IDENTITY & SECURITY

DEEP LEARNING

—

Télécom ParisTech – ML4BD
22/02/2017
27/02/2017

Stéphane Gentric Ph.D.
Research Unit Manager

SAFRAN

- **Intro**

- **Deep @ Safran I&S**
- **Data**
- **Learning Frameworks**

- **Neural Network**
  - Linear – perceptron
  - Tensorflow

- **CNN Architecture**
  - Activation
  - Convolution
  - Pooling
  - Loss

- **Optimization**
  - GD, SGD, batch GD
  - Momentum
  - Adagrad
  - Adam
  - Initialization

- **Overfitting**
  - Intro
  - Early stop
  - Regularization
  - More data – augmentation
  - Dropout

- **More DNN Architectures**
  - Batch normalization
  - Residual networks
  - AutoEncoder
  - RNN & LSTM
  - SSD & YOLO

nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
At last — a computer program that can beat a champion Go player PAGE 484
ALL SYSTEMS GO
CONSERVATION
SONGBIRDS À LA CARTE
Illegal harvest of millions of Mediterranean birds
PAGE 452
RESEARCH ETHICS
SAFEGUARD TRANSPARENCY
Don't let openness backfire on individuals
PAGE 459
POPULAR SCIENCE
WHEN GENES GOT 'SELFISH'
Dawkins's calling card forty years on
PAGE 462
NATURE.COM/NATURE

In "Nature" 27 January 2016:

"DeepMind's program AlphaGo beat Fan Hui, the European Go champion, five times out of five in tournament conditions..."

"AlphaGo was not preprogrammed to play Go: rather, it learned using a general-purpose algorithm that allowed it to interpret the game's patterns."

"...AlphaGo program applied deep learning in neural networks (convolutional NN) — brain-inspired programs in which connections between layers of simulated neurons are strengthened through examples and experience."

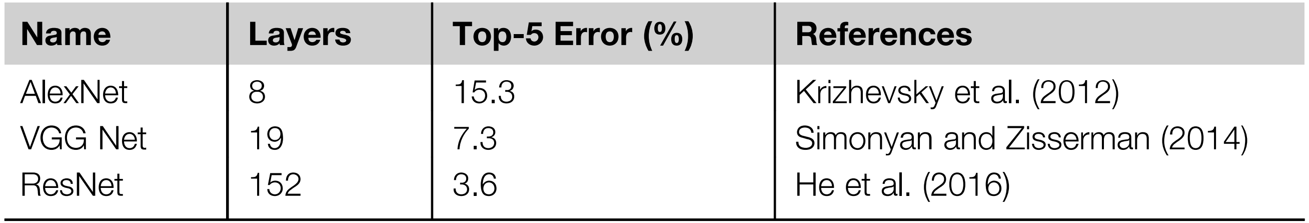
The ImageNet challenge

- Crucial in demonstrating the effectiveness of deep CNNs
- Problem: recognize object categories in Internet imagery
- The 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) classification task - classify image from Flickr and other search engines into 1 of 1000 possible object categories
- Serves as a standard benchmark for deep learning
- The imagery was hand-labeled based on the presence or absence of an object belonging to these categories
- There are 1.2 million images in the training set with 732-1300 training images available per class
- A random subset of 50,000 images was used as the validation set, and 100,000 images were used for the test set where there are 50 and 100 images per class respectively

A plateau, then rapid advances

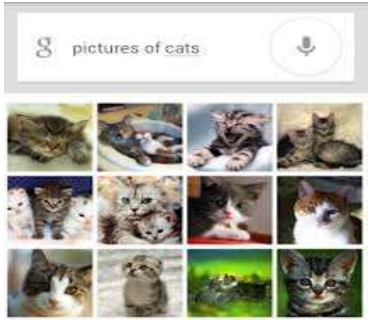
“Top-5 error” is the % of times that the target label does not appear among the 5 highest-probability predictions

Visual recognition methods not based on deep CNNs hit a plateau in performance at 25%



Name	Layers	Top-5 Error (%)	References
AlexNet	8	15.3	Krizhevsky et al. (2012)
VGG Net	19	7.3	Simonyan and Zisserman (2014)
ResNet	152	3.6	He et al. (2016)

Deep Learning – from Research to Technology



Deep Learning - breakthrough in visual and speech recognition

CNN applications

CNN is a big hammer

Plenty low hanging fruits

You need just a right nail!

ARTIFICIAL NEURAL NETWORK

1943. McCulloch and Pitts created a computational model for neural networks

1958. Rosenblatt created the perceptron

1975. Werbos . Backpropagation for Neural networks (NN)

1985. Multilayer Perceptrons (MLP)

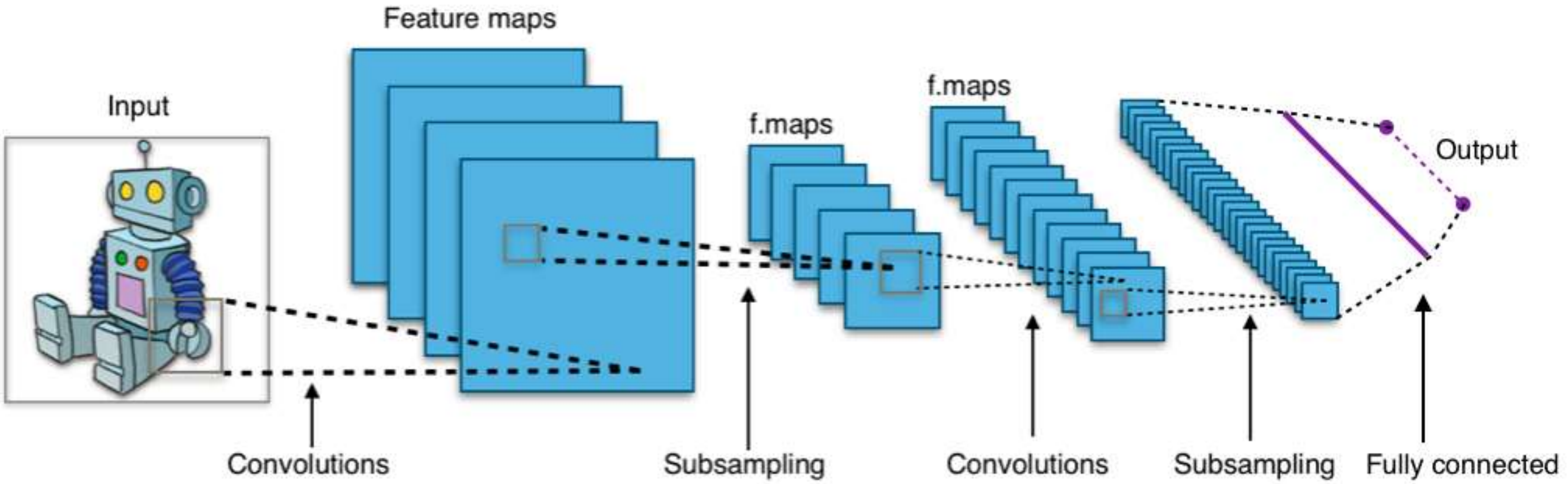
1986. Restricted Boltzmann Machine (RBM)

1995. Support Vector Machine(SVM)

1998. LeNet-5 (CNN)

Source: Sunila Gollapudi (2016), Practical Machine Learning, Packt Publishing

Typical CNN architecture

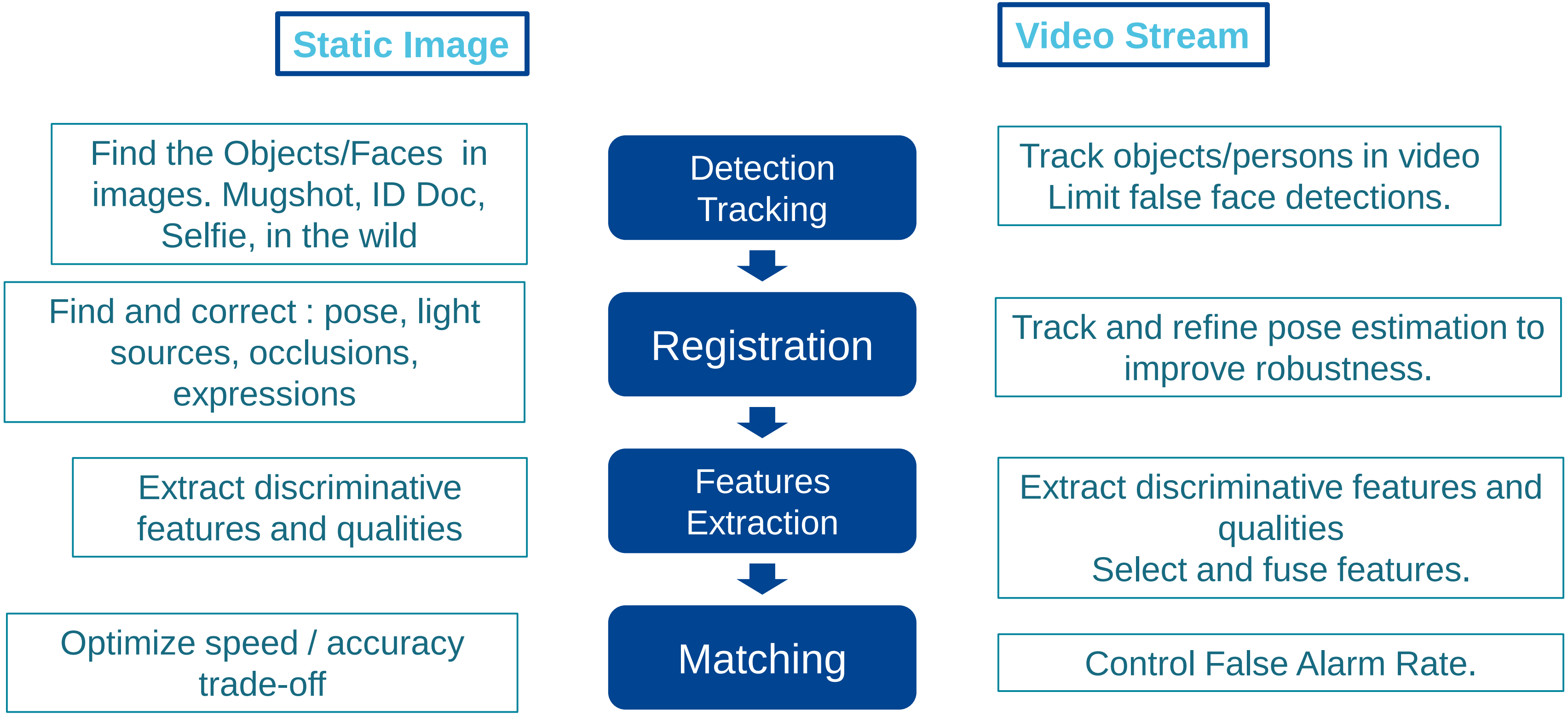


IDENTITY & SECURITY

DEEP @ SAFRAN I & S

SAFRAN

Object & Face Recognition Pipeline



Static Image

Video Stream

Find the Objects/Faces in images. Mugshot, ID Doc, Selfie, in the wild

Detection Tracking

Track objects/persons in video
Limit false face detections.

Find and correct : pose, light sources, occlusions, expressions

Registration

Track and refine pose estimation to improve robustness.

Extract discriminative features and qualities

Features Extraction

Extract discriminative features and qualities
Select and fuse features.

Optimize speed / accuracy trade-off

Matching

Control False Alarm Rate.

Face Detection and Tracking

Face Detection Algorithm

- Saliency Map for hypothesis generation
- Classifier
  - Robust features, ex: Haar, Filter bank, SIFT, SURF, HOG, color histograms
  - Machine learning, ex: CNN, SVM, Ada-Float boost, decision tree
- The classifiers are learned on a database of faces and non faces. Bootstrap and selection of hard examples.
- The detection pipeline is a fusion of those approaches for obtaining the best speed/accuracy tradeoff.

2D Face Alignment

Face Alignment algorithm based on landmark localization

- Landmark localization and/or regression
  - Filter bank responses + SVM
  - Random Forest
  - CNN

Face Representation

Comparing two faces by computing the Euclidean distance directly between the images leads to very poor results

- It does not take into account the variability of face appearance
- It does not rely enough on the discriminative information

Face recognition algorithm = Function which embeds a face image into a discriminative space

- A discriminative space is a space in which the Euclidean distance has the good semantic (small distance = same id)

Two families of approaches:

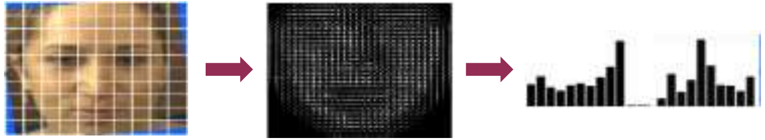
1. **Traditional face recognition:**

    Extract agnostic features + learning of a feature transformation

2. **Deep learning**

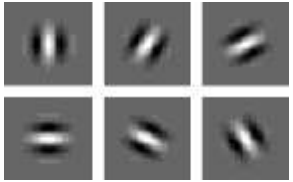
    A neural network performs the full transformation from the image to the face embedding

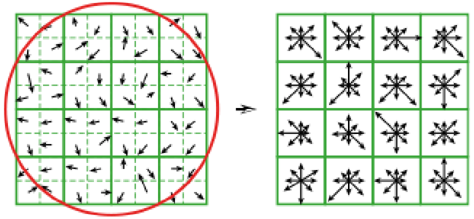
Traditional Approaches



These methods are composed of two steps

1. Agnostic features are extracted from the images
   - Most of the discriminative information is born by the gradients
   - Various representations of the gradients have been proposed
     - SIFT, HOG, LBP
     - Filter bank responses (Gaussian derivatives, Gabor, etc.)

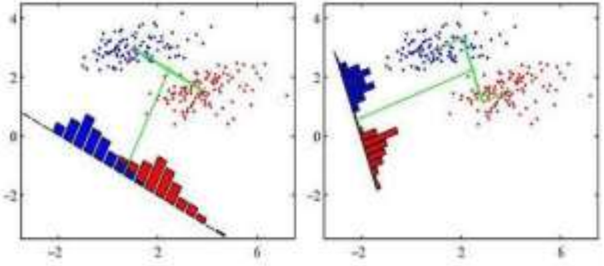


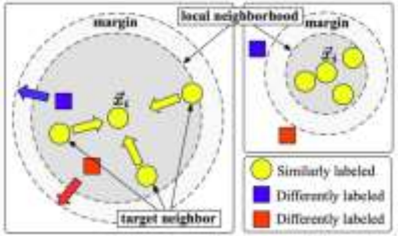


gradients d'image

descripteur de point-clé

2. A transformation of the initial feature space is learned
   - It reduces the dimensionality and concentrates the discriminative information (remove noise and redundancy)
   - Most methods simply use a linear transformation
     - LDA, ITML, LMNN, …
   - But more complex methods exist (e.g. local metric)





A well-engineered traditional method gives accurate results on well aligned images but lacks of robustness

Deep Learning for Face Representation

Deep Learning is a new name for *Neural Network*

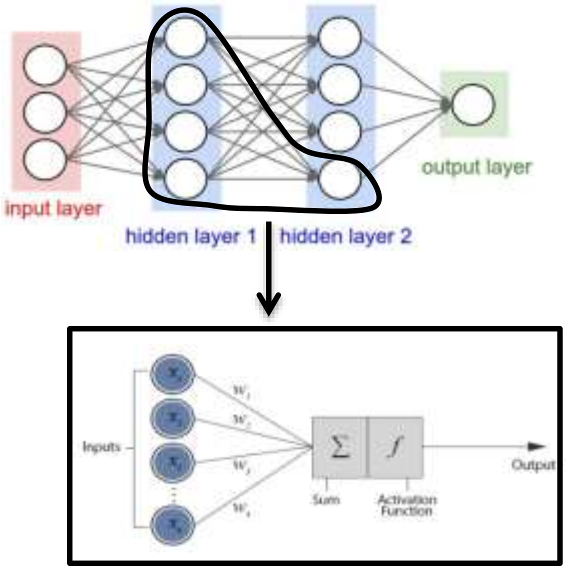
- Sequence of parametric linear and non-linear operations.
- Able to approximate well highly complex and non linear functions

CNN is a class of neural networks for processing images

- Most linear operations are convolutions: the spatial structure of images is exploited

The transformation from the image to the face representation is learned in one step

- The best basic features for the task are learned
- It requires a large amount of labeled images which should be as representative as possible of the client’s use case



DATA

Data and label for this session

Face images coming from the Web :

http://www.cbsr.ia.ac.cn/english/CASIA-WebFace-Database.html

♦ Dong Yi, Zhen Lei, Shengcai Liao and Stan Z. Li, “Learning Face Representation from Scratch”. arXiv preprint arXiv:1411.7923. 2014.

Pre-processed :

centred on the face, 48*48 pixels, grey levels.

Labels :

gender automatically deduced from first name

4 databases :

training : 1000, 10k and 100k

testing : 10k

Goal :

finding Gender from Image

LEARNING FRAMEWORKS

Deep learning software

Theano

A library in Python which has been developed with the specific goal of facilitating research in deep learning (Bergstra et al., 2010; Theano Development Team, 2016)

It is also a powerful general purpose tool for general mathematical programming

Theano extends NumPy (the main Python package for scientific computing) by adding symbolic differentiation and GPU support, among various other functions

It provides a high-level language for creating the mathematical expressions that underlie deep learning models, and a compiler that takes advantage of deep learning techniques, including calls to GPU libraries, to produce code that executes quickly

Deep learning software

Torch

An open-source machine learning library built using C and a high-level scripting language known as Lua (Collobert et al., 2011)

It uses multidimensional array data structures, and supports various basic numerical linear algebra manipulations

It has a neural network package with modules that permit the typical forward and backward methods needed for training neural networks

It also supports automatic differentiation

Further information:

http://torch.ch/

https://www.youtube.com/watch?v=uxja6iwOnc4&list=PLjJh1vlSEYgvGod9wWiydumYl8hOXixNu&index=19

Deep learning software

Tensor Flow

C++ and Python based software library for the types of numerical computation typically associated with deep learning (Abadi et al., 2016)

It is heavily inspired by Theano, and, like it, uses dataflow graphs to represent the ways in which multidimensional data arrays communicate between one another

These multidimensional arrays are referred to as “tensors.”

Tensor Flow also supports symbolic differentiation and execution on multiple GPUs

It was released in 2015

www.tensorflow.org

https://www.youtube.com/watch?v=bYeBL92v99Y

Deep learning software

CNTK (Computational Network Toolkit)

C++ library for manipulating computational networks (Yu et al., 2014)

It was produced by Microsoft Research, but has been released under a permissive license

It has been popular for speech and language processing, but also supports convolutional networks of the type used for images

It supports execution on multiple machines and using multiple GPUs

https://www.cntk.ai/

https://www.youtube.com/watch?v=-mLdConF1EU

Deep learning software

Caffe

Developed by Berkeley Vision and Learning Center (BLVC) and community contributors

C++ and Python based BSD-licensed convolutional neural network library (Jia et al., 2014).

Has a clean and extensible design which makes it a popular alternative to the original open-source implementation of Krizhevsky et al. (2012)’s famous AlexNet that won the 2012 ImageNet challenge.

http://caffe.berkeleyvision.org/

https://www.youtube.com/watch?v=bOIZ74rOik0

Deep learning software

Lasagne, Keras and cuDNN

Lasagne is a lightweight Python library built on top of Theano that simplifies the creation of neural network layers

Similarly, Keras is a Python library that runs on top of either Theano or TensorFlow (Chollet, 2015) that allows one to quickly define a network architecture in terms of layers and also includes functionality for image and text preprocessing

----

cuDNN is a highly optimized GPU library for NVIDIA units that allows deep learning networks to be trained more quickly

It can dramatically accelerate the performance of a deep network and is often called by the other packages above.

