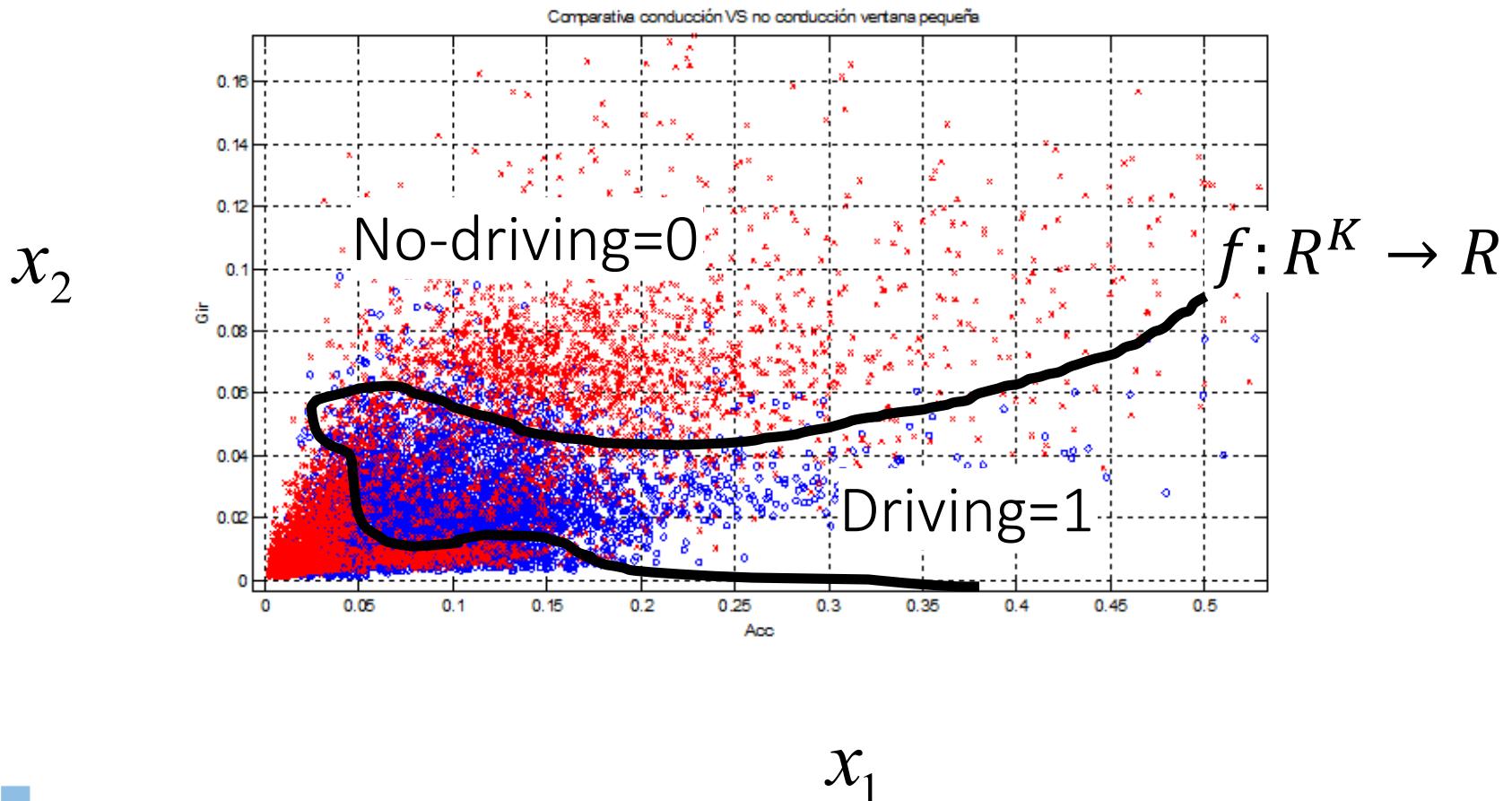
Deep Learning Seminar Day-2

Master of Science in Signal Theory and Communications
TRACK: Signal Processing and Machine Learning for Big Data

Departamento de Señales, Sistemas y Radiocomunicaciones
E.T.S. Ingenieros de Telecomunicación
Universidad Politécnica de Madrid

From linear classifiers TO Neural Networks

Nonlinear decision function?



From linear classifiers TO Neural Networks

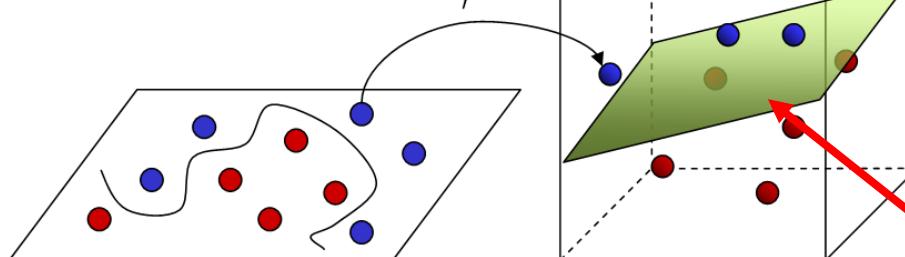
$$y = \phi(\mathbf{x})^T \mathbf{w}$$

$\phi(\mathbf{x})$ a non linear transformation

1. To manually engineer $\phi(\cdot)$
2. Use a very generic $\phi(\cdot)$ as kernel machines
(e.g. SVM, RBF kernel)
3. The strategy of deep learning : to learn $\phi(\cdot)$

SVM

Hand-crafted
kernel function



From: DL Tutorial
Hung-yi Lee

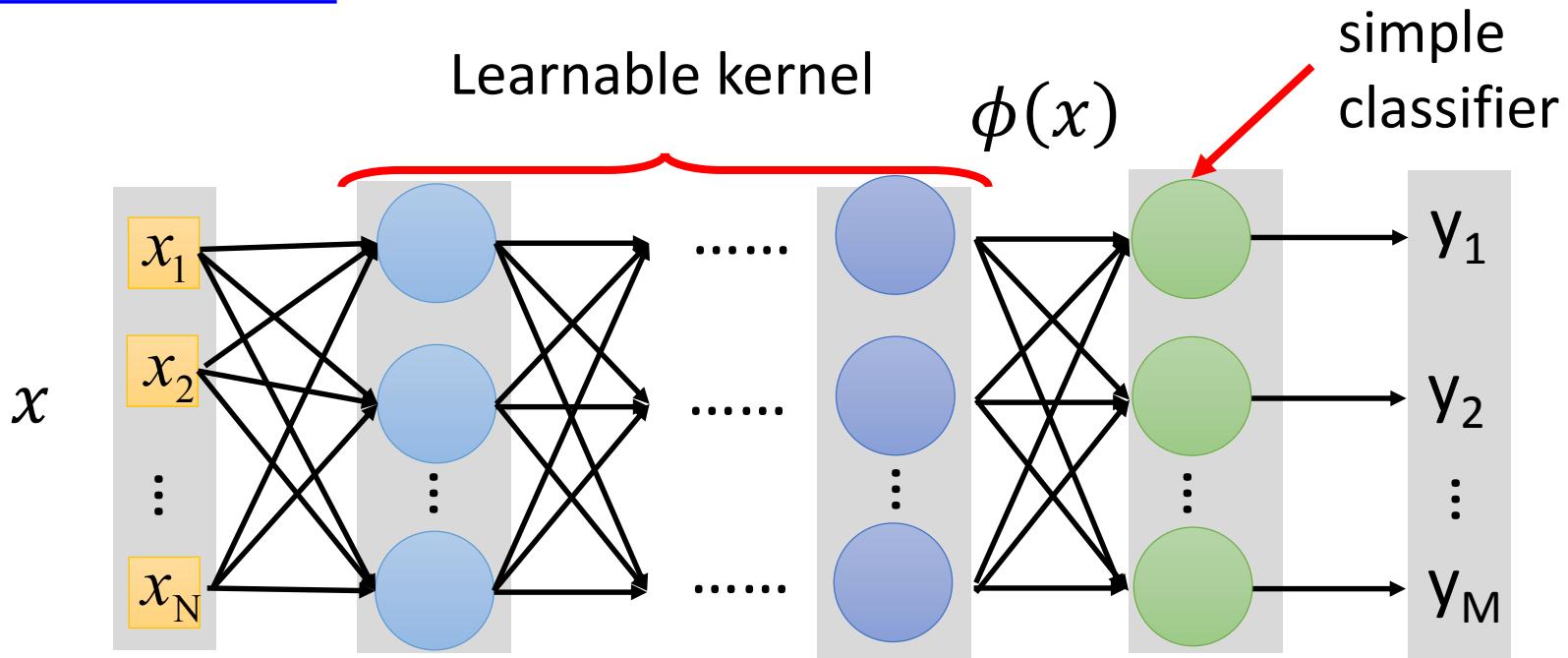
Input Space

Feature Space

Apply simple
classifier

Source of image: http://www.gipsa-lab.grenoble-inp.fr/transfert/seminaire/455_Kadri2013Gipsa-lab.pdf

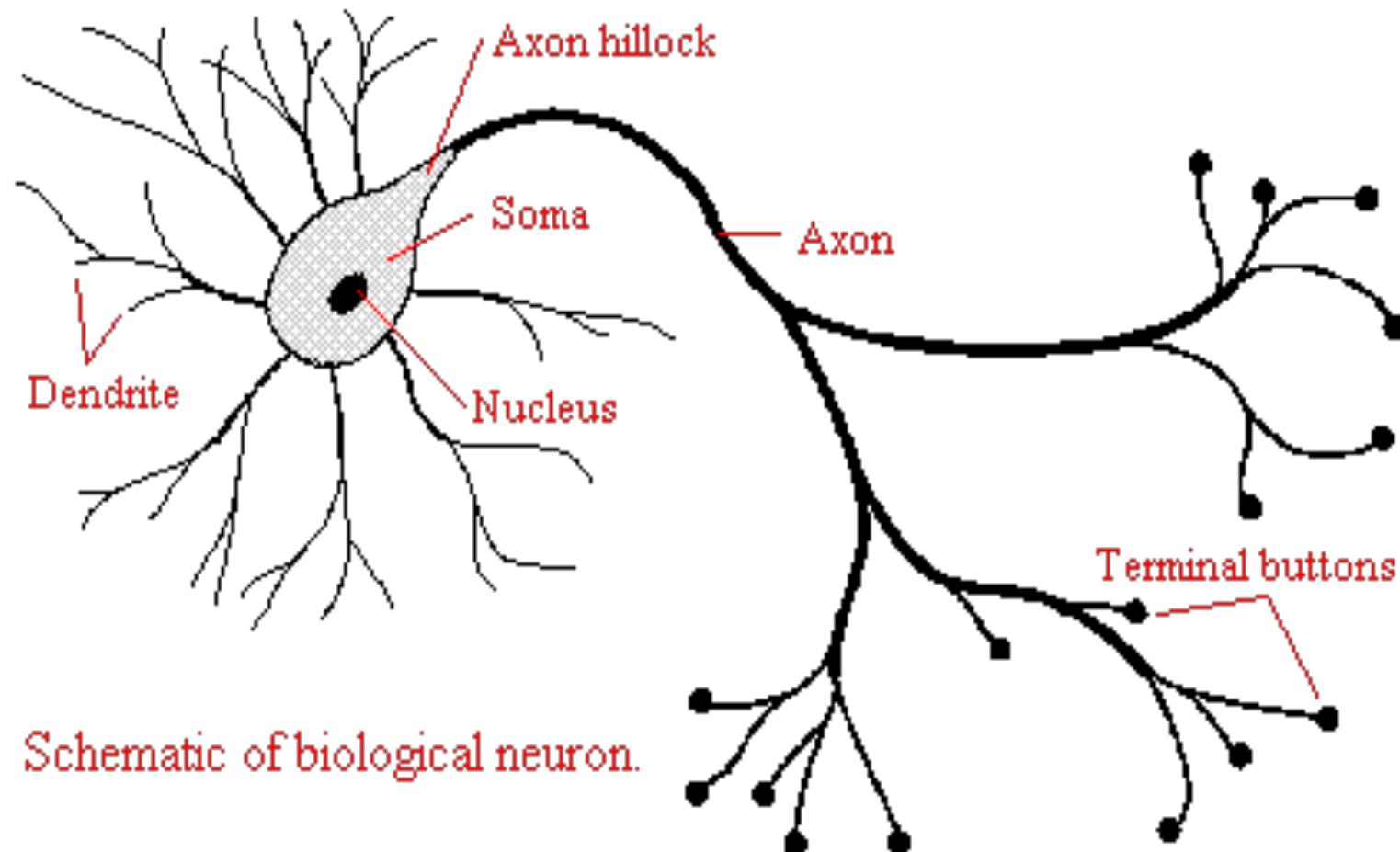
Deep Learning



From linear classifiers TO Neural Networks

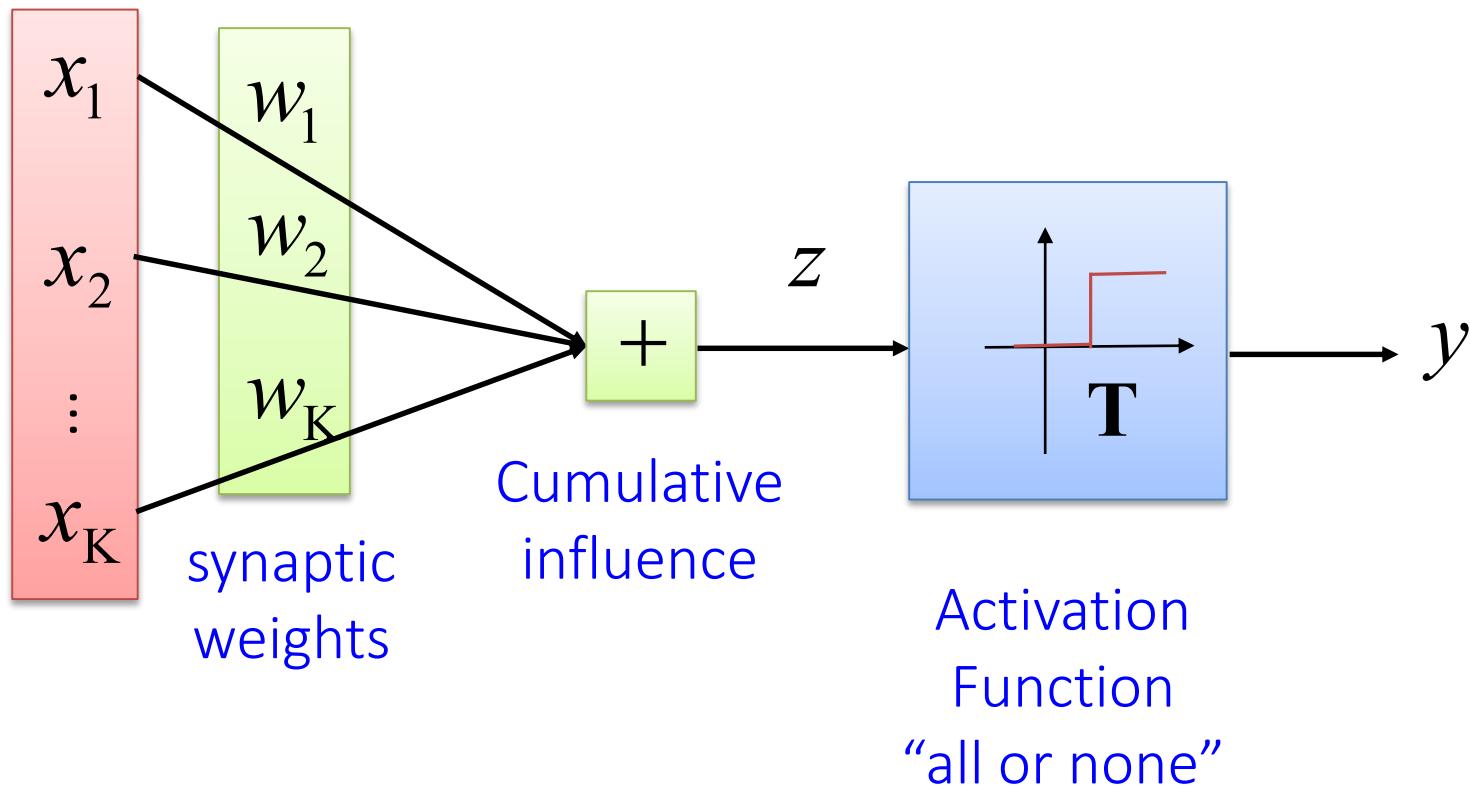
The DL approach: learn $\phi(\mathbf{x})$

...from a broad class of functions



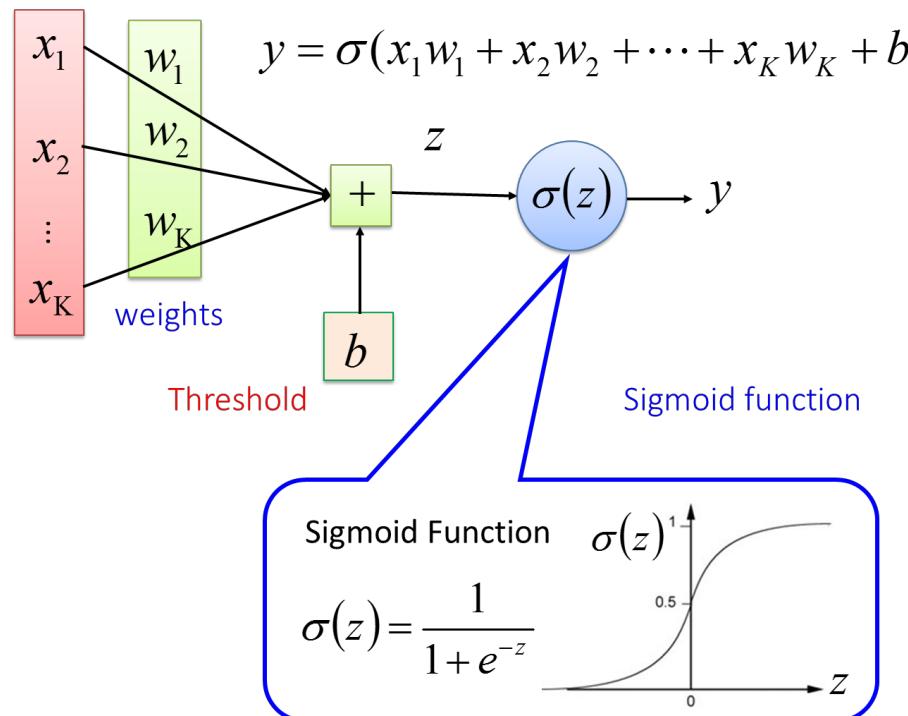
From linear classifiers TO Neural Networks

Neuron approach....



From linear classifiers TO Neural Networks

- Recall that this is also logistic regression...



- But a non-linearity just at the end of a linear combination doesn't make any difference (still linear discrimination!)

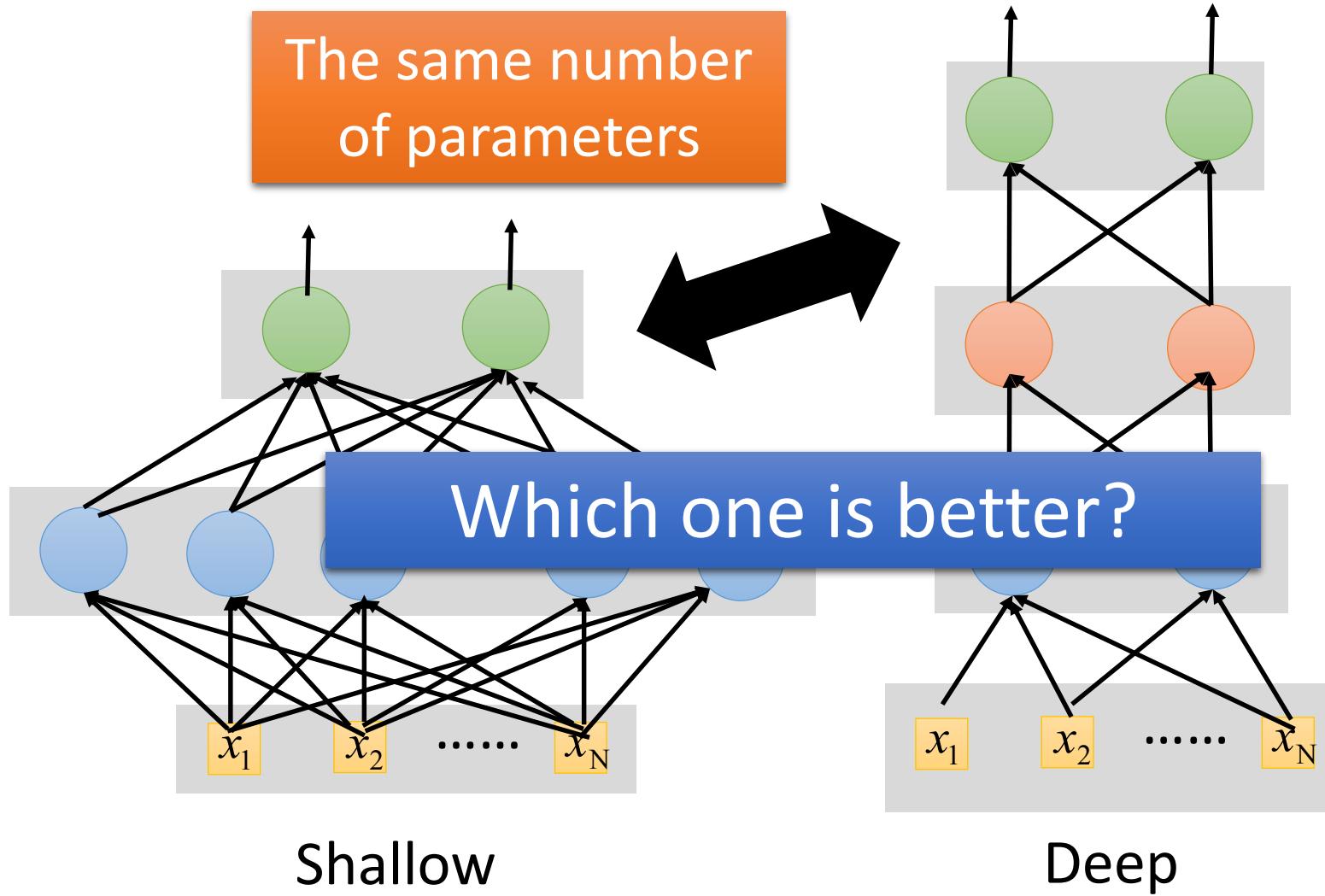
From linear classifiers TO Neural Networks

- What we need is a network of non-linear functions:

$$f(x) = f^3(f^2(f^1(x))))$$

- And if it is **deep!** It can do *feature extraction*
... abstraction...

