Deep Learning Seminar
Day-2

Master of Science in Signal Theory and Communications

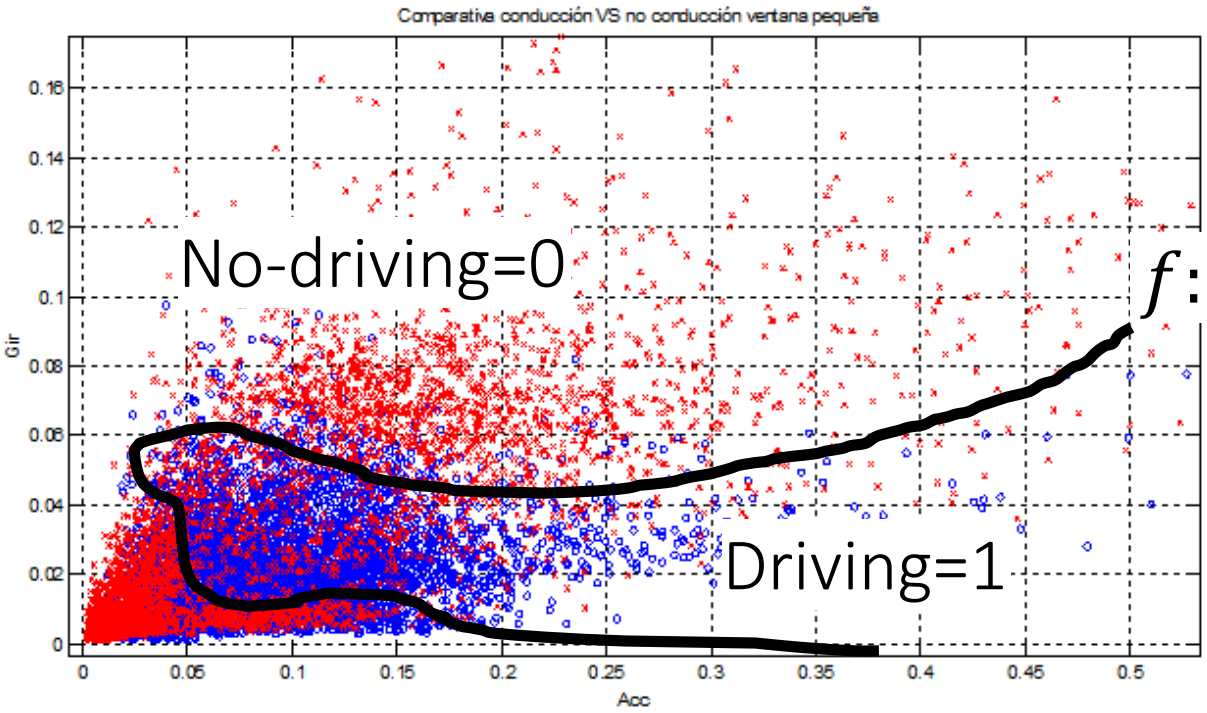
TRACK: Signal Processing and Machine Learning for Big Data

Prof. Luis A. Hernández Gómez
luisalfonso.hernandez@upm.es

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 $\phi(.)$

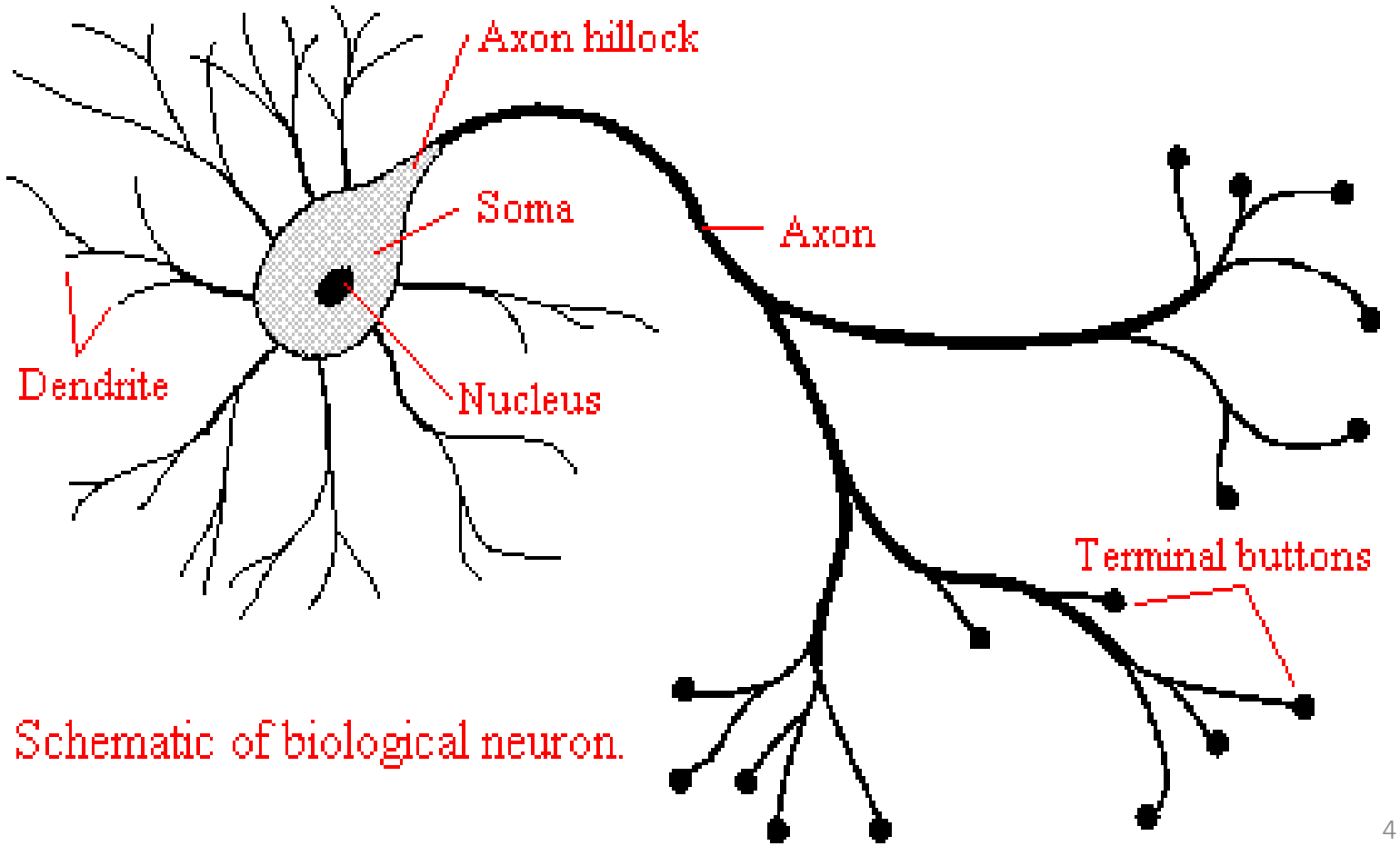
2. Use a very generic $\phi(.)$ as kernel machines (e.g. SVM, RBF kernel)

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

From linear classifiers TO Neural Networks

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

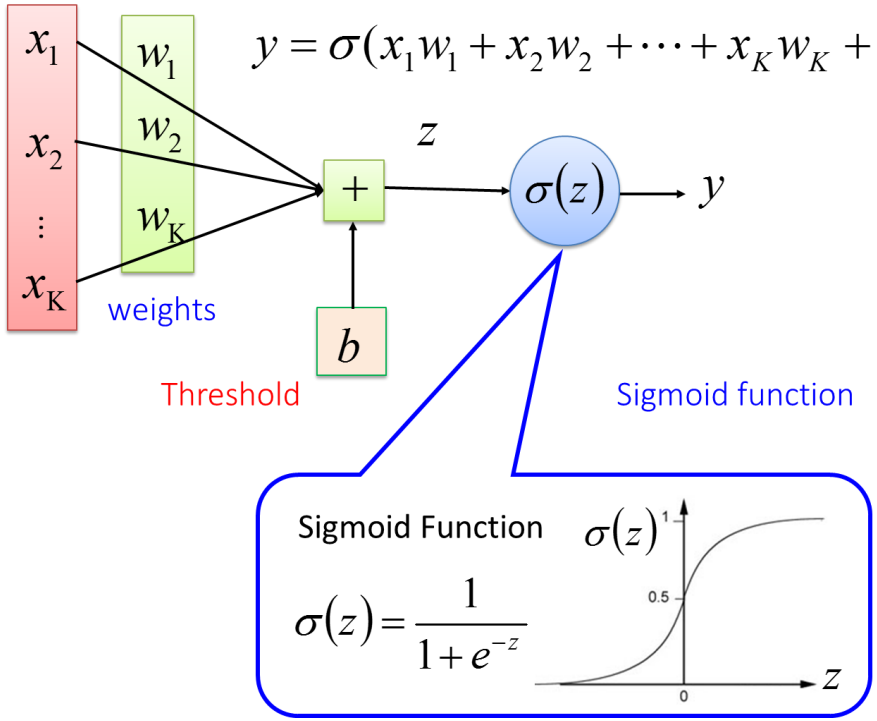
…from a broad class of functions



Schematic of biological neuron.

From linear classifiers TO Neural Networks

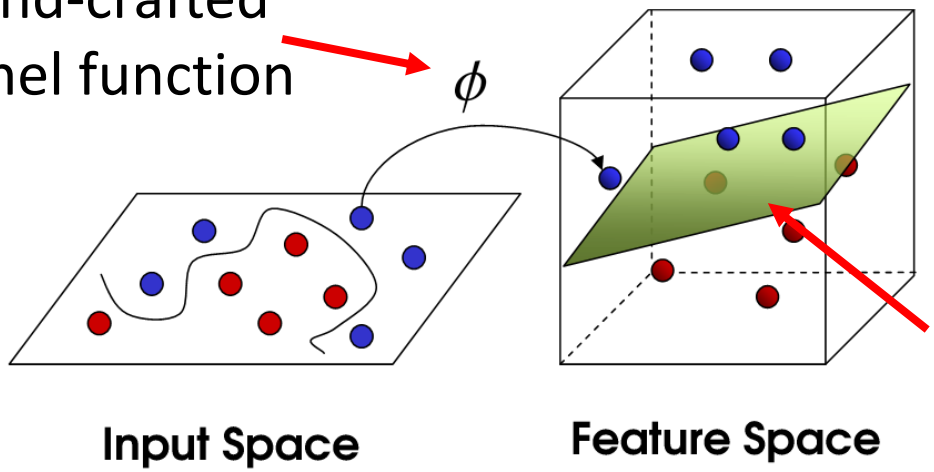
- Recall that this is also logistic regression…



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

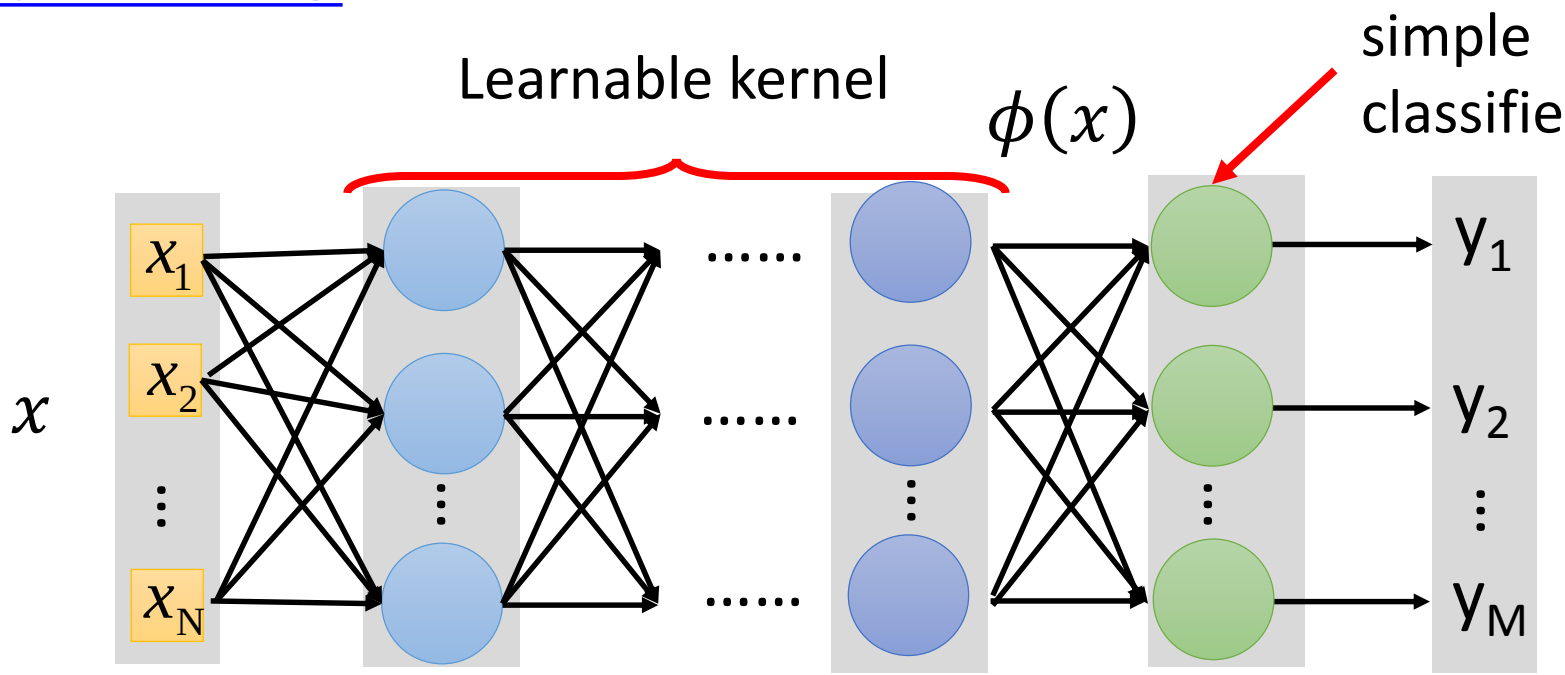
From: DL Tutorial
Hung-yi Lee

SVM



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

Deep Learning

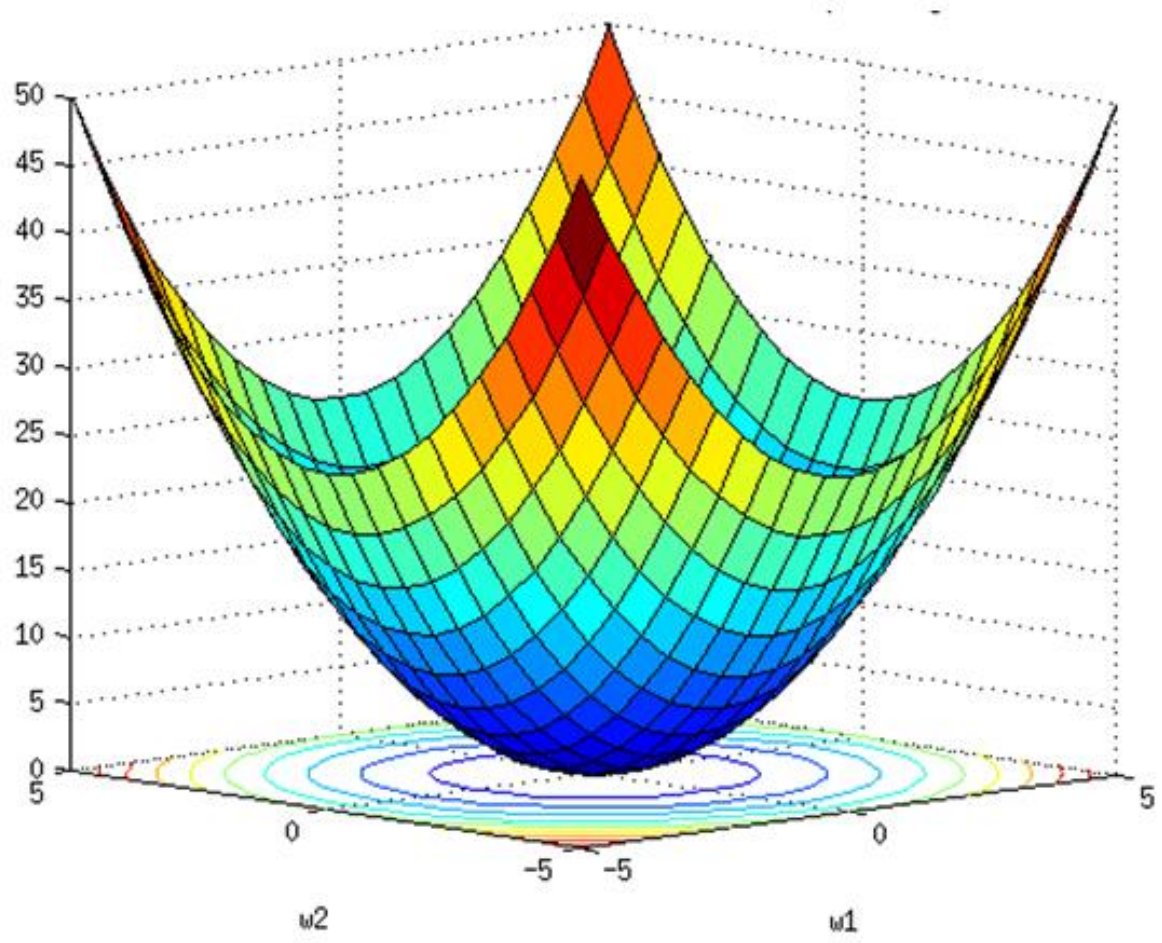


OPTIMIZATION

Gradient Descent

Cost
Or
Loss function

$C(\theta)$



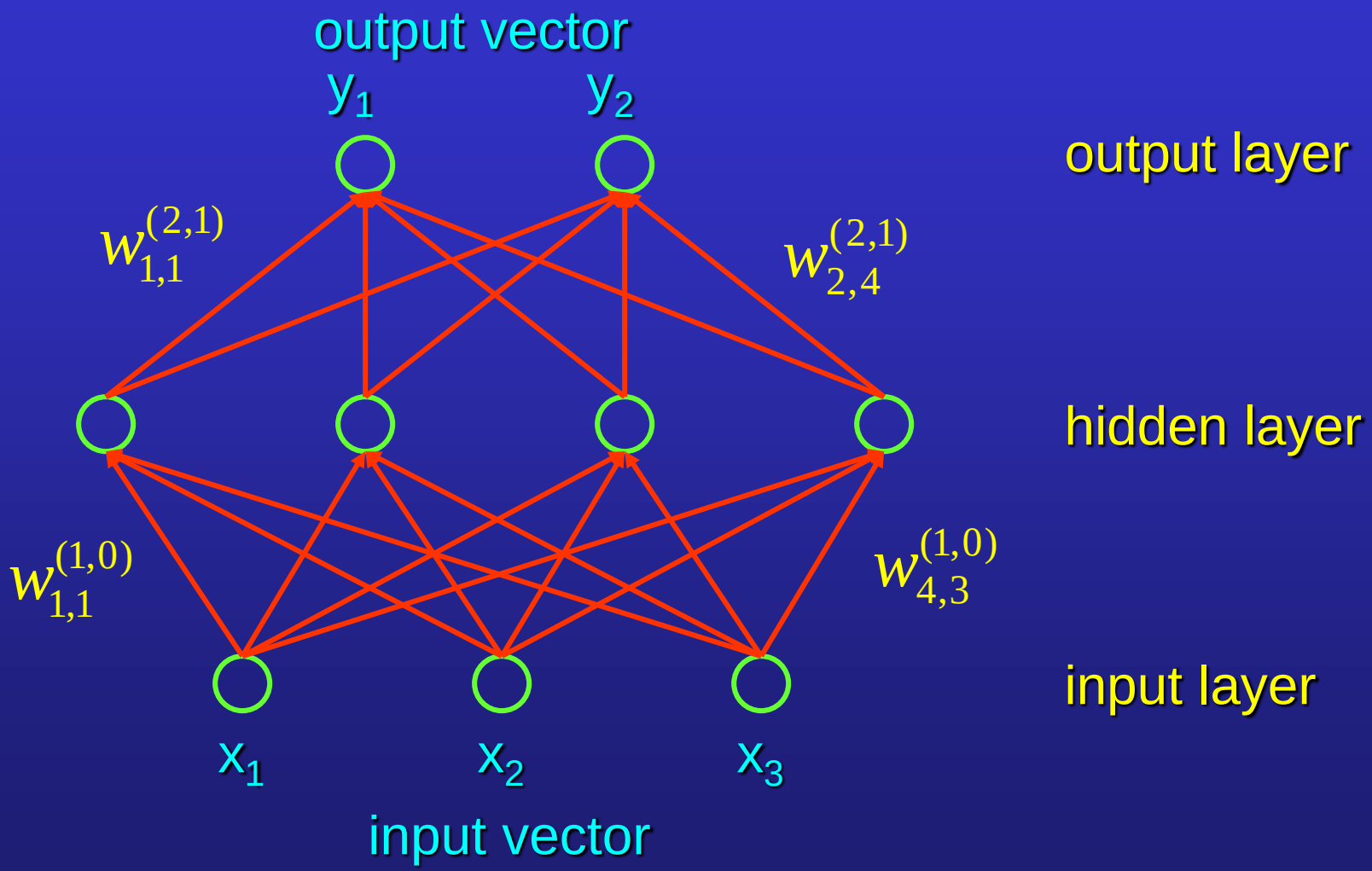
$$\theta = \{w_1, w_2\}$$

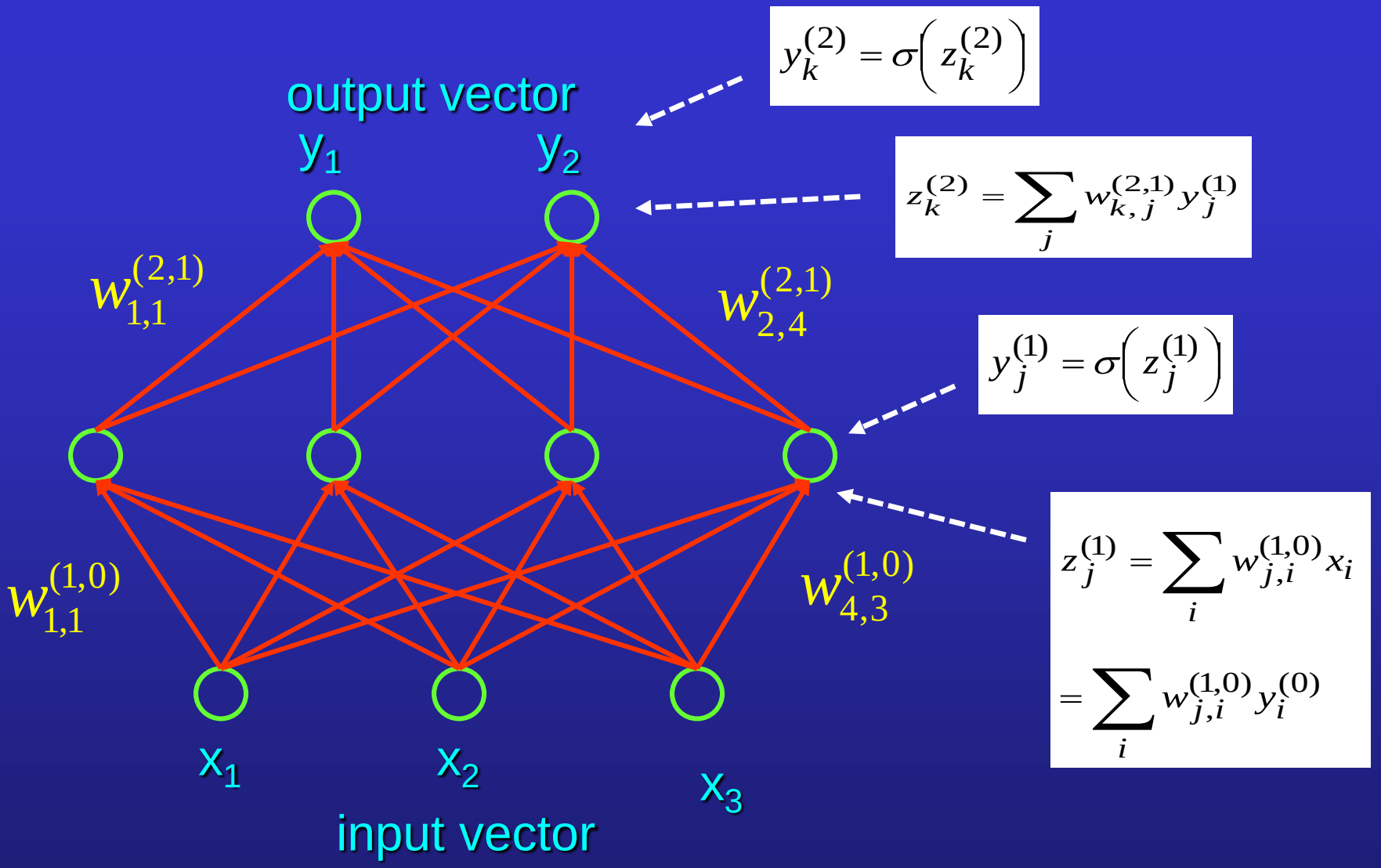
Training Neural Networks

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

Terminology

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



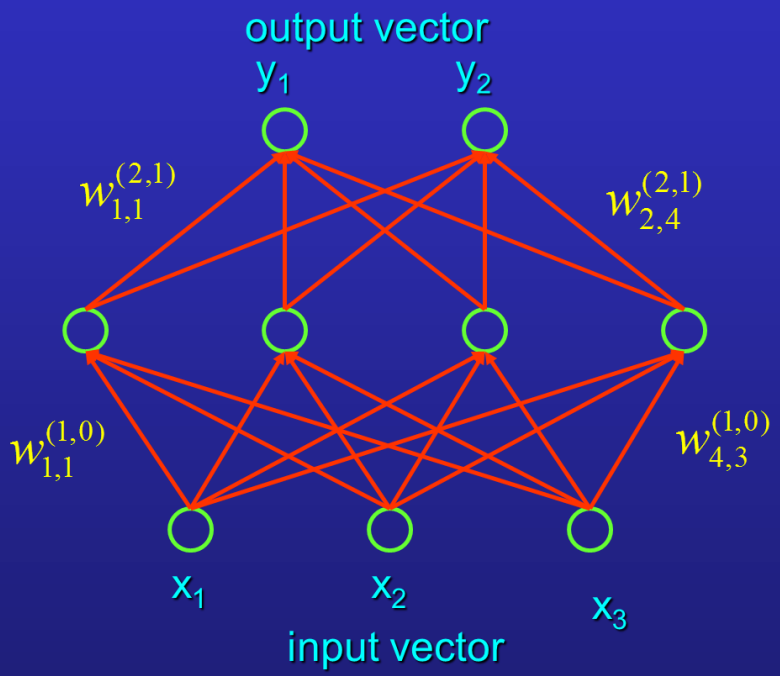

output vector
y_1
y_2
$w_{1,1}^{(2,1)}$
$w_{2,4}^{(2,1)}$
$w_{1,1}^{(1,0)}$
$w_{4,3}^{(1,0)}$
x_1
x_2
x_3
input vector
$y_k^{(2)} = \sigma\left(z_k^{(2)} \right)$
$z_k^{(2)} = \sum_j w_{k,j}^{(2,1)} y_j^{(1)}$
$y_j^{(1)} = \sigma\left(z_j^{(1)} \right)$
$z_j^{(1)} = \sum_i w_{j,i}^{(1,0)} x_i$
$= \sum_i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

