Introduction to Deep Learning

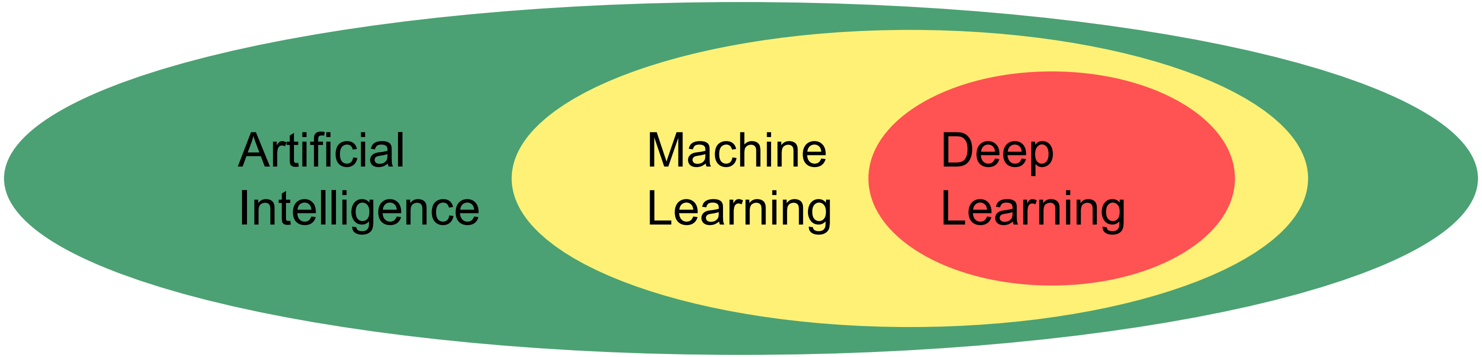
CS468 Spring 2017

Charles Qi

What is Deep Learning?

Deep learning allows computational models that are composed of **multiple processing layers** **to learn representations of data** with **multiple levels of abstraction.**

Deep Learning by Y. LeCun et al. Nature 2015



The traditional model of pattern recognition (since the late 50's)

- Fixed/engineered features (or fixed kernel) + trainable classifier

hand-crafted Feature Extractor → "Simple" Trainable Classifier →

From Y. LeCun's Slides

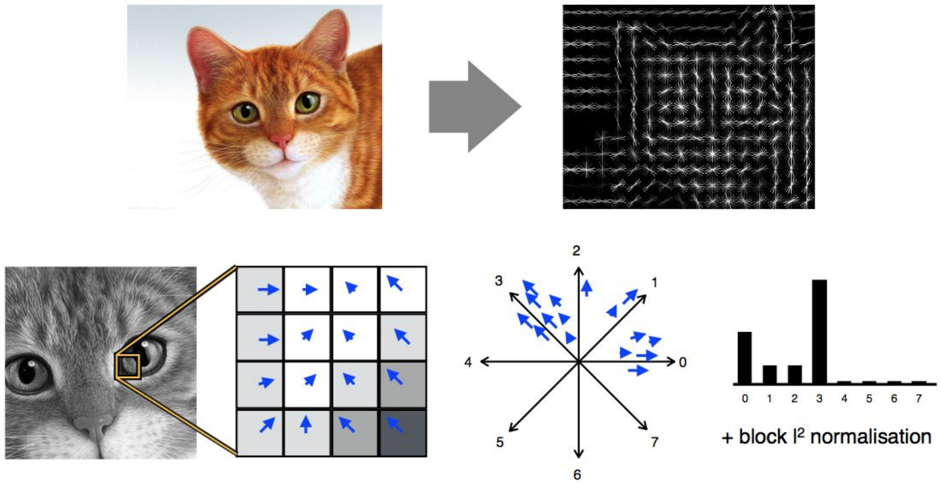
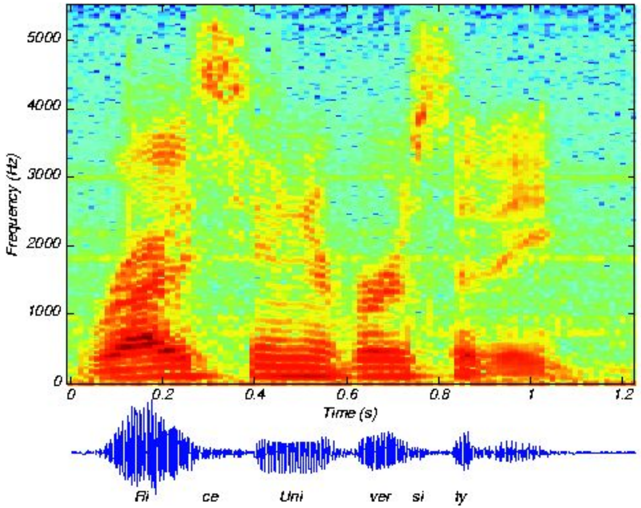