Fat + Short v.s. Thin + Tall

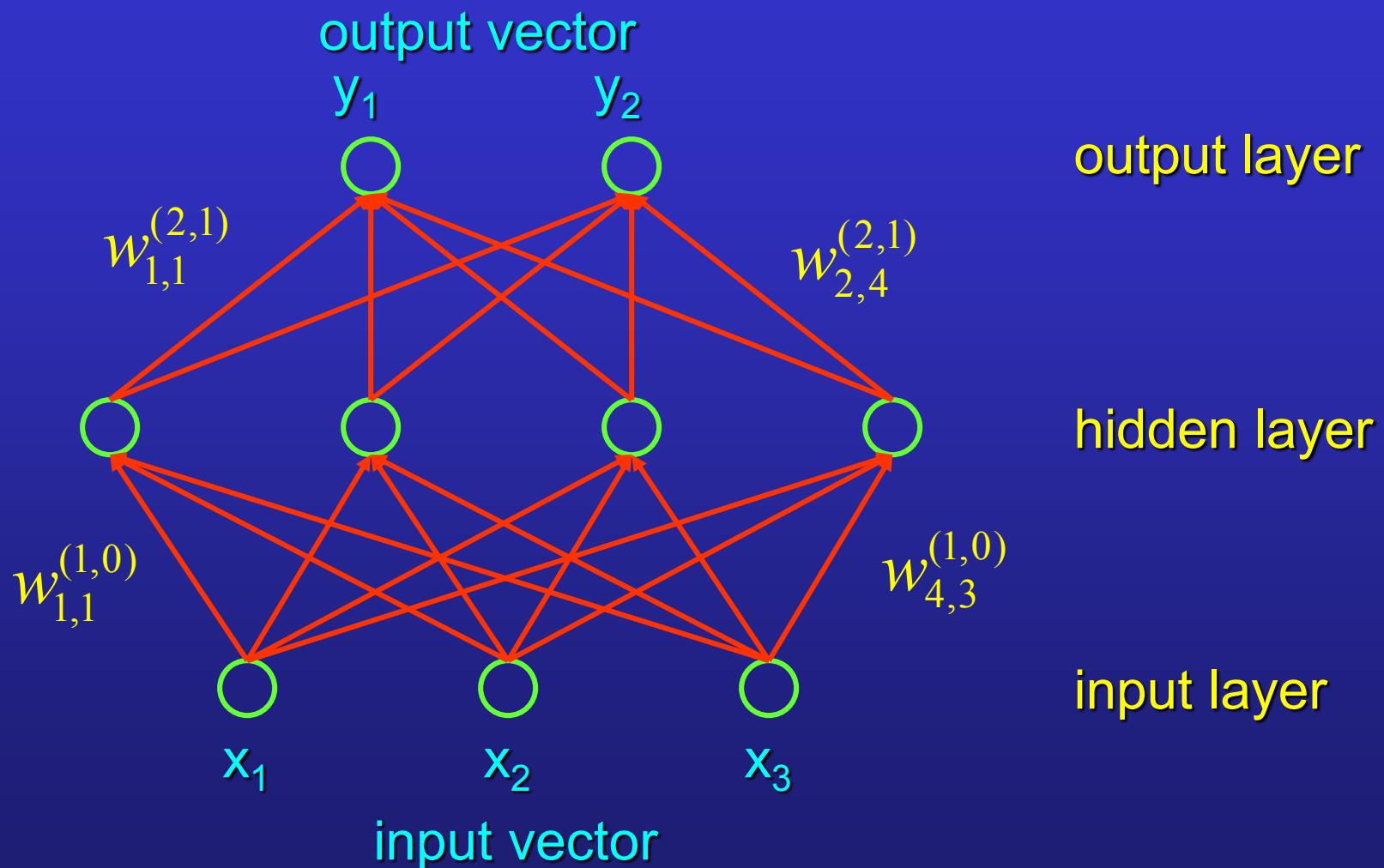


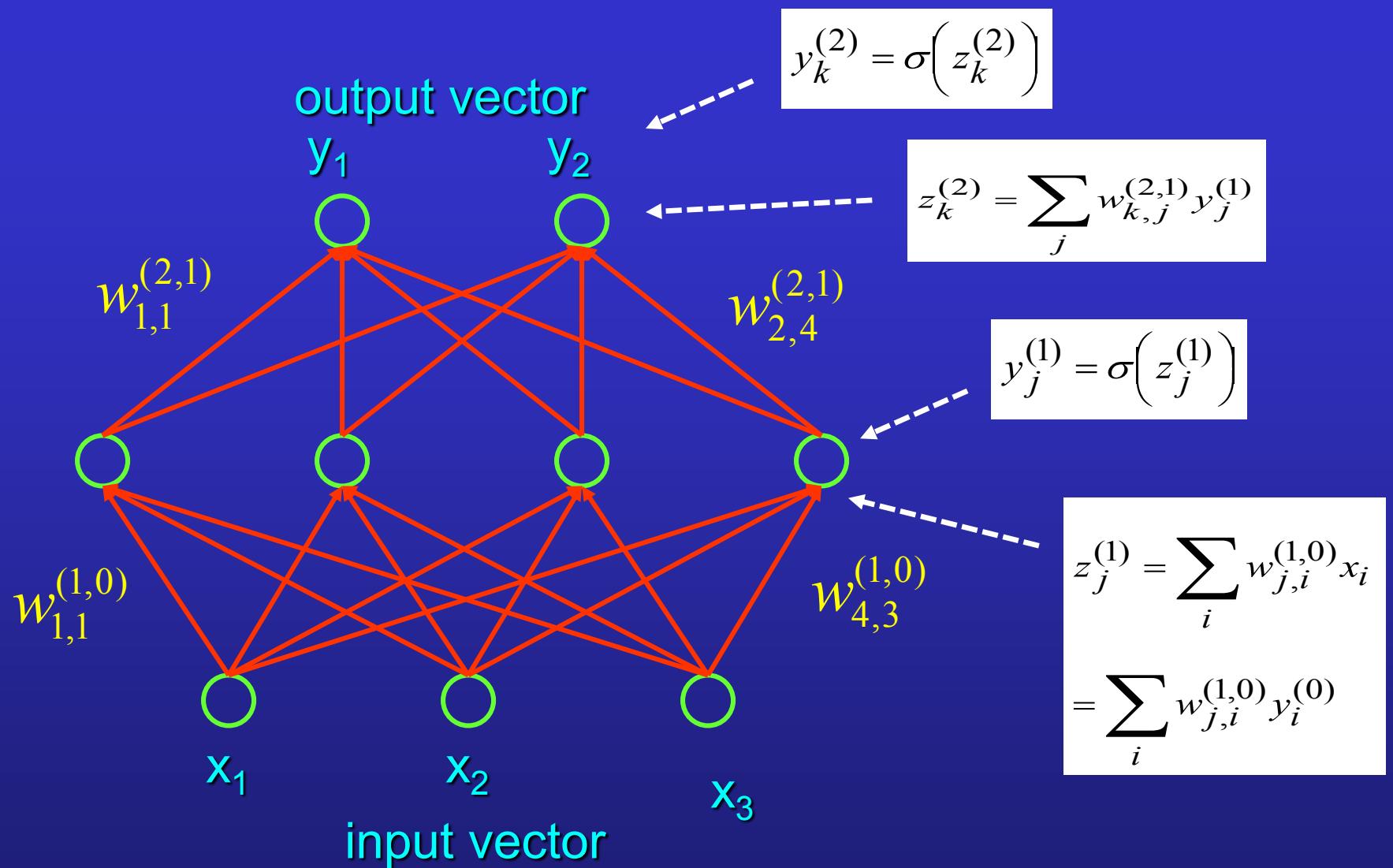
Training Neural Networks

- **Backpropagation** Learning
- Gradients estimation NOT an optimization method

Terminology

Example: Network function $f: \mathbf{R}^3 \rightarrow \mathbf{R}^2$





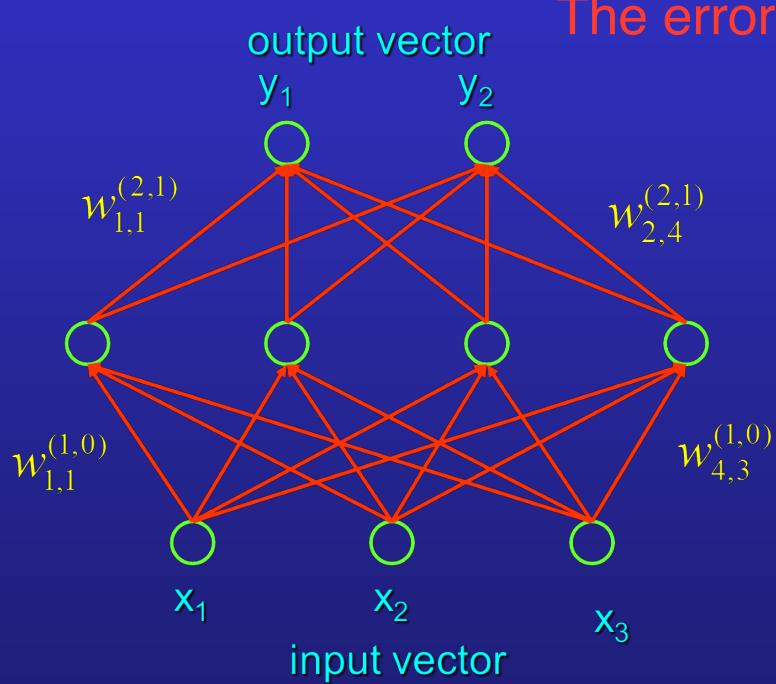
Activation function + linear combination
of the output of the previous layer

Backpropagation

E (cost or error) function of weights $w_{j,i}^{(k)}$

$$\text{Cost}(\theta) = - \sum_{n=1}^N (l_n \log(p_{1n}) + (1 - l_n) \log(1 - p_{1n}))$$

$$E(\theta) = \sum \left(y_k - y_k^{(2)} \right)^2$$



The error will be depend on the weight of the las layer

For using gradient descent algorithms we need:

$$\Delta w_{k,j}^{(2)} \propto \frac{-\partial E}{\partial w_{k,j}^{(2)}} \quad \text{Output layer: easy}$$

$$\Delta w_{j,i}^{(1)} \propto \frac{-\partial E}{\partial w_{j,i}^{(1)}} \quad \text{Hidden layer: difficult}$$

Backpropagation : apply the chain rule

$$E(\theta) = \sum \left(y_k - y_k^{(2)} \right)^2$$

$$y_k^{(2)} = \sigma\left(z_k^{(2)}\right)$$

$$z_k^{(2)} = \sum_j w_{k,j}^{(2,1)} y_j^{(1)}$$

$$\frac{\partial E}{\partial w_{k,j}^{(2)}} = \frac{\partial E}{\partial y_k^{(2)}} \frac{\partial y_k^{(2)}}{\partial z_k^{(2)}} \frac{\partial z_k^{(2)}}{\partial w_{k,j}^{(2)}}$$

$$\frac{\partial E}{\partial y_k^{(2)}} = -2(y_k - y_k^{(2)})$$

$$\frac{\partial y_k^{(2)}}{\partial z_k^{(2)}} = \sigma'(z_k^{(2)})$$

$$\frac{\partial z_k^{(2)}}{\partial w_{k,j}^{(2)}} = y_j^{(1)}$$

$$\frac{\partial E}{\partial w_{k,j}^{(2)}} = -2(y_k - y_k^{(2)}) \sigma'(z_k^{(2)}) y_j^{(1)}$$

Hidden layer : NOTICE that each $w_{j,i}^{(1)}$ influences

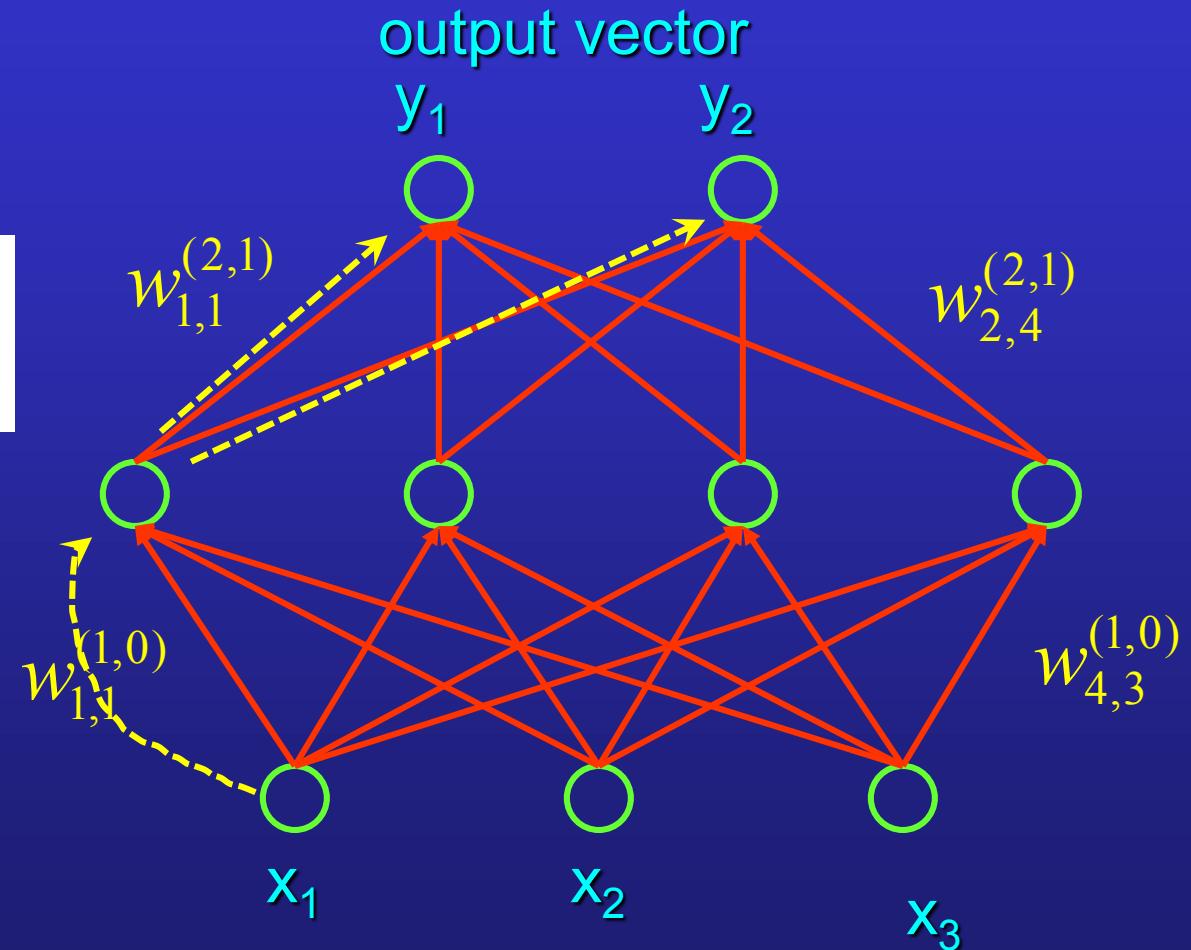
$$E(\theta) = \sum (y_k - y_k^{(2)})^2 \quad \text{each } y_k^{(2)} \text{ with } k=1, \dots, K \text{ (2 in our example)}$$

$$y_k^{(2)} = \sigma(z_k^{(2)})$$

$$z_k^{(2)} = \sum_j w_{k,j}^{(2,1)} y_j^{(1)}$$

$$y_j^{(1)} = \sigma(z_j^{(1)})$$

$$z_j^{(1)} = \sum_i w_{j,i}^{(1,0)} y_i^{(0)}$$



Hidden layer :

$$E(\theta) = \sum \left(y_k - y_k^{(2)} \right)^2$$

$$y_k^{(2)} = \sigma\left(z_k^{(2)}\right)$$

$$z_k^{(2)} = \sum_j w_{k,j}^{(2,1)} y_j^{(1)}$$

$$y_j^{(1)} = \sigma\left(z_j^{(1)}\right)$$

$$z_j^{(1)} = \sum_i w_{j,i}^{(1,0)} y_i^{(0)}$$

$$\frac{\partial E}{\partial w_{j,i}^{(1)}} = \sum_{k=1}^K \frac{\partial E}{\partial y_k^{(2)}} \frac{\partial y_k^{(2)}}{\partial z_k^{(2)}} \frac{\partial z_k^{(2)}}{\partial y_j^{(1)}} \frac{\partial y_j^{(1)}}{\partial z_j^{(1)}} \frac{\partial z_j^{(1)}}{\partial w_{j,i}^{(1)}}$$

$$\frac{\partial E}{\partial w_{j,i}^{(1)}} = \sum_{k=1}^K \left[-2(y_k - y_k^{(2)}) \sigma'\left(z_k^{(2)}\right) w_{k,j}^{(2)} \sigma'\left(z_j^{(1)}\right) y_i^{(0)} \right]$$

Weight changes :

Output layer

$$\frac{\partial E}{\partial w_{k,j}^{(2)}} = -2(y_k - y_k^{(2)})\sigma'(z_k^{(2)})y_j^{(1)}$$

$$\Delta w_{k,j}^{(2)} = \eta \cdot \delta_k \cdot y_j^{(1)} \quad \text{with} \quad \delta_k = (y_k - y_k^{(2)})\sigma'(z_k^{(2)})$$

Hidden layer

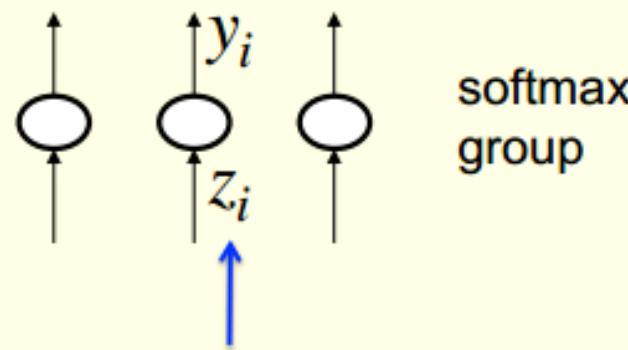
$$\frac{\partial E}{\partial w_{j,i}^{(1)}} = \sum_{k=1}^K \left[-2(y_k - y_k^{(2)})\sigma'(z_k^{(2)})w_{k,j}^{(2)}\sigma'(z_j^{(1)})y_i^{(0)} \right]$$

$$\Delta w_{j,i}^{(1)} = \eta \cdot \mu_j \cdot x_i \quad \text{with} \quad \mu_j = \left(\sum_{k=1}^K \delta_k w_{k,j}^{(2)} \right) \sigma'(z_j^{(1)})$$

Softmax activation function

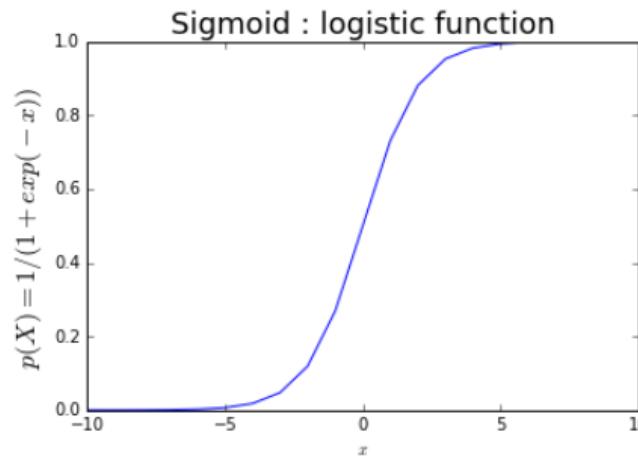
$i=1 \dots K$ output classes

The output units in a softmax group use a non-local non-linearity:



$$y_i = \frac{e^{z_i}}{\sum_{j \in \text{group}} e^{z_j}}$$

$$\frac{\partial y_i}{\partial z_i} = y_i (1 - y_i)$$



Softmax Layer

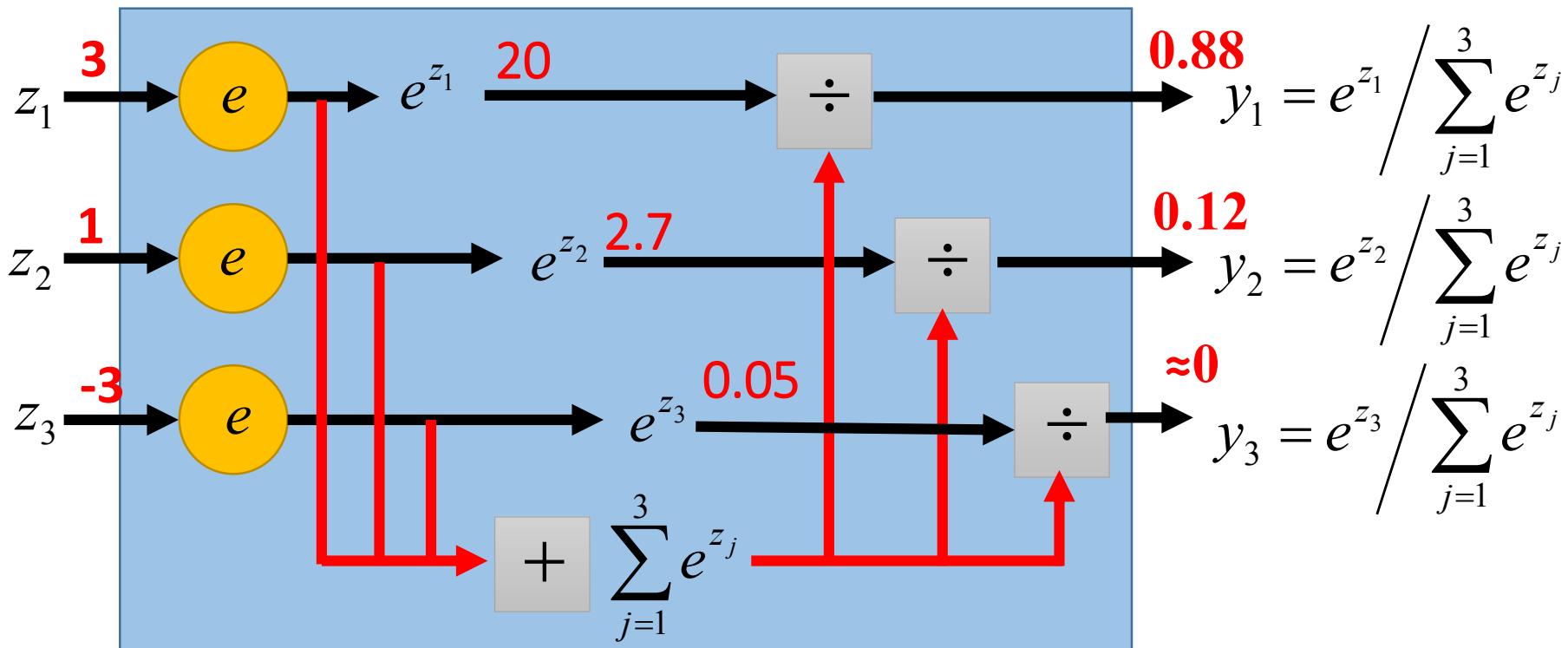
- Softmax layer as the output layer
Normalizing result to be 0 or 1