Backpropagation

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

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

$$\text{E}(\theta) = \sum \left(y_k - y_k^{(2)} \right)^2 \quad y_k : \text{is the desired output}$$



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

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

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

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

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

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

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

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

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

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

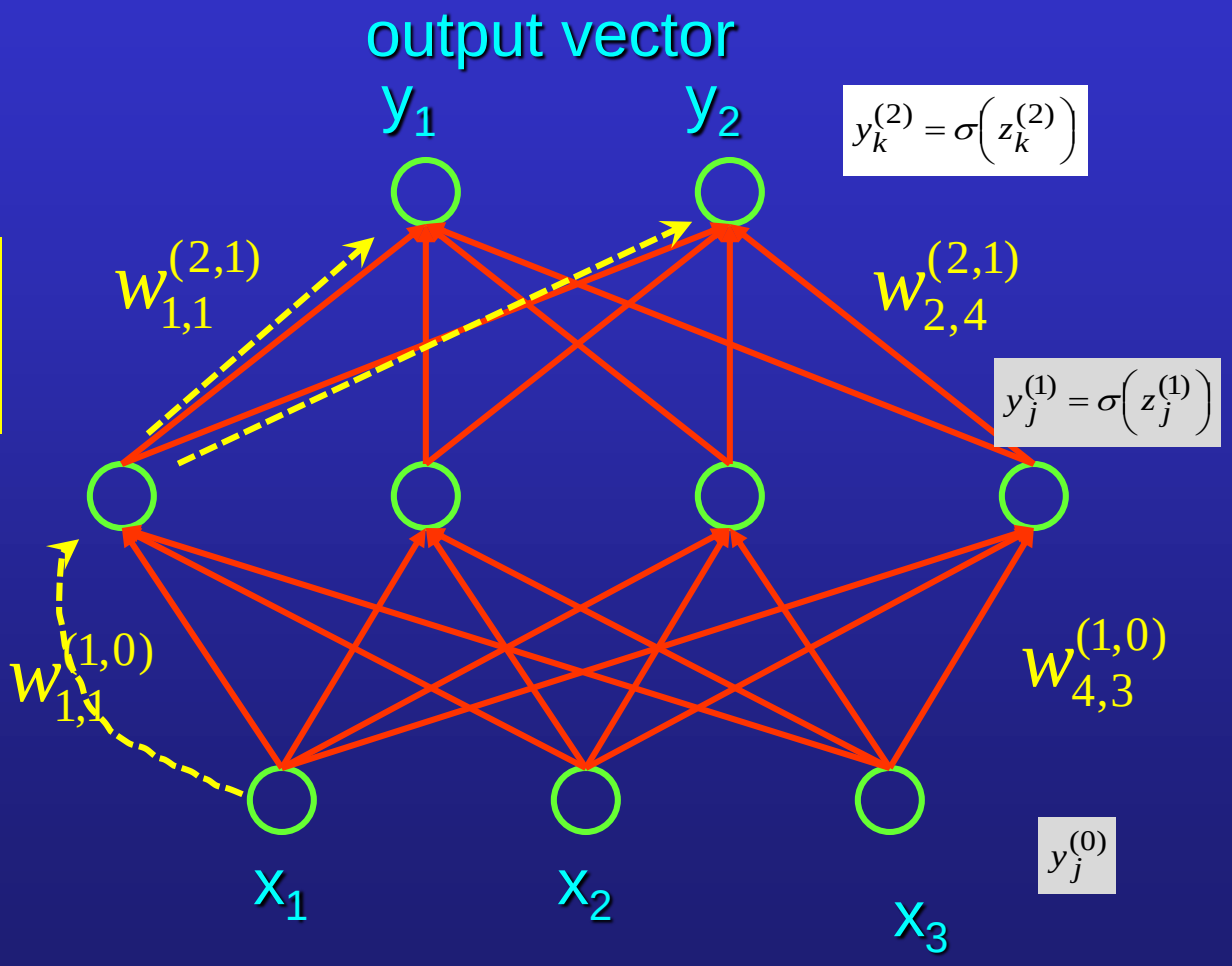
$$E(\theta) = \sum \left(y_k - y_k^{(2)} \right)^2$$

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

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

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

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



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

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

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

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

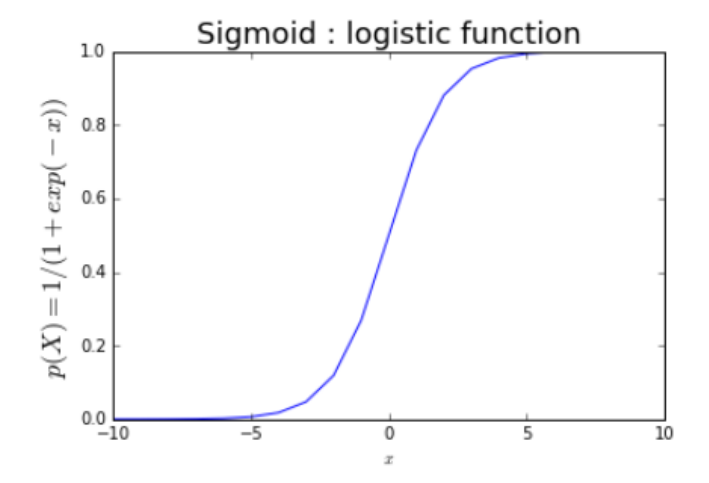
Sigmoidal Neurons

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

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

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

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

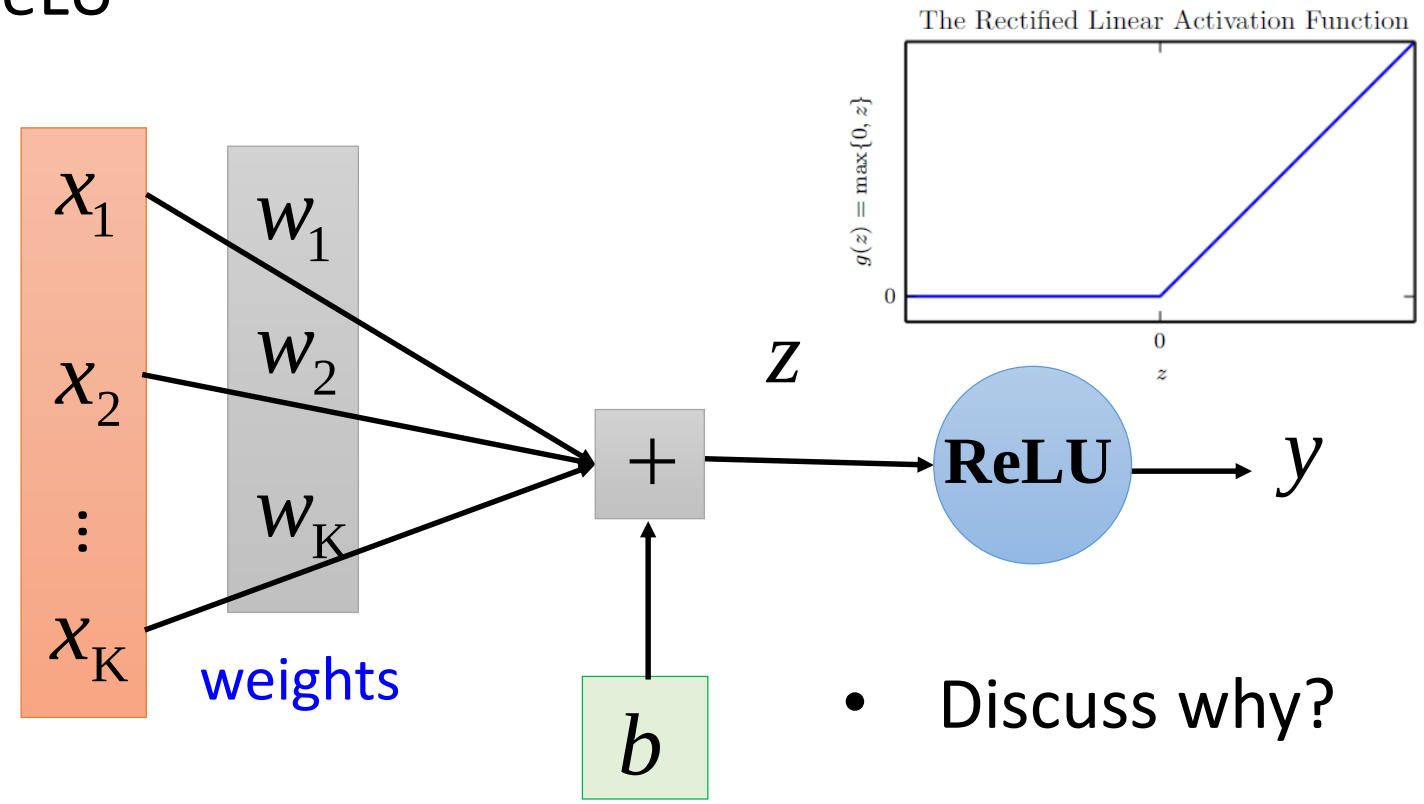


$$\delta_k = (y_k - y_k^{(2)})\sigma'\left(z_k^{(2)}\right)$$

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

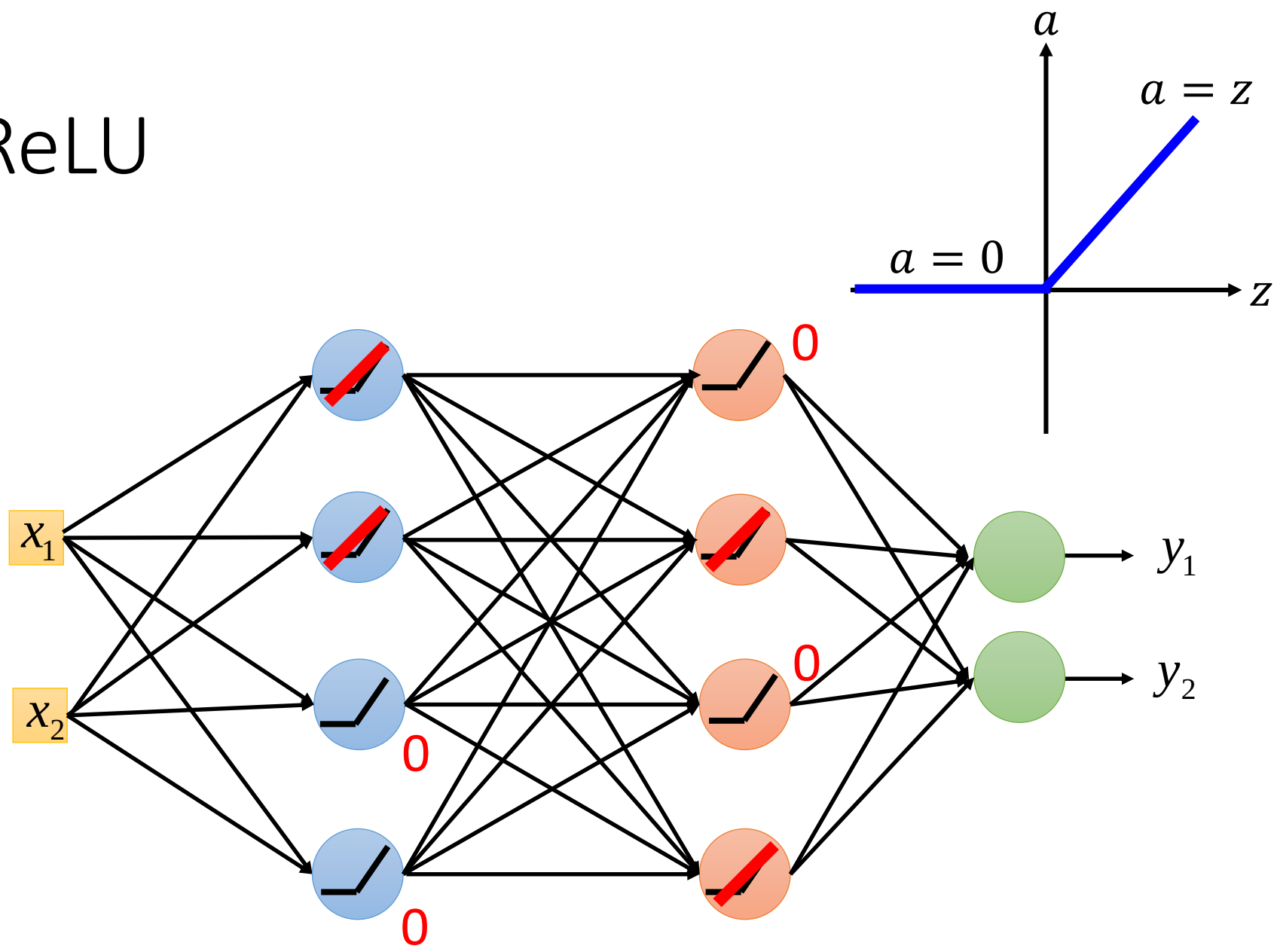
Activation Functions

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

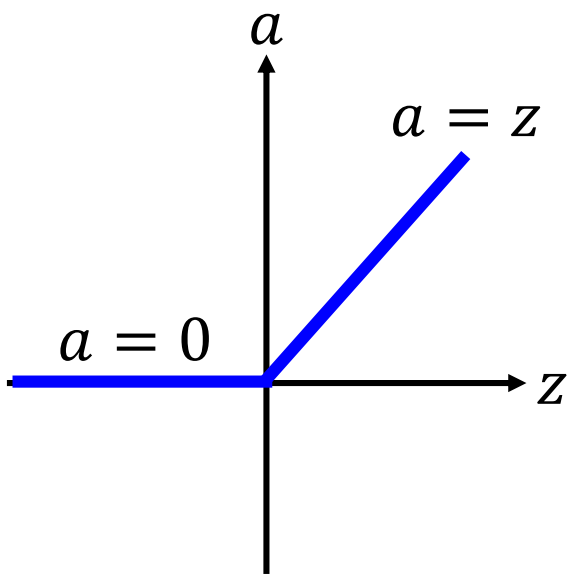


- Discuss why?

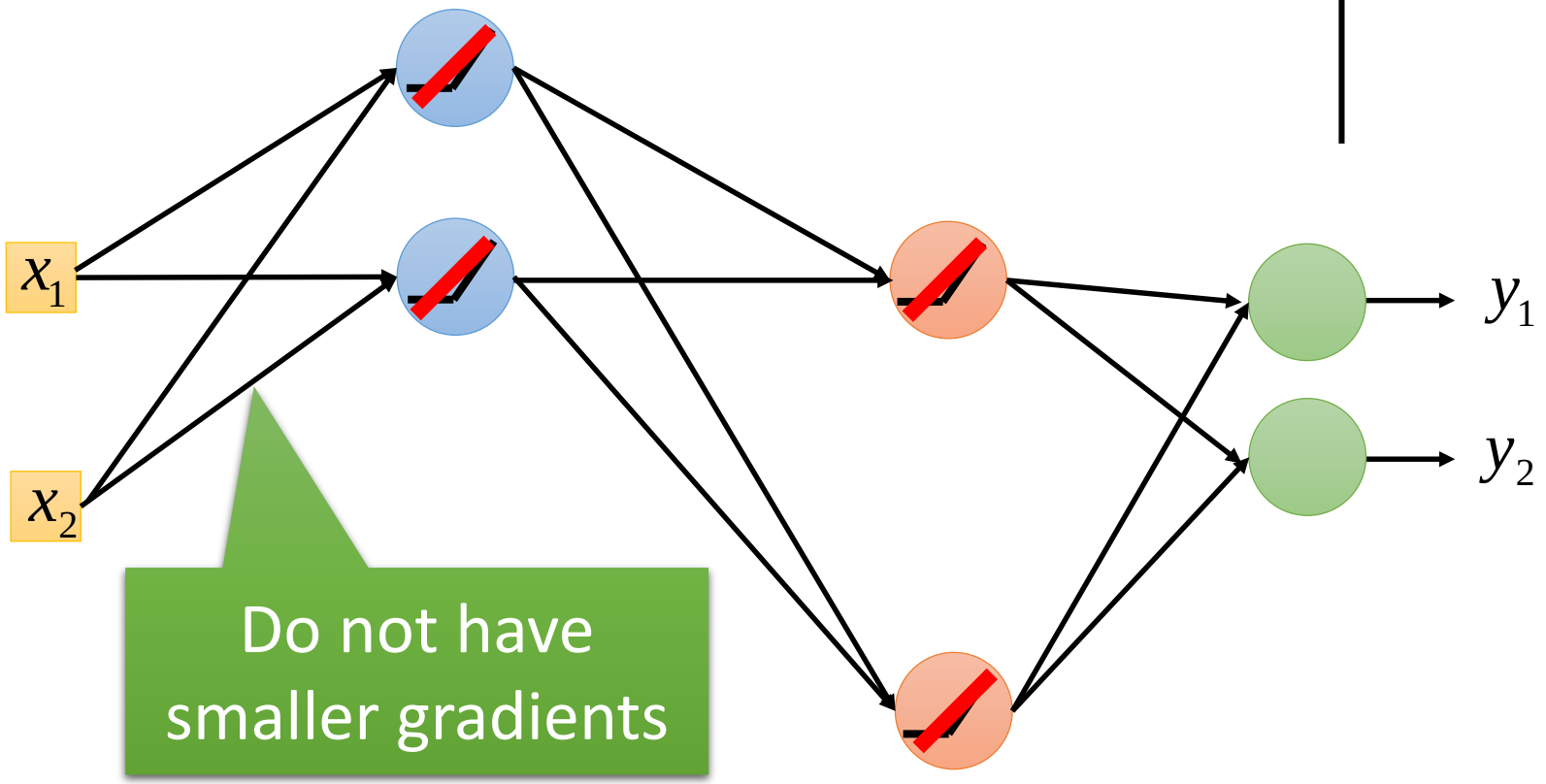
ReLU



ReLU



A Thinner linear network

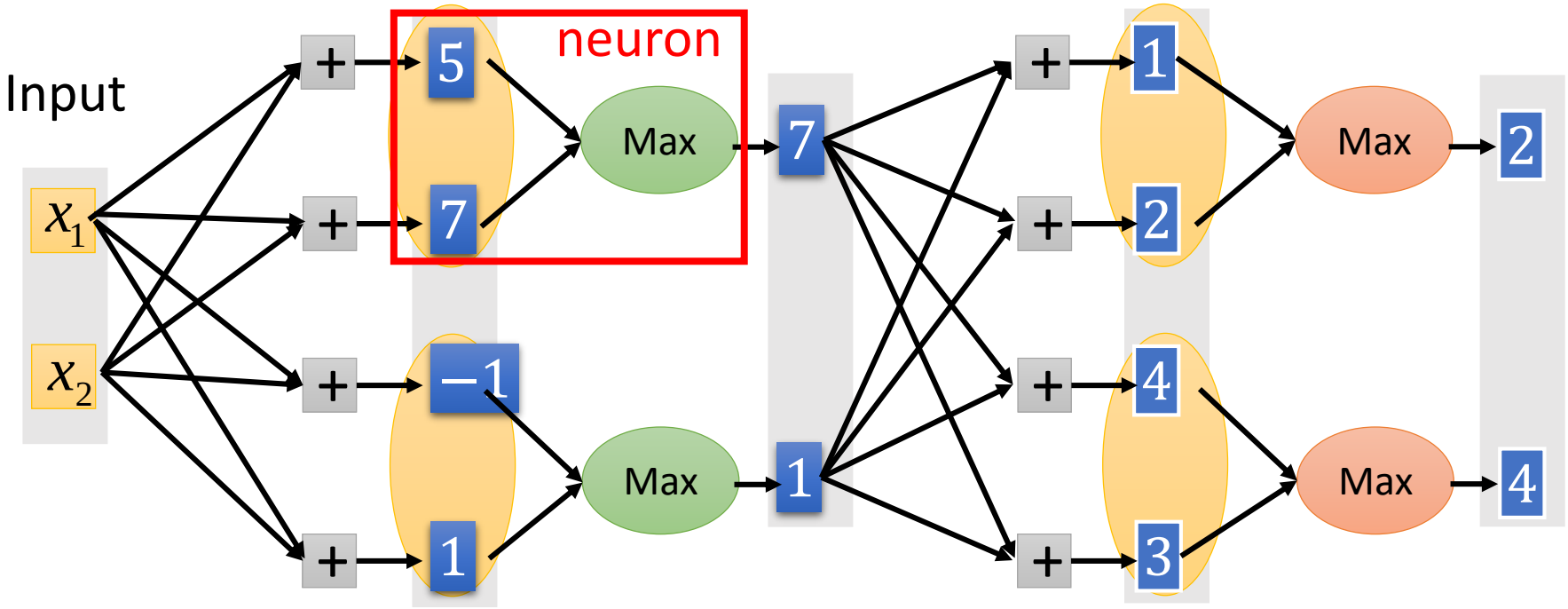


Do not have smaller gradients

Maxout

ReLU is a special cases of Maxout

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



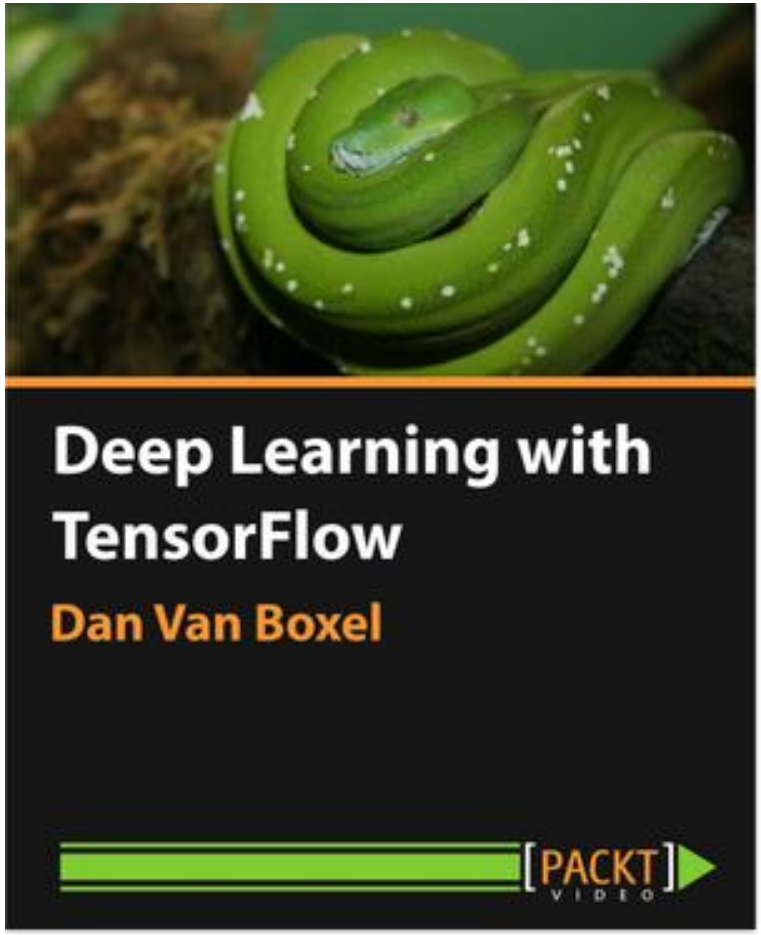
You can have more than 2 elements in a group.