Image: HoG



Audio: Spectrogram

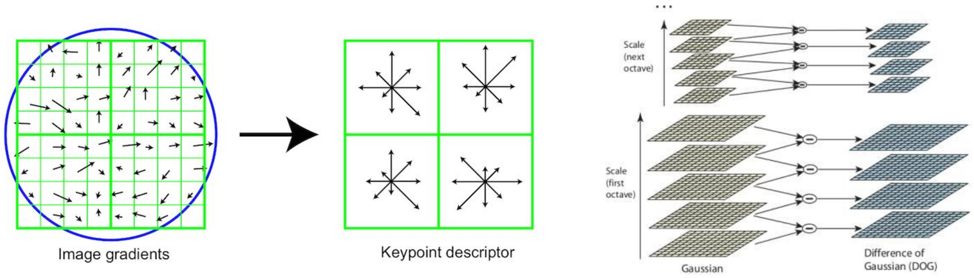
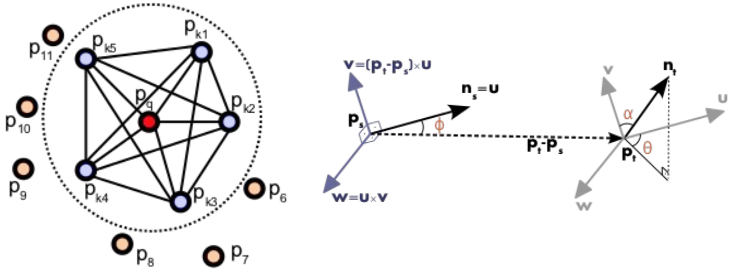
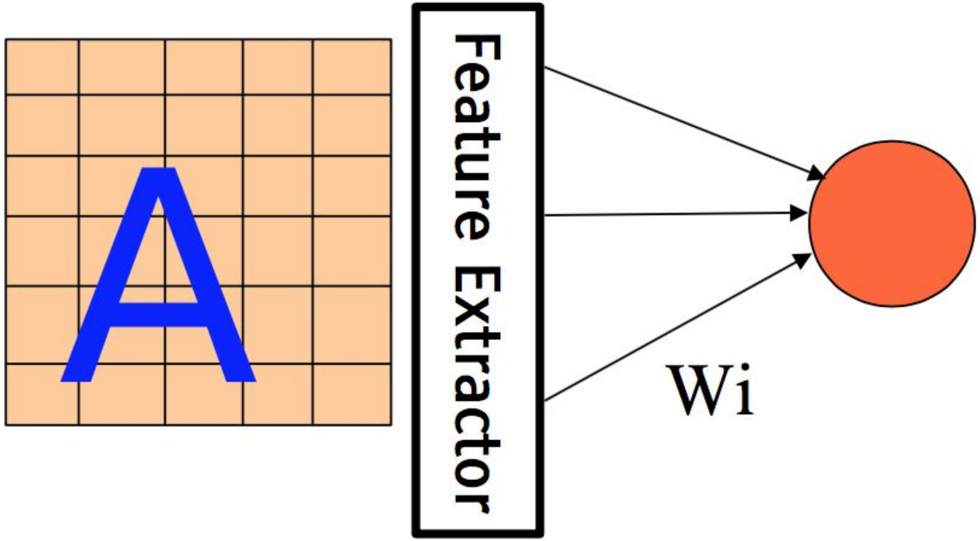