Probability:

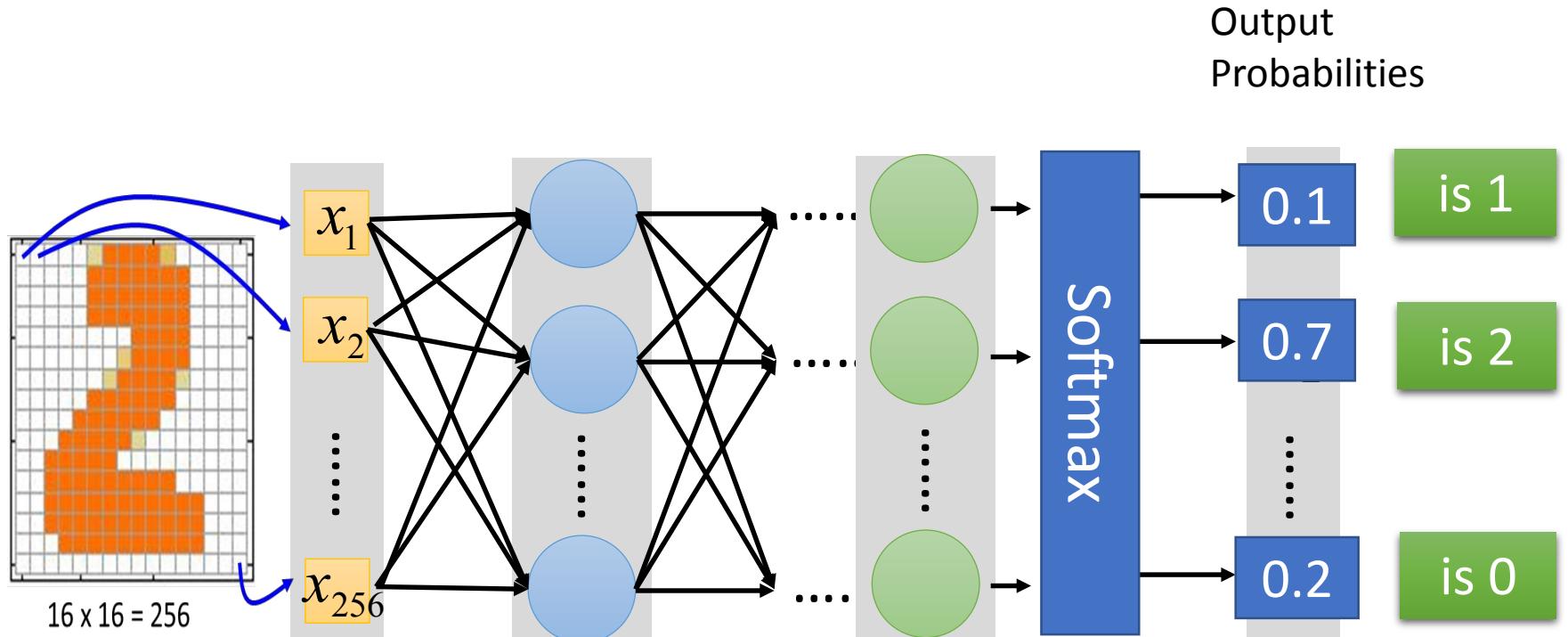
■ $1 > y_i > 0$

■ $\sum_i y_i = 1$

Softmax Layer



Softmax layer



Sigmoidal Neurons

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

$$\sigma'(x) = \frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2}$$

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

$$\sigma'(x) = \frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$$

Using sigmoid derivative properties :

Output layer

$$\begin{aligned}\delta_k &= (y_k - y_k^{(2)}) \sigma'(z_k^{(2)}) \\ &= (y_k - y_k^{(2)}) y_k^{(2)} (1 - y_k^{(2)})\end{aligned}$$

Hidden layer

$$\begin{aligned}\mu_j &= \left(\sum_{k=1}^K \delta_k w_{k,j}^{(2)} \right) \sigma'(z_j^{(1)}) \\ &= \left(\sum_{k=1}^K \delta_k w_{k,j}^{(2)} \right) y_j^{(1)} (1 - y_j^{(1)})\end{aligned}$$

The simplified δ_k and μ_j use variables that can
are calculated in the **feedforward** phase very
efficiently

Output layer

$$\Delta w_{k,j}^{(2)} = \eta \cdot \delta_k \cdot y_j^{(1)} \quad \text{with}$$

$$\delta_k = (y_k - y_k^{(2)}) y_k^{(2)} (1 - y_k^{(2)})$$

Hidden layer

$$\Delta w_{j,i}^{(1)} = \eta \cdot \mu_j \cdot x_i \quad \text{with}$$

$$\mu_j = \left(\sum_{k=1}^K \delta_k w_{k,j}^{(2)} \right) y_j^{(1)} (1 - y_j^{(1)})$$

- Algorithm first performs forward propagation, which maps parameters to loss or cost function associated training examples
- Then the corresponding computation for applying the back-propagation algorithm is done

Backpropagation algorithm computes the gradient, while another algorithm, such as stochastic gradient descent, is used to perform learning using this gradient

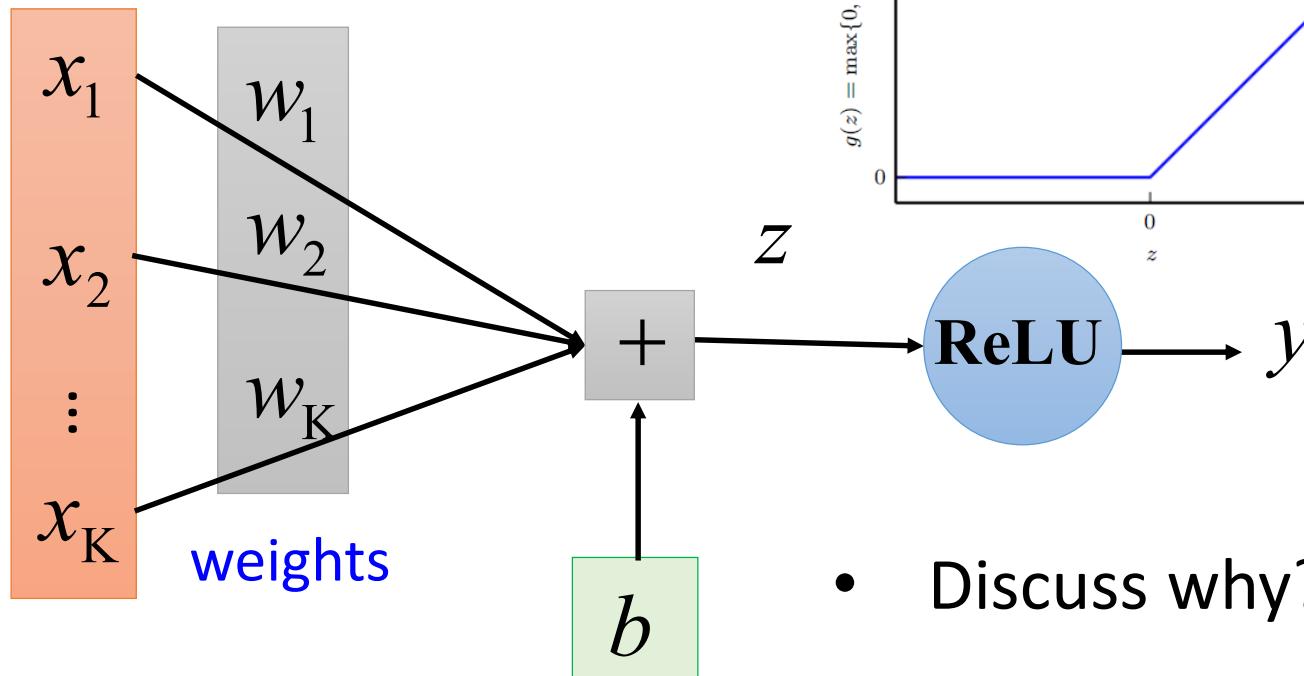
Deep Learning Tutorial

李宏毅

Hung-yi Lee

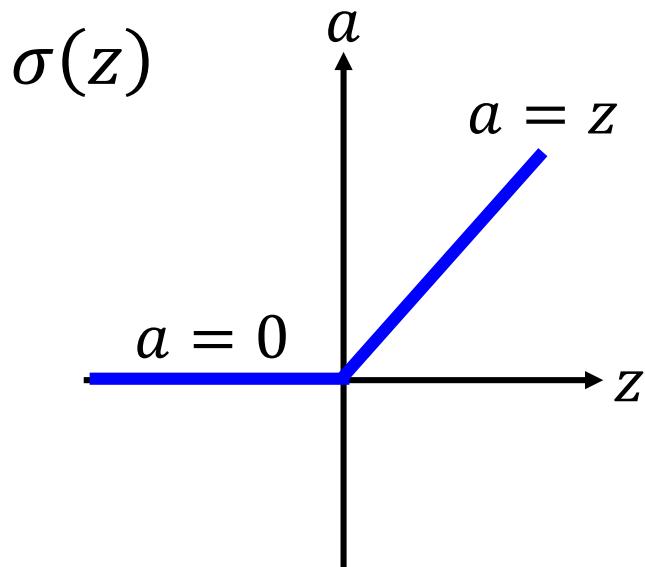
Activation Functions

- In modern neural networks, the default recommendation is to use the **rectified linear unit** or ReLU



ReLU

- Rectified Linear Unit (ReLU)

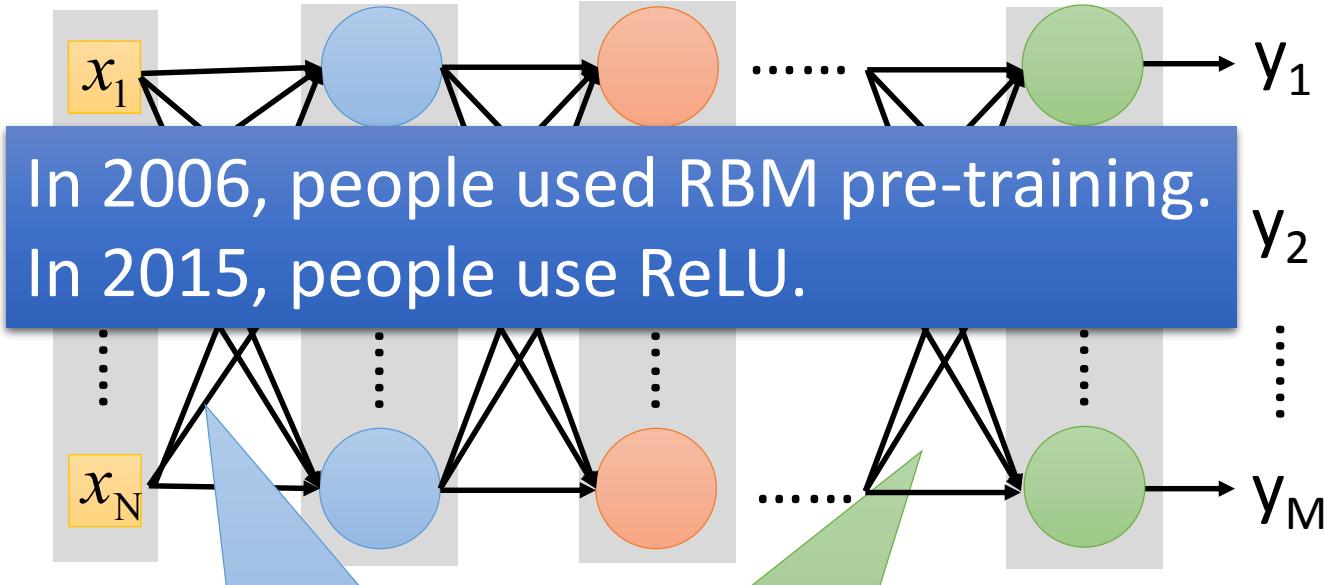


[Xavier Glorot, AISTATS'11]
[Andrew L. Maas, ICML'13]
[Kaiming He, arXiv'15]

Reason:

1. Fast to compute
2. Biological reason
3. Infinite sigmoid with different biases
4. Vanishing gradient problem

Vanishing Gradient Problem



Smaller gradients

Learn very slow

Almost random

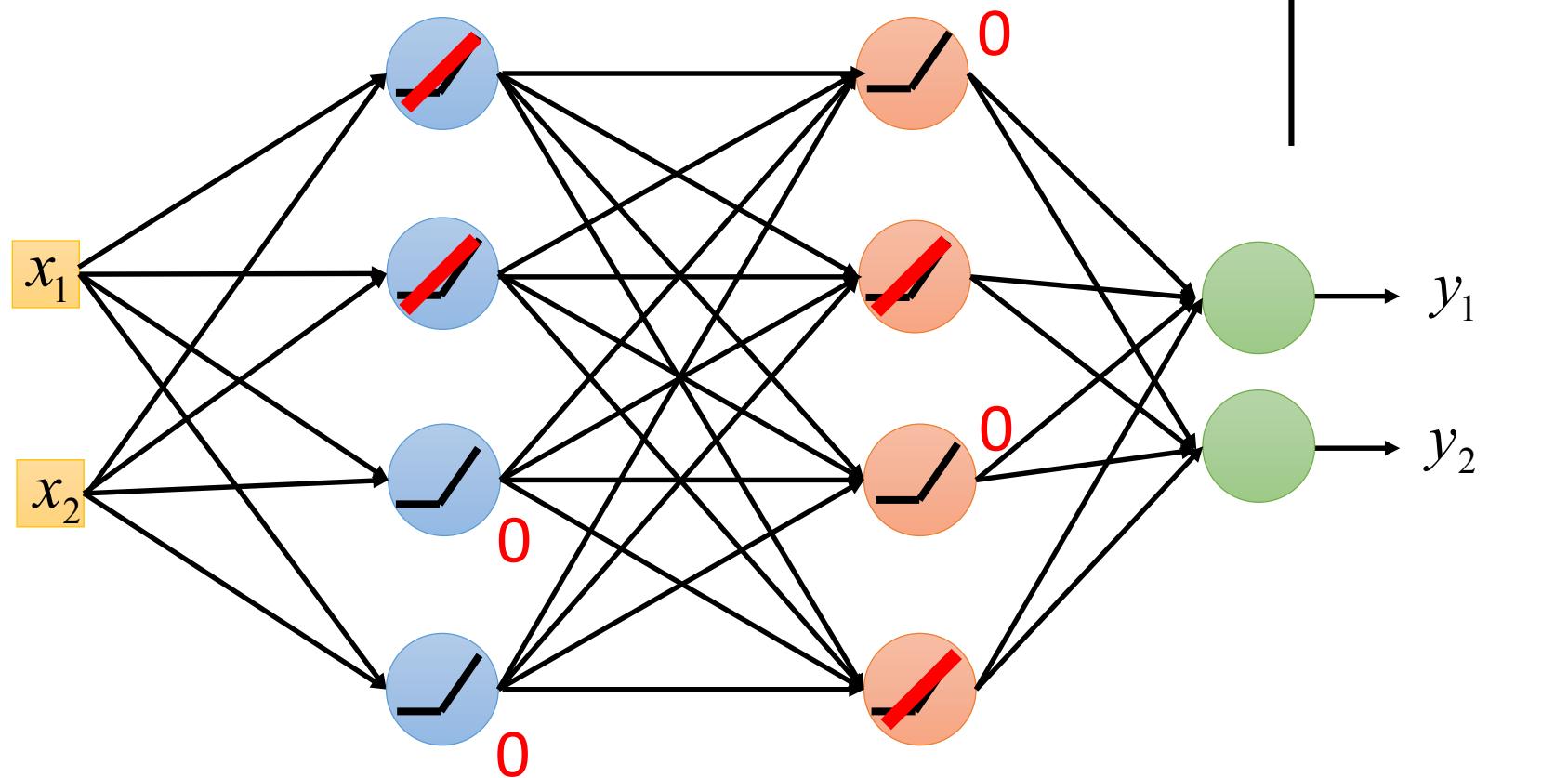
Larger gradients

Learn very fast

Already converge

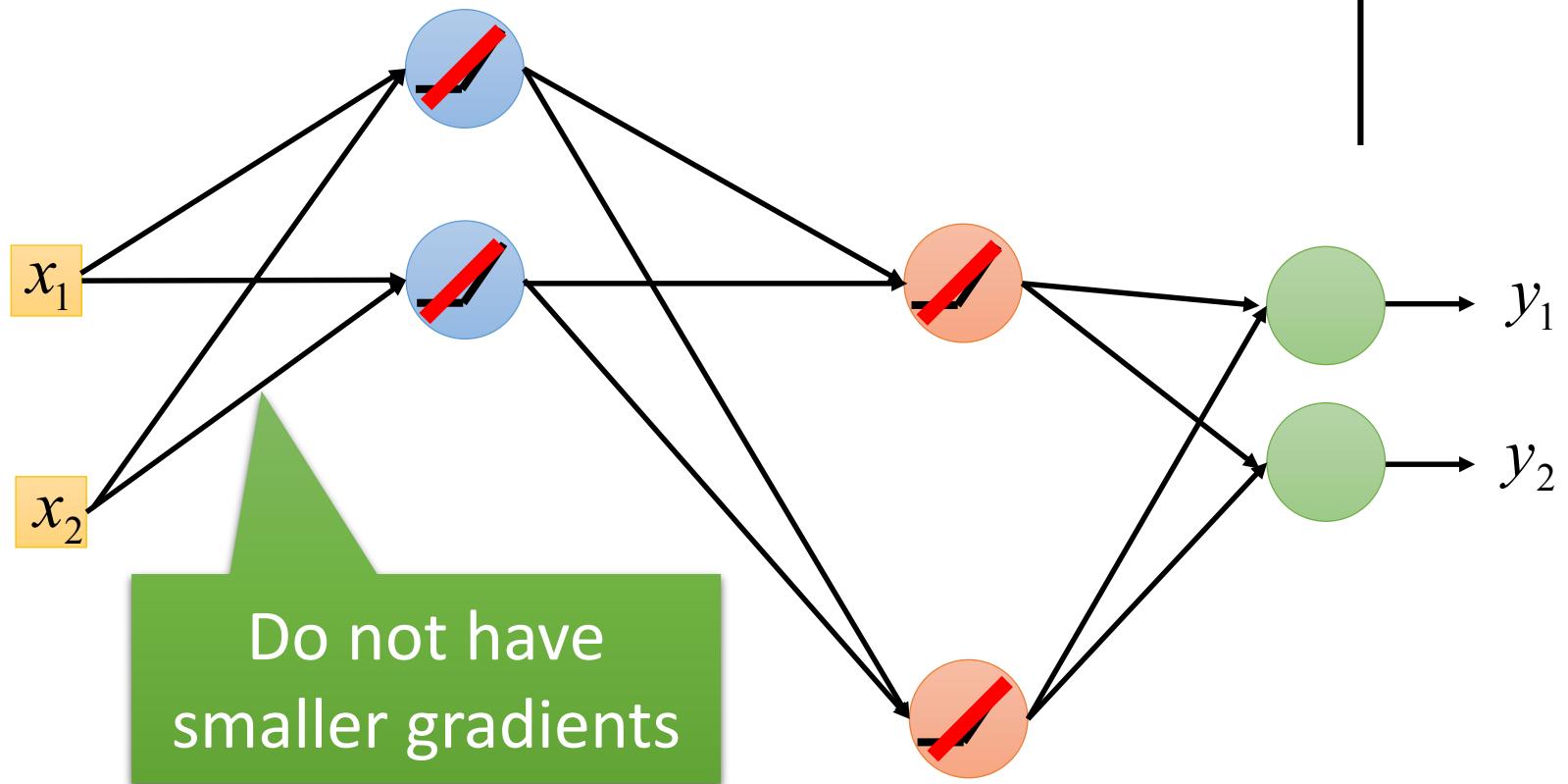
based on random!?

