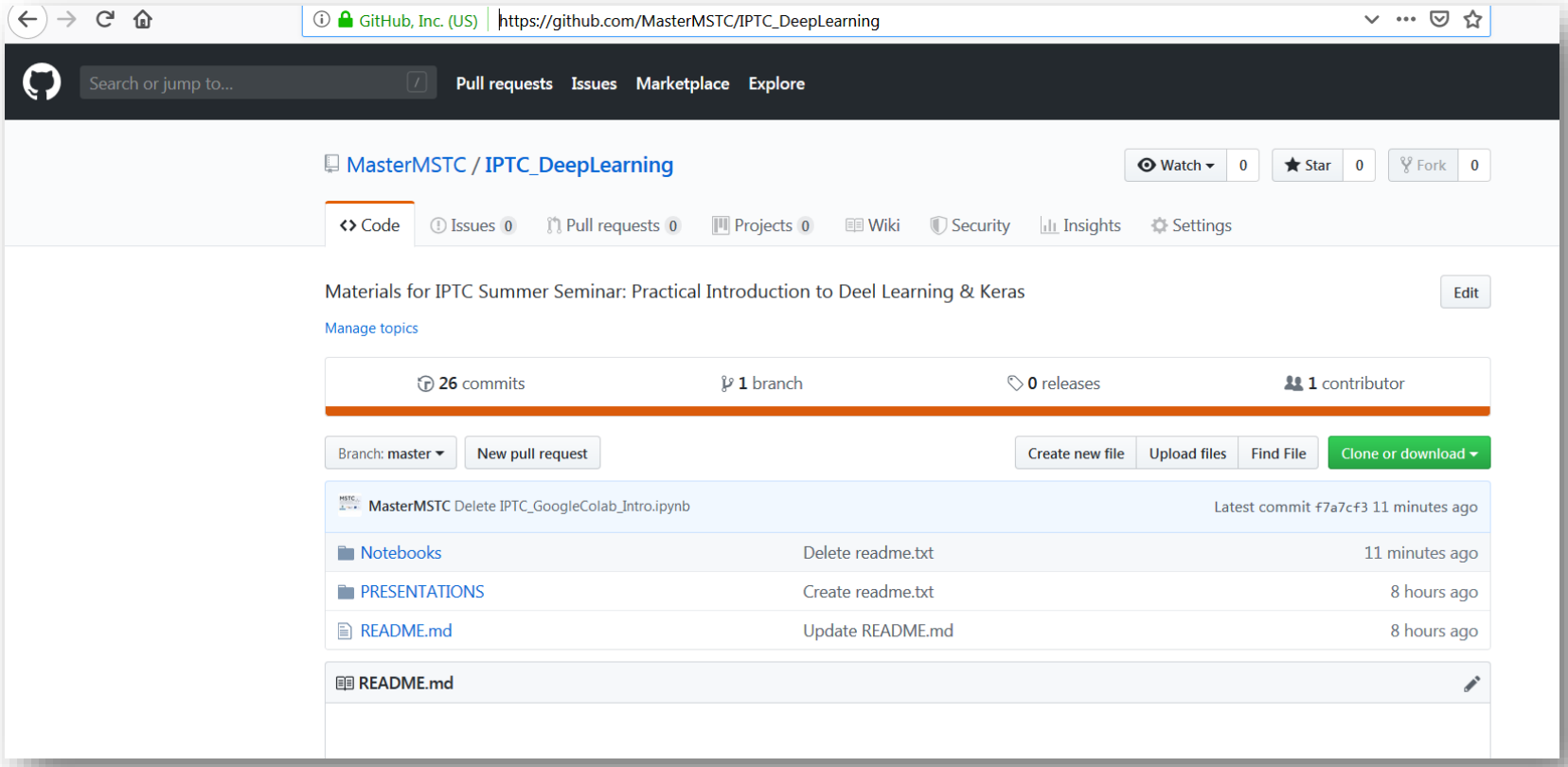
IPTC Seminar
Introduction to Deep Learning and Keras

July 15, 16 and 17, 2019
ETSIT UPM, Madrid

IPTC Summer Seminar MATERIALS

Please go to our MSTC GitHub:
https://github.com/MasterMSTC/IPTC_DeepLearning



Introduction to Deep Learning & Keras
Day 2

iptc

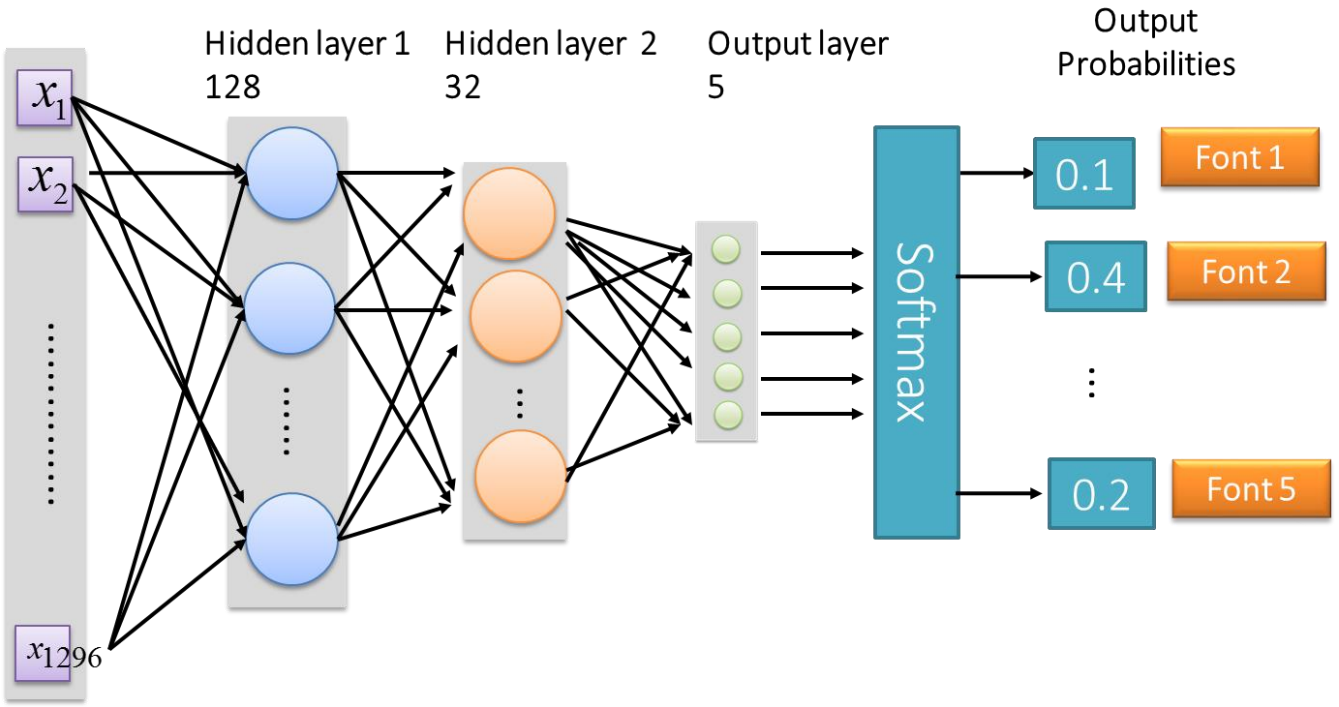
Information Processing and Telecommunications Center

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

E.T.S. Ingenieros de Telecomunicación
Universidad Politécnica de Madrid


POLITÉCNICA
iptc
Information Processing and Telecommunications Center
Day 1 (July 15) : Intro to Deep Learning & Keras.
Feed Forward models
Day 2 (July 16) : Backpropagation. Convolutional Networks
Transfer Learning. Data Augmentation.
Day 3 (July 17) : Recurrent Networks. Advanced Architectures.
GANs. Applications. Final discussion on AI
IPTC Seminar
Introduction to Deep Learning and Keras
July 15, 16 and 17, 2019
ETSIT UPM, Madrid

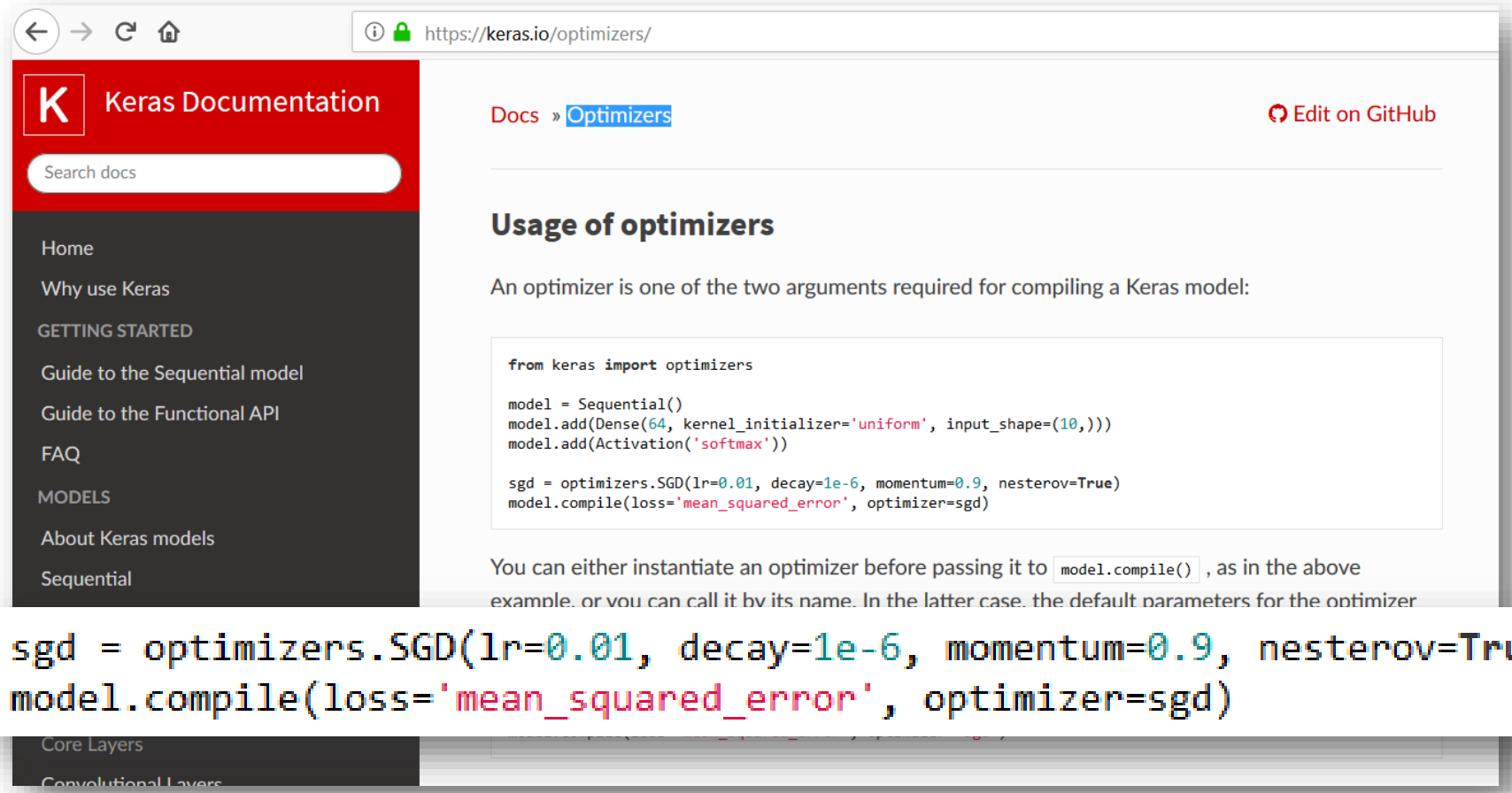
How are the Network Parameters estimated (best W and b values)?



W's and b's in every layer

Optimizers

https://keras.io/optimizers/




```
sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='mean_squared_error', optimizer=sgd)
```

Next Slides are from:

Deep Learning Tutorial

李宏毅

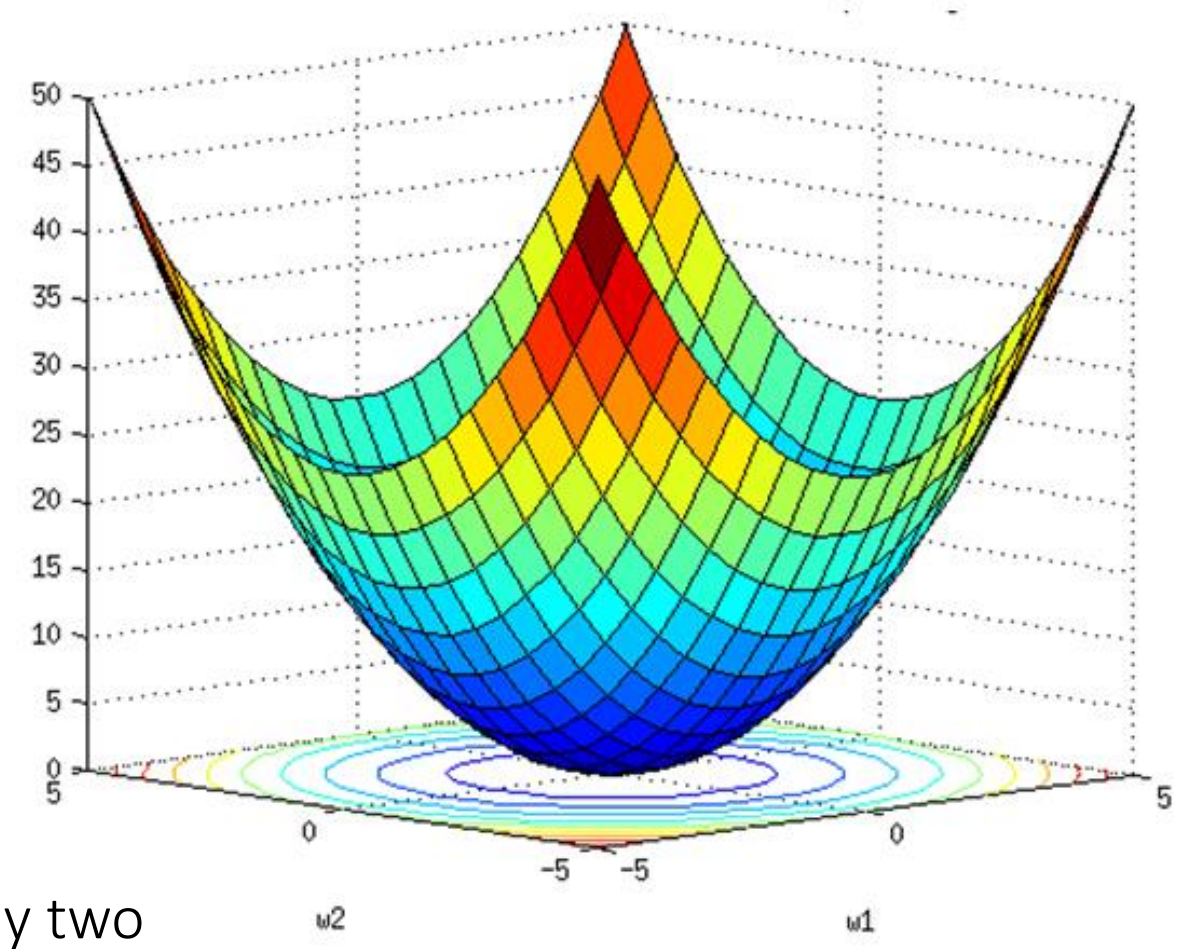
Hung-yi Lee

OPTIMIZATION

Gradient Descent

Cost
Or
Loss function

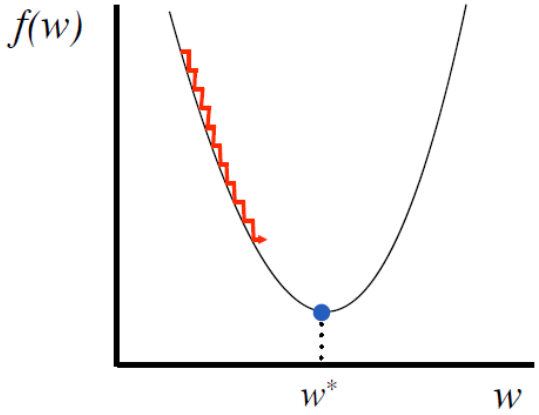
$C(\theta)$



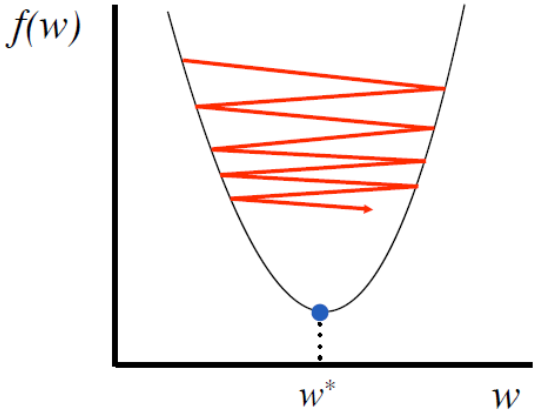
Assume there are only two parameters w_1 and w_2 in a network.