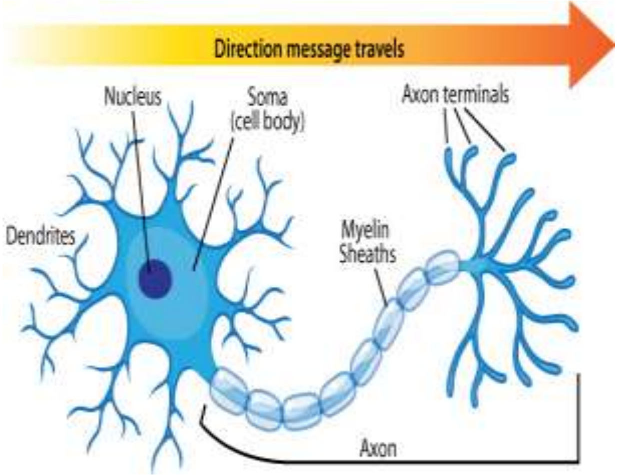
IDENTITY & SECURITY

NEURAL NETWORK

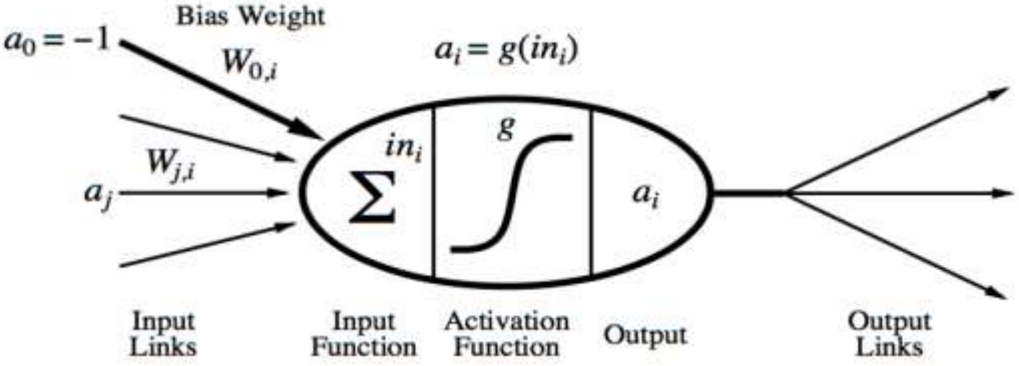
SAFRAN

Neural Network: A Neuron

A neuron is a computational unit in the neural network that exchanges messages with each other.

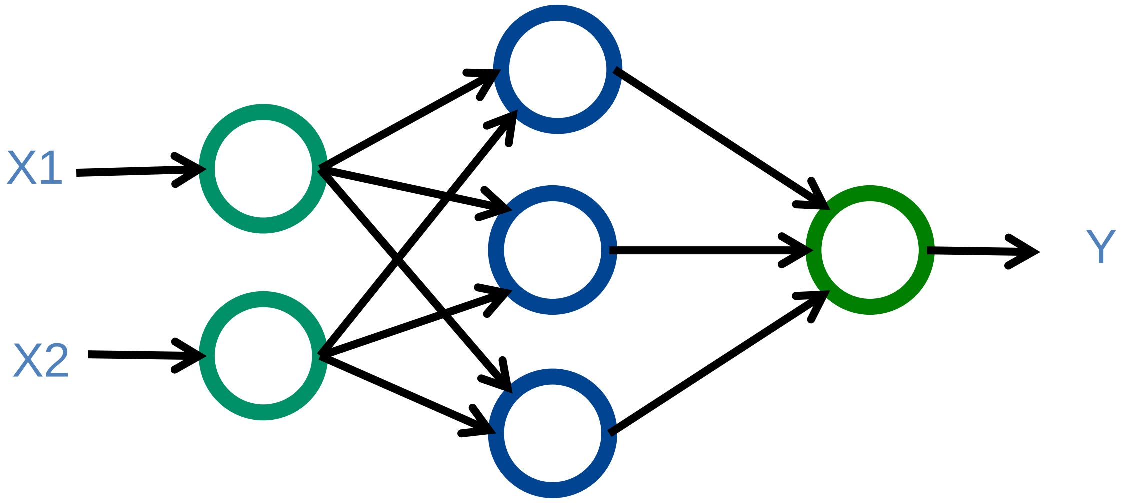


$$a_i \leftarrow g(in_i) = g\left(\Sigma_j W_{j,i} a_j\right)$$



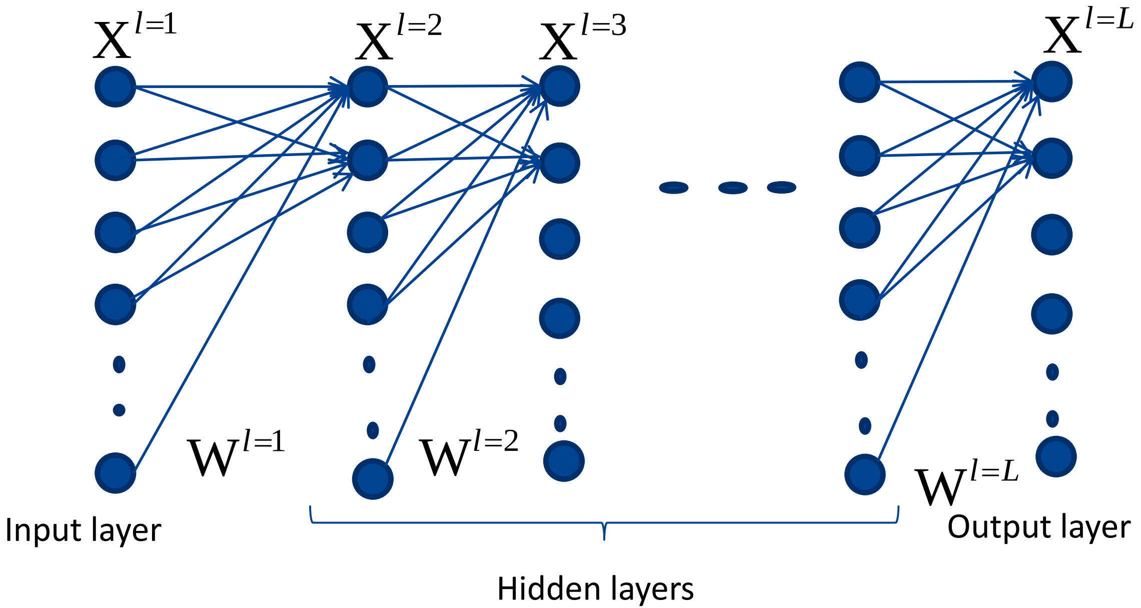
Neural Networks

Input Layer (X) | Hidden Layer (H) | Output Layer (Y)



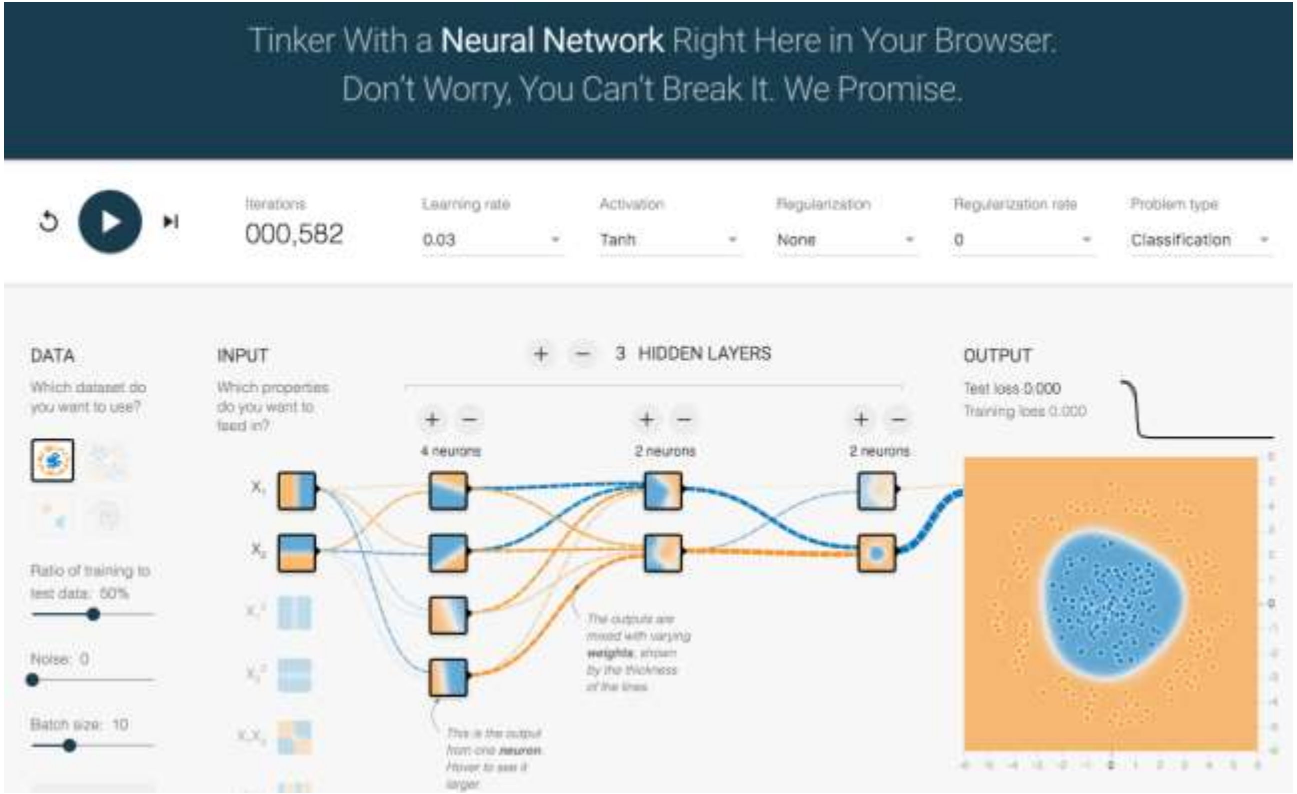
Source: https://www.youtube.com/watch?v=bxe2T-V8XRs&index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU

A deep neural network



TensorFlow Playground

http://playground.tensorflow.org/



Tinker With a **Neural Network** Right Here in Your Browser.
Don't Worry, You Can't Break It. We Promise.

Iterations
000,582

Learning rate
0.03

Activation
Tanh

Regularization
None

Regularization rate
0

Problem type
Classification

DATA
Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

INPUT
Which properties do you want to feed in?

3 HIDDEN LAYERS

OUTPUT
Test loss 0.000
Training loss 0.000

*This is the output from one **neuron**. Hover to see it larger.*

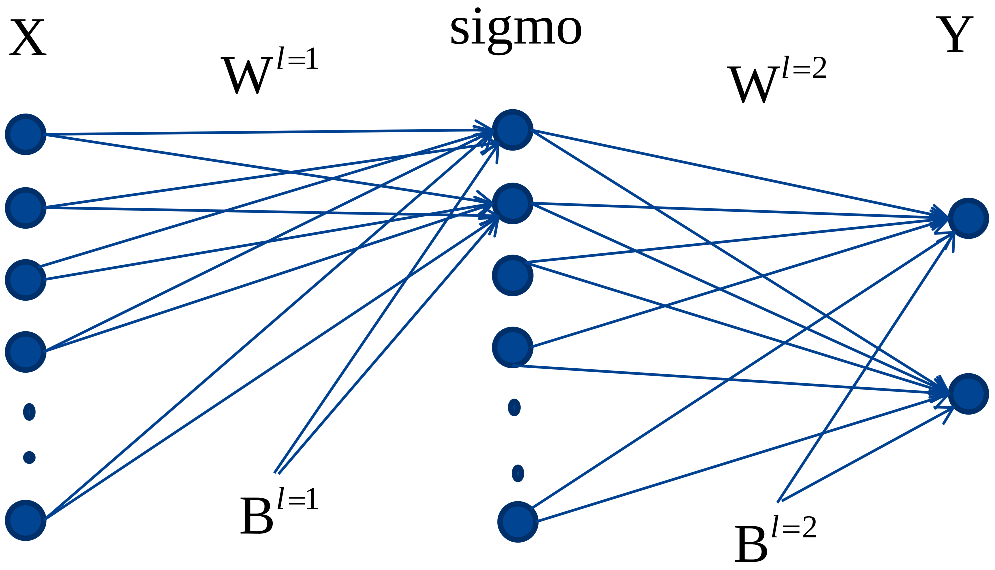
*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

nn1 : simplest.py

```
import tensorflow as tf
import numpy as np

# nombre d images
nbdata = 1000
trainDataFile = '../DataBases/data_1k.bin'
LabelFile = '../DataBases/gender_1k.bin'

# taille des images 48*48 pixels en niveau de gris
dim = 2304
f = open(trainDataFile, 'rb')
data = np.empty([nbdata, dim], dtype=np.float32)
for i in xrange(nbdata):
    data[i,:] = np.fromfile(f, dtype=np.uint8, count=dim).astype(np.float32)
f.close()

f = open(LabelFile, 'rb')
label = np.empty([nbdata, 2], dtype=np.float32)
for i in xrange(nbdata):
    label[i,:] = np.fromfile(f, dtype=np.float32, count=2)
f.close()


def fc_layer(tensor, input_dim, output_dim):
    Winit = tf.truncated_normal([input_dim, output_dim], stddev=0.1)
    print Winit
    W = tf.Variable(Winit)
    print W
    Binit = tf.constant(0.0, shape=[output_dim])
    B = tf.Variable(Binit)
    tensor = tf.matmul(tensor, W) + B
    return tensor
```

```
x = tf.placeholder(tf.float32, [None, dim])
y_desired = tf.placeholder(tf.float32, [None, 2])

layer1 = fc_layer(x,dim,50)
sigmo = tf.nn.sigmoid(layer1)
y = fc_layer(sigmo,50,2)

loss = tf.reduce_sum(tf.square(y - y_desired))
train_step = tf.train.GradientDescentOptimizer(1e-5).minimize(loss)

sess = tf.Session()
sess.run(tf.initialize_all_variables())
curPos = 0
batchSize = 256
nbIt = 1000000
for it in xrange(nbIt):
    if curPos + batchSize > nbdata:
        curPos = 0
    trainDict = {x:data[curPos:curPos+batchSize,:],y_desired:label[curPos:curPos+batchSize,:]}

    curPos += batchSize

    sess.run(train_step, feed_dict=trainDict)
    if it%1000 == 0:
        print "it= %6d - loss= %f" % (it, sess.run(loss, feed_dict=trainDict))

sess.close()
```

nn1 : A simple perceptron

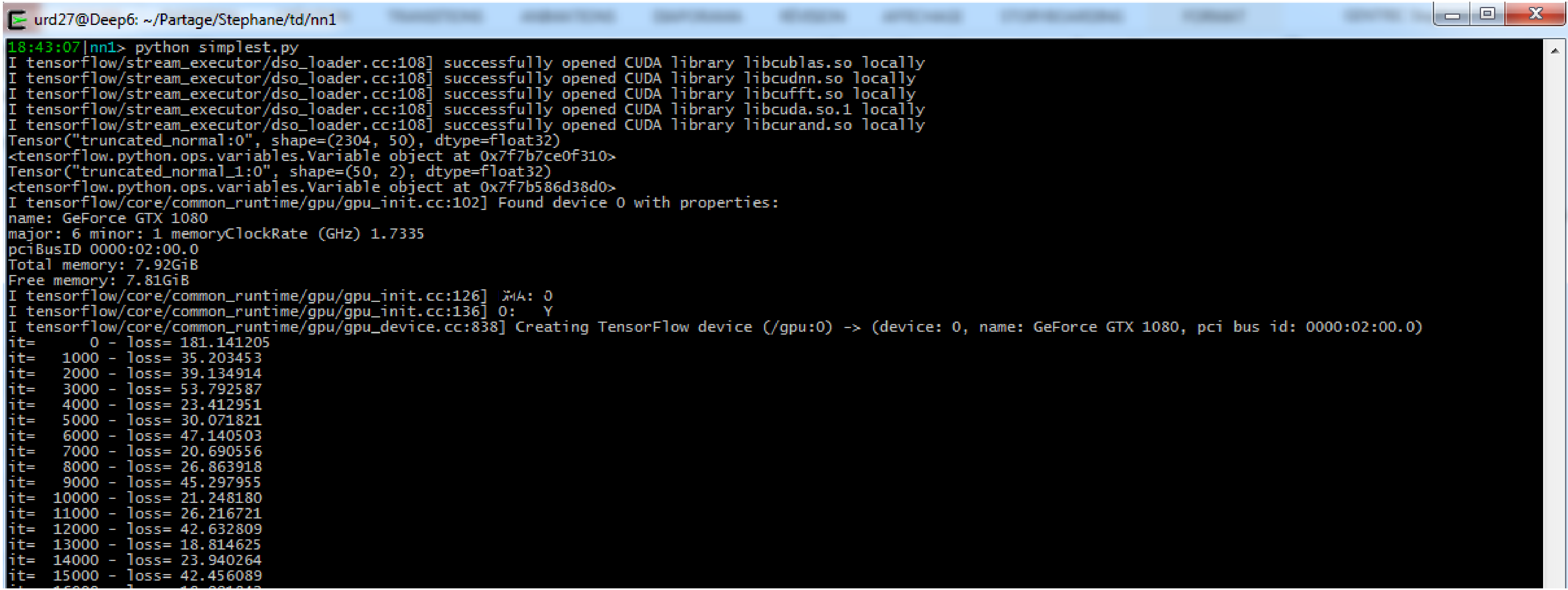


Input layer
2304 neurons

Hidden layers
50 neurons

Output layer
2 neurons

nn1 : let’s run



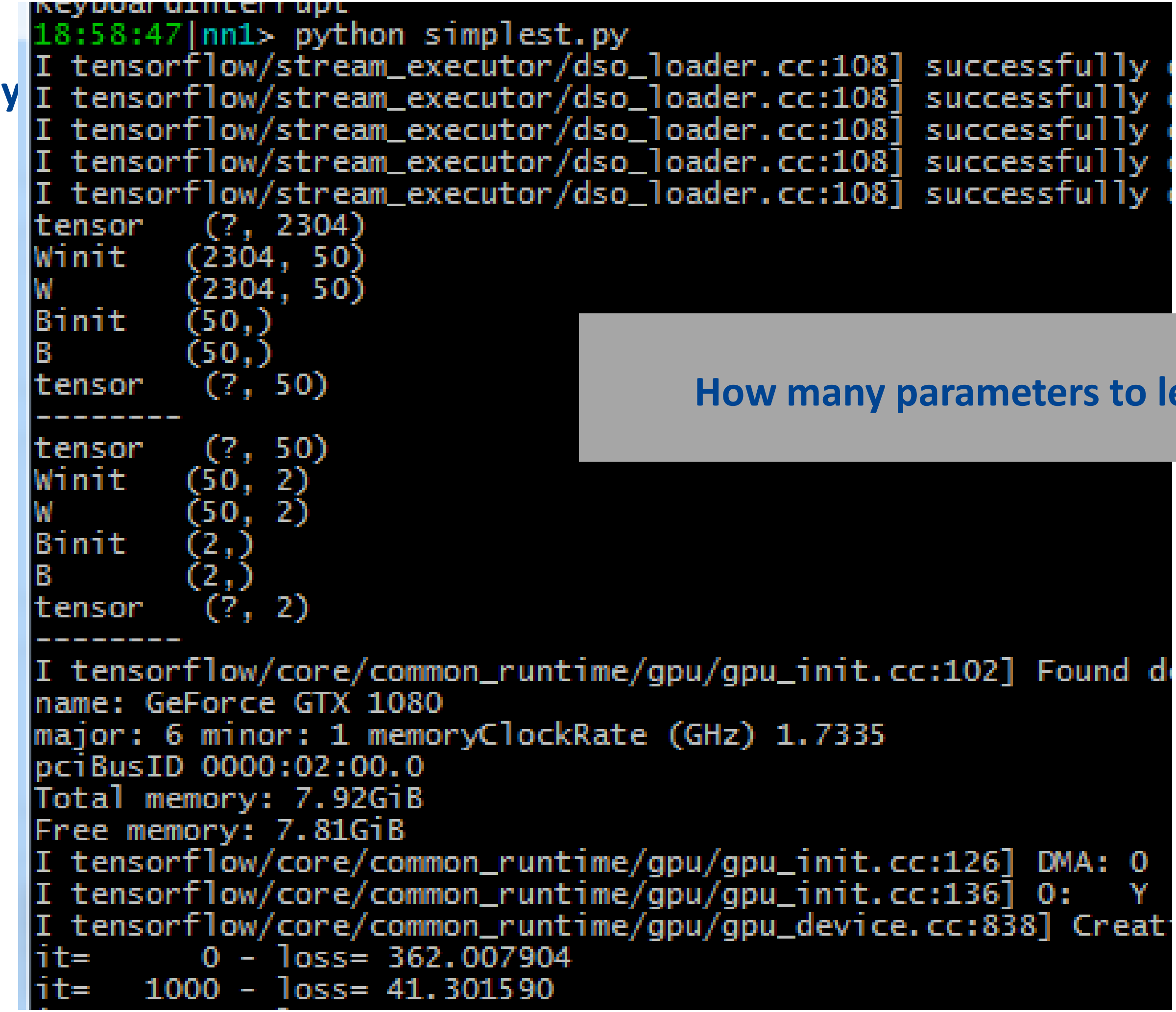
```
18:43:07|nn1> python simplest.py
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcublas.so locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcudnn.so locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcufft.so locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcuda.so.1 locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcurand.so locally
Tensor("truncated_normal:0", shape=(2304, 50), dtype=float32)
<tensorflow.python.ops.variables.Variable object at 0x7f7b7ce0f310>
Tensor("truncated_normal_1:0", shape=(50, 2), dtype=float32)
<tensorflow.python.ops.variables.Variable object at 0x7f7b586d38d0>
I tensorflow/core/common_runtime/gpu/gpu_init.cc:102] Found device 0 with properties:
name: GeForce GTX 1080
major: 6 minor: 1 memoryClockRate (GHz) 1.7335
pciBusID 0000:02:00.0
Total memory: 7.92GiB
Free memory: 7.81GiB
I tensorflow/core/common_runtime/gpu/gpu_init.cc:126] DMA: 0
I tensorflow/core/common_runtime/gpu/gpu_init.cc:136] 0:   Y
I tensorflow/core/common_runtime/gpu/gpu_device.cc:838] Creating TensorFlow device (/gpu:0) -> (device: 0, name: GeForce GTX 1080, pci bus id: 0000:02:00.0)
it=      0 - loss= 181.141205
it=   1000 - loss= 35.203453
it=   2000 - loss= 39.134914
it=   3000 - loss= 53.792587
it=   4000 - loss= 23.412951
it=   5000 - loss= 30.071821
it=   6000 - loss= 47.140503
it=   7000 - loss= 20.690556
it=   8000 - loss= 26.863918
it=   9000 - loss= 45.297955
it=  10000 - loss= 21.248180
it=  11000 - loss= 26.216721
it=  12000 - loss= 42.632809
it=  13000 - loss= 18.814625
it=  14000 - loss= 23.940264
it=  15000 - loss= 42.456089
```