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]

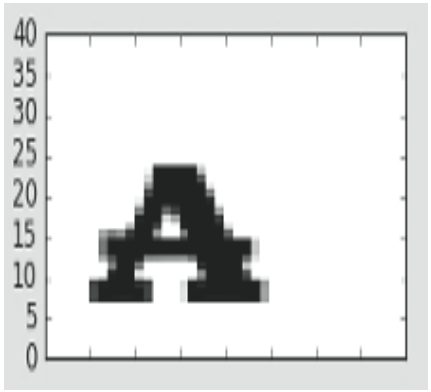
TensorFlow

Font type Recognition



Font type Recognition

Input



36 x 36 = 1296

reShape

x_1

x_2

x_{1296}

Output

0.1	is 1
0.7	is 2
0.2	is N

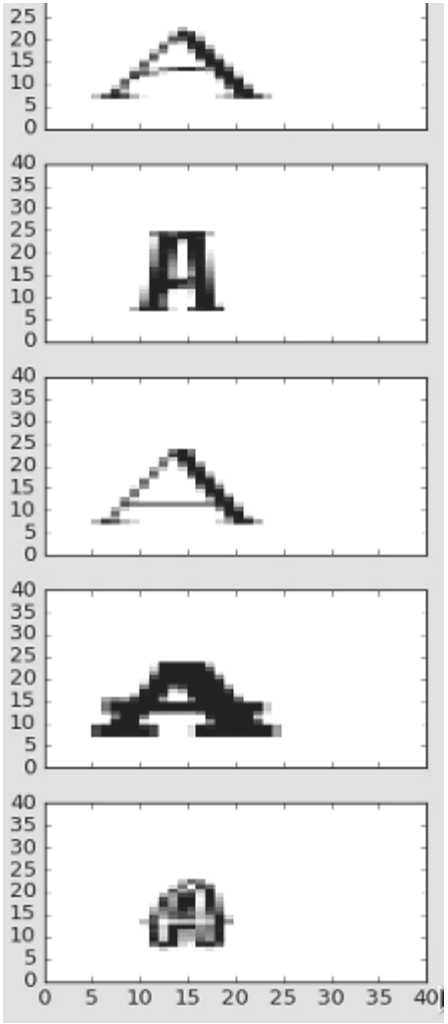
Font type is "2"

0
1
0

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

OHE
One-Hot-Encoding

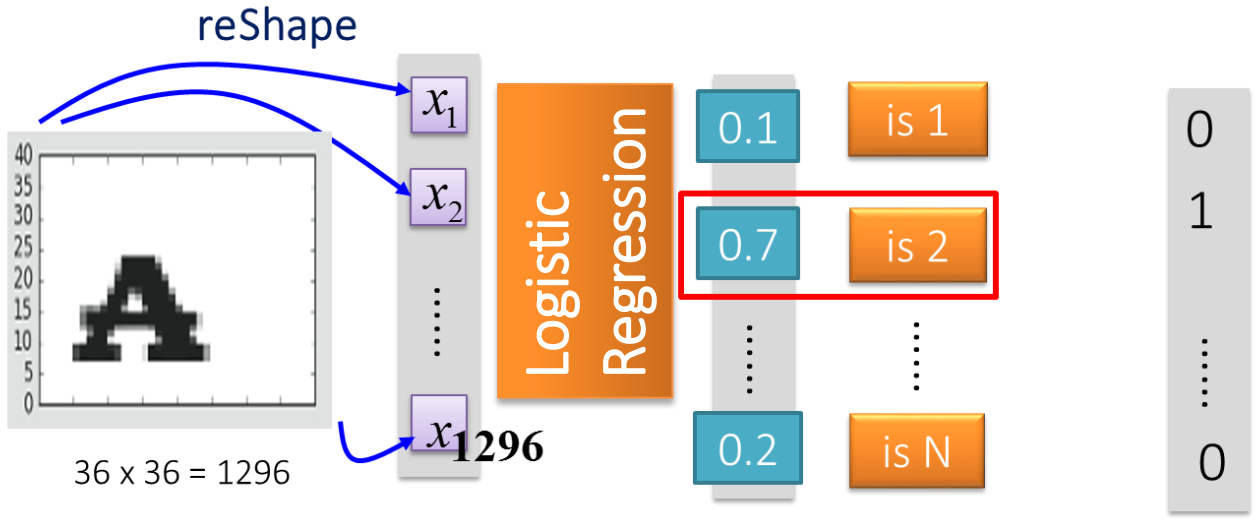
One-hot-encoding (OHE)



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_2018.ipynb



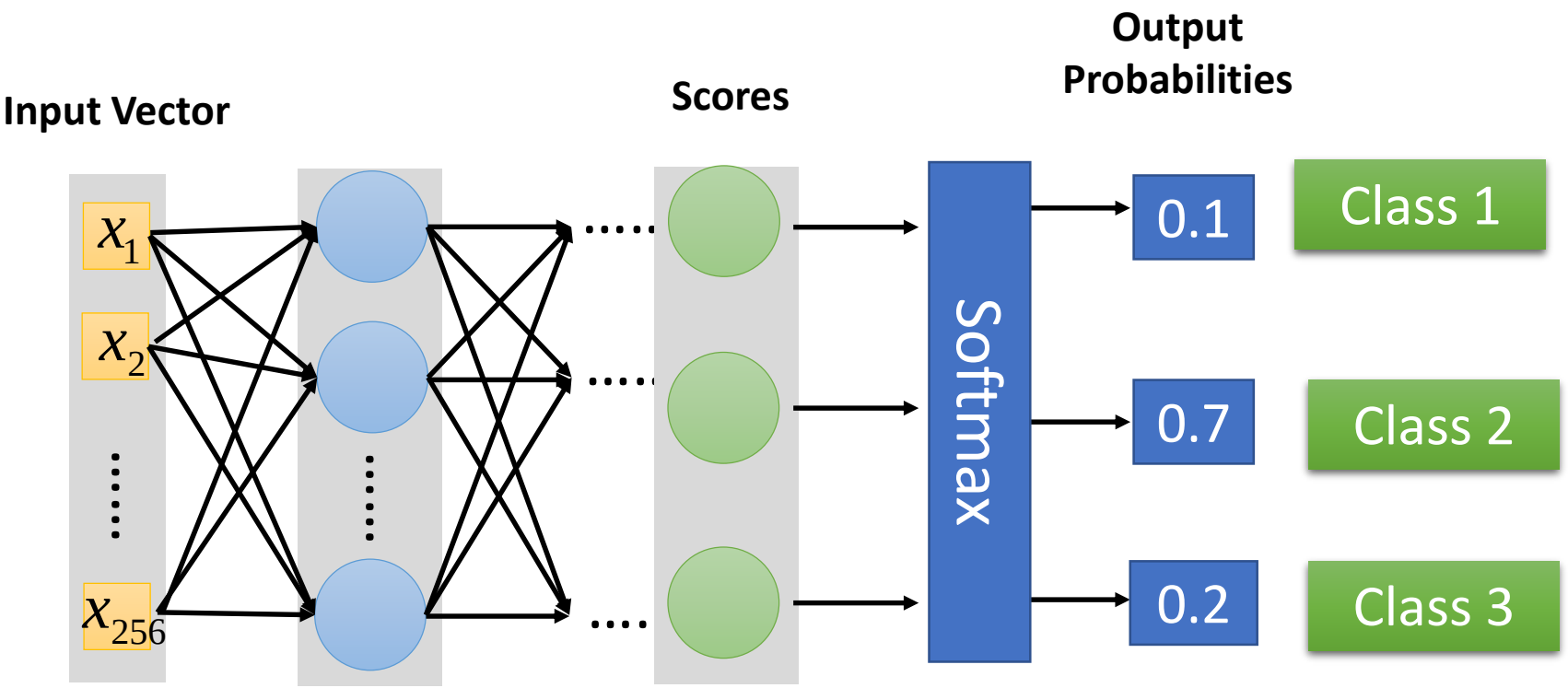
Elements [0:255]

White pixel → 0

Black pixel → 255

Softmax layer: as the output layer

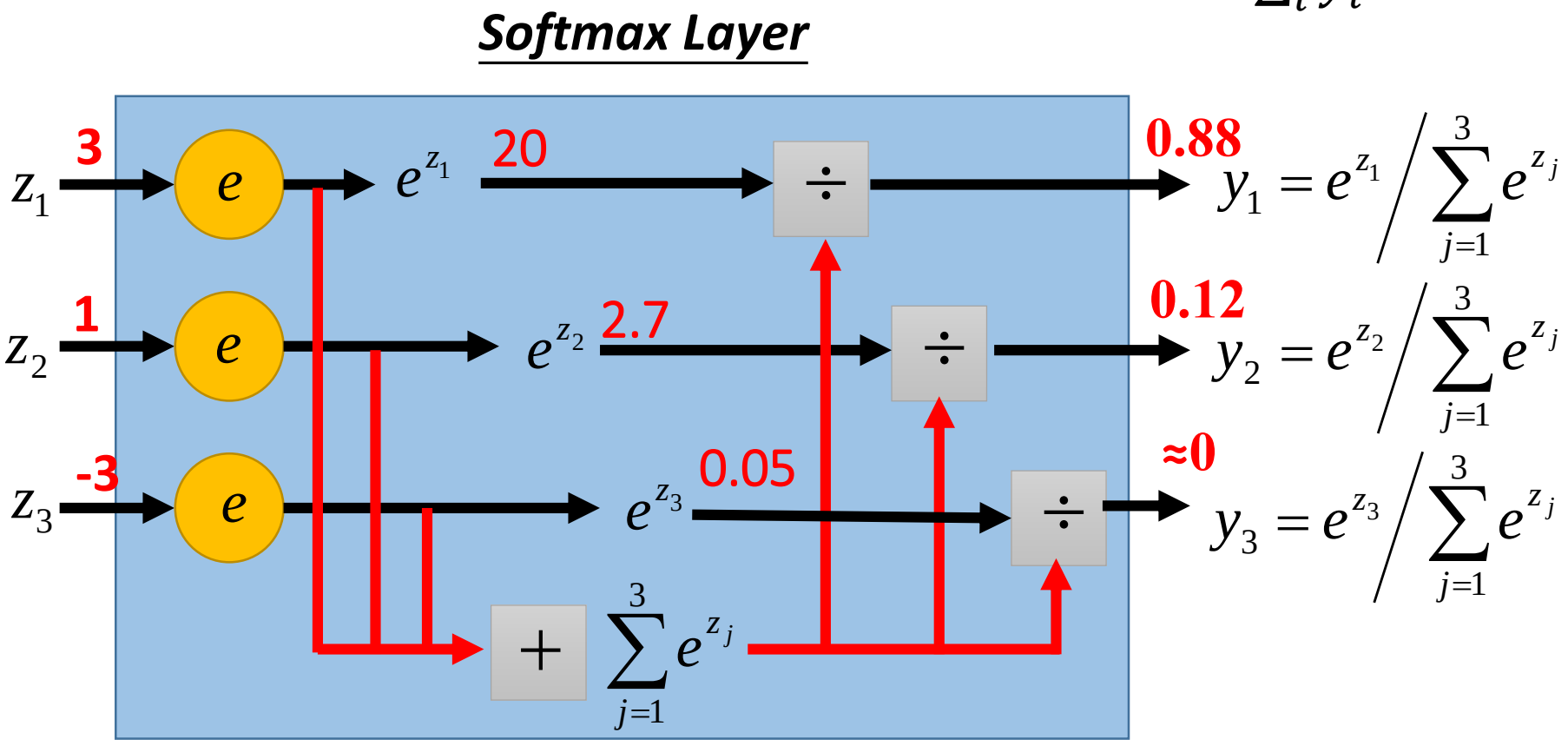
Three-classes example



Softmax Layer

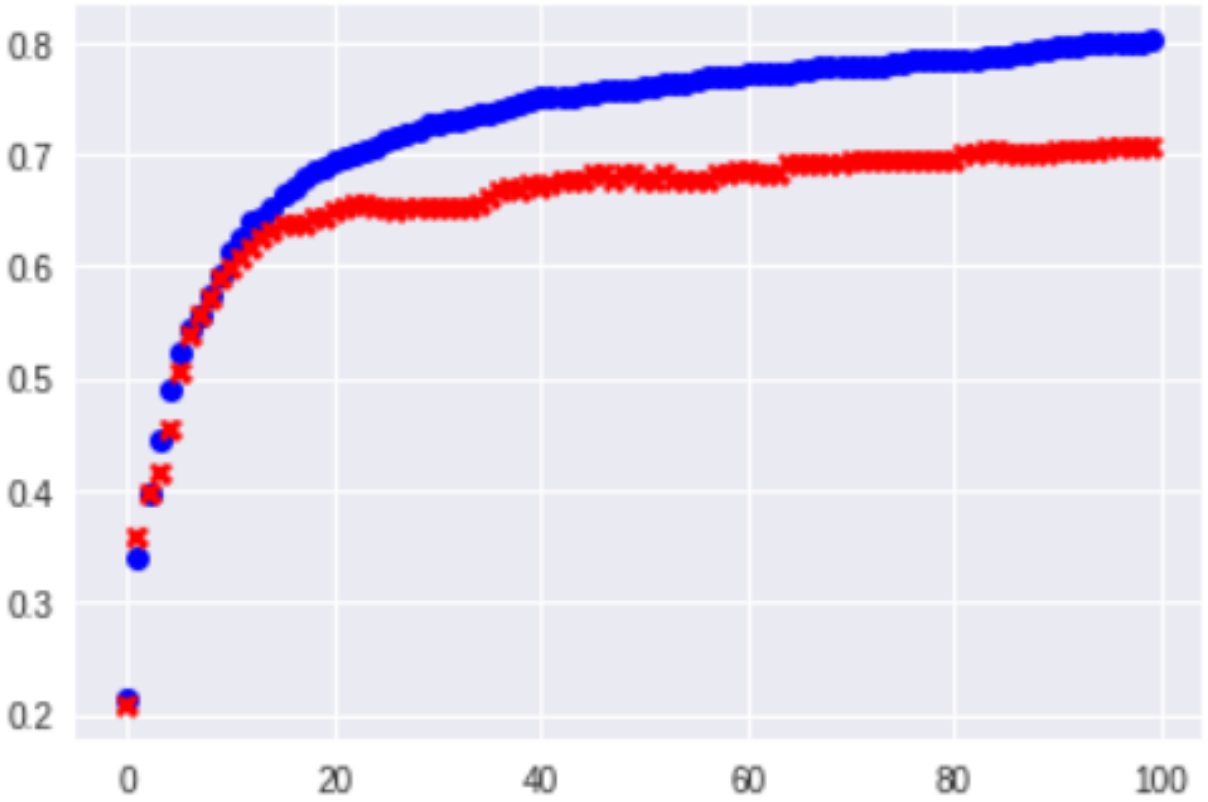
Probability:

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



Logistic Regression

```
Train Accuracy:  0.8
Test_Accuracy:  0.71
[<matplotlib.lines.Line2D at 0x7fd037dcf6a0>]
```



Acc (Train=blue; test=red)

Epoch/10 no.

Now Let's use Feed Foorward Architecture

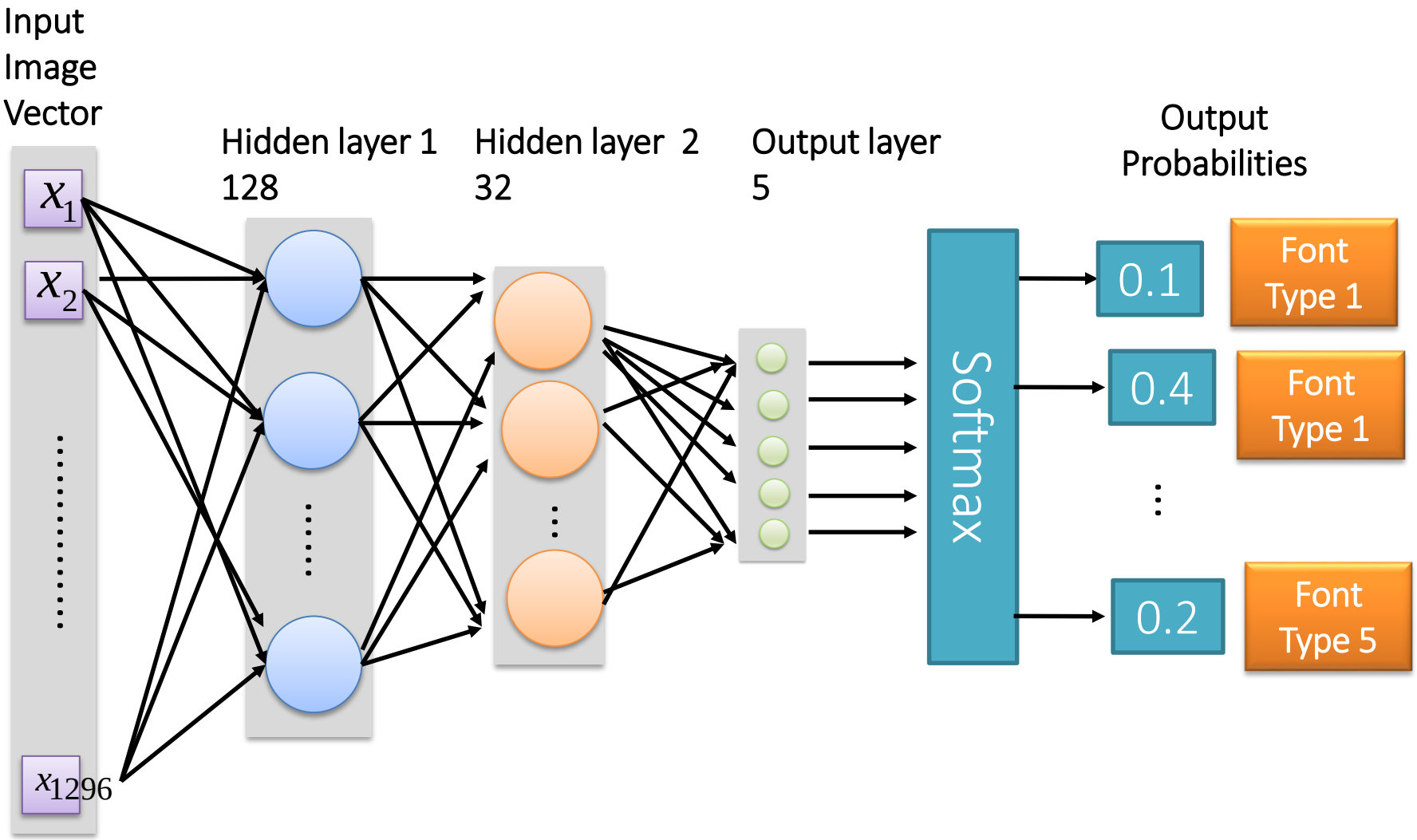
You can follow:

MSTC_FontReco_FeedForward_2018.ipynb

Available at:

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

MSTC_FontReco_FeedForward_2018.ipynb



MSTC_FontReco_FeedForward.ipynb

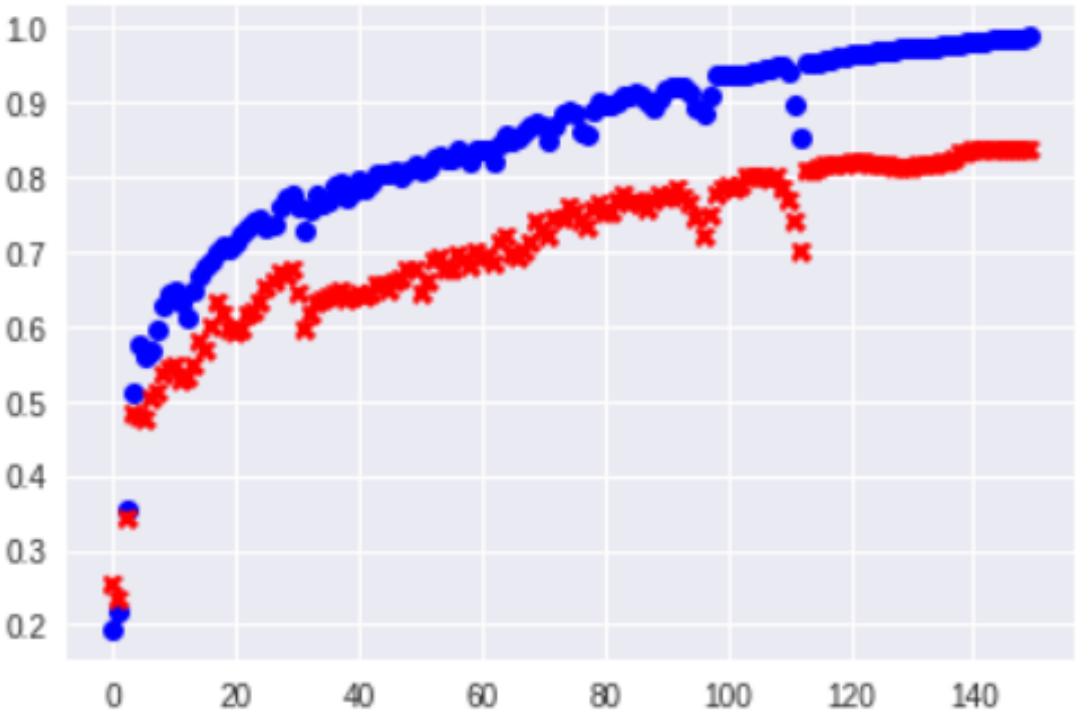
```
# These will be inputs…. pixels, flattened
x = tf.placeholder("float", [None, 1296])
## Known labels
y_ = tf.placeholder("float", [None,5])

# 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
init=tf.initialize_all_variables()

# Define model
y = tf.nn.softmax(tf.matmul(h2,W3) + b3)

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

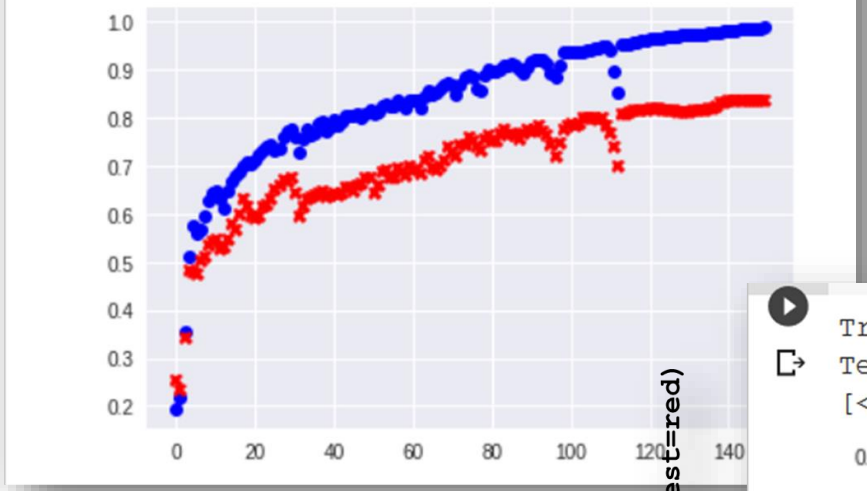
Feed Forward