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

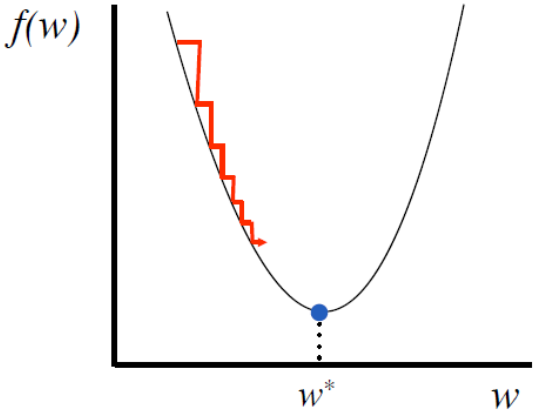
Choosing Step Size



Too small: converge very slowly



Too big: overshoot and even diverge



Reduce size over time

Theoretical convergence results for various step sizes

A common step size is $\alpha_i = \frac{\alpha}{n\sqrt{i}}$

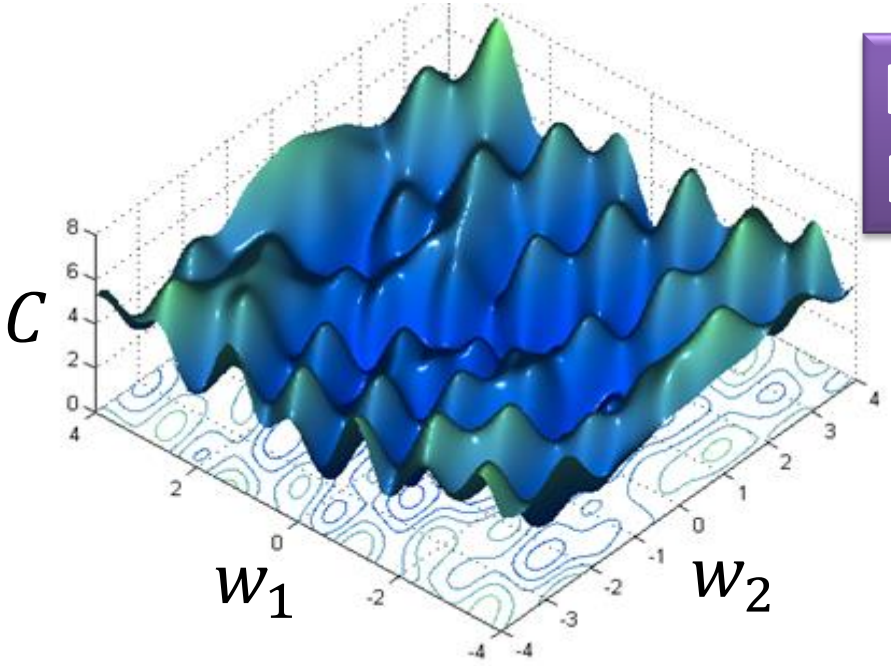
α — Constant

n — # Training Points

i — Iteration #

Source: edX | BerkeleyX: CS190.1x Scalable Machine Learning

Gradient descent never guarantee global minima

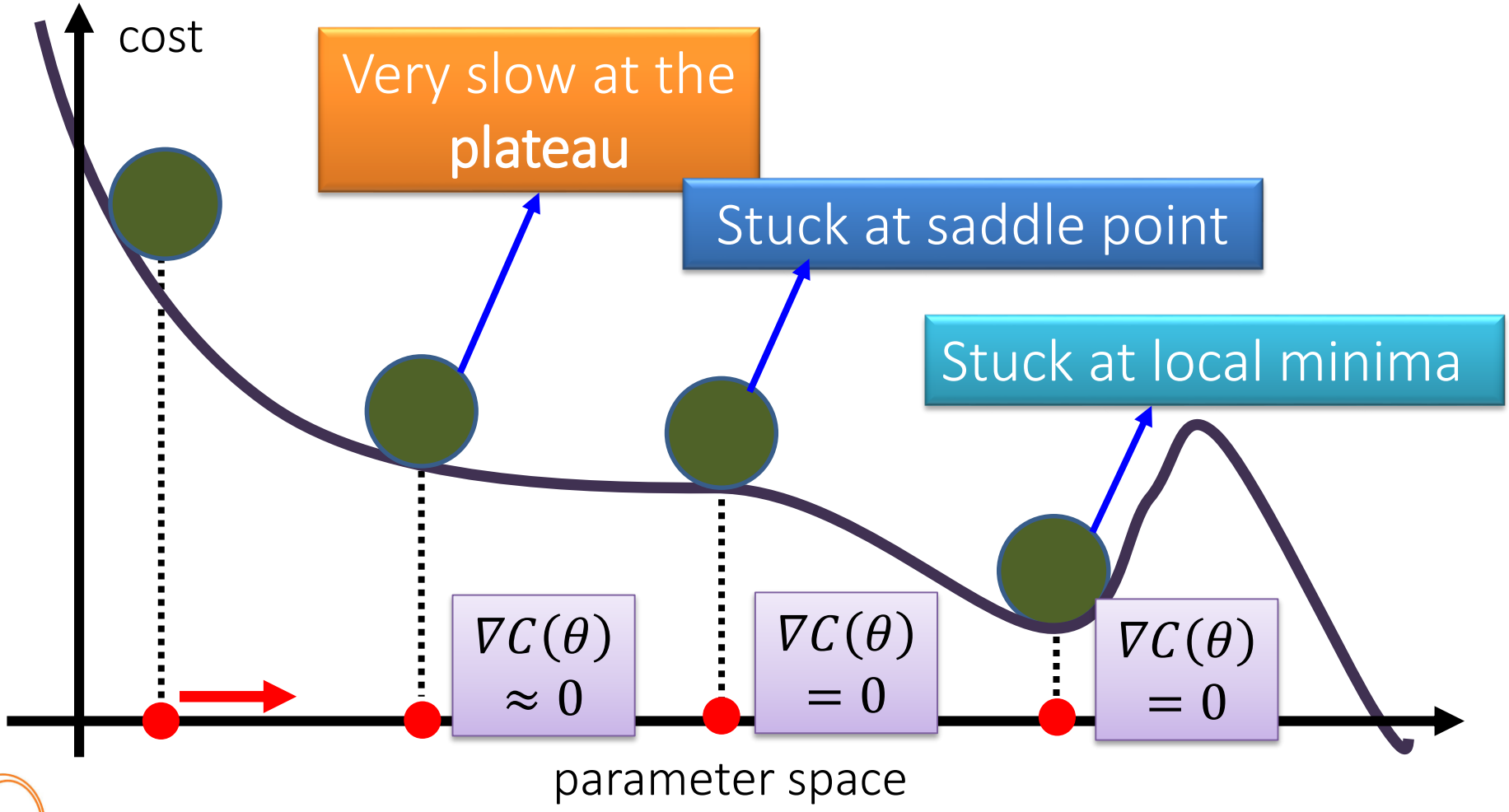


Keep in mind that:
Different initializations will

Reach different minima, so different results

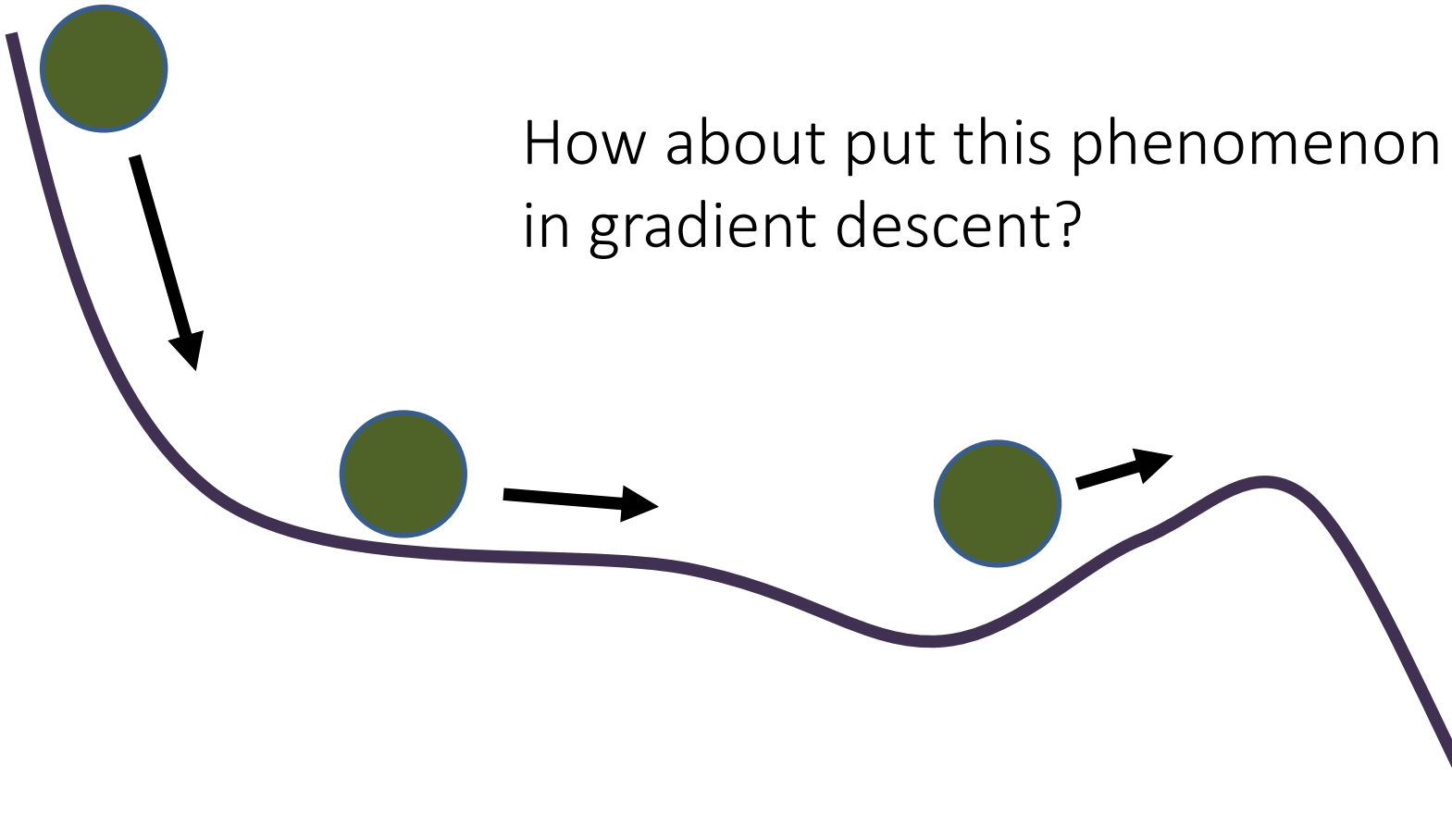
Who is Afraid of Non-Convex Loss Functions? (17:36 -27:16 -30:41 min)
http://videolectures.net/eml07_lecun_wia/

Besides local minima ……



In physical world ……

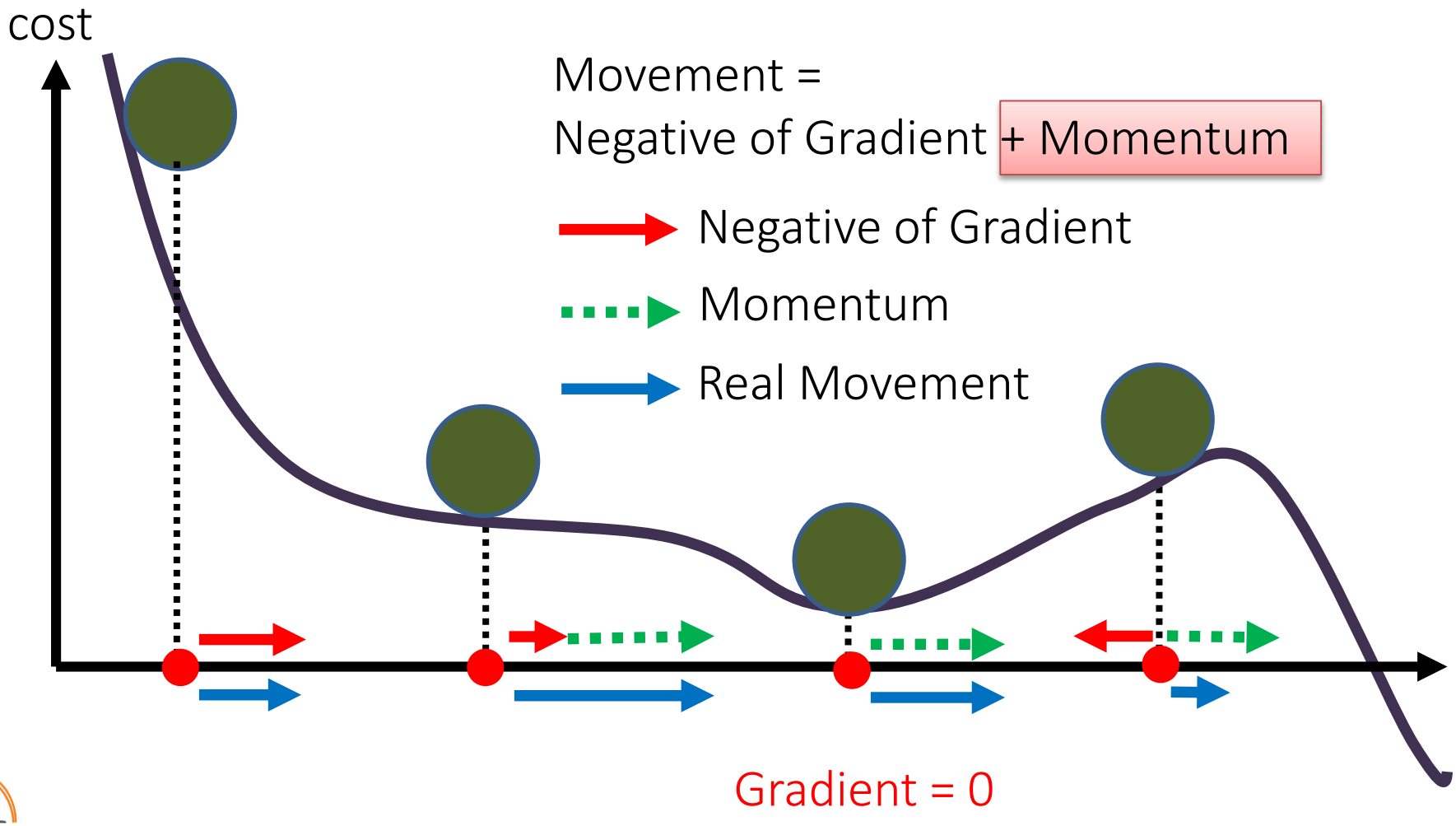
- Momentum



How about put this phenomenon in gradient descent?

Momentum

Still not guarantee reaching global minima, but give some hope ……



But What Cost function?

How to measure accuracy?

Cross-entropy
and
Maximum Likelihood Estimation

Intuitive approach for a two-class (binary) classifier

Take a data set:
$\{x_n, l_n\}$ of N vectors x_n belonging to two classes (labels l_n =0,1)

Consider our classifier as an estimator of the conditional probability:

- $p(l_n = 1|x_n) = p_{1n}$
- …and so… $p(l_n = 0|x_n)$ = 1 - p_{1n}

A PERFECT classifier

- If l_n=1 then $p(l_n = 1|x_n) = p_{1n}$=1
- If l_n=0 then $p(l_n = 1|x_n) = p_{1n}$=0

In a single expression:

$$p_{1n}^{l_n}(1 - p_{1n})^{(1-l_n)}$$

In a perfect classifier this must be:
a "probability" always 1 for every data *n*!!!

A PERFECT classifier

All the probabilities/likelihood for each sample should be 1

Maximum Likelihood estimation of θ

$$\theta_{ML} = \prod_{n=1}^{N} p_{1n}^{l_n}(1 - p_{1n})^{(1-l_n)}$$

From that a Cost Function can be obtained taking: **the negative logarithm = Cross-entropy**

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

Understand the intuitive "power "of this Loss function

Training Neural Networks

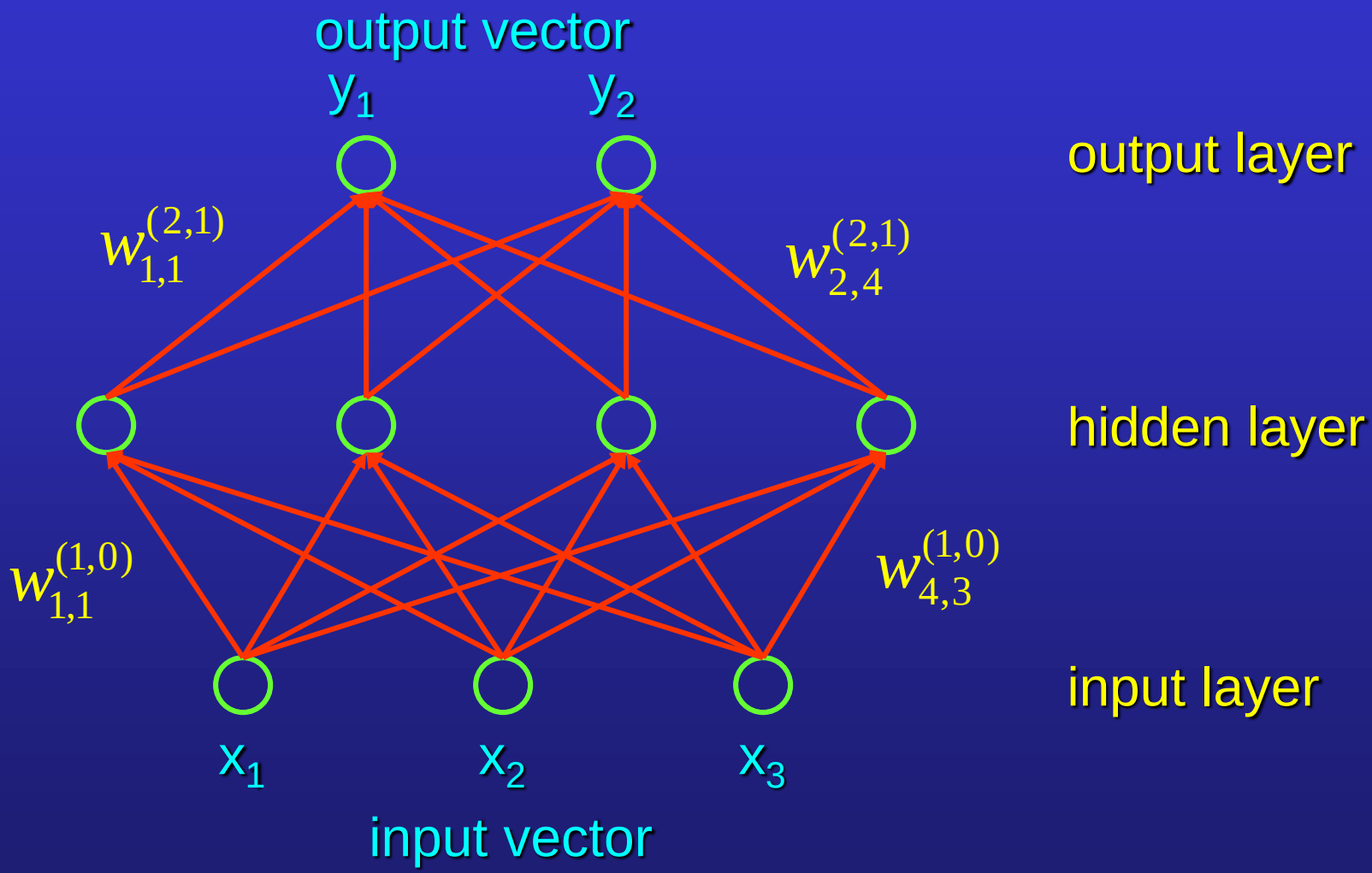
Backpropagation

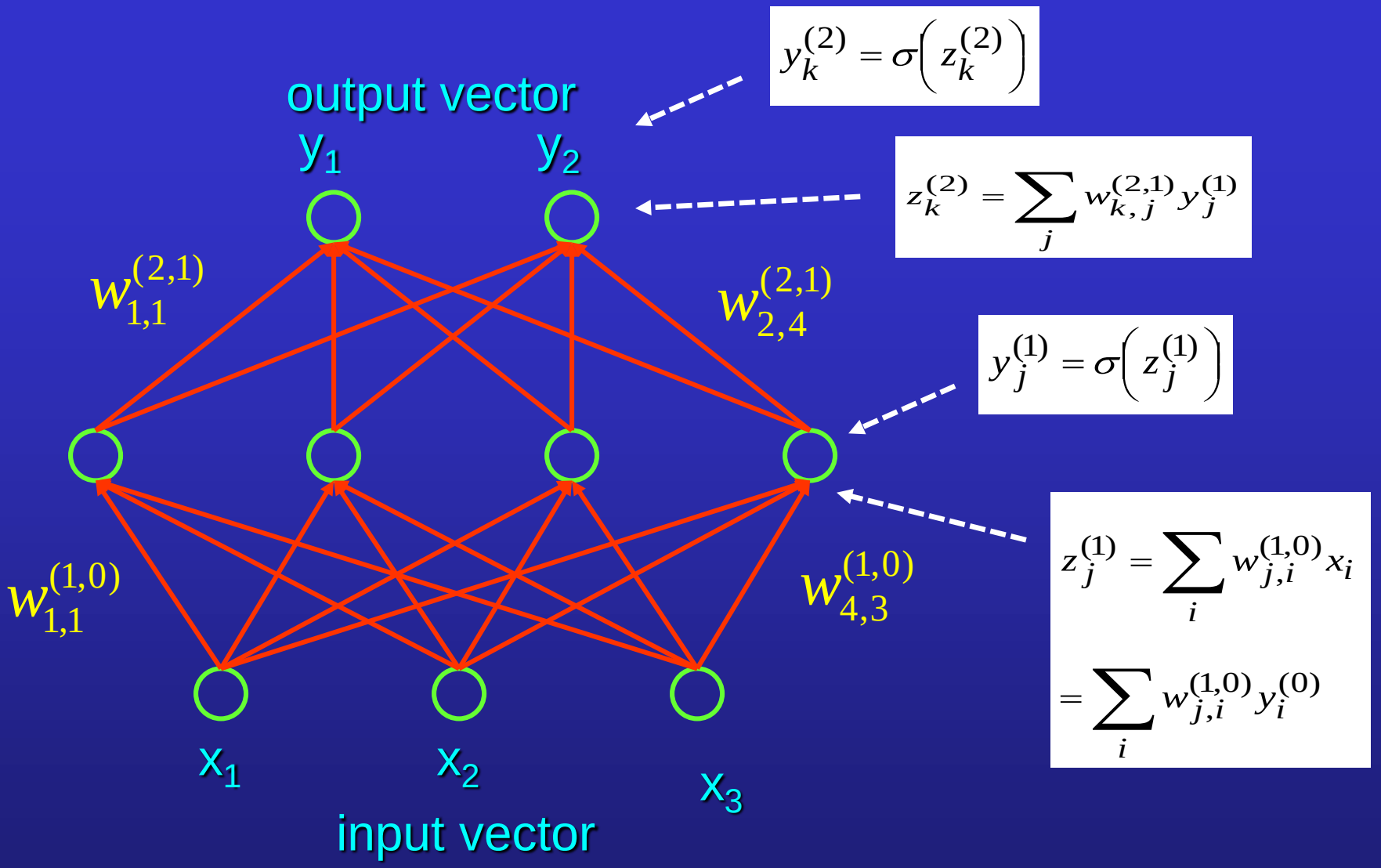
Refers to the method for Gradients estimation

it is NOT an optimization method

Terminology

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



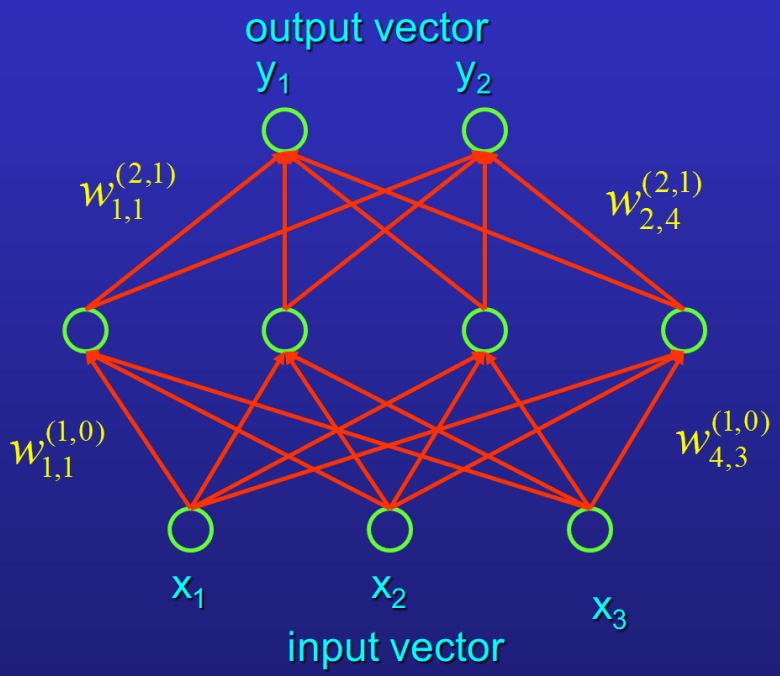

output vector
y_1
y_2
$w_{1,1}^{(2,1)}$
$w_{2,4}^{(2,1)}$
$w_{1,1}^{(1,0)}$
$w_{4,3}^{(1,0)}$
x_1
x_2
x_3
input vector
$y_k^{(2)} = \sigma\left(z_k^{(2)} \right)$
$z_k^{(2)} = \sum_j w_{k,j}^{(2,1)} y_j^{(1)}$
$y_j^{(1)} = \sigma\left(z_j^{(1)} \right)$
$z_j^{(1)} = \sum_i w_{j,i}^{(1,0)} x_i$
$= \sum_i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

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)