Image: SIFT



Point Cloud: PFH



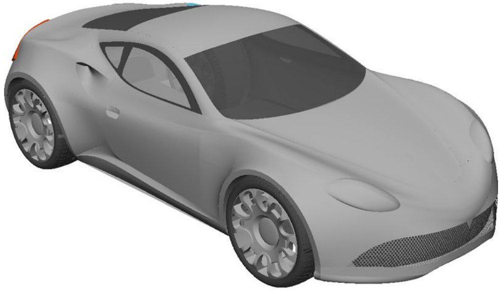
Linear Regression
SVM
Decision Trees
Random Forest
...

$$y = sign\left(\sum_{i=1}^{N} W_i F_i(X) + b\right)$$

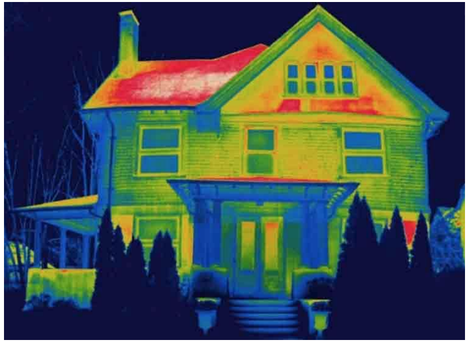
From Y. LeCun's Slides

Image

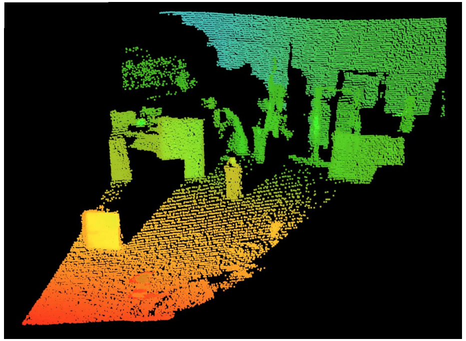
Video



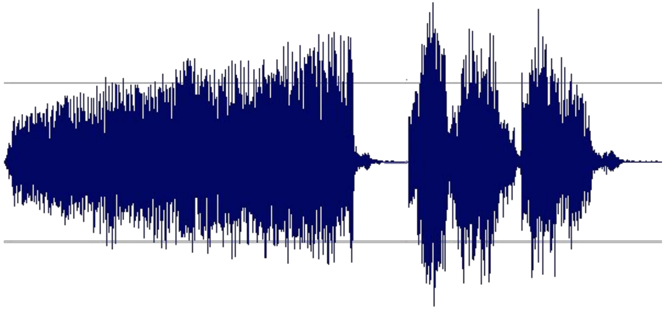
3D CAD Model



Thermal Infrared



Depth Scan

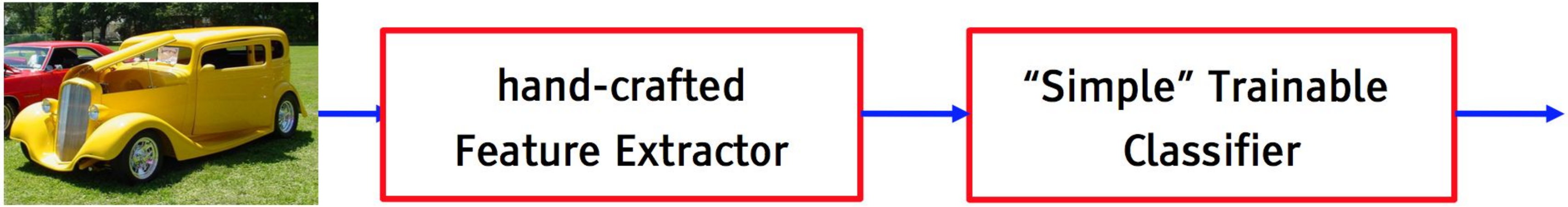


Audio

Can we automatically learn “good” feature representations?

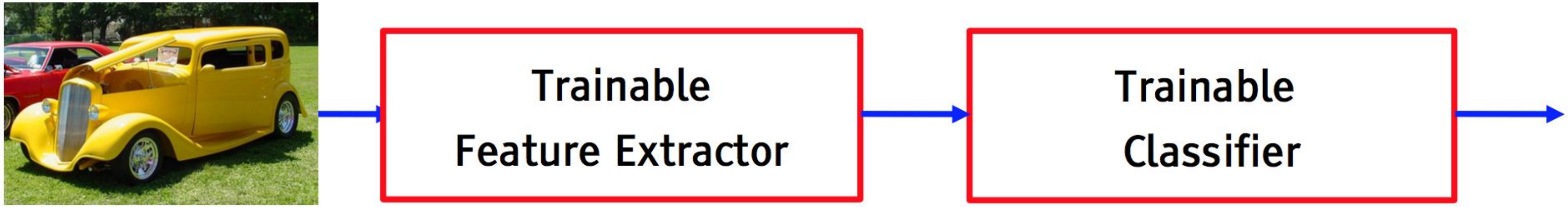
The traditional model of pattern recognition (since the late 50's)