nn1 : simplest.py

What are the shapes of these tensors ?

```
def fc_layer(tensor, input_dim, output_dim):
        print 'tensor  ',tensor.get_shape()
        Winit = tf.truncated_normal([input_dim, output_dim], stddev=0.1)
        W = tf.Variable(Winit)
        print 'Winit  ',Winit.get_shape()
        print 'W  ',W.get_shape()
        Binit = tf.constant(0.0, shape=[output_dim])
        B = tf.Variable(Binit)
        print 'Binit  ',Binit.get_shape()
        print 'B  ',B.get_shape()
        tensor = tf.matmul(tensor, W) + B
        print 'tensor  ',tensor.get_shape()
        print ‘--------'
        return tensor
```

nn1 : simplest.py



```
KeyboardInterrupt
18:58:47|nn1> python simplest.py
I tensorflow/stream_executor/dso_loader.cc:108] successfully
I tensorflow/stream_executor/dso_loader.cc:108] successfully
I tensorflow/stream_executor/dso_loader.cc:108] successfully
I tensorflow/stream_executor/dso_loader.cc:108] successfully
I tensorflow/stream_executor/dso_loader.cc:108] successfully
tensor    (?, 2304)
Winit    (2304, 50)
W        (2304, 50)
Binit    (50,)
B        (50,)
tensor    (?, 50)
---------
tensor    (?, 50)
Winit    (50, 2)
W        (50, 2)
Binit    (2,)
B        (2,)
tensor    (?, 2)
---------
I tensorflow/core/common_runtime/gpu/gpu_init.cc:102] Found d
name: GeForce GTX 1080
major: 6 minor: 1 memoryClockRate (GHz) 1.7335
pciBusID 0000:02:00.0
Total memory: 7.92GiB
Free memory: 7.81GiB
I tensorflow/core/common_runtime/gpu/gpu_init.cc:126] DMA: 0
I tensorflow/core/common_runtime/gpu/gpu_init.cc:136] 0:   Y
I tensorflow/core/common_runtime/gpu/gpu_device.cc:838] Creat
it=      0 - loss= 362.007904
it=   1000 - loss= 41.301590
```

How many parameters to learn ?

nn2 : simplest_v2.py

Save and Load the network

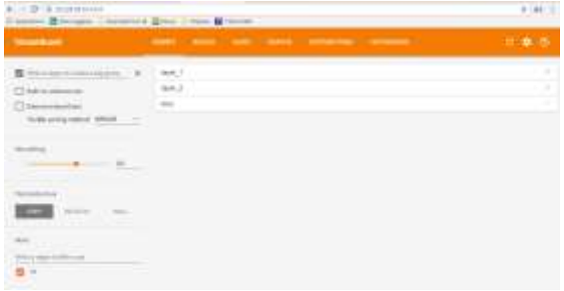
```
saver = tf.train.Saver()
saver.restore(sess, "./model.ckpt")
saver.save(sess, "./model.ckpt")
```

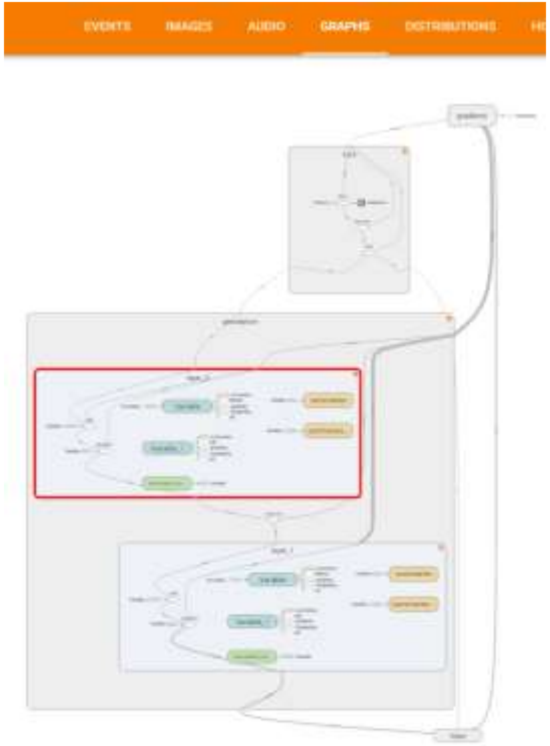
Following the learning process : Summary

```
tf.scalar_summary(name, tensor)
tf.histogram_summary(name, tensor)

merged = tf.merge_all_summaries()

writer = tf.train.SummaryWriter(exp_name, sess.graph)
summary_merged = sess.run(merged, )
writer.add_summary(summary_merged, it)
```

nn2 : simplest_v2.py

```
tensorboard --logdir nn2 --port 8888
```





nn3 : perceptron for gender classification … does it work ?

3 datasets for learning and 1 for testing

```
train = ds.DataSet('../DataBases/data_1k.bin','../DataBases/gender_1k.bin',1000)
#train = ds.DataSet('../DataBases/data_10k.bin','../DataBases/gender_10k.bin',10000)
#train = ds.DataSet('../DataBases/data_100k.bin','../DataBases/gender_100k.bin',100000)
test = ds.DataSet('../DataBases/data_test10k.bin','../DataBases/gender_test10k.bin',10000)
```

Compute Accuracy

```
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_desired, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

On a complete dataset

