Deep Learning Seminar
Day-2

Master of Science in Signal Theory and Communications

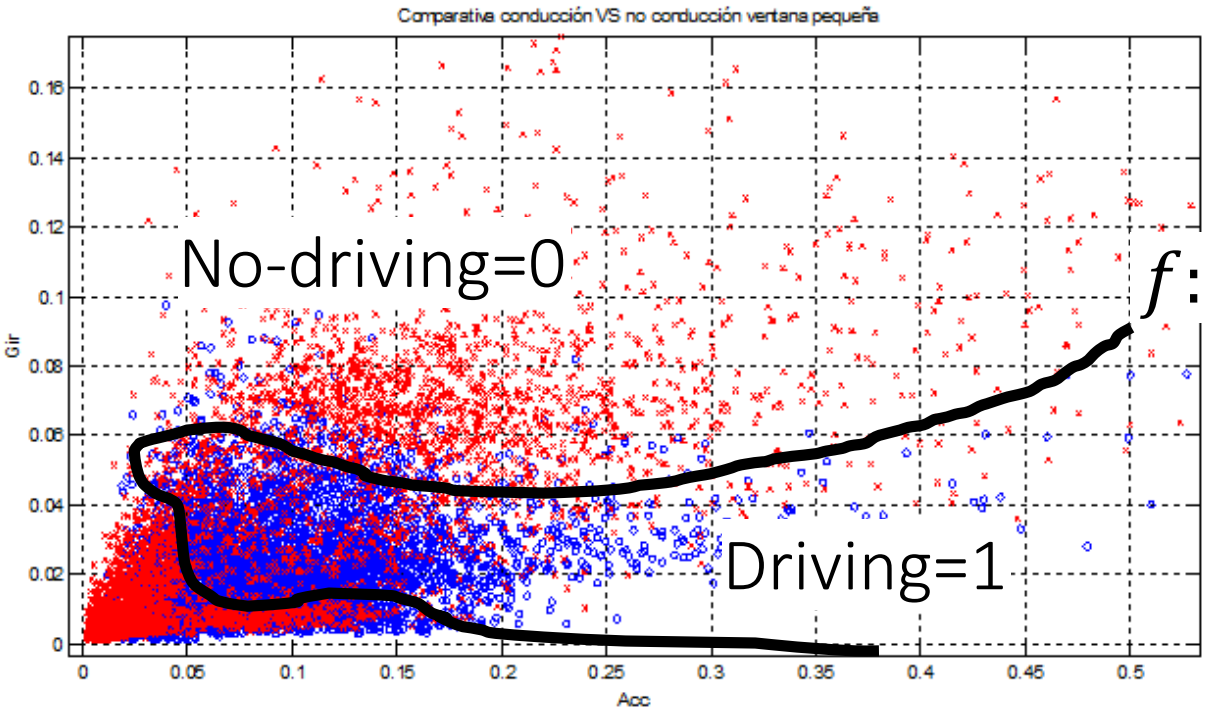
TRACK: Signal Processing and Machine Learning for Big Data

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?**

x_2



$f: R^K \rightarrow R$

x_1

From linear classifiers TO Neural Networks

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

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

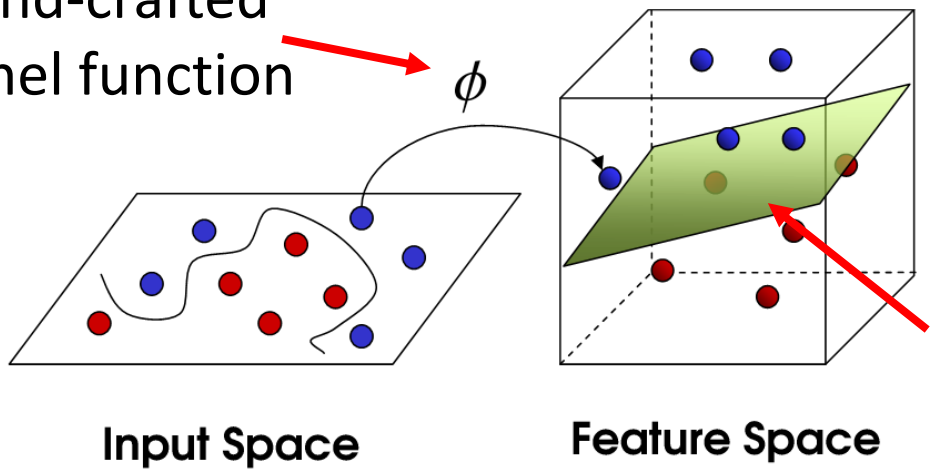
1. To manually engineer $\phi(.)$

2. Use a very generic $\phi(.)$ as kernel machines (e.g. SVM, RBF kernel)

3. The strategy of **deep learning** : to learn $\phi(.)$

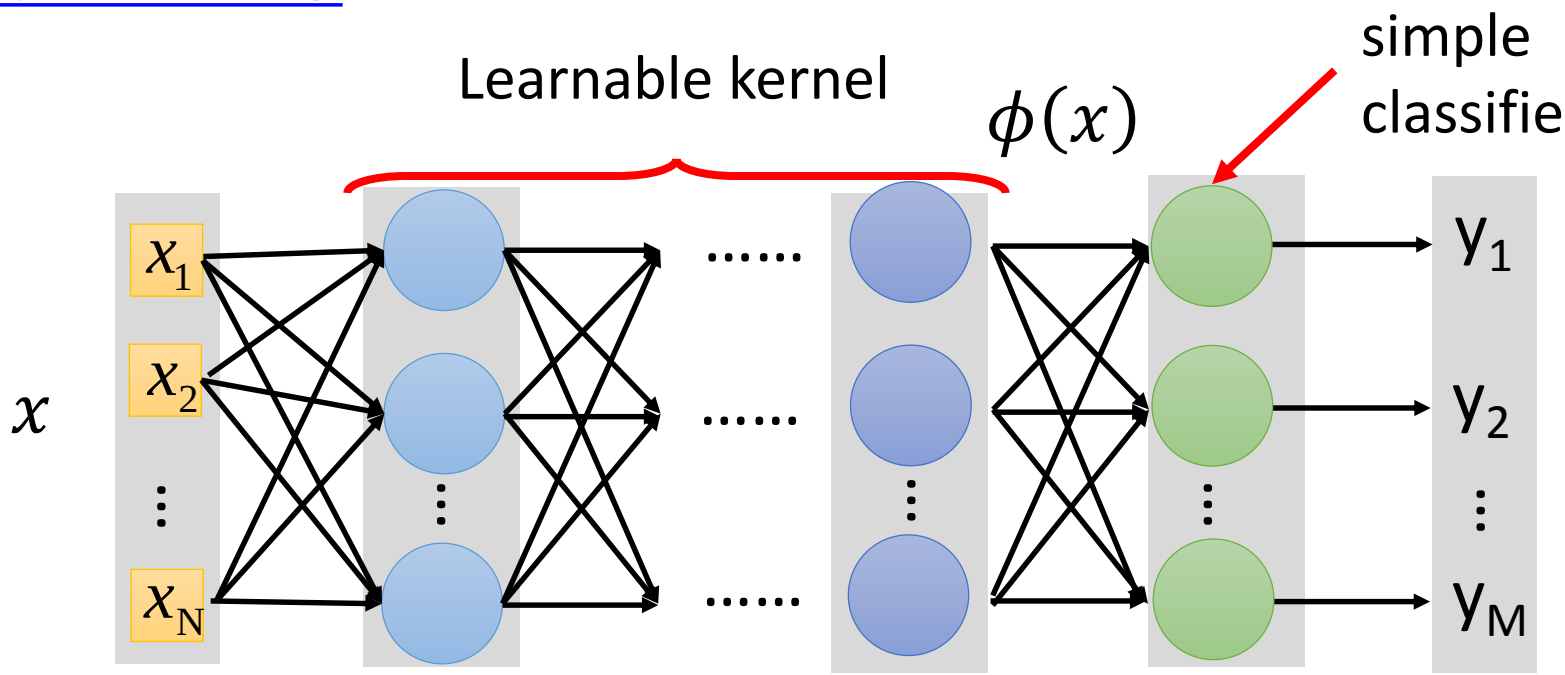
From: DL Tutorial
Hung-yi Lee

SVM



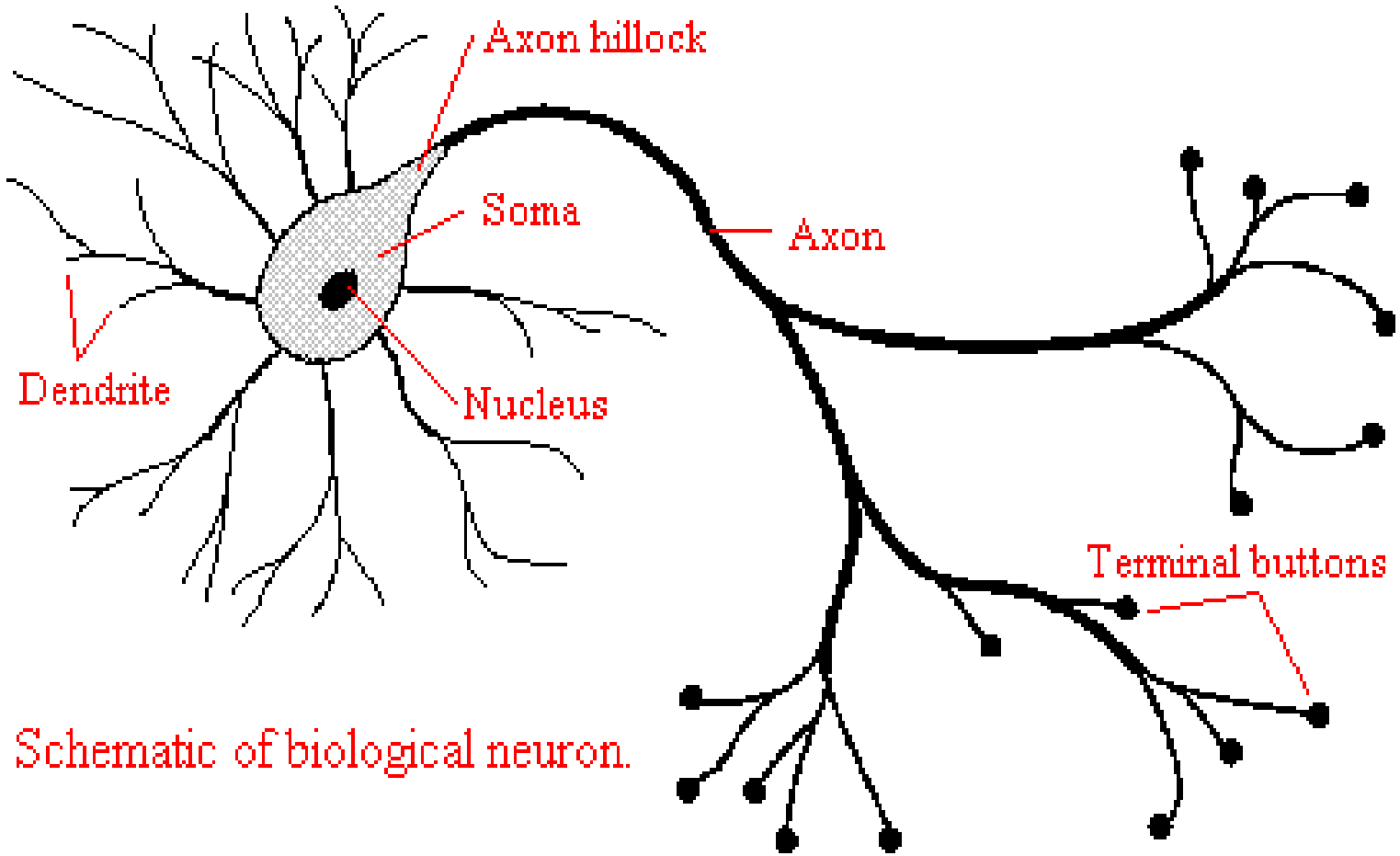
Source of image: http://www.gipsa-lab.grenoble-inp.fr/transfert/seminaire/455_Kadri2013Gipsa-lab.pdf

Deep Learning



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

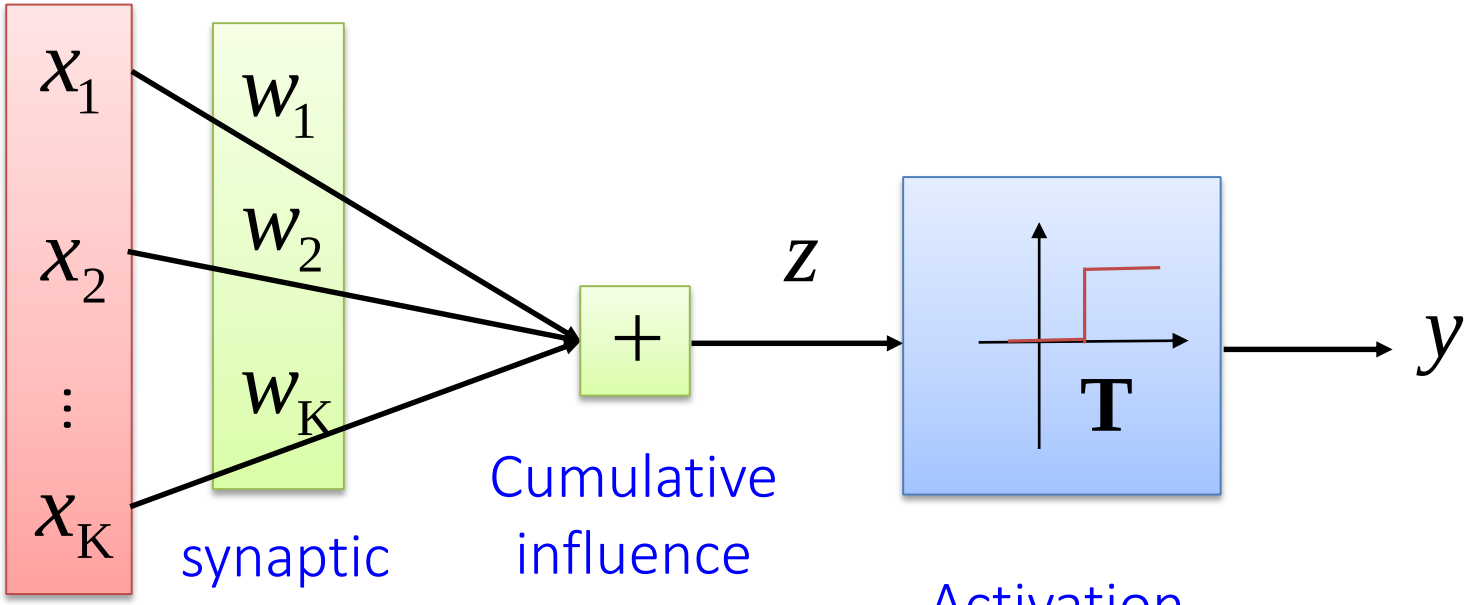
…from a broad class of functions



Schematic of biological neuron.

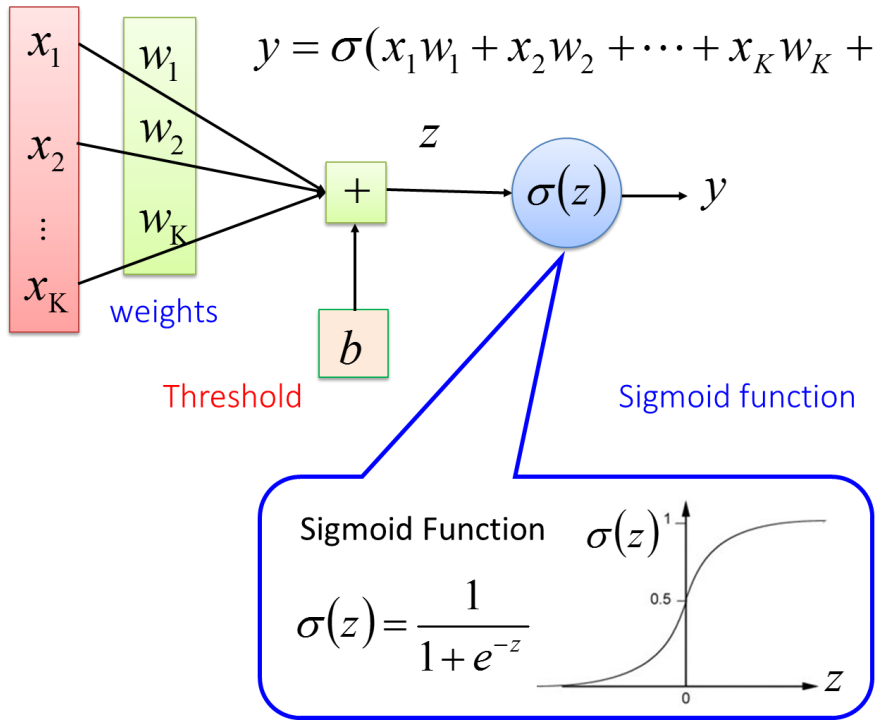
From linear classifiers TO Neural Networks

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!)

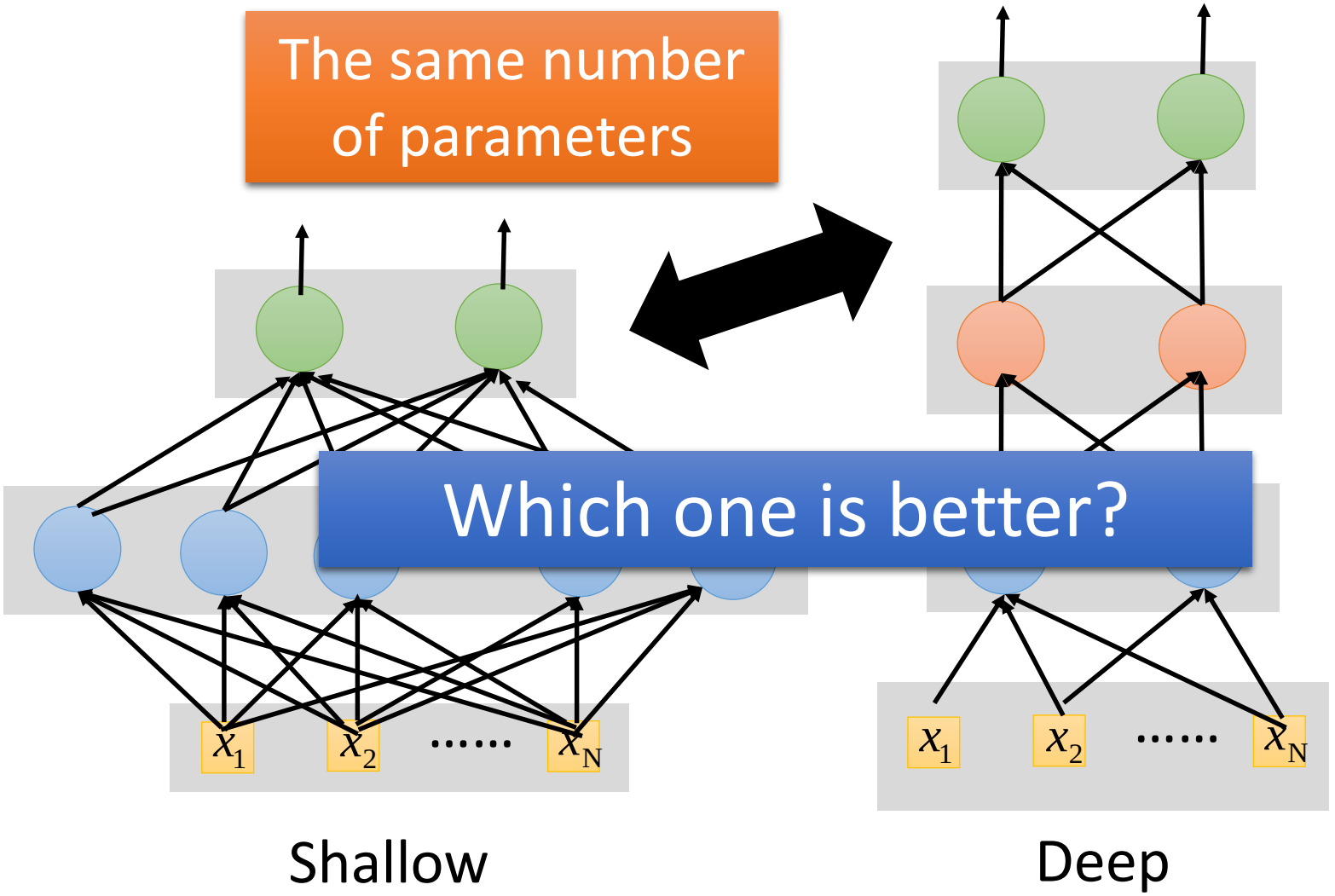
From linear classifiers TO Neural Networks