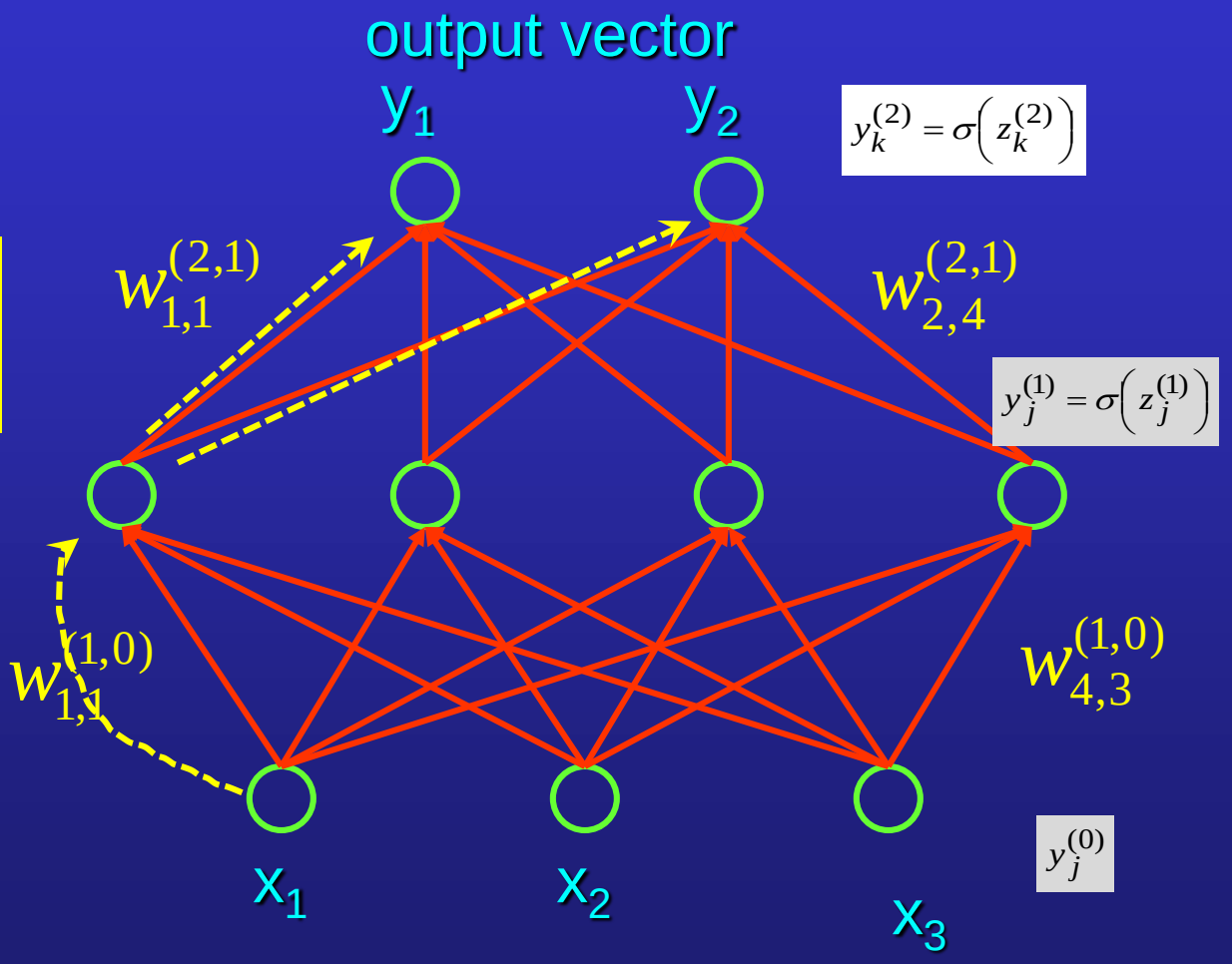
$$E(\theta) = \sum \left(y_k - y_k^{(2)} \right)^2$$

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

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

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

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



Hidden layer :

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

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

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

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

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

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

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

- 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

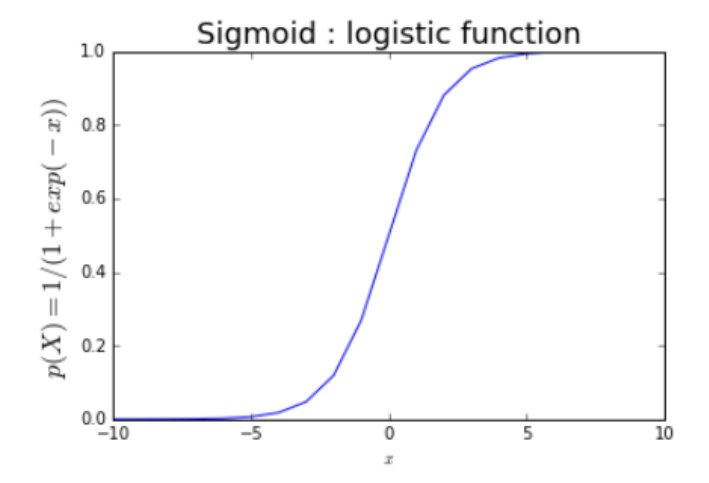
Sigmoidal Neurons

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

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

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

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

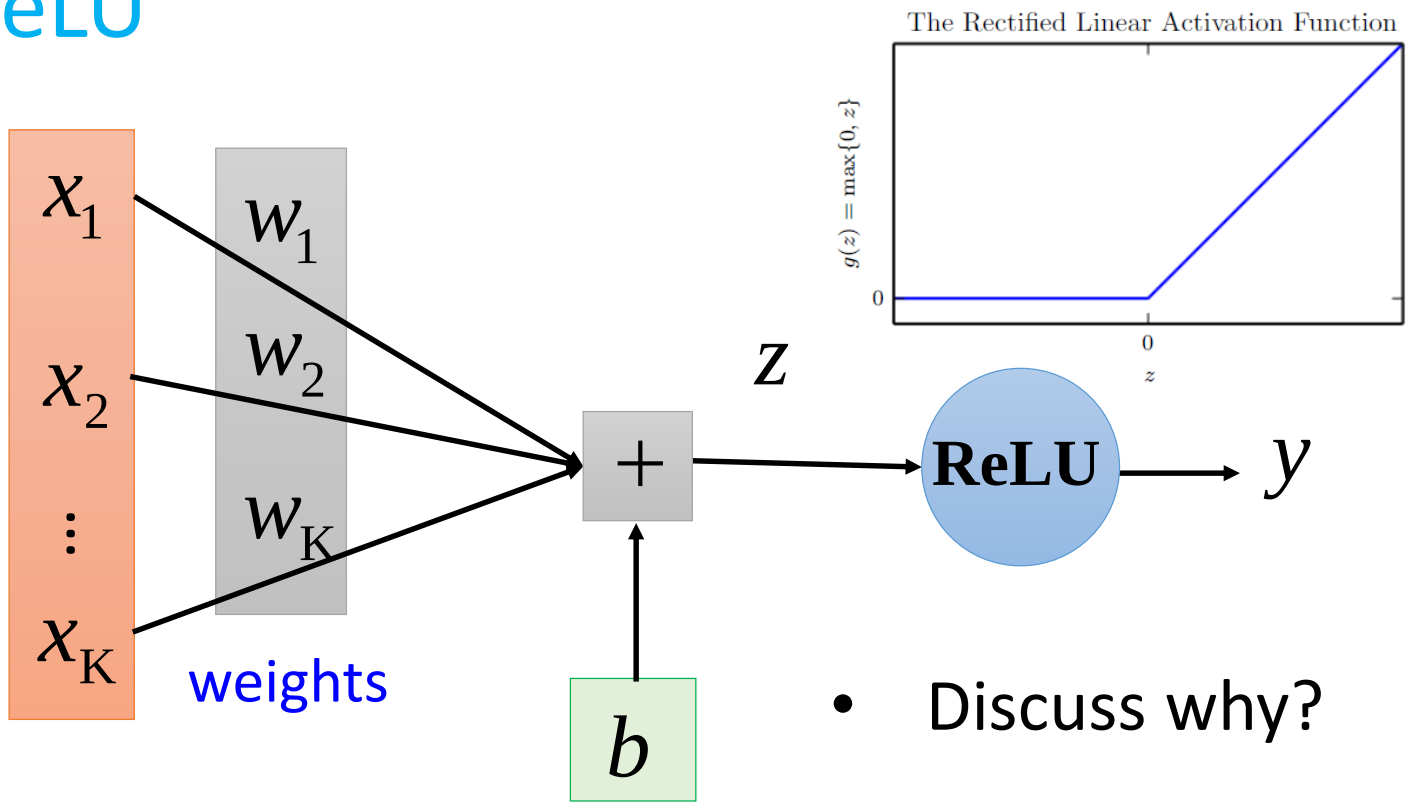


$$\delta_k = (y_k - y_k^{(2)})\sigma'\left(z_k^{(2)}\right)$$

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

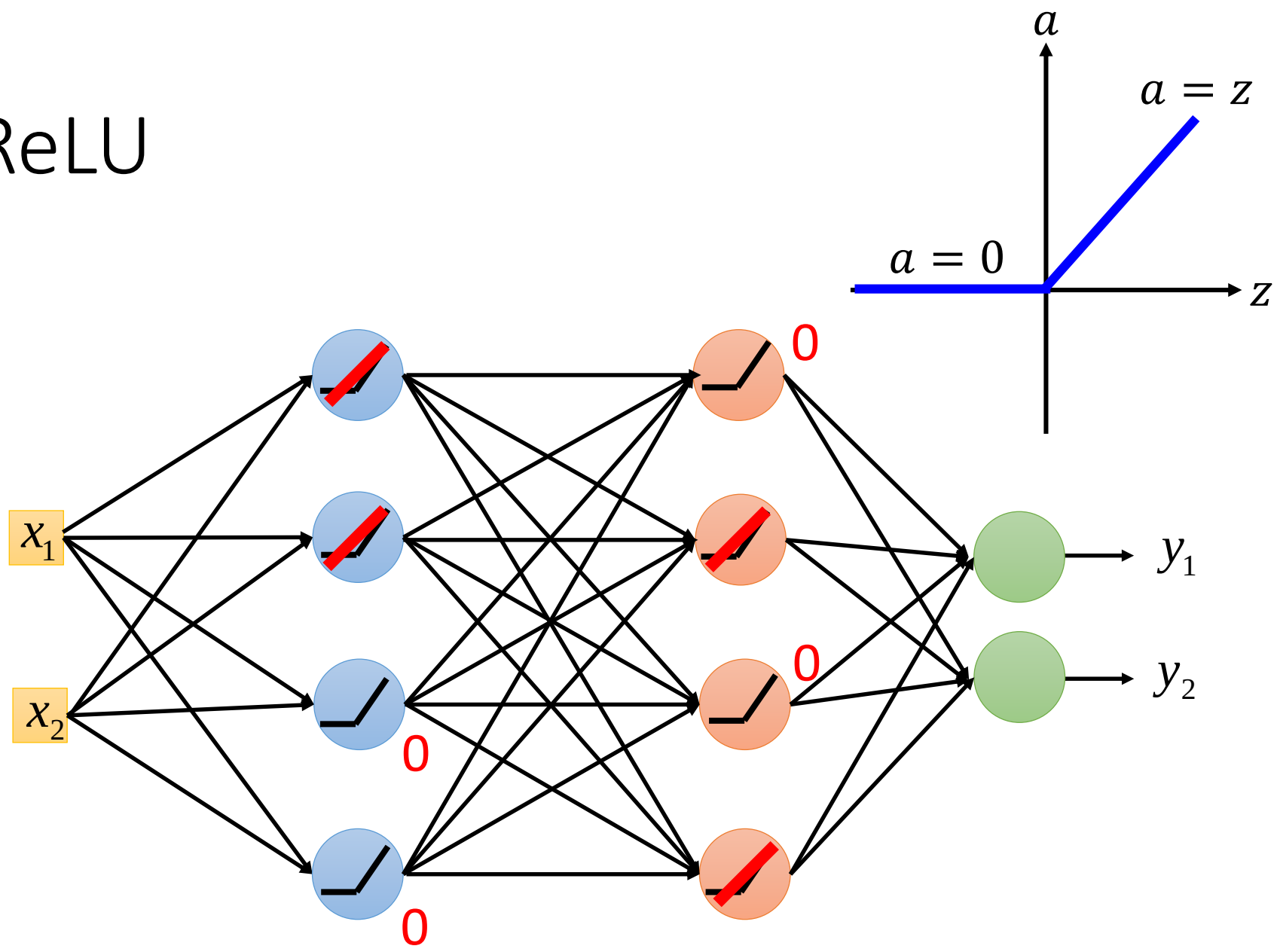
Activation Functions

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

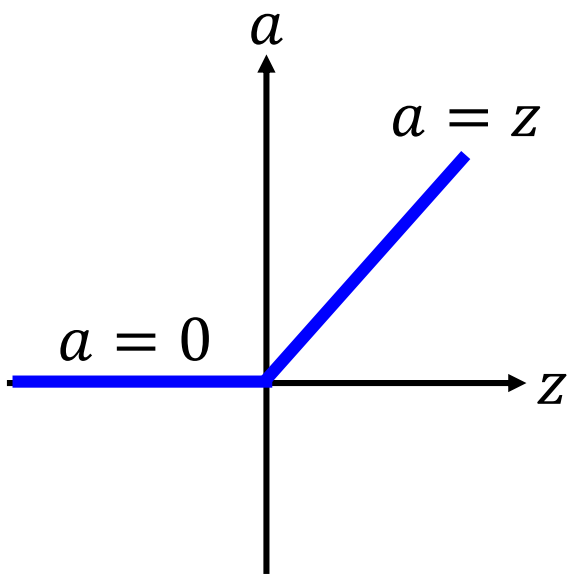


- Discuss why?

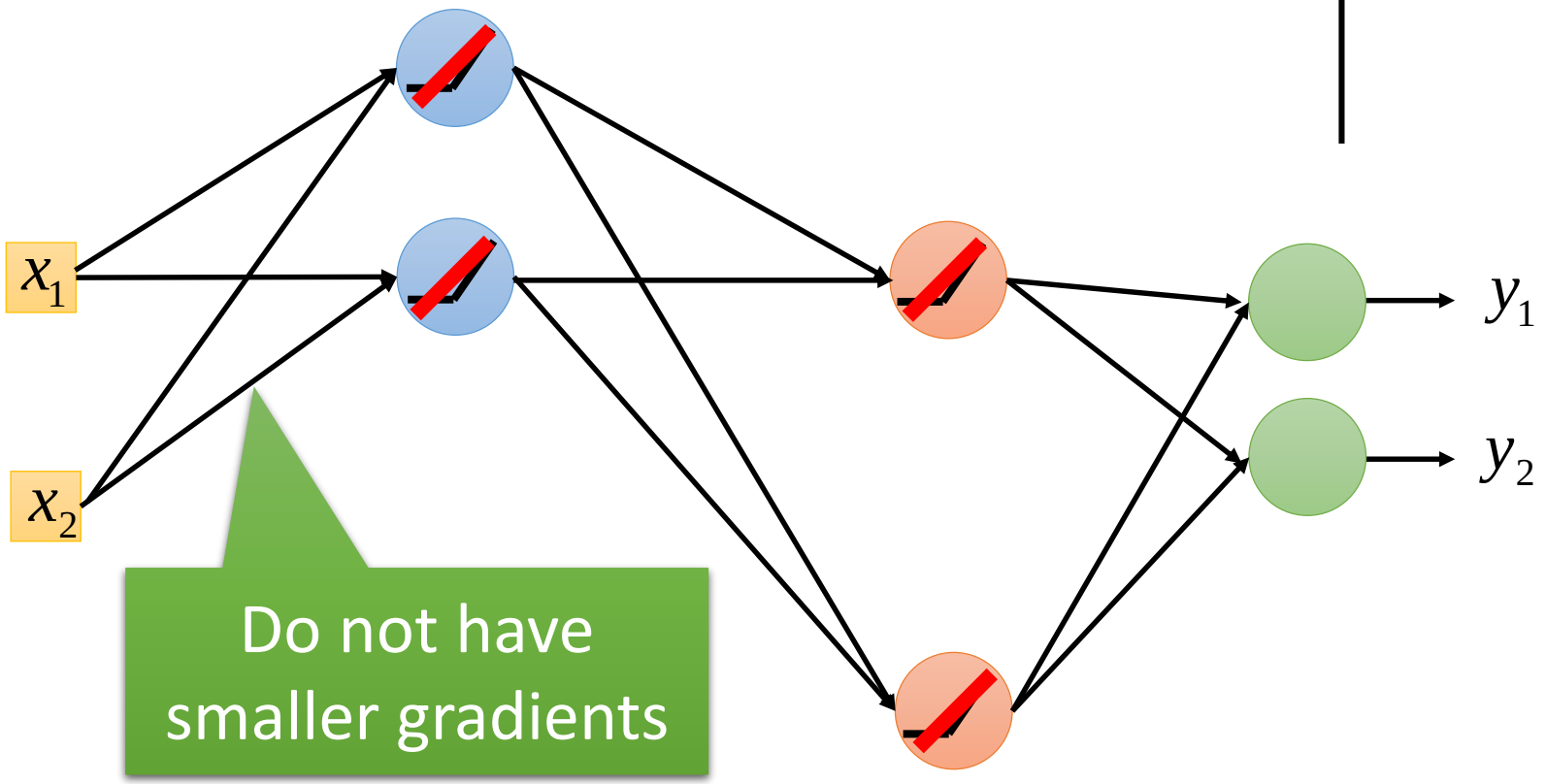
ReLU



ReLU



A Thinner linear network

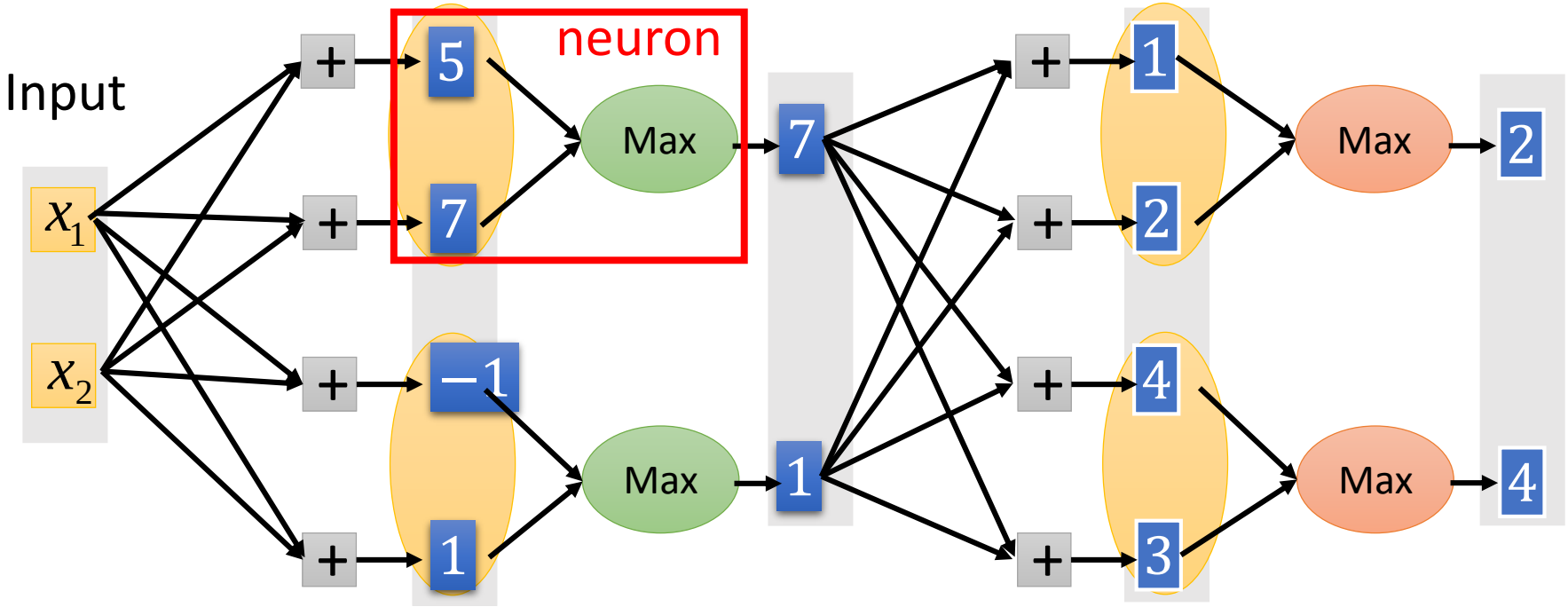


Do not have smaller gradients

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]

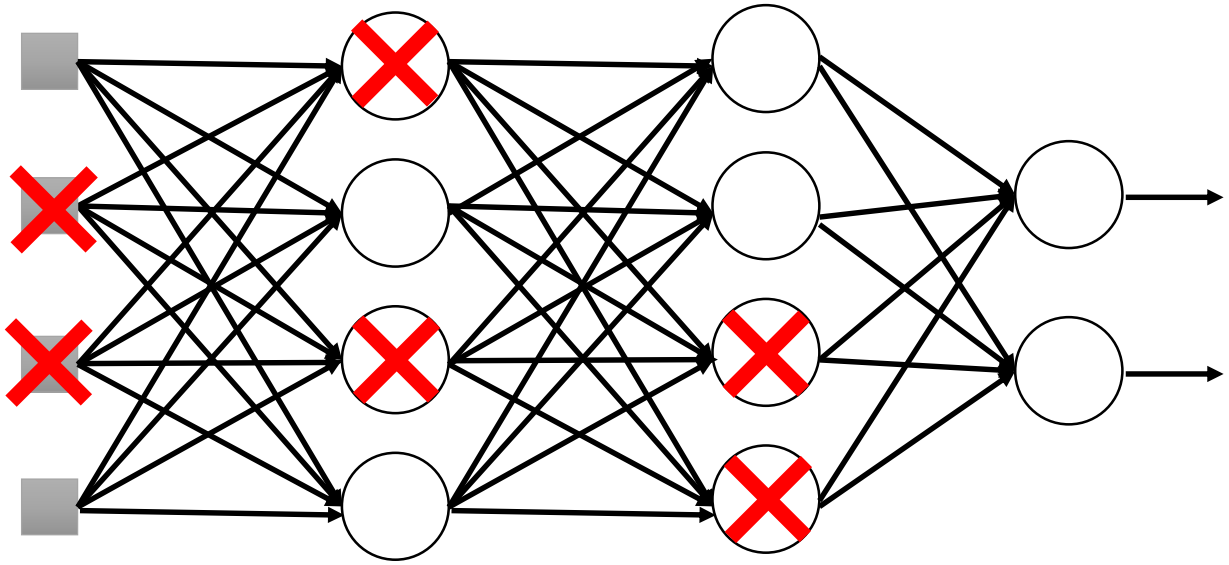
Tips for Training DNN

fighting against overfitting

Dropout

Dropout

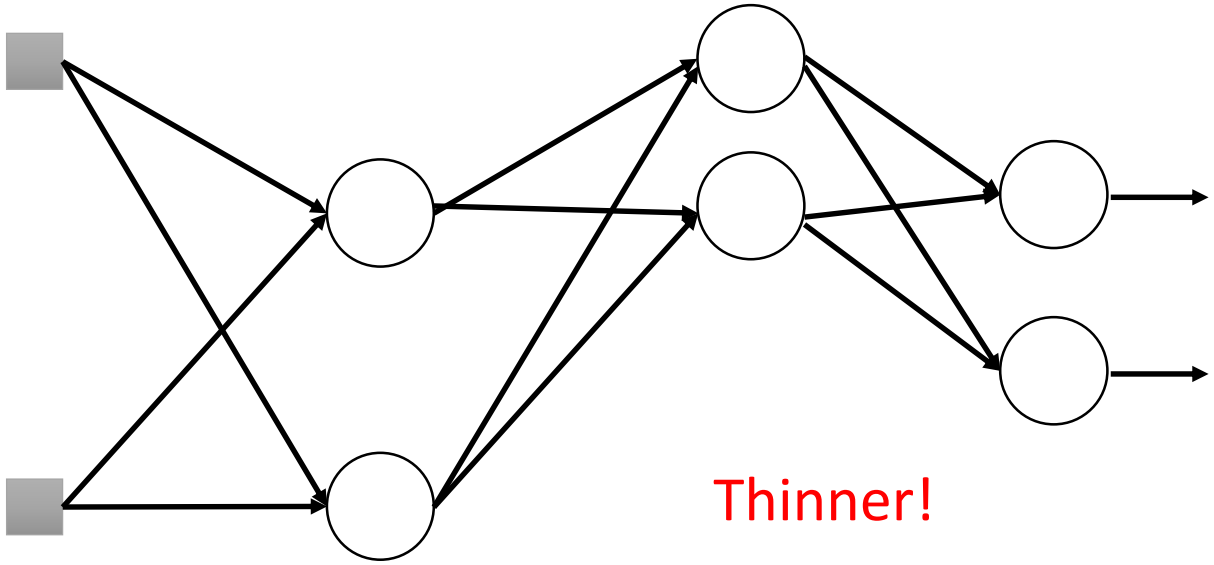
Training:



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

Dropout

Training:



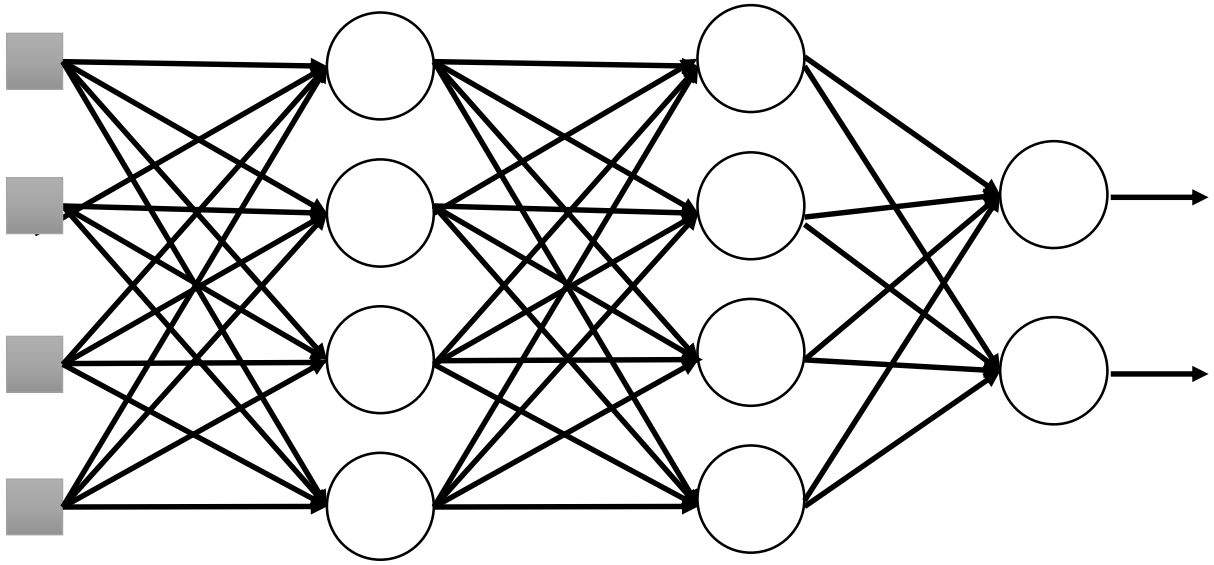
- **Each time before computing the gradients**
  - Each neuron has p% to dropout

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

For each mini-batch, we resample the dropout neurons

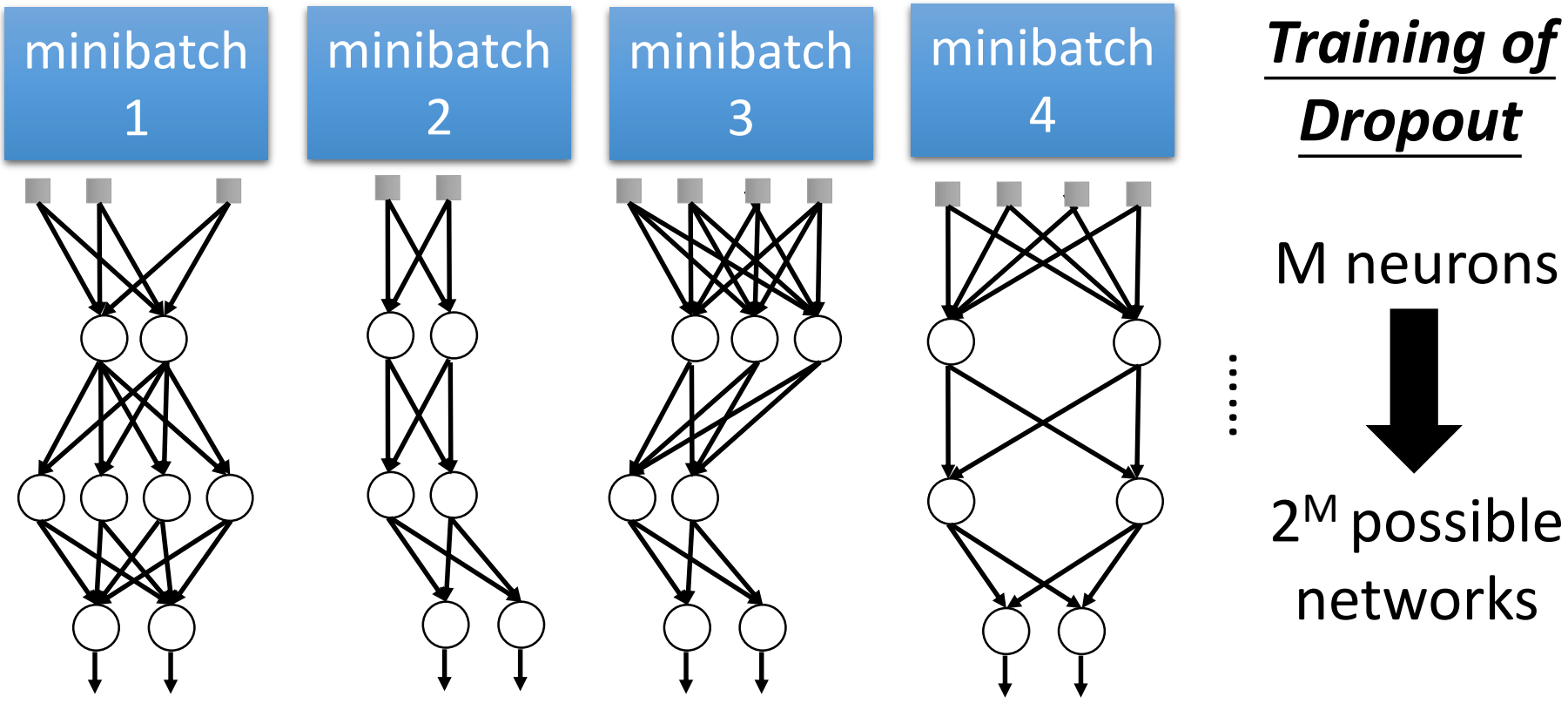
Dropout

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.

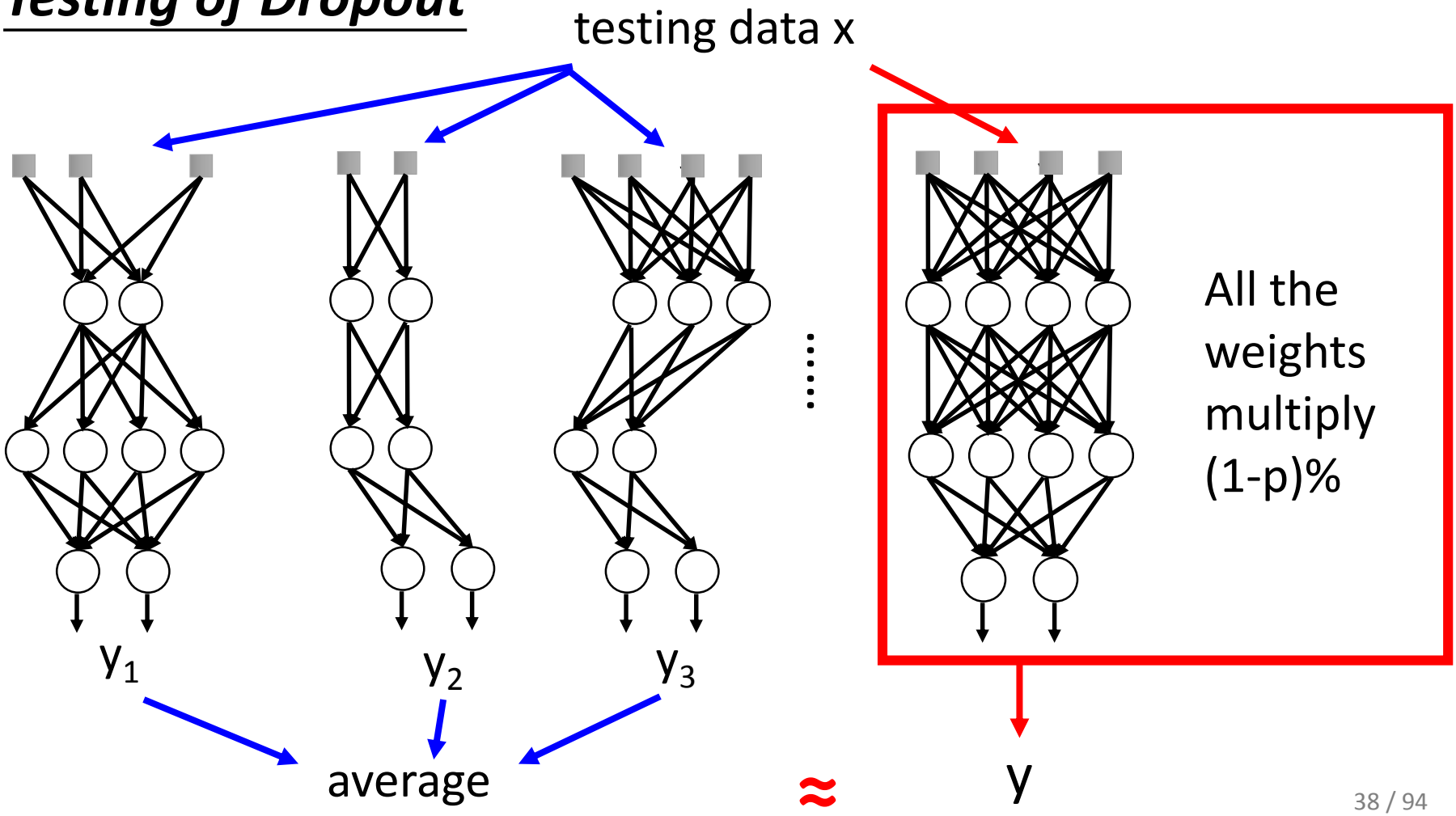
Dropout is a kind of ensemble.



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

Dropout is a kind of ensemble.

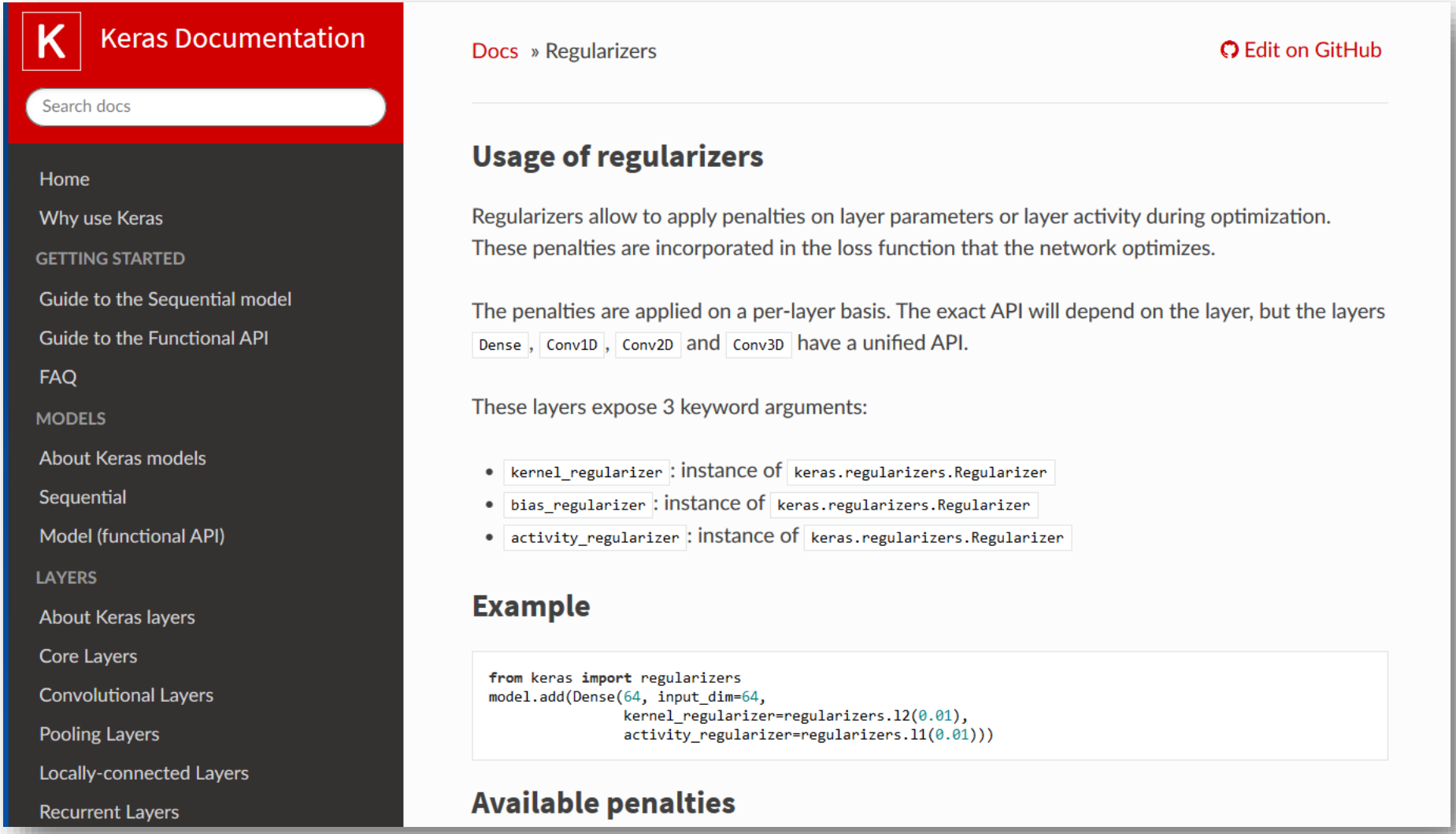
Testing of Dropout



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

Regularizers: *fighting against overfitting*



Keras Documentation

Search docs

Home
Why use Keras
GETTING STARTED
Guide to the Sequential model
Guide to the Functional API
FAQ
MODELS
About Keras models
Sequential
Model (functional API)
LAYERS
About Keras layers
Core Layers
Convolutional Layers
Pooling Layers
Locally-connected Layers
Recurrent Layers

Docs » Regularizers

Edit on GitHub

Usage of regularizers

Regularizers allow to apply penalties on layer parameters or layer activity during optimization. These penalties are incorporated in the loss function that the network optimizes.

The penalties are applied on a per-layer basis. The exact API will depend on the layer, but the layers `Dense`, `Conv1D`, `Conv2D` and `Conv3D` have a unified API.

These layers expose 3 keyword arguments:

- `kernel_regularizer`: instance of `keras.regularizers.Regularizer`
- `bias_regularizer`: instance of `keras.regularizers.Regularizer`
- `activity_regularizer`: instance of `keras.regularizers.Regularizer`

Example

```
from keras import regularizers
model.add(Dense(64, input_dim=64,
                kernel_regularizer=regularizers.l2(0.01),
                activity_regularizer=regularizers.l1(0.01)))
```

Available penalties

iptc
Information Processing and Telecommunications Center

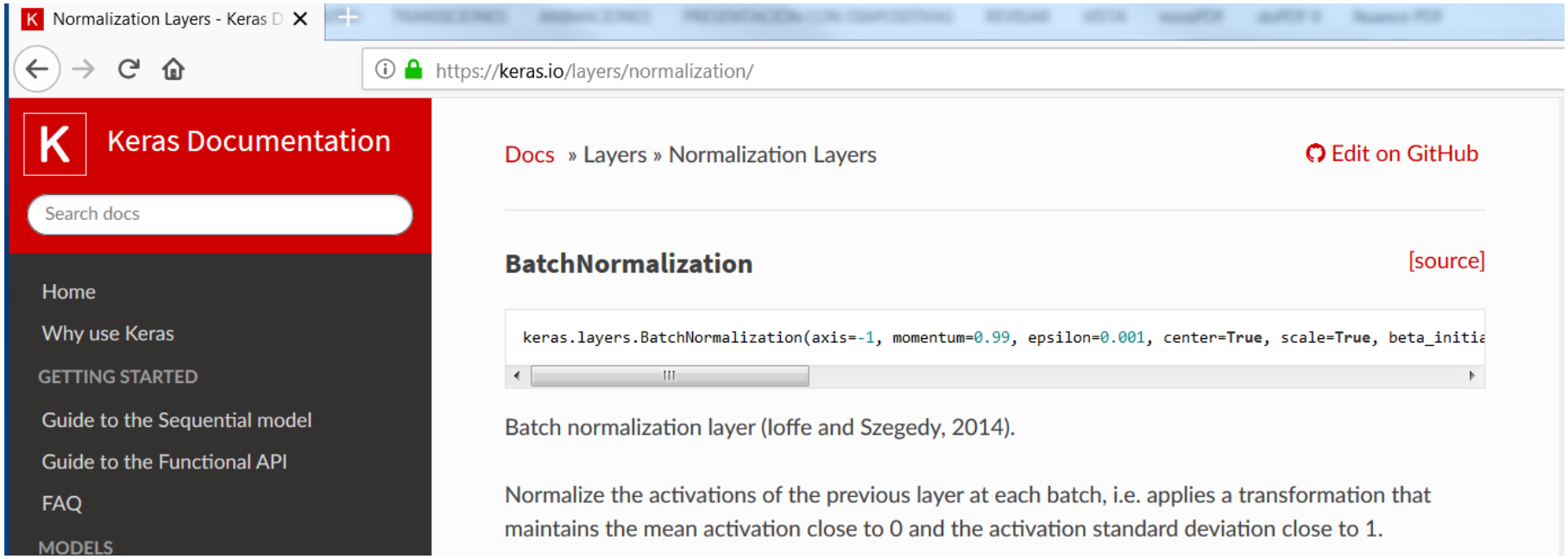
Batch normalization

Benefits of Batch Normalization

- Networks train faster converge much more quickly,
- Allows higher learning rates. Gradient descent usually requires small learning rates for the network to converge.
- Makes weights easier to initialize
- Makes more activation functions viable. Because batch normalization regulates the values going into each activation function, non-linearities that don't seem to work well in deep networks actually become viable again.
- May give better results overall.

https://www.dlology.com/blog/one-simple-trick-to-train-keras-model-faster-with-batch-normalization/

Batch normalization



Enabled Keras model with Batch Normalization

Dense layer

A normal **Dense** fully connected layer looks like this

```
model.add(layers.Dense(64, activation='relu'))
```

To make it Batch normalization enabled, we have to tell the Dense layer not using bias since it is not needed, it can save some calculation. Also, put the Activation layer after the `BatchNormalization()` layer

```
model.add(layers.Dense(64, use_bias=False))
model.add(layers.BatchNormalization())
model.add(Activation("relu"))
```

Conv2D layer

A normal Keras **Conv2D** layer can be defined as

```
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

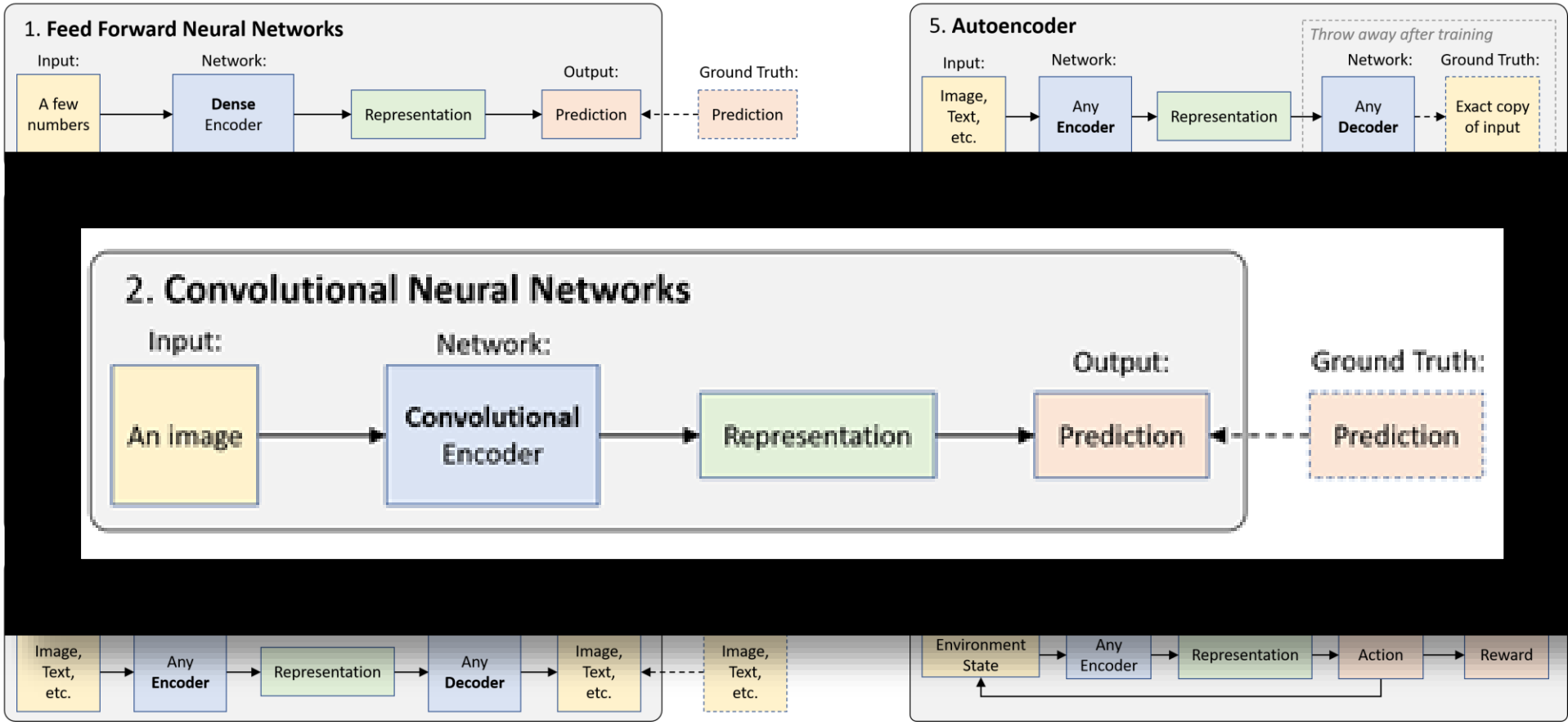
Turing it to Batch normalized Conv2D layer, we add the `BatchNormalization()` layer similar to Dense layer above