```
def mean_accuracy(self, TFsession,loc_acc,loc_x,loc_y):
    acc = 0
    for i in xrange(0, self.nbdata, self.batchSize):
        curBatchSize = min(self.batchSize, self.nbdata - i)
        dict = {loc_x:self.data[i:i+curBatchSize,:],loc_y:self.label[i:i+curBatchSize,:]}
        acc += TFsession.run(loc_acc, dict) * curBatchSize
    acc /= self.nbdata
    return acc
```

CNN ARCHITECTURE

CNN Building Blocks

- **Fully Connected layer**
- **Activation layer**
- **Convolutional Layers**
- **Pooling Layers**
- **Loss Layer**

ACTIVATION FUNCTIONS

Sigmoid function

$$f(u) = \frac{1}{1+e^{-\beta u}} = \frac{1}{1+e^{-u}}, \text{for simplicity set } \beta = 1$$

http://mathworld.wolfram.com/SigmoidFunction.html

$$\frac{df(u)}{du} = f^{'}(u)$$

Hence

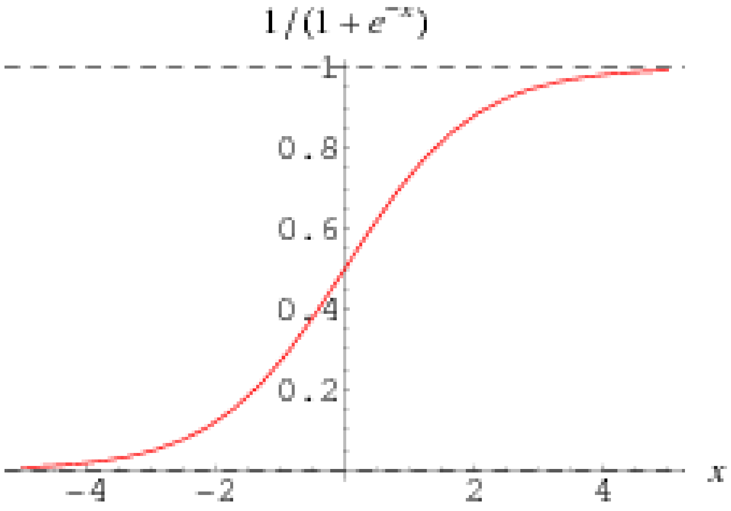
$$f^{'}(u) = \frac{df(u)}{du} = \frac{d\left(\frac{1}{1+e^{-u}}\right)}{d(1+e^{-u})} \frac{d(1+e^{-u})}{du}, \text{(using chain rule)}$$

$$f^{'}(u) = \frac{-1}{(1+e^{-u})^2}(-e^{-u}) = \frac{1}{(1+e^{-u})^2} e^{-u}$$

$$= \frac{1}{(1+e^{-u})} \frac{e^{-u}}{(1+e^{-u})} = \frac{1}{(1+e^{-u})} \frac{\left[(1+e^{-u}) - (1+e^{-u})\right] + e^{-u}}{(1+e^{-u})}$$

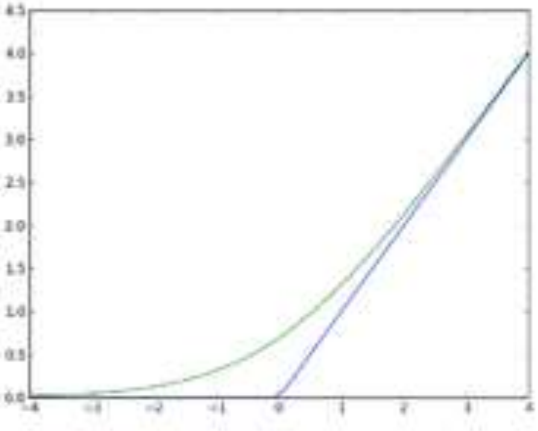
$$= \frac{1}{(1+e^{-u})}\left(1 - \frac{1}{(1+e^{-u})}\right) = f(u)(1-f(u))$$

$$\boxed{\text{Thus, } \frac{df(u)}{du} = f^{'}(u) = f(u)(1-f(u))}$$



http://link.springer.com/chapter/10.1007%2F3-540-59497-3_175#page-1

Rectified Linear Units



Relu (x) = MAX (0 , x)

More efficient gradient propagation, derivative is 0 or constant, just fold into learning rate

More efficient computation: Only comparison, addition and multiplication.

Leaky ReLU *f*(*x*) = *x* if > 0 else *ax* where 0 ≤ *a* <= 1, so that derivate is not 0 and can do some learning for that case.

Lots of other variations

Sparse activation: For example, in a randomly initialized networks, only about 50% of hidden units are activated (having a non-zero output)

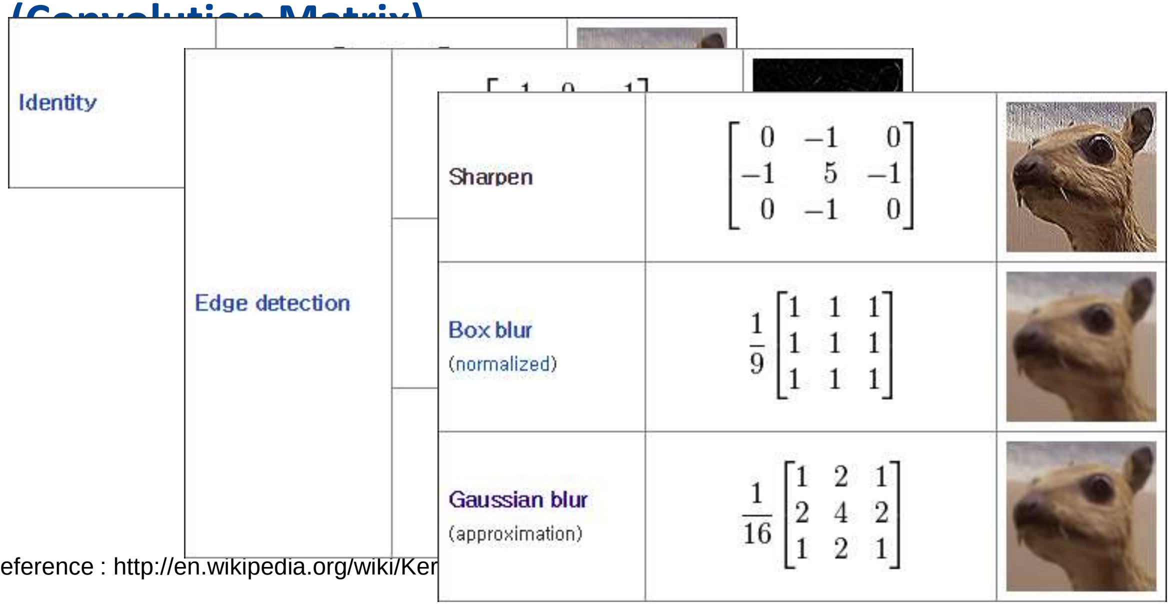
CONVOLUTIONAL LAYER

Background

Convolutional Neural Networks

Variation of multi-layer neural networks

Kernel (Convolution Matrix)

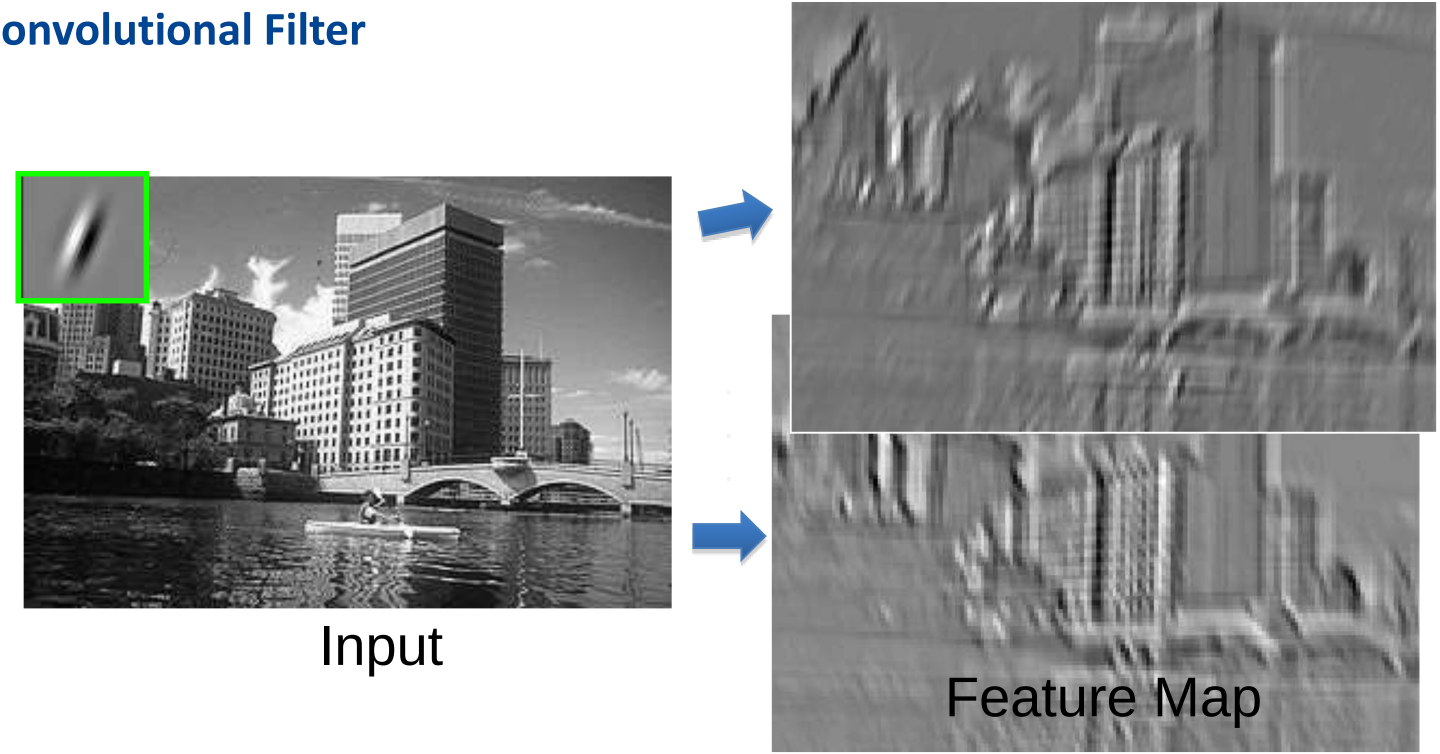


Identity		
Edge detection		
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

reference : http://en.wikipedia.org/wiki/Ker

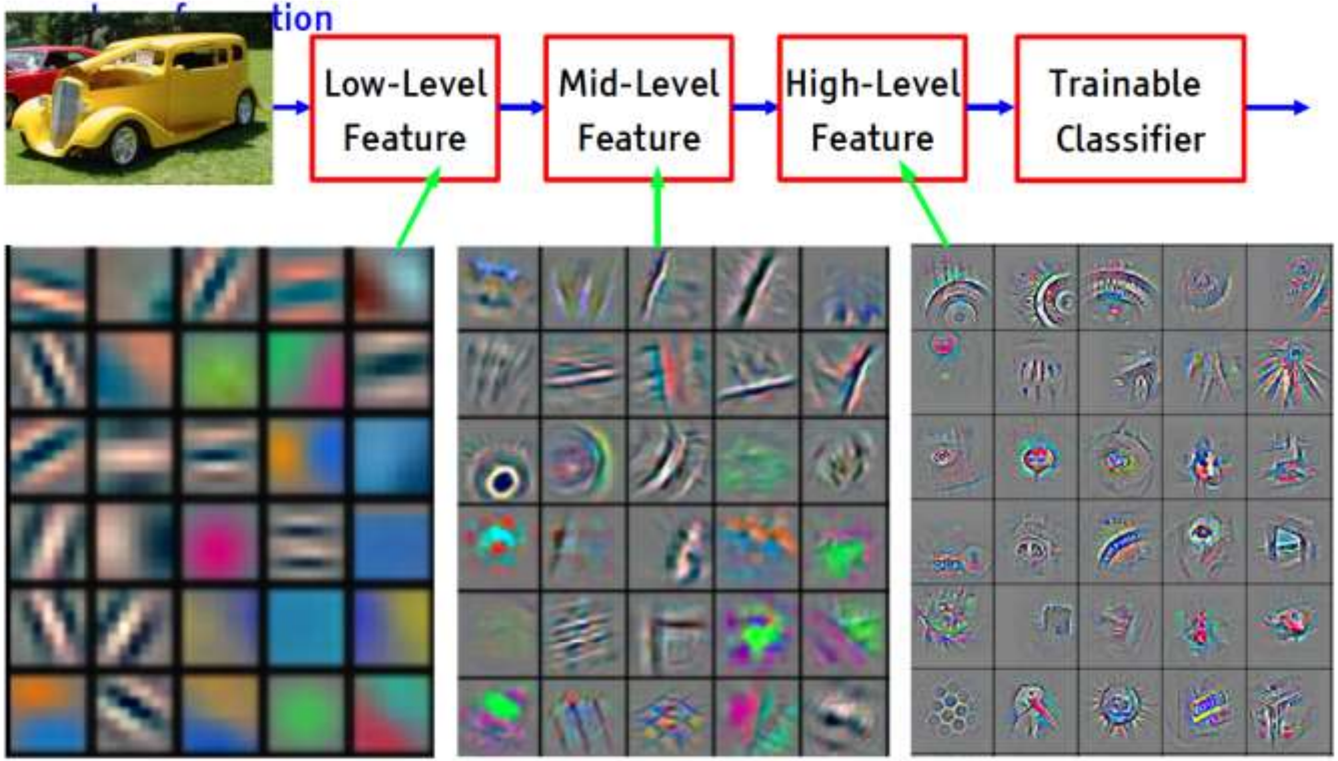
Background

Convolutional Filter



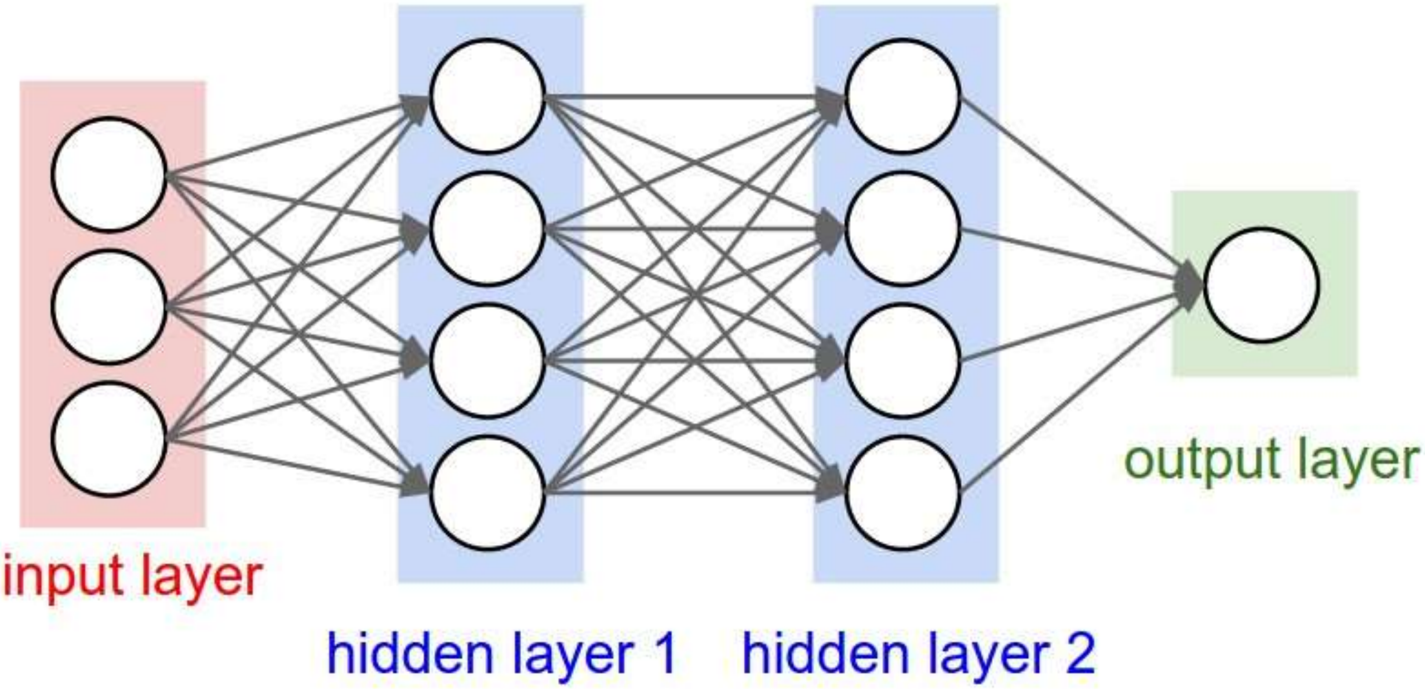
reference : http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/fergus_dl_tutorial_final.pptx

Deep Learning = Learning Hierarchical Representations



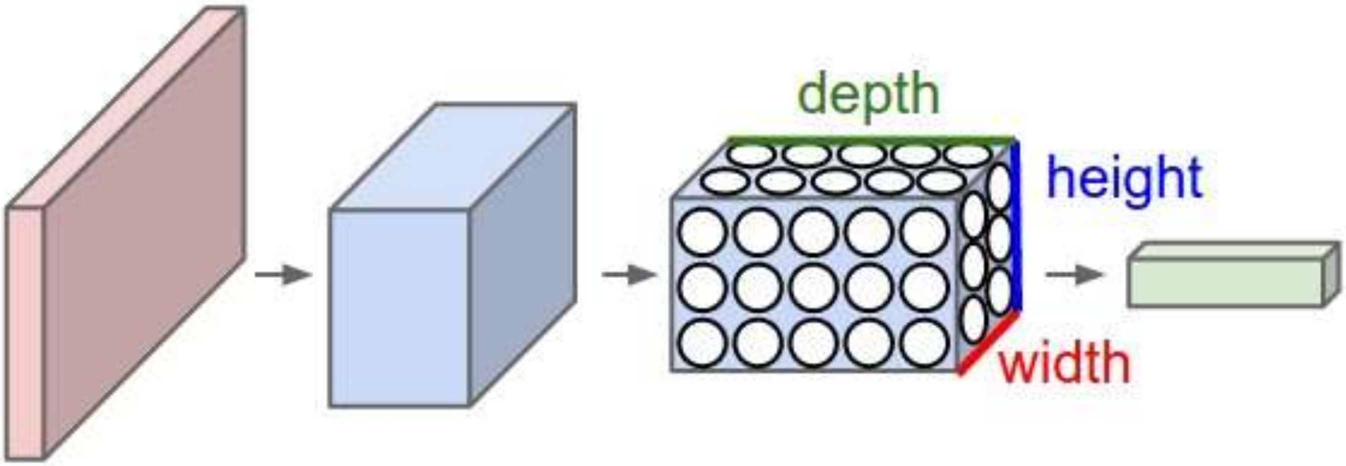
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

A regular 3-layer Neural Network



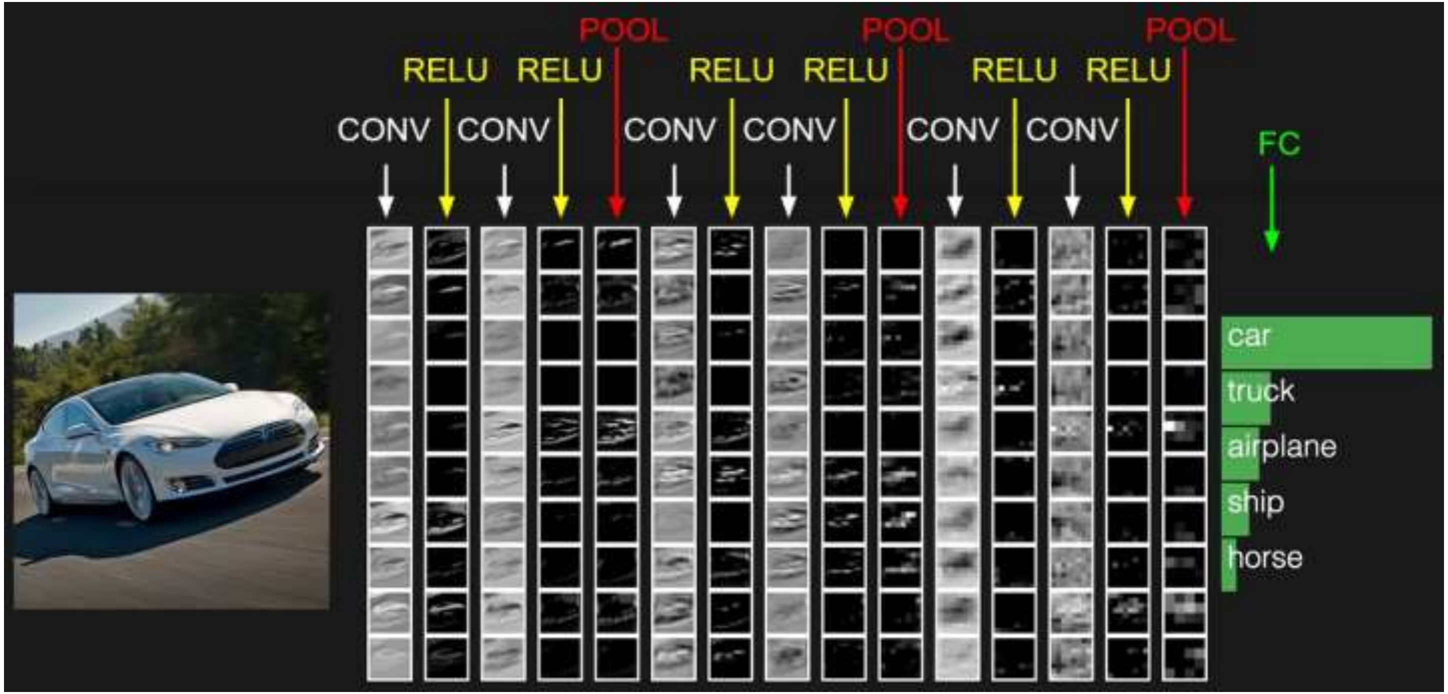
http://cs231n.github.io/convolutional-networks/

A ConvNet looks at data in three dimensions (width, height, depth)



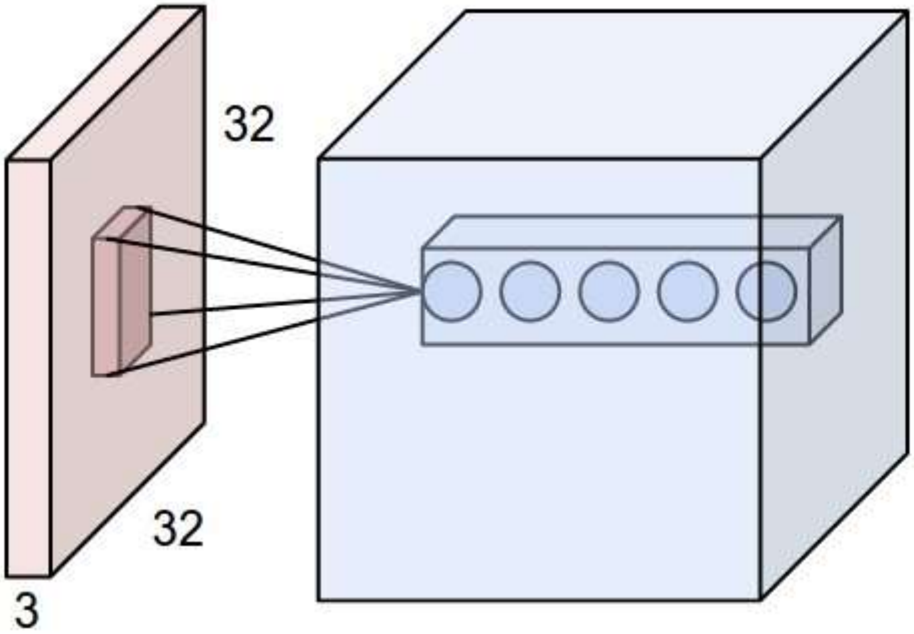
http://cs231n.github.io/convolutional-networks/

The activations of an example ConvNet architecture.



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

ConvNets

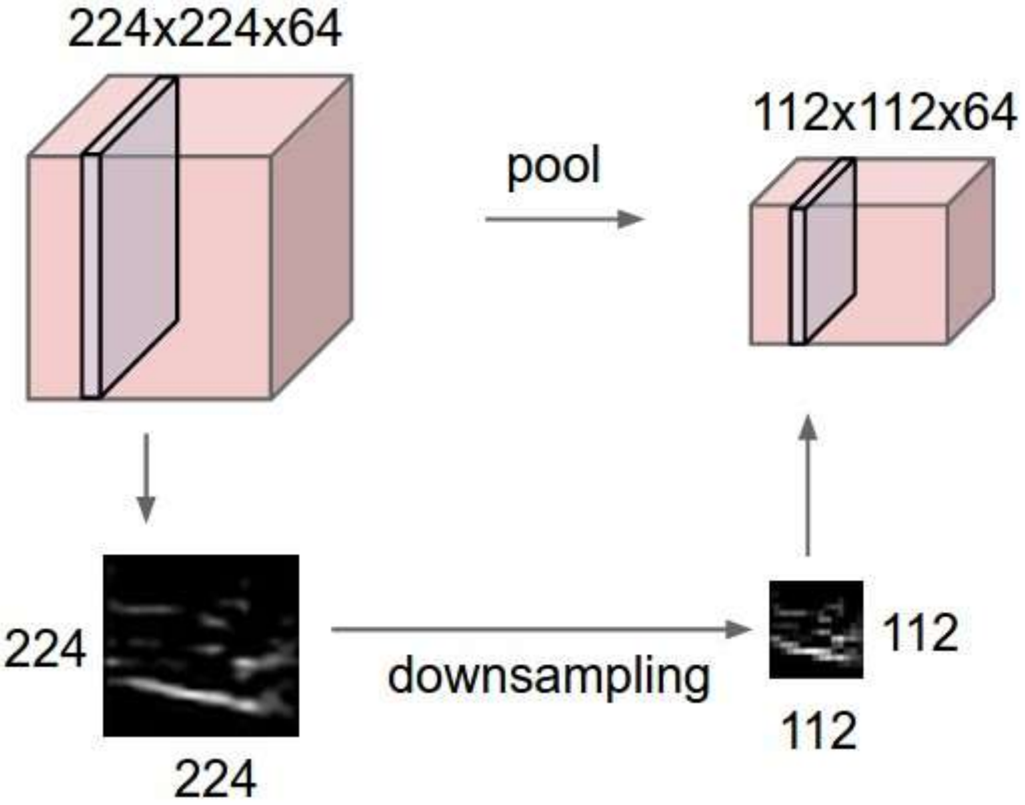


Input : “InDim” images with size H,W
A Convolution Layer : “outDim” filters with kernel size InDim x KsizeH x KsizeW.
Output : “outDim” images with size H,W (if stride == 1)

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