- 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



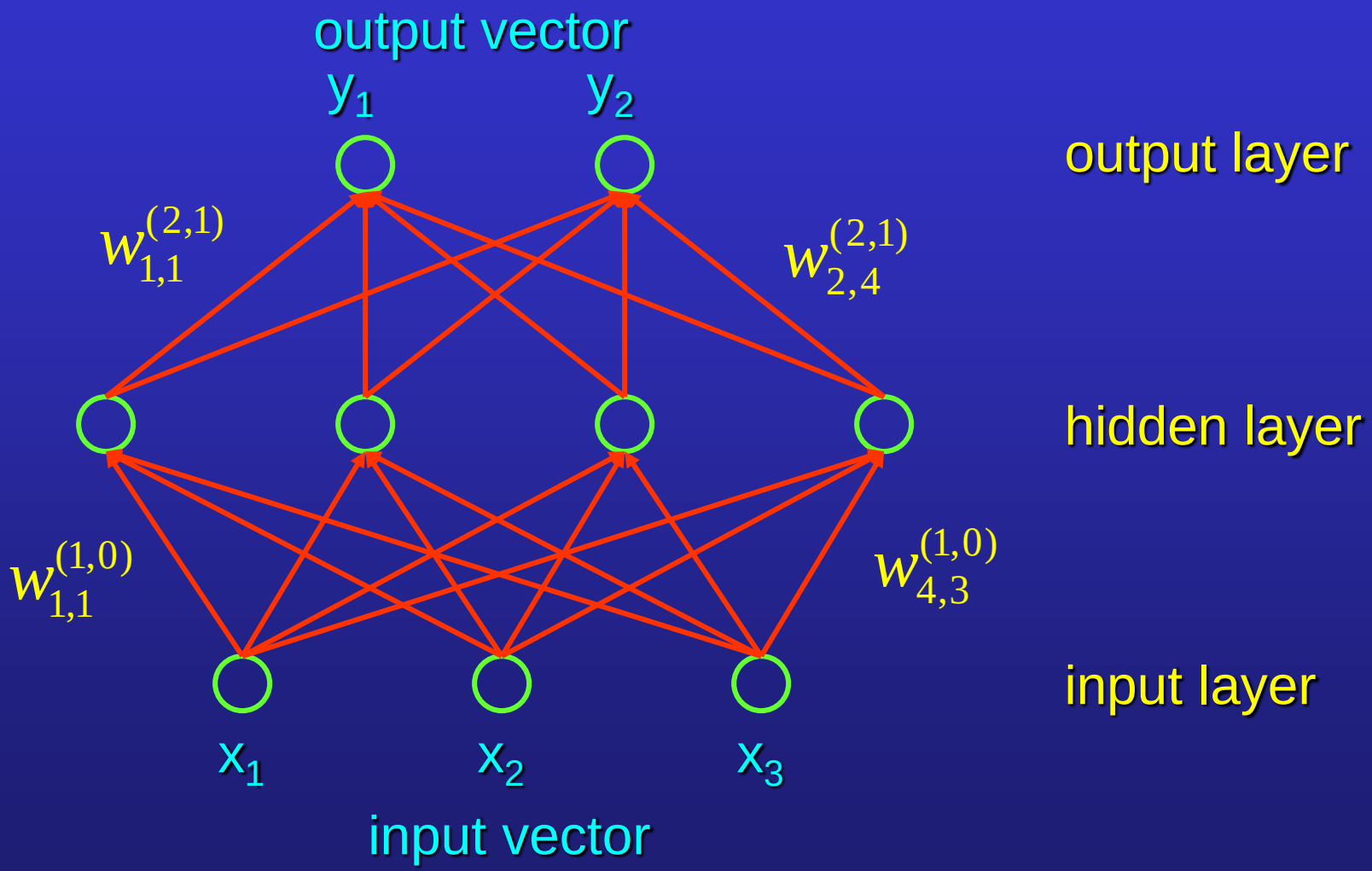
Training Neural Networks

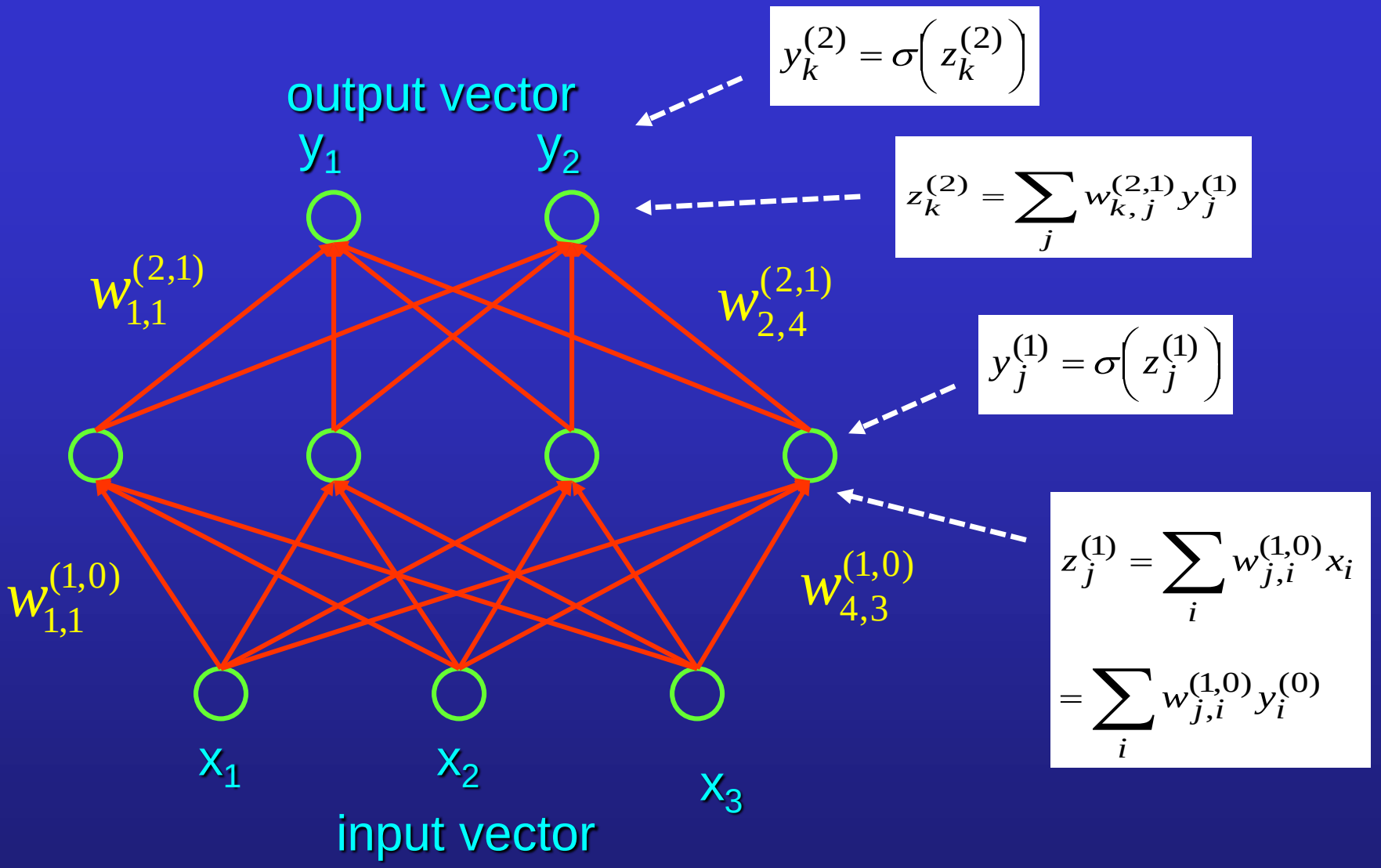
- **Backpropagation** Learning

- Gradients estimation NOT an optimization method

Terminology

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



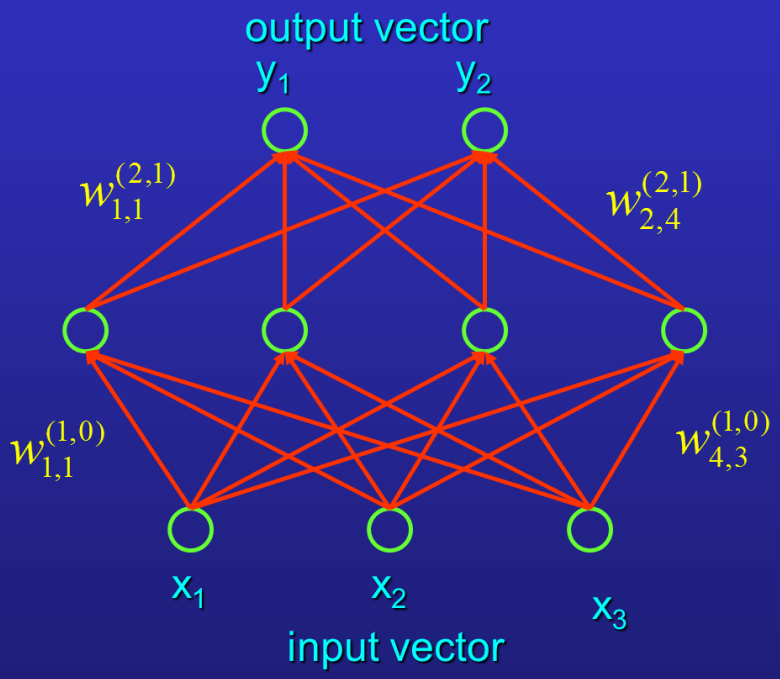

output vector
y_1
y_2
$w_{1,1}^{(2,1)}$
$w_{2,4}^{(2,1)}$
$w_{1,1}^{(1,0)}$
$w_{4,3}^{(1,0)}$
x_1
x_2
x_3
input vector
$y_k^{(2)} = \sigma\left(z_k^{(2)} \right)$
$z_k^{(2)} = \sum_j w_{k,j}^{(2,1)} y_j^{(1)}$
$y_j^{(1)} = \sigma\left(z_j^{(1)} \right)$
$z_j^{(1)} = \sum_i w_{j,i}^{(1,0)} x_i$
$= \sum_i w_{j,i}^{(1,0)} y_i^{(0)}$

Backpropagation

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

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

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



For using gradient descent algorithms we need:

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

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

Backpropagation : apply the chain rule

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

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

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

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

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

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

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

Hidden layer : NOTICE that each $w_{j,i}^{(1)}$ influences each $y_k^{(2)}$ with k=1,… K (2 in our example)

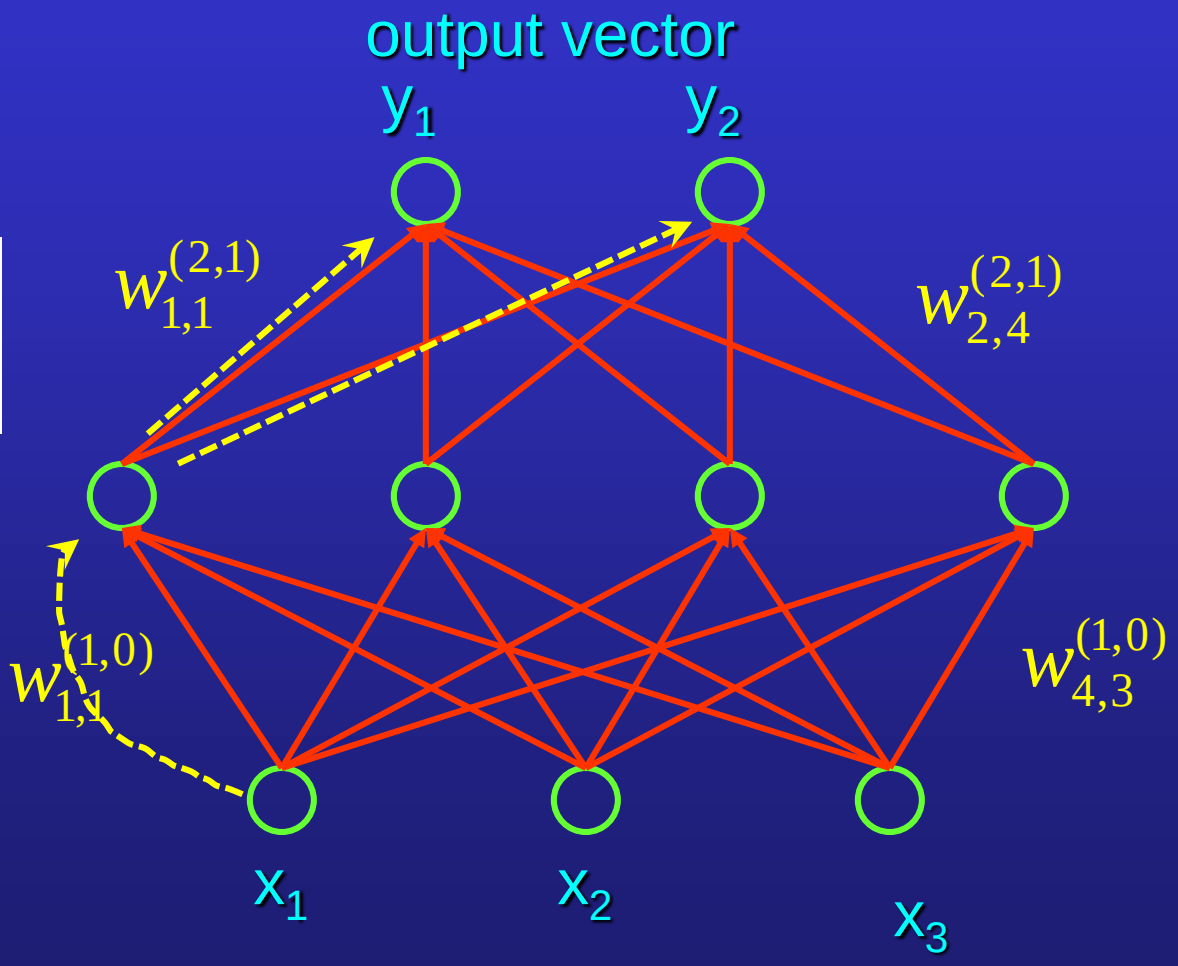
$$E(\theta) = \sum \left(y_k - y_k^{(2)} \right)^2$$

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

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

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

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



Hidden layer :

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

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

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

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

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

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

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

Weight changes :

Output layer

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

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

Hidden layer

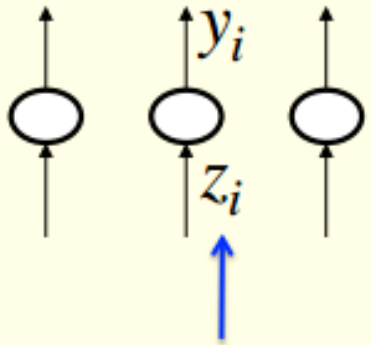
$$\frac{\partial E}{\partial w_{j,i}^{(1)}} = \sum_{k=1}^{K}\left[-2(y_k - y_k^{(2)})\sigma'\left(z_k^{(2)}\right)w_{k,j}^{(2)}\sigma'\left(z_j^{(1)}\right)y_i^{(0)}\right]$$

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

Softmax activation function

i=1…to K output classes

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

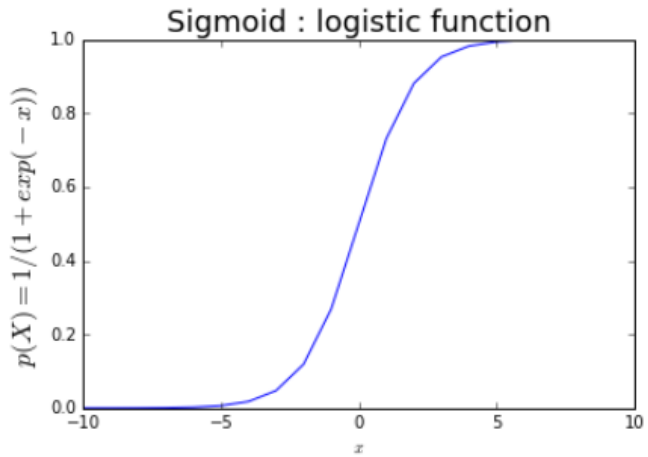


softmax group

this is called the "logit"

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

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

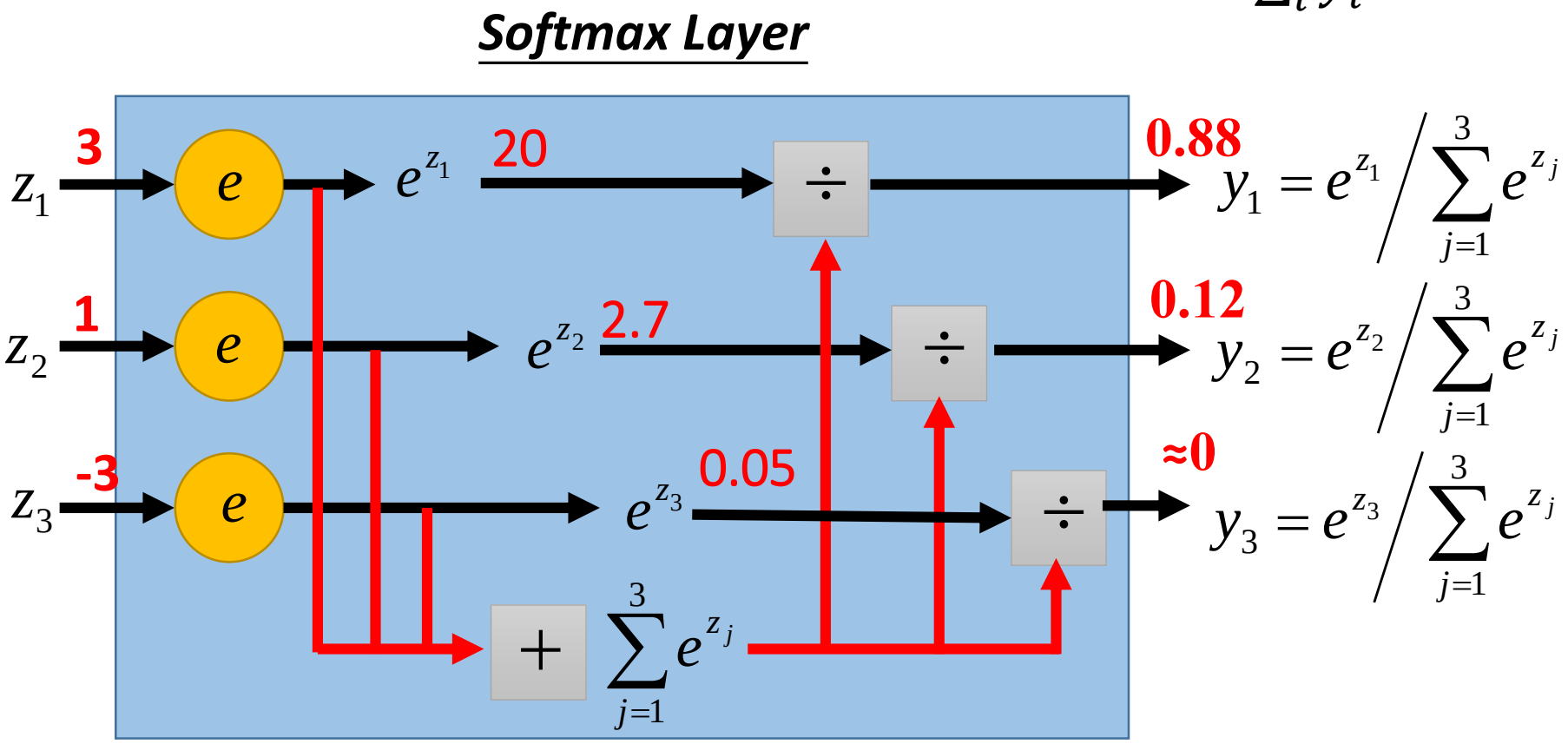


Softmax Layer

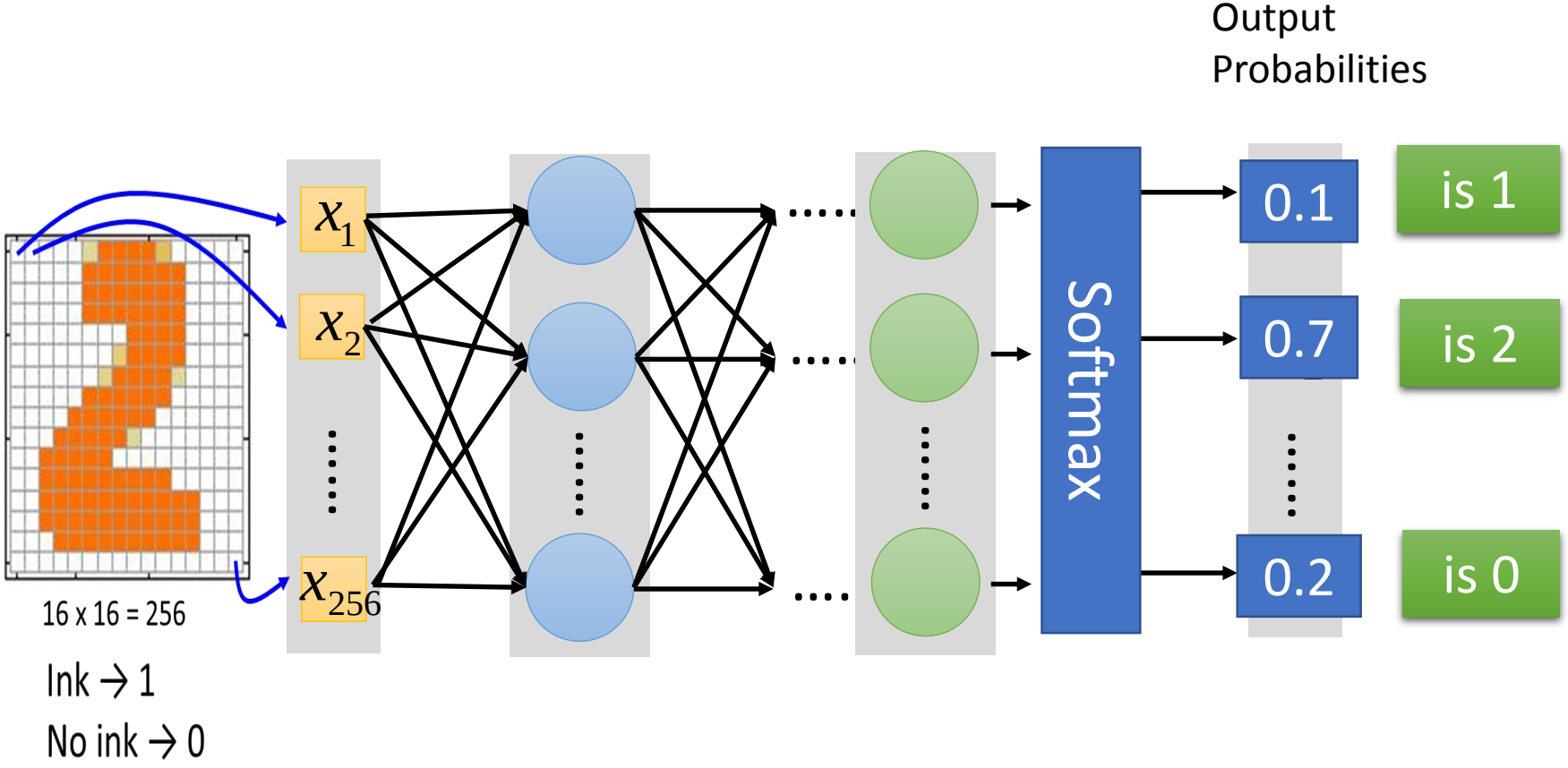
- Softmax layer as the output layer

Probability:

- $1 > y_i > 0$
- $\sum_i y_i = 1$



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

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

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

Hidden layer

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

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

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

Output layer

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

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

Hidden layer

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

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

- 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

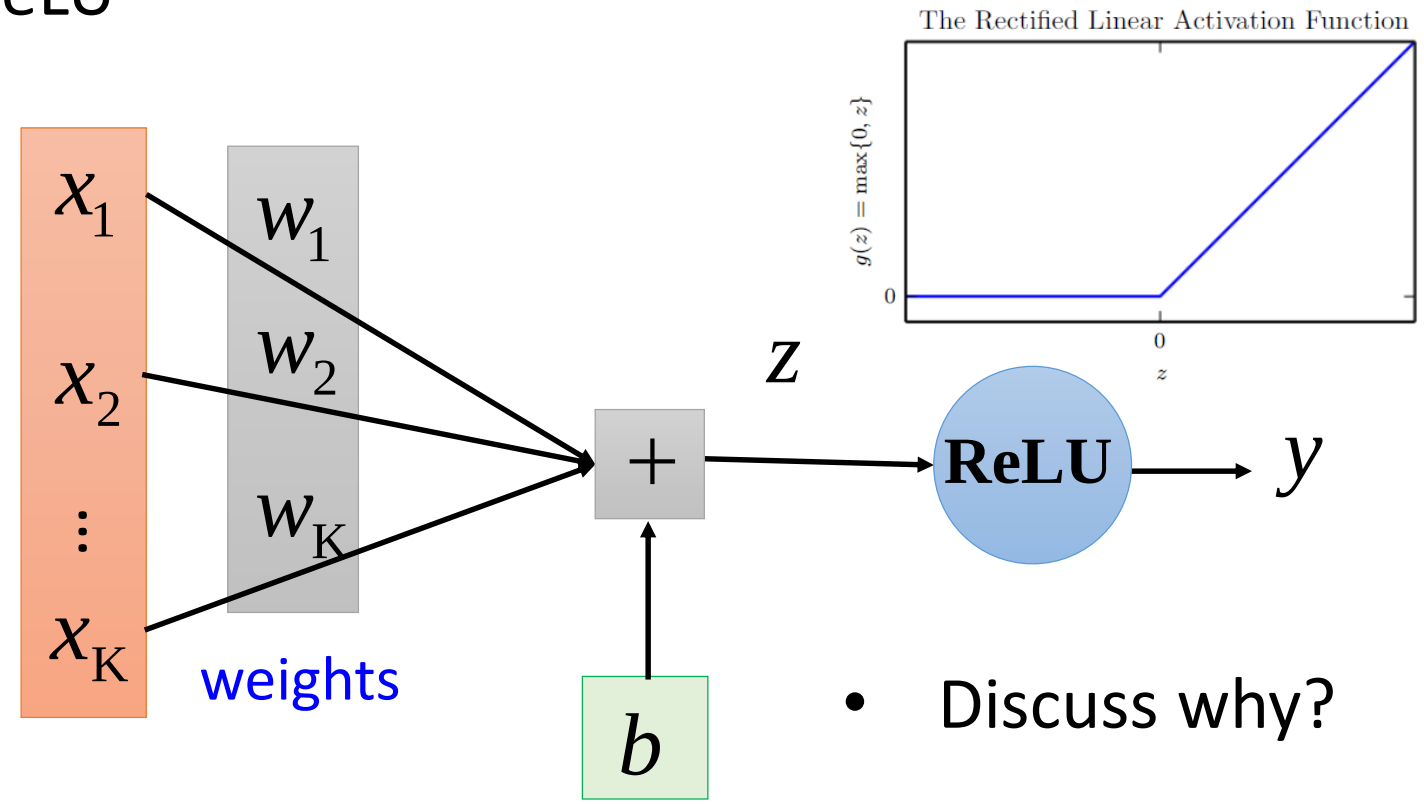
Deep Learning Tutorial

李宏毅

Hung-yi Lee

Activation Functions

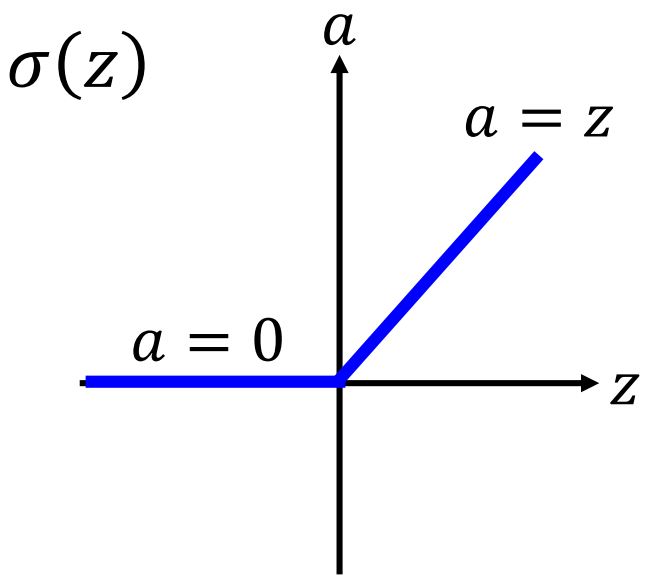
- In modern neural networks, the default recommendation is to use the rectified linear unit or ReLU



- Discuss why?

ReLU

- Rectified Linear Unit (ReLU)

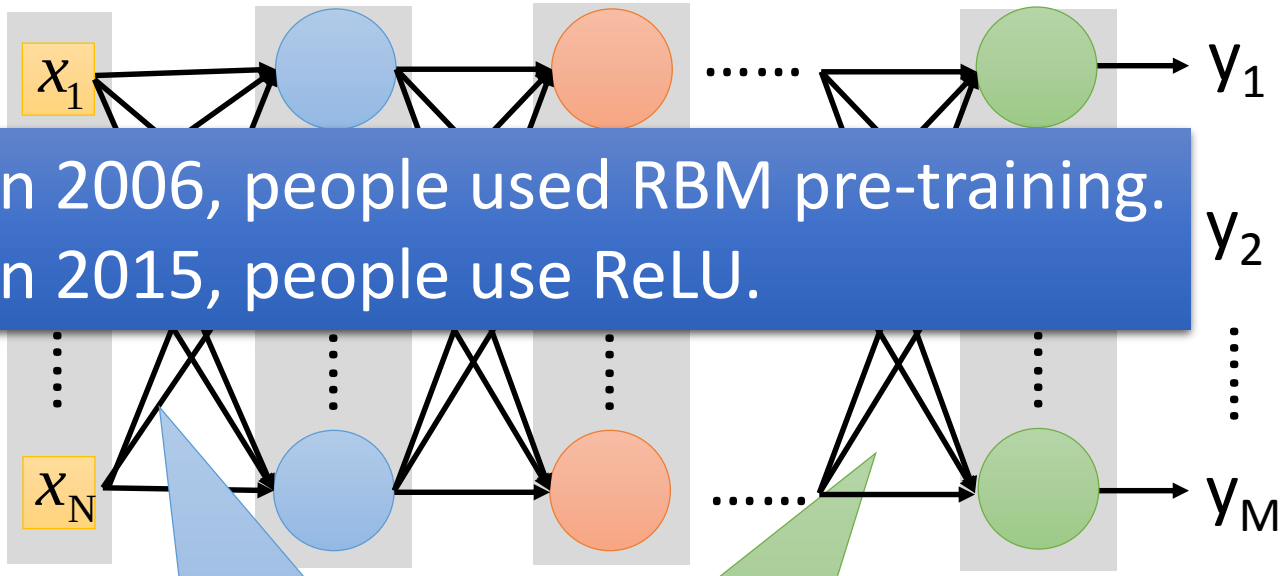


[Xavier Glorot, AISTATS'11]
[Andrew L. Maas, ICML'13]
[Kaiming He, arXiv'15]

Reason:

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

Vanishing Gradient Problem



In 2006, people used RBM pre-training.
In 2015, people use ReLU.

Smaller gradients

Learn very slow

Almost random

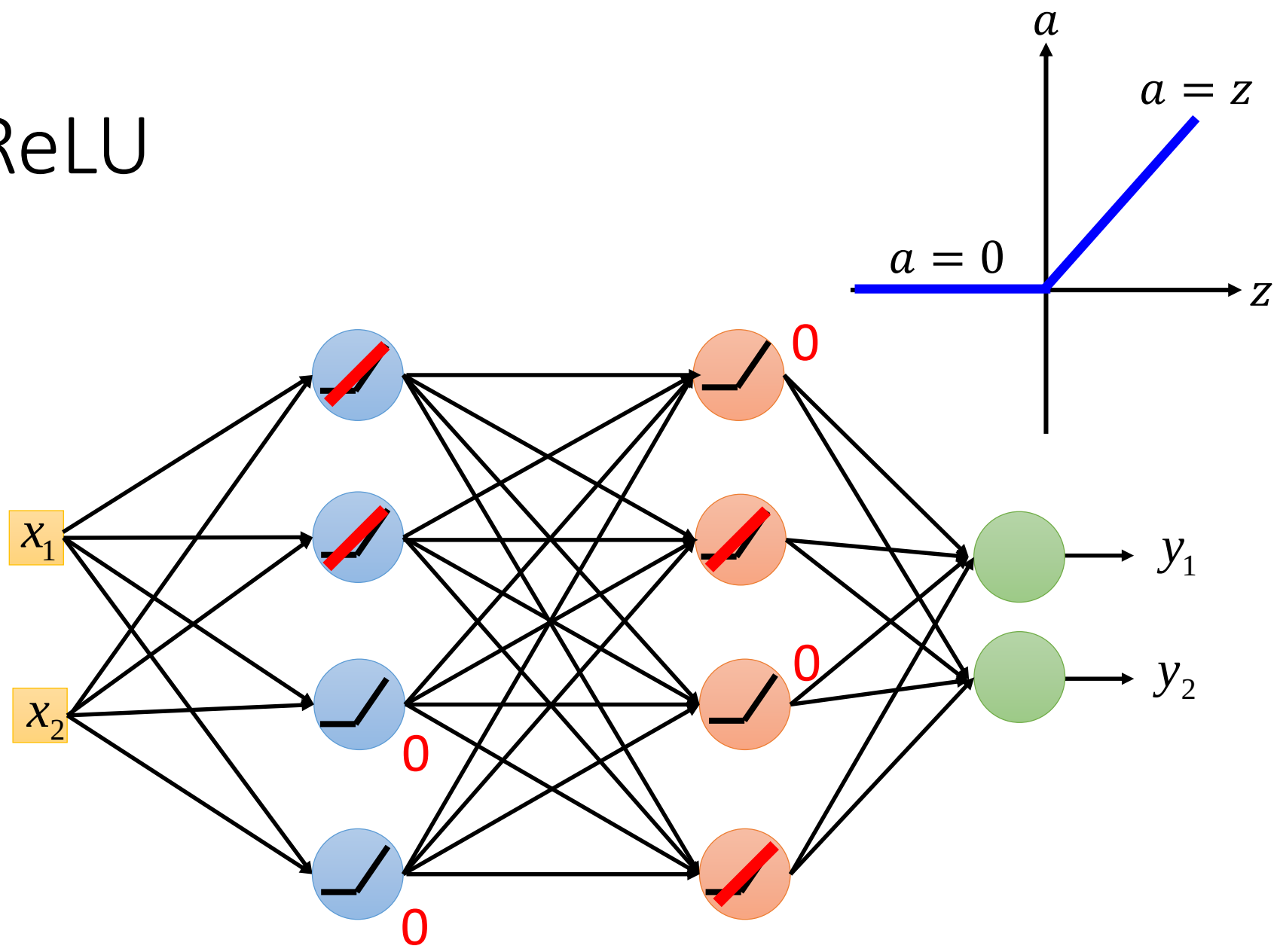
Larger gradients

Learn very fast

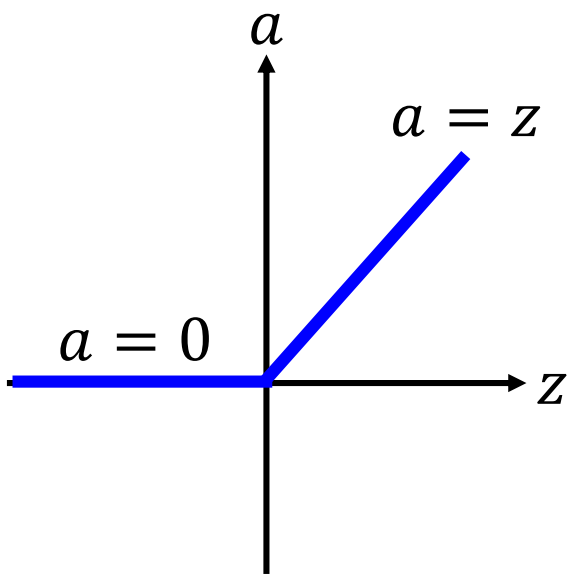
Already converge

based on random!?

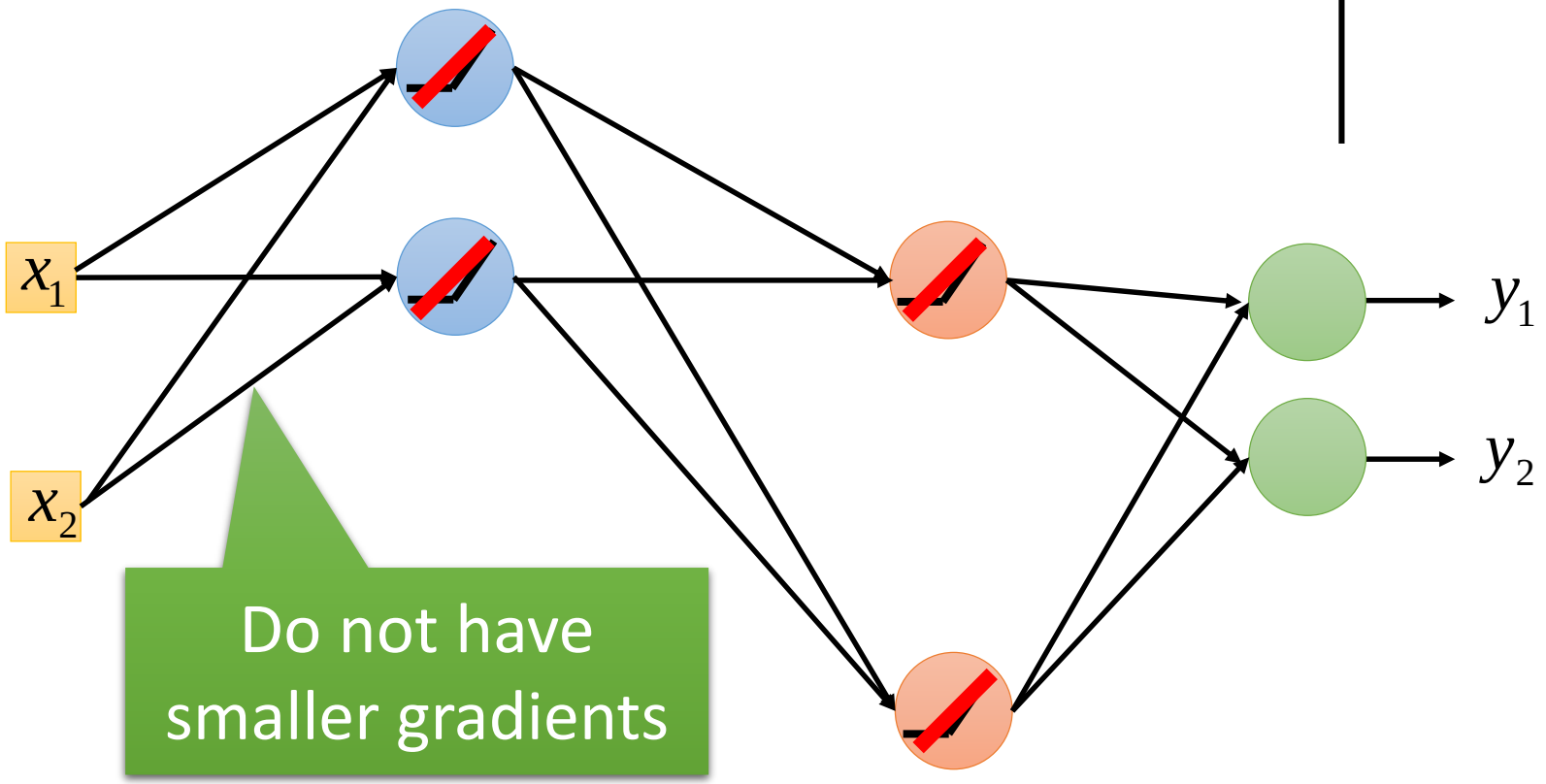
ReLU



ReLU



A Thinner linear network

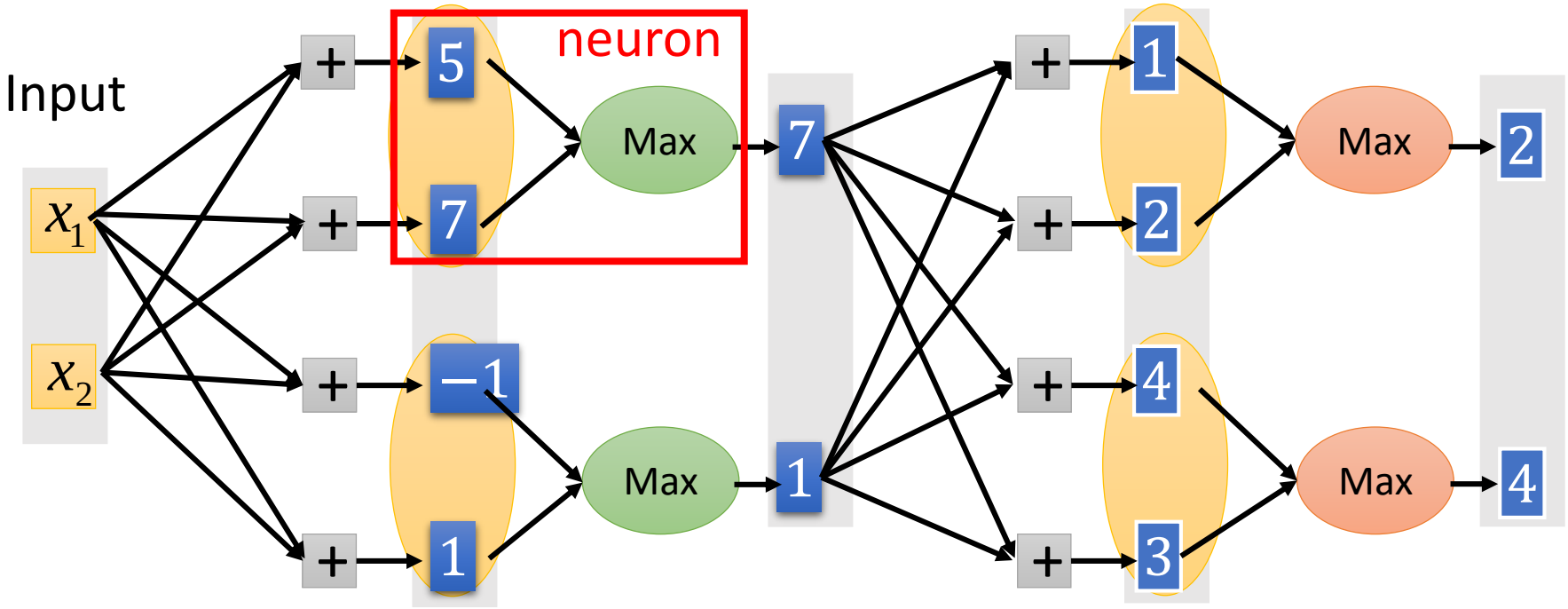


Do not have smaller gradients

Maxout

ReLU is a special cases of Maxout

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



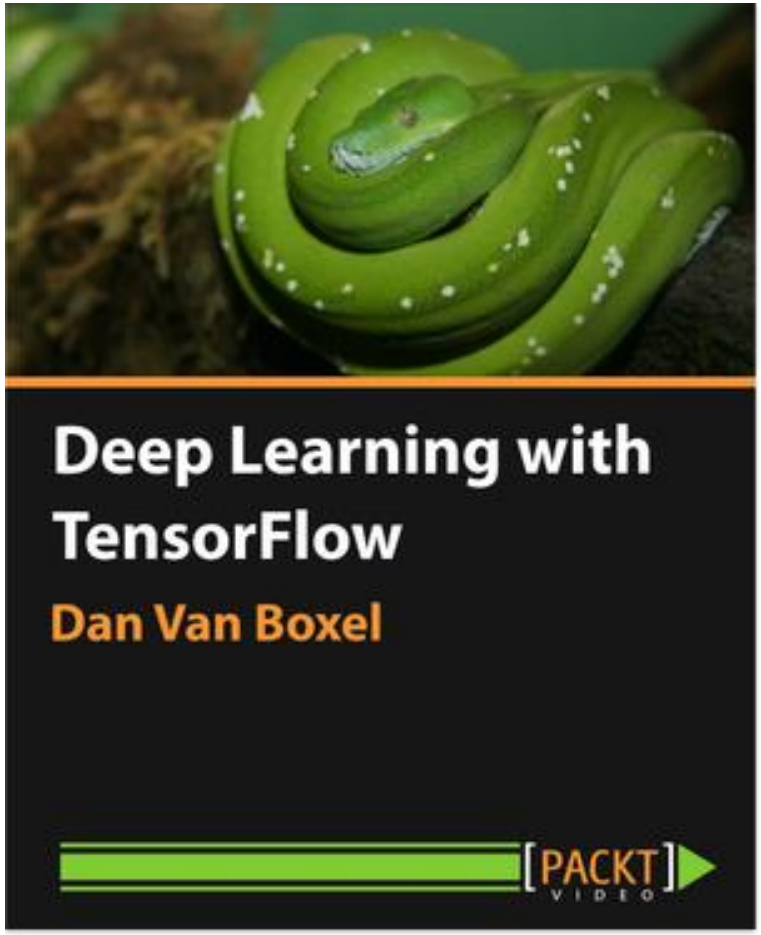
You can have more than 2 elements in a group.