```
model.add(layers.Conv2D(64, (3, 3), use_bias=False))
model.add(layers.BatchNormalization())
model.add(layers.Activation("relu"))
```

Deep learning basics

Supervised Learning

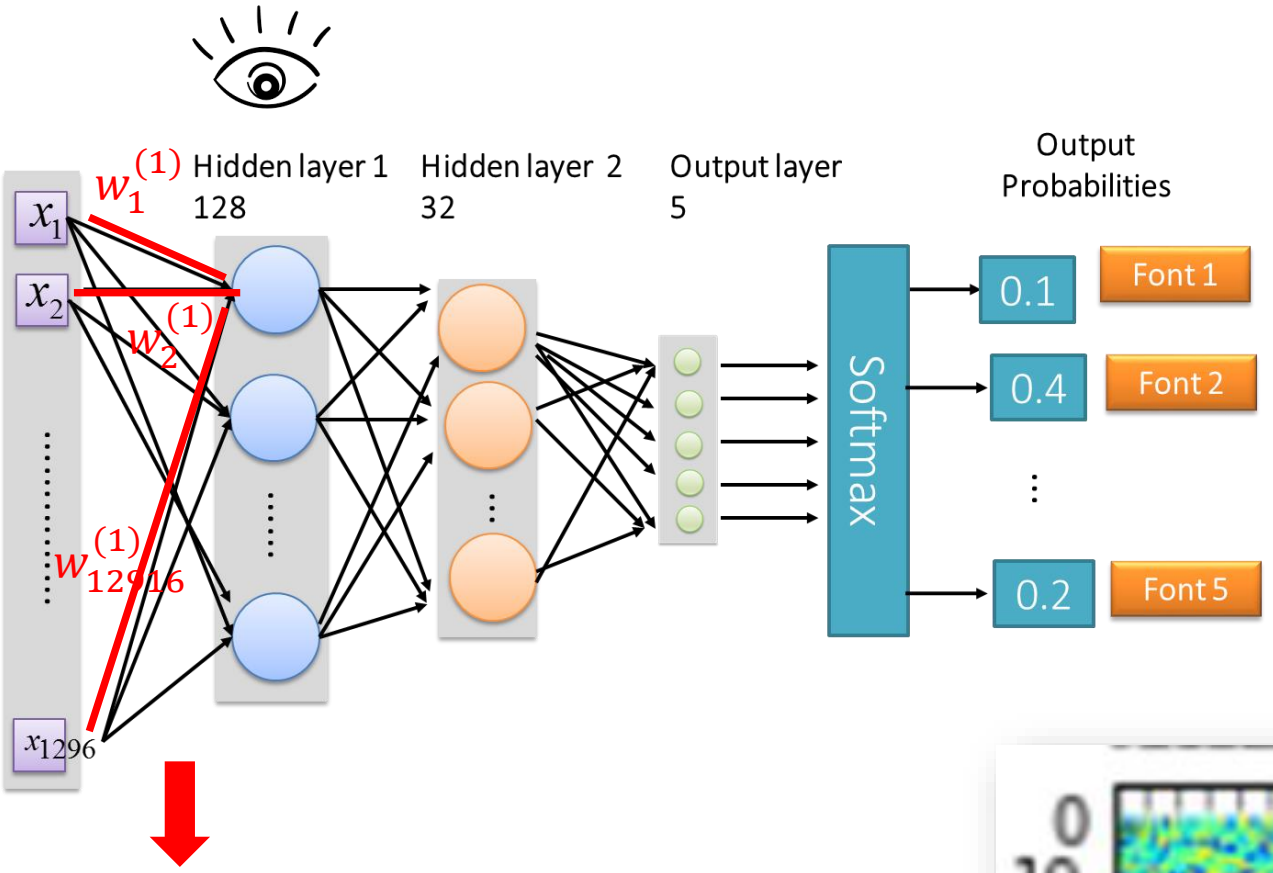
Unsupervised Learning



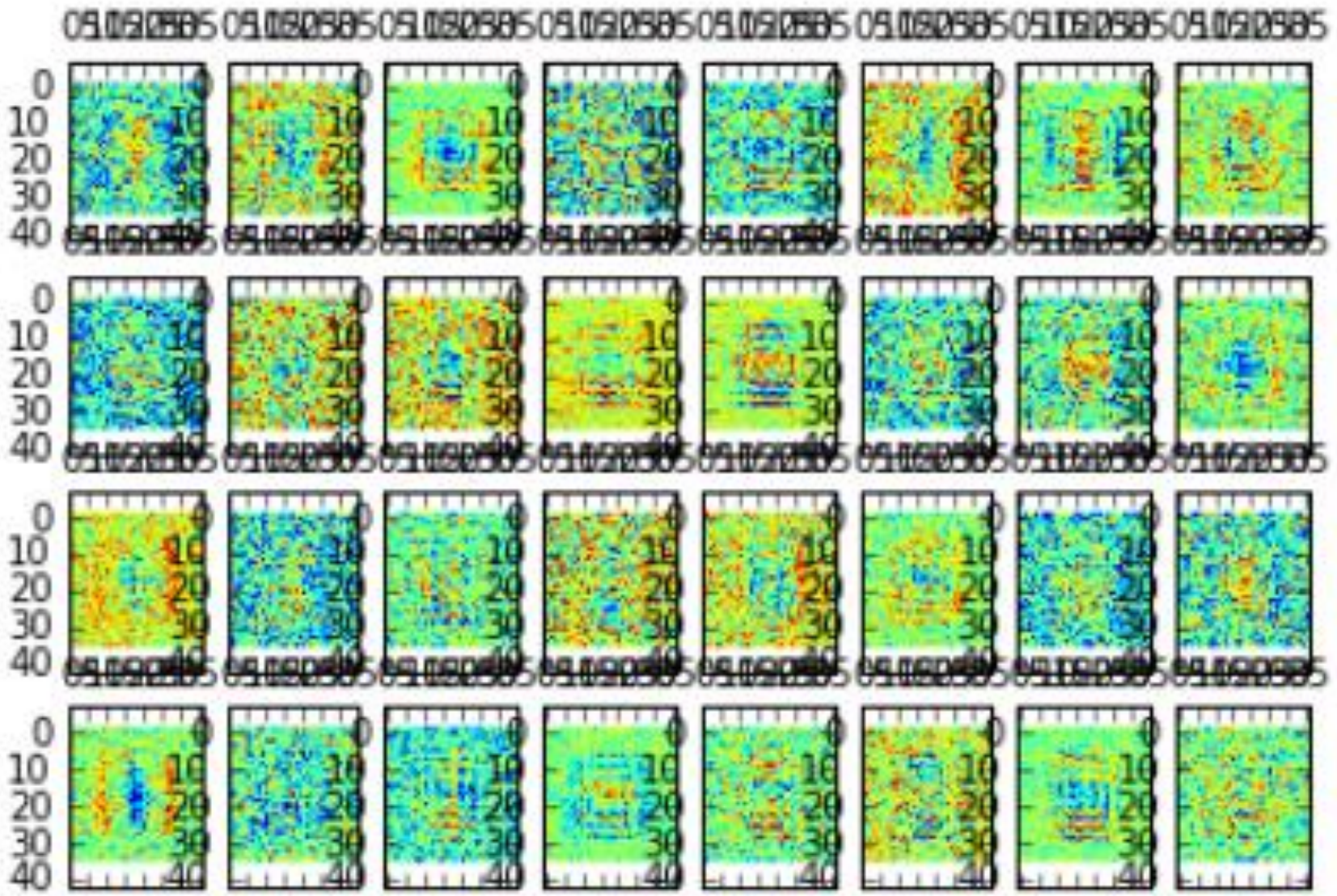
https://github.com/lexfridman/mit-deep-learning/blob/master/tutorial_deep_learning_basics/deep_learning_basics.ipynb

Try to INTERPRET the Feed Forward Architechure:

- Visualize the first hidden layer



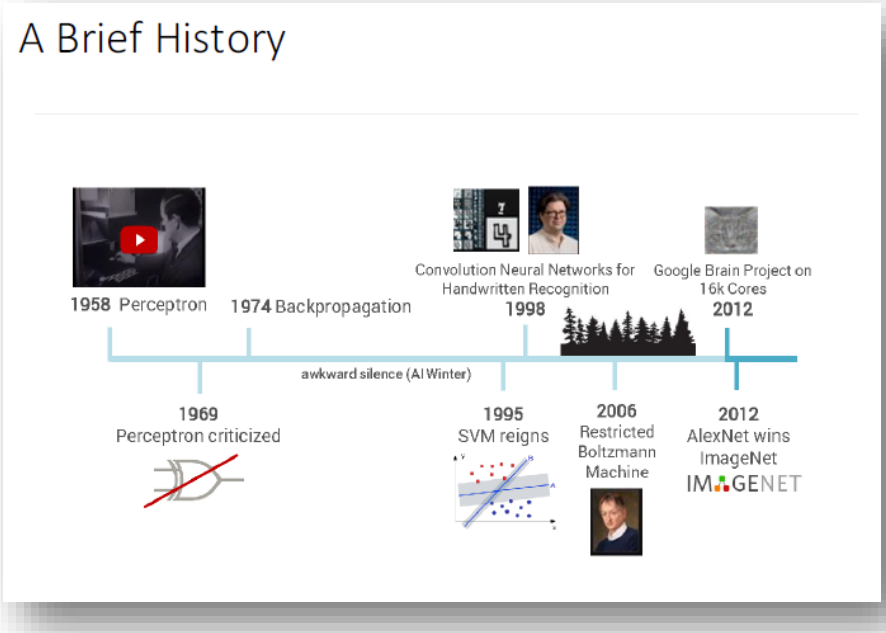
32 First hidden layers



CNN / ConvNets: Convolutional Neural Network

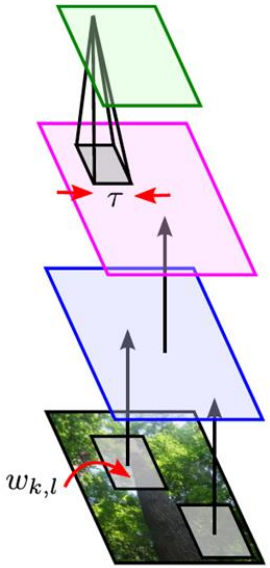
organizes neurons based on animal's visual cortex system, which allows for learning patterns at both local level and global level.

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**

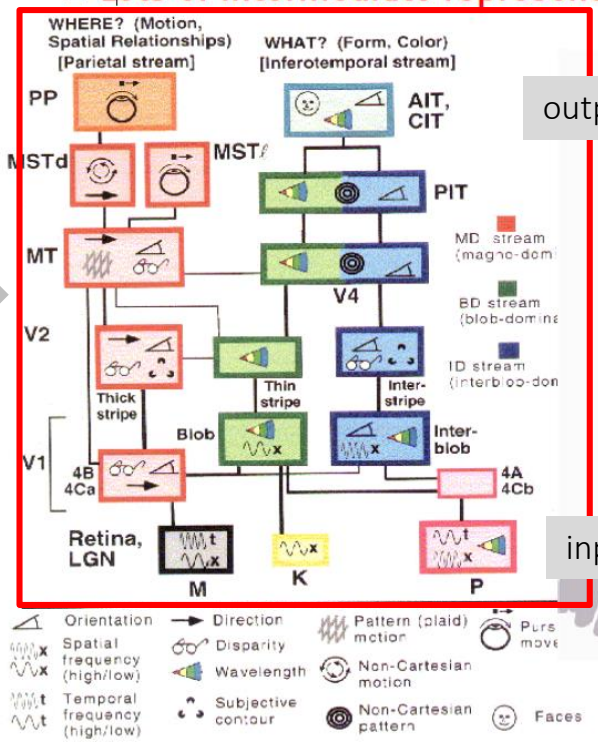


The Mammalian Visual Cortex Inspires CNN

Convolutional Neural Net

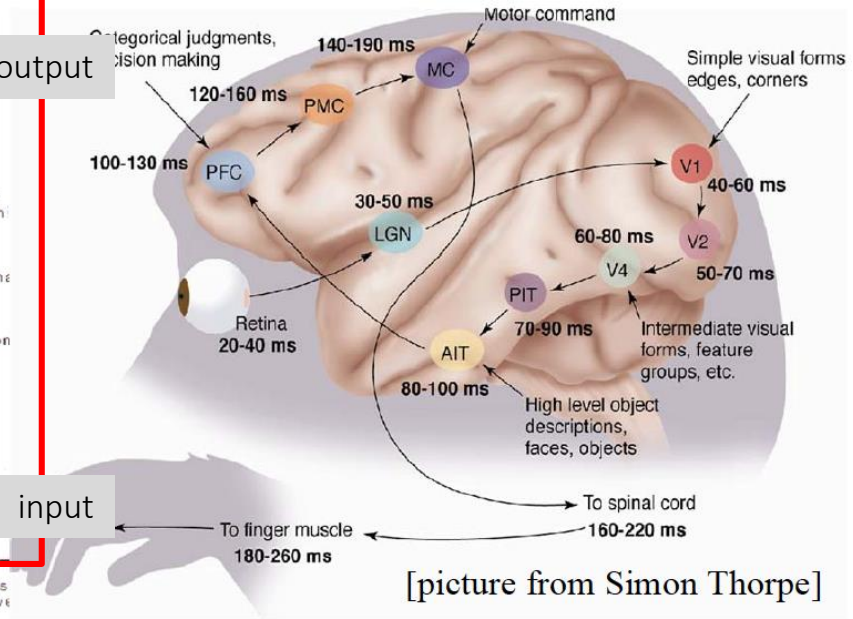


- The ventral (recognition) pathway in the visual cortex has multiple stages
- Retina - LGN - V1 - V2 - V4 - PIT - AIT ....
- Lots of intermediate representations



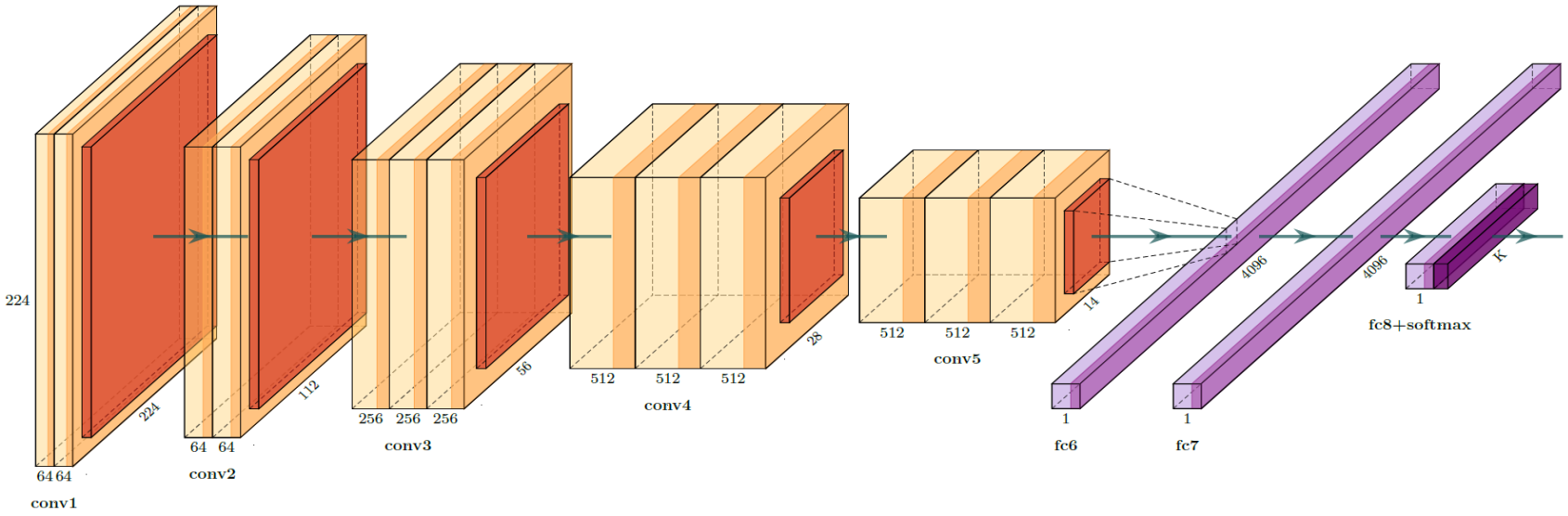
output

input



[Gallant & Van Essen]

CNN as : Feature extraction + Classification (or Regression)

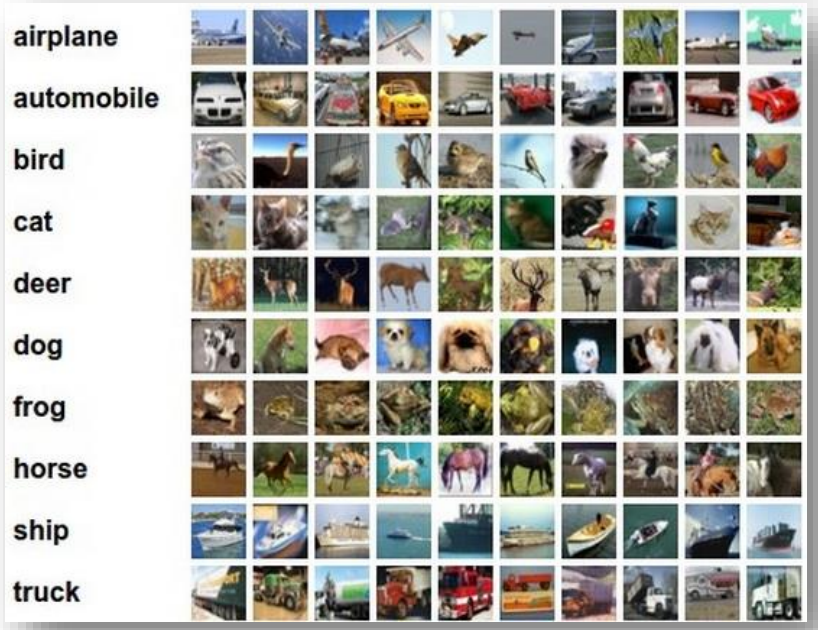


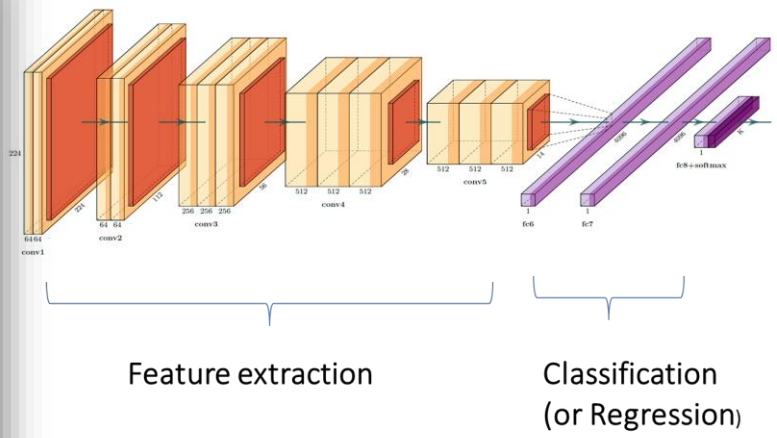
Feature extraction

Classification
(or Regression)

CNN as : Feature extraction + Classification (or Regression)

… but what about the
Input?
… images?





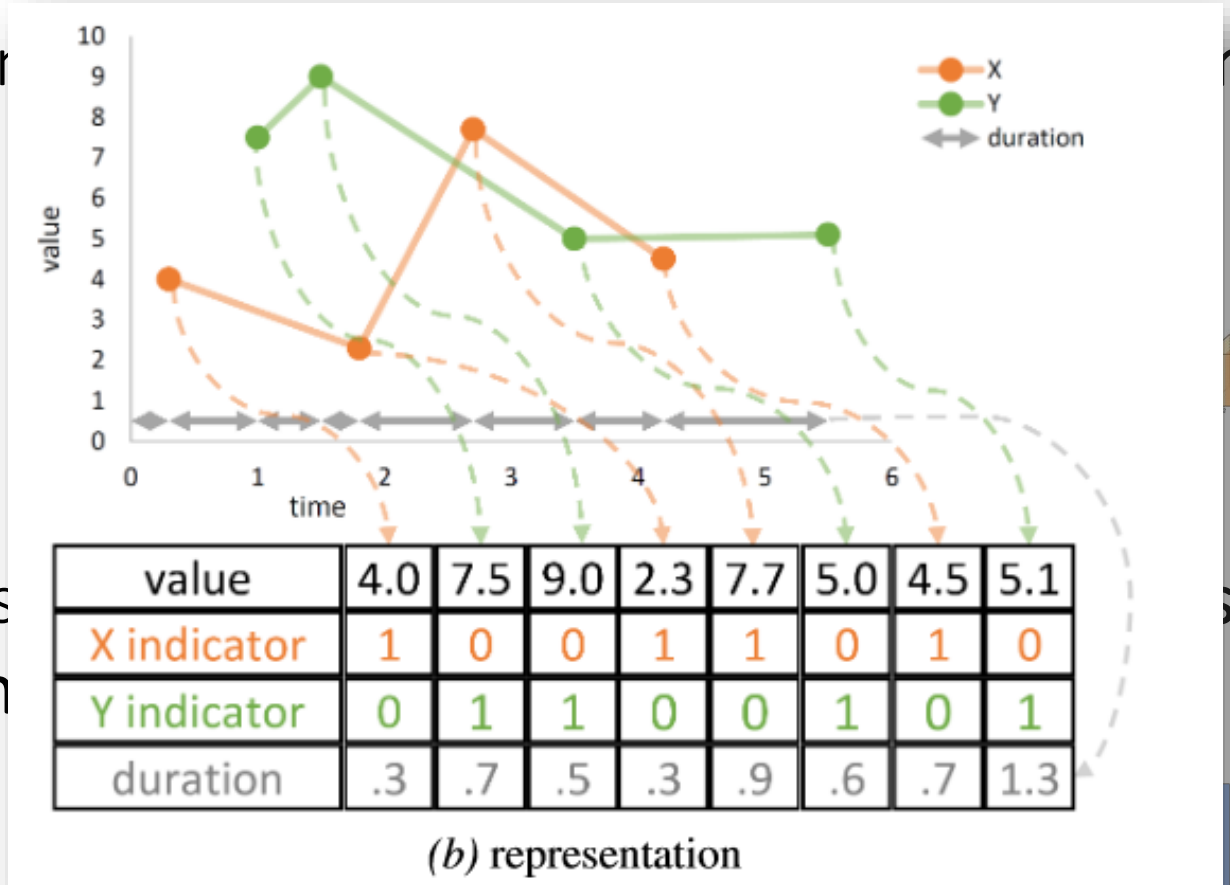
… time series, signals,…?

1D, 2D, 3D CNN

… time series, signals,…?

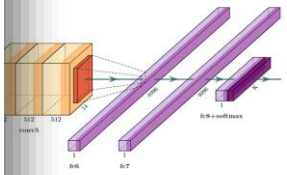
- 2D Co… …n-like inputs

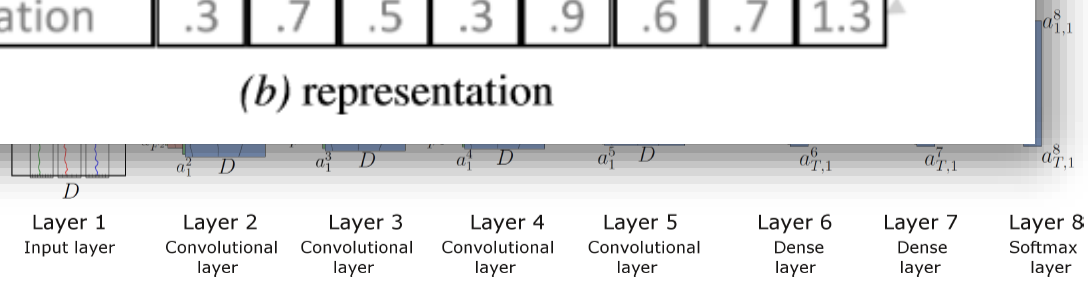
- But als… …series or sign…



value	4.0	7.5	9.0	2.3	7.7	5.0	4.5	5.1
X indicator	1	0	0	1	1	0	1	0
Y indicator	0	1	1	0	0	1	0	1
duration	.3	.7	.5	.3	.9	.6	.7	1.3

(b) representation

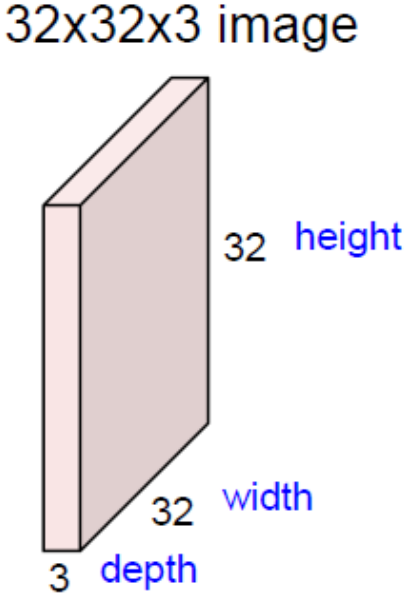




Convolutional Neural Networks

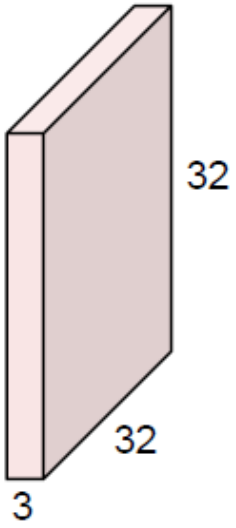
(First without the brain stuff)

Convolution Layer

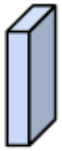


Convolution Layer

32x32x3 image

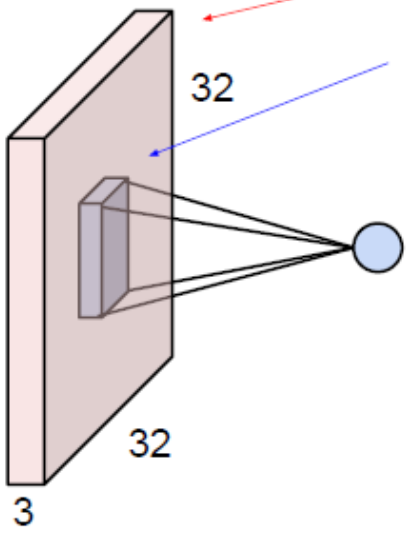