- Fixed/engineered features (or fixed kernel) + trainable classifier



End-to-end learning / Feature learning / Deep learning

- Trainable features (or kernel) + trainable classifier

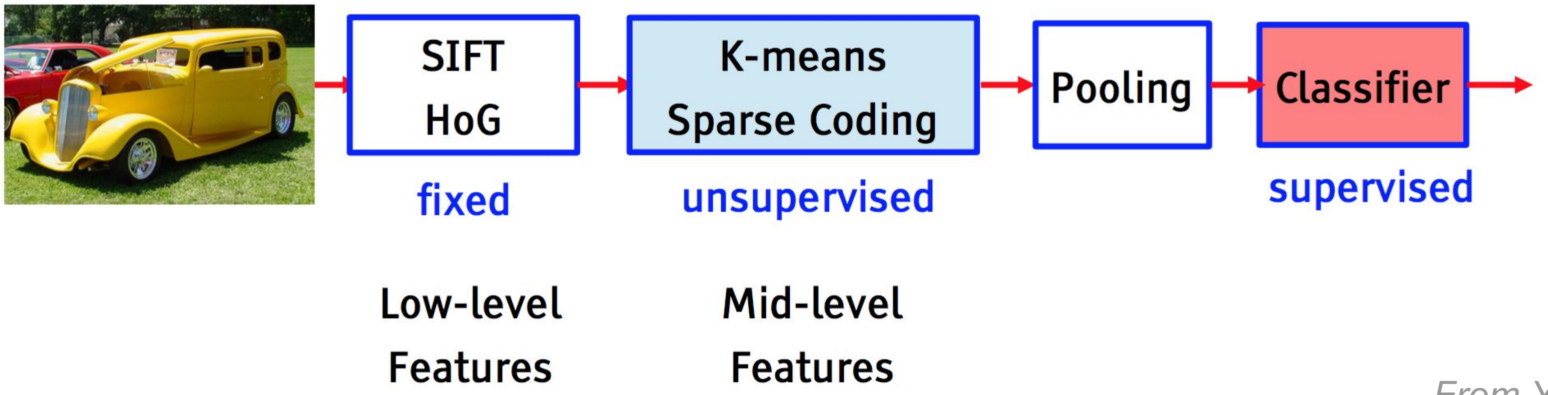


From Y. LeCun's Slides