ReLU



ReLU

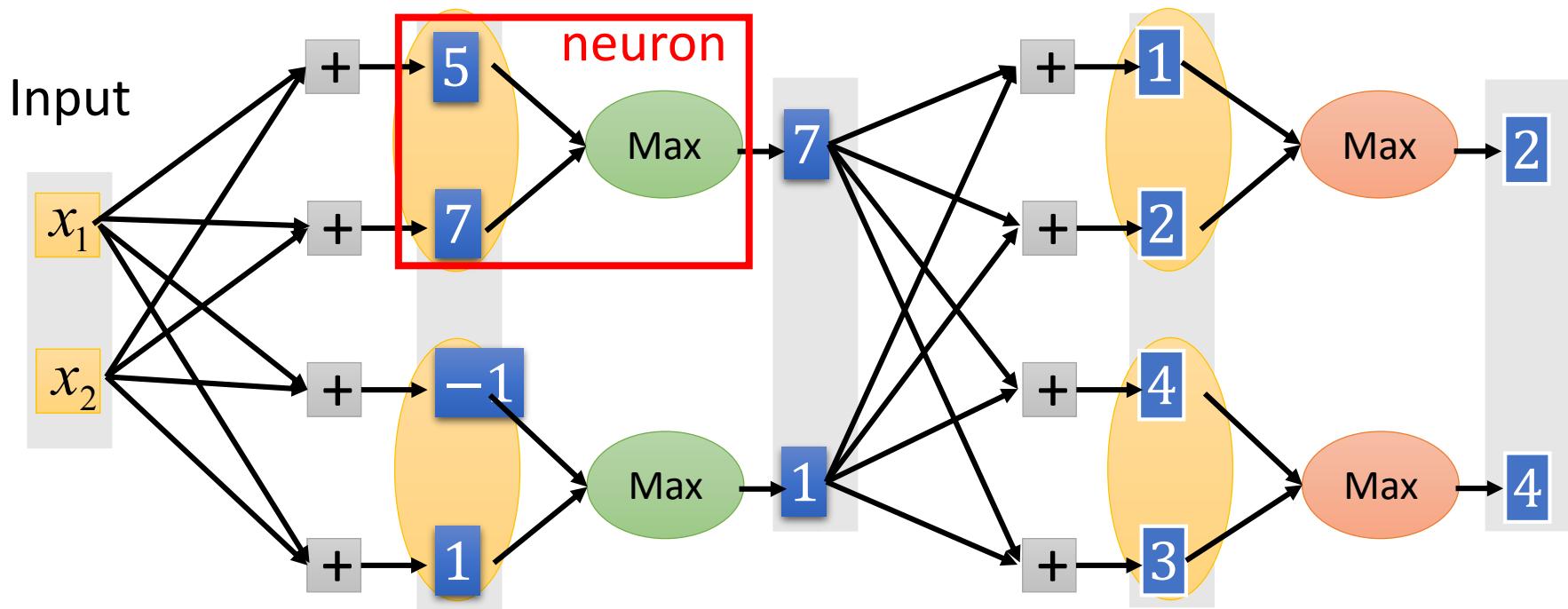
A Thinner linear network



Maxout

ReLU is a special cases of Maxout

- Learnable activation function [Ian J. Goodfellow, ICML'13]



You can have more than 2 elements in a group.

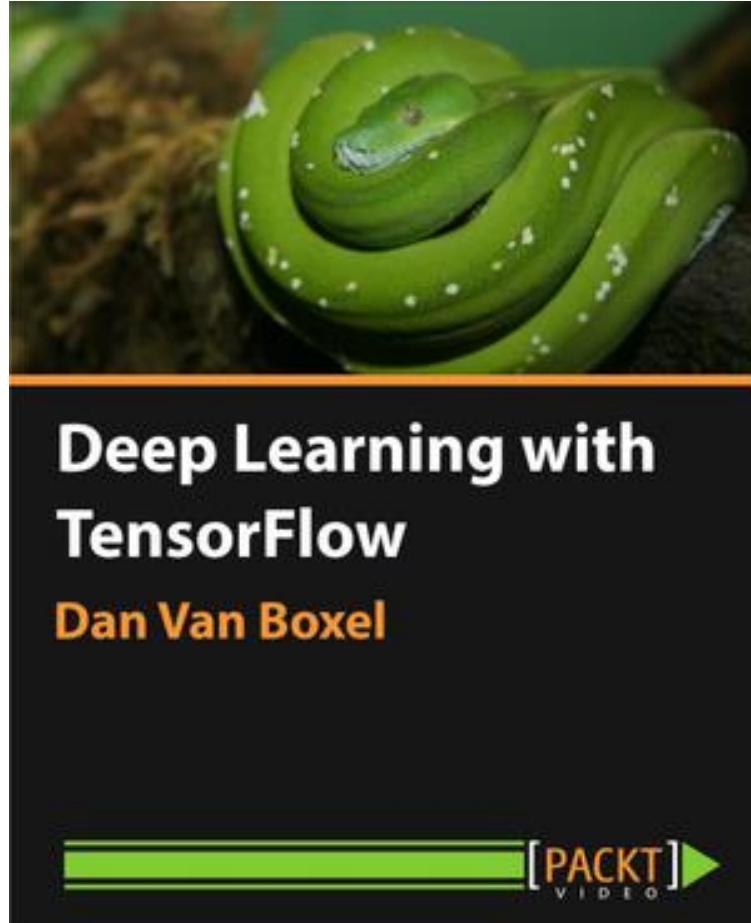
Adaptive learning rate

Different algorithms to optimize

- Adagrad [John Duchi, JMLR'11]
- RMSprop
 - <https://www.youtube.com/watch?v=O3sxAc4hxZU>
- Adadelta [Matthew D. Zeiler, arXiv'12]
- Adam [Diederik P. Kingma, ICLR'15]
- AdaSecant [Caglar Gulcehre, arXiv'14]
- “No more pesky learning rates” [Tom Schaul, arXiv'12]



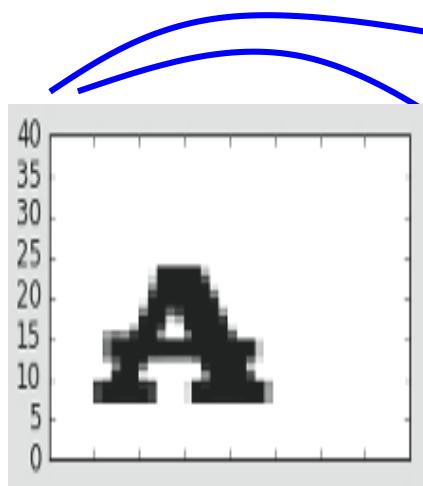
Font type Recognition





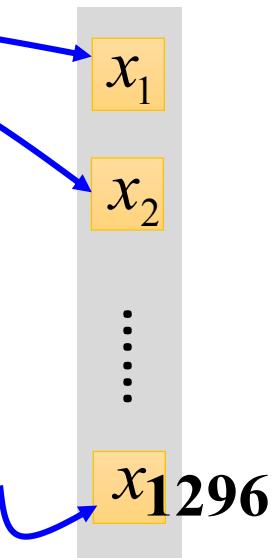
Font type Recognition

Input

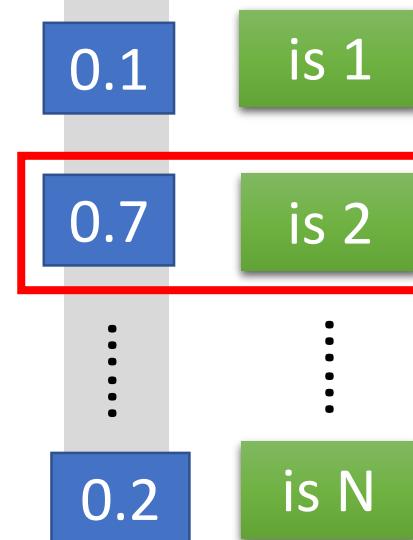


$$36 \times 36 = 1296$$

reShape



Output



Font type
is "2"



Elements [0:255]

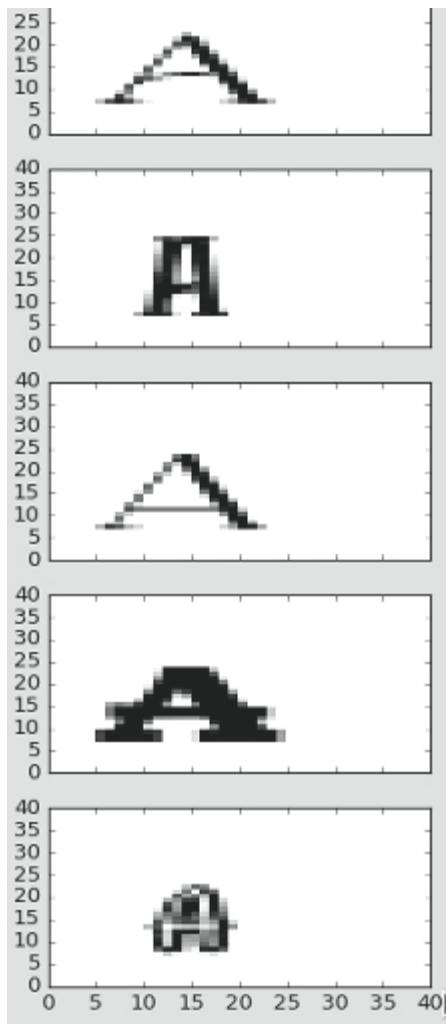
White pixel \rightarrow 0

Black pixel \rightarrow 255

Each dimension
represents the
confidence of a char
type

OHE
One-Hot-Encoding

One-hot-encoding (OHE)

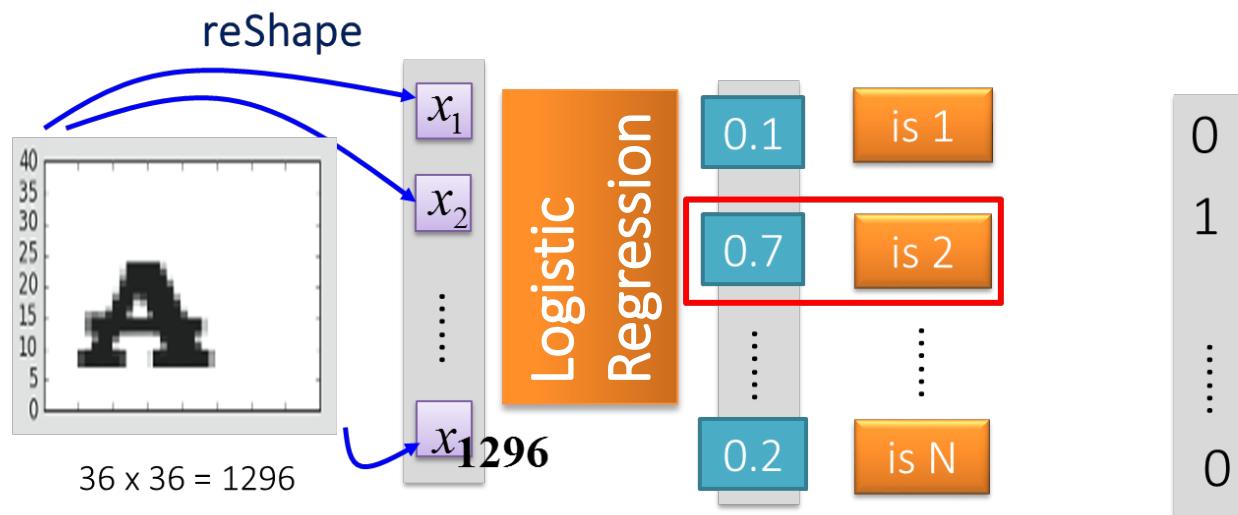


Categories	Number of Categories (N=5)				
Type 1	1	0	0	0	0
Type 2	0	1	0	0	0
Type 3	0	0	1	0	0
Type 4	0	0	0	1	0
Type 5	0	0	0	0	1



Font type Recognition using Logistic Regression

MSTC_FontReco_LogisticReg.ipynb



Elements [0:255]

White pixel → 0

Black pixel → 255



MSTC_FontReco_FeedForward.ipynb

Available at:

https://github.com/MasterMSTC/DeepLearning_TF/

Or

A screenshot of the Data Science Experience web interface. At the top, there's a navigation bar with a teal 'DSE' logo, the text 'Data Science Experience', and two tabs: 'Projects' (which is underlined in teal) and 'Tools'. Below the navigation bar, the path 'My Projects > MSTC_DeepLearning' is shown. The main content area has tabs for 'Overview', 'Analytics Assets', 'Data Assets', and 'Notebooks' (which is underlined in blue). Under the 'Notebooks' tab, there's a link 'view all (5)'. A table follows, with columns for 'NAME' and preview icons. The rows show three notebook entries: 'MSTC_FontReco_LogisticReg', 'MSTC_FontReco_CNN', and 'MSTC_FontReco_FeedForward'.

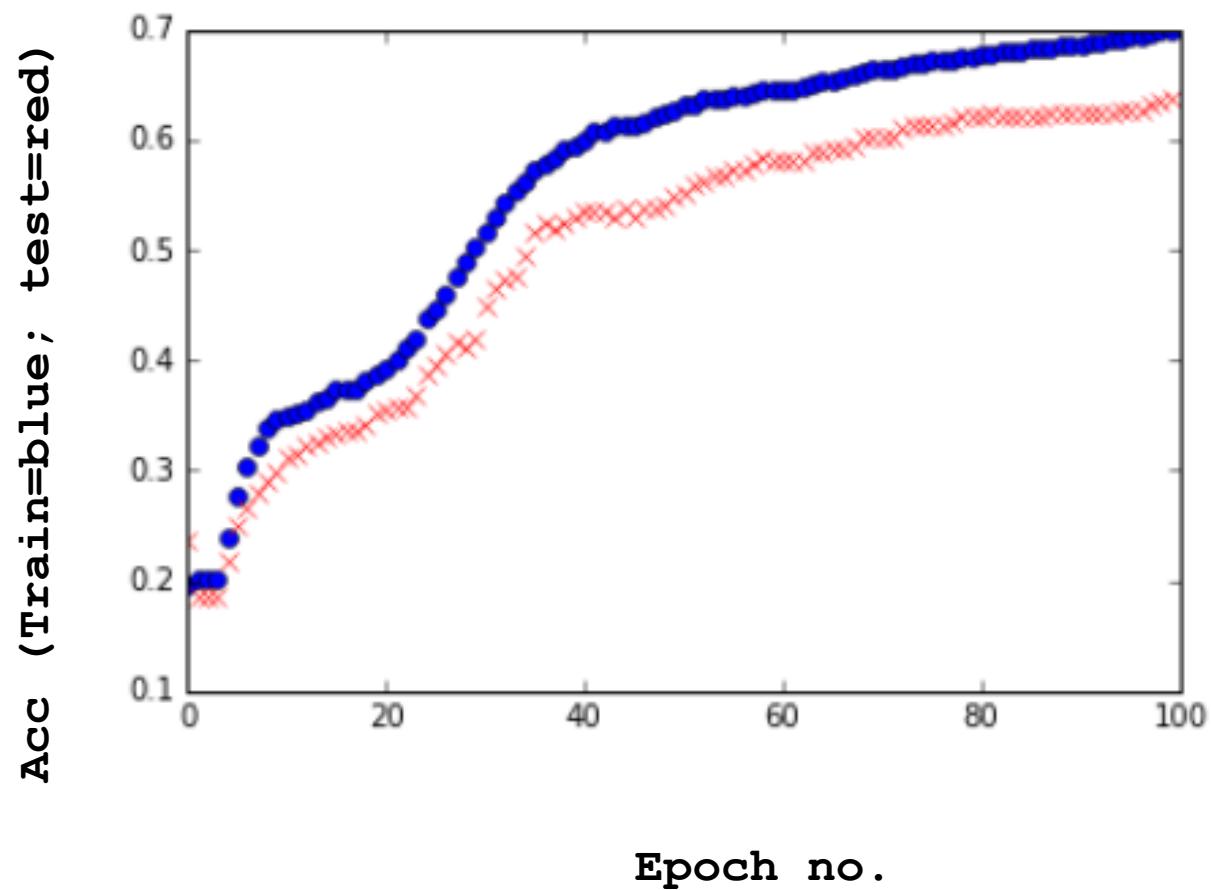
NAME
MSTC_FontReco_LogisticReg
MSTC_FontReco_CNN
MSTC_FontReco_FeedForward



MSTC_FontReco_FeedForward.ipynb

In all FontReco examples we will use TF interactive sessions:

```
sess = tf.InteractiveSession()
```





MSTC_FontReco_FeedForward.ipynb

```
# These will be inputs.... pixels, flattened
x = tf.placeholder("float", [None, 1296])
## Known labels
y_ = tf.placeholder("float", [None,5])

# Hidden layer 1
num_hidden1 = 128
W1 = tf.Variable(tf.truncated_normal([1296,num_hidden1],
                                     stddev=1./math.sqrt(1296)))
b1 = tf.Variable(tf.constant(0.1,shape=[num_hidden1]))
h1 = tf.sigmoid(tf.matmul(x,W1) + b1)

# Hidden Layer 2
num_hidden2 = 32
W2 = tf.Variable(tf.truncated_normal([num_hidden1,
                                     num_hidden2],stddev=2./math.sqrt(num_hidden1)))
b2 = tf.Variable(tf.constant(0.2,shape=[num_hidden2]))
h2 = tf.sigmoid(tf.matmul(h1,W2) + b2)
```



MSTC_FontReco_FeedForward.ipynb

Output Layer

```
W3 = tf.Variable(tf.truncated_normal([num_hidden2, 5],  
                                     stddev=1./math.sqrt(5)))  
b3 = tf.Variable(tf.constant(0.1, shape=[5]))
```

Just initialize

```
sess.run(tf.initialize_all_variables())
```

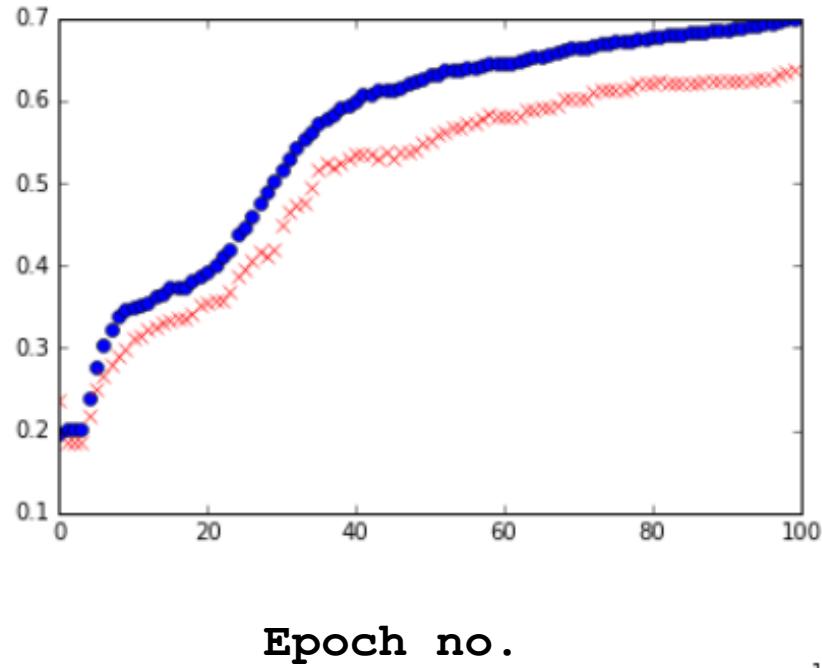
Define model

```
y = tf.nn.softmax(tf.matmul(h2, W3) + b3)
```

```
### End model specification, begin training code
```

Acc (Train=blue; test=red)

Logistic Regression

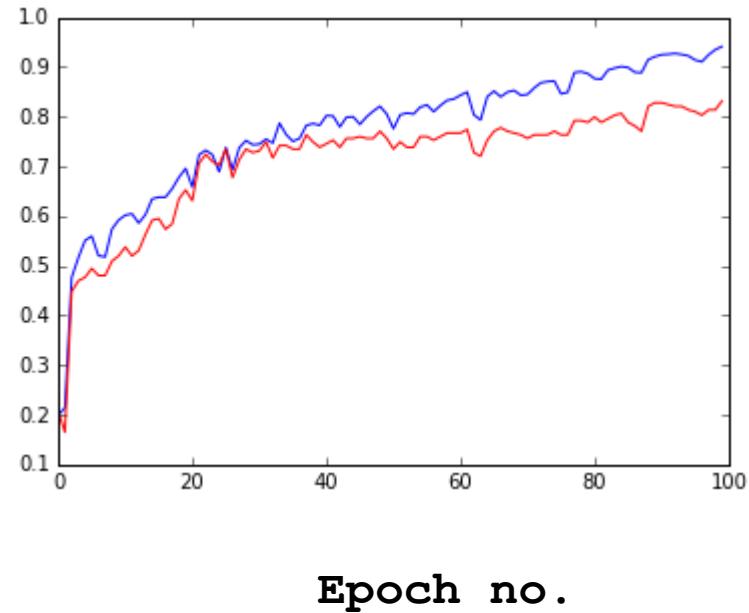


Epoch no.

```
train_step =  
tf.train.GradientDescentOptimizer(0.55).  
minimize(cross_entropy)
```

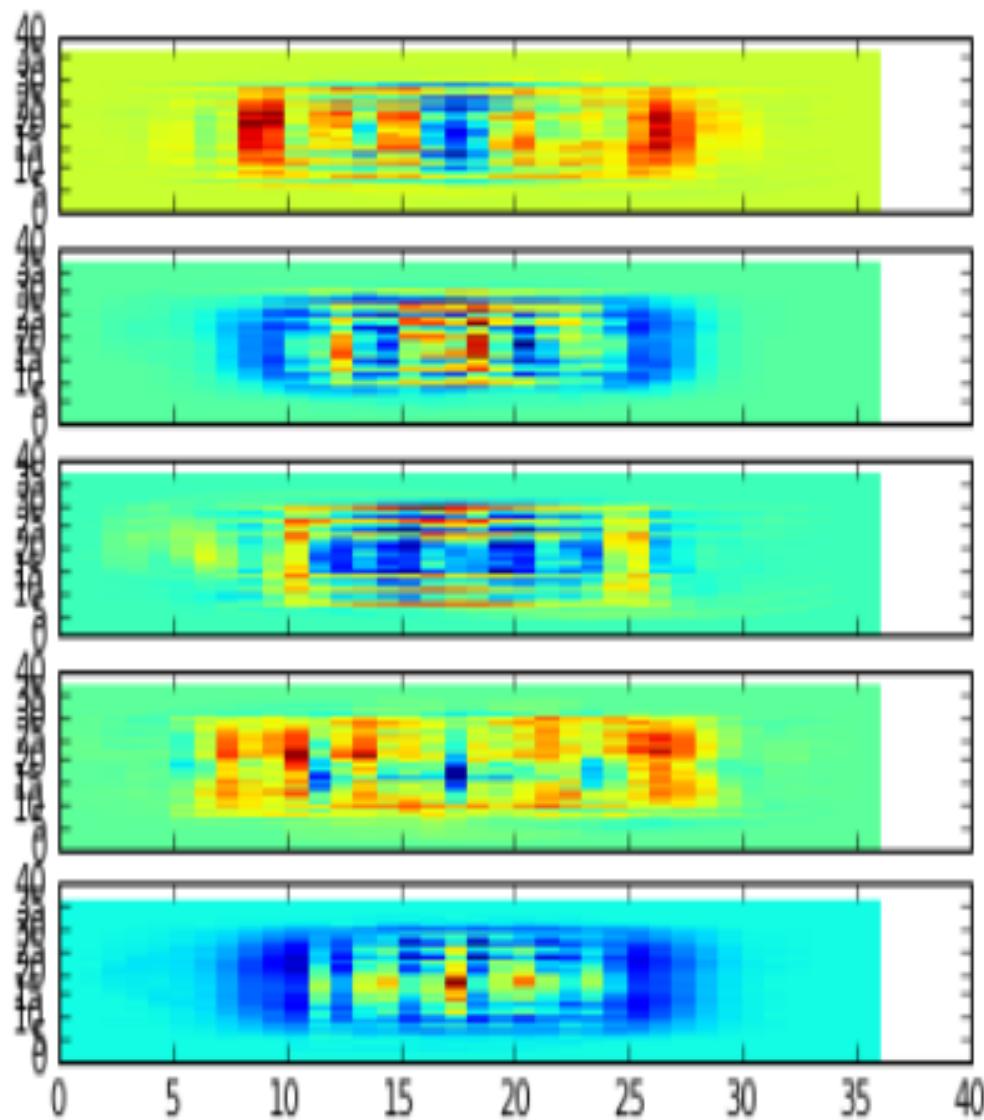
Acc (Train=blue;
test=red)