5x5x3 filter



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

Convolution Layer



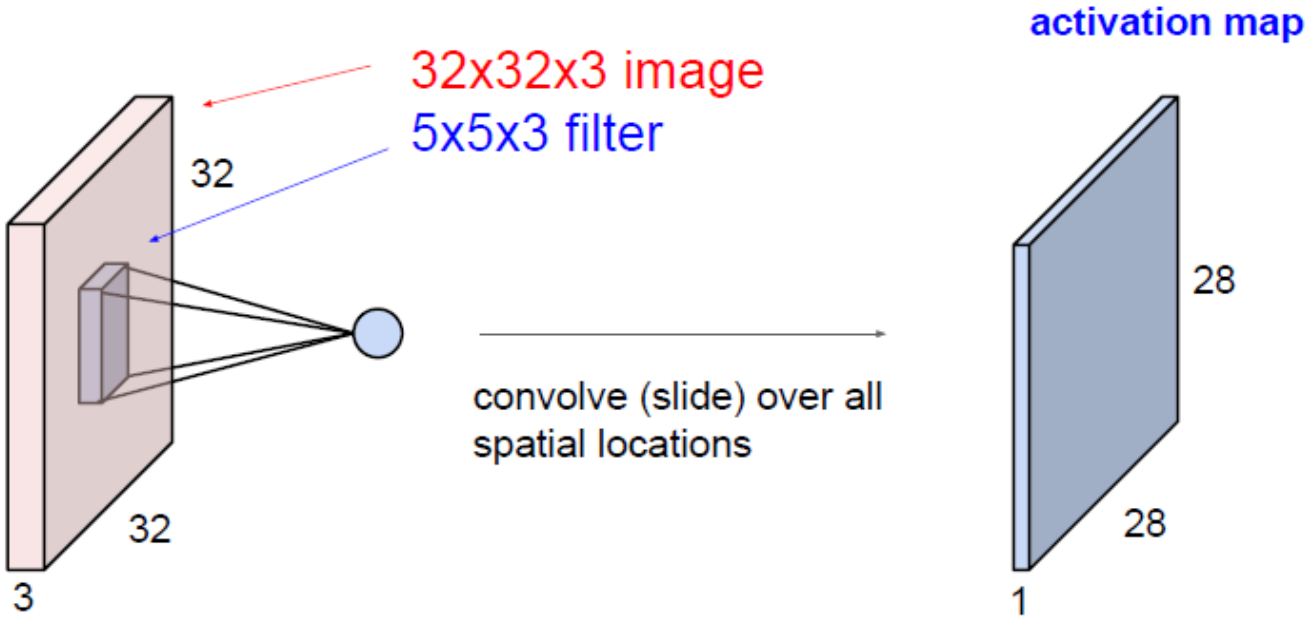
32x32x3 image

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)

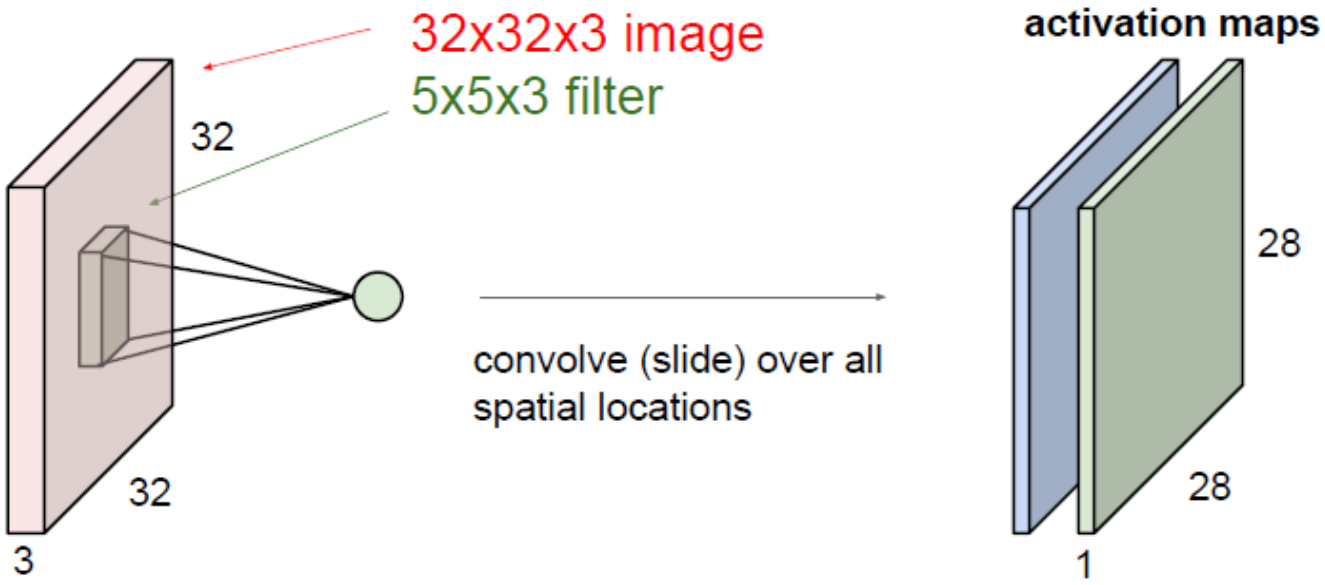
$$w^T x + b$$

Convolution Layer

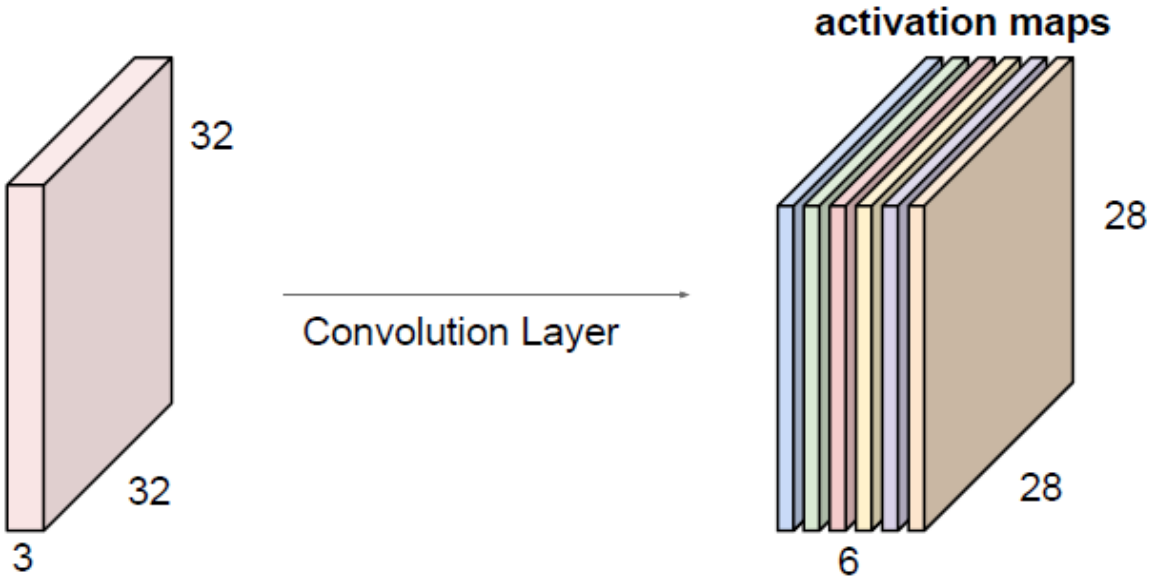


Convolution Layer

consider a second, green filter



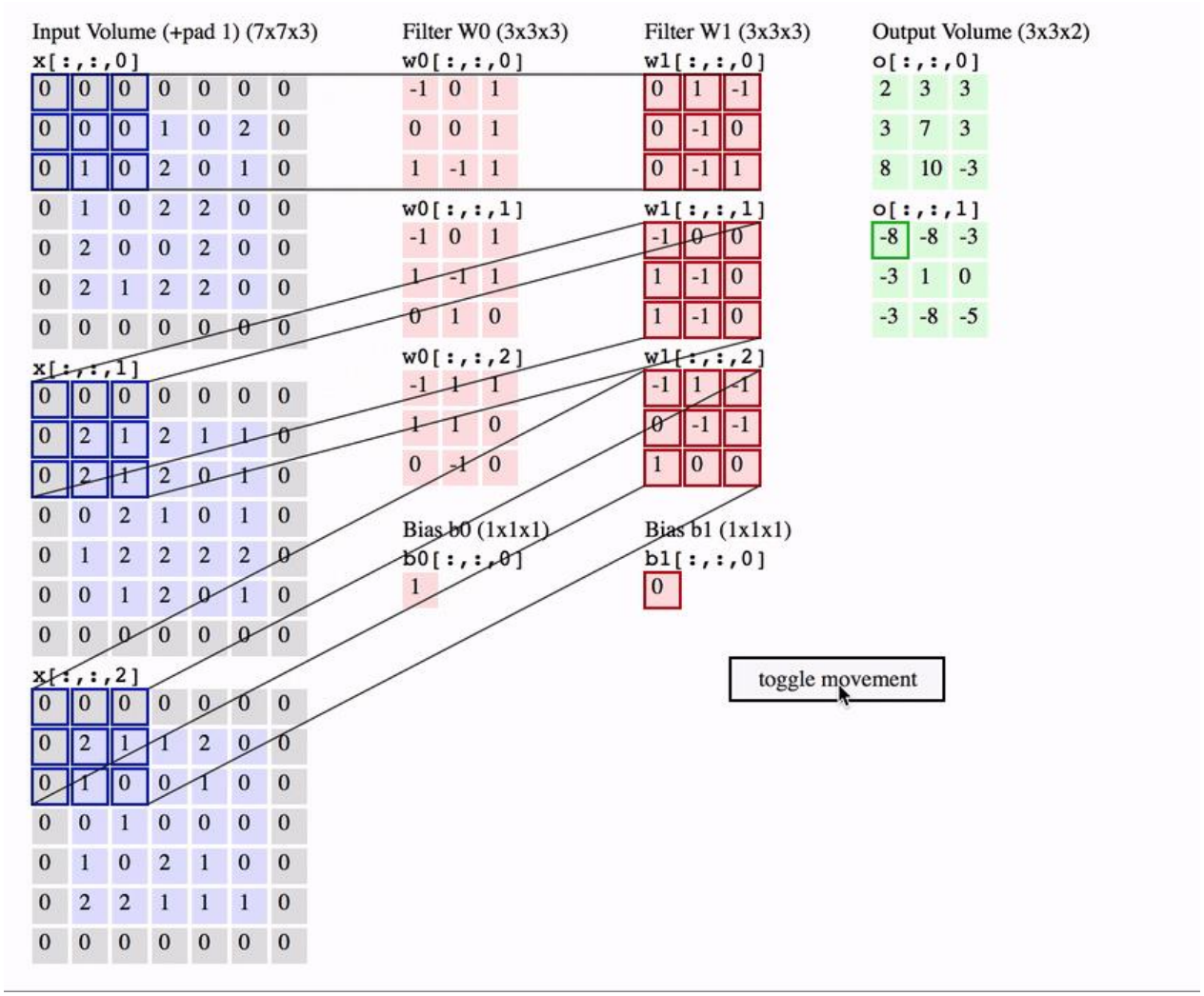
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



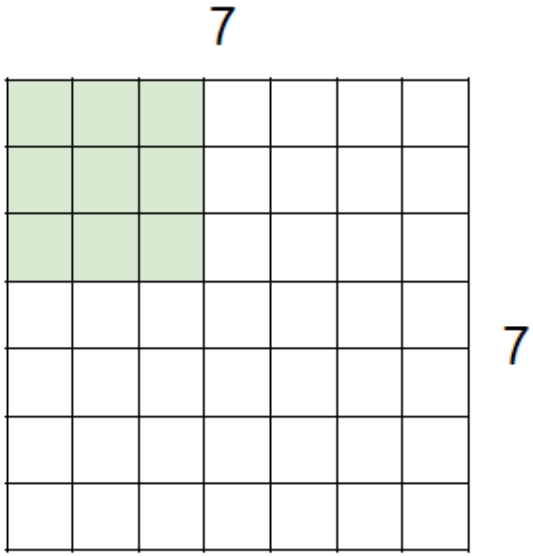
We stack these up to get a "new image" of size 28x28x6!

CNN nice visualization

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

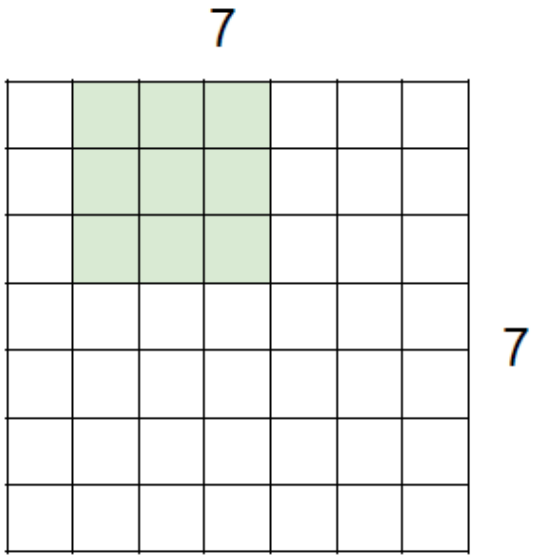


Stride and Padding



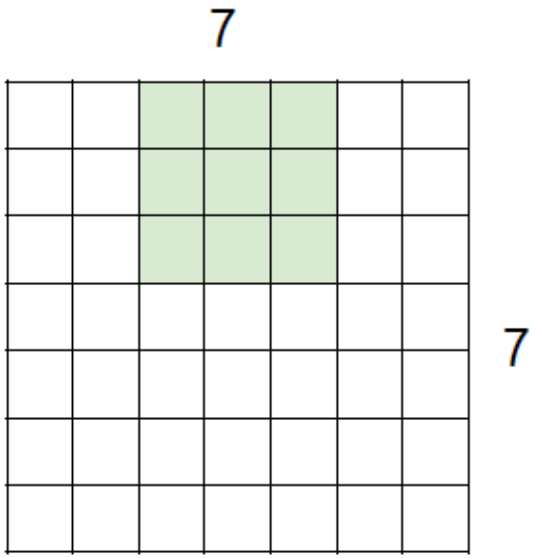
7x7 input (spatially)
assume 3x3 filter

Stride and Padding



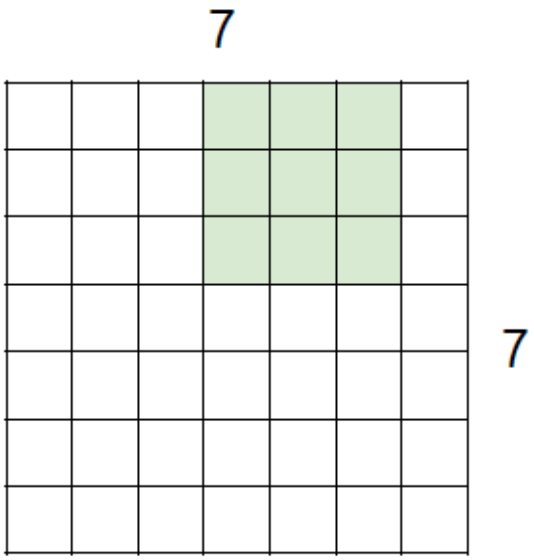
7x7 input (spatially)
assume 3x3 filter

Stride and Padding



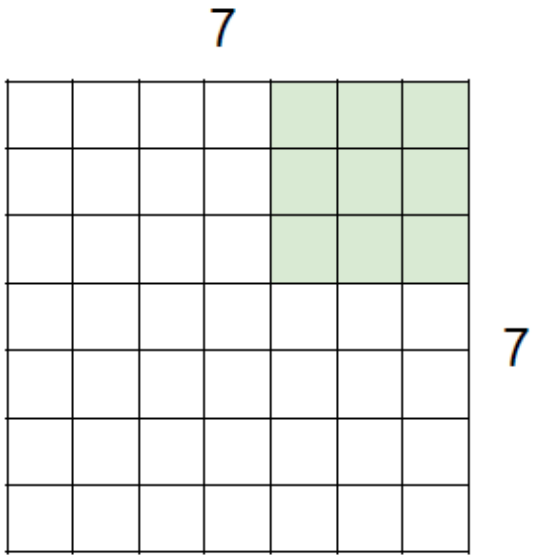
7x7 input (spatially)
assume 3x3 filter

Stride and Padding



7x7 input (spatially)
assume 3x3 filter

Stride and Padding



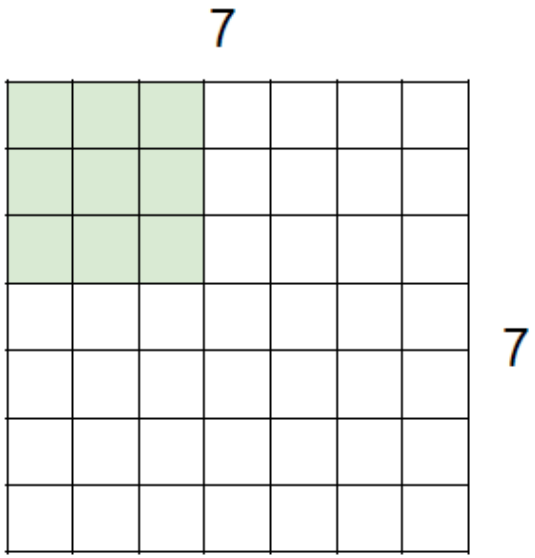
Stride = 1

Padding = 0

7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

Stride and Padding

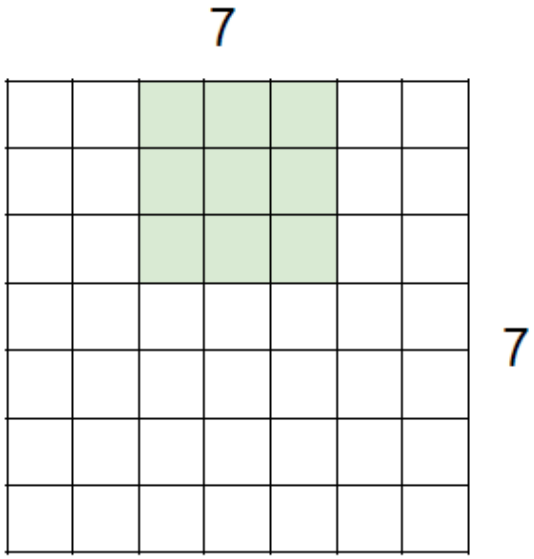


Stride = 2

Padding = 0

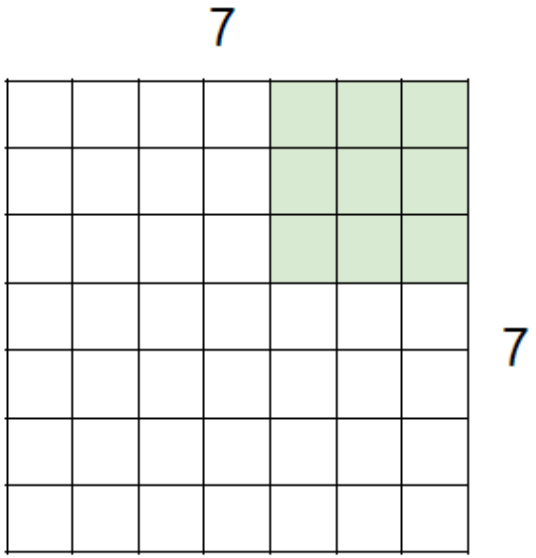
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Stride and Padding



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

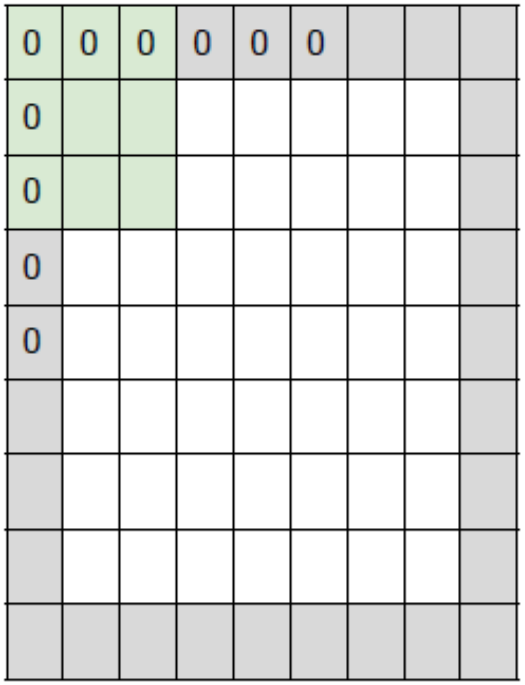
Stride and Padding



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

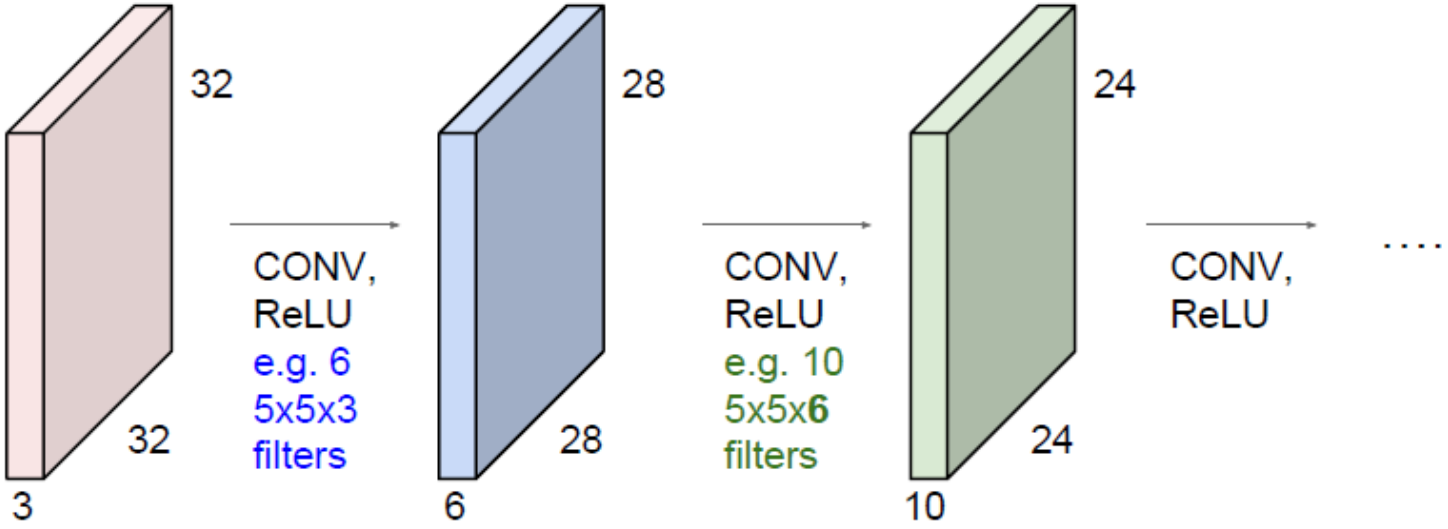
$n \times n$ image $\quad f \times f$ filter

padding p $\quad$ stride s

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

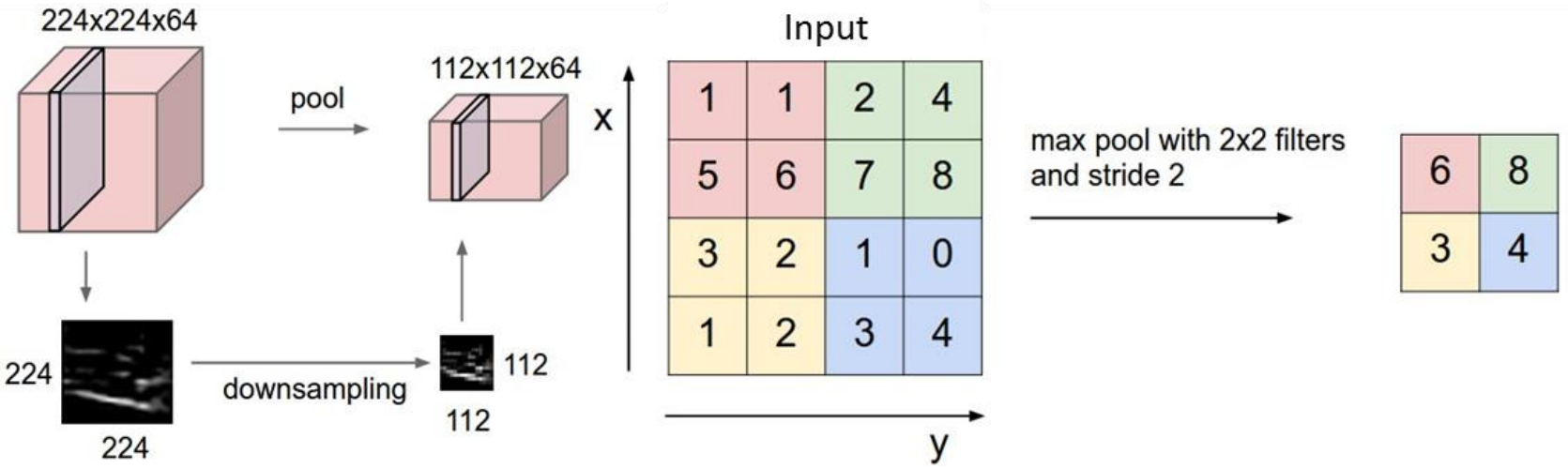
From: Andrew Ng

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



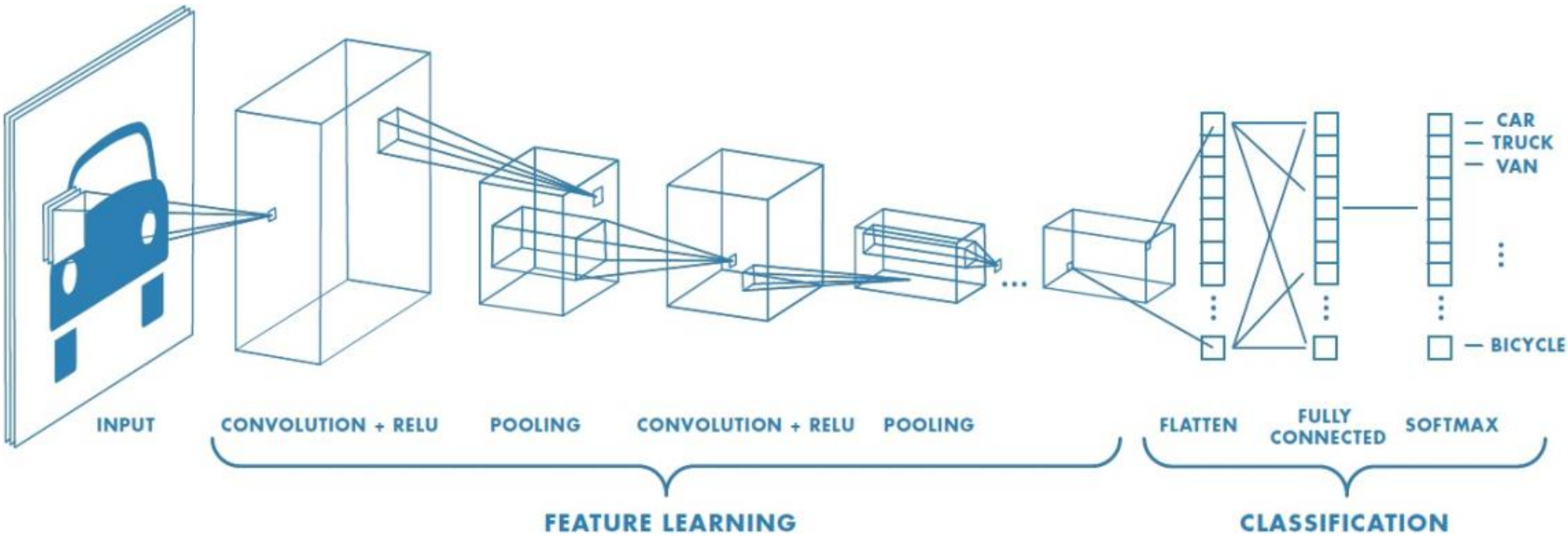
CNN Architecture: Pooling Layer

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



Fei-Fei Li & Andrej Karpathy & Justin Johnson

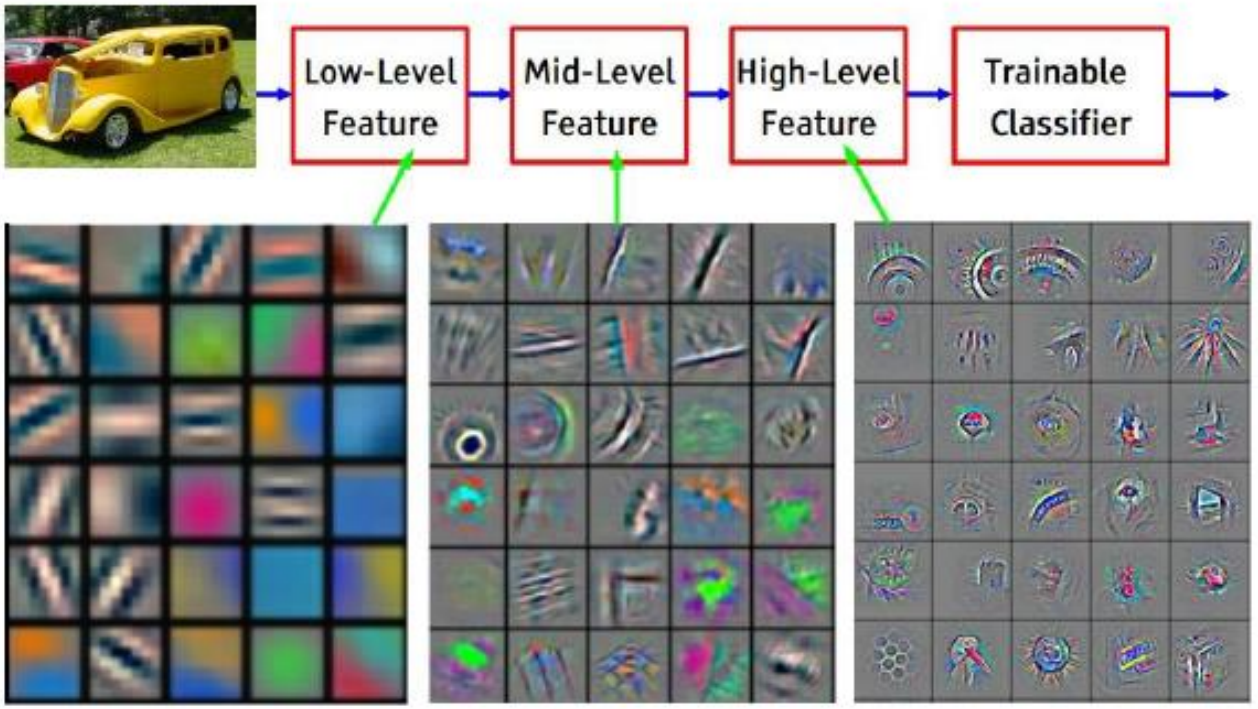
CNN / ConvNet Architecture:



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

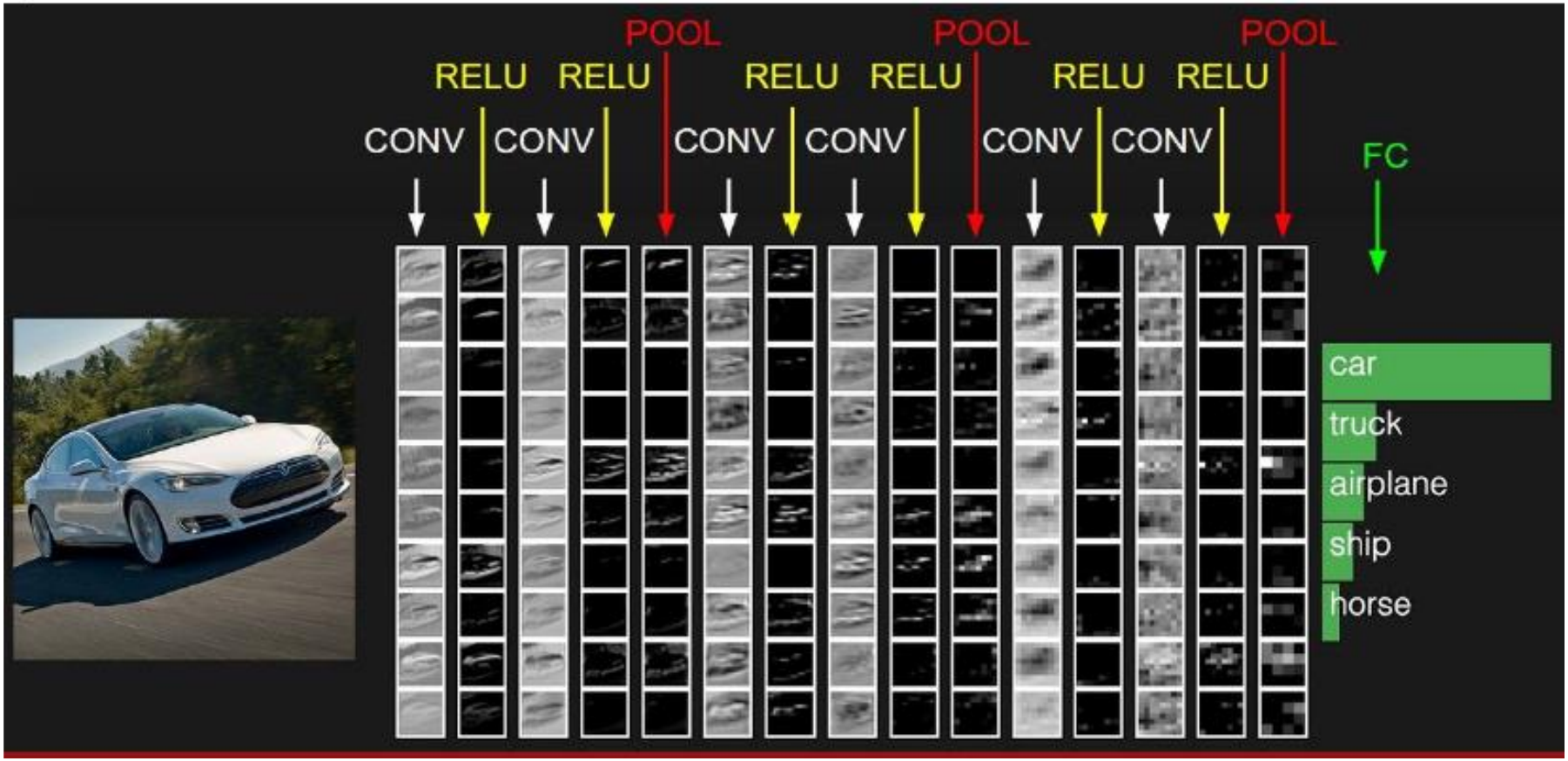