```
Train Accuracy:  0.99
Test_Accuracy:  0.84
[<matplotlib.lines.Line2D at 0x7efff740dc18>]
```



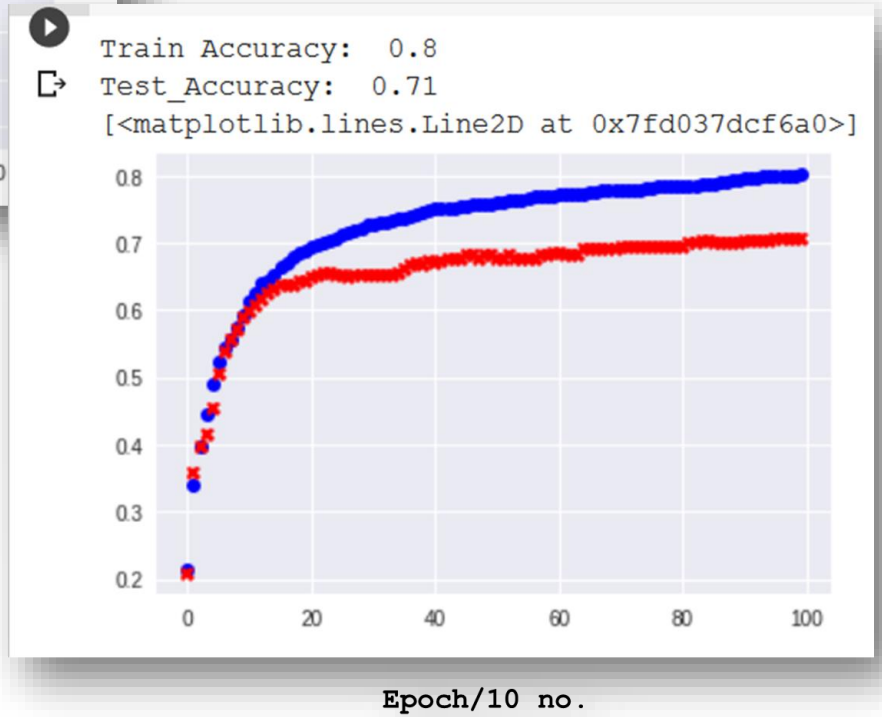
Feed Forward

```
Train Accuracy:  0.99
Test_Accuracy:  0.84
[<matplotlib.lines.Line2D at 0x7efff740dc18>]
```

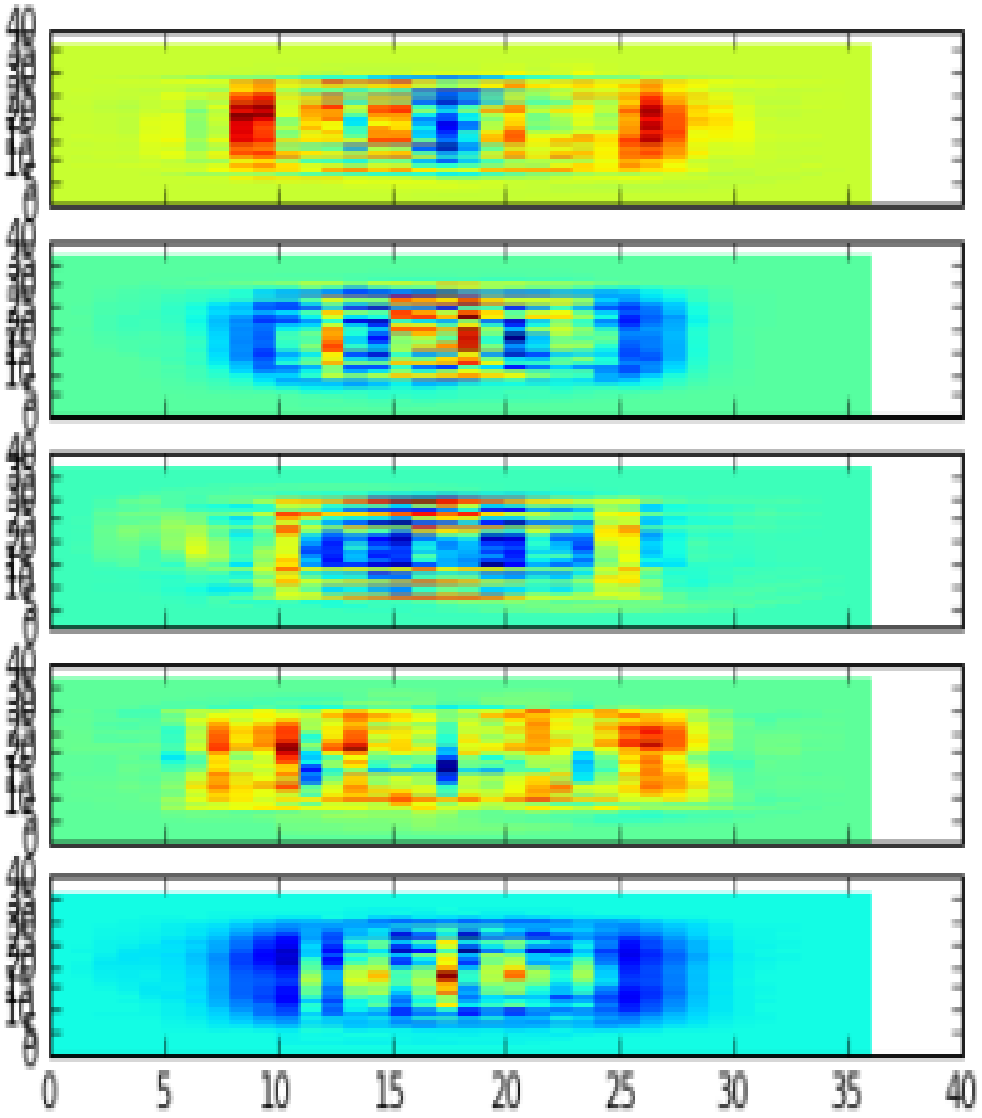


Logistic Regression

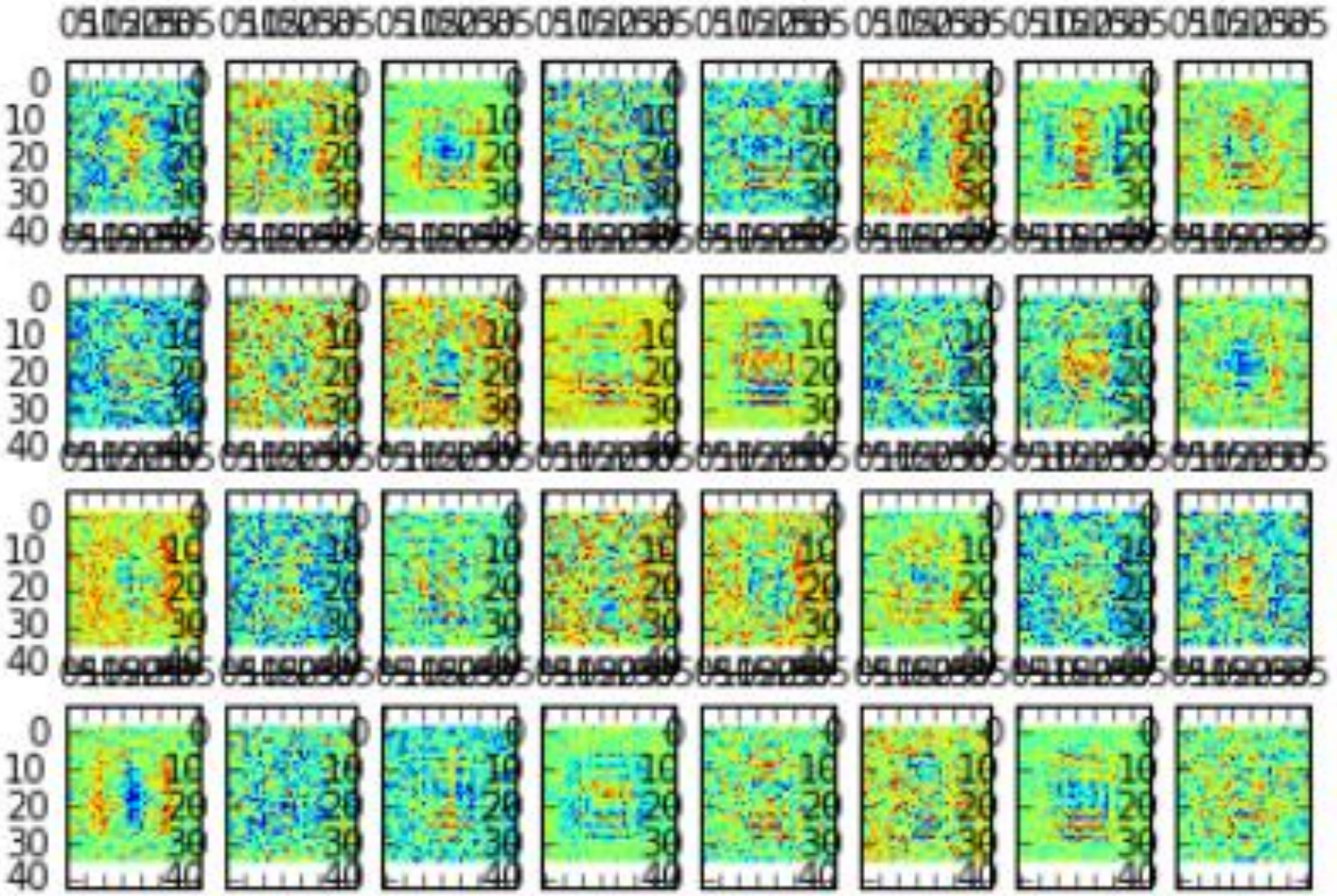
```
Train Accuracy:  0.8
Test_Accuracy:  0.71
[<matplotlib.lines.Line2D at 0x7fd037dcf6a0>]
```



Logistic Regression



Feed Forward



Keras: The Python Deep Learning library

Keras

Guiding principles

- User friendliness
- Modularity
- Easy extensibility
- Work with Python

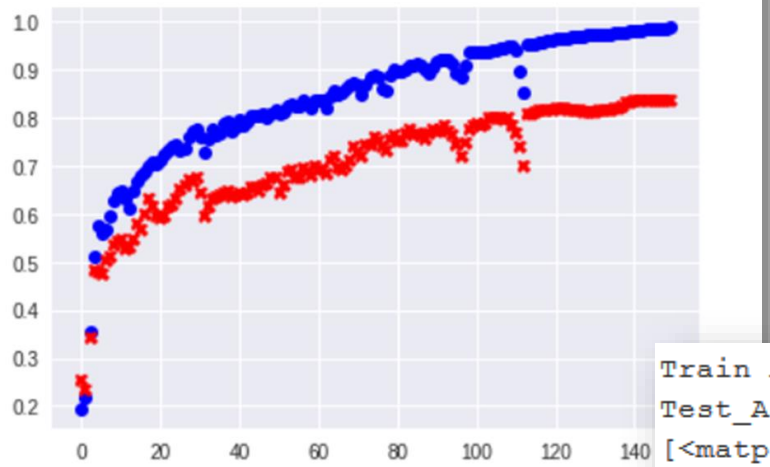
Activities:

- Introduction to Keras: Font Recognition using:
  MSTC_Keras_FontReco_FeedFordward.ipynb

Feed Forward

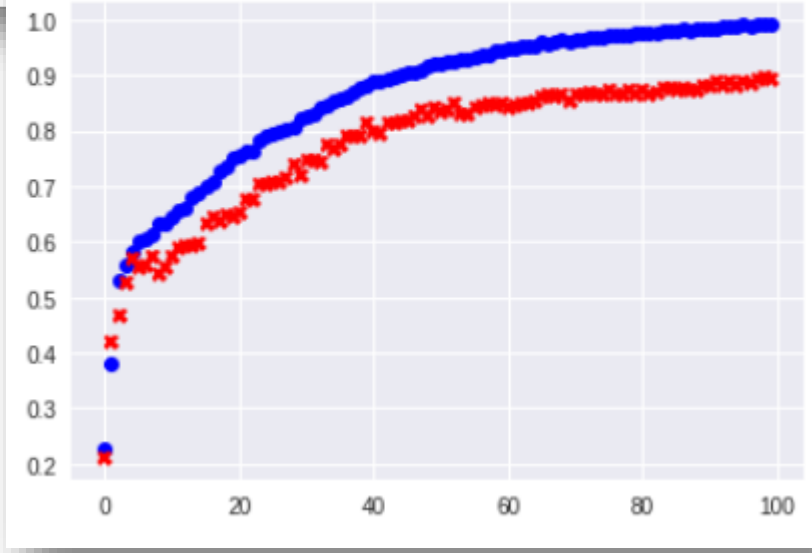
Acc (Train=blue; test=red)

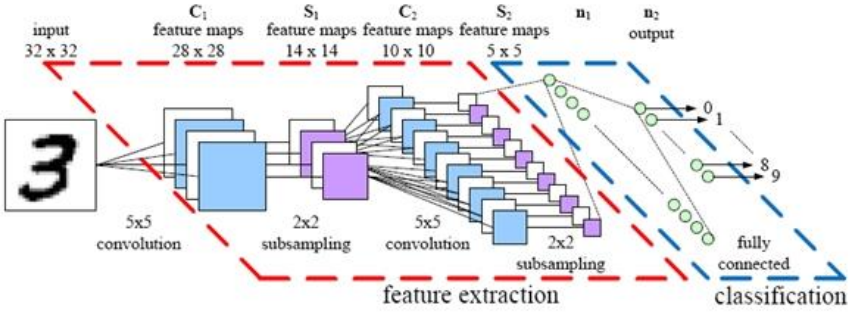
```
Train Accuracy:  0.99
Test_Accuracy:  0.84
[<matplotlib.lines.Line2D at 0x7efff740dc18>]
```

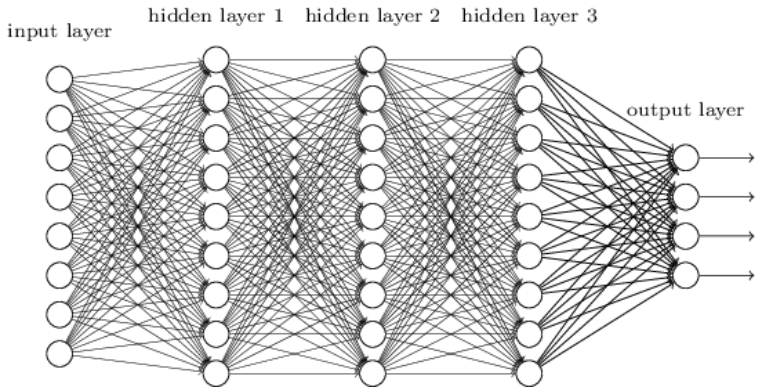


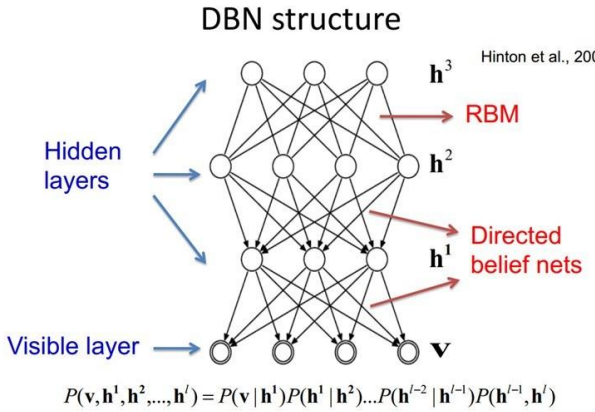
Epoch/10 no.