Modern architecture for pattern recognition

- Speech recognition: early 90's – 2011

- Object Recognition: 2006 - 2012



From Y. LeCun's Slides

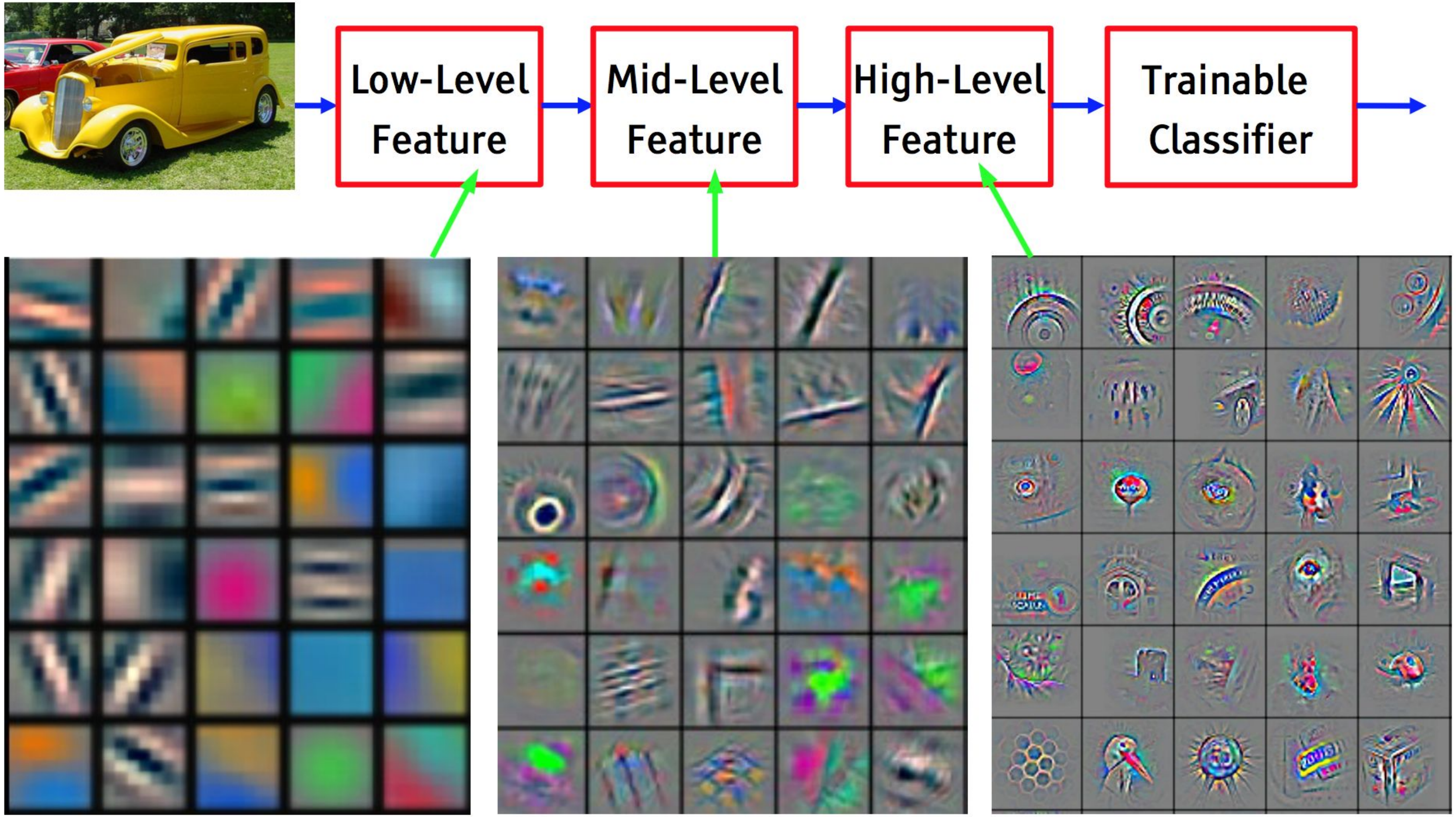
Traditional Pattern Recognition: Fixed/Handcrafted Feature Extractor