Preview

[From recent Yann LeCun slides]



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

preview:



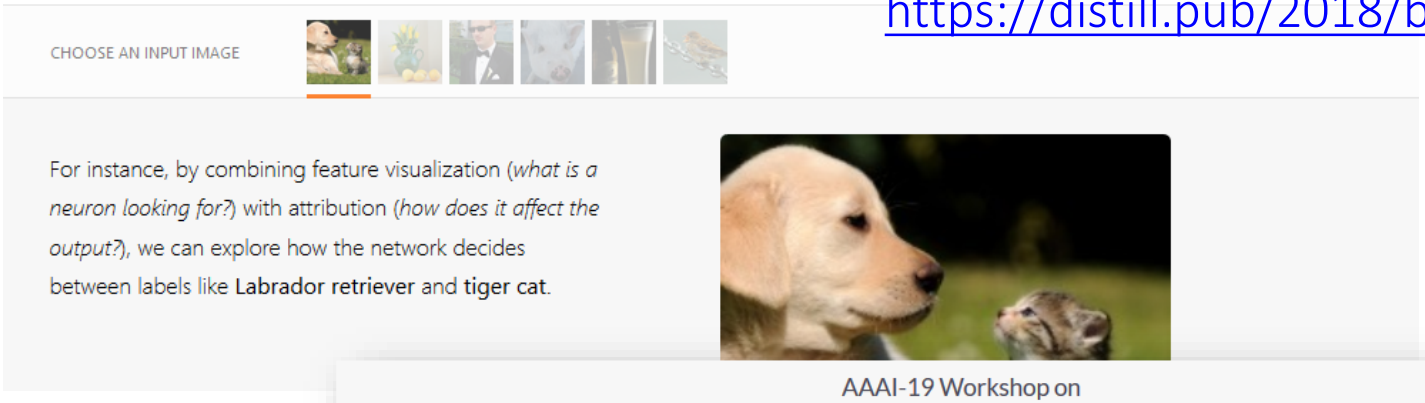
Fei-Fei Li & Andrej Karpathy & Justin Johnson

Interpretability

The Building Blocks of Interpretability

Interpretability techniques are normally studied in isolation.
We explore the powerful interfaces that arise when you combine them —
and the rich structure of this combinatorial space.

https://distill.pub/2018/building-blocks/



AAAI-19 Workshop on

Network Interpretability for Deep Learning

Overview

This workshop aims to bring together researchers, engineers, students in both academic and industrial communities who concern about the interpretability of deep learning models and, more importantly, the safety of applying these complex deep models in critical applications such as the medical diagnosis and the autonomous driving. Efforts along

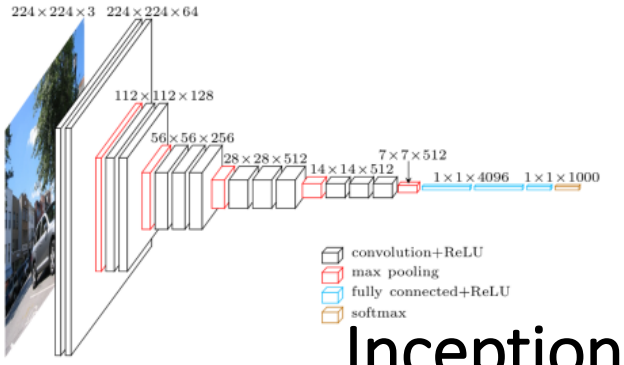
iptc
Information Processing and Telecommunications Center

...there are many Convolutional Network Architectures

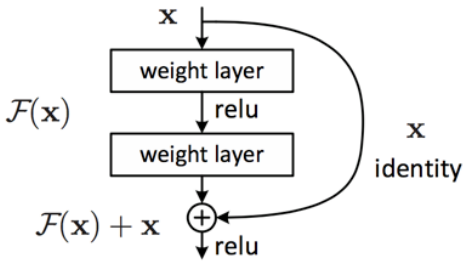
Common architectures in convolutional neural networks

https://www.jeremyjordan.me/convnet-architectures/

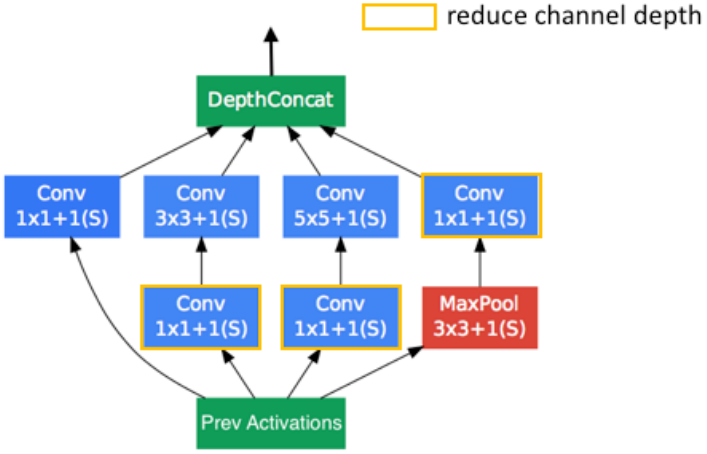
VGG (16)



ResNet (50)



Inception-v3 (GoogLeNet)



- **Font Recognition using CNN with Keras:**

IPTC_Keras_FontReco_CNN.ipynb

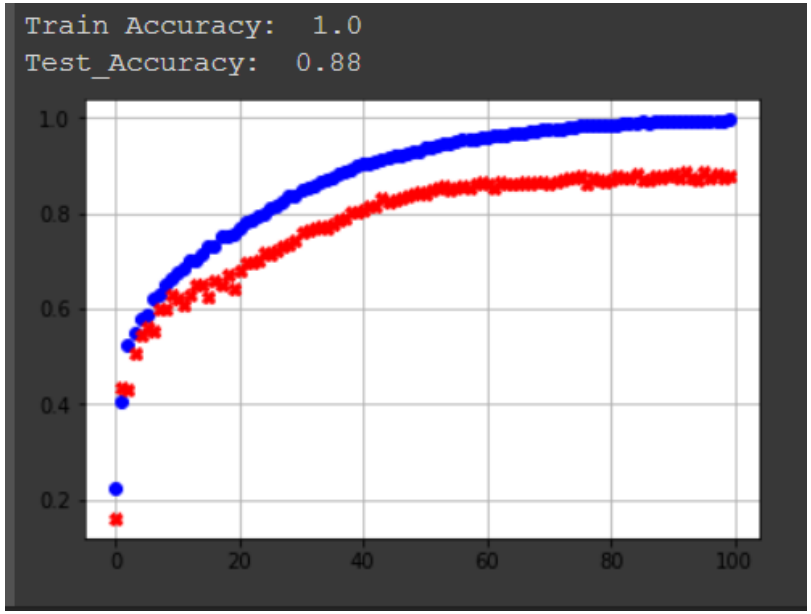
Font Recognition

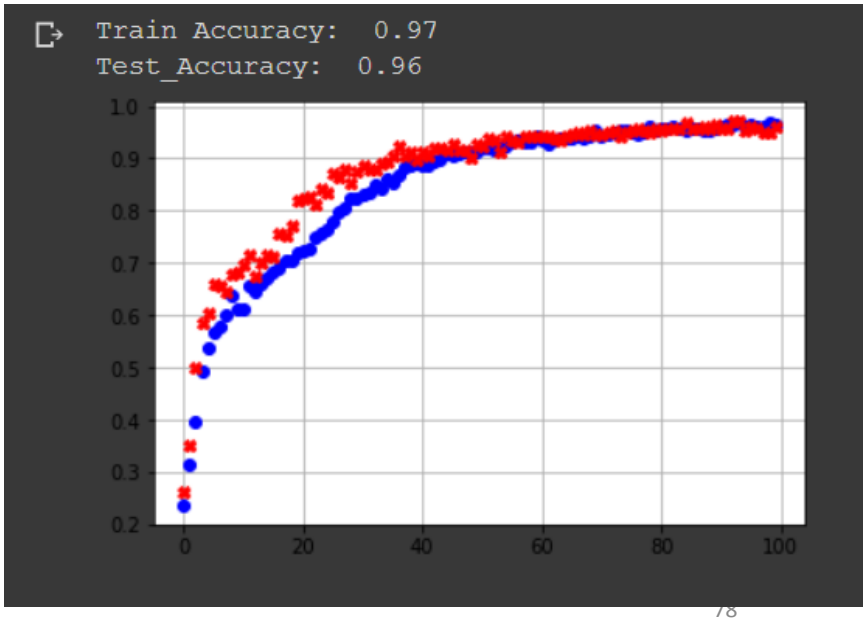
Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 128)	166016
dense_2 (Dense)	(None, 32)	4128
dense_3 (Dense)	(None, 5)	165

Total params: 170,309
Trainable params: 170,309
Non-trainable params: 0

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 4)	104
max_pooling2d_1 (MaxPooling2	(None, 16, 16, 4)	0
flatten_1 (Flatten)	(None, 1024)	0
dense_1 (Dense)	(None, 32)	32800
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 5)	165

Total params: 33,069
Trainable params: 33,069
Non-trainable params: 0

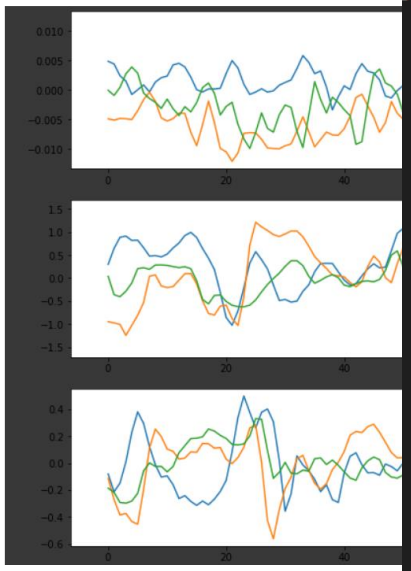




A final example: Convolutional Networks for Time Series

See how CNN can be used for time series, in this case: Human-Activity-Recognition (HAR)

IPTC_Keras_HAR_CNN.ipynb