Adaptive learning rate ……

- 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]

TensorFlow

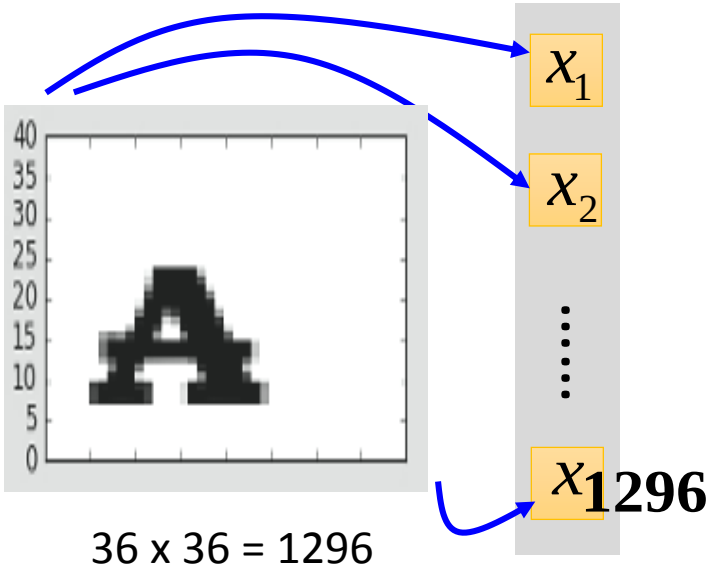
Font type Recognition

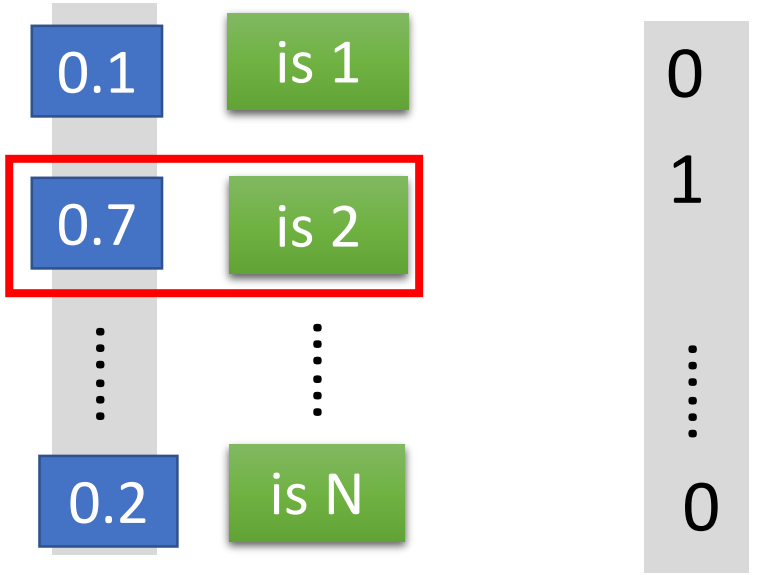


Font type Recognition

Input **reShape** **Output**

Font type is "2"



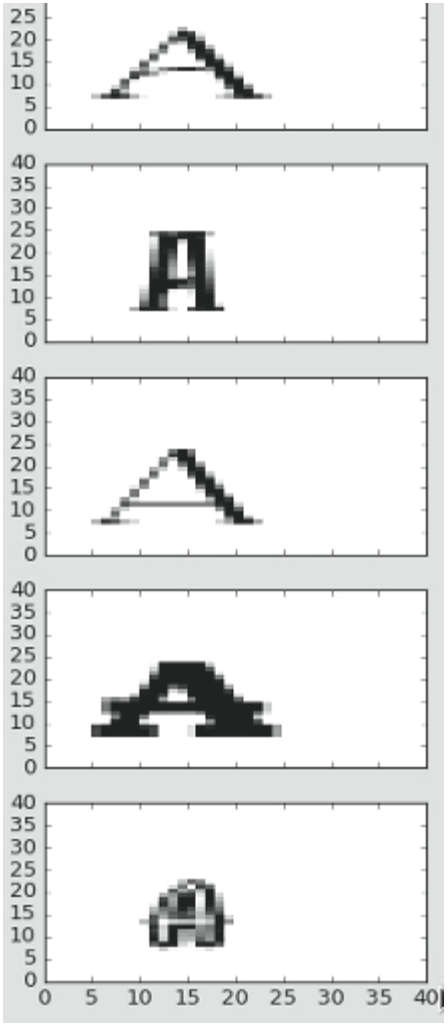


Elements [0:255]
White pixel → 0
Black pixel → 255

Each dimension represents the confidence of a char type

OHE
One-Hot-Encoding

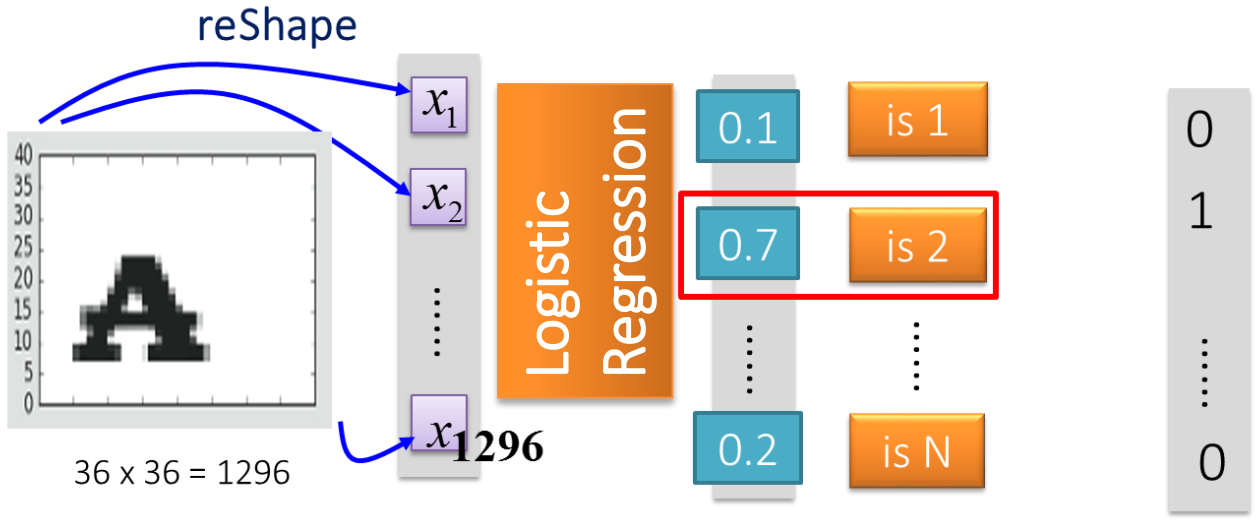
One-hot-encoding (OHE)



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

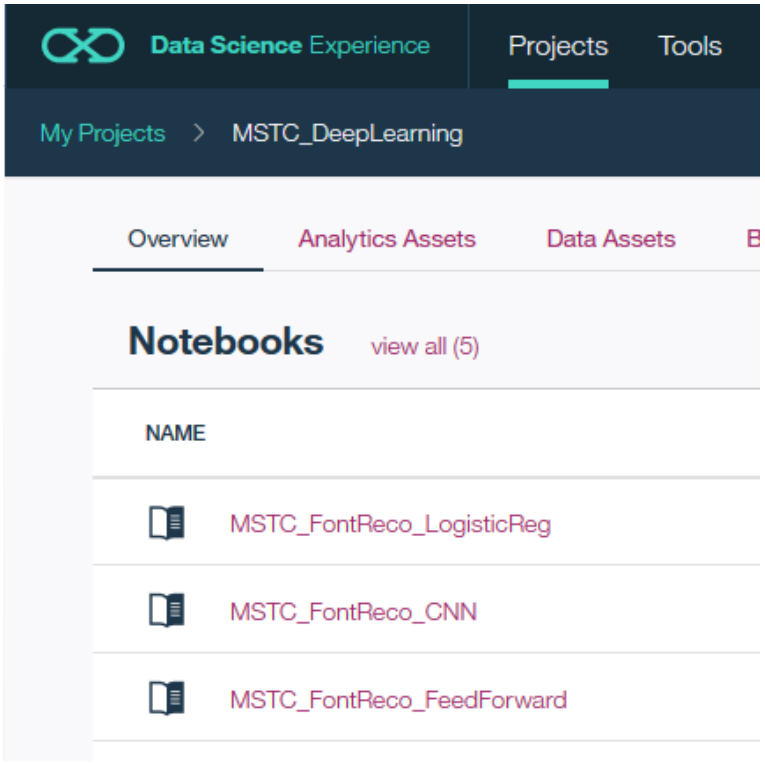
Black pixel → 255

MSTC_FontReco_FeedForward.ipynb

Available at:

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

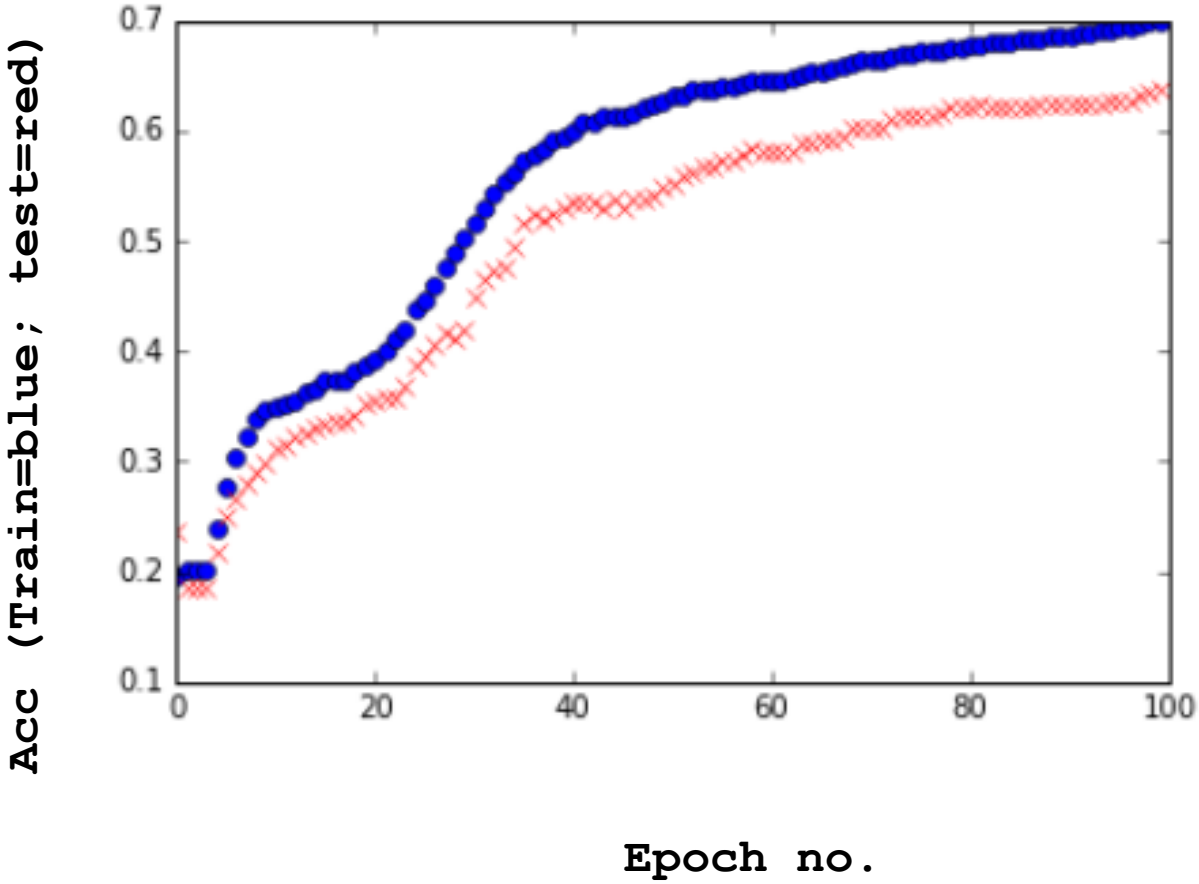
Or



MSTC_FontReco_FeedForward.ipynb

In all FontReco examples we will use TF interactive sessions:

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


Acc (Train=blue; test=red)
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
20
40
60
80
100
Epoch no.

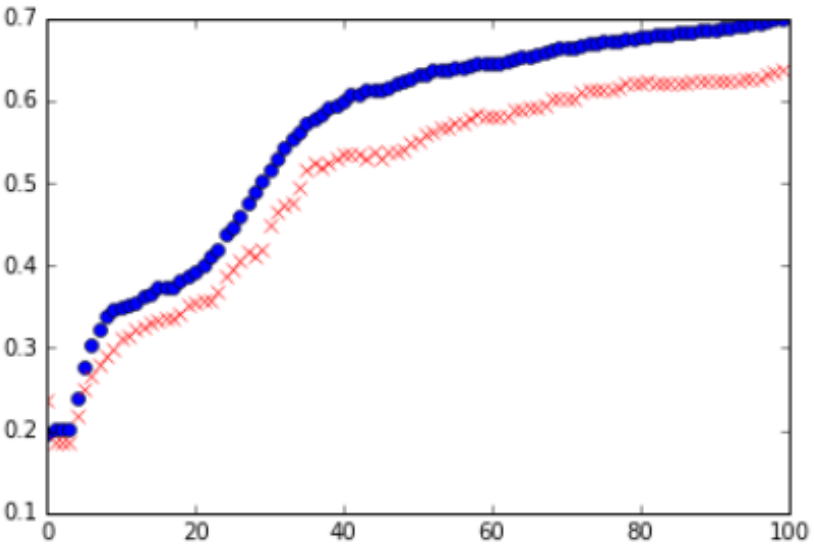
MSTC_FontReco_FeedForward.ipynb

```
# 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

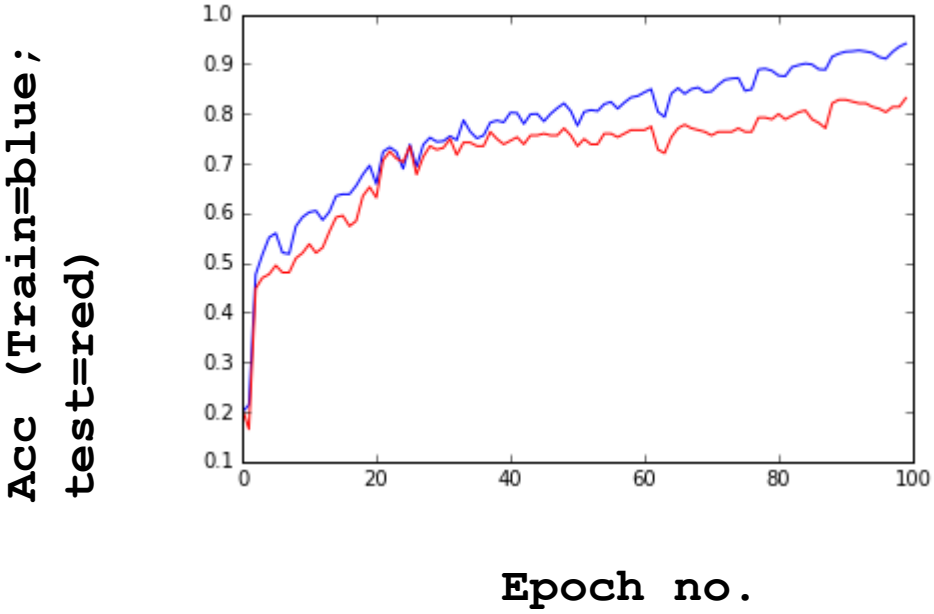
```
# Output Layer
W3 = tf.Variable(tf.truncated_normal([num_hidden2, 5],
                                     stddev=1./math.sqrt(5)))
b3 = tf.Variable(tf.constant(0.1,shape=[5]))