POOLING LAYER

ConvNets



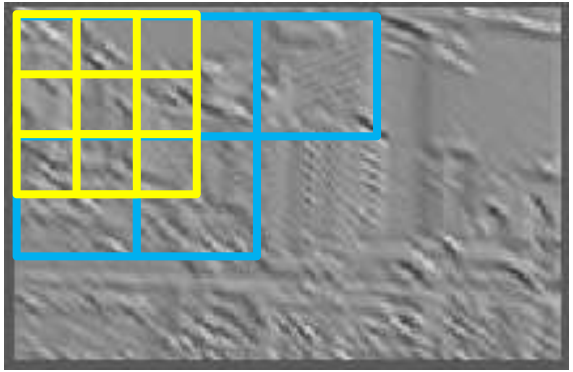
http://cs231n.github.io/convolutional-networks/

Pooling

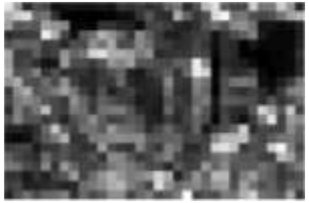
Spatial Pooling

Non-overlapping / overlapping regions

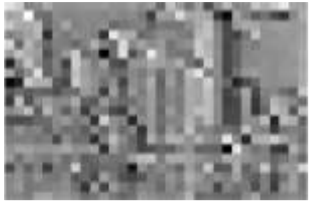
Average or Max



Max

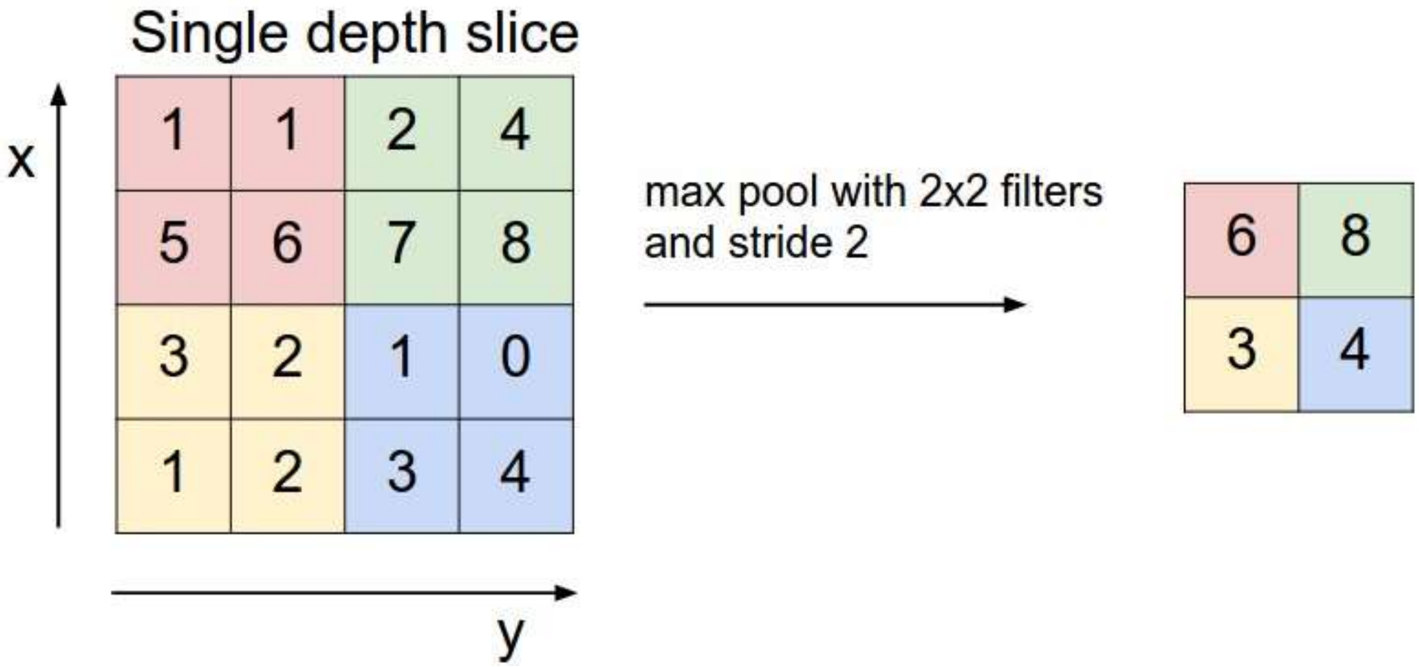


Avg



reference : http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/fergus_dl_tutorial_final.pptx

ConvNets
max pooling



Single depth slice

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters and stride 2

6	8
3	4

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

LOSS LAYER

Minimum Mean Squared Error

Works generally well for learning tasks

$$\mathcal{L} = \sum_n \left(y^{(n)} - t^{(n)}\right)^2$$

$y^{(n)}$ **- output of the classifier**
$t^{(n)}$ **- labels**

$$y^{(n)} = F\left(\boldsymbol{x}^{(n)}\right)$$
$$F \in \mathcal{F}$$

$\mathcal{F}$ **- the set of all functions representable by the neural network architecture.**

$$\min_{F \in \mathcal{F}} \sum\nolimits_n \left(F\left(\boldsymbol{x}^{(n)}\right) - t^{(n)}\right)^2$$

Softmax function

Definition

$$\boldsymbol{y}^{(n)}(i) = \frac{\exp\left(-\boldsymbol{a}^{(n)}(i)\right)}{\sum_j \exp\left(-\boldsymbol{a}^{(n)}(j)\right)}$$

It can be easily verify that

$$\sum_i \boldsymbol{y}^{(n)}(i) = 1$$

Cross Entropy

Definition: $p_X(i), q_X(i)$

$$H(p_X|q_X) = -\sum_i p_X(i) \log q_X(i)$$

The cross entropy loss is closely related to the Kullback-Leibler divergence between the empirical distribution and the predicted distribution.

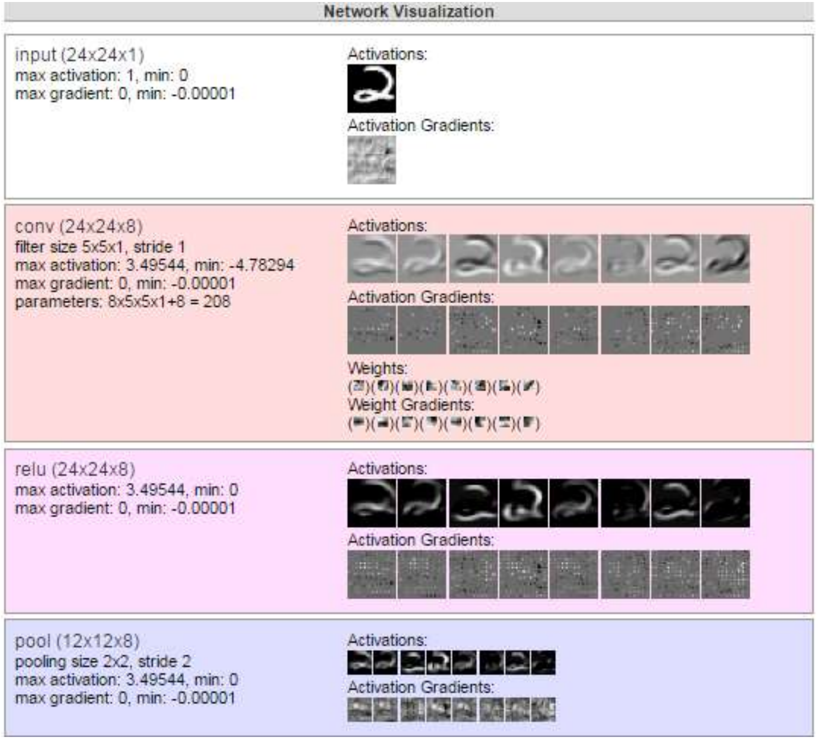
Cross Entropy as a Loss Function:

$$\ell^{(n)} = H\left(t^{(n)}|y^{(n)}\right) = -\sum_i t^{(n)}(i) \log y^{(n)}(i)$$

$$\mathcal{L} = \sum_n \ell^{(n)} = -\sum_n \sum_i t^{(n)}(i) \log y^{(n)}(i)$$

Another CNN visualization

https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html



Network Visualization

input (24x24x1)
max activation: 1, min: 0
max gradient: 0, min: -0.00001

Activations:
Activation Gradients:

conv (24x24x8)
filter size 5x5x1, stride 1
max activation: 3.49544, min: -4.78294
max gradient: 0, min: -0.00001
parameters: 8x5x5x1+8 = 208

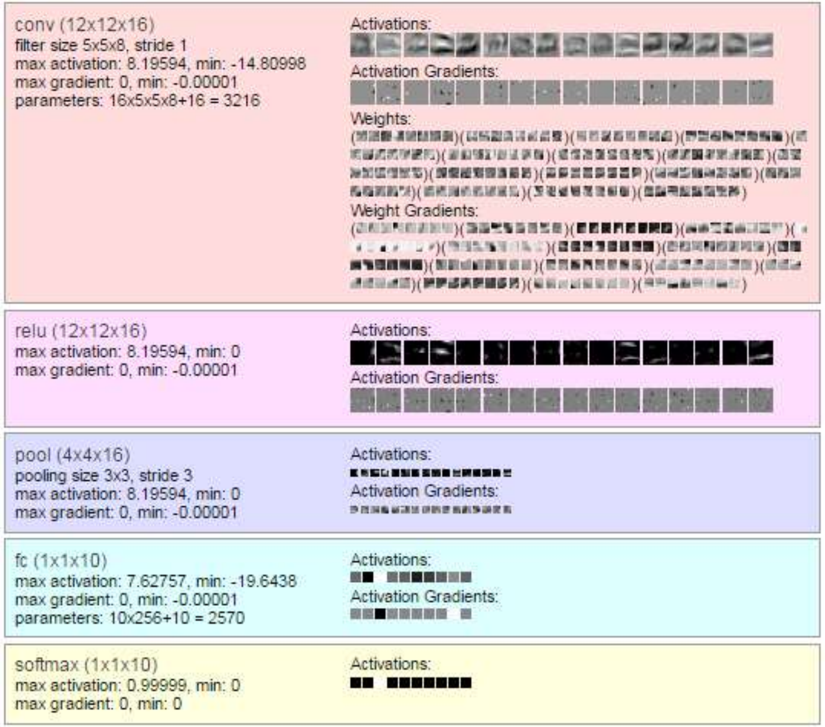
Activations:
Activation Gradients:
Weights:
Weight Gradients:

relu (24x24x8)
max activation: 3.49544, min: 0
max gradient: 0, min: -0.00001

Activations:
Activation Gradients:

pool (12x12x8)
pooling size 2x2, stride 2
max activation: 3.49544, min: 0
max gradient: 0, min: -0.00001

Activations:
Activation Gradients:



conv (12x12x16)
filter size 5x5x8, stride 1
max activation: 8.19594, min: -14.80998
max gradient: 0, min: -0.00001
parameters: 16x5x5x8+16 = 3216

Activations:
Activation Gradients:
Weights:
Weight Gradients:

relu (12x12x16)
max activation: 8.19594, min: 0
max gradient: 0, min: -0.00001

Activations:
Activation Gradients:

pool (4x4x16)
pooling size 3x3, stride 3
max activation: 8.19594, min: 0
max gradient: 0, min: -0.00001

Activations:
Activation Gradients:

fc (1x1x10)
max activation: 7.62757, min: -19.6438
max gradient: 0, min: -0.00001
parameters: 10x256+10 = 2570

Activations:
Activation Gradients:

softmax (1x1x10)
max activation: 0.99999, min: 0
max gradient: 0, min: 0

Activations:

CNN Building Blocks

- **Fully Connected layer**