```
from keras.models import Sequential

from keras.layers import MaxPooling2D, Dropout, Dense, Flatten

from keras.layers import Convolution2D as Conv2D

model = Sequential()
# input: 128x9 images with 1 channel -> (128, 9) tensors.
# this applies 18 convolution filters of size 2x2 each.
model.add(Conv2D(18, (2, 2), activation='relu', input_shape=(128, 9,1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(6, activation='softmax'))
```

iptc
Information Processing and Telecommunications Center

Building-powerful-image-classification-models-using-very-little-data

Transfer Learning, Data Augmentation, ….

https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

The Keras Blog

Keras is a Deep Learning library for Python, that is simple, modular, and extensible.

Archives Github Documentation Google Group

Building powerful image classification models using very little data

In this tutorial, we will present a few simple yet effective methods that you can use to build a powerful image classifier, using only very few training examples --just a few hundred or thousand pictures from each class you want to be able to recognize.

Sun 05 June 2016
By Francois Chollet
In Tutorials.

We will go over the following options:

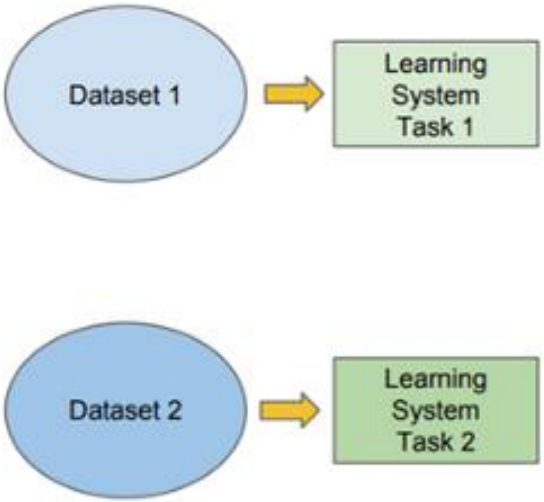
- training a small network from scratch (as a baseline)
- using the bottleneck features of a pre-trained network
- fine-tuning the top layers of a pre-trained network

This will lead us to cover the following Keras features:

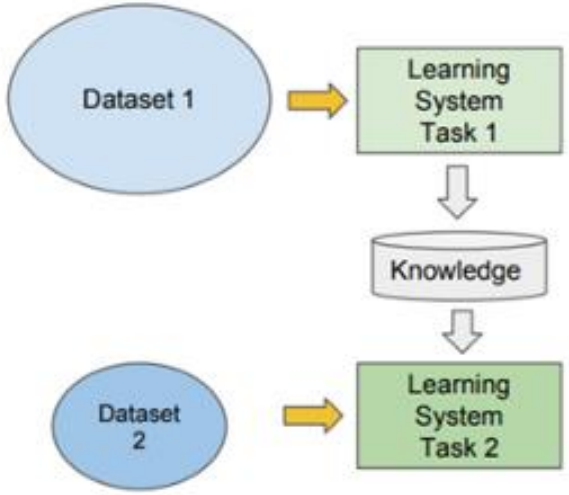
- `fit_generator` for training Keras a model using Python data generators
- `ImageDataGenerator` for real-time data augmentation
- layer freezing and model fine-tuning
- ...and more.

Traditional ML vs Transfer Learning

- Isolated, single task learning:
  - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks



- Learning of a new tasks relies on the previous learned tasks:
  - Learning process can be faster, more accurate and/or need less training data



Transfer Learning

You can have a quick overview

Transfer learning and Image classification using Keras on Kaggle kernels.

https://towardsdatascience.com/transfer-learning-and-image-classification-using-keras-on-kaggle-kernels-c76d3b030649

- Transfer learning is a popular training technique used in deep learning; where models that have been trained for a task are reused as base/starting point for another model.
- It has been mainly used in image classification but can be used/extended to many other fields such audio

Transfer Learning: pre-trained models

Keras comes prepackaged with many types of these **pretrained** models. Some of them are:

- **VGGNET** : Introduced by Simonyan and Zisserman in their 2014 paper, *Very Deep Convolutional Networks for Large Scale Image Recognition*.
- **RESNET** : First introduced by He et al. in their 2015 paper, *Deep Residual Learning for Image Recognition*
- **INCEPTION**: The "Inception" micro-architecture was first introduced by Szegedy et al. in their 2014 paper, *Going Deeper with Convolutions*:
- **XCEPTION**: Xception was proposed by François Chollet , the creator of the **Keras** library.

and many more. Detailed explanation of some of these architectures can be found **here**.

https://towardsdatascience.com/transfer-learning-and-image-classification-using-keras-on-kaggle-kernels-c76d3b030649

Transfer Learning : Font-type recognition

IPTC_Keras_FontReco_TransferLearning.ipynb

See an example of using Transfer Learning (VGG 16) for Font-type recognition

Resize your images to have: 3 channels (RGB) and same dimensions as the Pre-trained network you are going to use

```
[6]  from skimage.transform import resize
     from skimage.color import gray2rgb

     # 224 x 224 for VGG
     hori=224
     vert=224

     train = np.zeros((train_ori.shape[0],hori,vert,3))
     test = np.zeros((test_ori.shape[0],hori,vert,3))

     for n,i in enumerate(train_ori):
         new_img = gray2rgb(train_ori[n,:,:])
         train[n,:,:,:] = resize(new_img, train.shape[1:], anti_aliasing=True)



     for n,i in enumerate(test_ori):
         new_img = gray2rgb(test_ori[n,:,:])
         test[n,:,:,:] = resize(new_img, test.shape[1:], anti_aliasing=True)
```

```
[7]  print('Train images shape=', train.shape , '\nTest images shape=', test.shape)
```

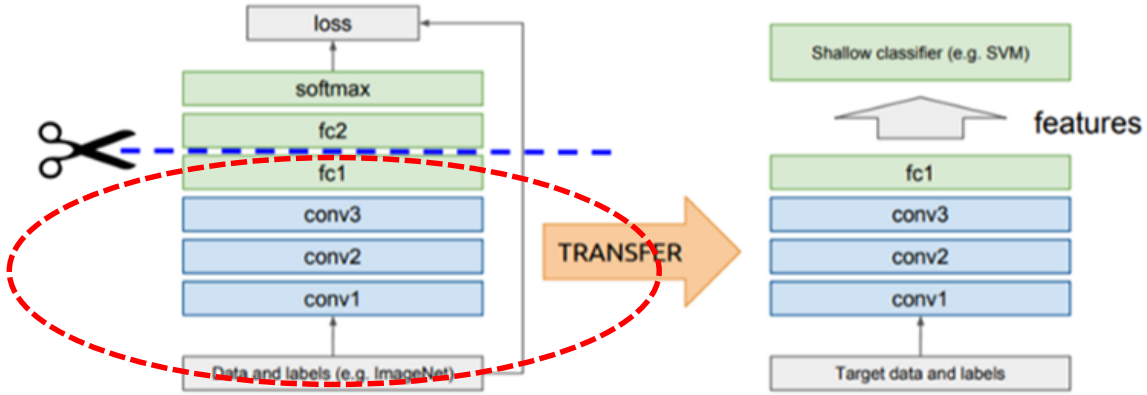
```
Train images shape= (2511, 224, 224, 3)
Test images shape= (279, 224, 224, 3)
```

```
from keras import applications

# In this way input tensor shape is forced
# to be (224, 224, 3)

vgg_model = applications.VGG16(weights='imagenet', include_top=True)
```

- **We can remove the last added layer in a Sequential model by calling .pop():**

```
# Note that if you call it several times you remove the last after the previous

vgg_model.layers.pop()
vgg_model.layers.pop()
vgg_model.layers.pop()
```

Assumes that $D_S = D_T$



- **Now we can add some more layers (we only add one softmax):**

- **`vgg_model.layers[-1].output` is the last output of VGG16 (after pop-ing out the last layers)**

```
[ ]  import keras
     from keras.layers import Dense, Dropout
     from keras.models import Model


     x = Dropout(rate=0.4)(vgg_model.layers[-1].output)
     x = Dense(1024, activation='relu')(x)  # new softmax layer
     predictions = Dense(41, activation='softmax', name='softmax_new1')(x)  # new softmax layer
     transfer_model = Model(input=vgg_model.input, output=predictions)
```

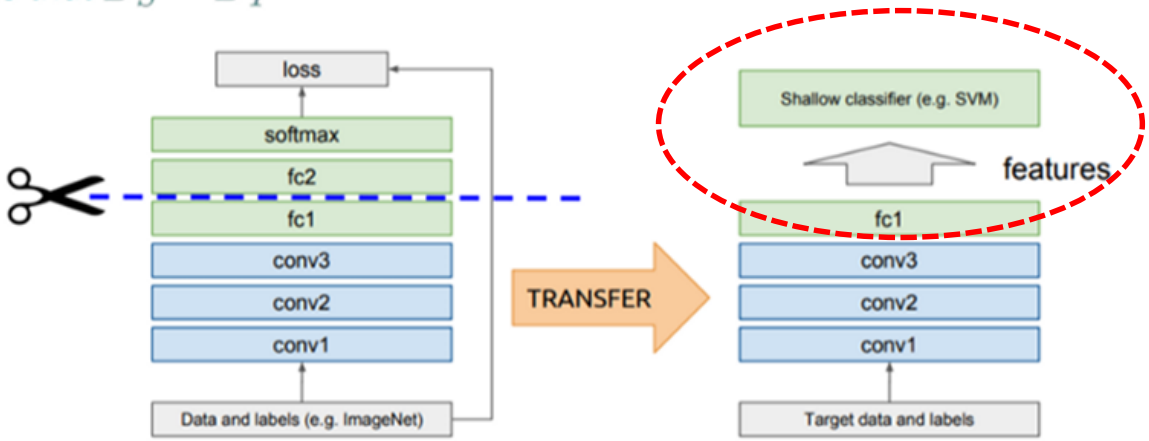
```
[ ]  transfer_model.summary()
```

- **We can Freeze Layers ... or NOT**

```
[ ]  # Make sure that the pre-trained bottom layers are not trainable
     for layer in vgg_model.layers:
         layer.trainable = False
```

- **COMPILE... as usual**

Assumes that $D_S = D_T$



Learn the use of Keras API

Now we use KERAS API

We can add some more layers (we only add one softmax):

- `vgg_model.layers[-1].output` is the last output of VGG16 (after pop-ing out the last layers)

```
[13]  import keras
      from keras.layers import Dense, Dropout
      from keras.models import Model


      x = Dropout(rate=0.4)(vgg_model.layers[-1].output)
      x = Dense(64, activation='relu')(x)   # new softmax layer
      predictions = Dense(5, activation='softmax', name='softmax_new1')(x)   # new softmax layer
      transfer_model = Model(input=vgg_model.input, output=predictions)
```

Keras: Functional API
see: https://keras.io/getting-started/functional-api-guide/

Getting started with the Keras functional API

- The Keras functional API is the way to go for defining complex models, such as multi-output models, directed acyclic graphs, or models with shared layers.

- With the functional API, it is easy to reuse trained models: you can treat any model as if it were a layer, by calling it on a tensor

```
from keras.layers import Input, Dense
from keras.models import Model

# This returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
predictions = Dense(10, activation='softmax')(x)
```

- A layer instance is callable (on a tensor), and it returns a tensor
- Input tensor(s) and output tensor(s) can then be used to define a Model

```
model = Model(inputs=inputs, outputs=predictions)
```

- Such a model can be trained just like Keras Sequential models.

```
from keras.layers import Input, Dense
from keras.models import Model

# This returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels)  # starts training
```

All models are callable, just like layers

With the functional API:

- it is easy to reuse trained models: you can treat any model as if it were a layer, by calling it on a tensor. (Transfer Learning)

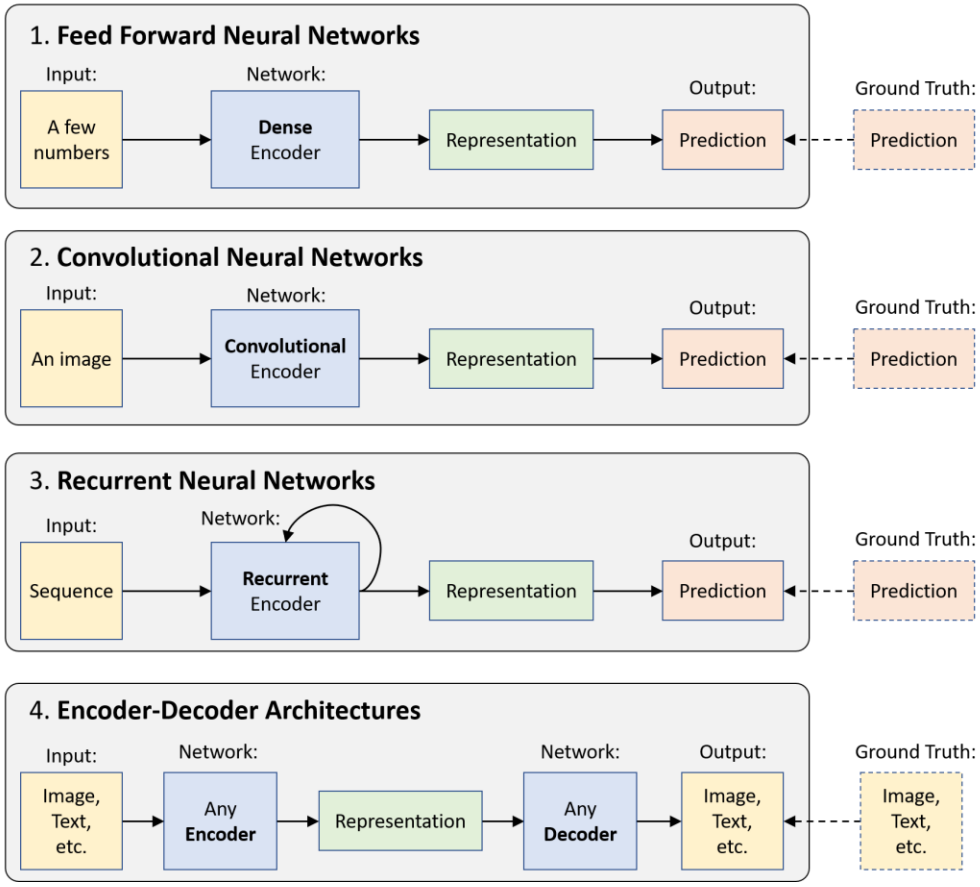
- Note that **by calling a model you aren't just reusing the *architecture* of the model, you are also reusing its weights**

```
x = Input(shape=(784,))
# This works, and returns the 10-way softmax we defined above.
y = model(x)
```

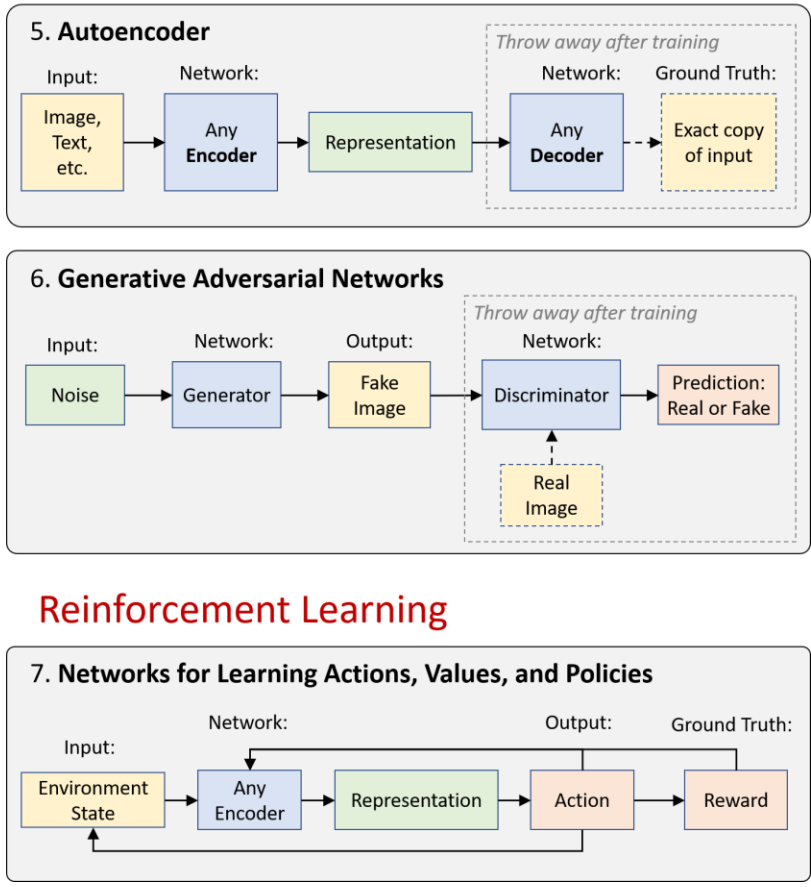
https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

Deep learning basics

Supervised Learning



Unsupervised Learning



Reinforcement Learning

https://github.com/lexfridman/mit-deep-learning/blob/master/tutorial_deep_learning_basics/deep_learning_basics.ipynb

iptc
Information Processing and Telecommunications Center