```
Train Accuracy:  0.99
Test_Accuracy:  0.89
[<matplotlib.lines.Line2D at 0x7fec4831d2b0>]
```

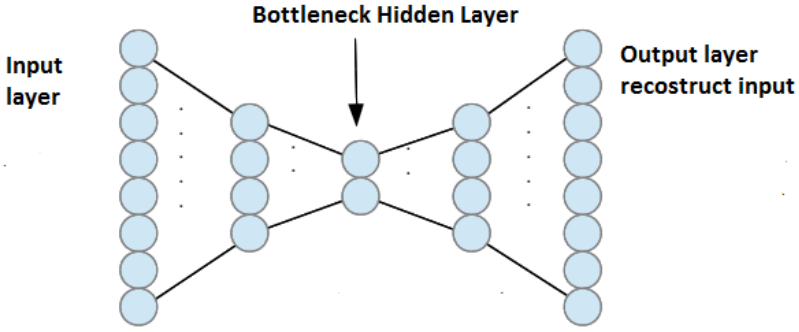


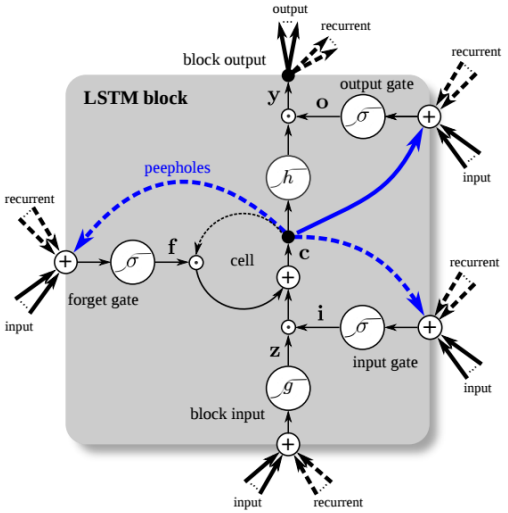


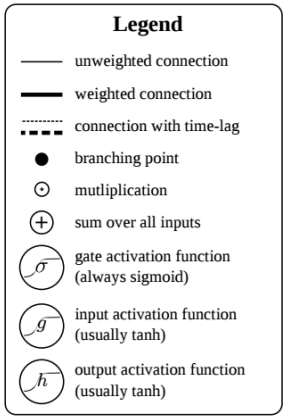




Deep Learning Architectures







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.

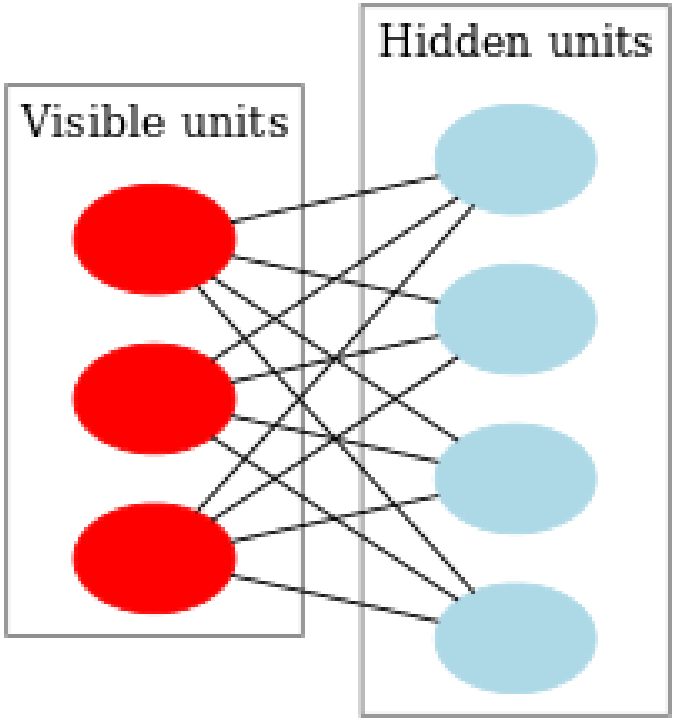
Autoencoders

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.

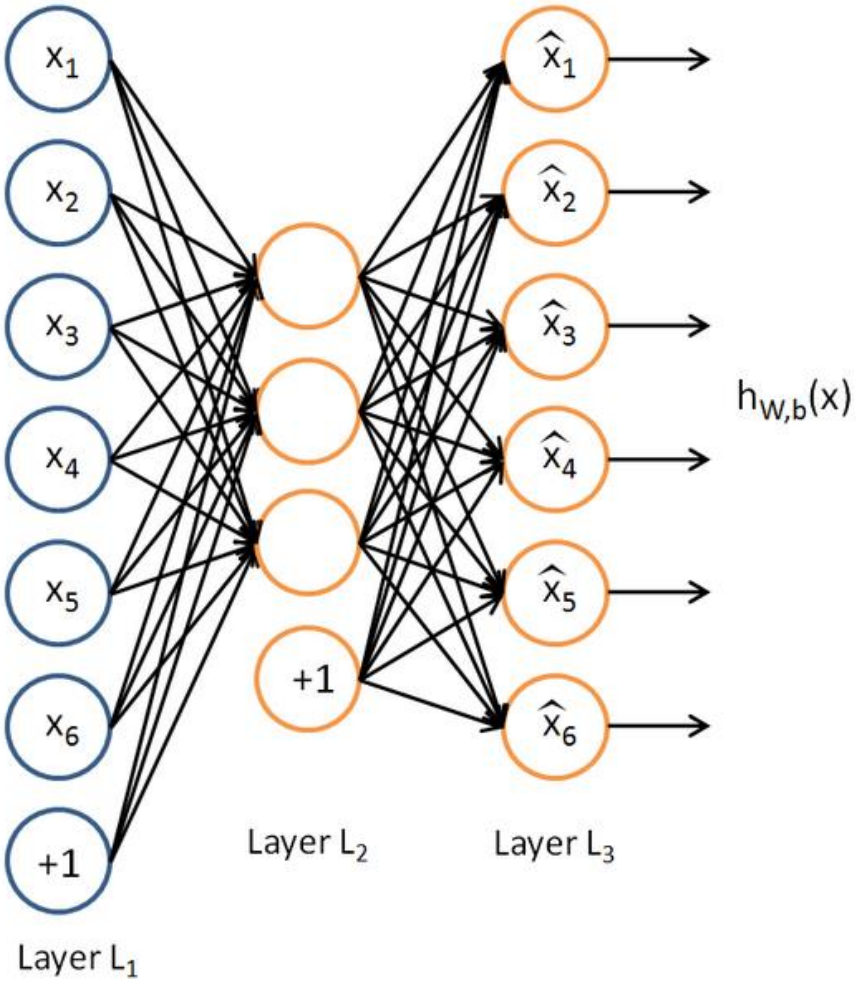
- DBN can be used to generatively pre-train a DNN The learned DBN weights as the initial DNN weights.
  - Back-propagation later for fine tuning of these weights.



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



Supervised:

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

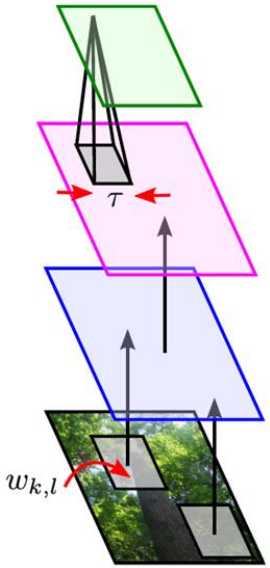
CNN / ConvNets:
Convolutional Neural Network

organizes neurons based on animal's visual cortex system, which allows for learning patterns at both local level and global level.

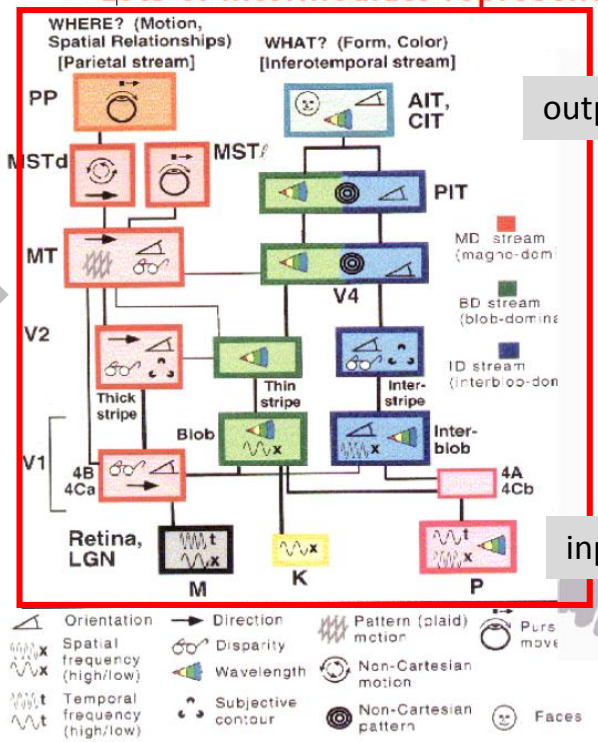
- Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, 86(11):2278-2324, November 1998

The Mammalian Visual Cortex Inspires CNN

Convolutional Neural Net

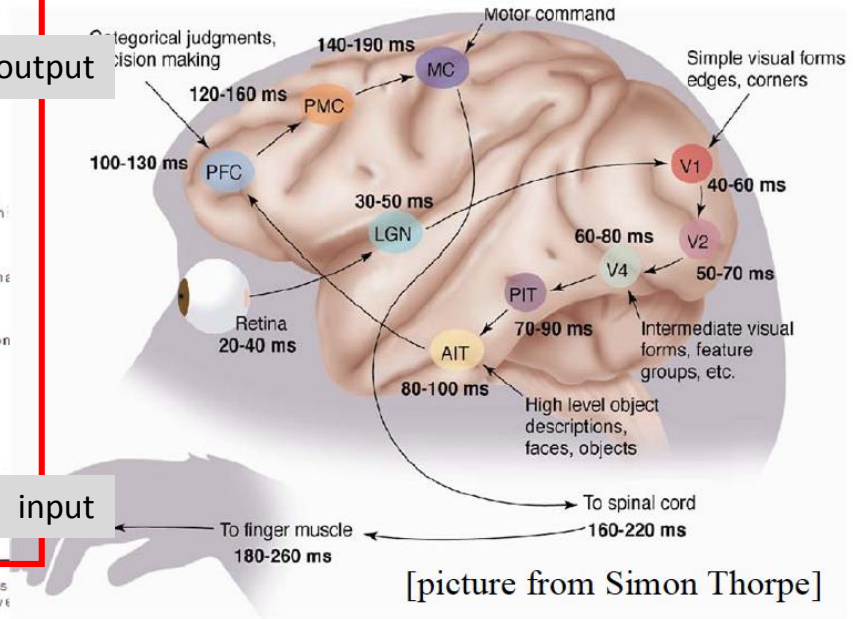


- The ventral (recognition) pathway in the visual cortex has multiple stages
- Retina - LGN - V1 - V2 - V4 - PIT - AIT ....
- Lots of intermediate representations



output

input



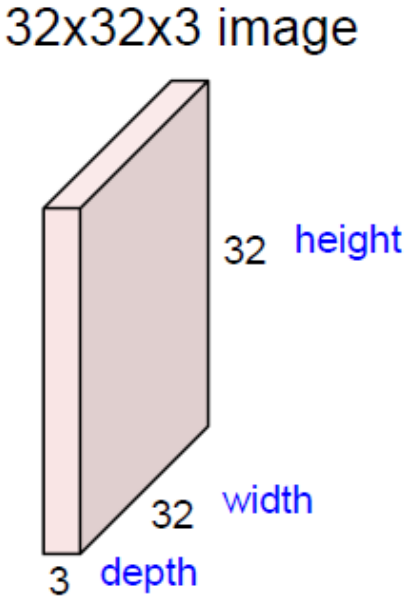
[picture from Simon Thorpe]

[Gallant & Van Essen]

Convolutional Neural Networks

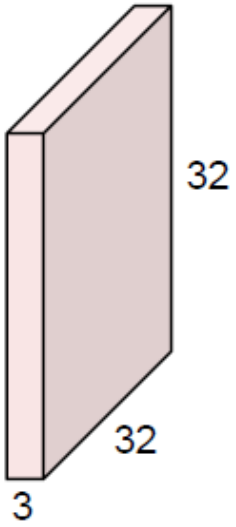
(First without the brain stuff)

Convolution Layer

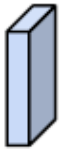


Convolution Layer

32x32x3 image

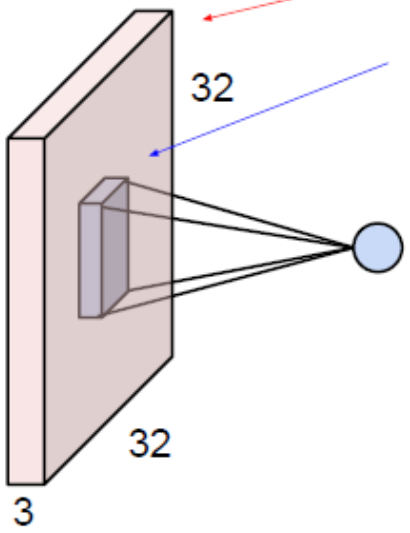


5x5x3 filter



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

Convolution Layer



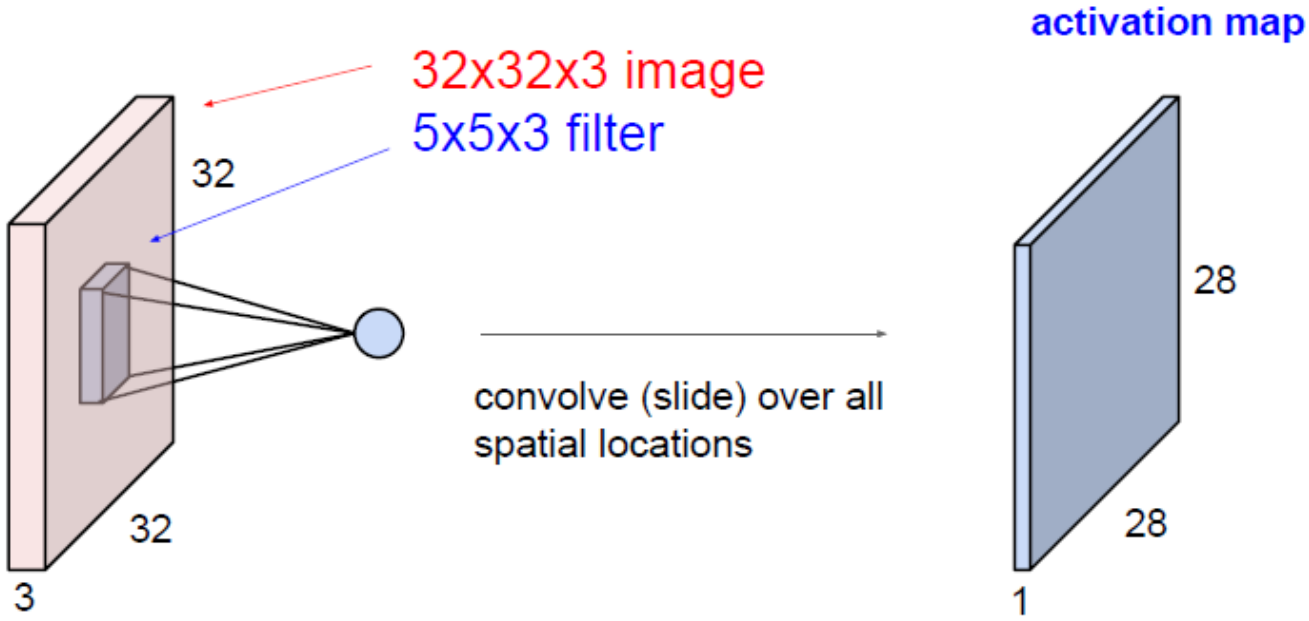
32x32x3 image

5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

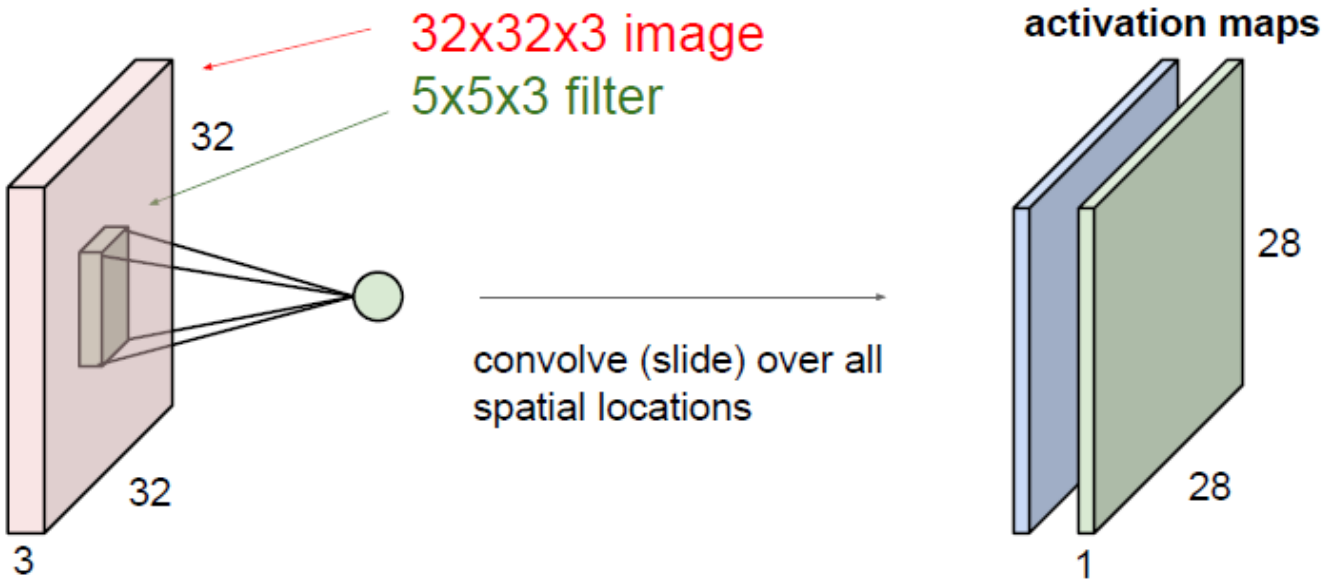
$$w^T x + b$$

Convolution Layer

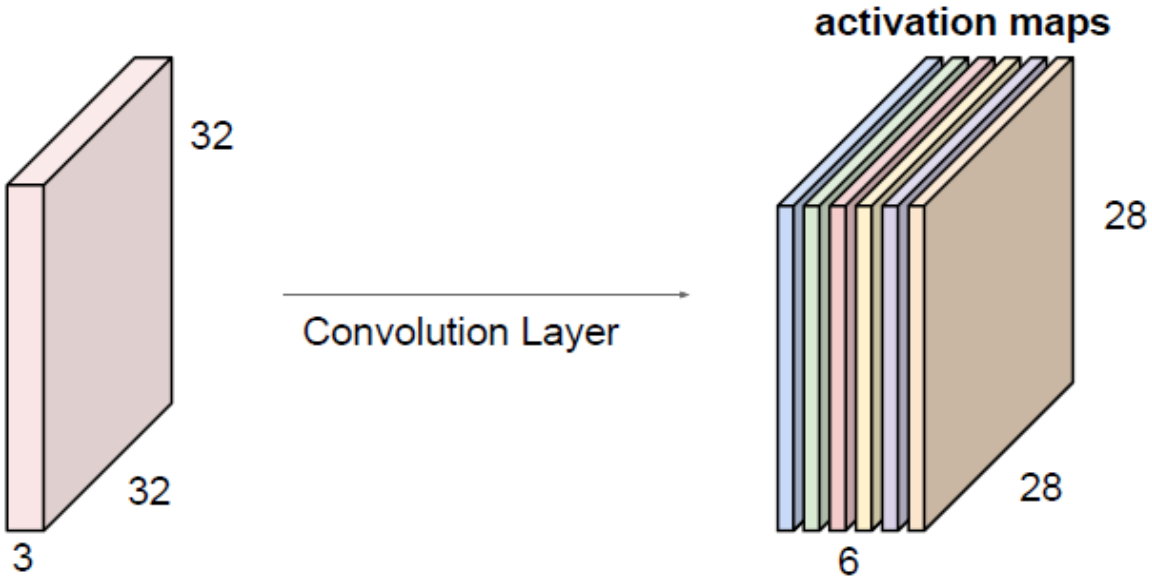


Convolution Layer

consider a second, green filter



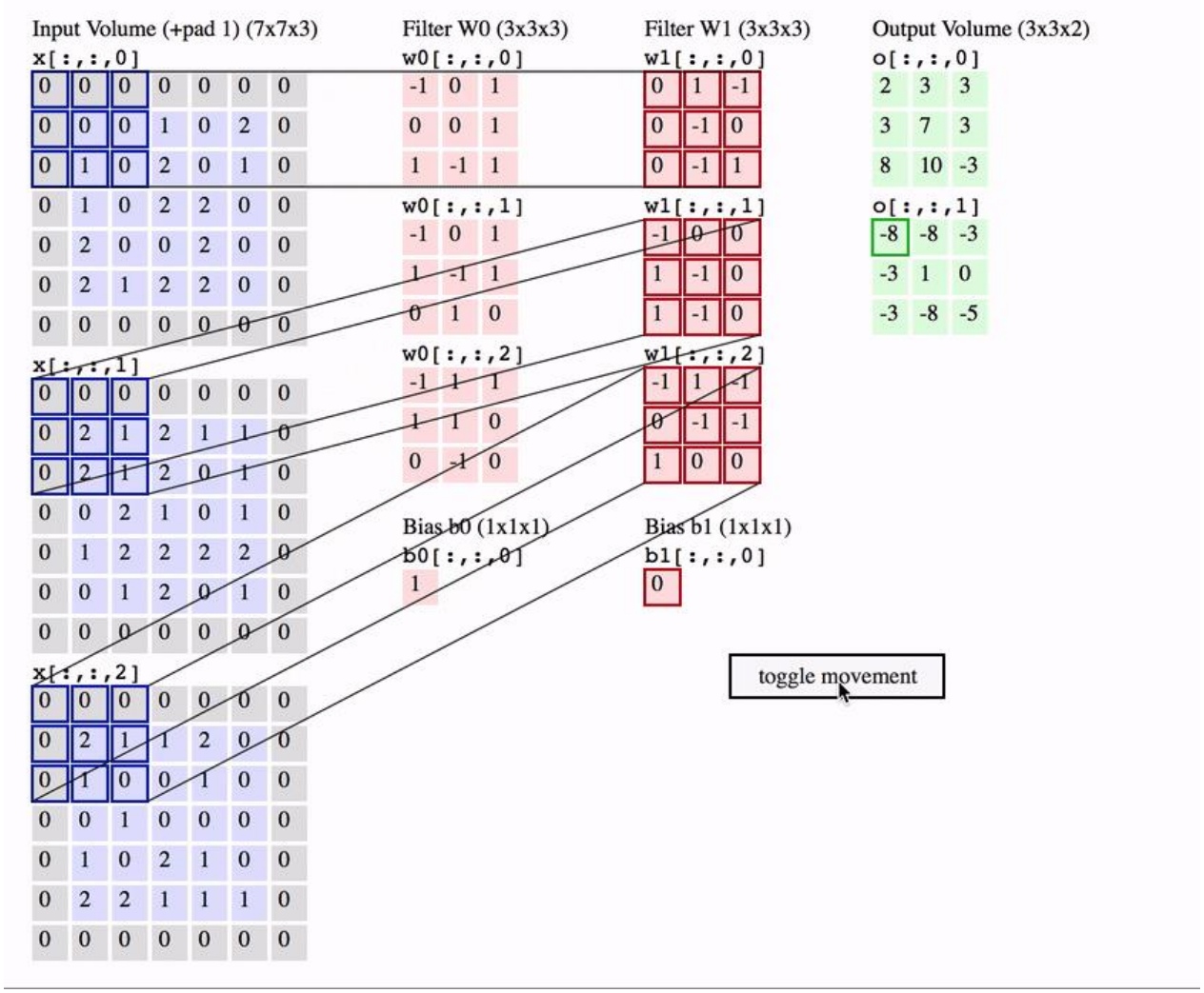
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



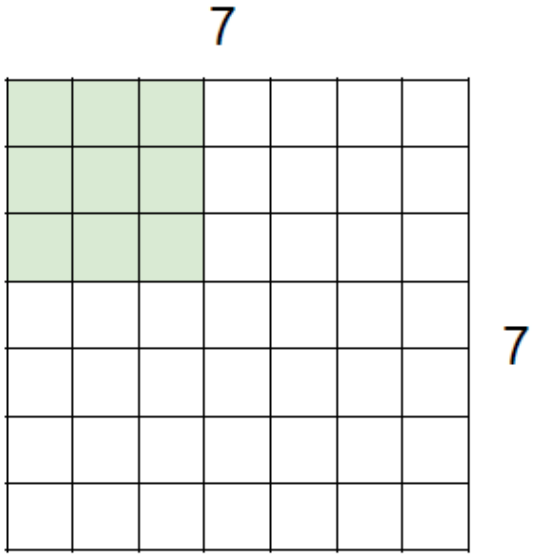
We stack these up to get a "new image" of size 28x28x6!

CNN nice visualization