Feed Forward



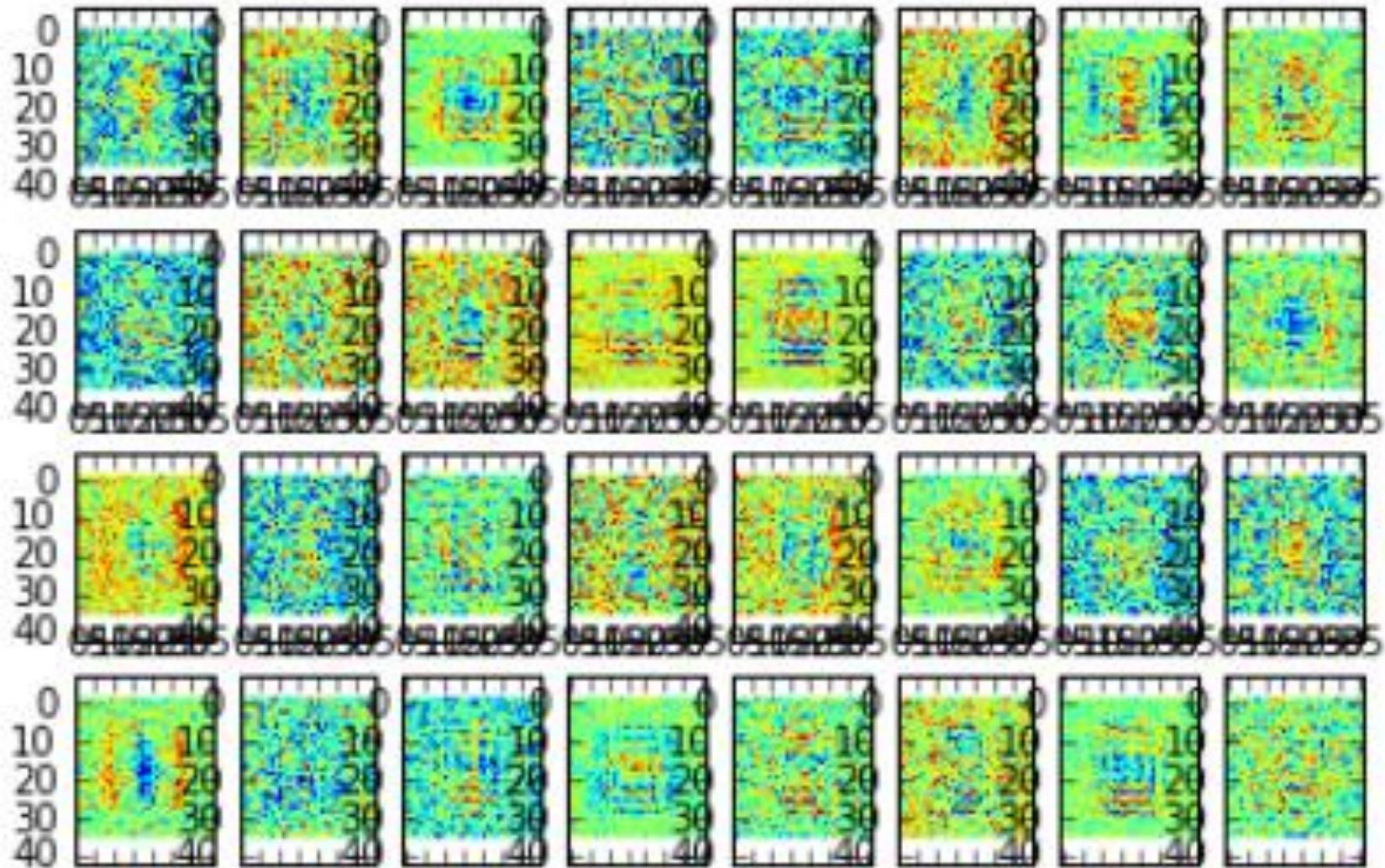
Epoch no.

Logistic Regression



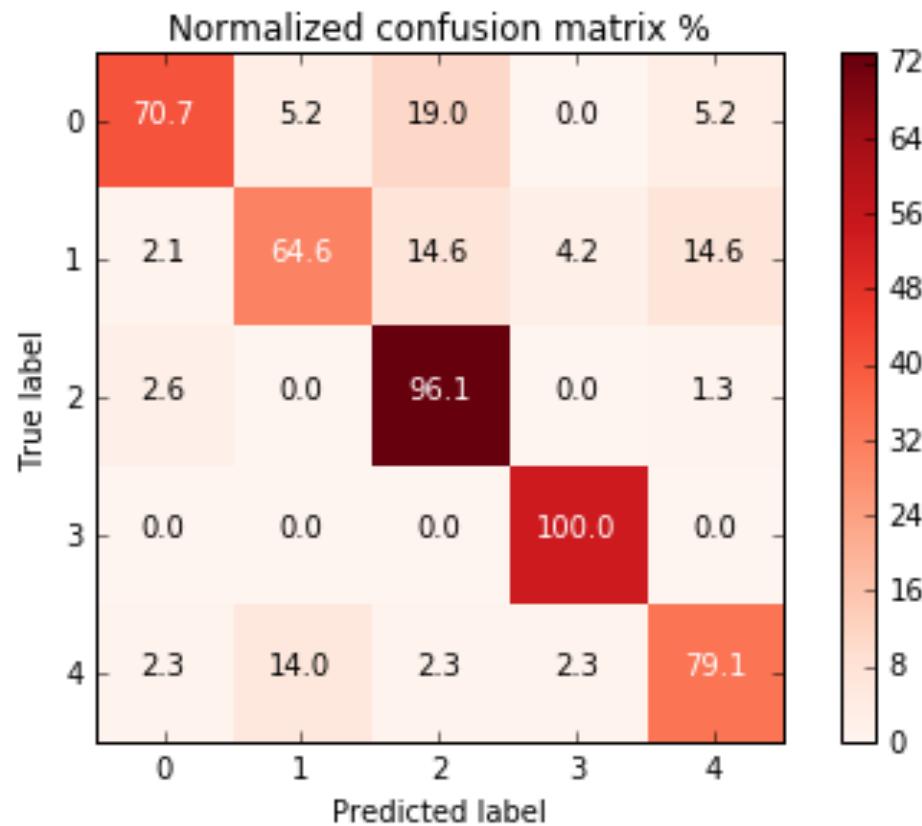
Feed Forward

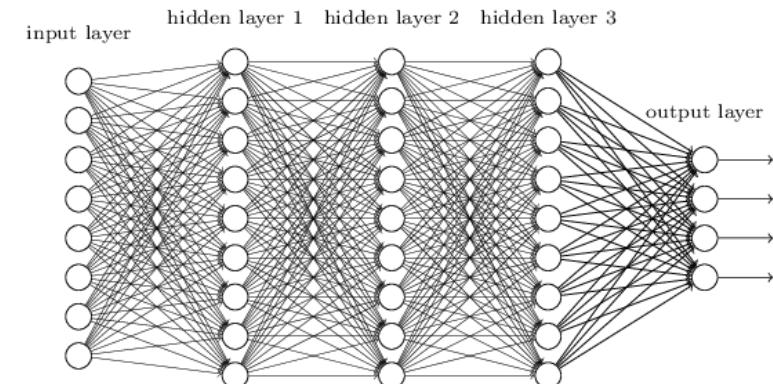
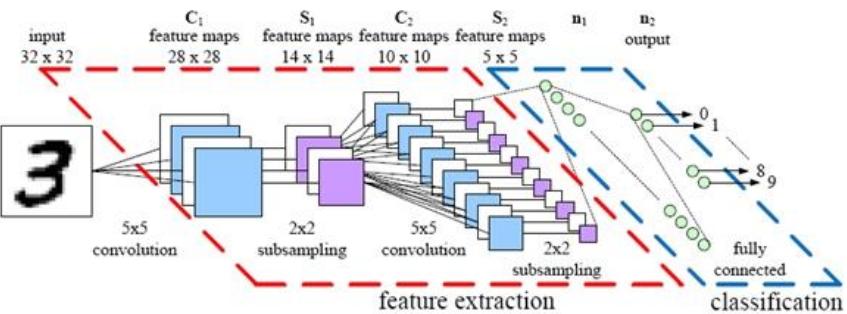
05.08.08.85 05.08.08.85 05.08.08.85 05.08.08.85 05.08.08.85 05.08.08.85 05.08.08.85 05.08.08.85



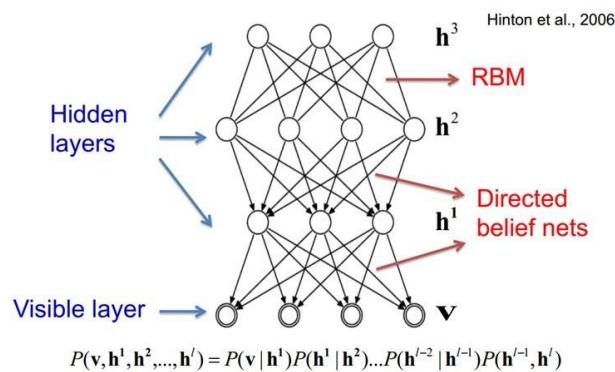
Font Recognition Results

Feed Forward

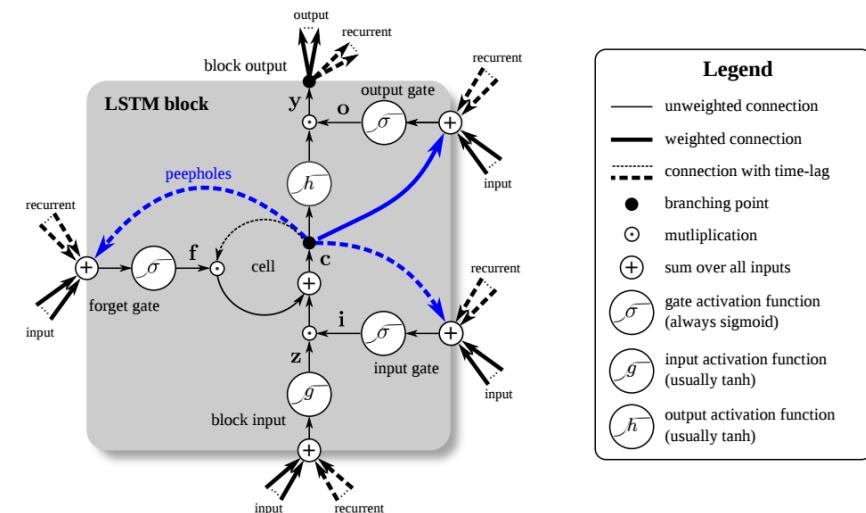
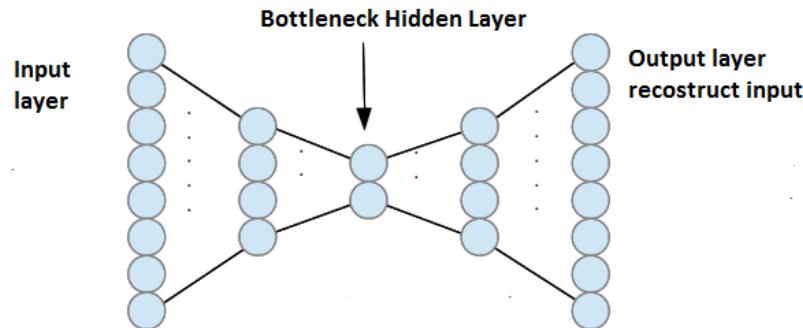




DBN structure



Deep Learning Architectures



Legend

- unweighted connection
- weighted connection
- - - connection with time-lag
- branching point
- multiplication
- (+) sum over all inputs
- (σ) gate activation function (always sigmoid)
- (g) input activation function (usually tanh)
- (h) output activation function (usually tanh)

Unsupervised ?

Breakthrough

Deep Belief Networks (DBN)

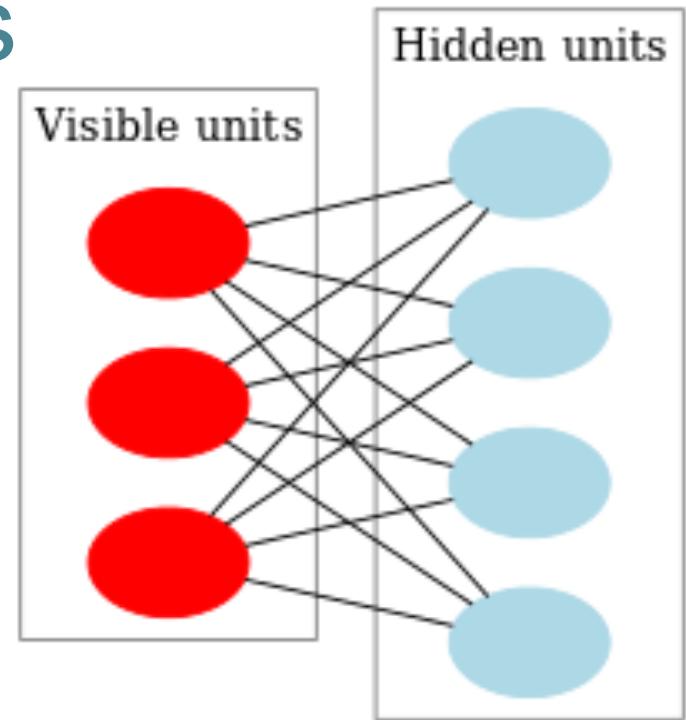
Hinton, G. E, Osindero, S., and Teh, Y. W. (2006).
A fast learning algorithm for deep belief nets.
Neural Computation, 18:1527-1554.

Autoencoders

Bengio, Y., Lamblin, P., Popovici, P., Larochelle, H. (2007).
Greedy Layer-Wise Training of Deep Networks,
Advances in Neural Information Processing Systems 19

Deep Belief Networks

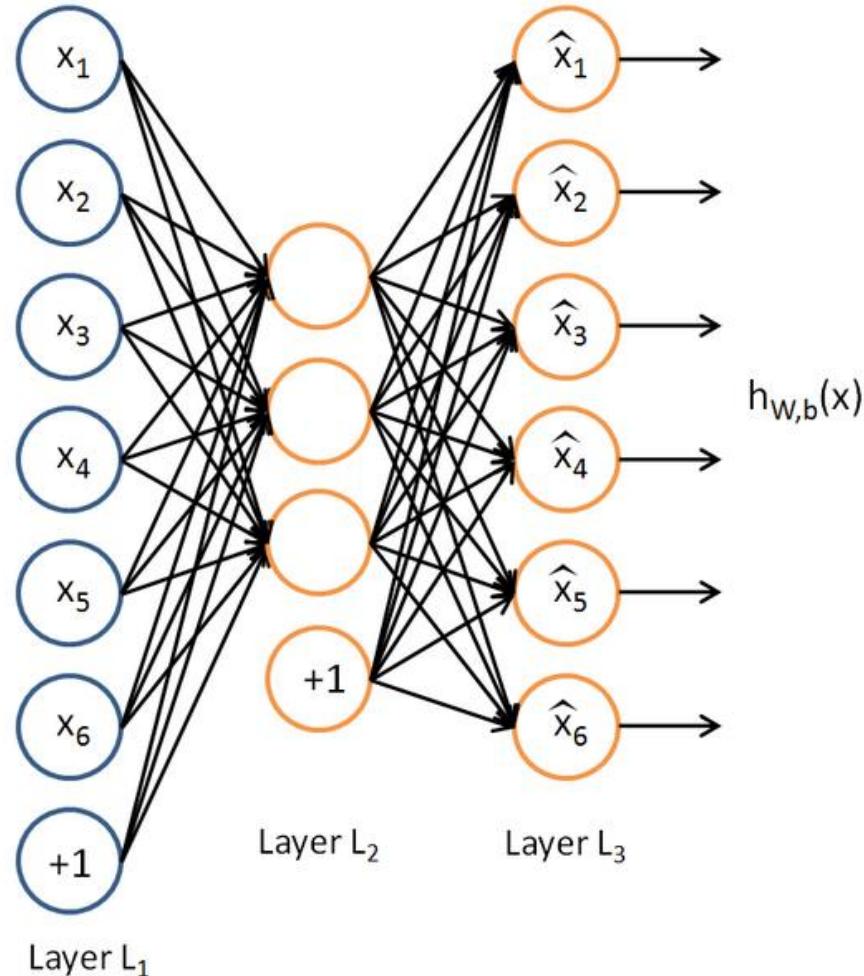
- DBN is a probabilistic, generative model made up of multiple layers of hidden units.
- DBN can be used to generatively pre-train a DNN
The learned DBN weights as the initial DNN weights.
 - Back-propagation later for fine tuning of these weights.



A DBN can be efficiently trained in an **unsupervised, layer-by-layer manner**, where the layers are typically made of ***restricted Boltzmann machines (RBM)***.

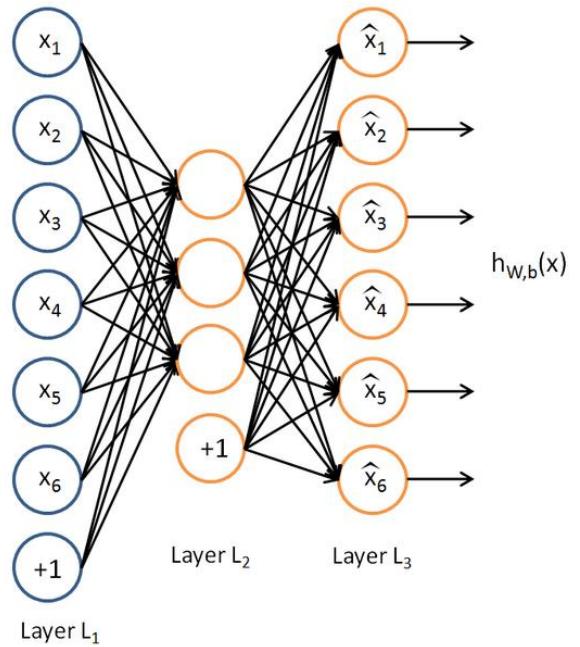
RBM: undirected, generative energy-based model with a "visible" input layer and a hidden layer, and connections between the layers but not within layers

Autoencoder

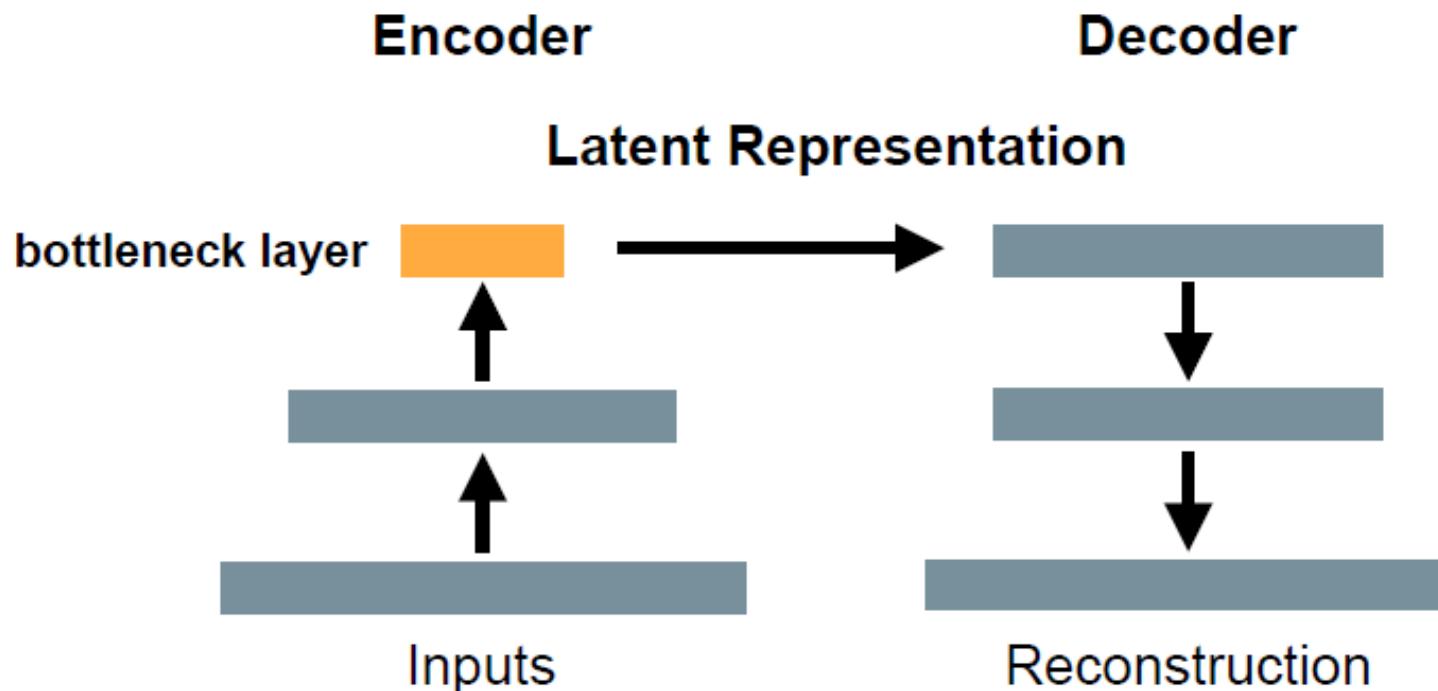


Autoencoder

- Reduce dimensionality, something in common with PCA, but focus more on the reconstruction of the input
- Retrieve high level features using **unsupervised learning**