# 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
```

Logistic Regression

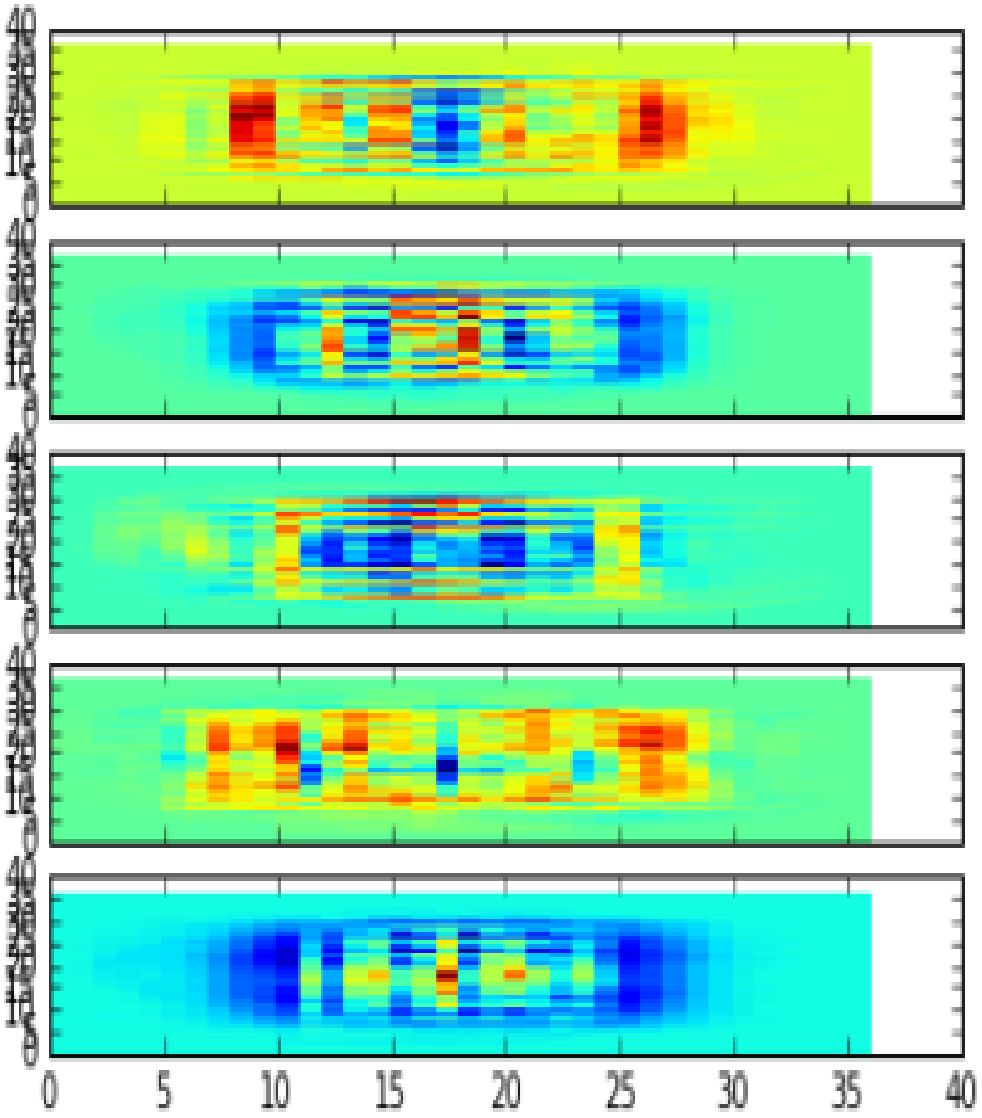


train_step = tf.train.GradientDescentOptimizer(0.55).minimize(cross_entropy)

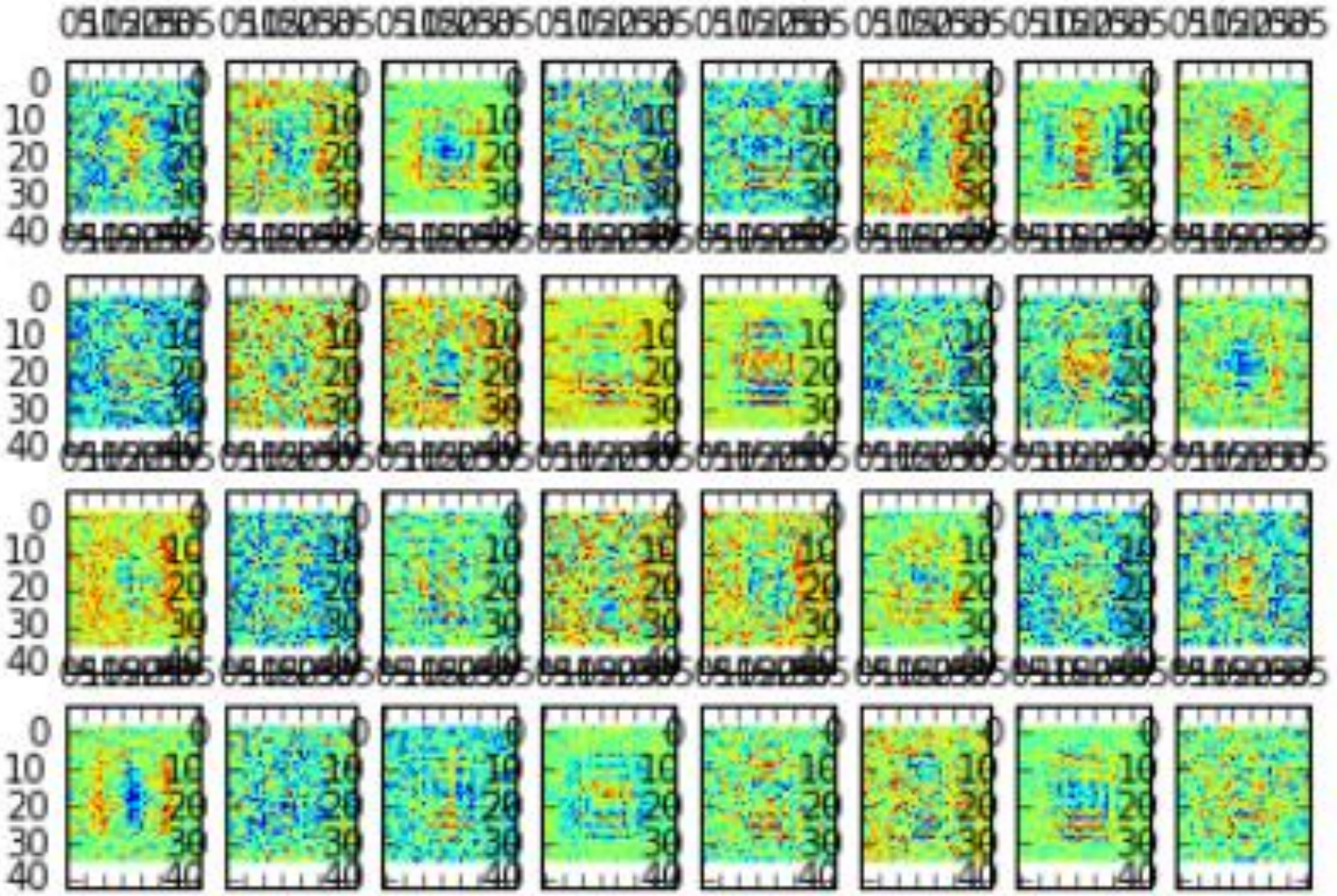
Feed Forward



Logistic Regression

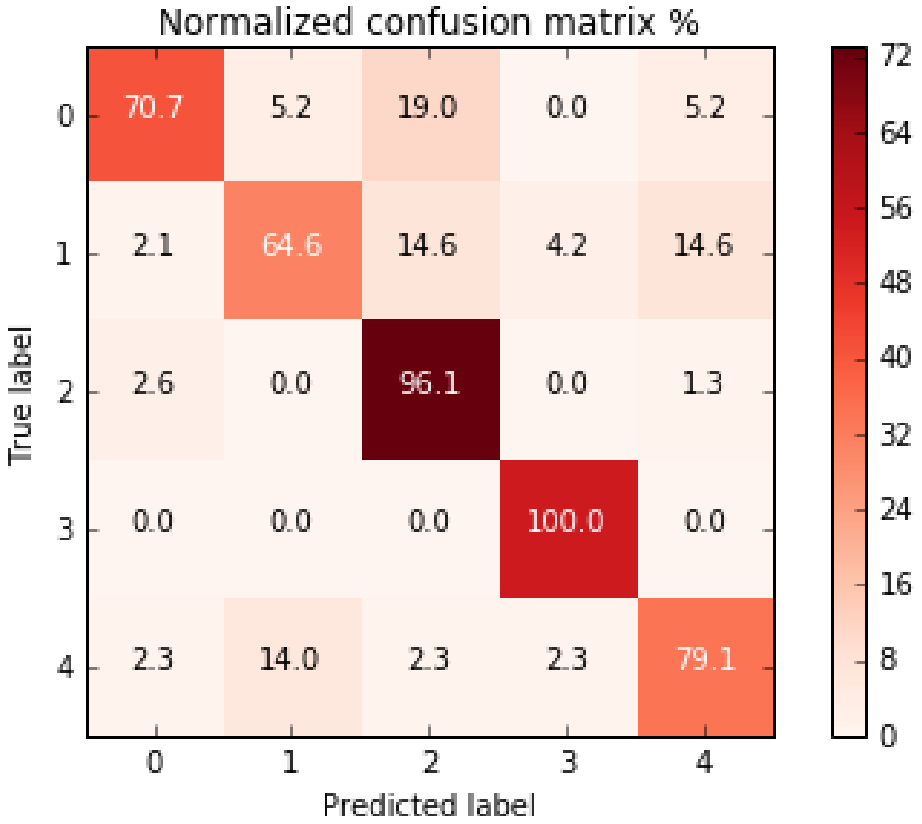


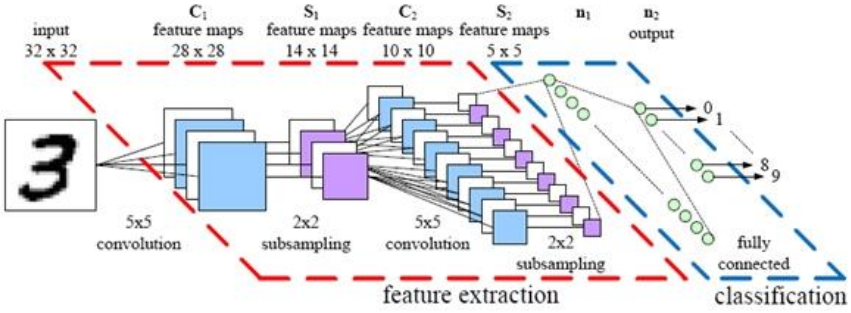
Feed Forward

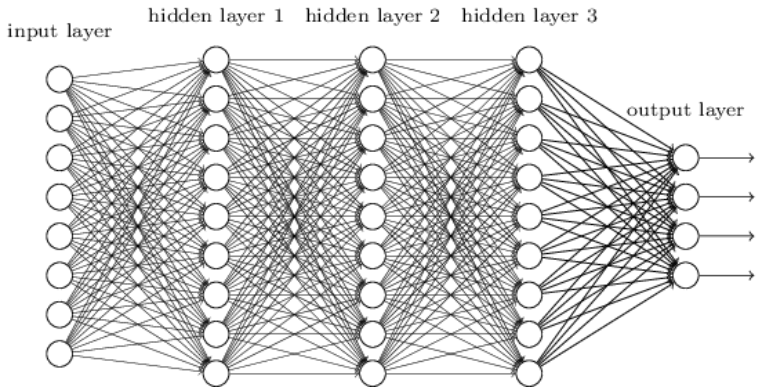


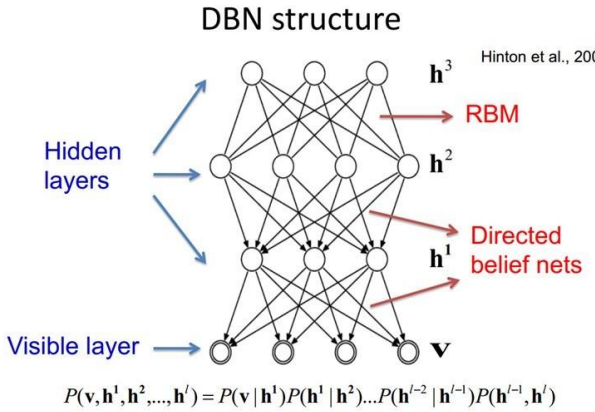
Font Recognition Results

Feed Forward

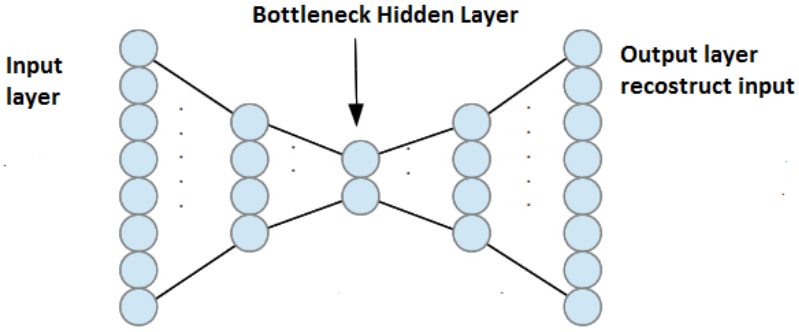


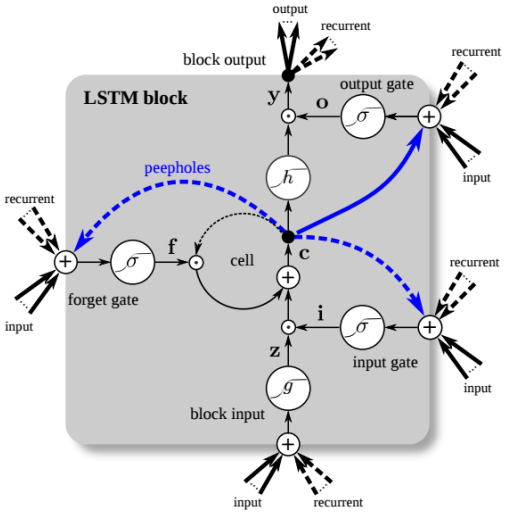


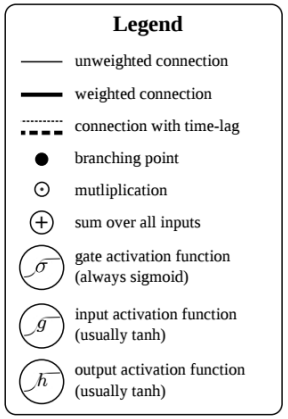




Deep Learning Architectures







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.

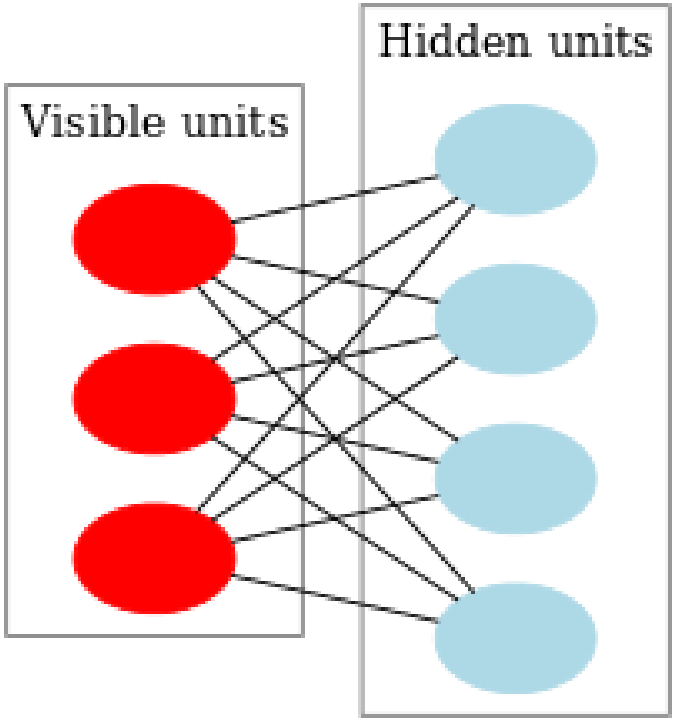
Autoencoders

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.

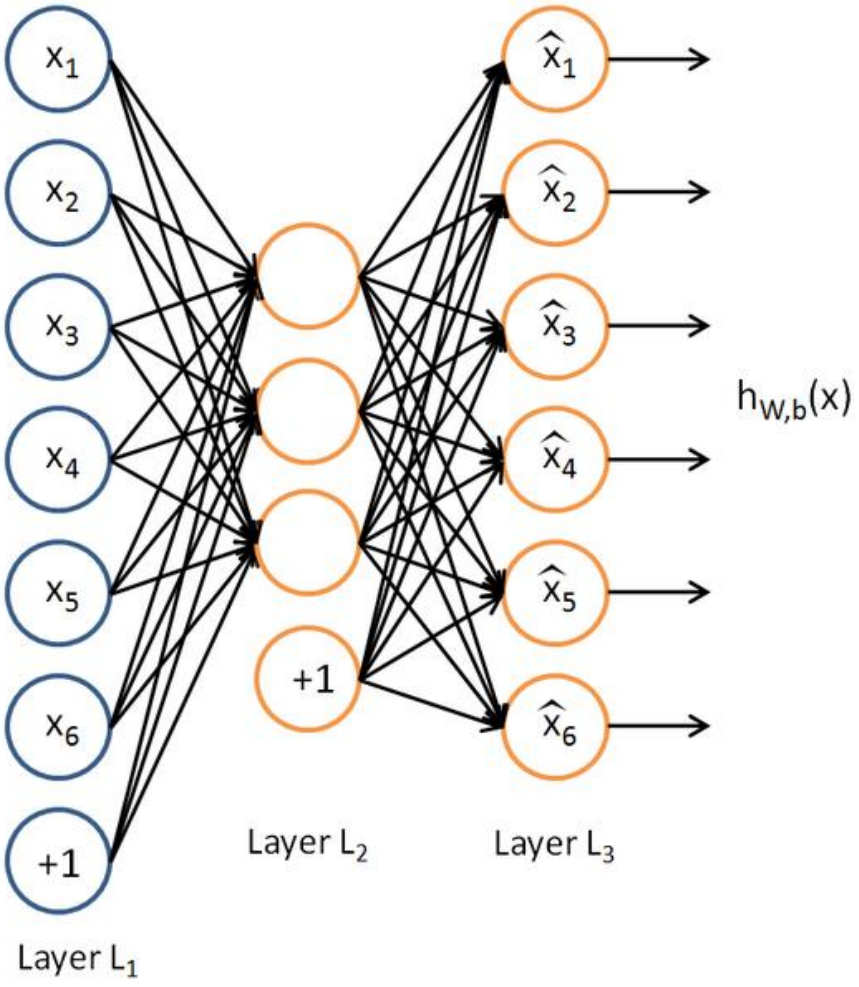
- DBN can be used to generatively pre-train a DNN The learned DBN weights as the initial DNN weights.
  - Back-propagation later for fine tuning of these weights.



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

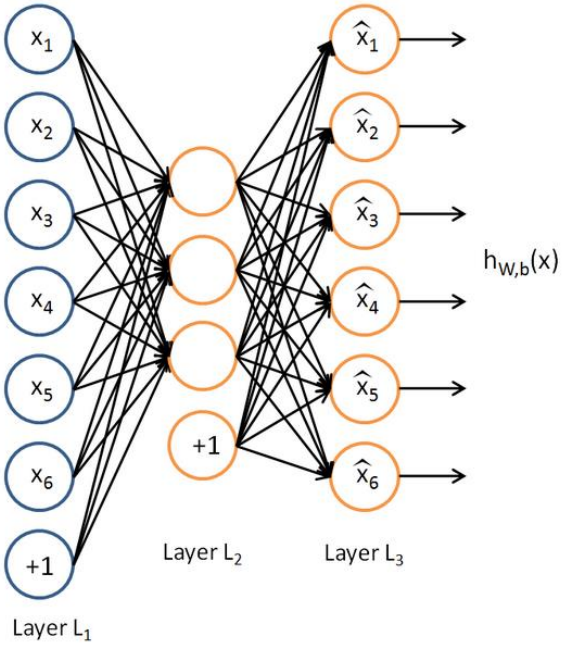
RBM: undirected, generative energy-based model with a "visible" input layer and a hidden layer, and connections between the layers but not within layers

Autoencoder

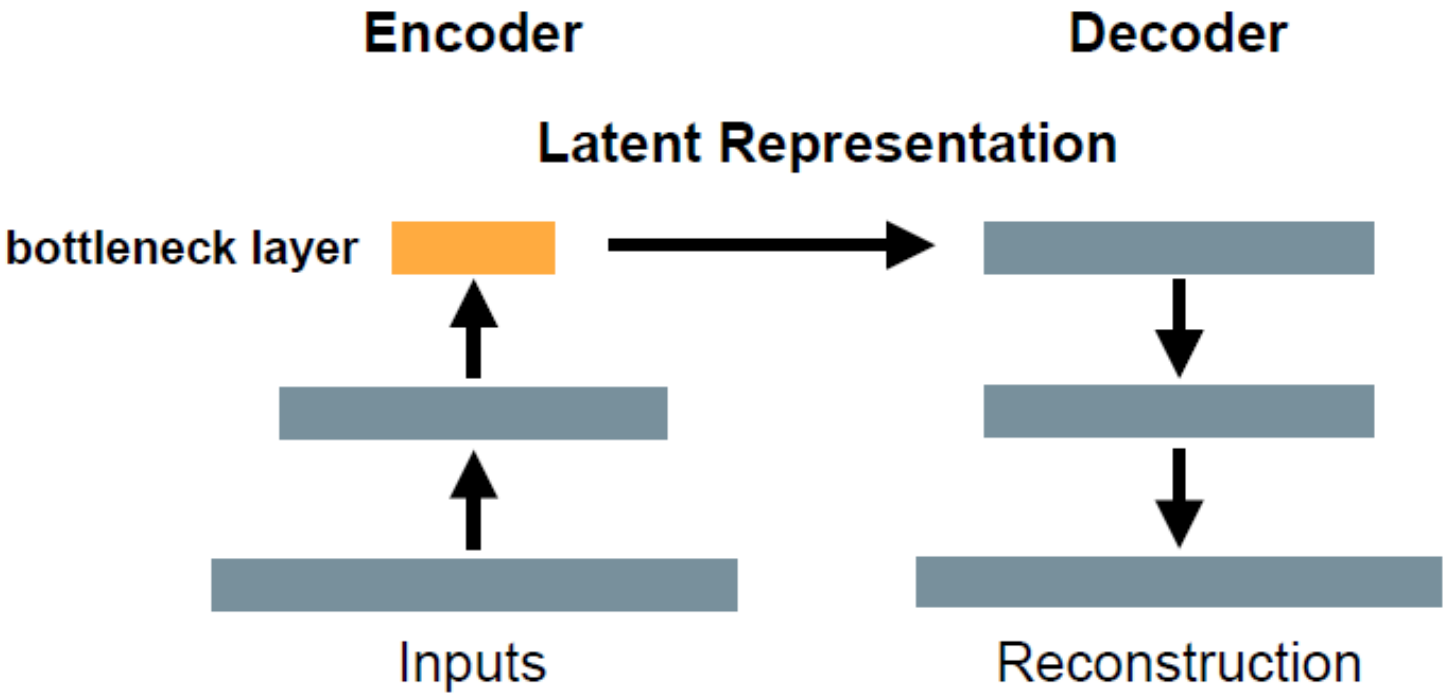


Autoencoder

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

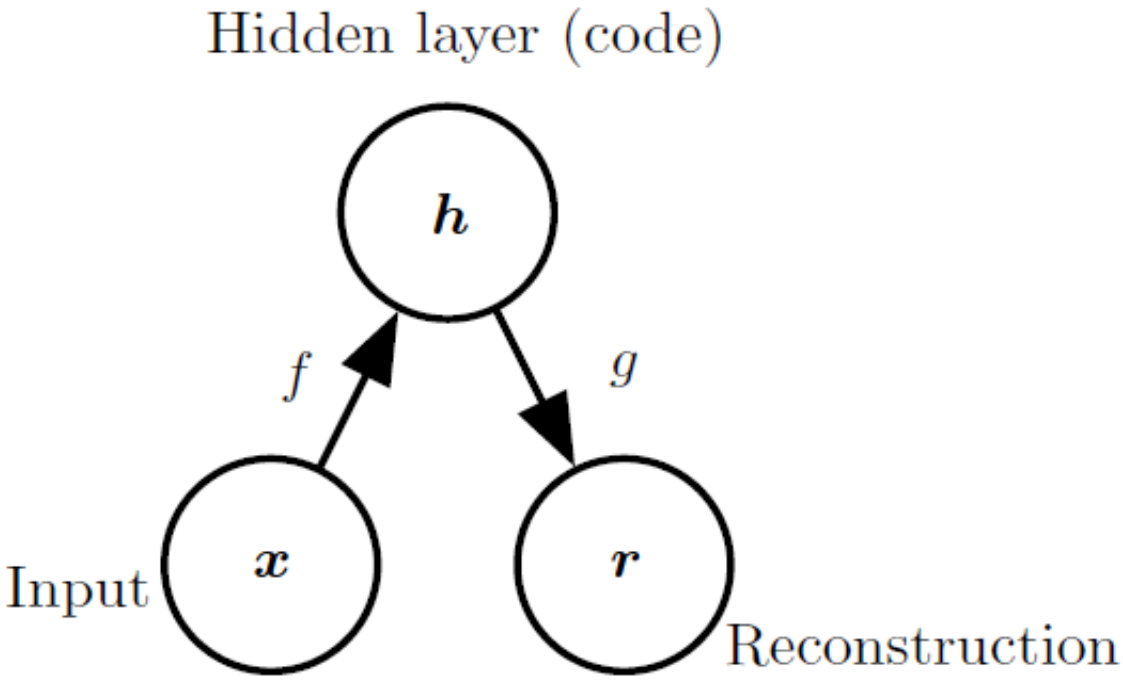


Autoencoders for Representation Learning

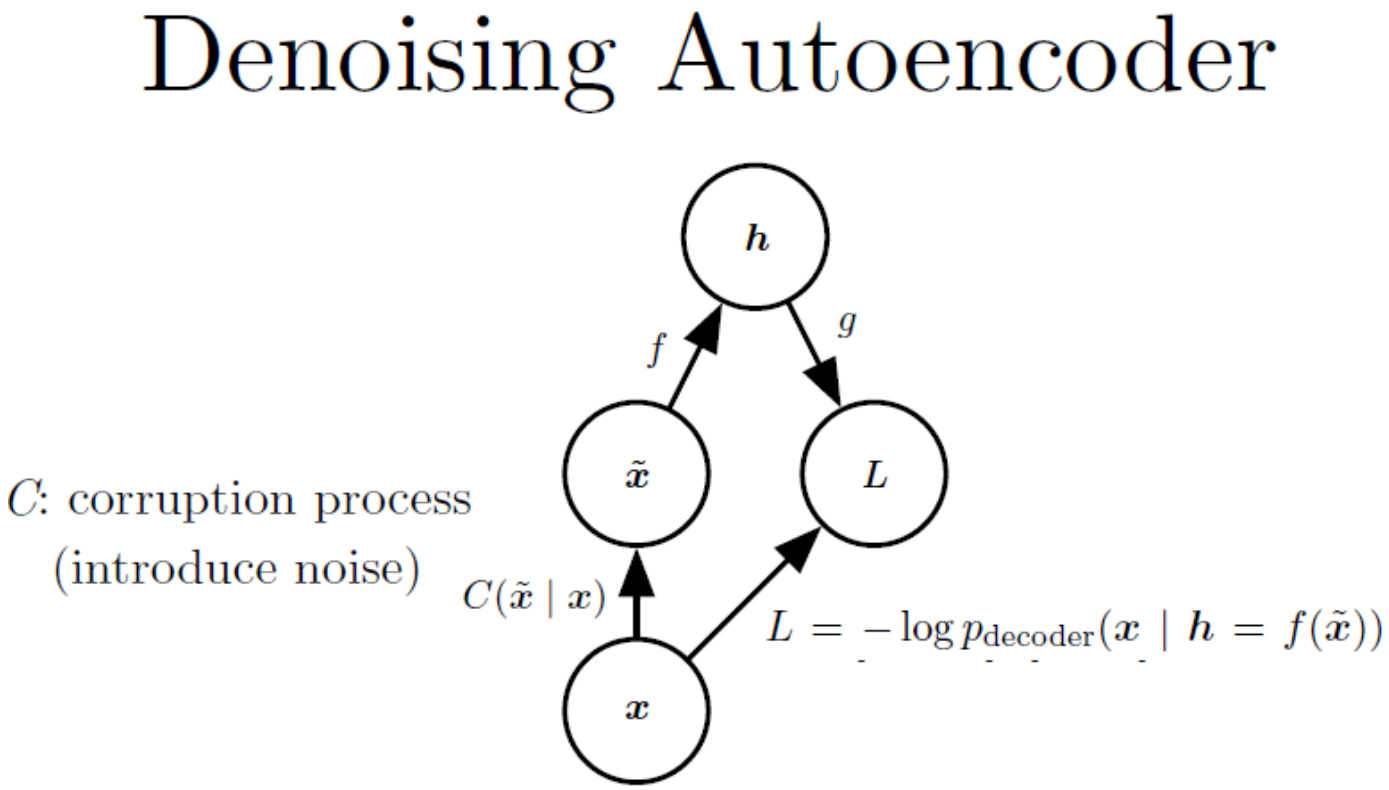


$$L = (x - \hat{x})^2$$

Structure of an Autoencoder

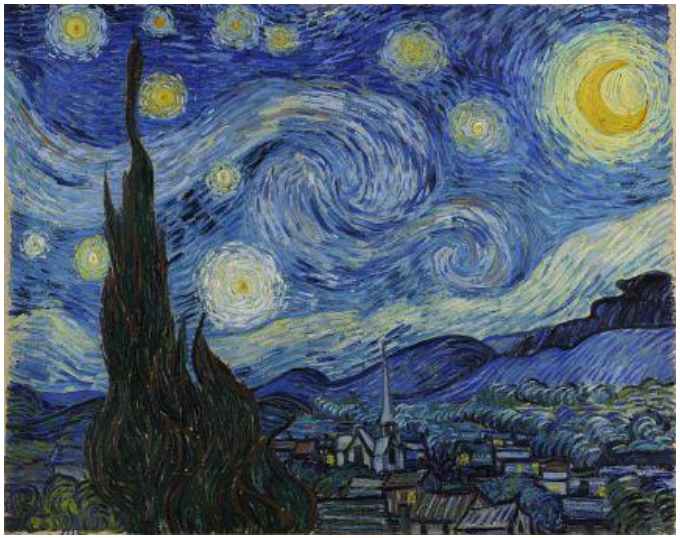


Avoiding Trivial Identity



- 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