http://cs231n.github.io/convolutional-networks/

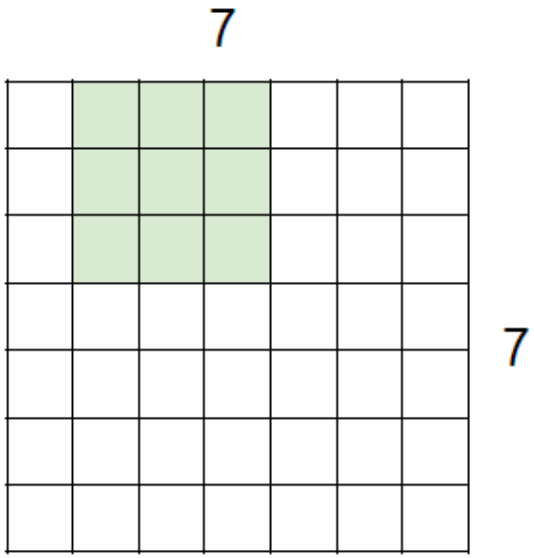


Stride and Padding



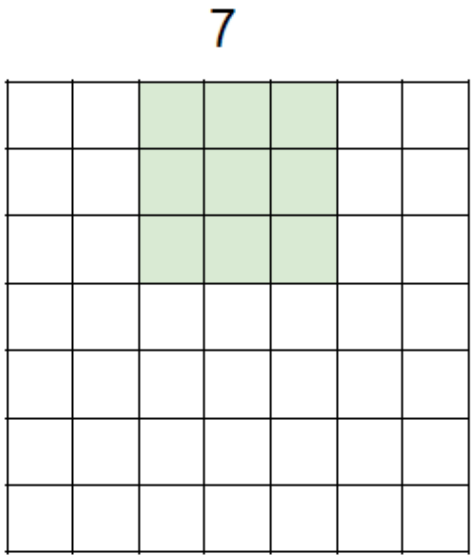
7x7 input (spatially)
assume 3x3 filter

Stride and Padding



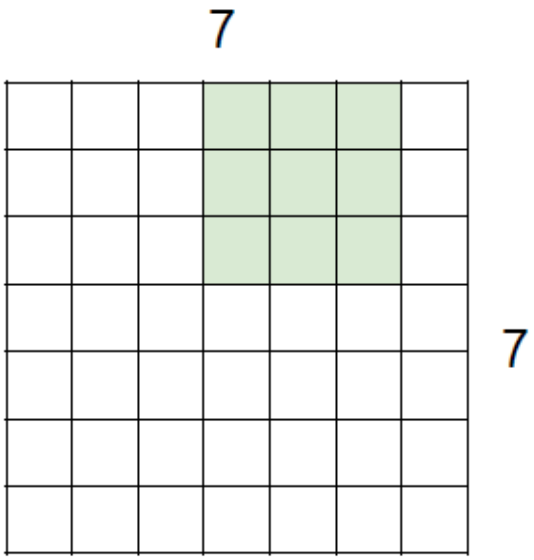
7x7 input (spatially)
assume 3x3 filter

Stride and Padding



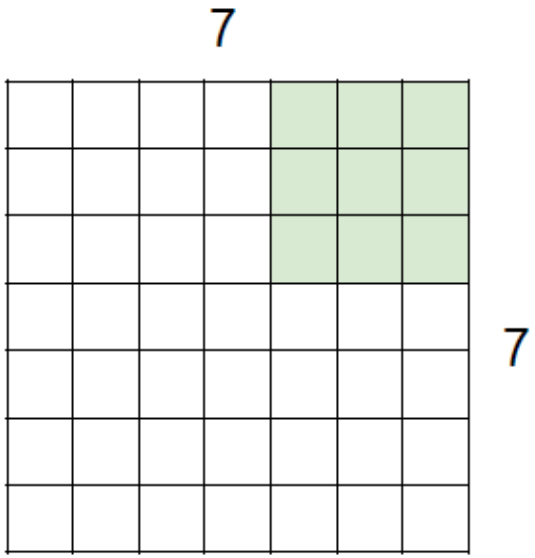
7x7 input (spatially)
assume 3x3 filter

Stride and Padding



7x7 input (spatially)
assume 3x3 filter

Stride and Padding



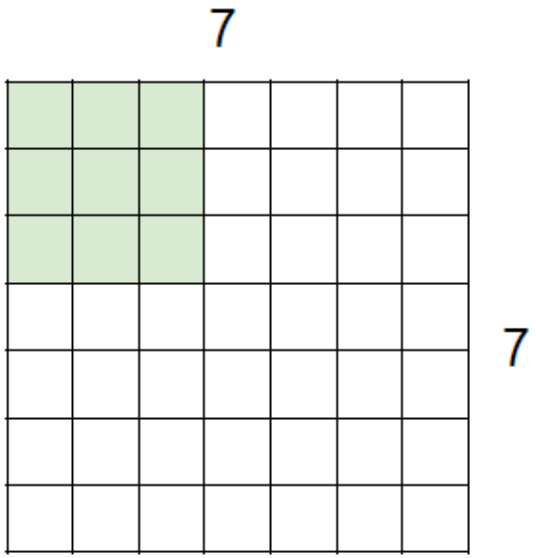
Stride = 1

Padding = 0

7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

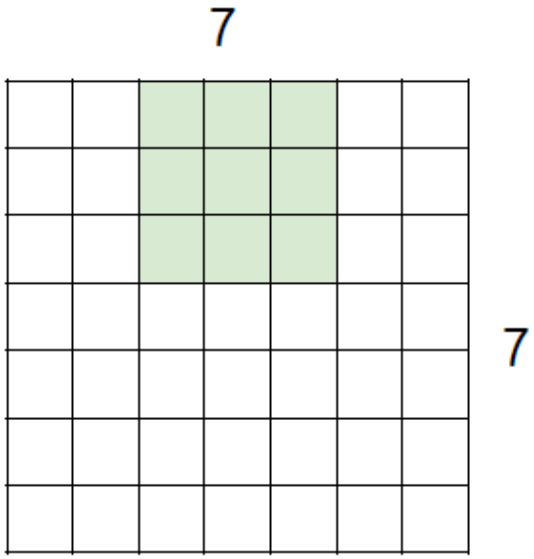
Stride and Padding



Stride = 2
Padding = 0

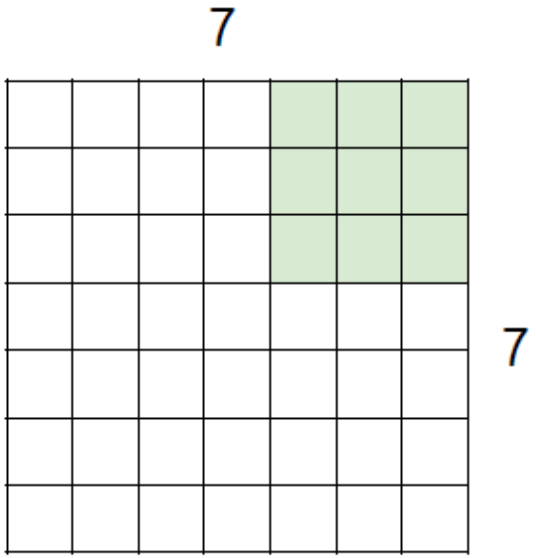
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Stride and Padding



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

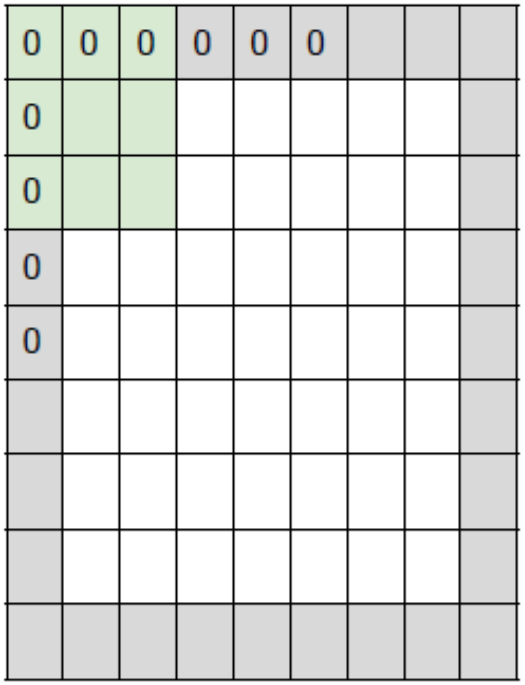
Stride and Padding



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Stride and Padding

In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

Summary of convolutions

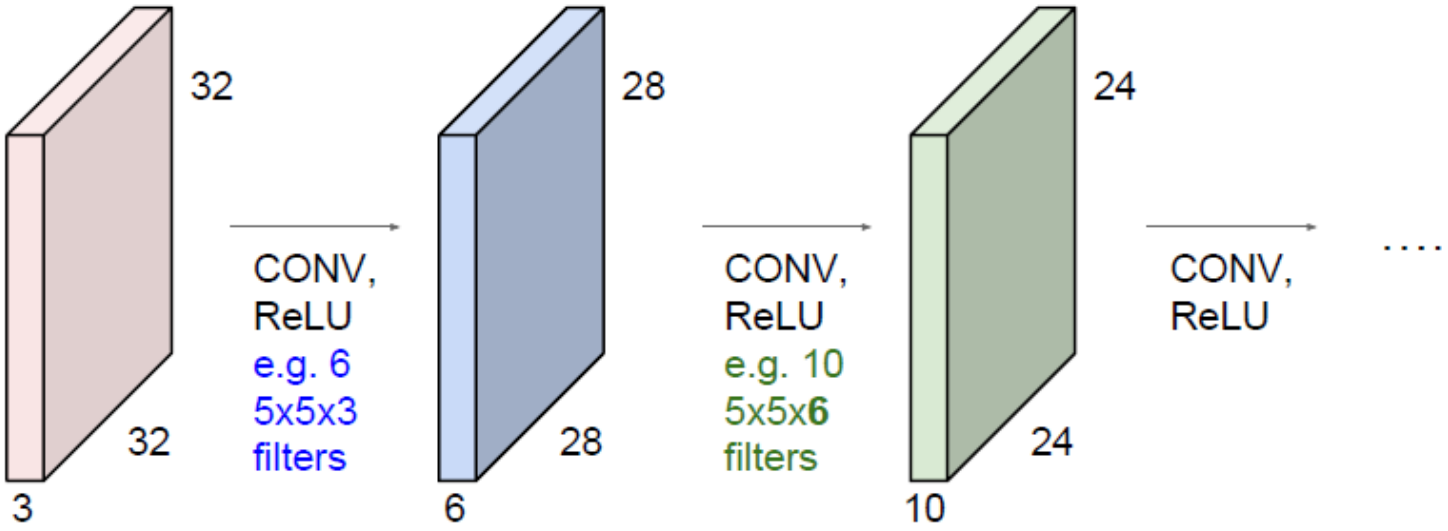
$n \times n$ image $\quad f \times f$ filter

padding p $\quad$ stride s

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

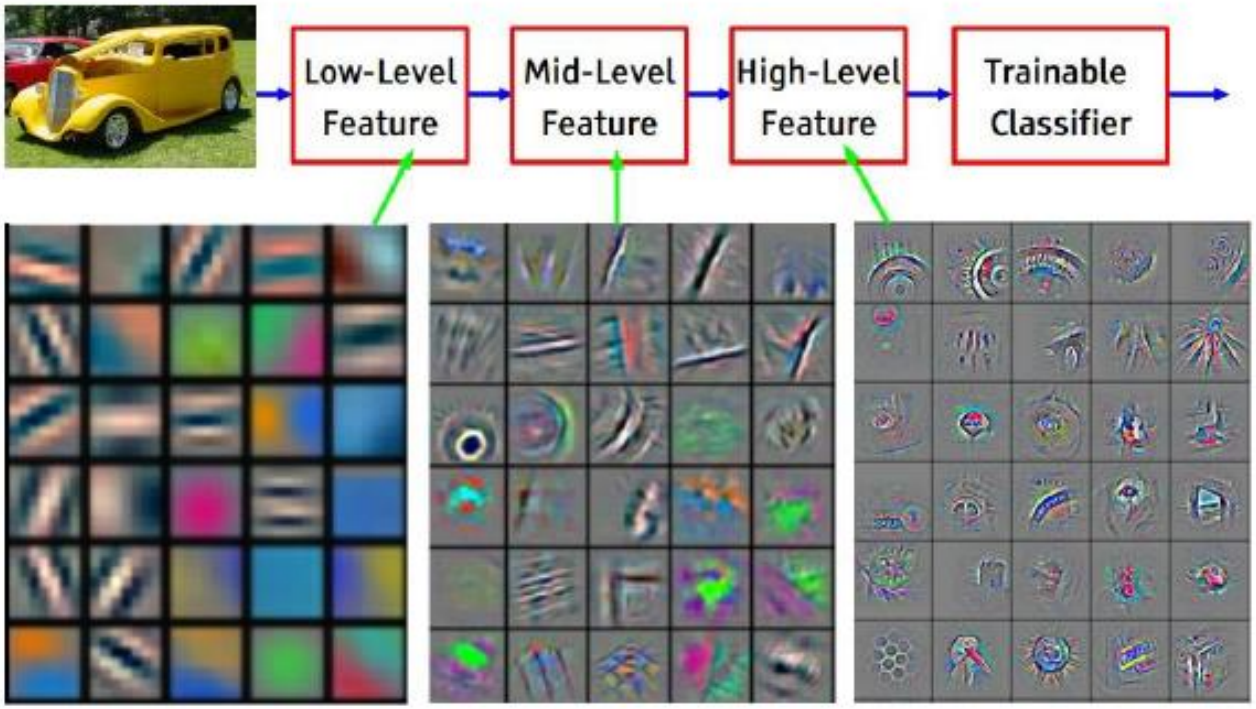
From: Andrew Ng

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



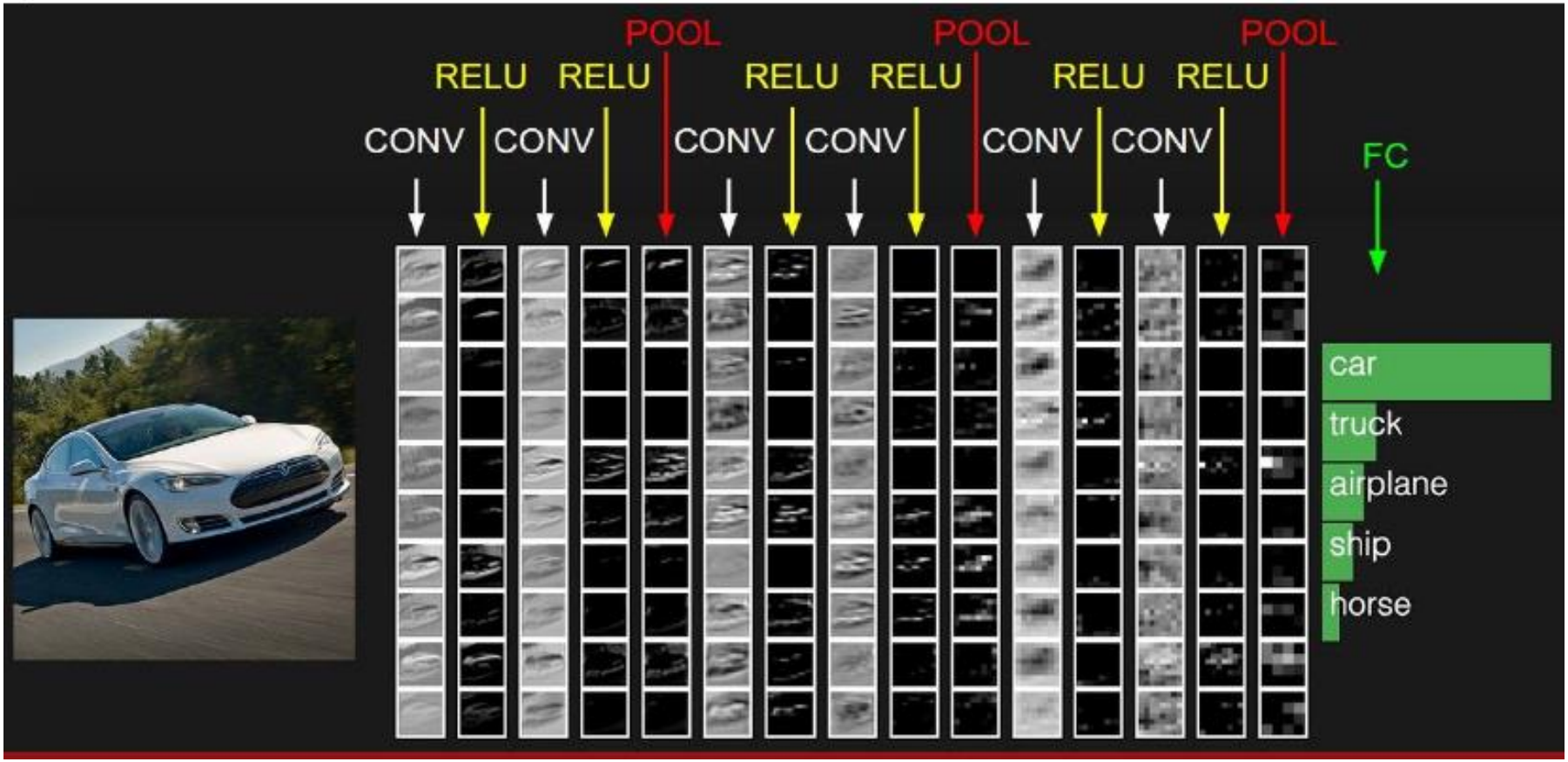
Preview

[From recent Yann LeCun slides]



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

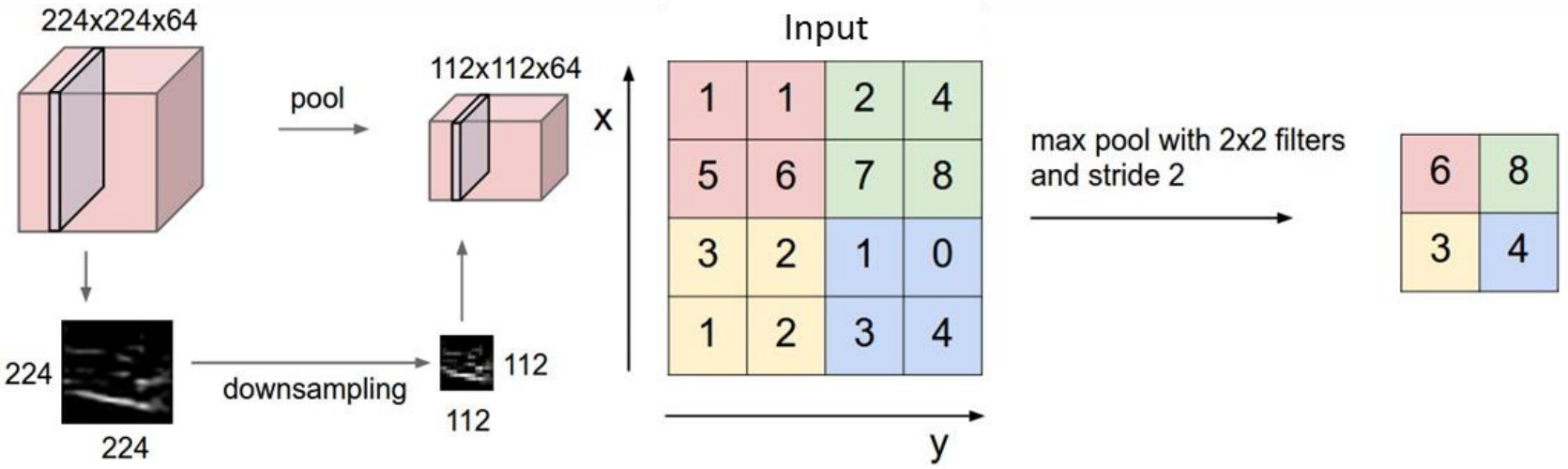
preview:



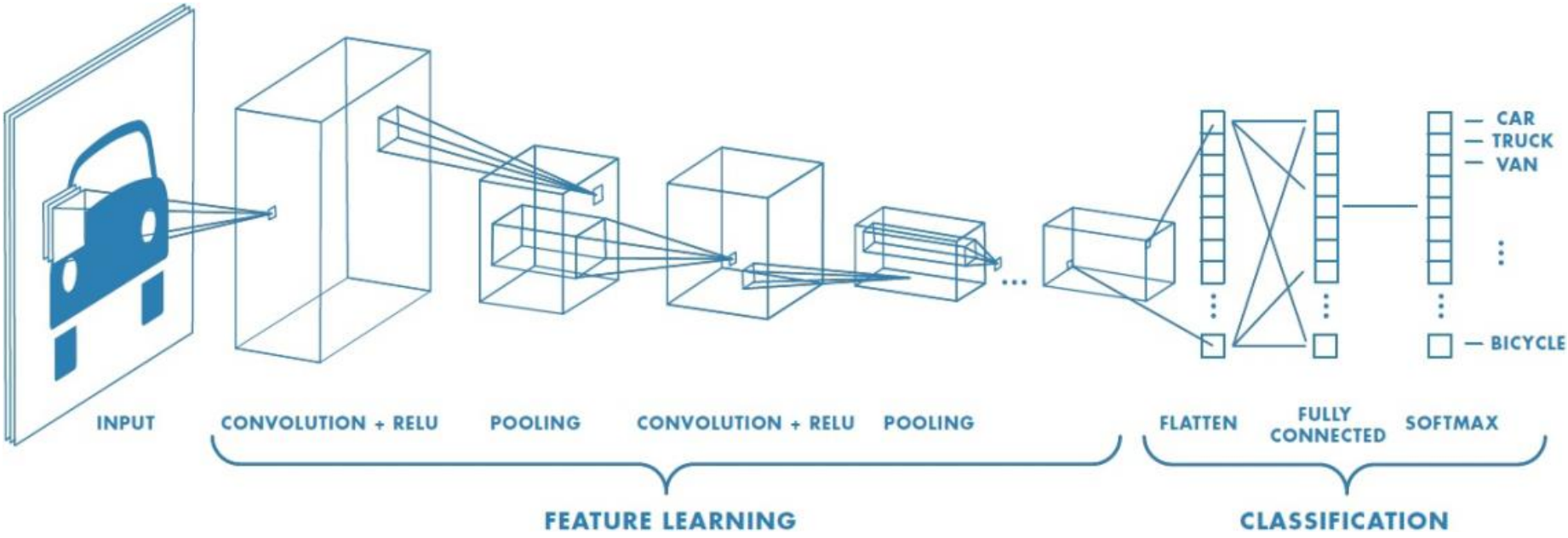
Fei-Fei Li & Andrej Karpathy & Justin Johnson

CNN Architecture: Pooling Layer

- Makes the representation smaller and more manageable
- Operates over each activation map independently



CNN / ConvNet Architecture:



https://sefiks.com/2017/11/03/a-gentle-introduction-to-convolutional-neural-networks/

MSTC_FontReco_CNN_2018.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_2018.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_2018.ipynb (III)

 # You can feed a dropout probability to the Graph and try…

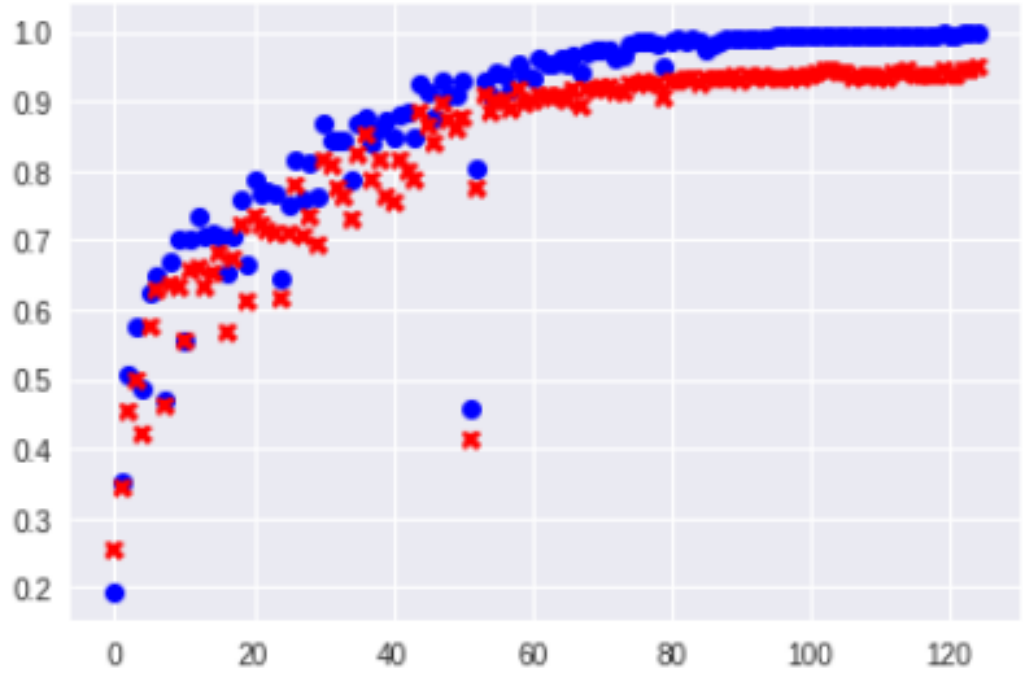
```
keep_prob = tf.placeholder("float")
h2_drop = tf.nn.dropout(h2, keep_prob)
```

 # Define model

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

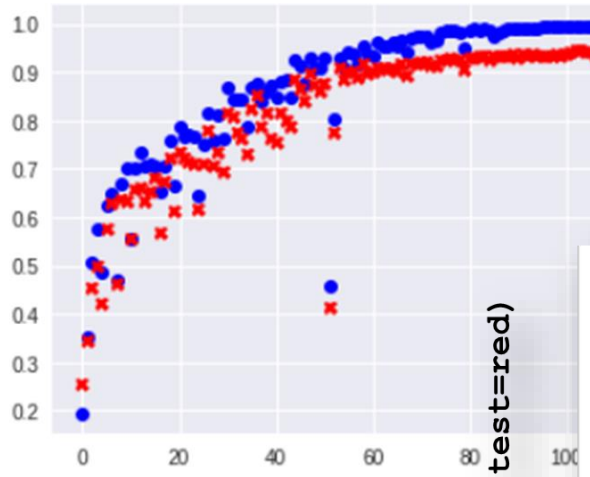
CNN ConvNet