```
tensor = tf.matmul(tensor, W) + B
```

- **Activation layer**

```
tensor = tf.nn.relu(tensor)
```

- **Convolutional Layers**

```
tensor = tf.nn.conv2d(tensor, W, strides, padding) + B
```

- **Pooling Layers**

```
tensor = tf.nn.max_pool(tensor, ksize, strides,padding
```

- **Loss Layer**

```
tensor = tf.nn.log_softmax(tensor)
```

nn4 : a CNN

Definition

```
t = Layers.unflat(x,48,48,1)
t = Layers.conv(t,64,3,1,'conv_1')
t = Layers.maxpool(t,2,'pool_2')
t = Layers.conv(t,32,3,1,'conv_3')
t = Layers.maxpool(t,2,'pool_4')
t = Layers.conv(t,16,3,1,'conv_5')
t = Layers.maxpool(t,2,'pool_6')
t = Layers.flat(t)
y = Layers.fc(t,2,'fc_7',tf.nn.log_softmax)
```

nn4 : a CNN

Total number of parameters learned in nn4 ?

Conv 1 => 3*3*1*64 + 64
Conv 3 => 3*3*64*32 + 32
Conv 5 => 3*3*32*16 + 16
FC 7 => 6*6*16*2 + 2
Total = 24882

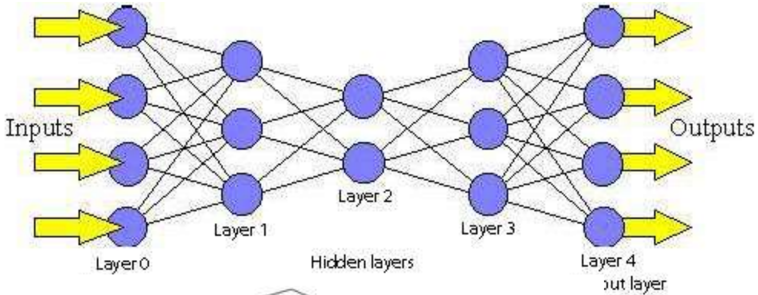
Tensor shape as input for Conv 3 ?

In Dataset.py, default batchsize = 128

Shape = (128,32,32,64)

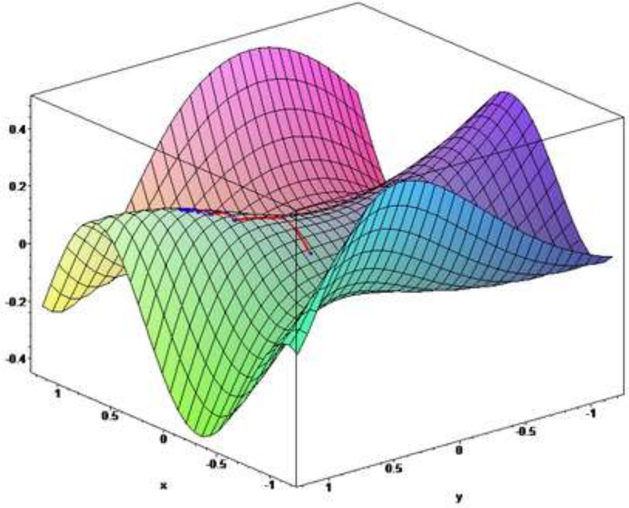
OPTIMIZATION

Feed forward/Backpropagation Neural Network



Feed forward algorithm:

Activate the neurons from the bottom to the top.



Gradient Descent Method is a first-order optimization algorithm. To find a local minimum of a function, one takes a step proportional to the negative of the gradient of the function at the current point.

Gradient Descent

- Neural network: $\mathrm{f}(x;\theta)$
  - x: input (vector)
  - θ: model parameters (weights, biases)
- Cost Function: $C(\theta)$
  - How good your model parameters θ is
  - $C(\theta) = \sum_{(x,\hat{y})\in\mathbb{T}} C_x(\theta)$
  - $C_x(\theta) = l(\mathrm{f}(x;\theta), \hat{y})$
    - $\hat{y}$: reference output when x is input
    - l$(\mathrm{f}(x;\theta), \hat{y})$: loss function between $\hat{y}$ and the output of neural network with model parameters θ
- Target: find θ that minimizes $C(\theta)$

Gradient Descent

Taylor series by definition

$$C(\theta_{new}) = C(\theta_{old}) + \nabla C \cdot [\theta_{new} - \theta_{old}] + ..$$

Here $\Delta\theta = [\theta_{new} - \theta_{old}], \nabla C = \frac{\partial C}{\partial \theta}$

$$C(\theta_{new}) \approx C(\theta_{old}) + \frac{\partial C}{\partial \theta} \cdot \Delta\theta \qquad (1)$$

If we set $\Delta\theta = -\eta \frac{\partial C}{\partial \theta}$ with η a small positive learning rate

equation (1) becomes:

$$C(\theta_{new}) \approx C(\theta_{old}) + \frac{\partial C}{\partial \theta} \cdot \left(-\eta \frac{\partial C}{\partial \theta}\right) = C(\theta_{old}) - \eta \left(\frac{\partial C}{\partial \theta}\right)\left(\frac{\partial C}{\partial \theta}\right)$$

$C(\theta_{new}) \leq C(\theta_{old})$, since $\eta \left(\frac{\partial C}{\partial \theta}\right)\left(\frac{\partial C}{\partial \theta}\right)$ is always positive

Conclusion: if we set $\Delta\theta = -\eta \frac{\partial C}{\partial \theta}$ it will decrease C

- Target: find θ that minimizes $C(\theta)$

- Need to Compute $\frac{\partial C}{\partial \theta}$

How to back propagate?

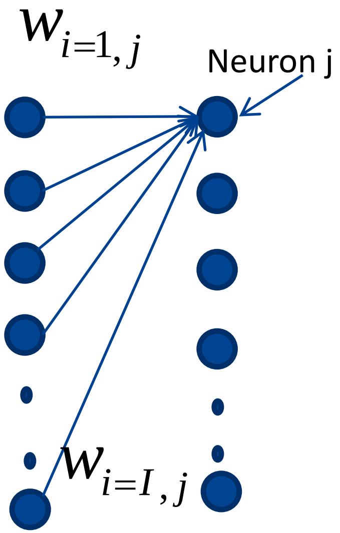
For a neuron j, Output is y_j

By definition, $u_j = \sum_{i=1}^{i=I} x_i w_{ij} + b_j$

$$y_j = f(u_j) = f\left(\sum_{i=1}^{i=I} x_i w_{ij} + b_j\right)$$

We want to find $\frac{\partial C}{\partial w_{ij}}$, so

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial y_j} \frac{\partial y_j}{\partial u_j} \frac{\partial u_j}{\partial w_{ij}}$$, by chain rule we will compute all derivative



i=1,2,..,I

I inputs to neuron j

Output of neuron j is y_j

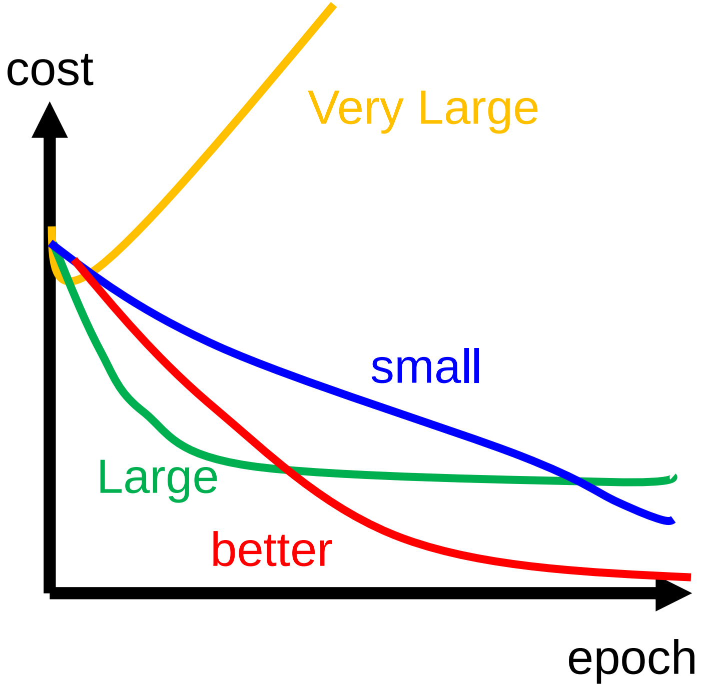
Gradient Descent variants

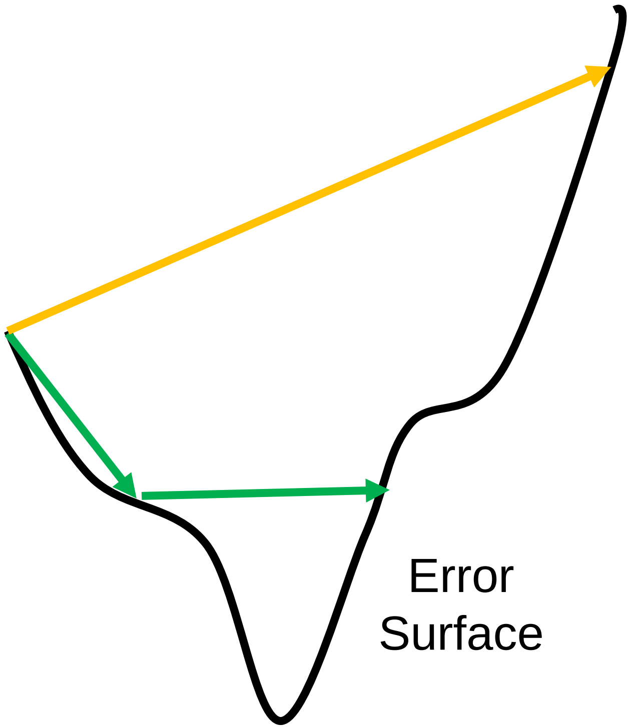
- Target: find θ that minimizing $C(\theta)$
  - $C(\theta) = \sum_{(x,\hat{y})\in\mathbb{T}} C_x(\theta)$
- Gradient descent
  - Initialization: θ^0
  - $\theta^t \leftarrow \theta^{t-1} - \eta\nabla C(\theta^{t-1})$
- SGD = Stochastic gradient descent
  - Pick $(x, \hat{y})$ from training data set $\mathbb{T}$
  - $\theta^t \leftarrow \theta^{t-1} - \eta\nabla C_x(\theta^{t-1})$

Tensorflow works with batches.
Gradients in a batch are computes in parallel on GPU.

Learning Rates

- $\theta^t \leftarrow \theta^{t-1} - \eta \nabla C(\theta^{t-1})$
- What is a good learning rate η ?





Learning Rates

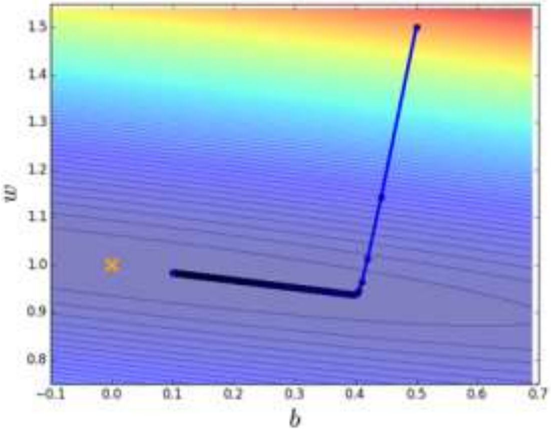
Popular & Simple Idea: Reduce the learning rate by some factor every few epochs.

At the beginning, we are far from a minimum, so we use larger learning rate

After several epochs, we are close to a minimum, so we reduce the learning rate

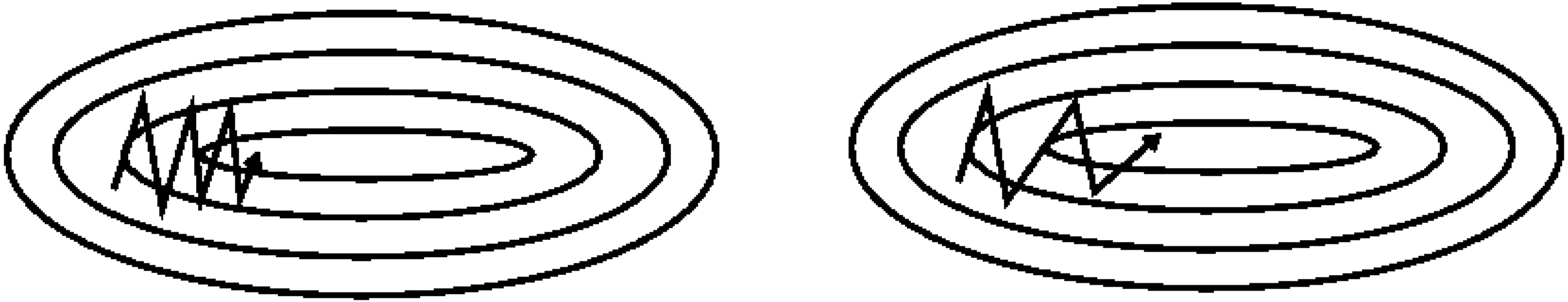
E.g. 1/t decay: $\eta^t = \eta/(t+1)$

Not always true



Momentum

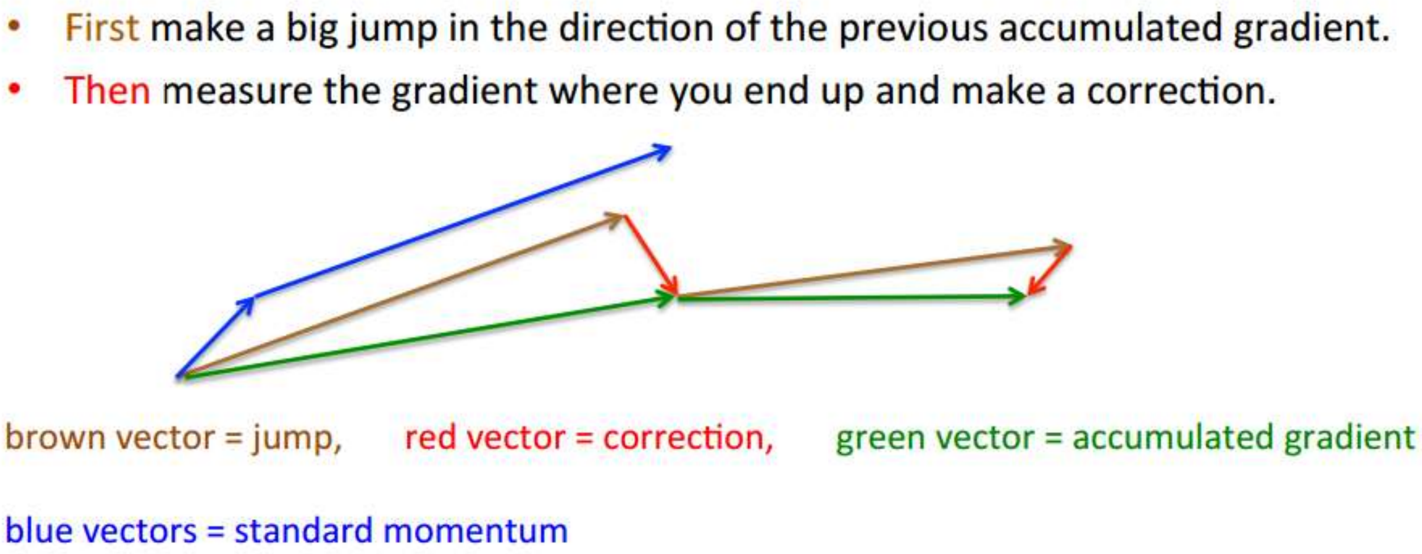
Momentum is a method that helps accelerate SGD in the relevant direction and dampens oscillations. It does this by adding a fraction of the update vector of the past time step to the current update vector:



http://doi.org/10.1016/S0893-6080(98)00116-6

A better type of momentum (Nesterov 1983)

The standard momentum method first computes the gradient at the current location and then takes a big jump in the direction of the updated accumulated gradient.



- First make a big jump in the direction of the previous accumulated gradient.
- Then measure the gradient where you end up and make a correction.

brown vector = jump, red vector = correction, green vector = accumulated gradient

blue vectors = standard momentum

Adagrad

In a multilayer net, the magnitudes of the gradients are often very different for different layers, especially if the initial weights are small. The appropriate learning rates can vary widely between weights.

$$w^{t+1} \leftarrow w^{t} - \frac{\eta}{\sigma} g^{t}$$

σ: Average gradient of parameter w

If w has small average gradient σ → Larger learning rate

If w has large average gradient σ → Smaller learning rate

Adagrad

Divide the learning rate by “*average*” *gradient*

The “average” gradient is obtained while updating the parameters

$$\sigma^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t}(g^i)^2}$$

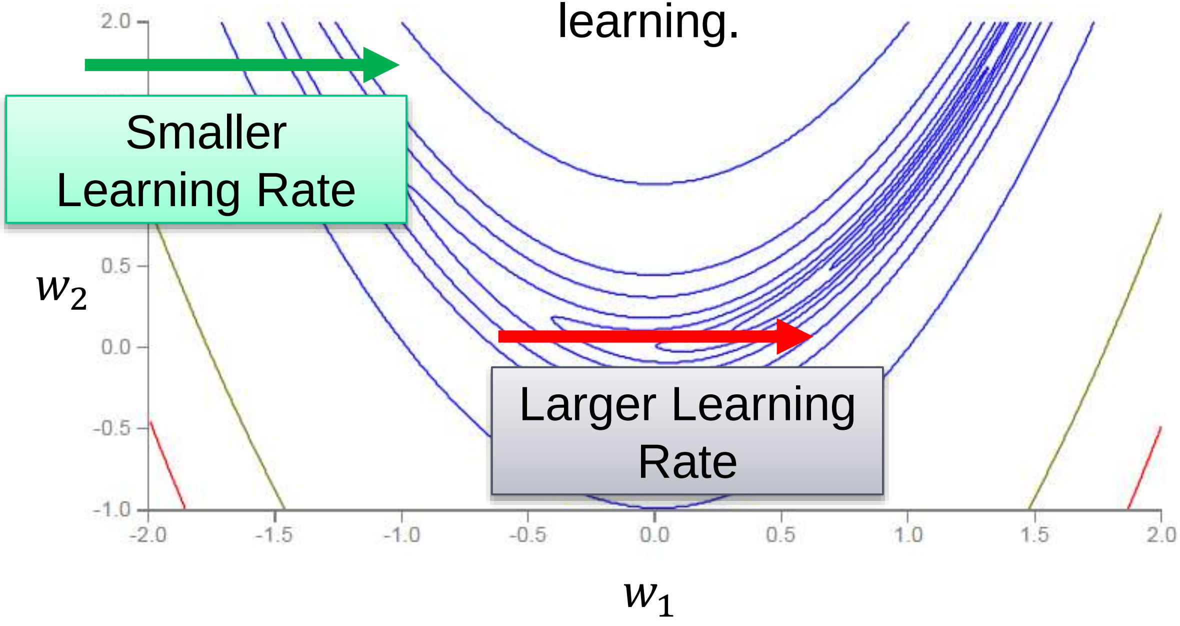
$$\eta^t = \frac{\eta}{\sqrt{t+1}}$$

$$w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t$$

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^{t}(g^i)^2}} g^t$$

RMSProp

Error Surface can be complex in deep learning.



Smaller Learning Rate

Larger Learning Rate

RMSProp

Same as adagrad , but with an exponentially decaying average

$$w^1 \leftarrow w^0 - \frac{\eta}{\sigma^0} g^0 \qquad \sigma^0 = g^0$$

$$w^2 \leftarrow w^1 - \frac{\eta}{\sigma^1} g^1 \qquad \sigma^1 = \sqrt{\alpha(\sigma^0)^2 + (1-\alpha)(g^1)^2}$$

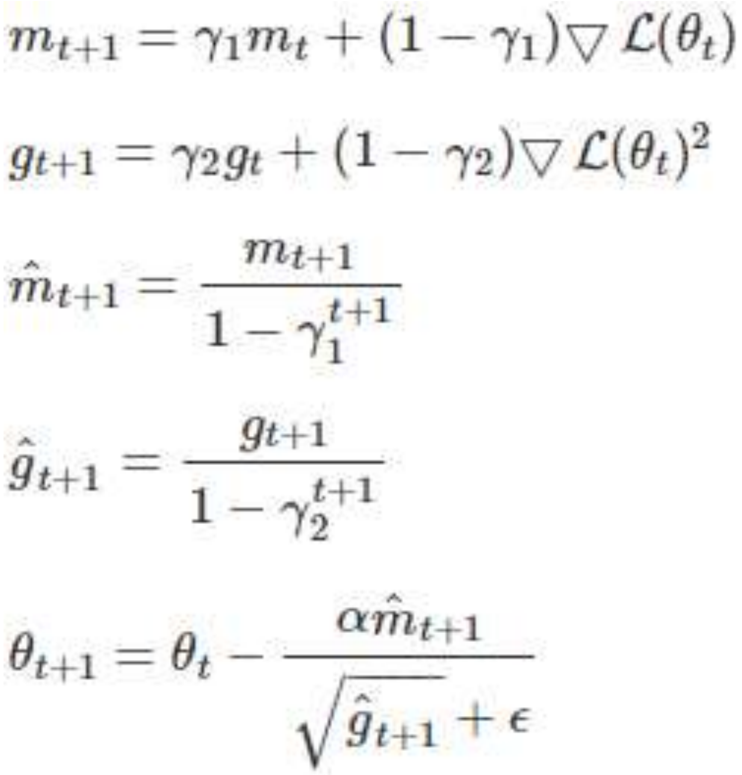
$$w^3 \leftarrow w^2 - \frac{\eta}{\sigma^2} g^2 \qquad \sigma^2 = \sqrt{\alpha(\sigma^1)^2 + (1-\alpha)(g^2)^2}$$

$$\vdots$$

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sigma^t} g^t \qquad \sigma^t = \sqrt{\alpha(\sigma^{t-1})^2 + (1-\alpha)(g^t)^2}$$

Adam – ADAptive Moment estimation

Like RMSprop, but in addition to storing an exponentially decaying average of past squared gradients g_t, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum:



$$m_{t+1} = \gamma_1 m_t + (1 - \gamma_1) \triangledown \mathcal{L}(\theta_t)$$

$$g_{t+1} = \gamma_2 g_t + (1 - \gamma_2) \triangledown \mathcal{L}(\theta_t)^2$$

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1 - \gamma_1^{t+1}}$$

$$\hat{g}_{t+1} = \frac{g_{t+1}}{1 - \gamma_2^{t+1}}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha \hat{m}_{t+1}}{\sqrt{\hat{g}_{t+1}} + \epsilon}$$

Parameter initialization

Can be deceptively important!

Bias terms are often initialized to 0 with no issues

Weight matrices more problematic, ex.

- If initialized to all 0s, can be shown that the tanh activation function will yield zero gradients
- If the weights are all the same, hidden units will produce same gradients and behave the same as each other (wasting params)

One solution: initialize all elements of weight matrix from uniform distribution over interval [–b, b]

Different methods have been proposed for selecting the value of b, often motivated by the idea that units with more inputs should have smaller weights

Weight matrices of rectified linear units have been successfully initialized using a zero-mean isotropic Gaussian distribution with standard deviation of 0.01

Weight Initialization

Random from a distribution with zero mean and a specific variance

- **Standard initialization**

$$var(W) = \frac{1}{N^{in}}$$

- **“Xavier” initialization. picks std according to blob size.**

See “Understanding the difficulty of training deep feedforward neural networks”. Glorot and Bengio 2010.

$$var(W) = \frac{2}{N^{in} + N^{out}}$$

Weight Initialization

- Whitening

  A whitening transformation is a linear transformation that transforms a vector of random variables with a known covariance matrix into a set of new variables whose covariance is the identity matrix meaning that they are uncorrelated and all have variance 1.

  idea : avoid initialization with close filters

- Mishkin & Matas. All you need is a good init. https://arxiv.org/pdf/1511.06422

  init W with gaussian distribution

  SVD on W

  minibatch, rescale to have variance 1, iterates on layers

Unsupervised pre-training

Idea: model the distribution of unlabeled data using a method that allows the parameters of the learned model to inform or be somehow transferred to the network

Can be an effective way to both initialize and regularize a feedforward network

Particularly useful when the volume of labeled data is small relative to the model’s capacity.

The use of activation functions such as rectified linear units (which improve gradient flow in deep networks), along with good parameter initialization techniques, can mitigate the need for sophisticated pre-training methods

OVERFITTING

Overfitting

We want to builds models that performs well on unknown data, by only learning on known data !

Different levels on overfitting exists :

- **Overfitting on the learning data.**

  Technical solutions exist to prevent this. (we will see this !)

- **Overfitting by experience.**

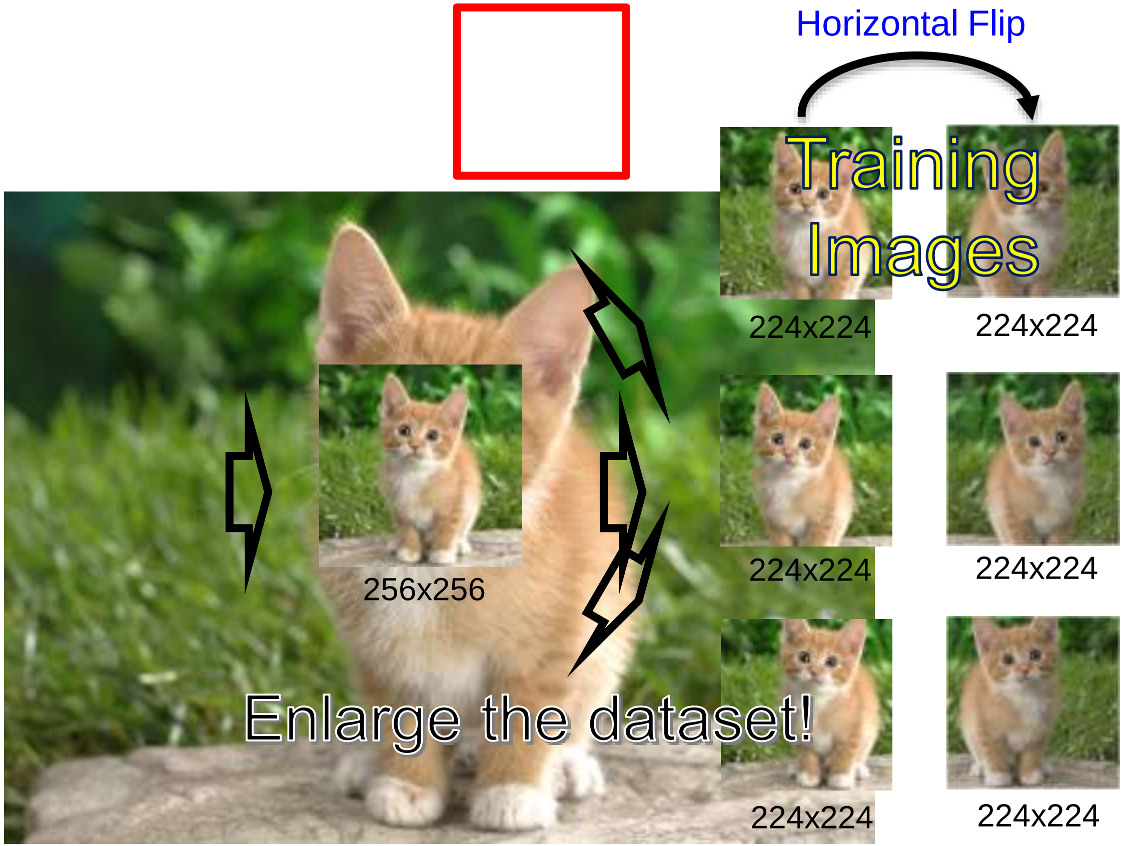
  Working too long on the same test data, may lead to tune meta-parameters and algorithmic choices.

- **Overfitting on a scenario.**

  Test data are drawn from the same distribution as the training data.

  Test performances are ok, but not on other use-cases

Data Augmentation



Training Image

Dropout

Independently set each hidden unit activity to zero with α probability

- **Intention: reducing hidden unit co-adaptation & combat over-fitting**
- **Has been argued it corresponds to sampling from an exponential number of networks with shared parameters & missing connections**
- **One averages over models at test time by using original network without dropped-out connections, but with scaled-down weights**
- **If a unit is retained with probability *p* during training, its outgoing weights are rescaled or multiplied by a factor of *p* at test time**
- **By performing dropout a neural network with *n* units can be made to behave like an ensemble of 2^n smaller networks**
- **Often only used in the fully-connected layers at the net's output**

Regularization

- To avoid over-fitting, it is possible to regularize the cost function.
- Here we use L2 regularization, by changing the cost function to:

$$\widetilde{L}(\mathbf{w}) = L(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^2$$

- In practice this penalizes large weights and effectively limits the freedom in the model.
- The regularization parameter λ determines how you trade off the original loss L with the large weights penalization.

- Applying gradient descent to this new cost function we obtain:

$$w_i \leftarrow w_i - \eta \frac{\partial L}{\partial w_i} - \eta \lambda w_i$$

- The new term $-\eta\lambda w_i$ coming from the regularization causes the weight to decay in proportion to its size.

Early stopping

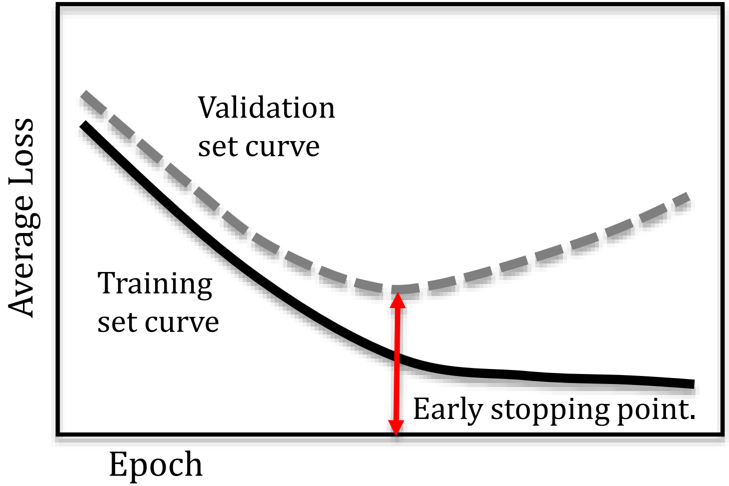
Deep learning involves high capacity architectures, which are susceptible to overfitting even when data is plentiful,

Early stopping is standard practice even when other methods to reduce overfitting are employed, ex. regularization and dropout

The idea is to monitor learning curves that plot the average loss for the training and validation sets as a function of epoch

The key is to find the point at which the validation set average loss begins to deteriorate

Early stopping

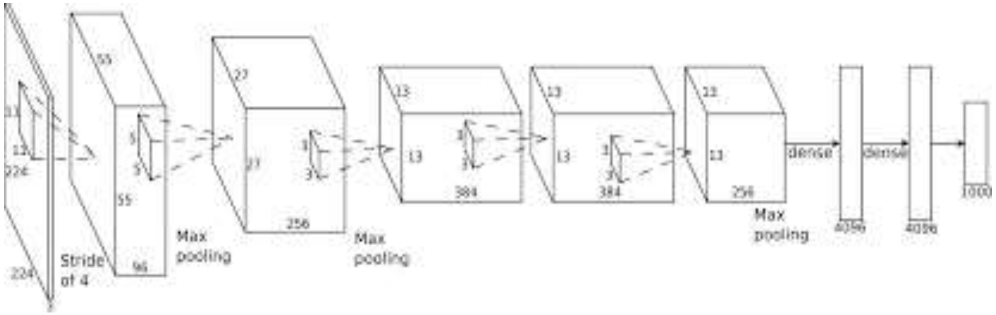


- In practice the curves above can be more noisy due to the use of stochastic gradient descent
- As such, it is common to keep the history of the validation set curve when looking for the minimum – even if it goes back up it might come back down

MORE DNN ARCHITECTURES

Batch Normalization

- Batch normalization comes from Sergey Ioffe and Christian Szegedy [ICML2015]



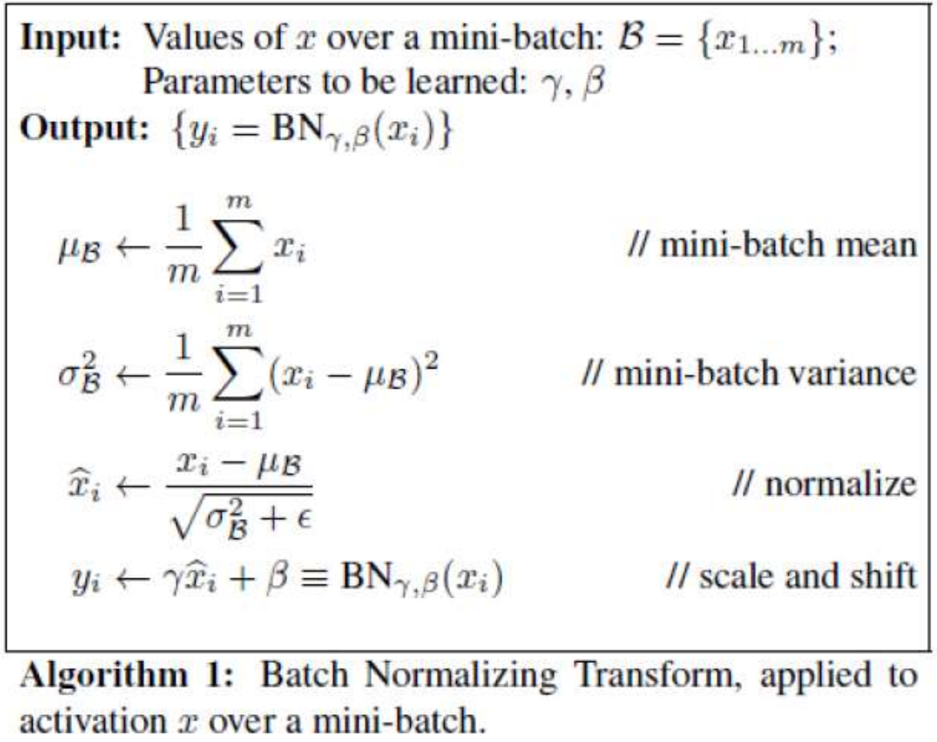
- We apply whitening to the input, and choose our initial weights in such a way that the activations in between are close to whitened (‘Xavier’ inialization).

It would be nice to also ensure whitened activations during training for all layers.

- Training of networks is complicated because the distribution of layer inputs changes during training *(internal covariate shift)*

Making normalization at all layers part of the training prevents the internal covariate shift.

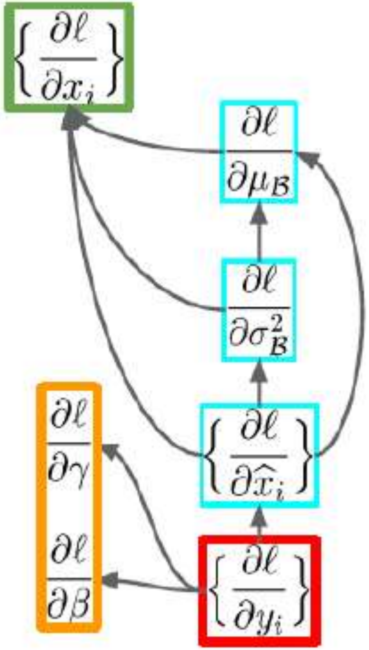
Batch Normalization



```
Input: Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
       Parameters to be learned: $\gamma$, $\beta$
Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