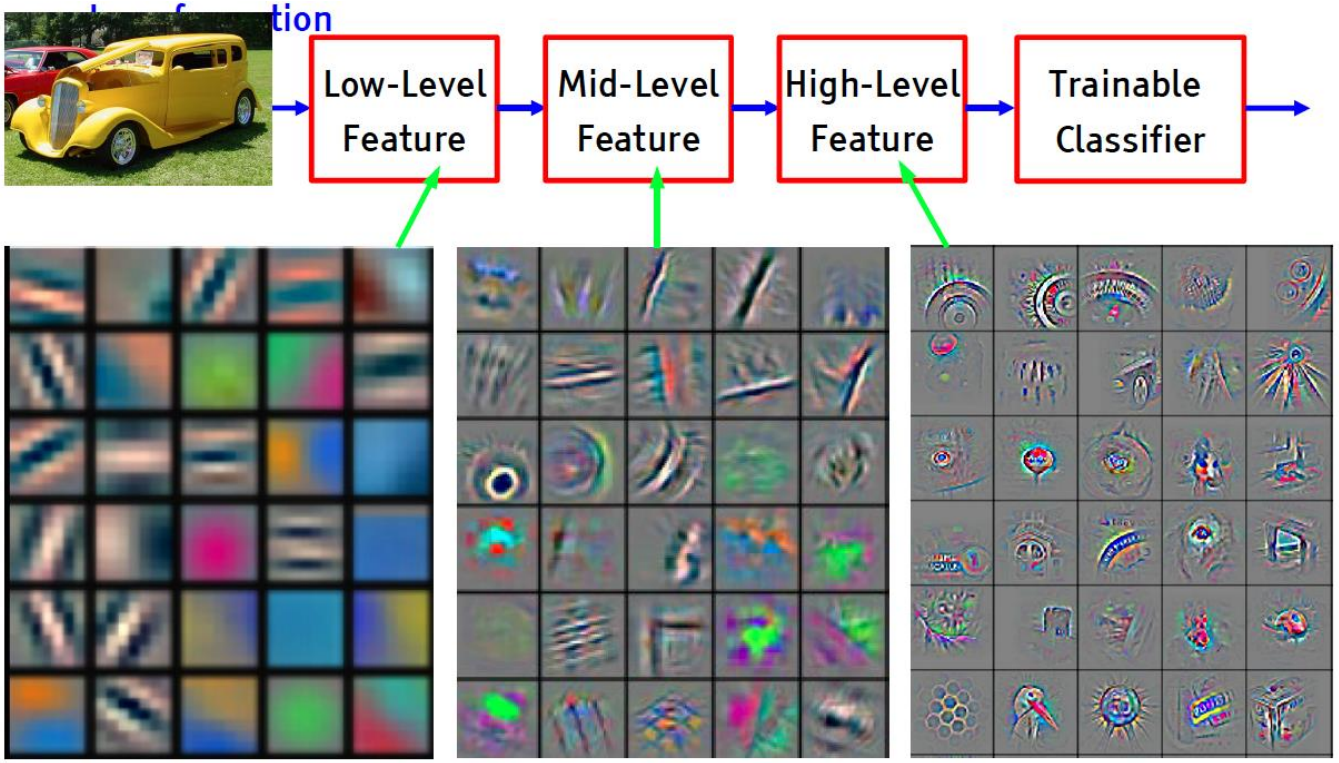
Stack more layers
stack more layers
stack more layers
stack more layers
Stack more layers

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



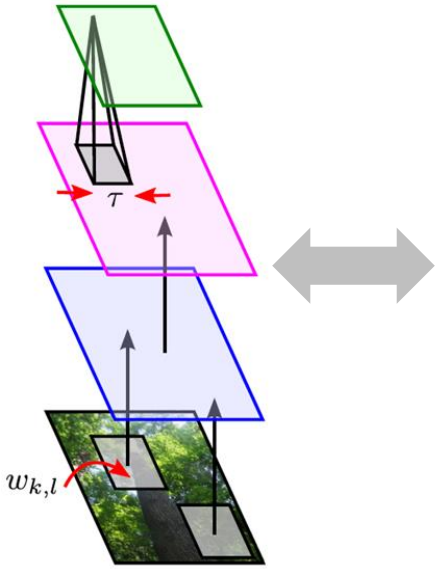
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

CNN: Convolutional Neural Network

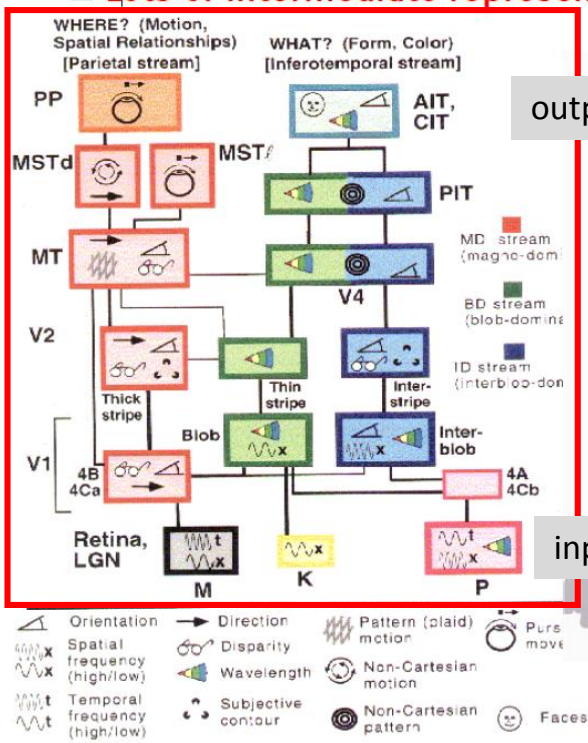
- ***Convolutional Neural Network*** organizes neurons based on animal's visual cortex system, which allows for learning patterns at both local level and global level.
  - Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE,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

The Mammalian Visual Cortex Inspires CNN

Convolutional Neural Net

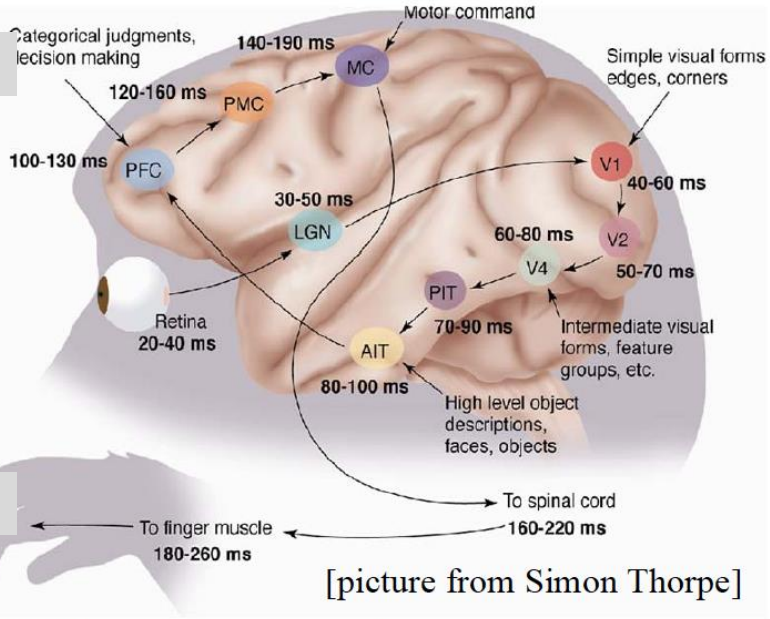


- The ventral (recognition) pathway in the visual cortex has multiple stages
- Retina - LGN - V1 - V2 - V4 - PIT - AIT ....
- Lots of intermediate representations



output

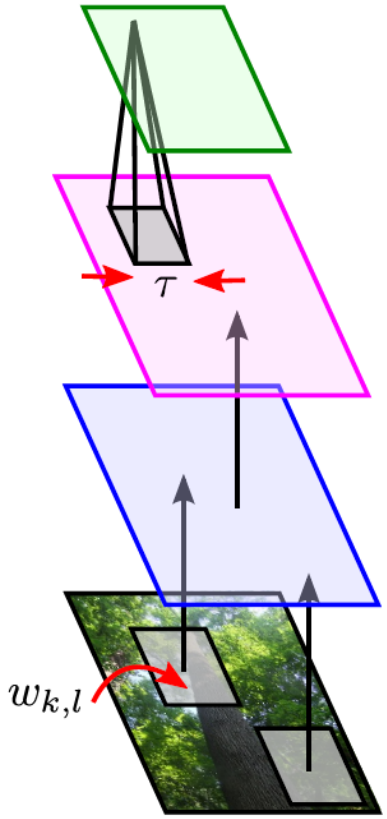
input



[picture from Simon Thorpe]

[Gallant & Van Essen]

Building-blocks for CNN's



$$x_{i,j} = \max_{|k|<\tau,|l|<\tau} y_{i-k,j-l}$$

mean or subsample also used

pooling stage

Feature maps of a larger region are combined.

$$y_{i,j} = f(a_{i,j})$$

e.g. $f(a) = [a]_+$

$f(a) = \text{sigmoid}(a)$

non-linear stage

Feature maps are trained with neurons.

$$a_{i,j} = \sum_{k,l} w_{k,l} z_{i-k,j-l}$$

Shared weights

convolutional stage

Each sub-region yields a feature map, representing its feature.

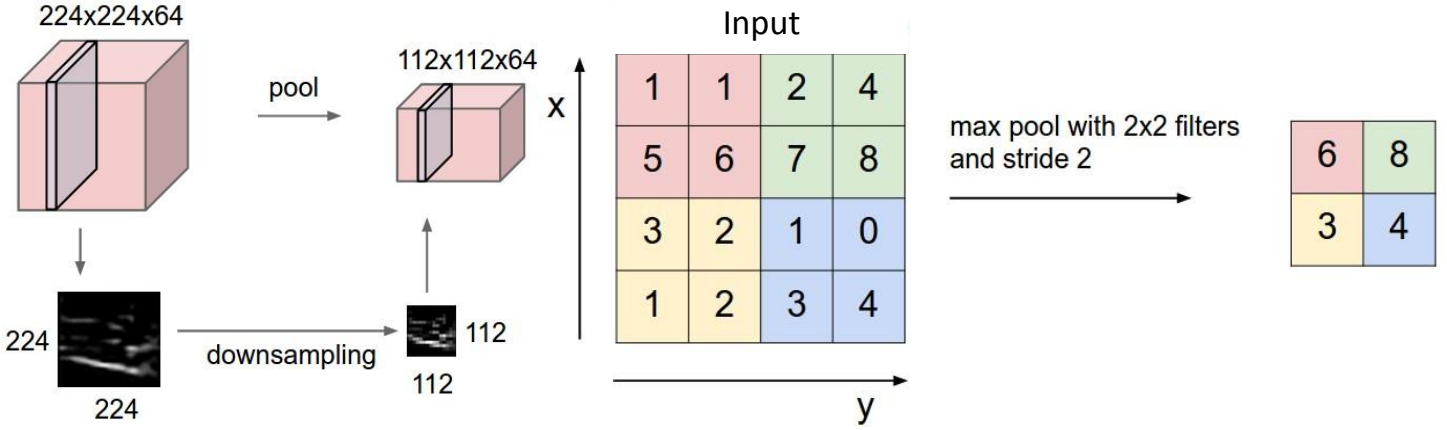
$z_{i,j}$

Images are segmented into sub-regions.

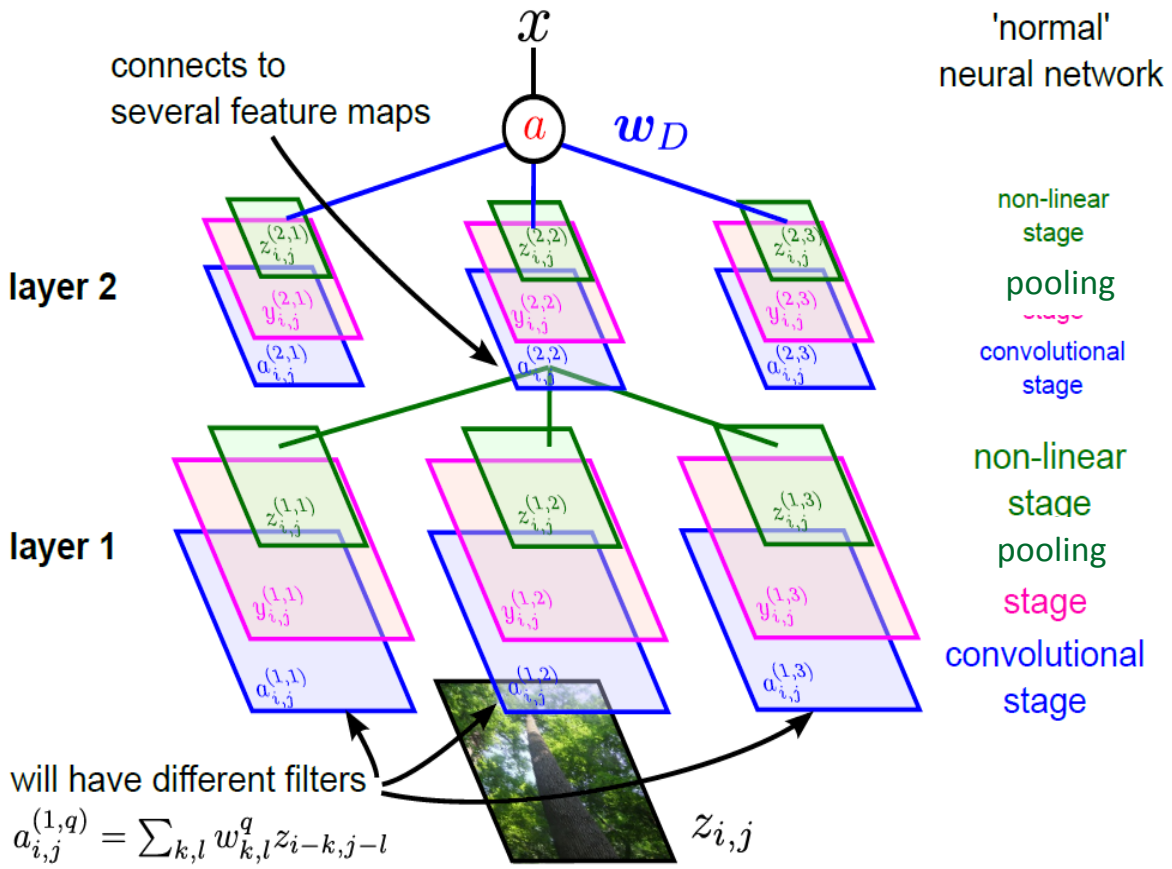
input image

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')
```