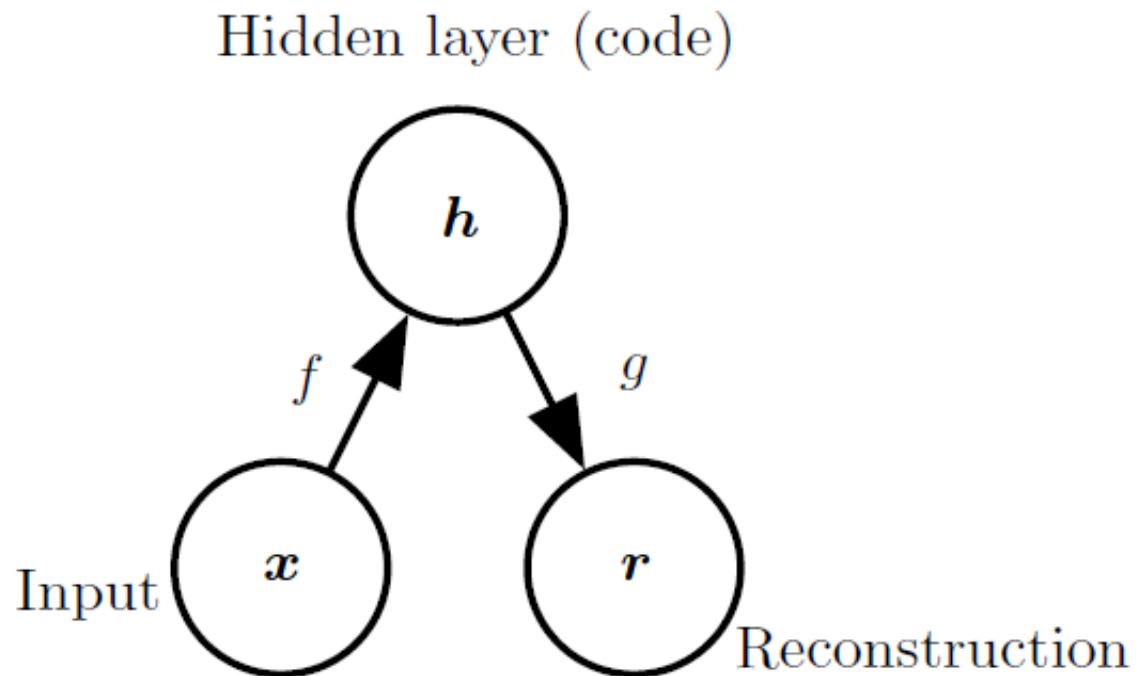


Autoencoders for Representation Learning



$$L = (x - \hat{x})^2$$

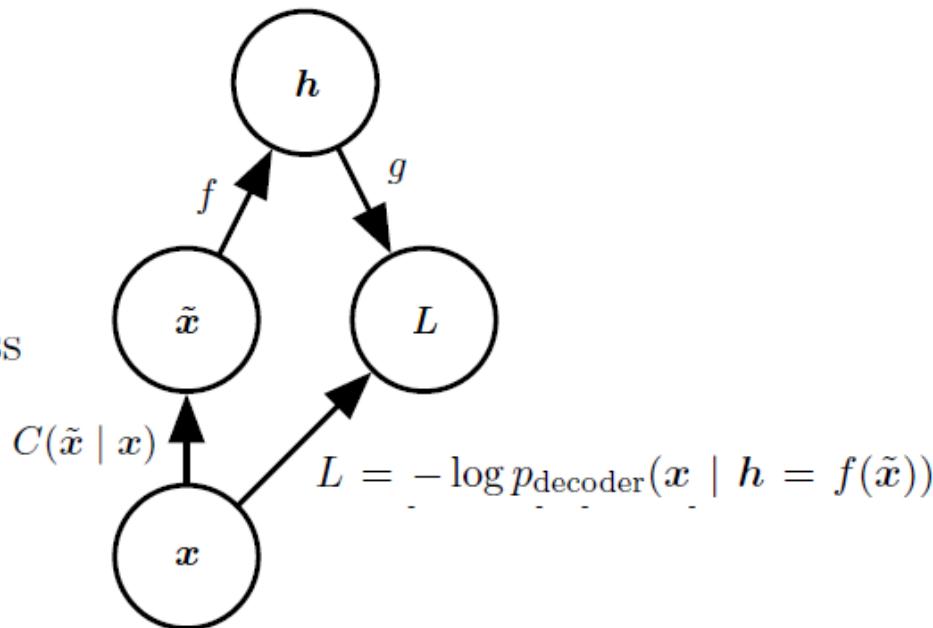
Structure of an Autoencoder



Avoiding Trivial Identity

Denoising Autoencoder

C : corruption process
(introduce noise)



- Vincent et al. (2010), “*a good representation* is one that can be obtained **robustly** from a **corrupted input** and that will be useful for **recovering** the corresponding **clean input**.”

Compression as implicit generative modeling



Stack more layers
stack more layers
stack more layers
stack more layers
stack more layers



Human: *i am seeing an error related to vpn*

Machine: *what is the error message that you are getting when connecting to vpn using network connect ?*

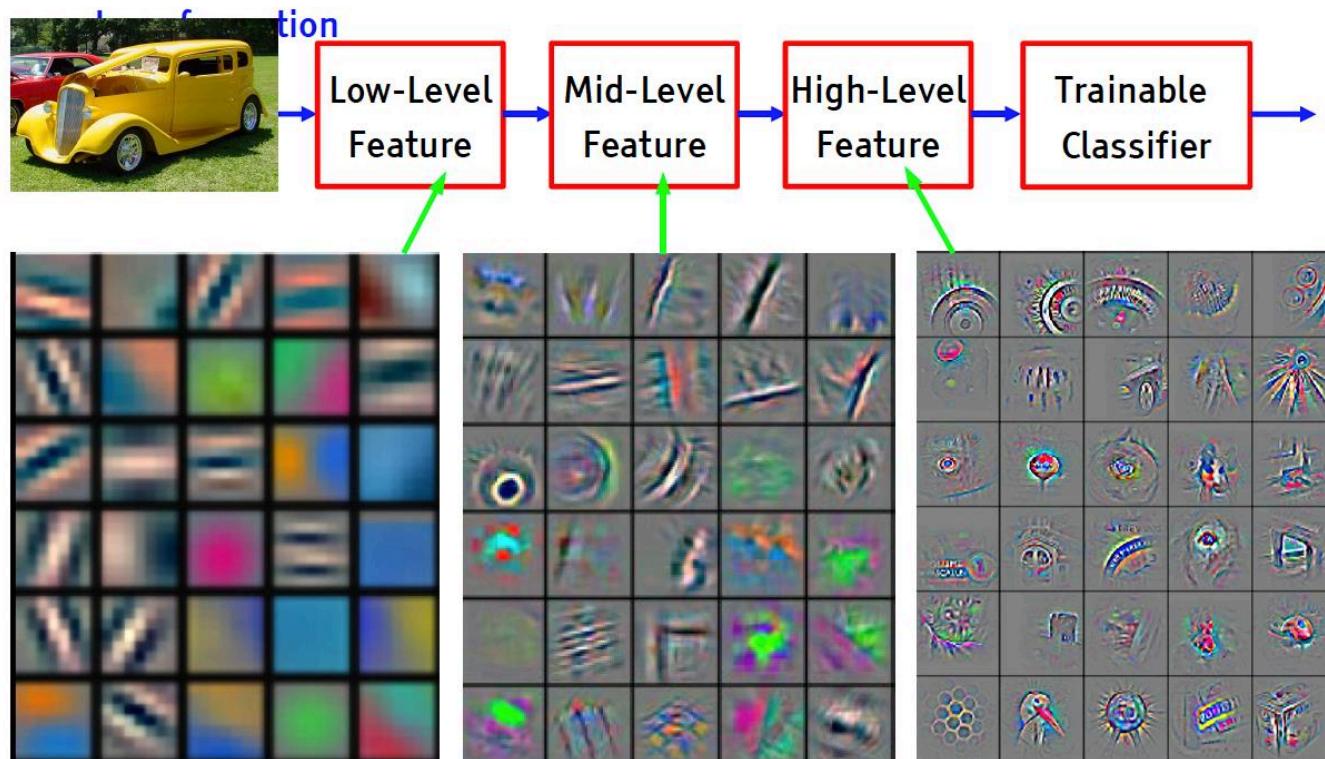
Human: *connection refused or something like that*

Machine: *may i know the version of network connect you connect ?*

Supervised:

- Feed Forward
- CNN: Convolutional Neural Network
- RNN: Recurrent Neural Network, LSTM, GRU..
- ...

Deep Learning = Learning Hierarchical Representations



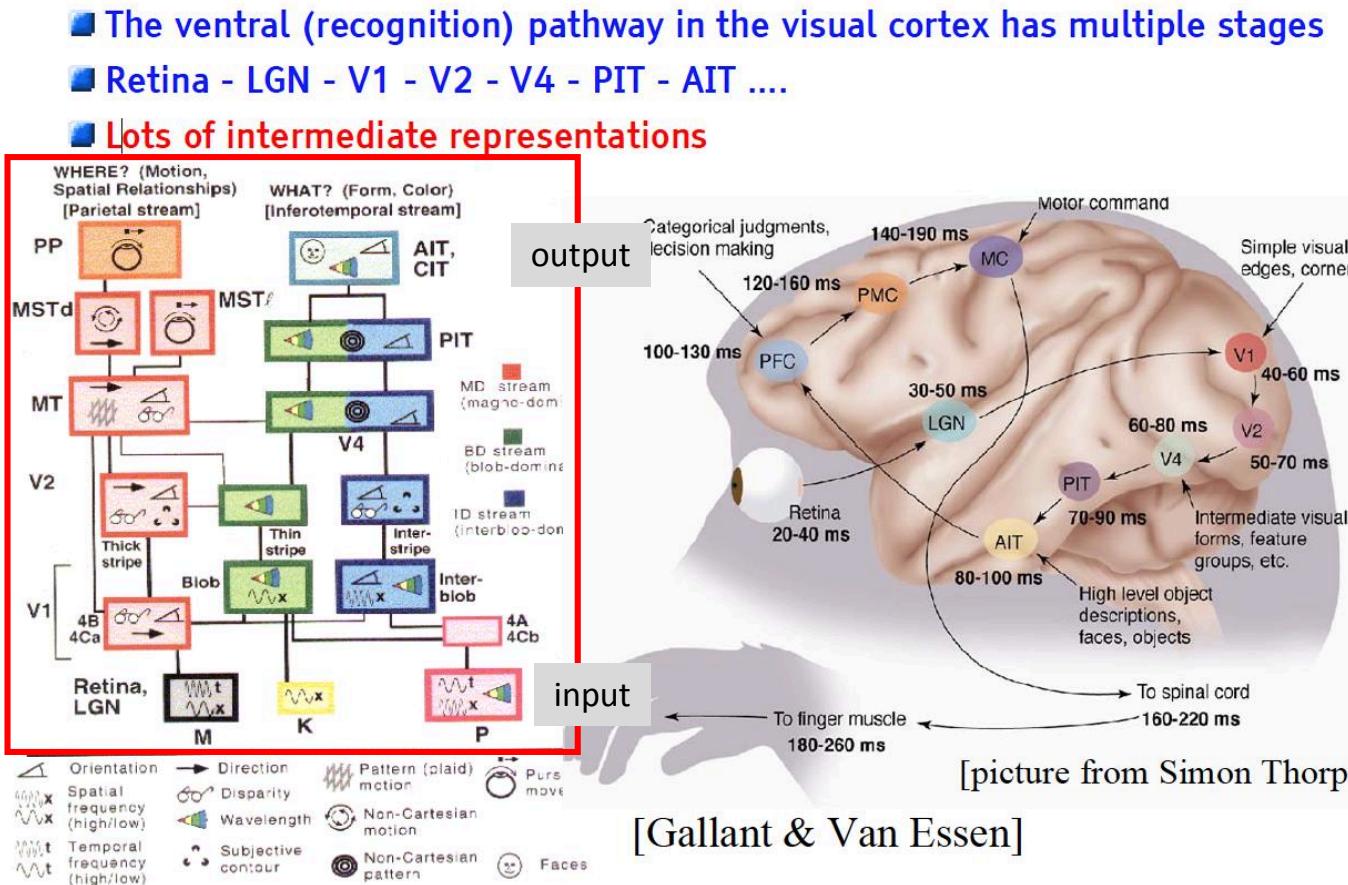
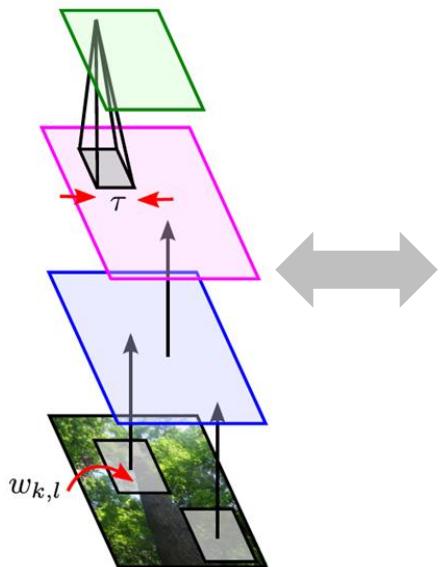
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

CNN: Convolutional Neural Network

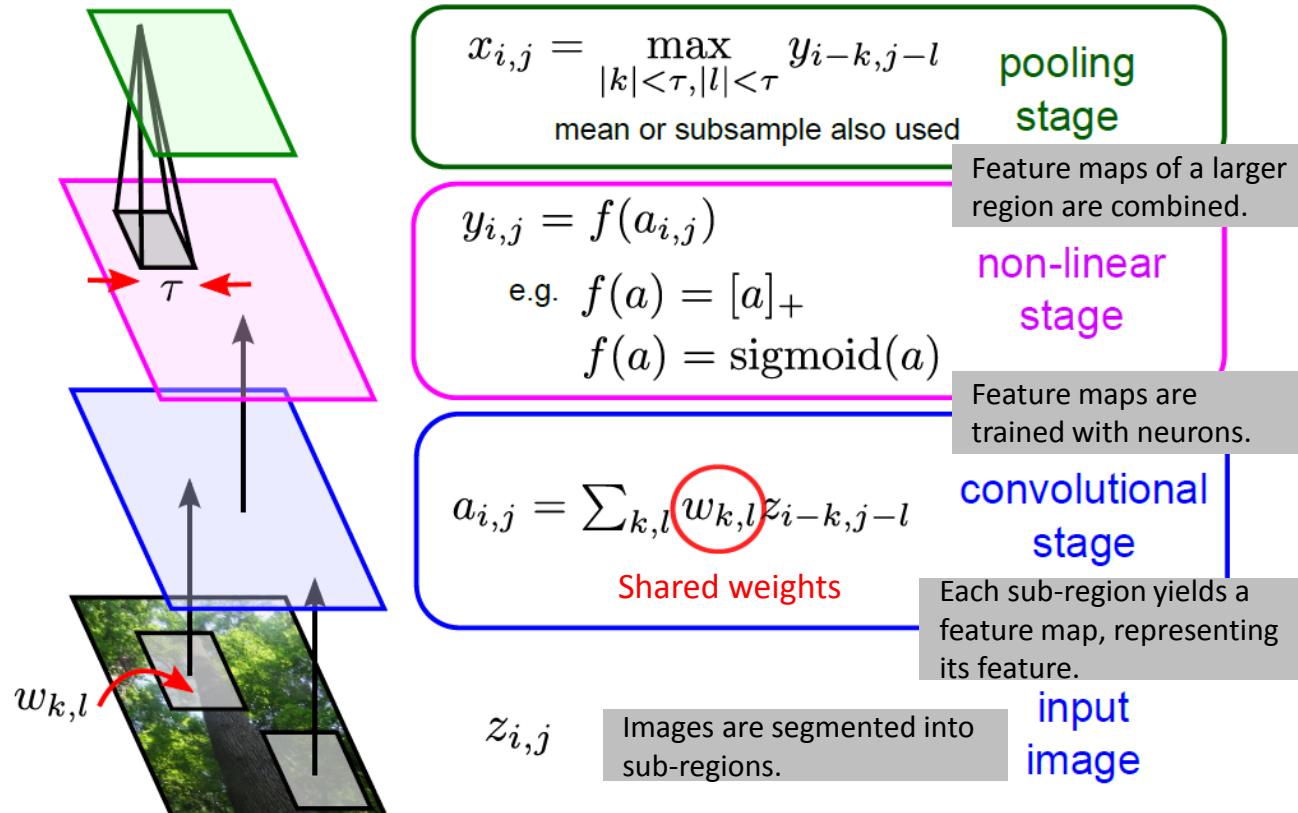
- ***Convolutional Neural Network*** organizes neurons based on animal's visual cortex system, which allows for learning patterns at both local level and global level.
 - Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, 86(11):2278-2324, November 1998

The Mammalian Visual Cortex Inspires CNN

Convolutional Neural Net

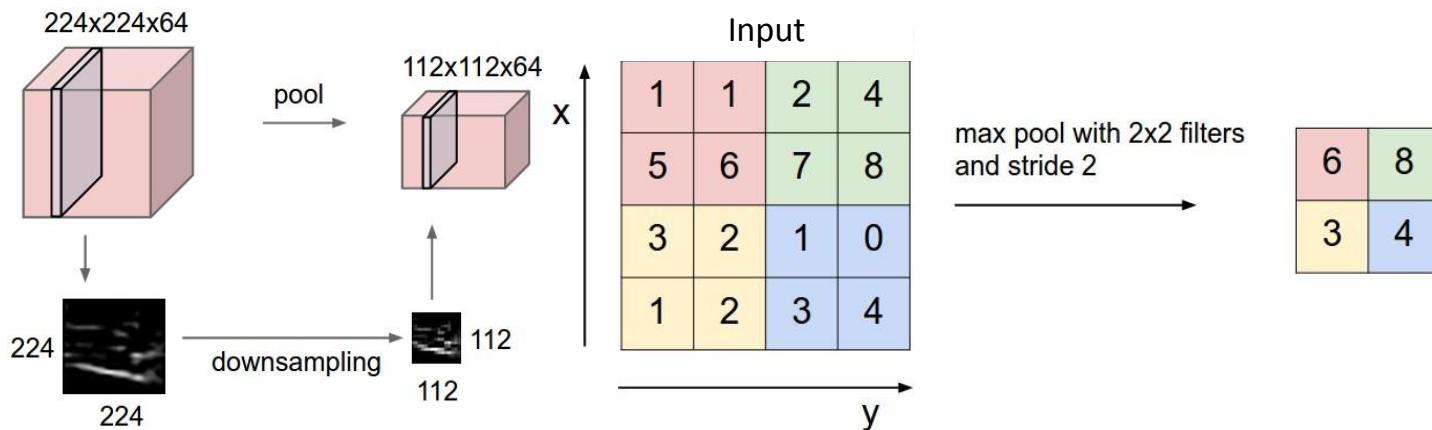


Building-blocks for CNN's

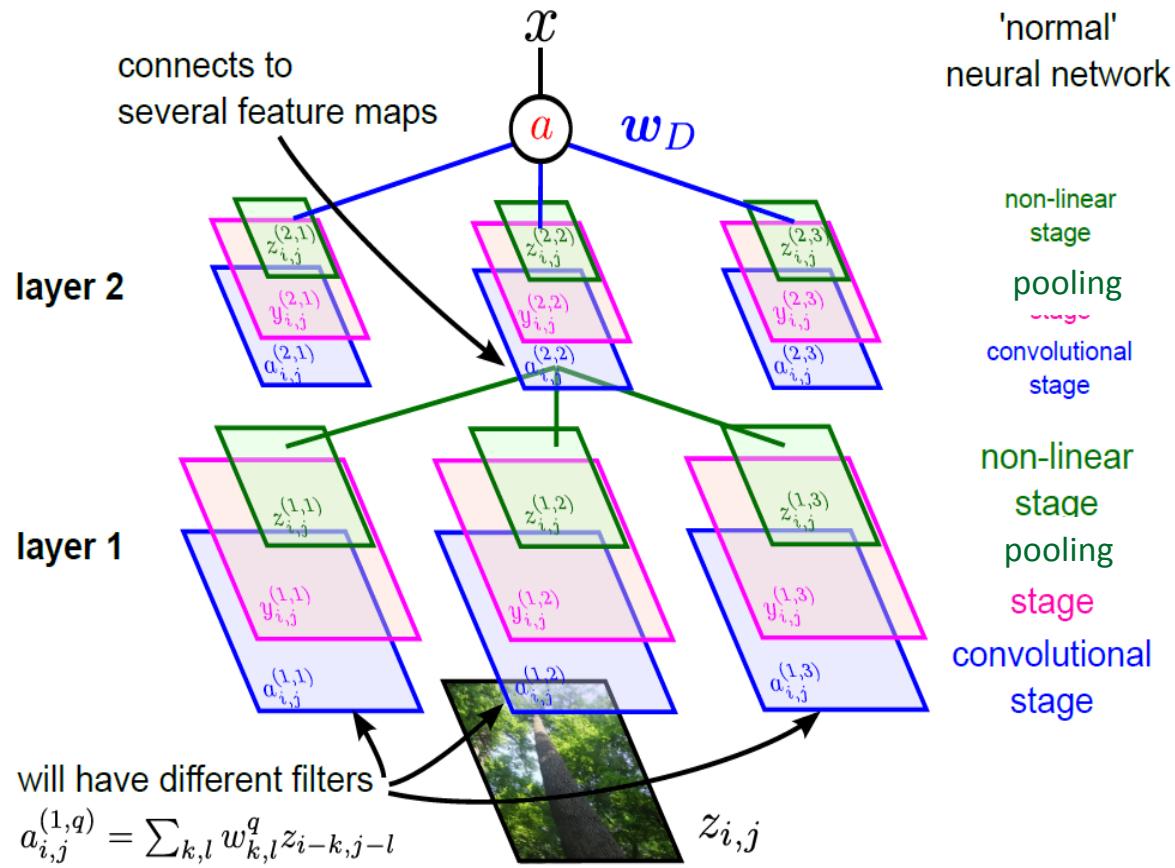


CNN Architecture: Pooling Layer

- Intuition: to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting
- Pooling partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum value of the features in that region.