    $\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$                              // mini-batch mean
    $\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2$    // mini-batch variance
    $\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}$                // normalize
    $y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i)$          // scale and shift
```

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Batch Normalization



$\left\{\frac{\partial \ell}{\partial x_i}\right\}$
$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}}$
$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2}$
$\frac{\partial \ell}{\partial \gamma}$
$\frac{\partial \ell}{\partial \beta}$
$\left\{\frac{\partial \ell}{\partial \widehat{x}_i}\right\}$
$\left\{\frac{\partial \ell}{\partial y_i}\right\}$

(a)

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

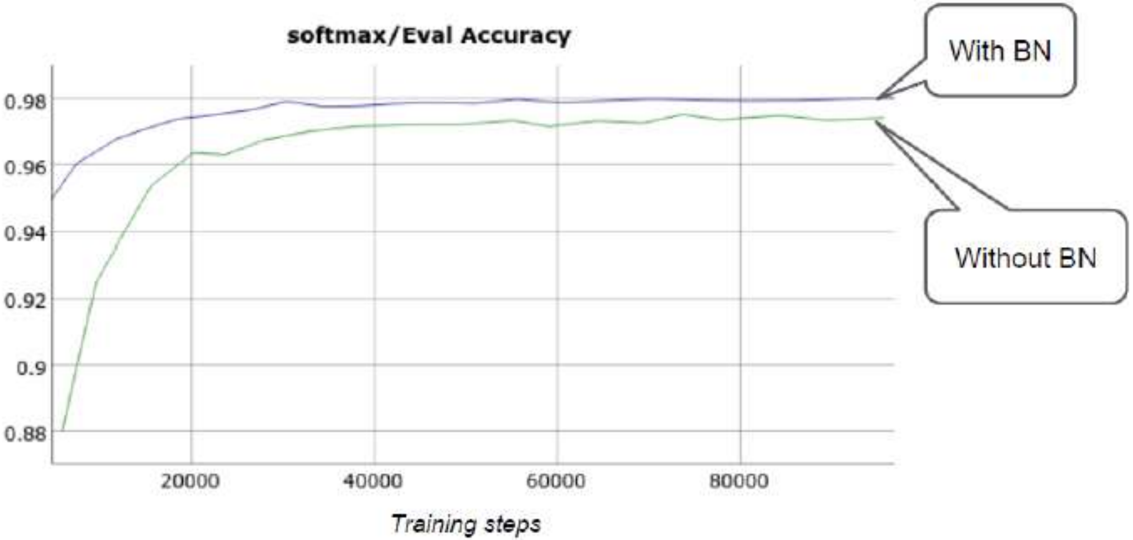
$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

(b)

Backpropagation with Batch Normalization

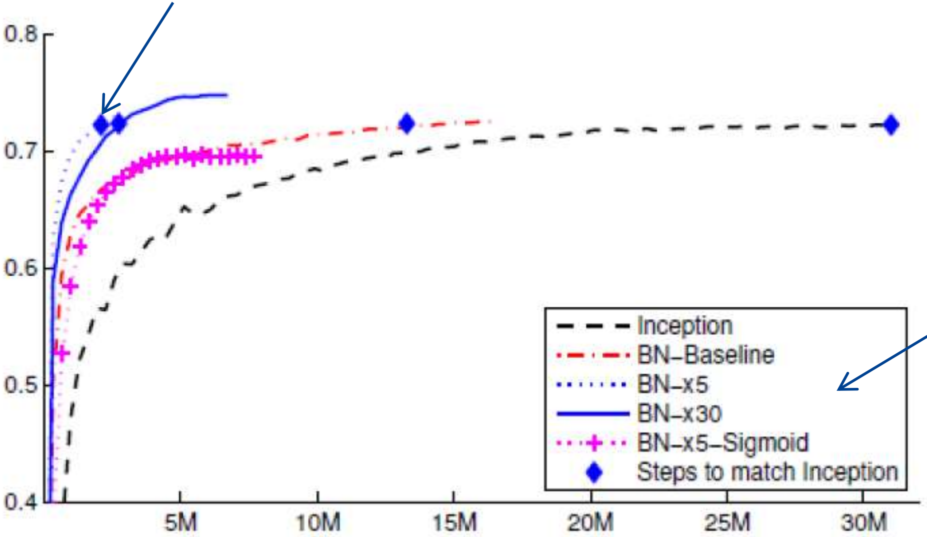
Batch Normalization



Results on MNIST [Ioffe & Szegedy2015]

Batch Normalization

14x faster to reach same results



Learning rate multiplied with 5 and 30.

Results on ImageNet [Ioffe & Szegedy2015]

Batch Normalization with tensorflow

Definition :

```
class BatchNormalizer(object):

def __init__(self, depth, decay=0.99, epsilon=0.01):
    self.beta  = tf.Variable(tf.constant(0.0, shape=[depth]), trainable=True)
    self.gamma = tf.Variable(tf.constant(1.0, shape=[depth]), trainable=True)
    self.ema_trainer = tf.train.ExponentialMovingAverage(decay=decay)
    self.epsilon = epsilon

def normalize(self, x, train):
    batch_mean, batch_var = tf.nn.moments(x, range(x.get_shape().ndims-1))
    def updateStat():
        ema_apply = self.ema_trainer.apply([batch_mean, batch_var])
        with tf.control_dependencies([ema_apply]):
            return tf.identity(batch_mean), tf.identity(batch_var)
    def loadStat():
        return self.ema_trainer.average(batch_mean), self.ema_trainer.average(batch_var)
    mean, var = tf.cond(train, updateStat, loadStat)
    bn = tf.nn.batch_normalization(x, mean, var, self.beta, self.gamma, self.epsilon)
    return bn
```

Usage :

```
local_bn = bn.BatchNormalizer(outDim)
tensor = local_bn.normalize(tensor, train=IsTrainingMode)
```

The deeper, the better

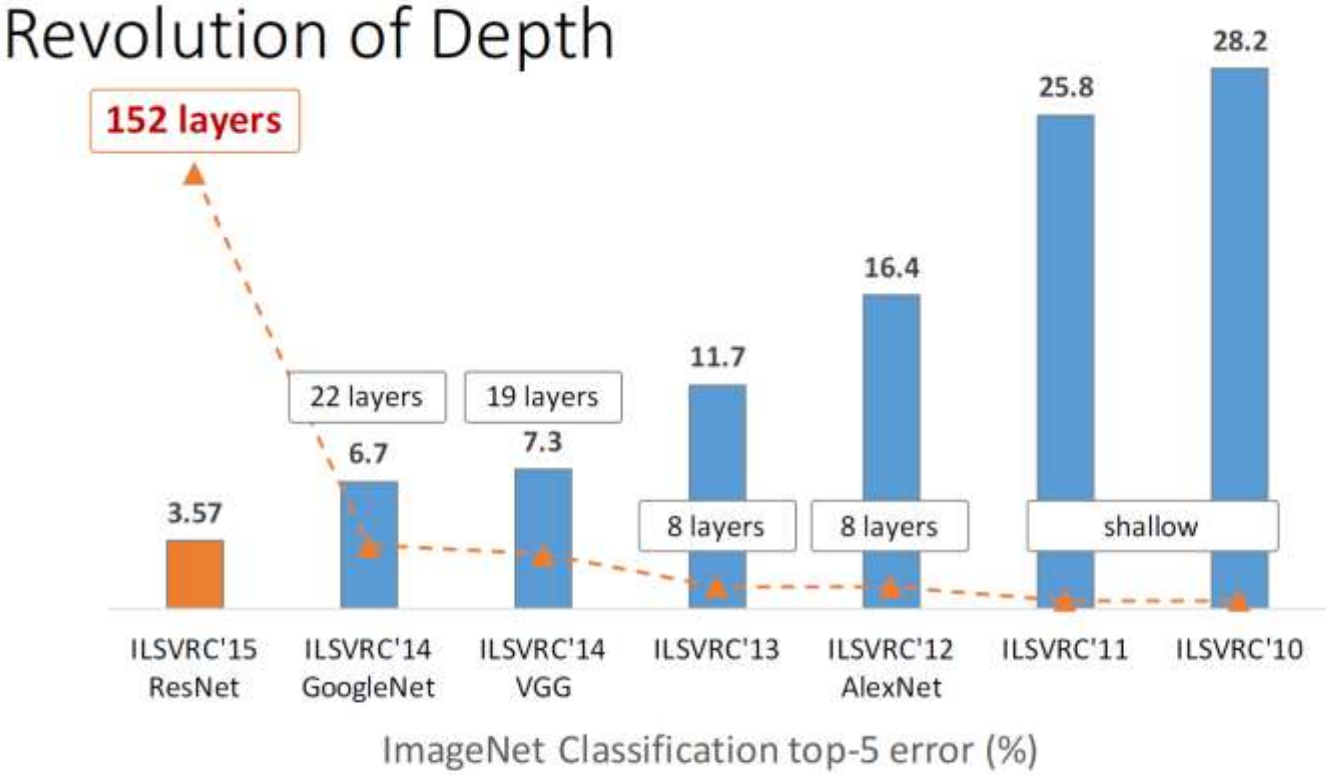
The deeper network can cover more complex problems

- Receptive field size ↑
- Non-linearity ↑

However, training the deeper network is more difficult because of vanishing/exploding gradients problem

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. arXiv 2015.

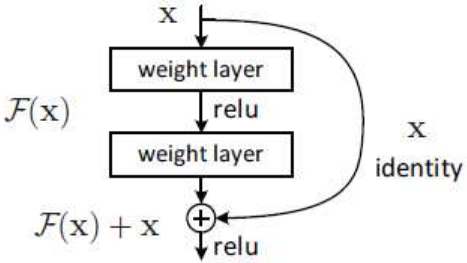
Evolution of deep networks



Residual Learning Block

Define H(x) = F(x) + x, the stacked weight layers try to approximate F(x) instead of H(x).

If the optimal function is close to identity



ing, the nonlinear stacked weight layers

apture the small perturbations easier.

① No extra parameter and computation complexity introduced.

② Element-wise addition is performed on all feature maps.

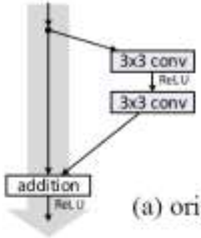
Residual Learning Block

What if shortcut mapping h(x) ≠ identity?

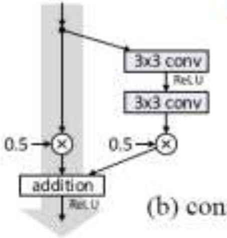
* ResNet-110 on CIFAR-10

$h(x) = x$

error: 6.6%



(a) original



(b) constant scaling

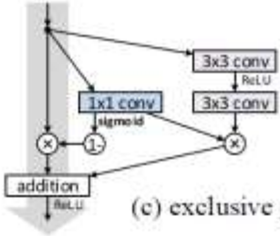
$h(x) = 0.5x$

error: 12.4%

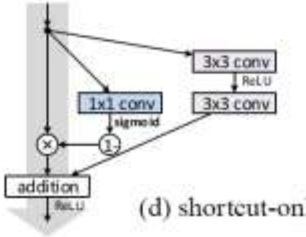
$h(x) = \text{gate} \cdot x$

error: 8.7%

*similar to "Highway Network"



(c) exclusive gating



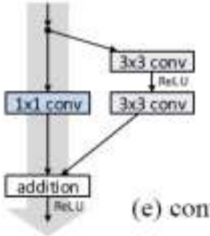
(d) shortcut-only gating

$h(x) = \text{gate} \cdot x$

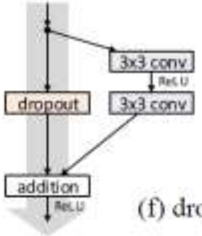
error: 12.9%

$h(x) = \text{conv}(x)$

error: 12.2%



(e) conv shortcut



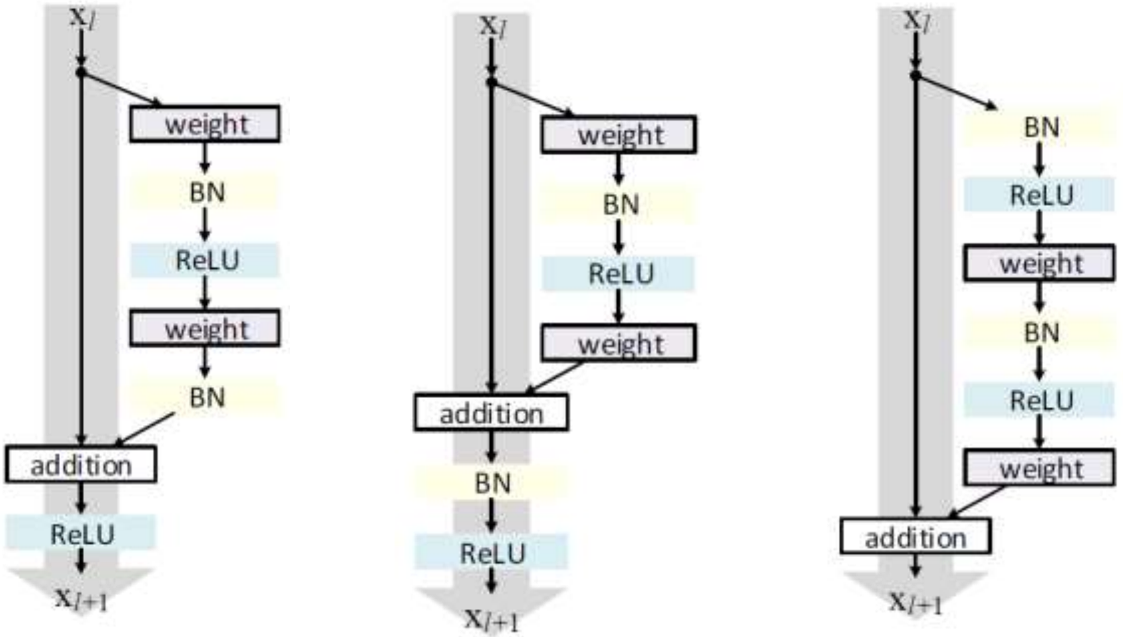
(f) dropout shortcut

$h(x) = \text{dropout}(x)$

error: > 20%

Residual Learning Block

If after-adding f(x) is identity mapping ?



Keep the shortest path as smooth(clean) as possible!

Deeper Neural Network

Escape from few layers

ReLU for solving gradient vanishing problem

Dropout …

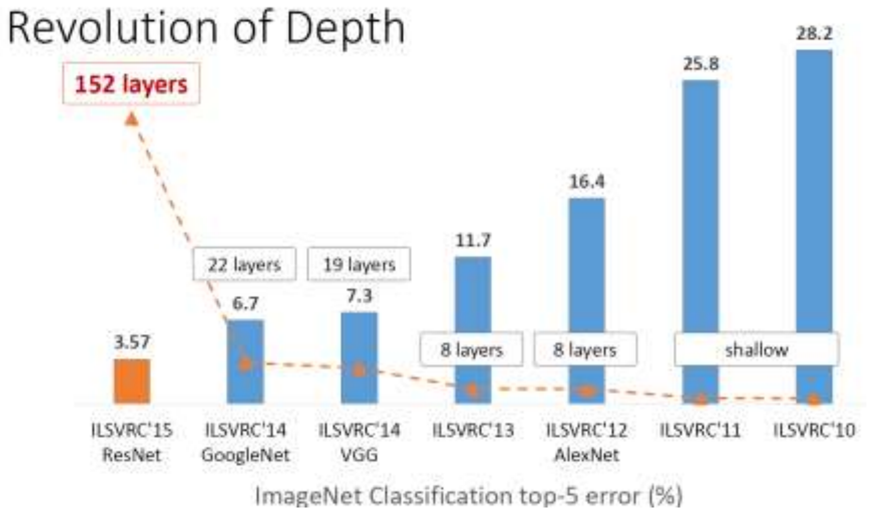
Escape from 10 layers

Normalized initialization

Intermediate normalization

Escape from 100 layers

Residual network layers



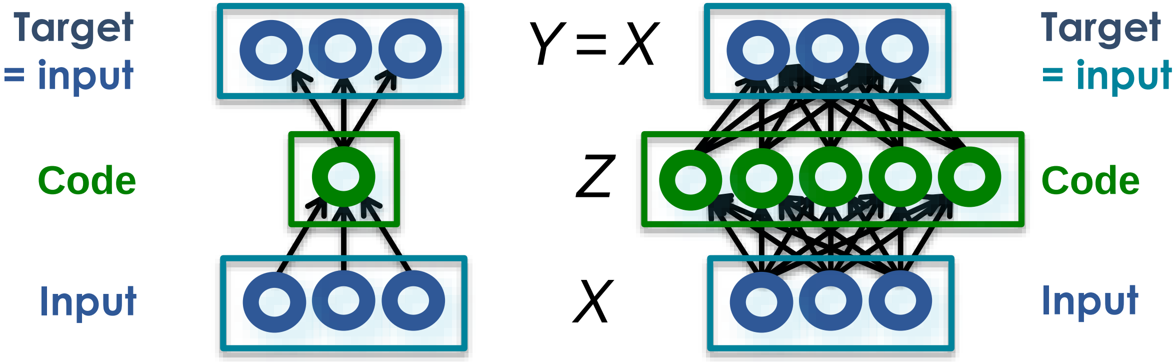
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. arXiv 2015.

Residual networks with tensorflow