```
Train Accuracy:  1.0
Test_Accuracy:  0.95
[<matplotlib.lines.Line2D at 0x7f1c8efd9550>]
```



CNN ConvNet

```
Train Accuracy:  1.0
Test_Accuracy:  0.95
[<matplotlib.lines.Line2D at 0x7f1c8efd9550>]
```

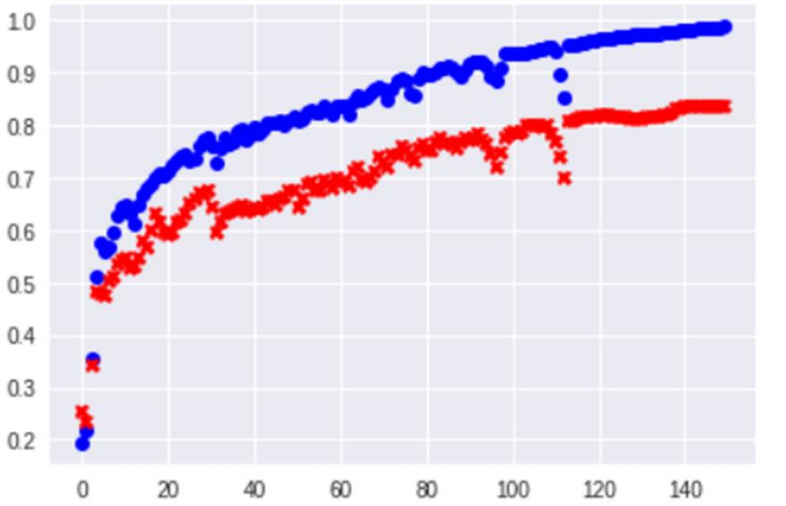


Acc (Train=blue; test=red)

Epoch/10 no.

Feed Forward