Full CNN



See: <https://cs231n.github.io/convolutional-networks/>
<https://deeplearning4j.org/convolutionalnets.html>



MSTC_FontReco_CNN.ipynb (I)

Conv layer 1

```
num_filters = 4
```

```
winx = 5
```

```
winy = 5
```

```
W1 = tf.Variable(tf.truncated_normal([winx, winy, 1 , num_filters],  
stddev=1./math.sqrt(winx*winy)))
```

```
b1 = tf.Variable(tf.constant(0.1, shape=[num_filters]))
```

5x5 convolution, pad with zeros on edges

```
xw = tf.nn.conv2d(x_im, W1, strides=[1, 1, 1, 1], padding='SAME')
```

```
h1 = tf.nn.relu(xw + b1)
```

2x2 Max pooling, no padding on edges

```
p1 = tf.nn.max_pool(h1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],  
padding='VALID')
```



MSTC_FontReco_CNN.ipynb (II)

```
# Need to flatten convolutional output for use in dense layer
```

```
p1_size = np.product( [s.value for s in p1.get_shape()[1:]])  
p1f = tf.reshape(p1, [-1, p1_size ])
```

```
# Dense layer
```

```
num_hidden = 32
```

```
W2 = tf.Variable(tf.truncated_normal( [p1_size, num_hidden],  
stddev=2./math.sqrt(p1_size)))
```

```
b2 = tf.Variable(tf.constant(0.2, shape=[num_hidden]))
```

```
h2 = tf.nn.relu(tf.matmul(p1f,W2) + b2)
```

```
# Output Layer
```

```
W3 = tf.Variable(tf.truncated_normal( [num_hidden, 5],  
stddev=1./math.sqrt(num_hidden)))
```

```
b3 = tf.Variable(tf.constant(0.1,shape=[5]))
```

```
....
```



MSTC_FontReco_CNN.ipynb (III)

```
keep_prob = tf.placeholder("float")
h2_drop = tf.nn.dropout(h2, keep_prob)
```

Just initialize

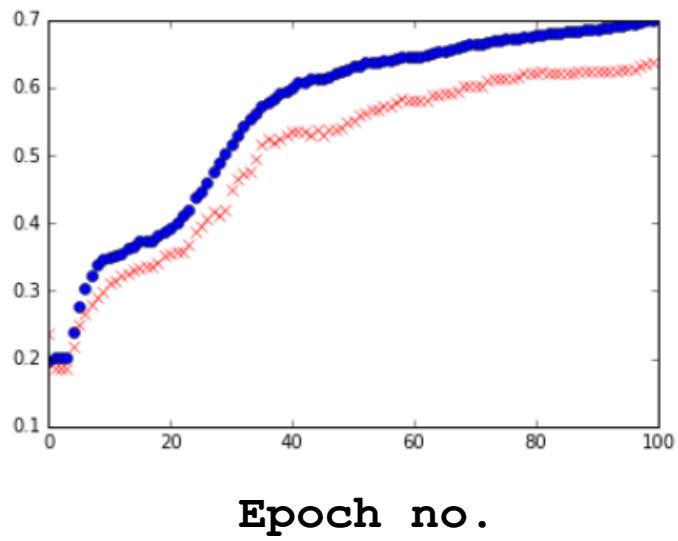
```
sess.run(tf.initialize_all_variables())
```

Define model

```
y = tf.nn.softmax(tf.matmul(h2_drop,W3) + b3)
```

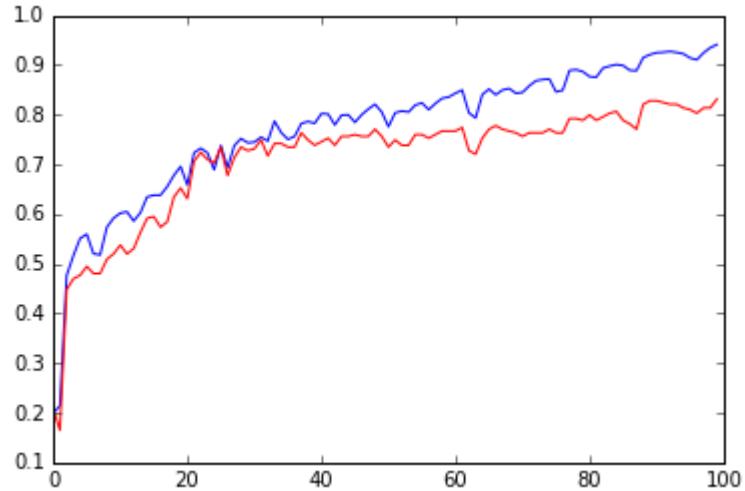
Acc (Train=blue; test=red)

Logistic Regression



Epoch no.

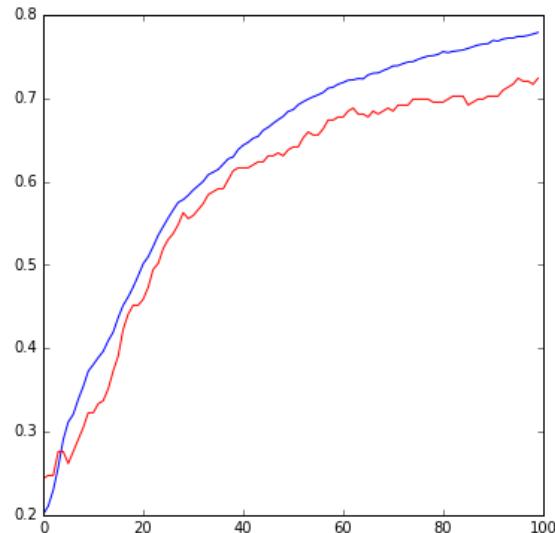
Feed Forward



Epoch no.

CNN

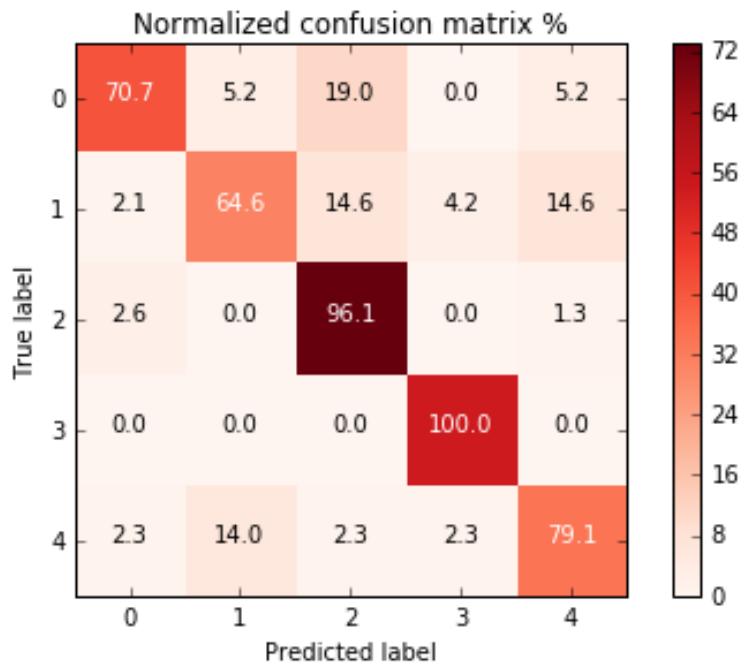
Acc (Train=blue;
test=red)



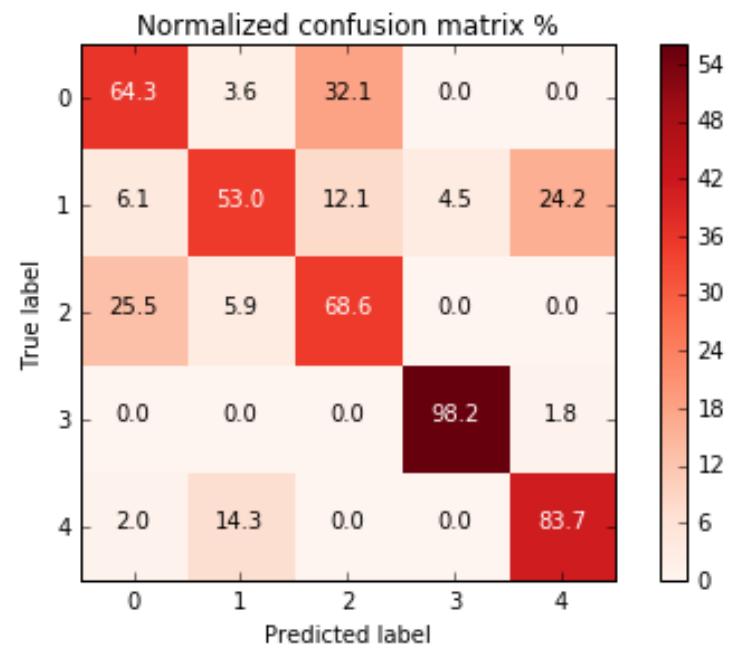
Epoch no.

Font Recognition Results

Feed Forward

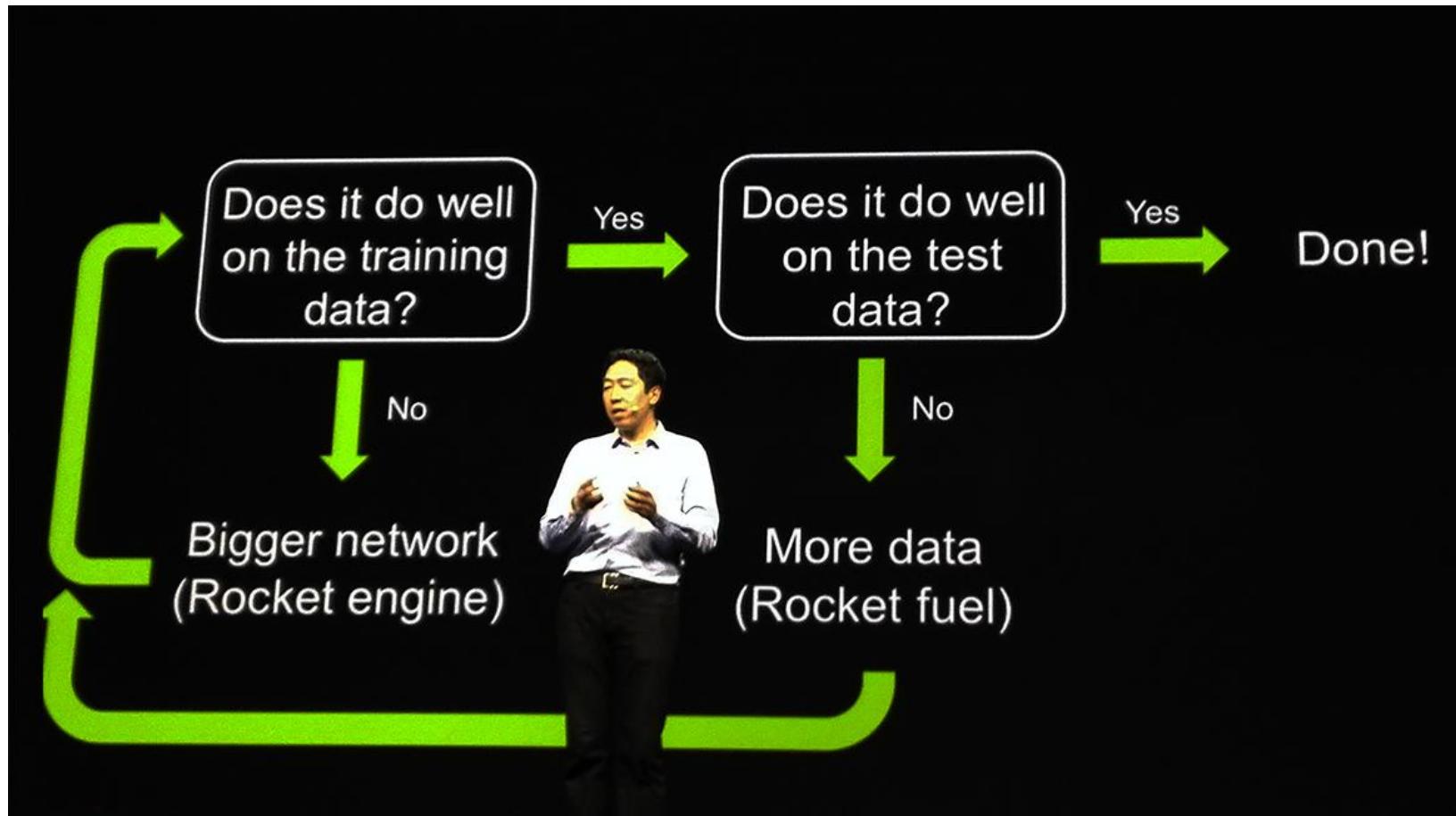


CNN



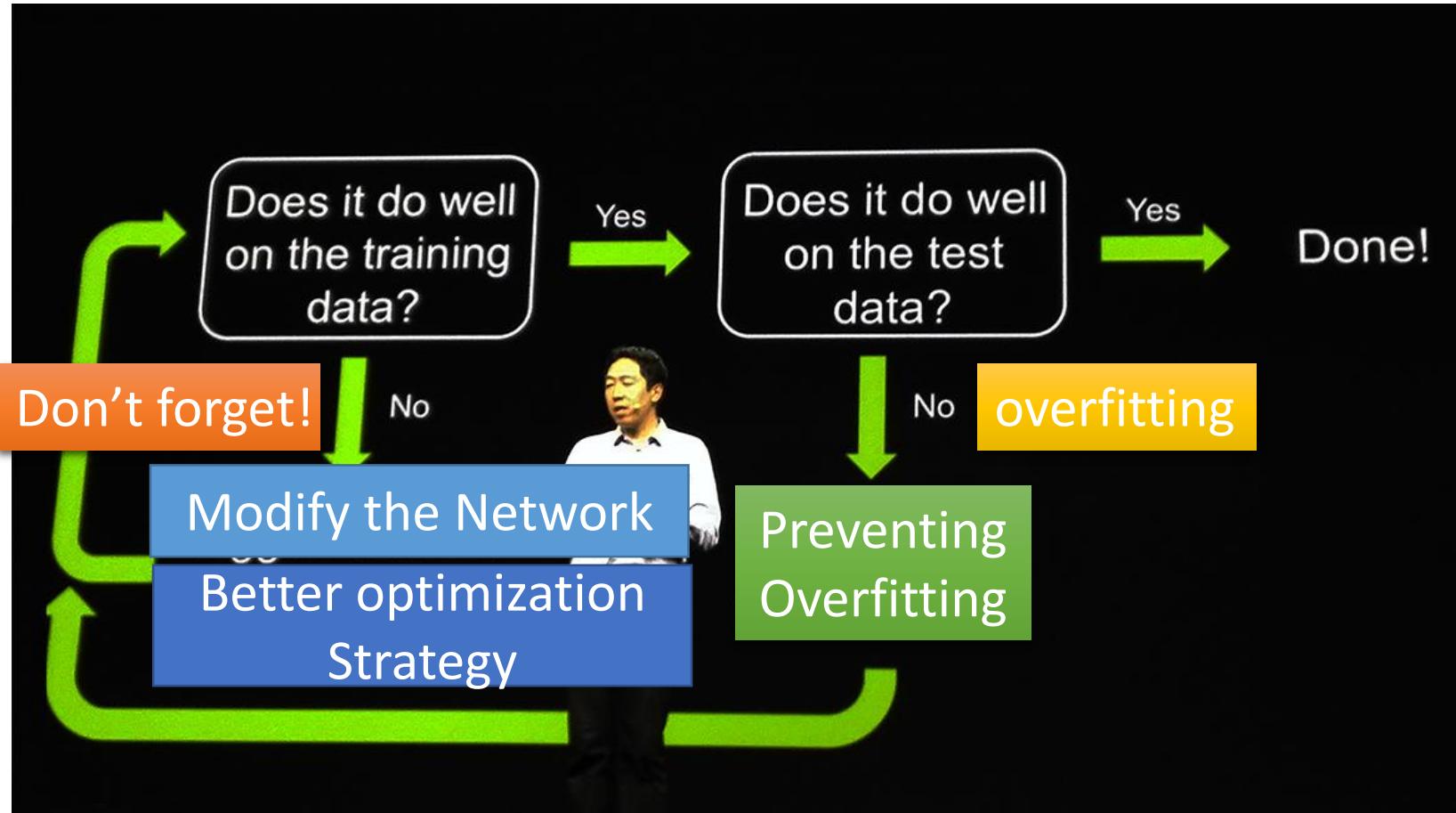
Tips for Training DNN

Recipe for Learning



<http://www.gizmodo.com.au/2015/04/the-basic-recipe-for-machine-learning-explained-in-a-single-powerpoint-slide/>

Recipe for Learning



<http://www.gizmodo.com.au/2015/04/the-basic-recipe-for-machine-learning-explained-in-a-single-powerpoint-slide/>

Tips for Training DNN

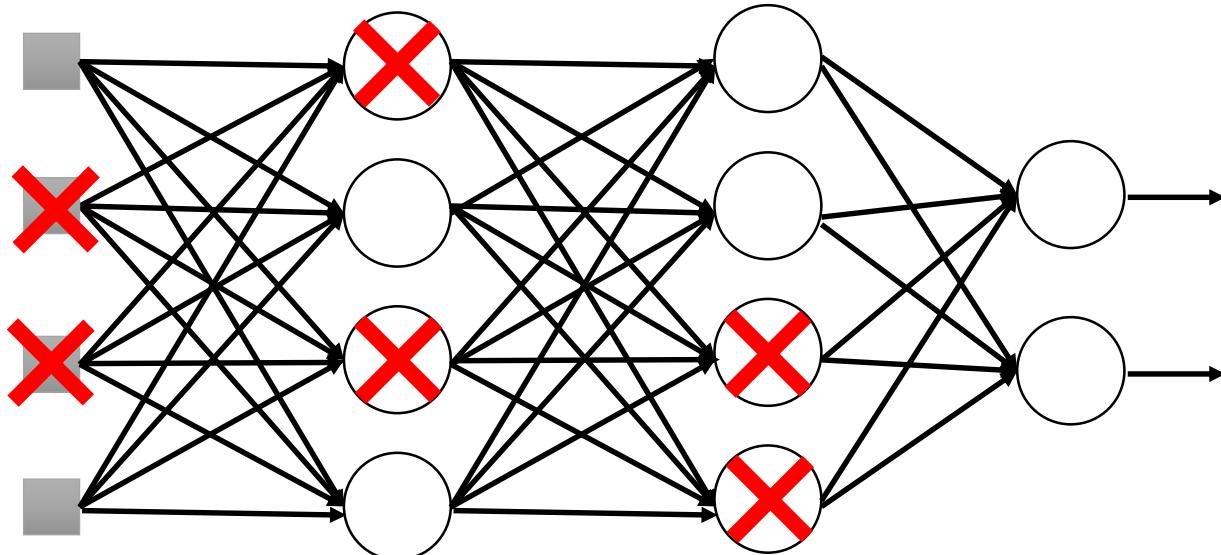
Dropout

Dropout

Pick a mini-batch

$$\theta^t \leftarrow \theta^{t-1} - \eta \nabla C(\theta^{t-1})$$

Training:



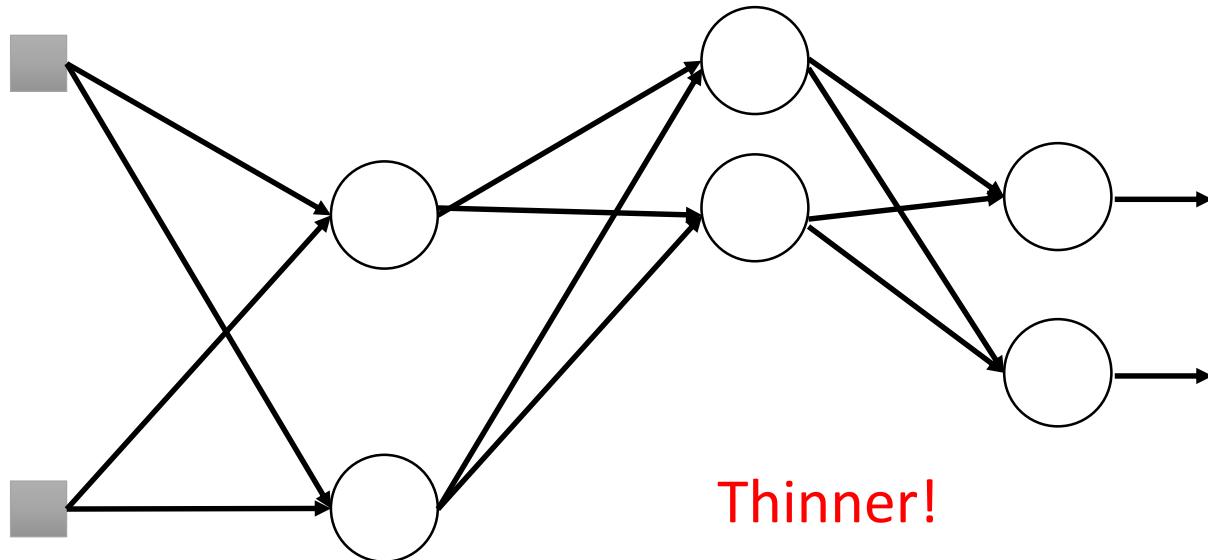
- **Each time before computing the gradients**
 - Each neuron has p% to dropout

Dropout

Pick a mini-batch

$$\theta^t \leftarrow \theta^{t-1} - \eta \nabla C(\theta^{t-1})$$

Training:

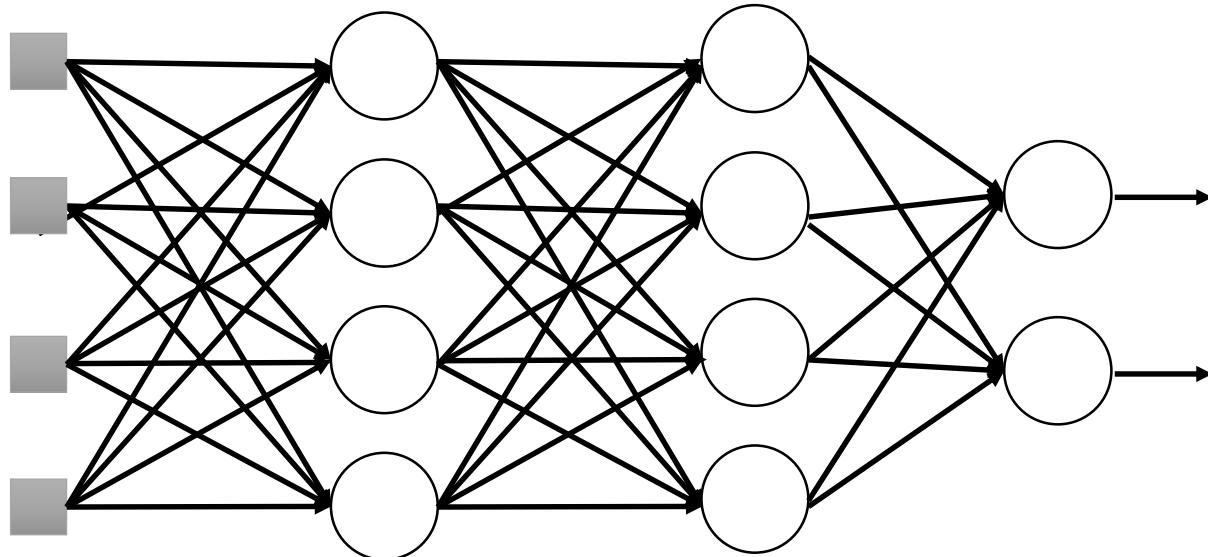


- **Each time before computing the gradients**
 - Each neuron has p% to dropout
 - ➡ **The structure of the network is changed.**
 - Using the new network for training

For each mini-batch, we resample the dropout neurons

Dropout

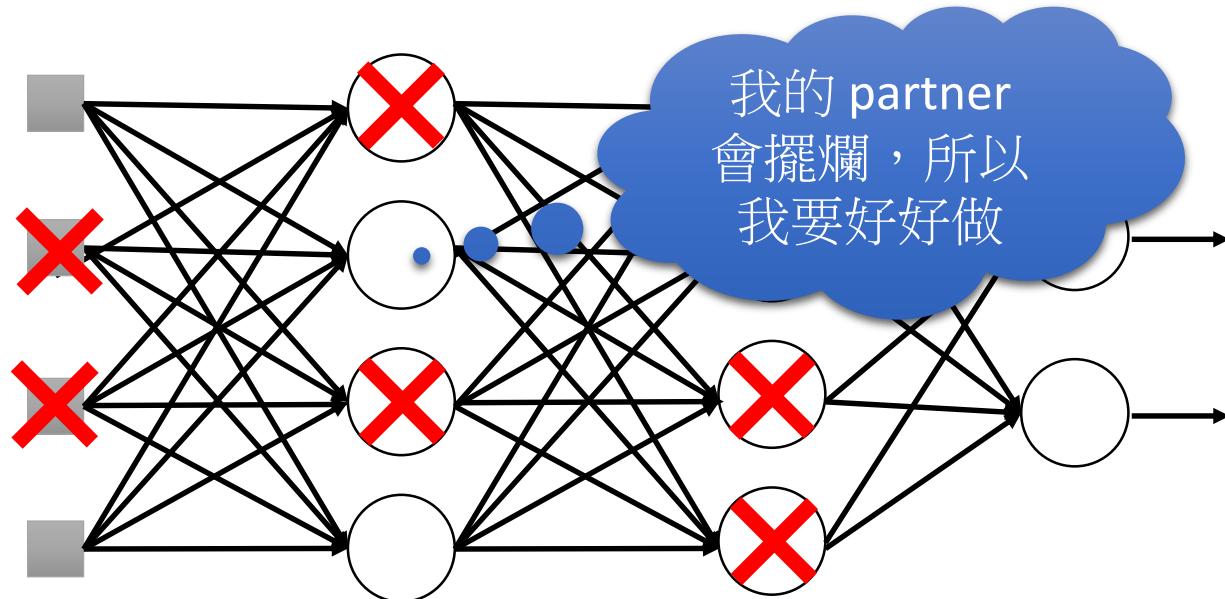
Testing:



➤ No dropout

- If the dropout rate at training is $p\%$,
all the weights times $(1-p)\%$
- Assume that the dropout rate is 50%.
If a weight $w = 1$ by training, set $w = 0.5$ for testing.

Dropout - Intuitive Reason



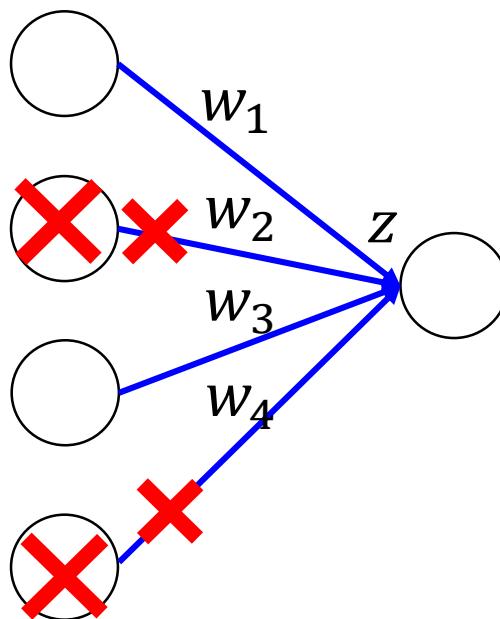
- When teams up, if everyone expect the partner will do the work, nothing will be done finally.
- However, if you know your partner will dropout, you will do better.
- When testing, no one dropout actually, so obtaining good results eventually.

Dropout - Intuitive Reason

- Why the weights should multiply $(1-p)\%$ (dropout rate) when testing?

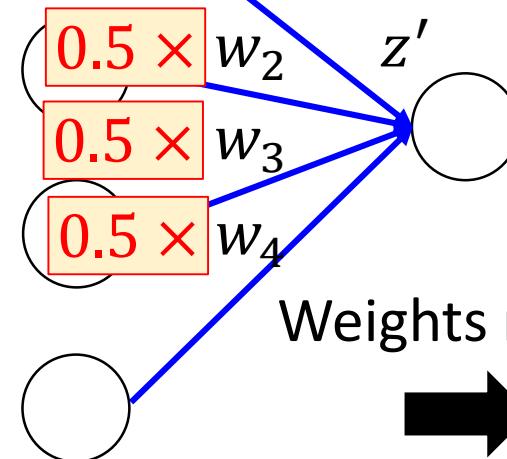
Training of Dropout

Assume dropout rate is 50%



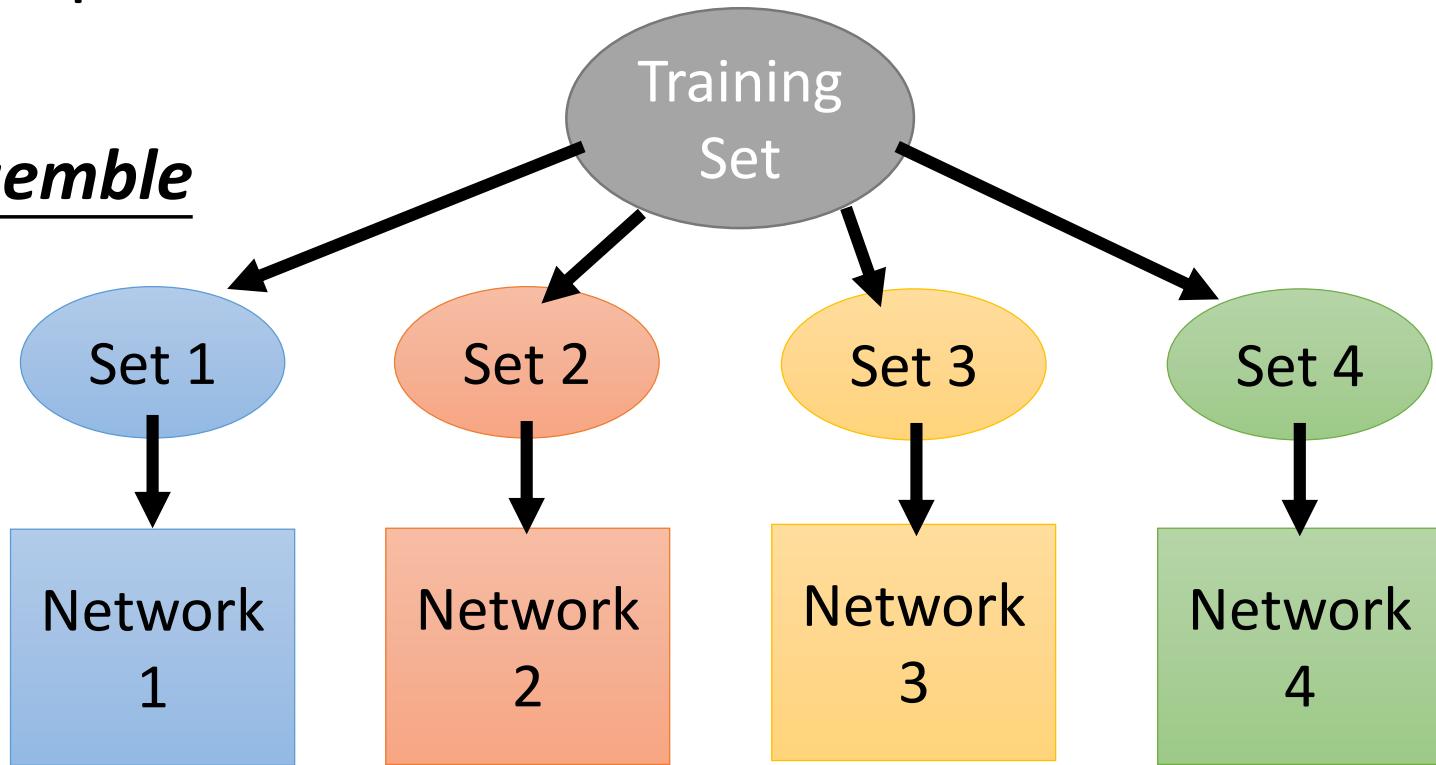
Testing of Dropout

No dropout



Dropout is a kind of ensemble.

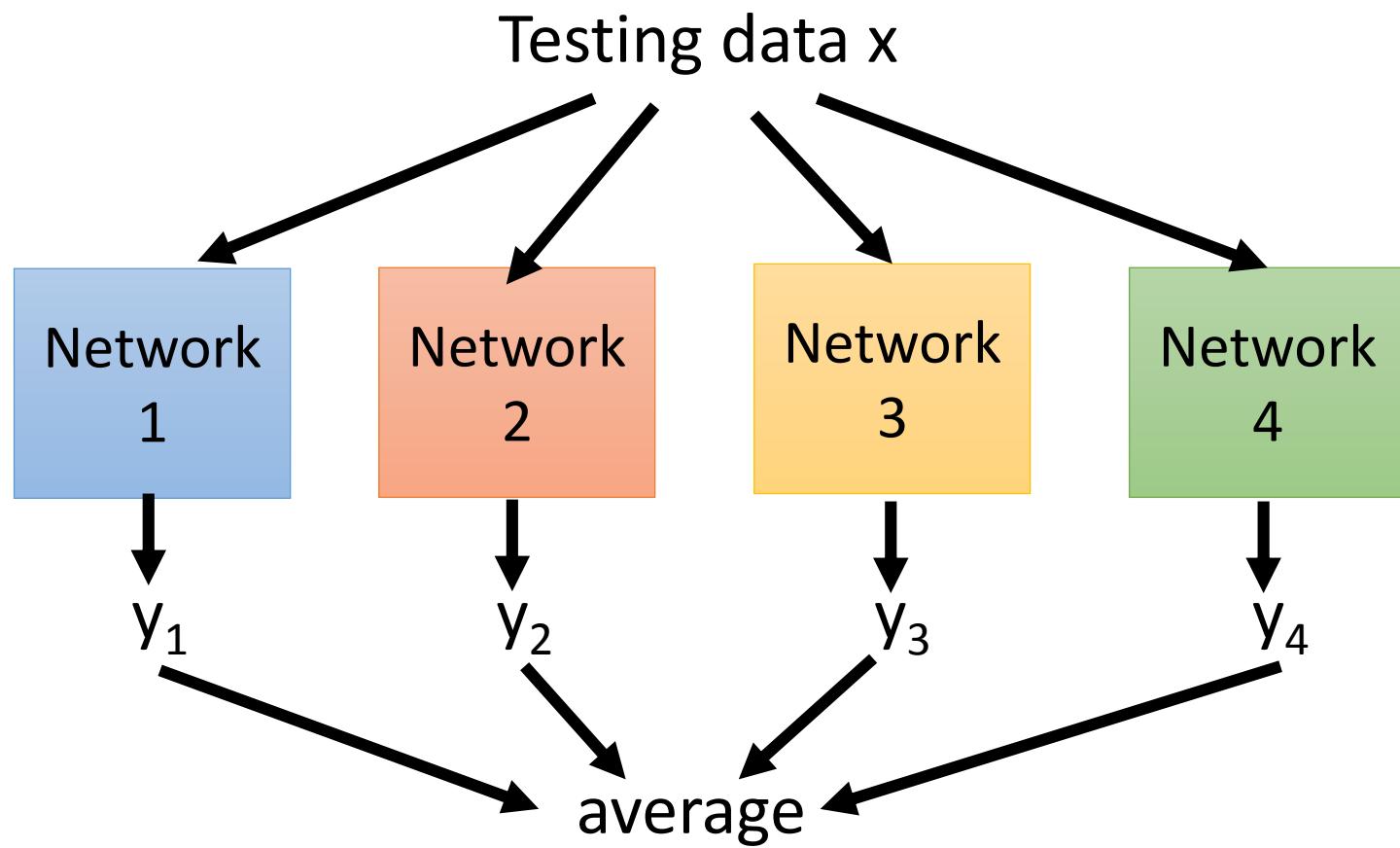
Ensemble



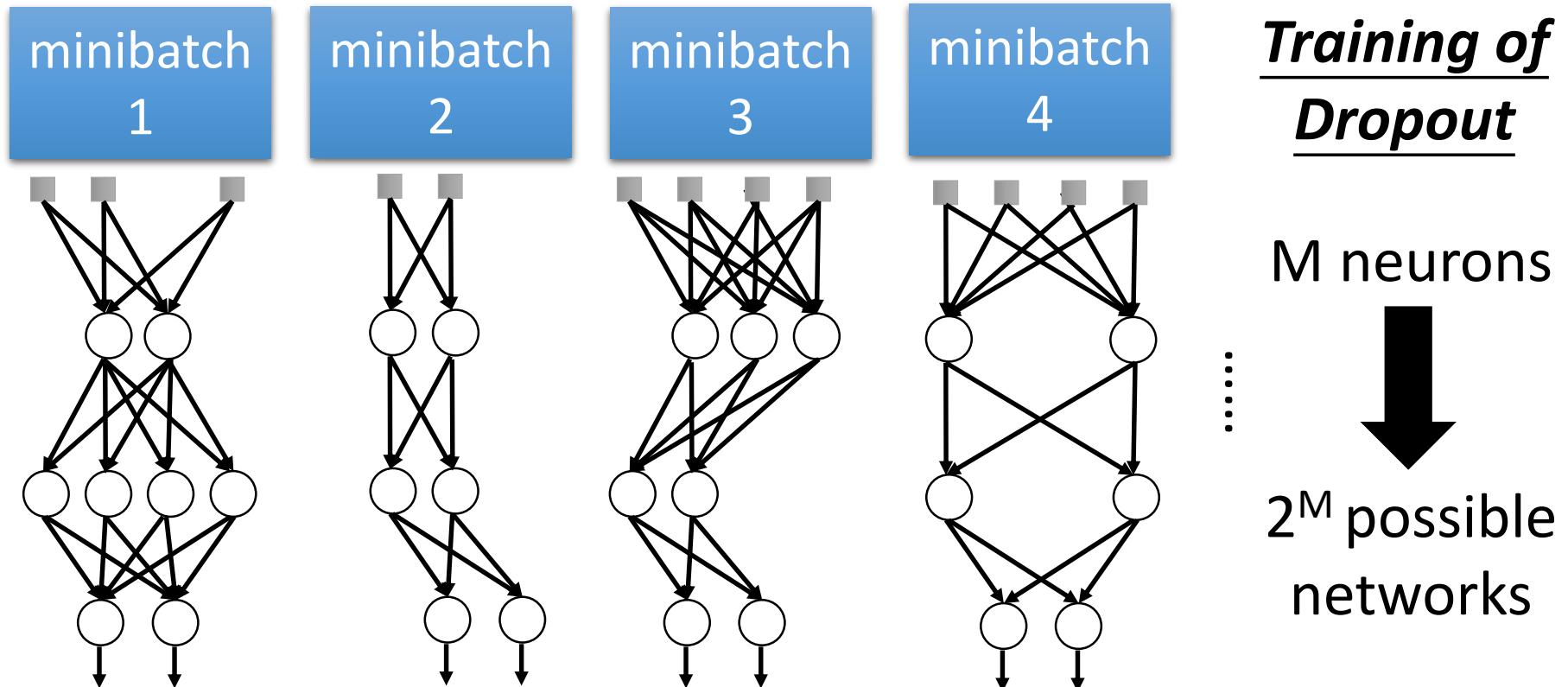
Train a bunch of networks with different structures

Dropout is a kind of ensemble.

Ensemble



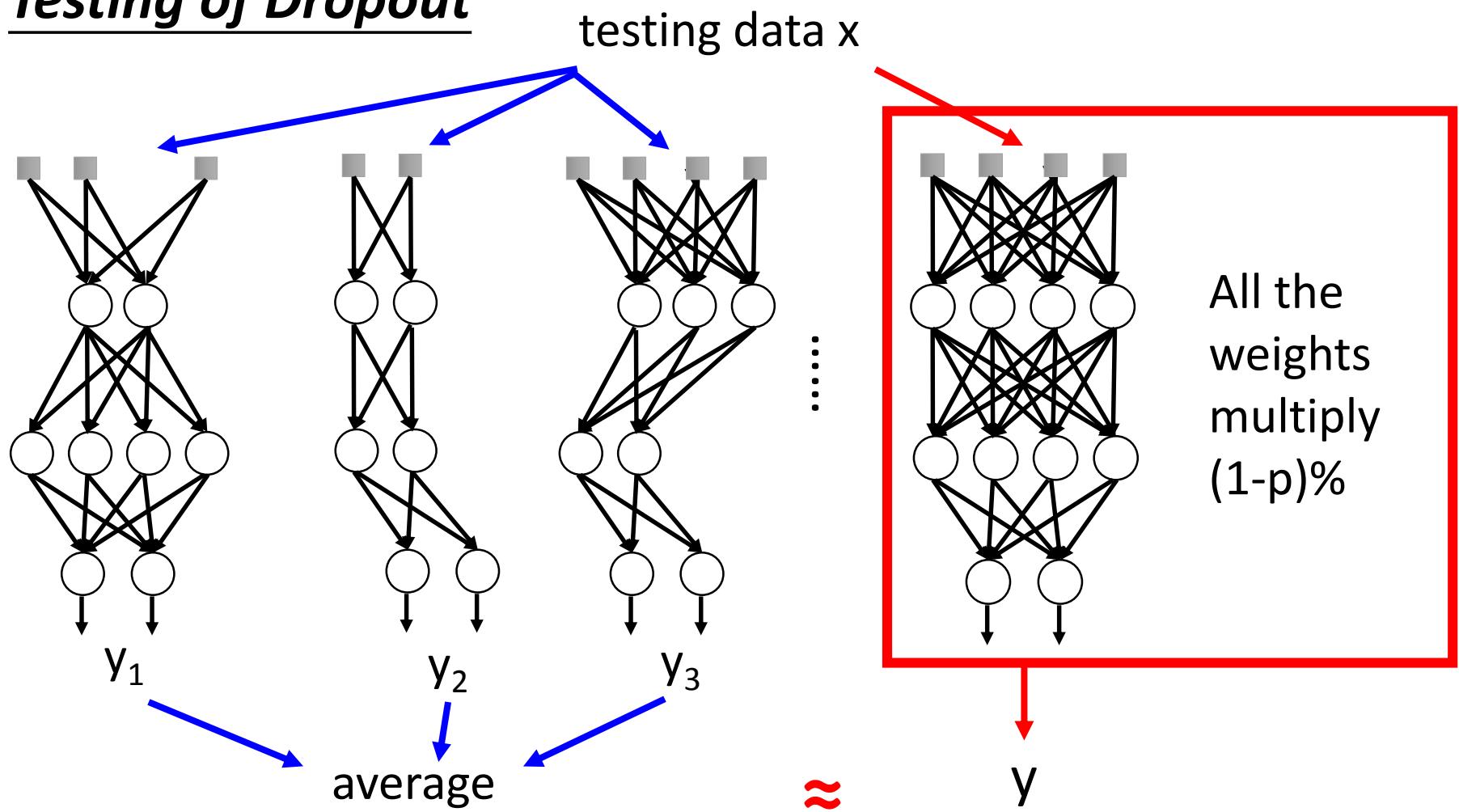
Dropout is a kind of ensemble.



- Using one mini-batch to train one network
- Some parameters in the network are shared

Dropout is a kind of ensemble.

Testing of Dropout



More about dropout

- More reference for dropout [Nitish Srivastava, JMLR'14] [Pierre Baldi, NIPS'13][Geoffrey E. Hinton, arXiv'12]
- Dropout works better with Maxout [Ian J. Goodfellow, ICML'13]
- Dropconnect [Li Wan, ICML'13]
 - Dropout delete neurons
 - Dropconnect deletes the connection between neurons
- Annealed dropout [S.J. Rennie, SLT'14]
 - Dropout rate decreases by epochs
- Standout [J. Ba, NISP'13]
 - Each neural has different dropout rate

Some on-line courses

Practical:

- **UDACITY**: Deep Learning by Google (DNN + TF)
- Deep Learning with TensorFlow : **Packt Video** (**our examples today from this!**)
- TensorFlow and Deep Learning without a PhD, Part 1 (Google Cloud Next '17)

<https://www.youtube.com/watch?v=u4alGiomYP4>

Theory:

- **Coursera** Neural Networks for Machine Learning, as taught by Geoffrey Hinton (University of Toronto)
- **Andrew Ng** : Deep Learning and Unsupervised Feature Learning