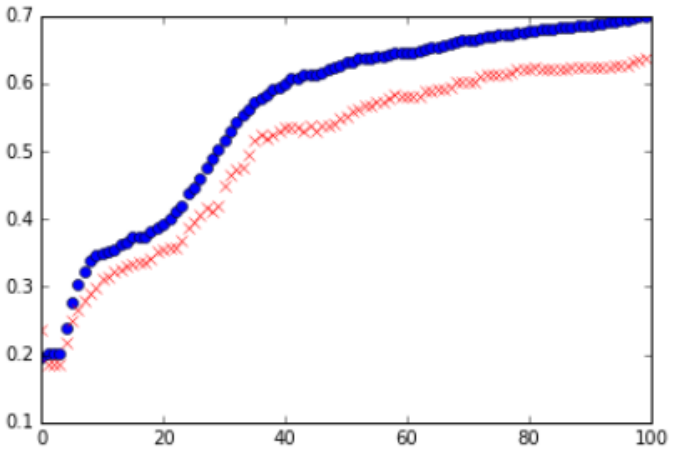
MSTC_FontReco_CNN.ipynb (II)

```
# 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)
```

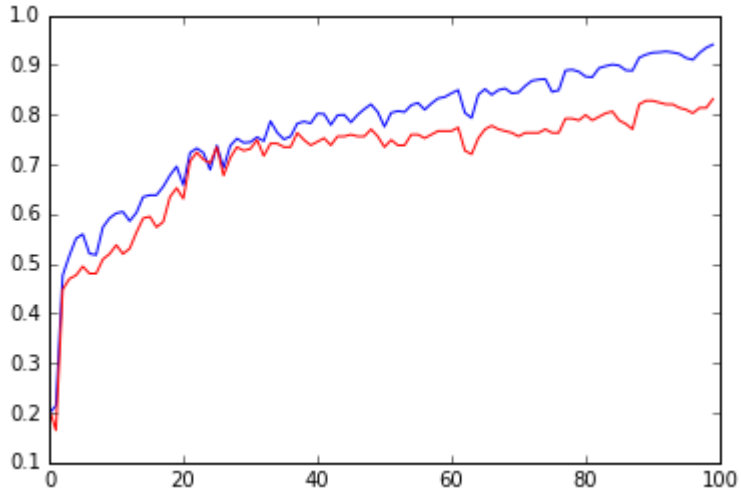
Logistic Regression



Acc (Train=blue; test=red)

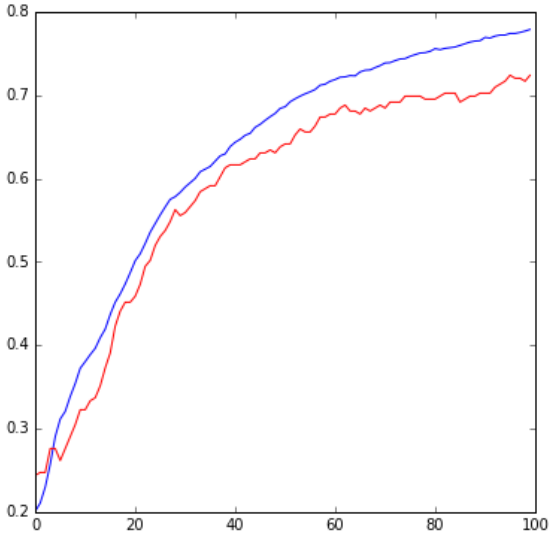
Epoch no.

Feed Forward



Epoch no.

CNN

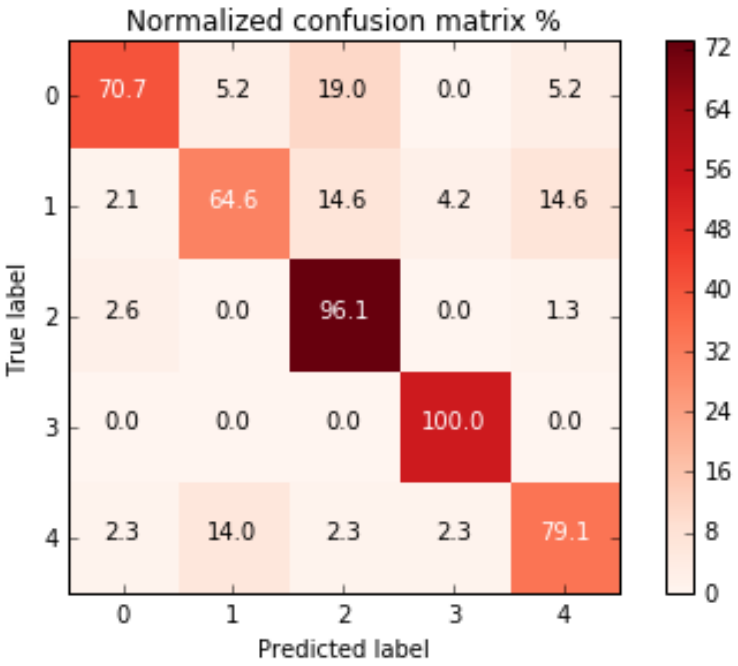


Acc (Train=blue; test=red)

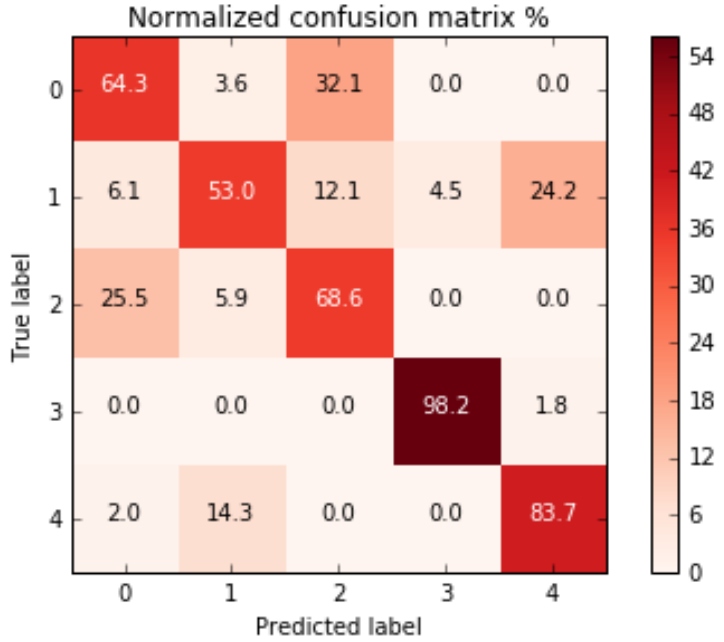
Epoch no.

Font Recognition Results

Feed Forward

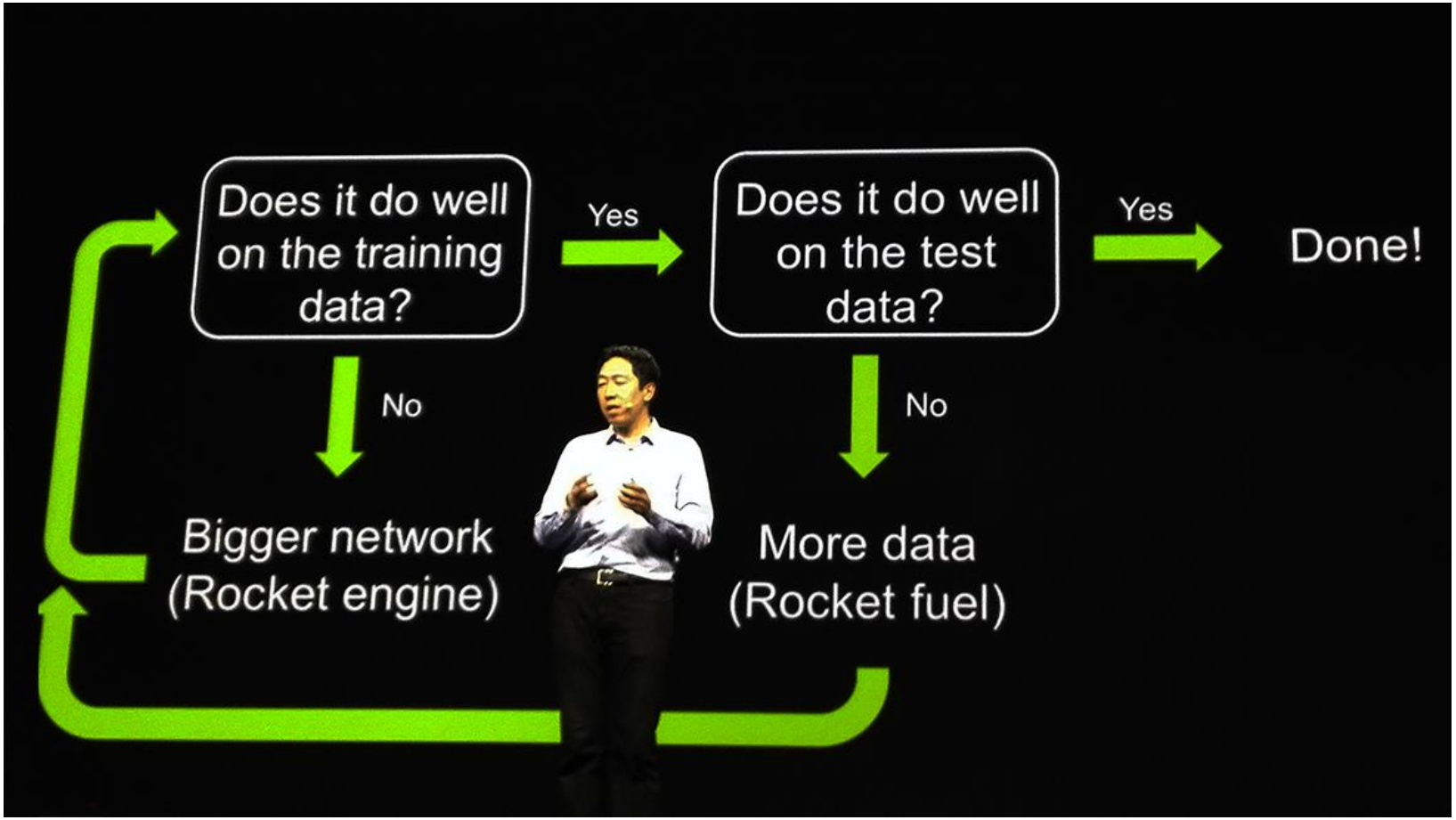


CNN



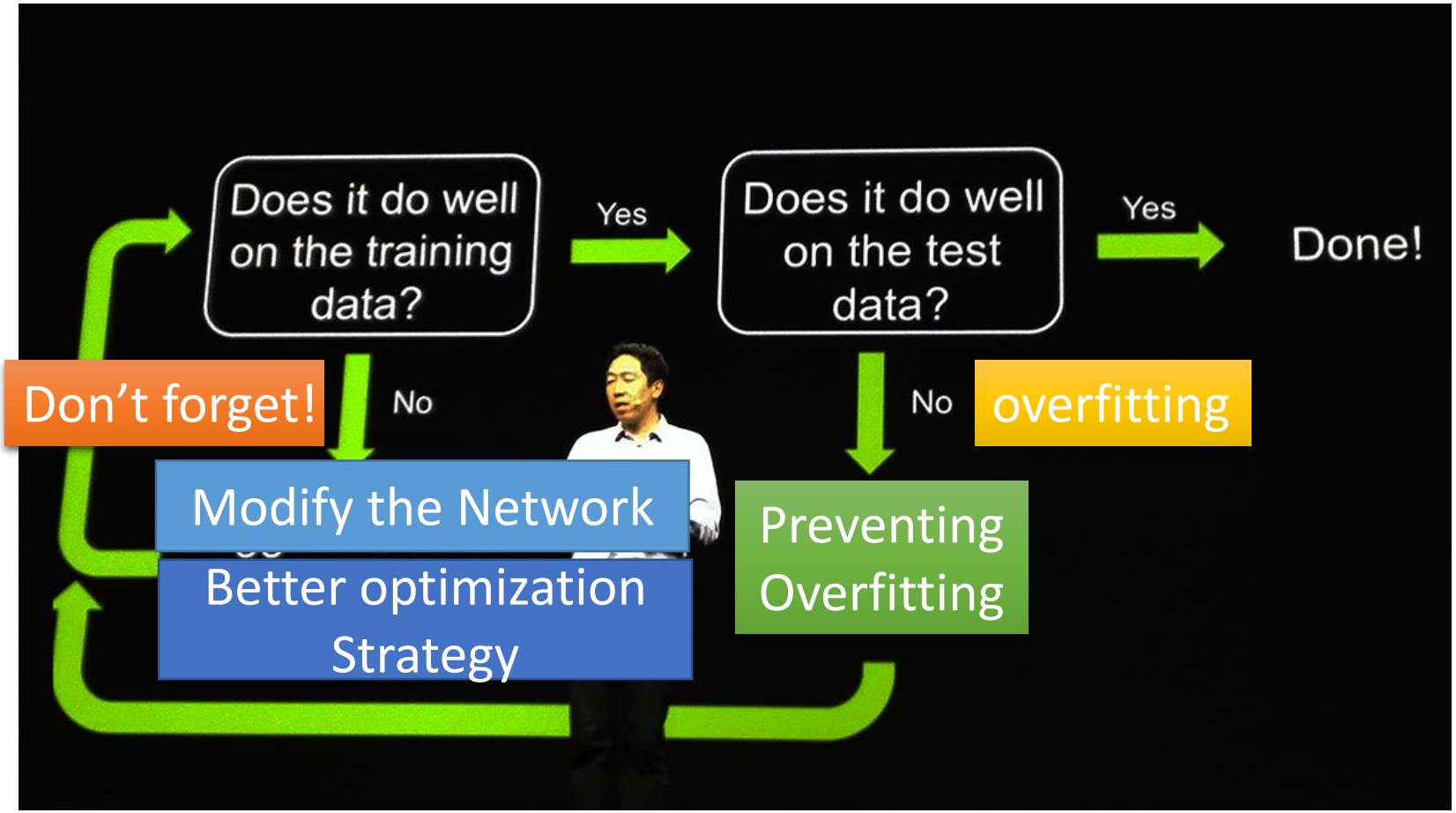
Tips for Training DNN

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

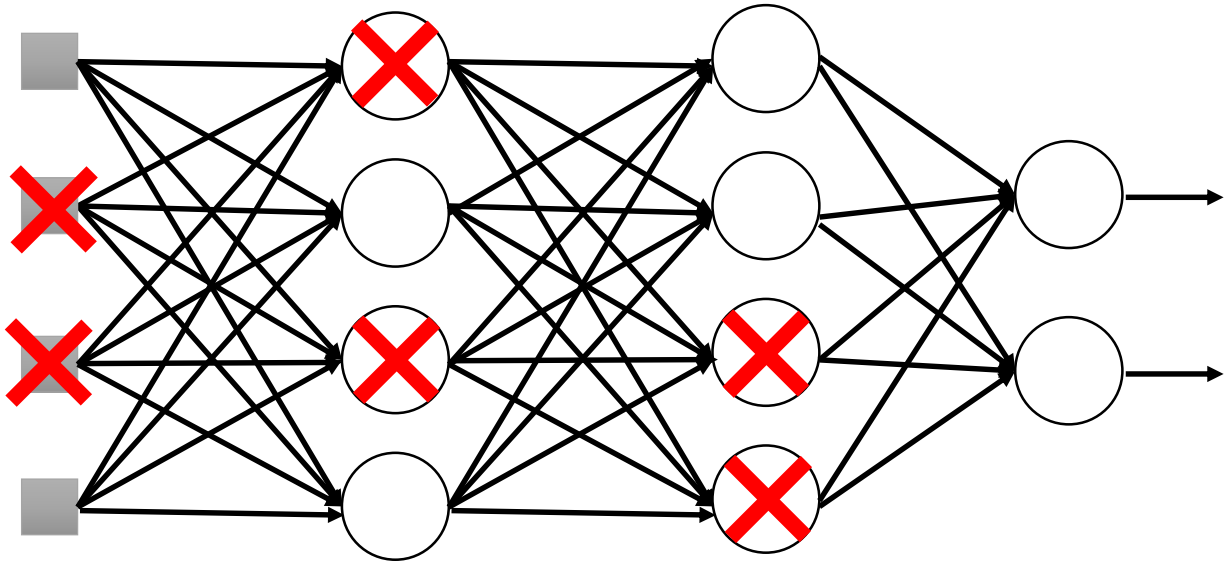
Dropout

Dropout

Pick a mini-batch

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

Training:



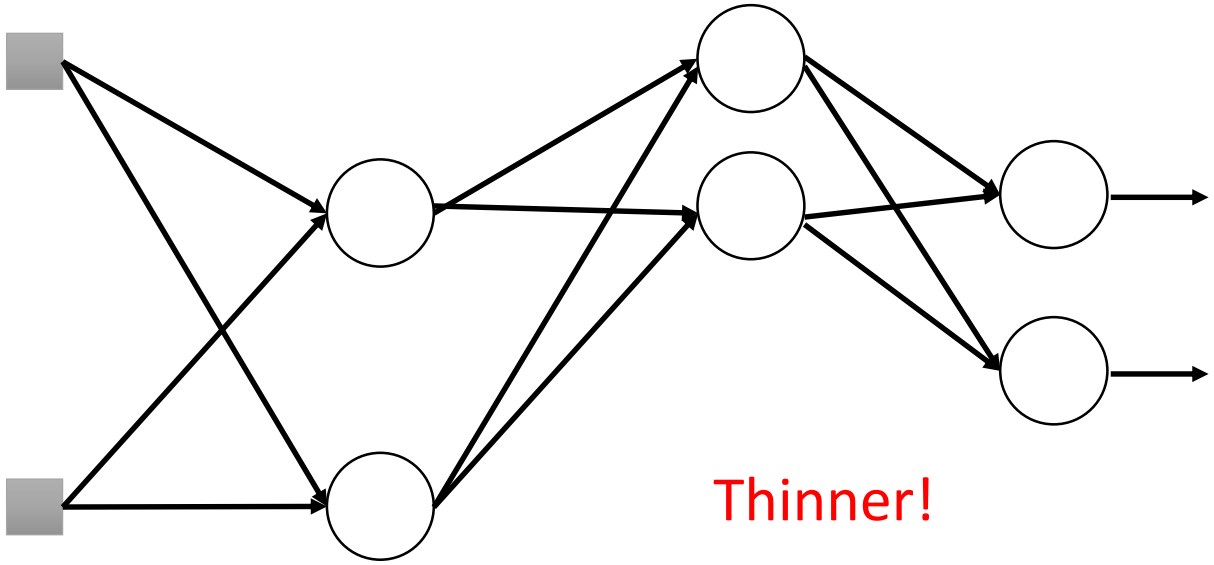
- **Each time before computing the gradients**
  - Each neuron has p% to dropout

Dropout

Pick a mini-batch

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

Training:



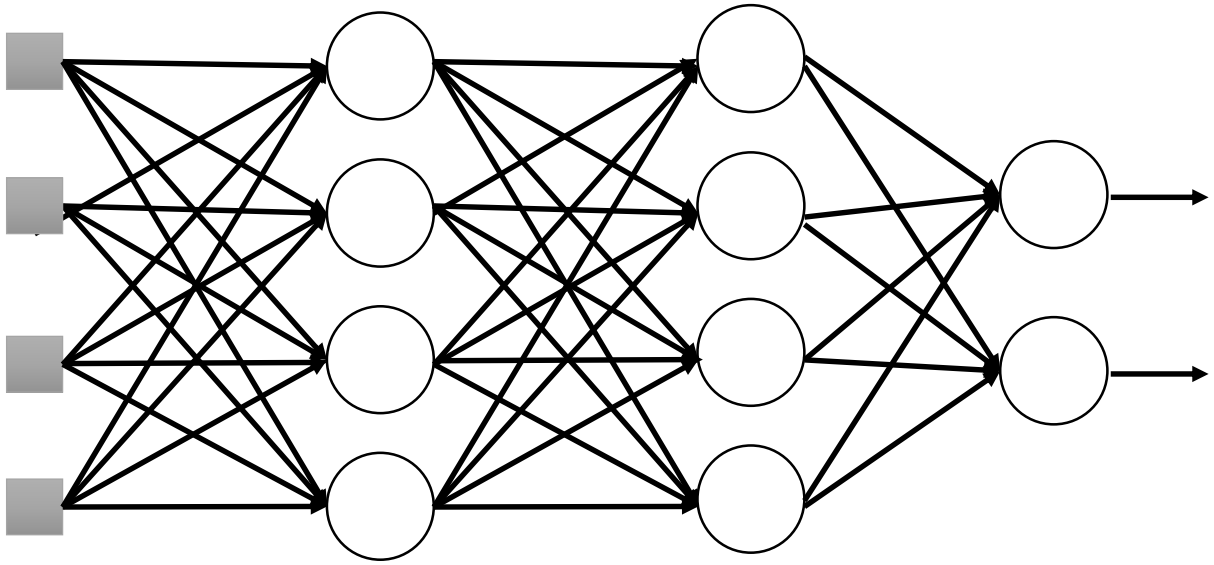
- **Each time before computing the gradients**
  - Each neuron has p% to dropout