```
def residual(tensor, outDim, filterSize, stride, …):
        tensor_lin = conv(tensor, outDim, 1, …)
        tensor = conv(tensor, outDim, filterSize, stride, …)
        tensor = conv(tensor, outDim, filterSize, stride, …)
        return tensor + tensor_lin
```

Solution for inDim ≠ outDim
Convolution with a 1x1 filter

Auto-encoder

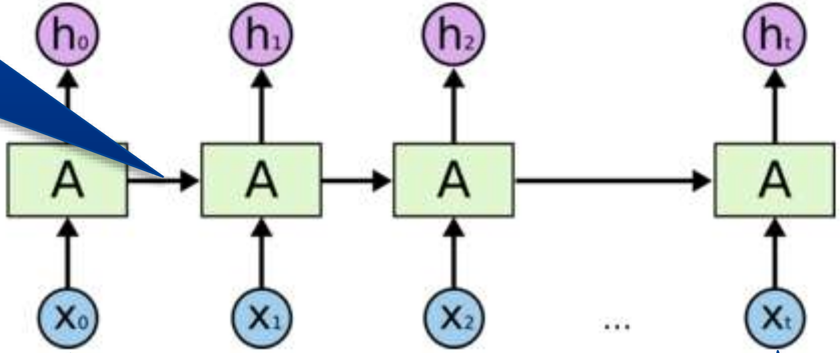


“Bottleneck” code
i.e., low-dimensional,
typically dense,
distributed
representation

“Overcomplete” code
i.e., high-dimensional,
always sparse,
distributed
representation

Architecture for an RNN

Sequence of outputs



Some information is passed from one subunit to the next

Start of sequence marker

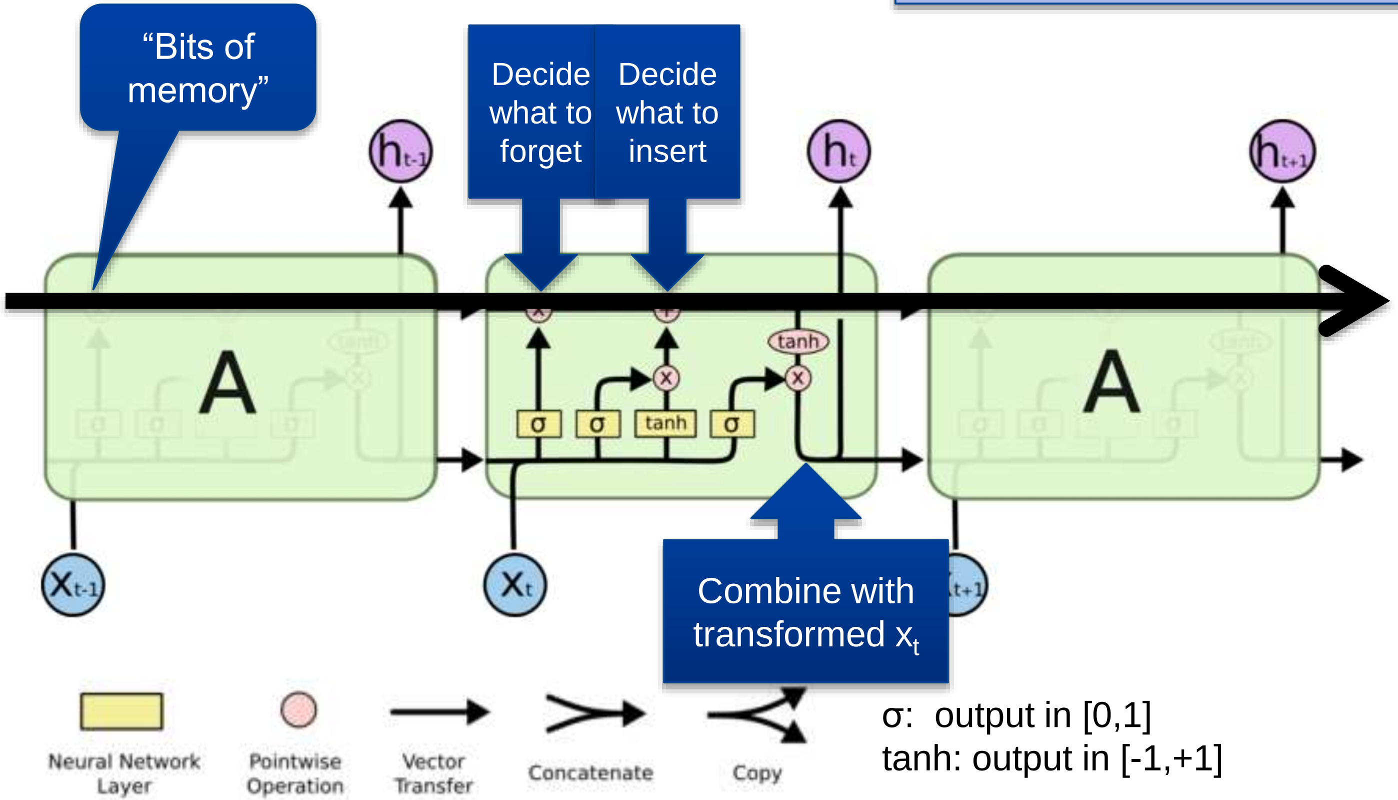
Sequence of inputs

End of sequence marker

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Architecture for an LSTM

Longterm-short term model



“Bits of memory”

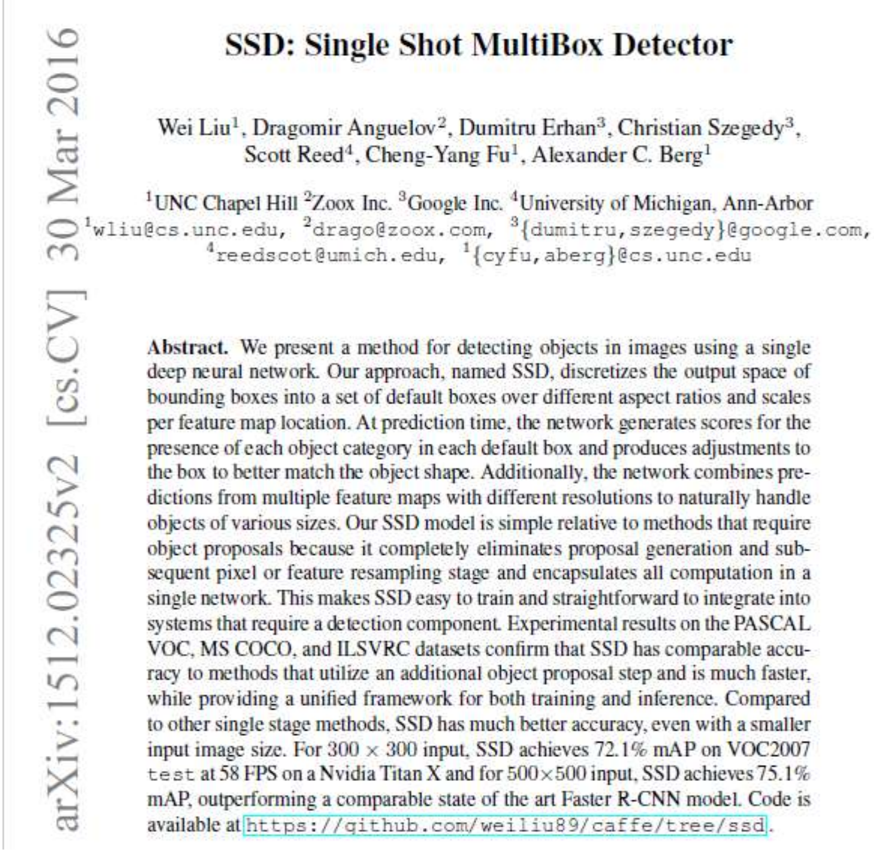
Decide what to forget

Decide what to insert

Combine with transformed x_t

σ: output in [0,1]
tanh: output in [-1,+1]

http://colah.github.io/posts/2015-08-Understanding-LSTMs/



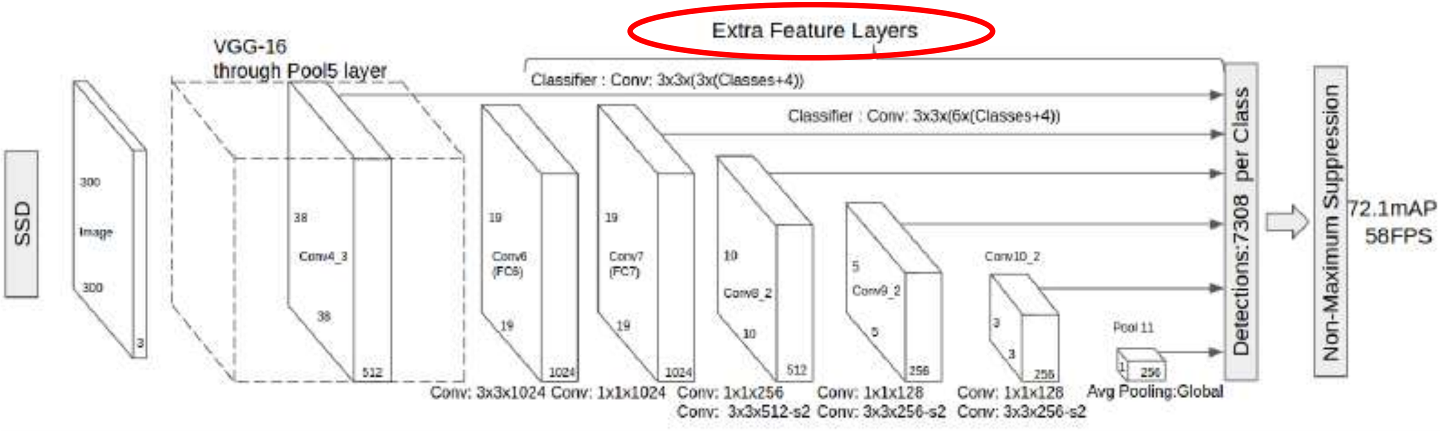
SSD: Single Shot MultiBox Detector

Wei Liu[1], Dragomir Anguelov[2], Dumitru Erhan[3], Christian Szegedy[3],
Scott Reed[4], Cheng-Yang Fu[1], Alexander C. Berg[1]

[1]UNC Chapel Hill [2]Zoox Inc. [3]Google Inc. [4]University of Michigan, Ann-Arbor
[1]wliu@cs.unc.edu, [2]drago@zoox.com, [3]{dumitru,szegedy}@google.com,
[4]reedscot@umich.edu, [1]{cyfu,aberg}@cs.unc.edu

Abstract. We present a method for detecting objects in images using a single deep neural network. Our approach, named SSD, discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. Additionally, the network combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes. Our SSD model is simple relative to methods that require object proposals because it completely eliminates proposal generation and subsequent pixel or feature resampling stage and encapsulates all computation in a single network. This makes SSD easy to train and straightforward to integrate into systems that require a detection component. Experimental results on the PASCAL VOC, MS COCO, and ILSVRC datasets confirm that SSD has comparable accuracy to methods that utilize an additional object proposal step and is much faster, while providing a unified framework for both training and inference. Compared to other single stage methods, SSD has much better accuracy, even with a smaller input image size. For 300×300 input, SSD achieves 72.1% mAP on VOC2007 `test` at 58 FPS on a Nvidia Titan X and for 500×500 input, SSD achieves 75.1% mAP, outperforming a comparable state of the art Faster R-CNN model. Code is available at https://github.com/weiliu89/caffe/tree/ssd.

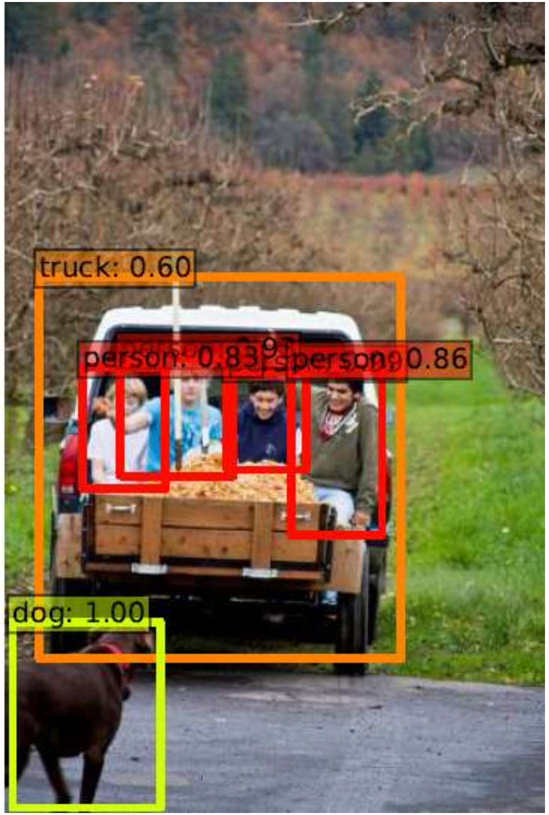
How SSD gets its goal?

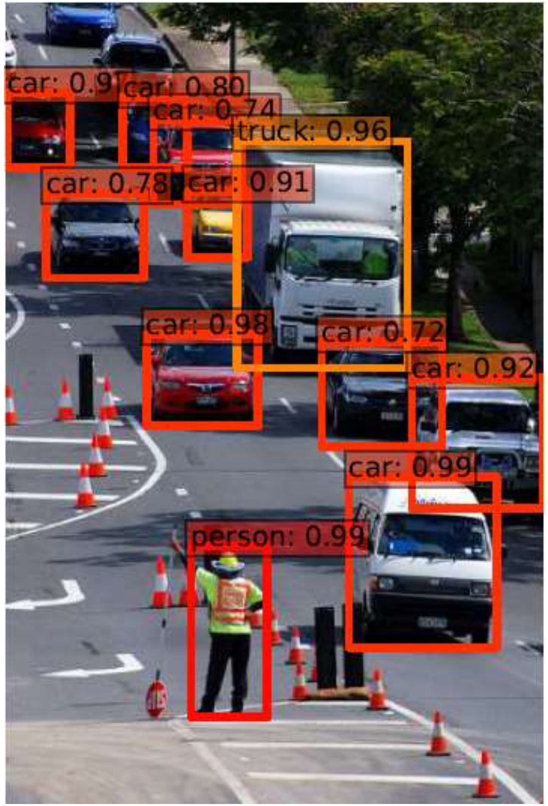


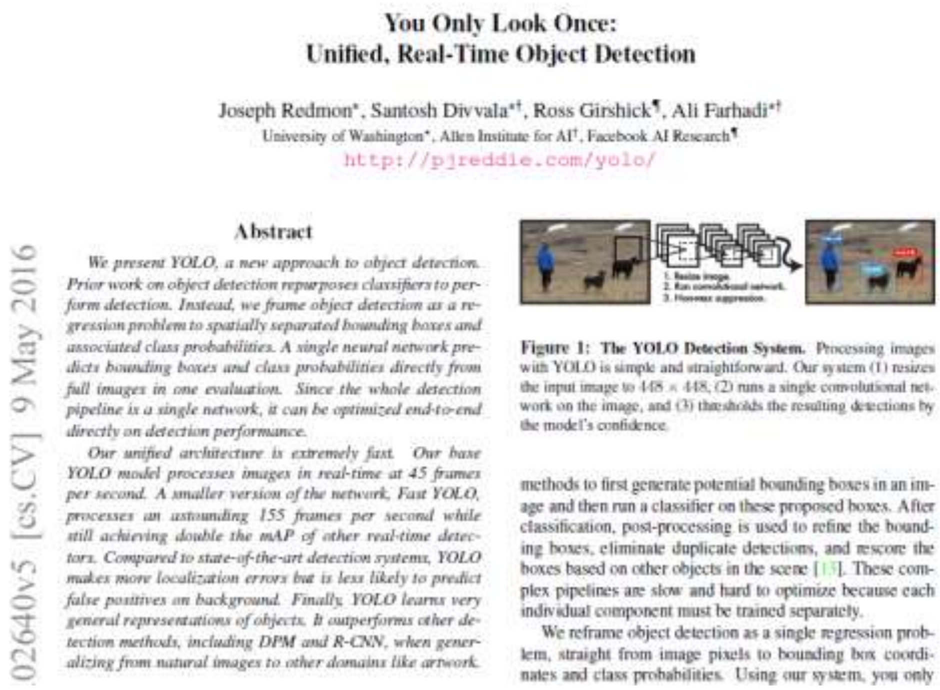
For speed: structure advantage!

- ➢**Eliminating bounding box proposals and subsequent pixel or feature resampling stage**

- ✓**Adding convolution feature layers to the end of the network to predict detections at multiple scales**


truck: 0.60
person: 0.83
0.93
person: 0.86
dog: 1.00


car: 0.97
car: 0.80
car: 0.74
truck: 0.96
car: 0.78
car: 0.91
car: 0.98
car: 0.72
car: 0.92
car: 0.99
person: 0.99



You Only Look Once: Unified, Real-Time Object Detection

Joseph Redmon[*], Santosh Divvala[*†], Ross Girshick[¶], Ali Farhadi[*†]
University of Washington[*], Allen Institute for AI[†], Facebook AI Research[¶]
http://pjreddie.com/yolo/

Abstract

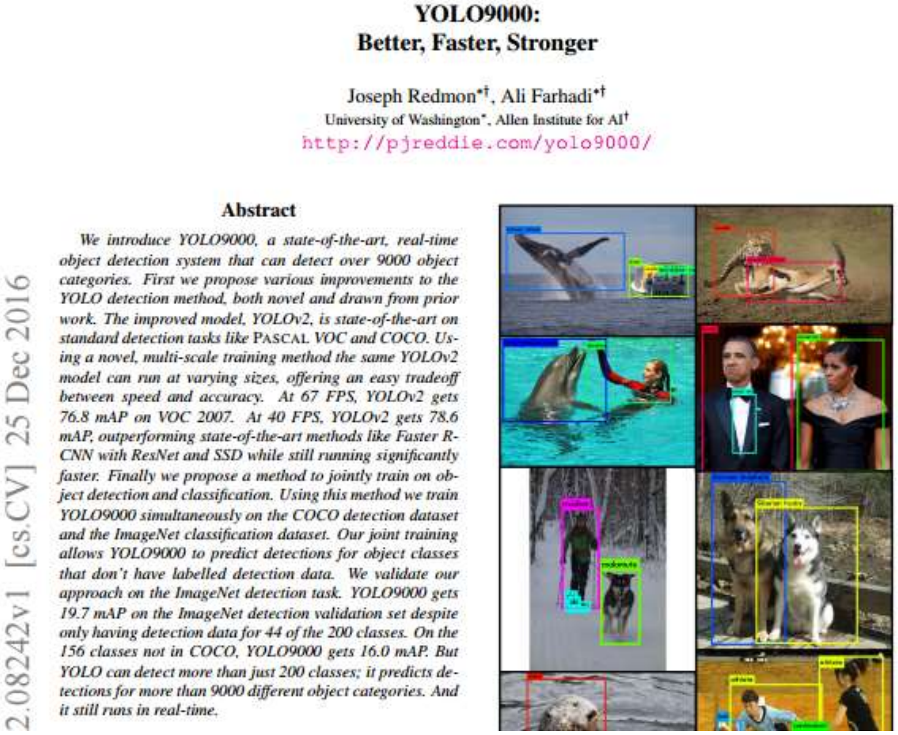
We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is less likely to predict false positives on background. Finally, YOLO learns very general representations of objects. It outperforms other detection methods, including DPM and R-CNN, when generalizing from natural images to other domains like artwork.

Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding boxes, eliminate duplicate detections, and rescore the boxes based on other objects in the scene [13]. These complex pipelines are slow and hard to optimize because each individual component must be trained separately.

We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using our system, you only



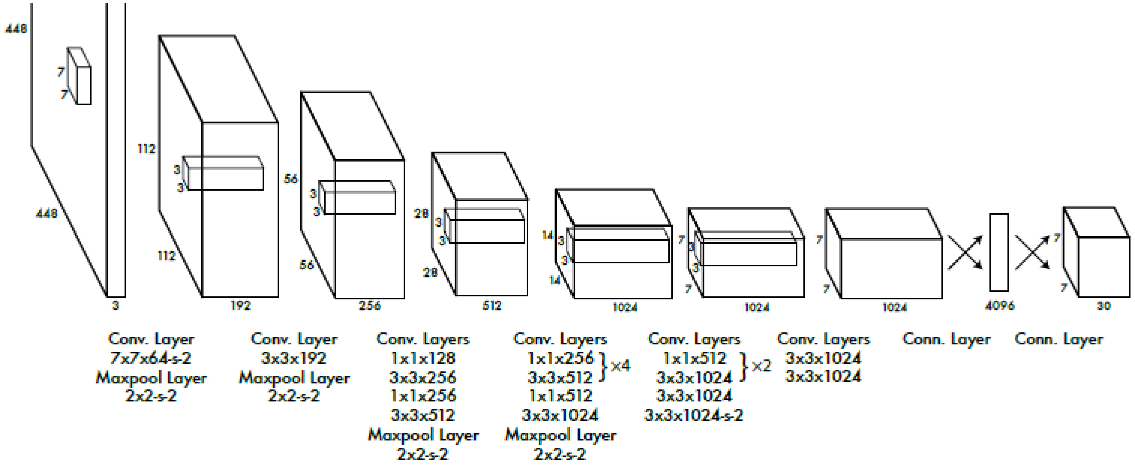
YOLO9000: Better, Faster, Stronger

Joseph Redmon[*†], Ali Farhadi[*†]
University of Washington[*], Allen Institute for AI[†]
http://pjreddie.com/yolo9000/

Abstract

We introduce YOLO9000, a state-of-the-art, real-time object detection system that can detect over 9000 object categories. First we propose various improvements to the YOLO detection method, both novel and drawn from prior work. The improved model, YOLOv2, is state-of-the-art on standard detection tasks like PASCAL *VOC and COCO. Using a novel, multi-scale training method the same YOLOv2 model can run at varying sizes, offering an easy tradeoff between speed and accuracy. At 67 FPS, YOLOv2 gets 76.8 mAP on VOC 2007. At 40 FPS, YOLOv2 gets 78.6 mAP, outperforming state-of-the-art methods like Faster R-CNN with ResNet and SSD while still running significantly faster. Finally we propose a method to jointly train on object detection and classification. Using this method we train YOLO9000 simultaneously on the COCO detection dataset and the ImageNet classification dataset. Our joint training allows YOLO9000 to predict detections for object classes that don't have labelled detection data. We validate our approach on the ImageNet detection task. YOLO9000 gets 19.7 mAP on the ImageNet detection validation set despite only having detection data for 44 of the 200 classes. On the 156 classes not in COCO, YOLO9000 gets 16.0 mAP. But YOLO can detect more than just 200 classes; it predicts detections for more than 9000 different object categories. And it still runs in real-time.*

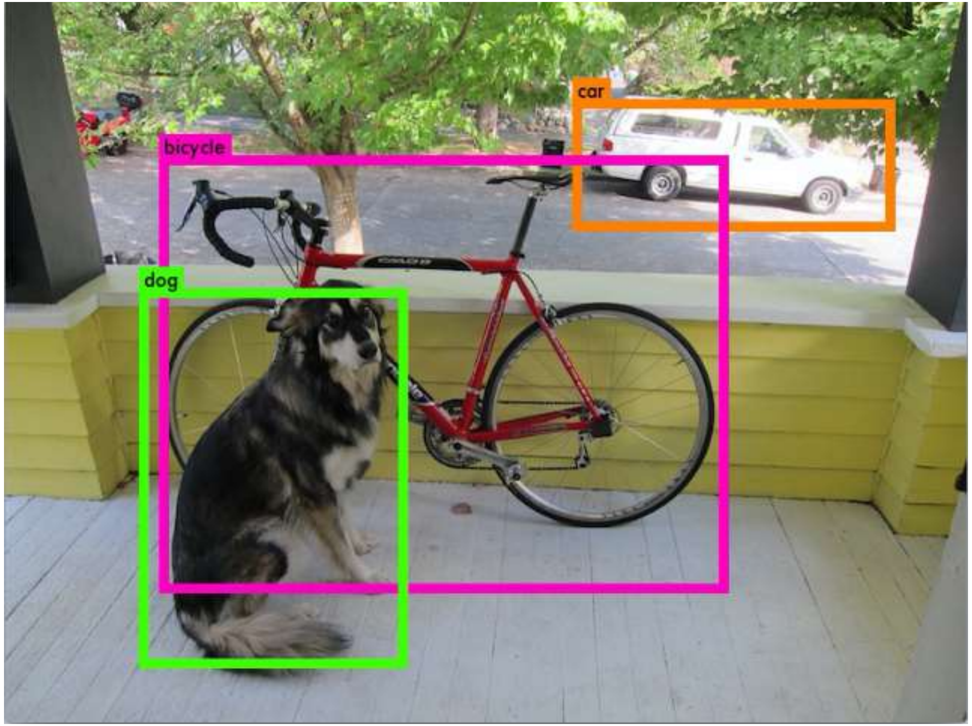
How YOLO gets its goal?



For speed: Also! structure advantage!

➢**No bounding box proposals and subsequent pixel or feature resampling stage**

✓**A neural network predicts bounding boxes and class probabilities directly from full images in one evaluation**



car
bicycle
dog

YOLO v2

http://pjreddie.com/yolo

TIPS

Tips

- **Gather Data, more Data, use all you can**
  - If enough => deeper Networks
  - If not => Data Augmentation, Drop Out
- **Add small L2 regularization, it never hurts**
- **Use Batch Normalization**
- **Initialization with whitening**
- **Use the newest optimizer**
- **Define a cost function that solves your final problem**
- **Look at performances on different validations datasets**
- **Look at weights evolution**
- **Play with hyper-parameters**

IDENTITY & SECURITY

THANK YOU

SAFRAN