```
Train Accuracy:  0.99
Test_Accuracy:  0.84
[<matplotlib.lines.Line2D at 0x7efff740dc18>]
```



Acc (Train=blue; test=red)

Epoch/10 no.

Activities:

- **Font Recognition using CNN with Keras:**
  Try it yourselt!

Some Interesting Working Lines

GAN : Generative Adversarial Networks

Yann LeCun called adversarial training
"**the most interesting idea in the last 10 years in ML**."

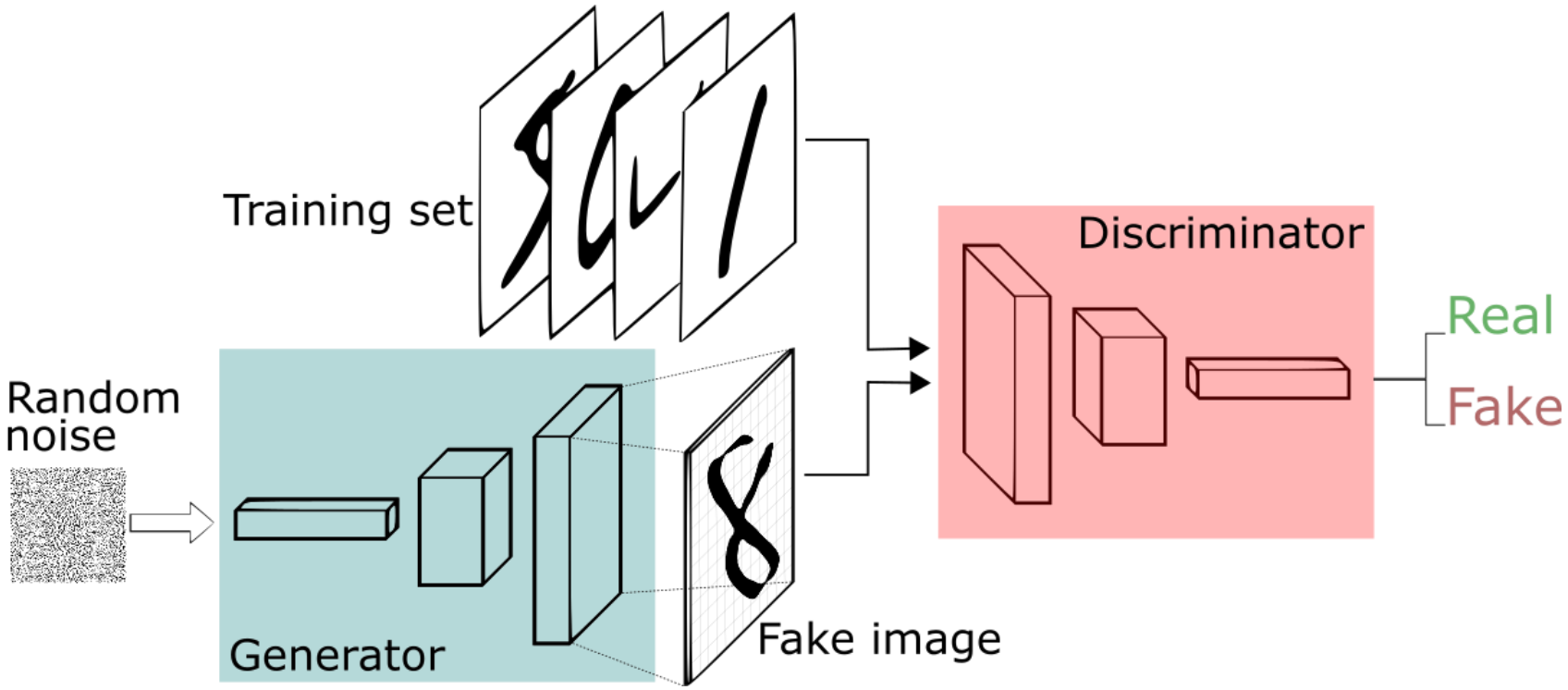
GANs' potential is huge, because they can learn
to mimic any distribution of data

What I cannot build, I do not understand.
- Richard Feynman

Generative Adversarial Networks

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio

(Submitted on 10 Jun 2014)



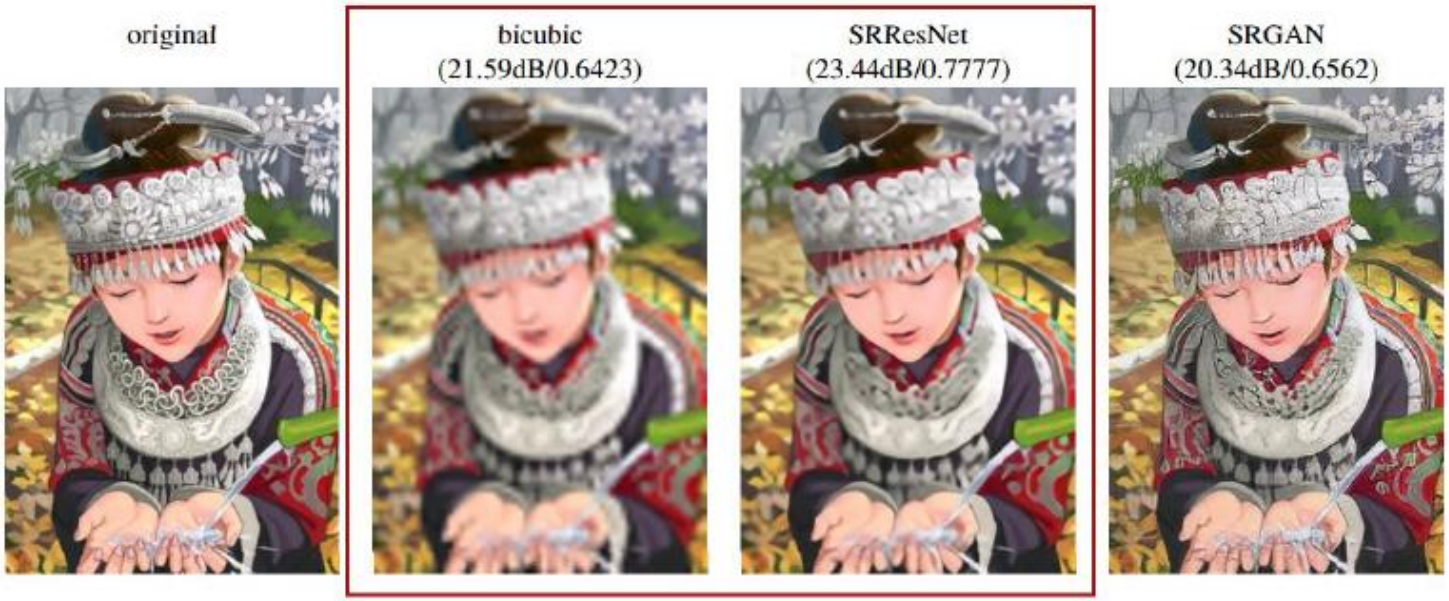
Inverse convolutional network

https://deeplearning4j.org/generative-adversarial-network

GAN : Generative Adversarial Networks

Applications (I):

- Images: noise, super-resolution



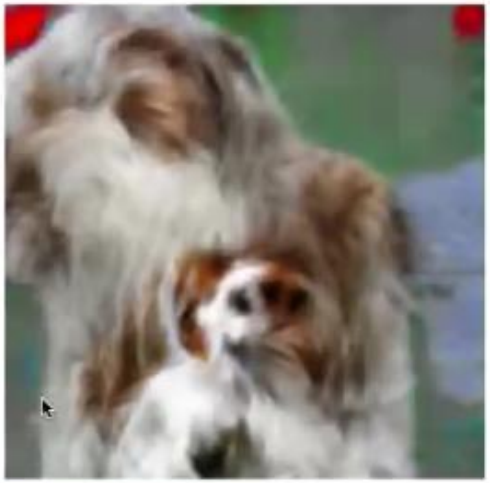
https://www.cs.toronto.edu/~duvenaud/courses/csc2541/slides/gan-applications.pdf

GAN : Applications (II):

- Text-to-Image Synthesis
- Parametric Speech Synthesis
- Adversarial Perturbation Elimination
- Disentangling factors of variation; sources of variability

https://www.cs.toronto.edu/~duvenaud/courses/csc2541/slides/gan-applications.pdf

Cherry-Picked Results

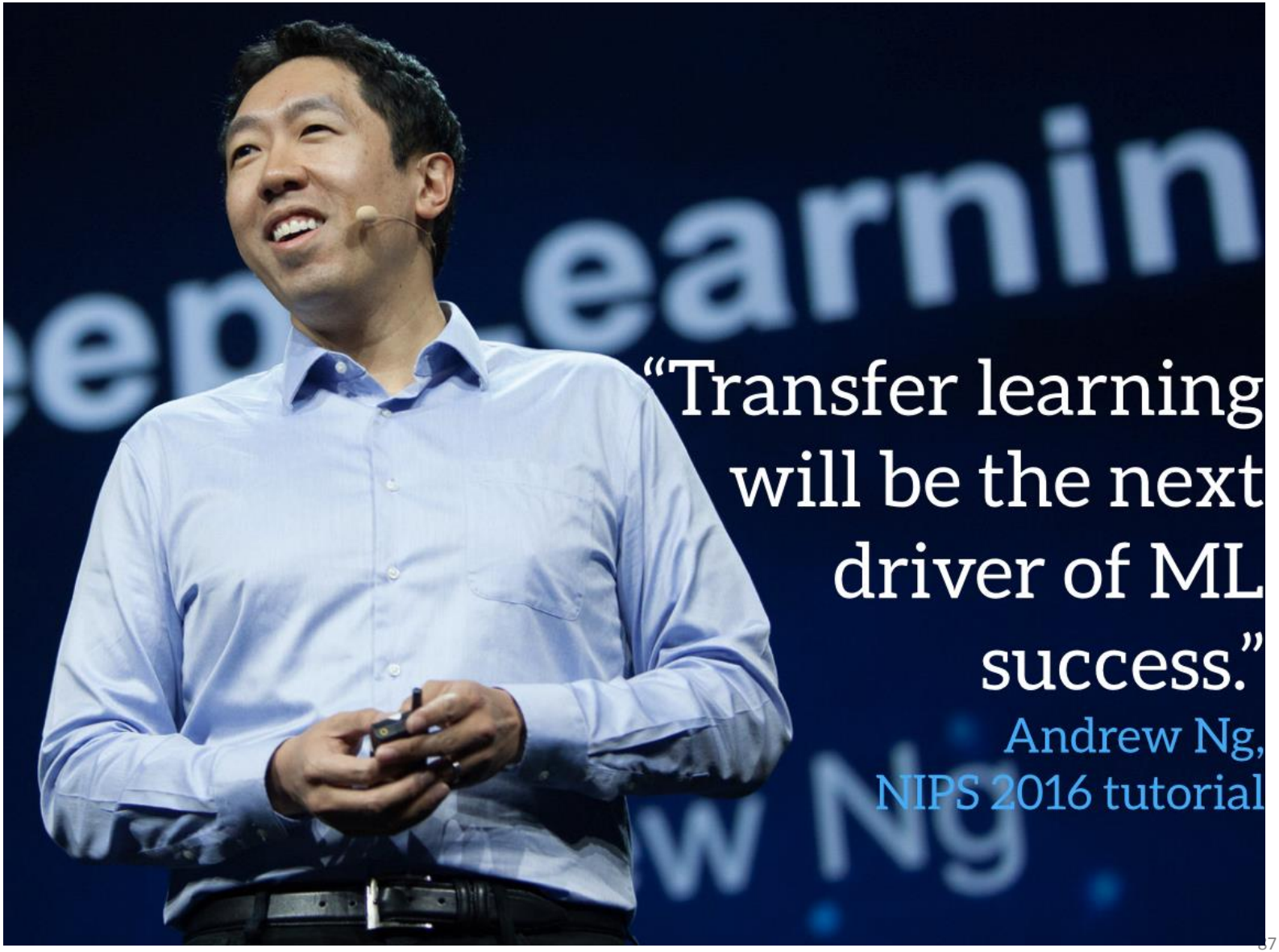




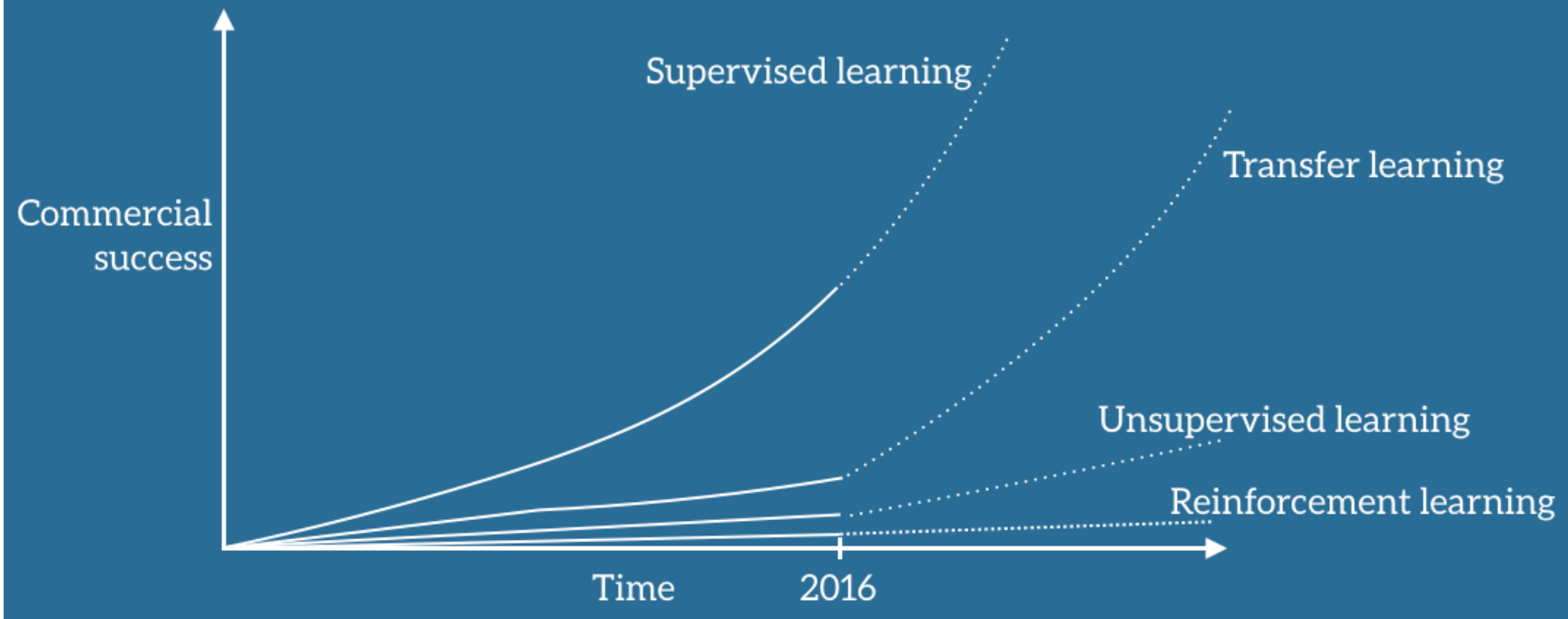


(Goodfellow 2016)

Transfer Learning



Drivers of ML success in industry



- Andrew Ng, NIPS 2016 tutorial

Transfer Learning

Given a source domain $\mathcal{D}_S$, a corresponding source task $\mathcal{T}_S$, as well as a target domain $\mathcal{D}_T$ and a target task $\mathcal{T}_T$, the objective of transfer learning now is to enable us to learn the target conditional probability distribution $P(Y_T|X_T)$ in $\mathcal{D}_T$ with the information gained from $\mathcal{D}_S$ and $\mathcal{T}_S$ where $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$. In most cases, a limited number of labeled target examples, which is exponentially smaller than the number of labeled source examples are assumed to be available.

http://ruder.io/transfer-learning/

Transfer Learning

- Using pre-trained models
- Domain-invariant representations
- Learning from simulations

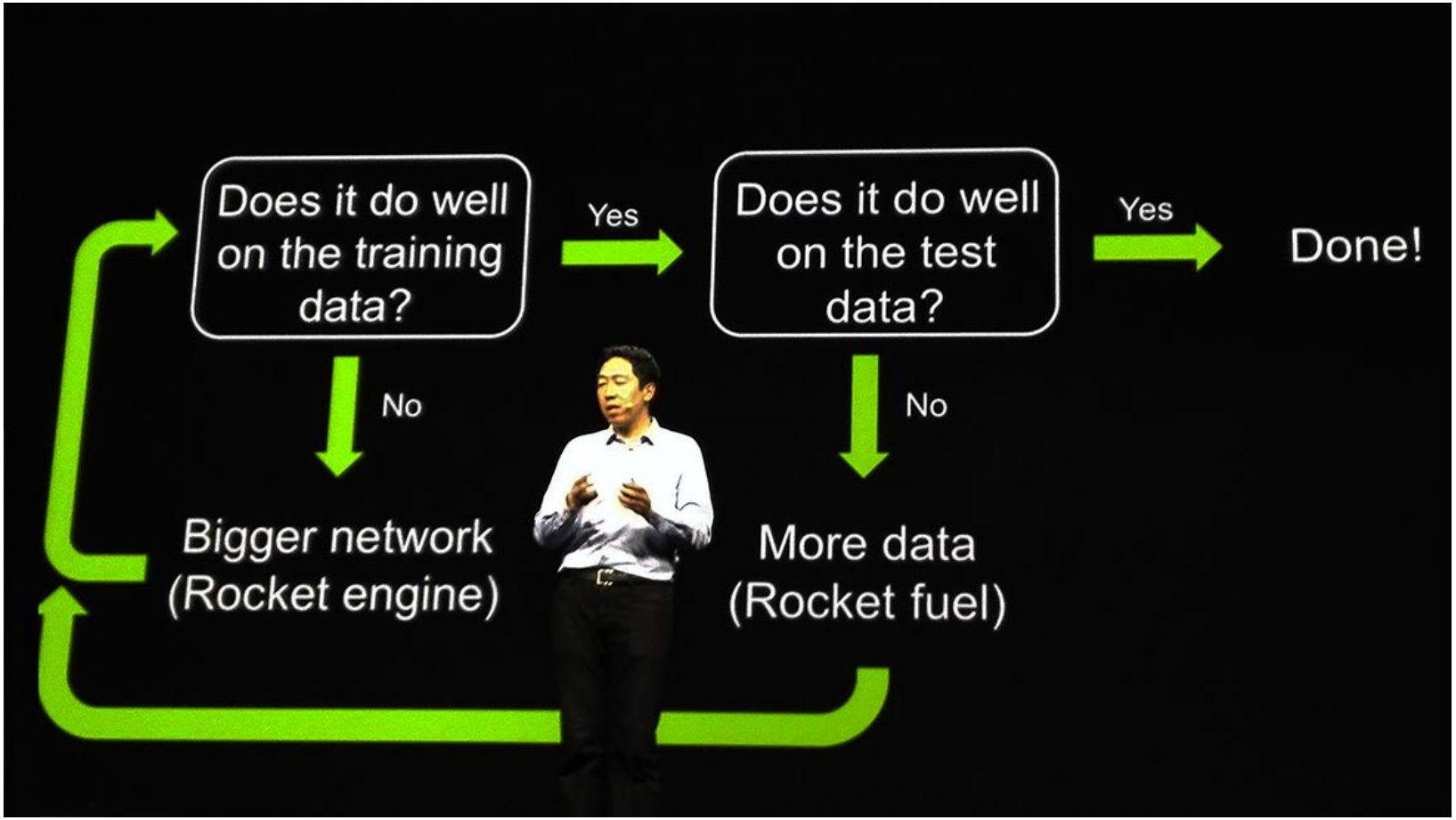
Udacity's self-driving car simulator (source: TechCrunch)

- Robots
- Training an agent to achieve general AI

http://ruder.io/transfer-learning/

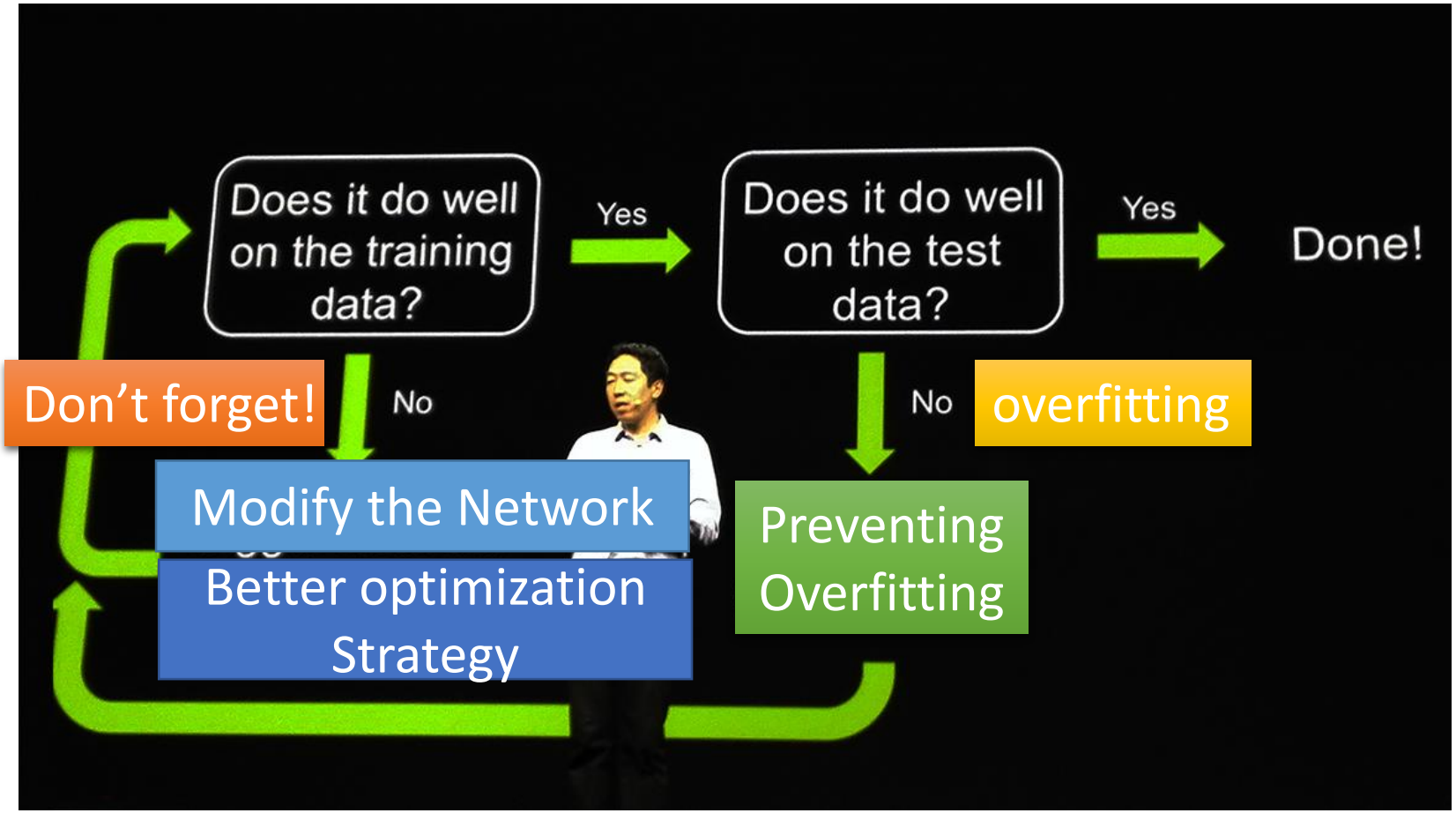
Tips for Training DNN

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

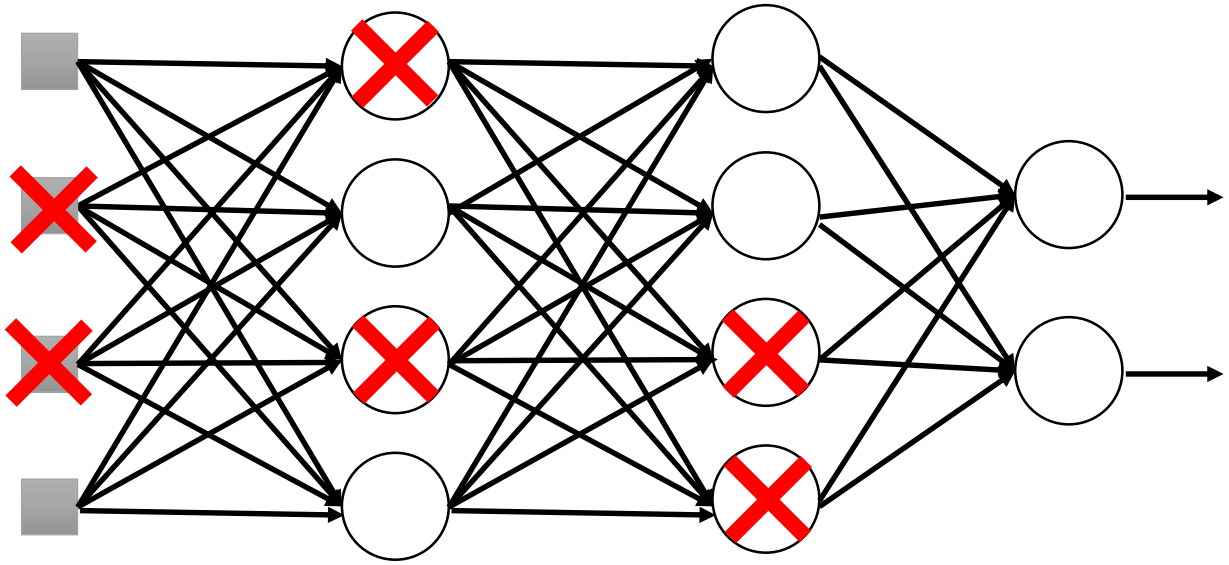
Dropout

Dropout

Pick a mini-batch

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

Training:



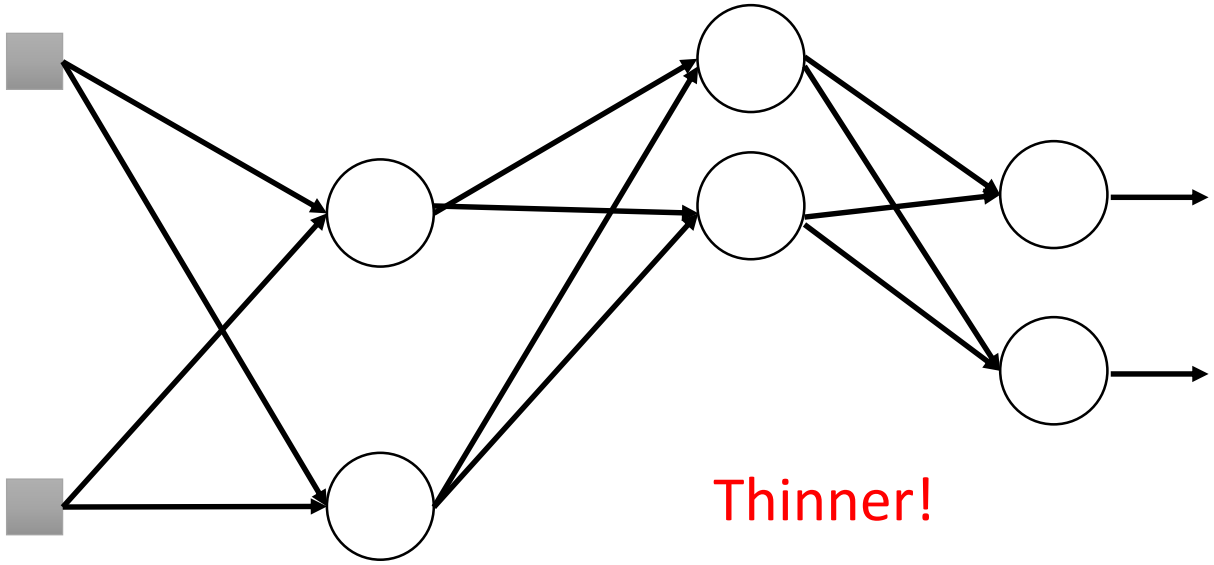
- **Each time before computing the gradients**
  - Each neuron has p% to dropout

Dropout

Pick a mini-batch

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

Training:



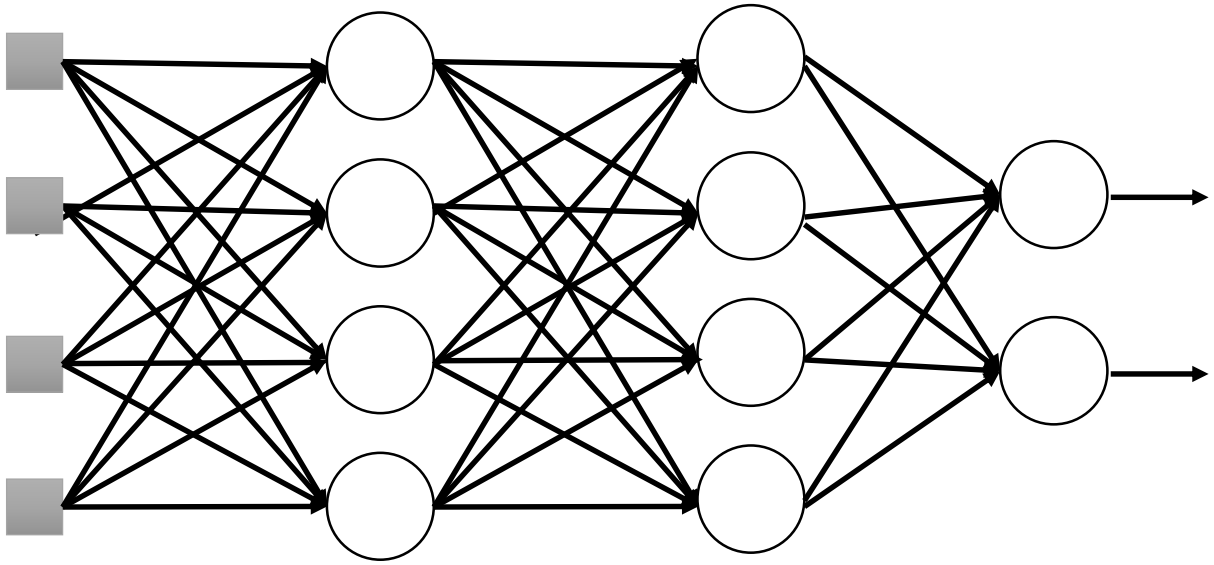
- **Each time before computing the gradients**
  - Each neuron has p% to dropout

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

For each mini-batch, we resample the dropout neurons

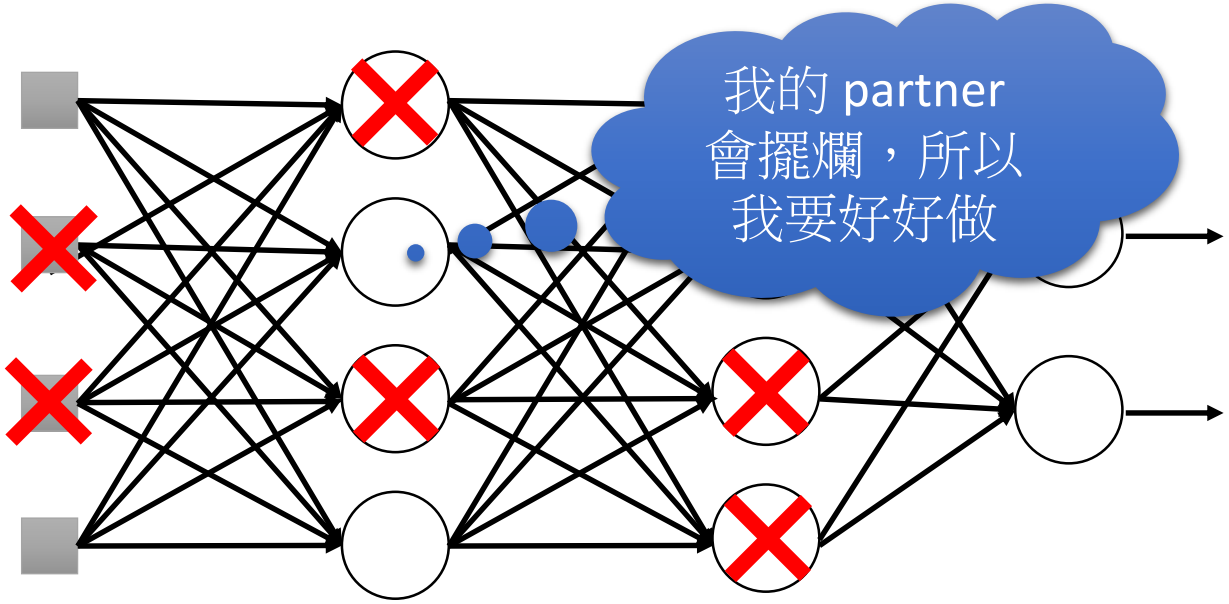
Dropout

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



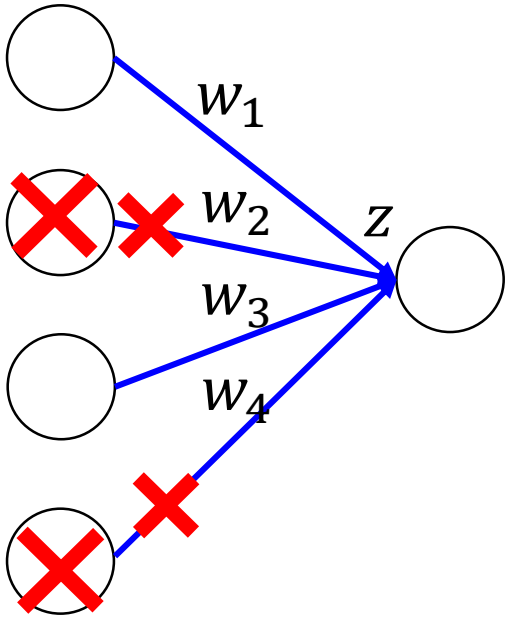
- When teams up, if everyone expect the partner will do the work, nothing will be done finally.
- However, if you know your partner will dropout, you will do better.
- When testing, no one dropout actually, so obtaining good results eventually.

Dropout - Intuitive Reason

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

Training of Dropout

Assume dropout rate is 50%

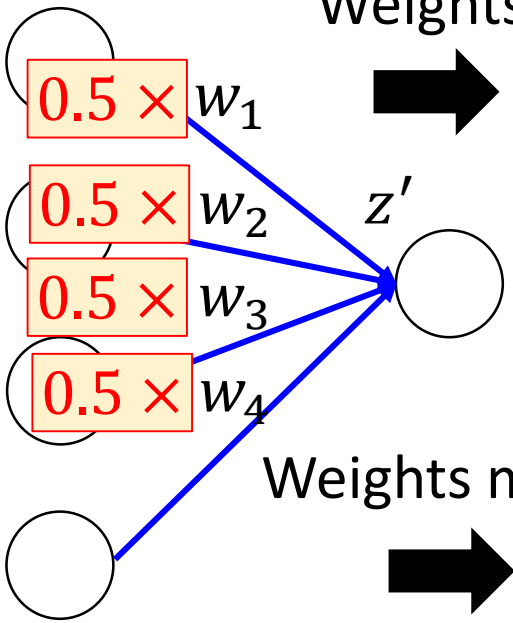


Testing of Dropout

No dropout

Weights from training

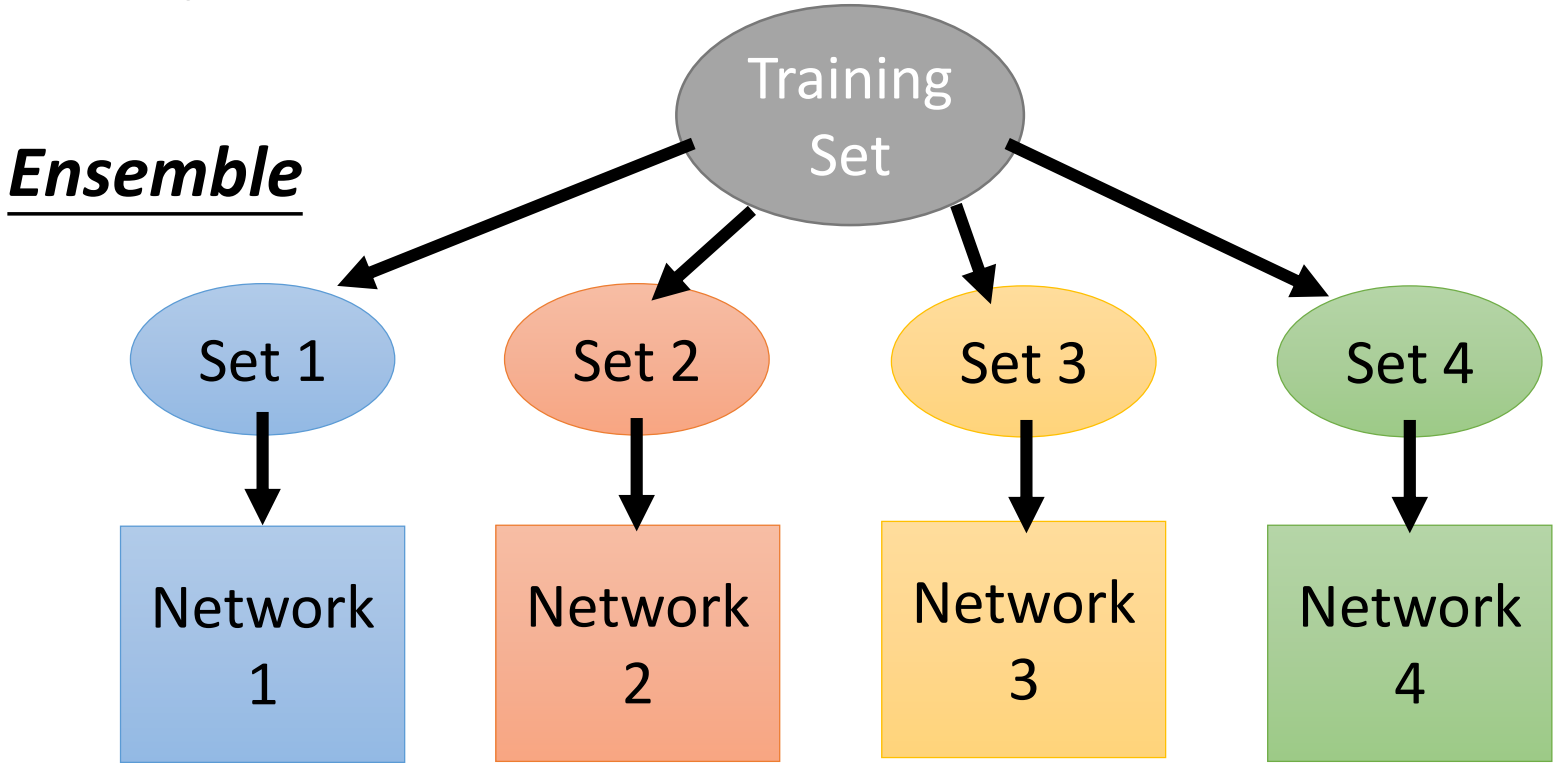
$z' \approx 2z$



Weights multiply (1-p)%

$z' \approx z$

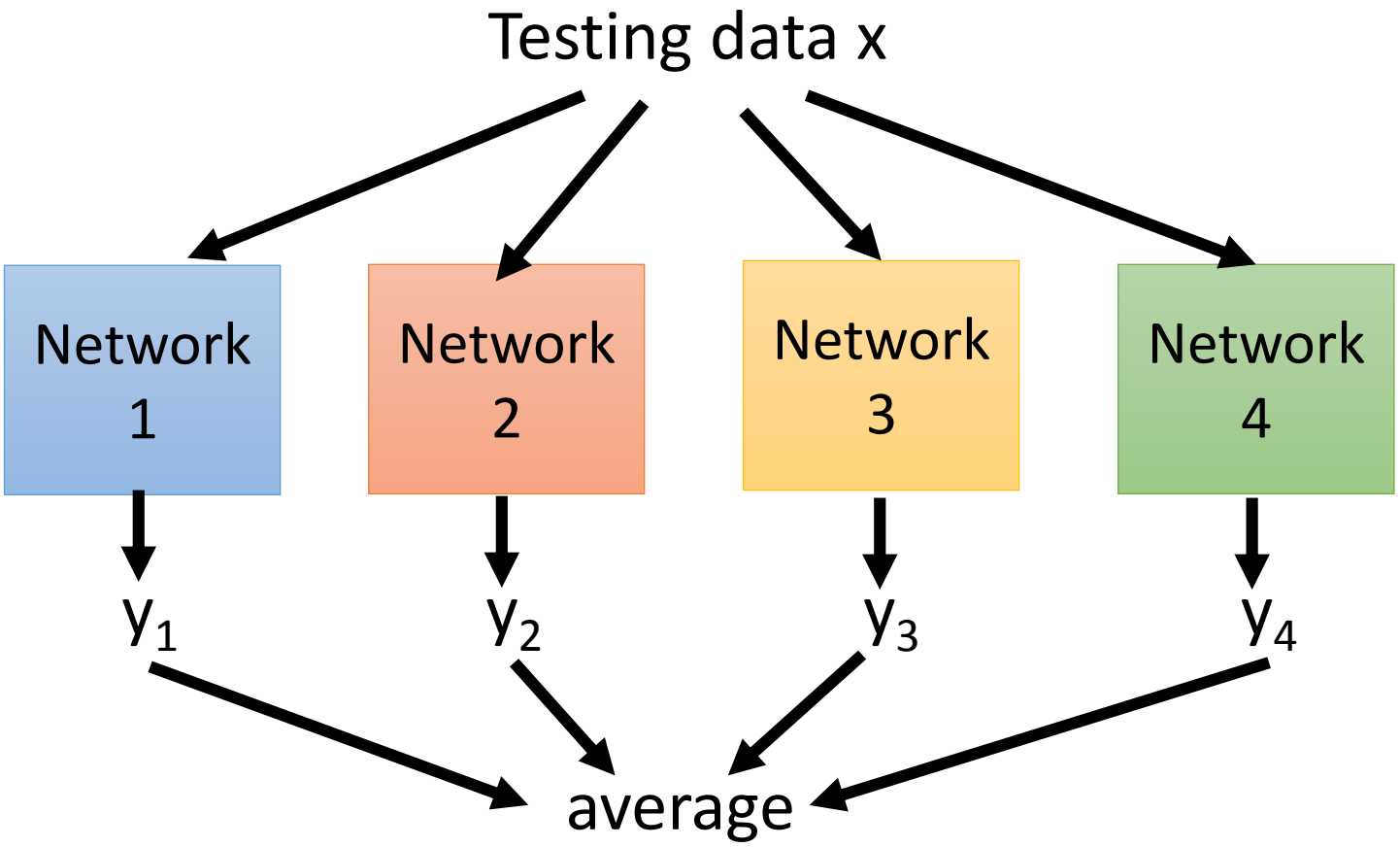
Dropout is a kind of ensemble.



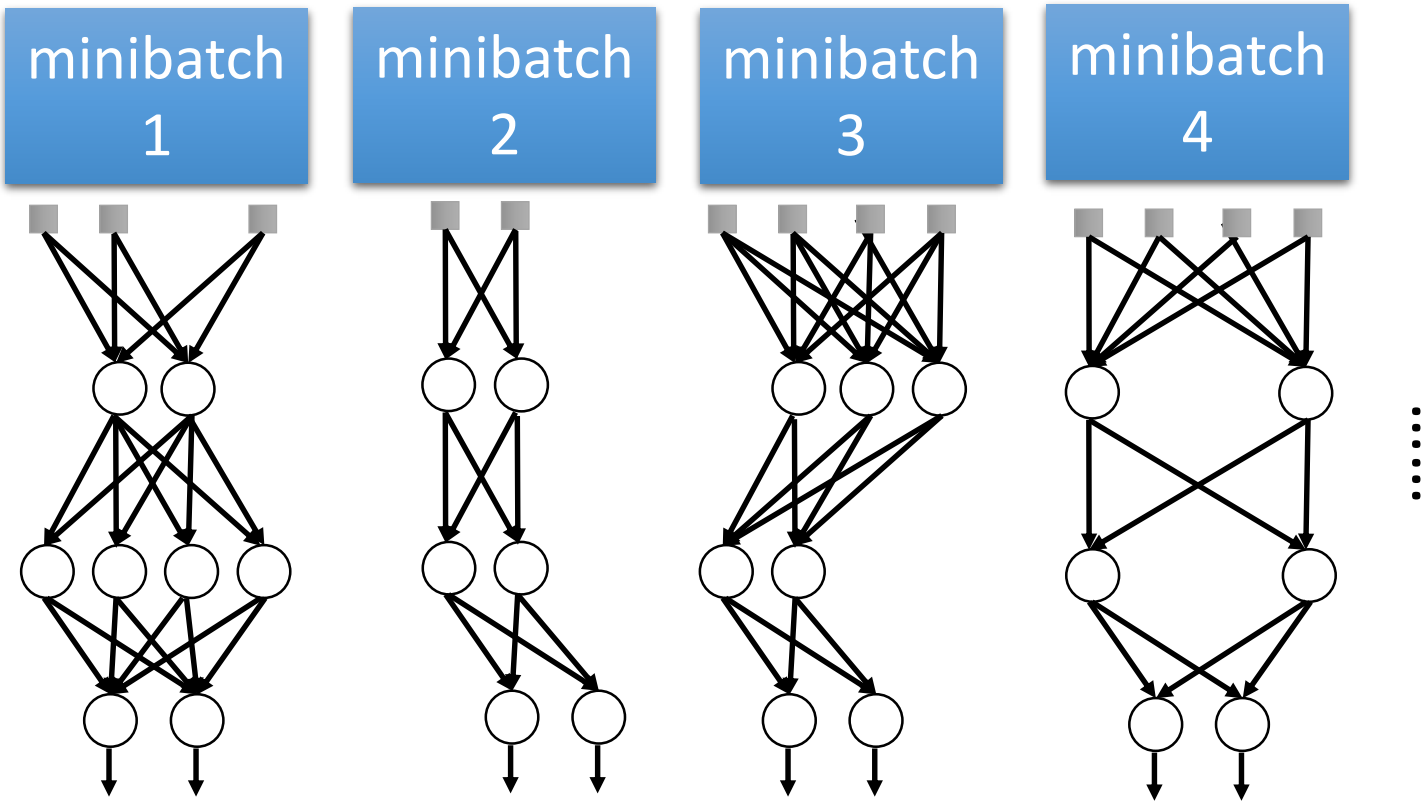
Train a bunch of networks with different structures

Dropout is a kind of ensemble.

Ensemble



Dropout is a kind of ensemble.



Training of Dropout

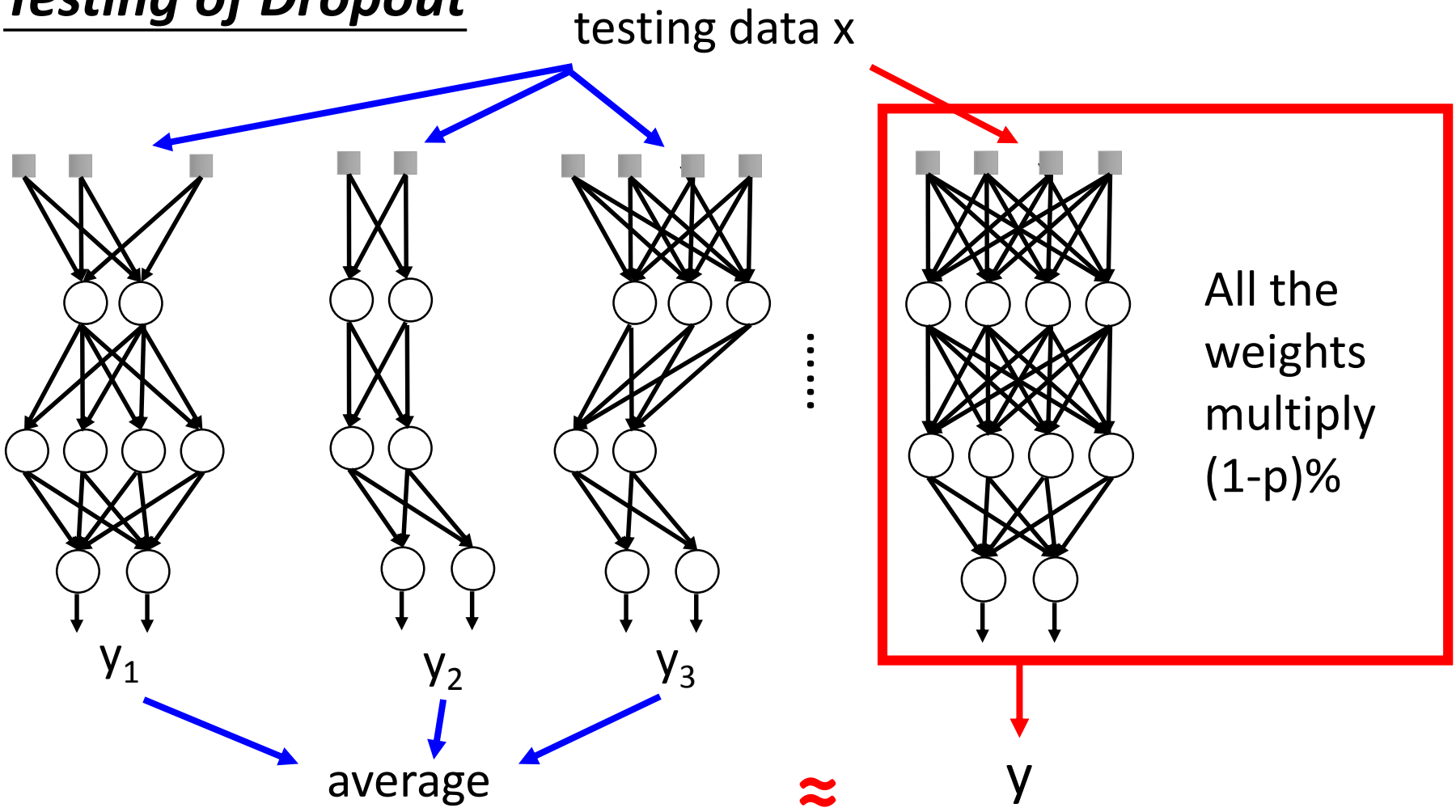
M neurons

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