    The structure of the network is changed.
  - Using the new network for training

For each mini-batch, we resample the dropout neurons

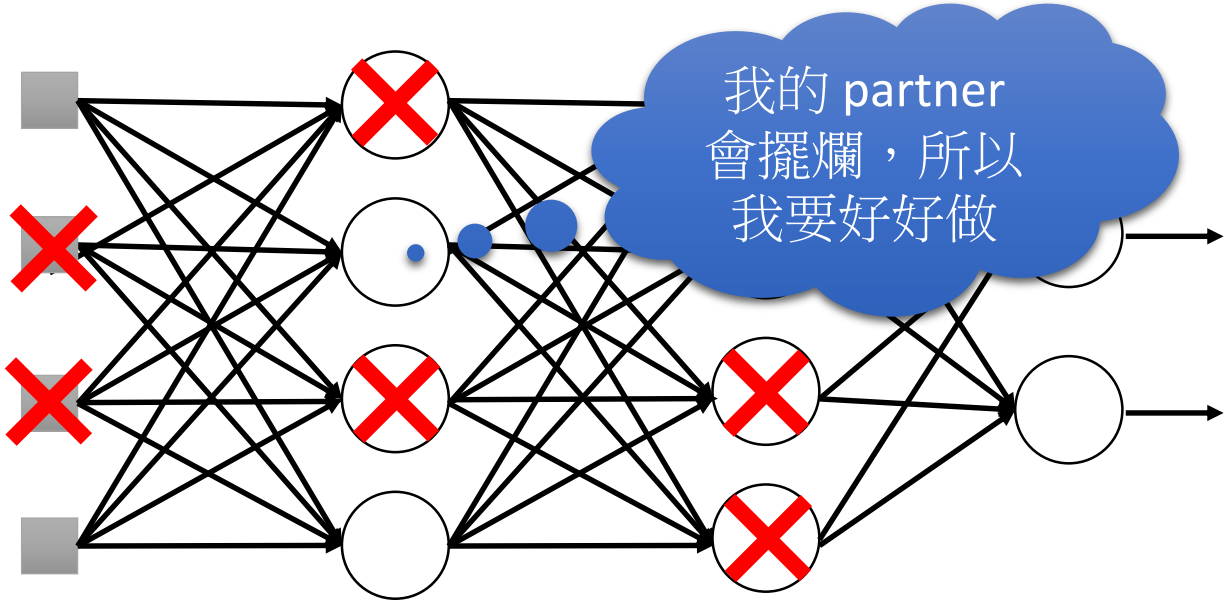
Dropout

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



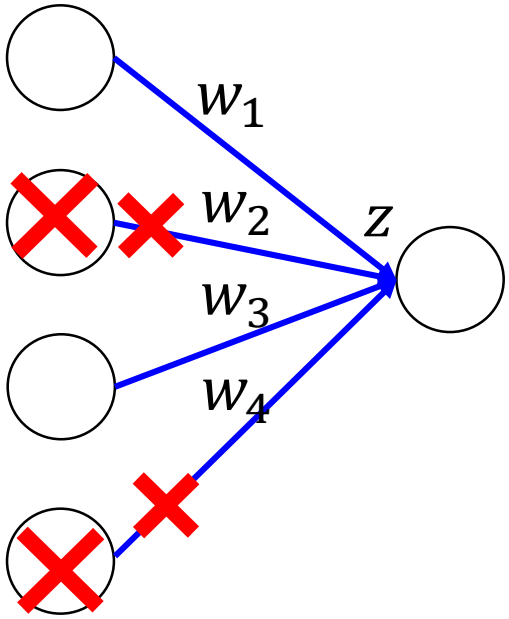
- When teams up, if everyone expect the partner will do the work, nothing will be done finally.
- However, if you know your partner will dropout, you will do better.
- When testing, no one dropout actually, so obtaining good results eventually.

Dropout - Intuitive Reason

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

Training of Dropout

Assume dropout rate is 50%

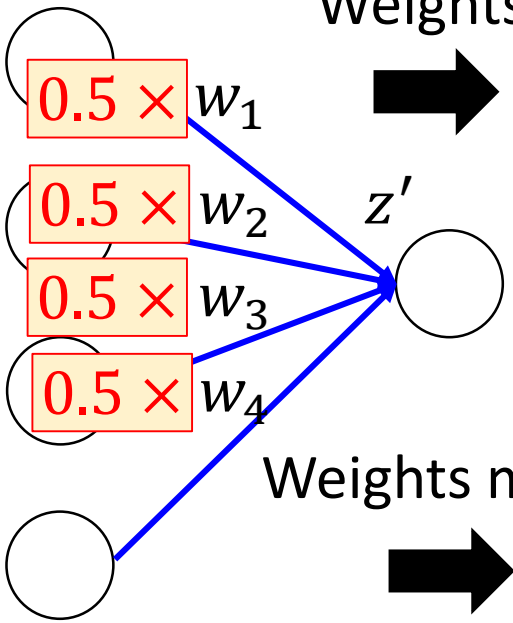


Testing of Dropout

No dropout

Weights from training

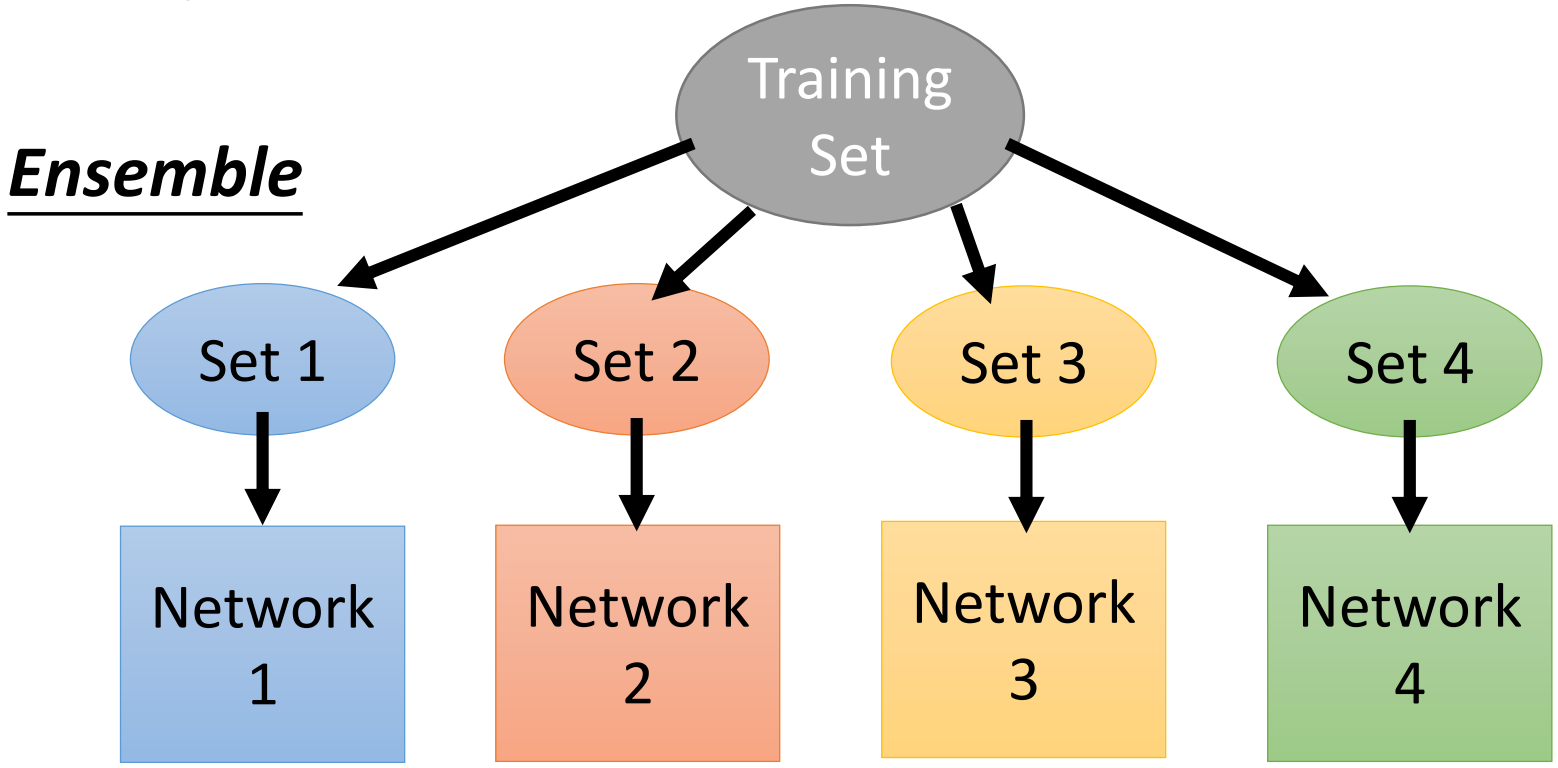
$z' \approx 2z$



Weights multiply (1-p)%

$z' \approx z$

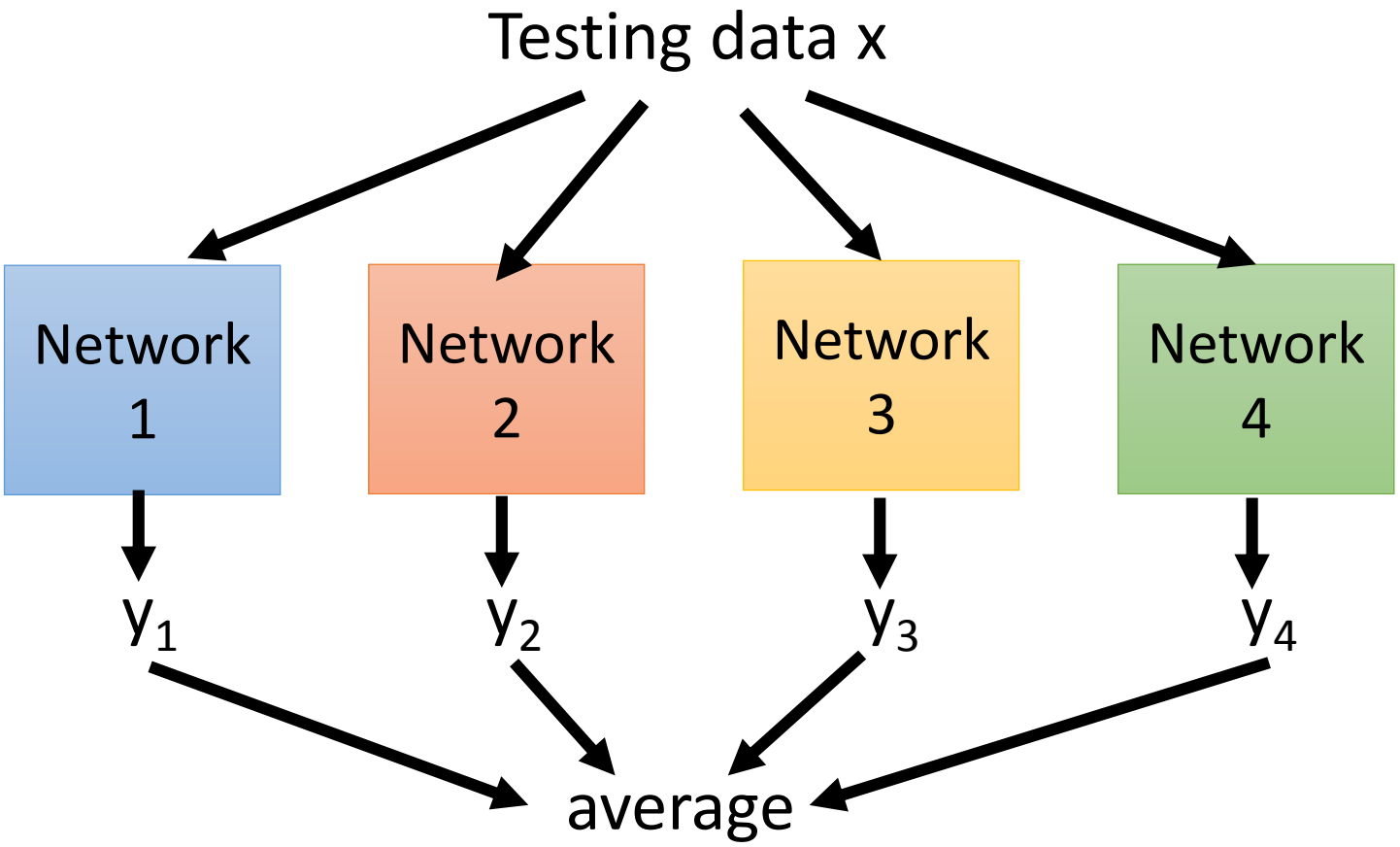
Dropout is a kind of ensemble.



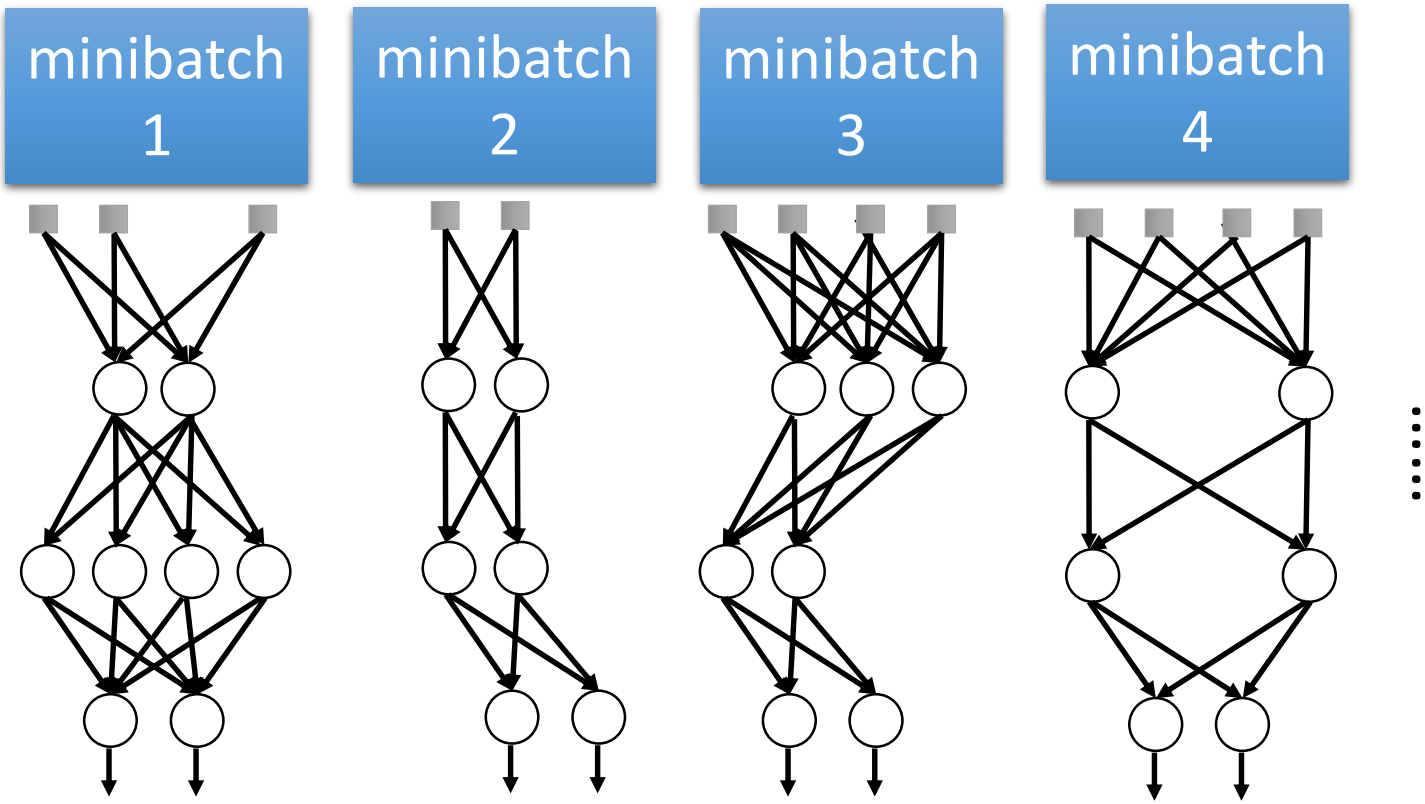
Train a bunch of networks with different structures

Dropout is a kind of ensemble.

Ensemble



Dropout is a kind of ensemble.



Training of Dropout

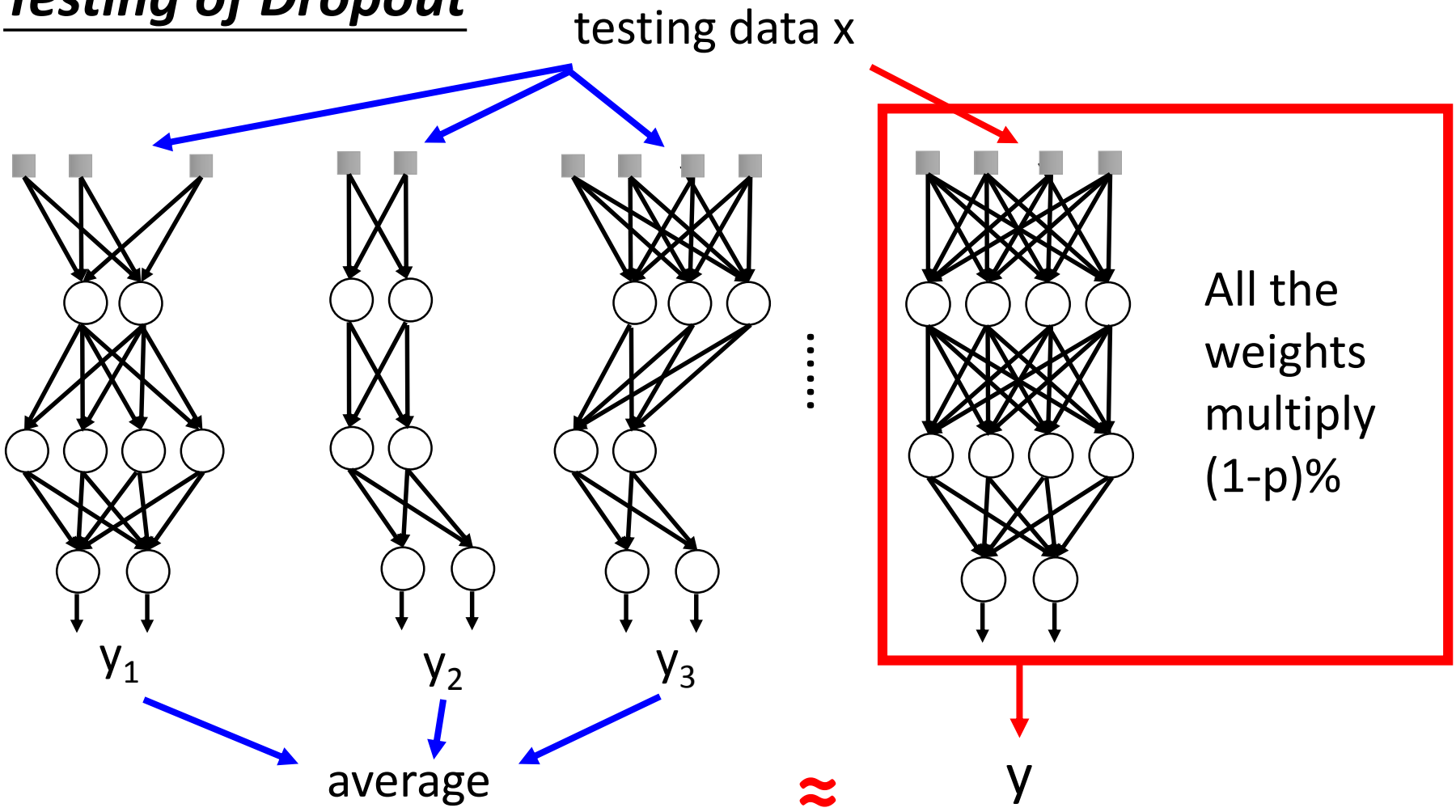
M neurons

2^M possible networks

- 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 Learing 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