Mainstream Modern Pattern Recognition: Unsupervised mid-level features

Deep Learning: Representations are hierarchical and trained

From Y. LeCun's Slides

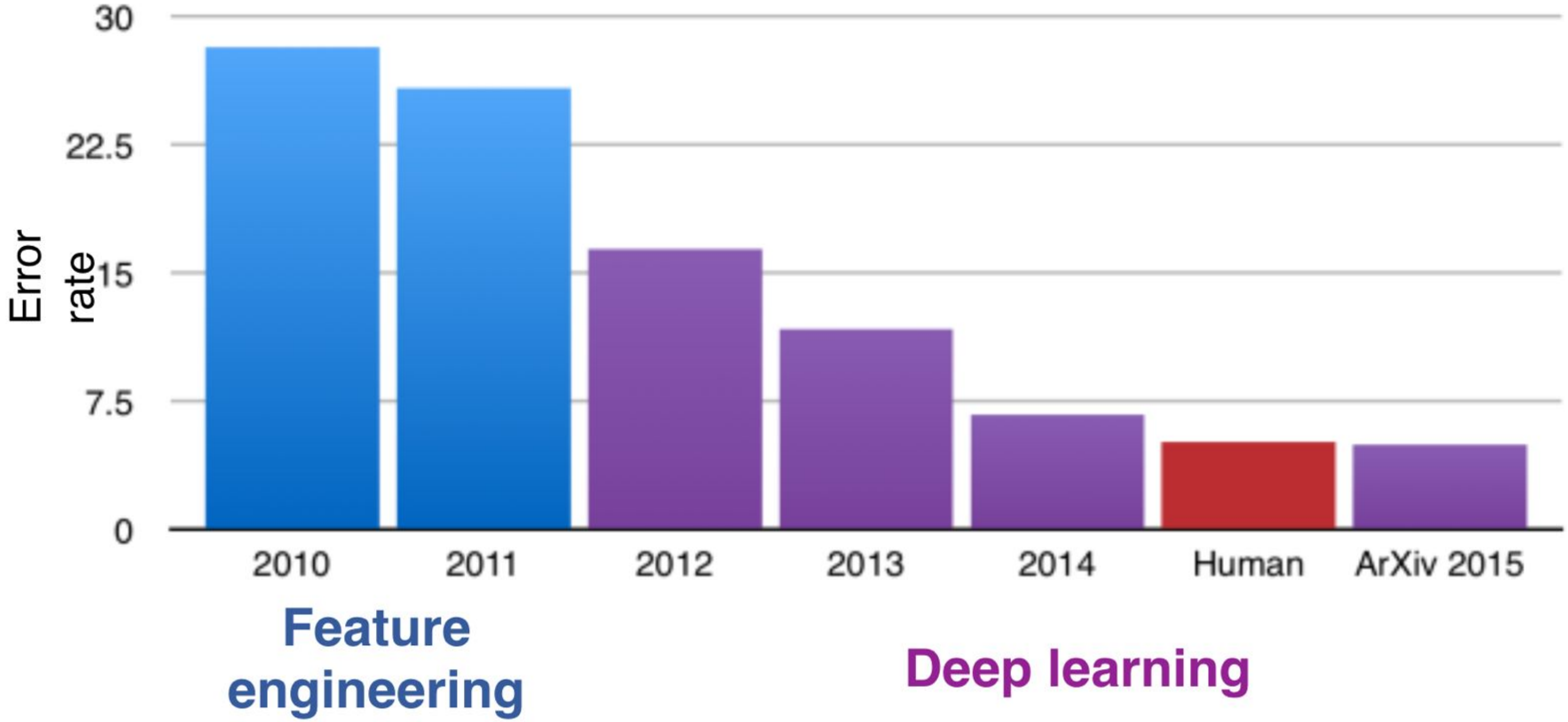
It's deep if it has more than one stage of non-linear feature transformation



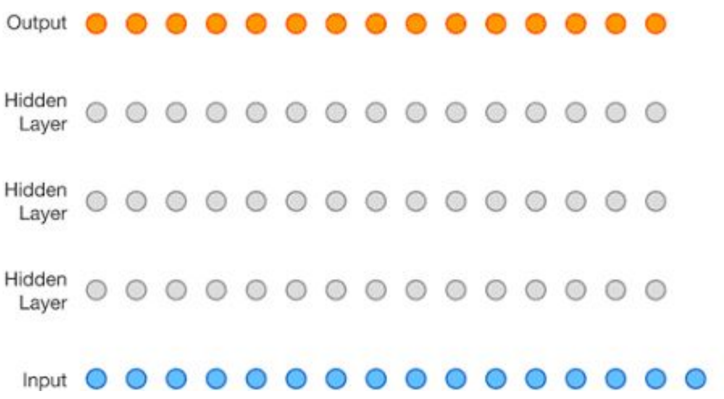
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

From Y. LeCun's Slides

ImageNet 1000 class image classification accuracy



Big Data + Representation Learning with Deep Nets



Acoustic Modeling

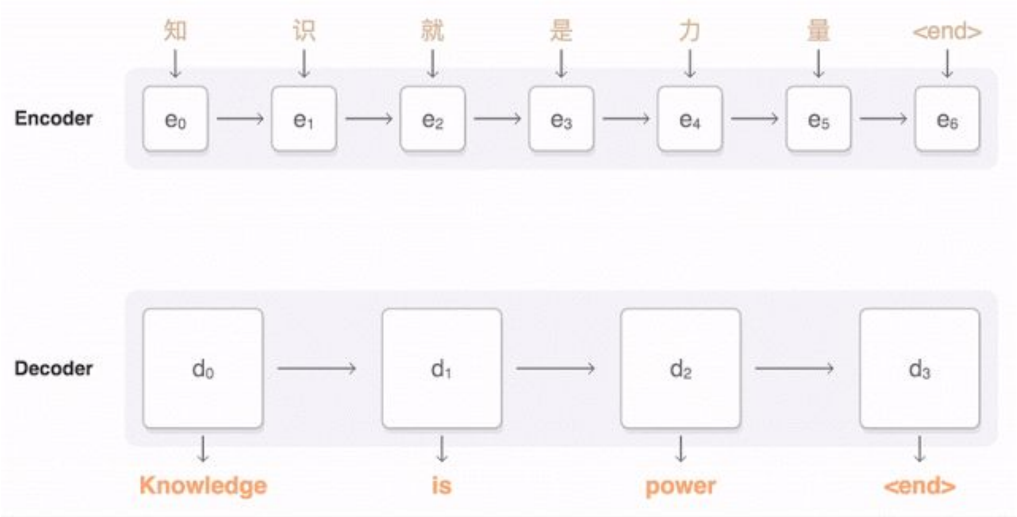
Near human-level
Text-To-Speech performance

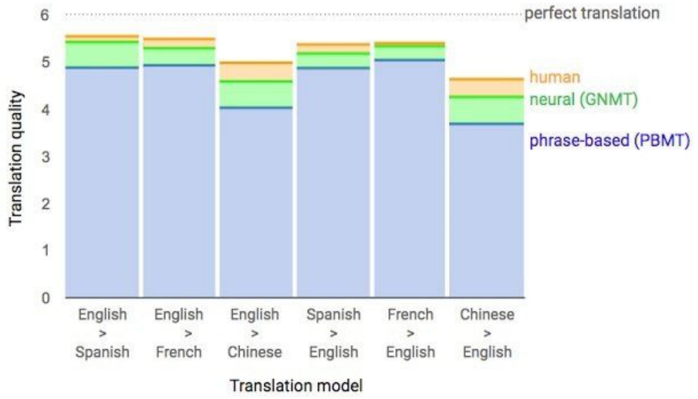
https://deepmind.com/blog/wavenet-generative-model-raw-audio/

Big Data + Representation Learning with Deep Nets

Neural Translation Machine

by Quac V. Le et al at Google Brain.





Data from side-by-side evaluations, where human raters compare the quality of translations for a given source sentence. Scores range from 0 to 6, with 0 meaning "completely nonsense translation", and 6 meaning "perfect translation."

https://research.googleblog.com/2016/09/a-neural-network-for-machine.html

Outline

- Motivation
- A Simple Neural Network
- Ideas in Deep Net Architectures
- Ideas in Deep Net Optimization
- Practicals and Resources

Outline

- Motivation
- **A Simple Neural Network**
- Ideas in Deep Net Architectures
- Ideas in Deep Net Optimization
- Practicals and Resources

A Simple Neural Network

Use recent three days' average temperature to predict tomorrow's average temperature.

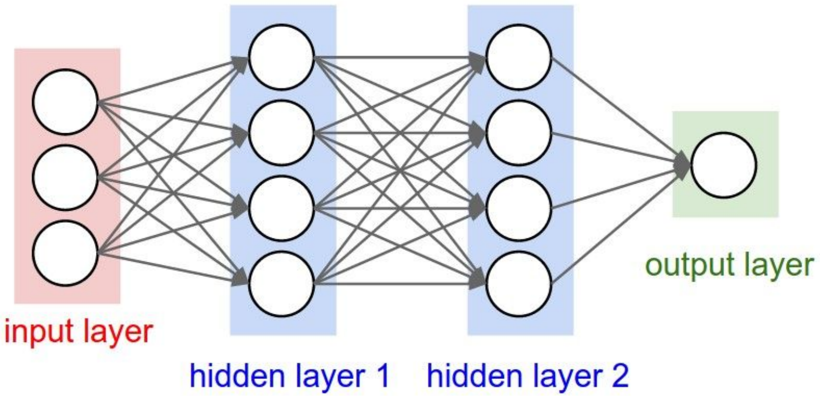
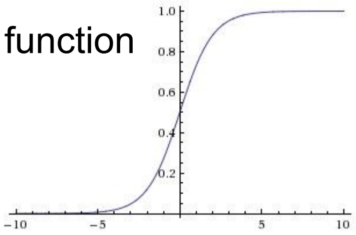
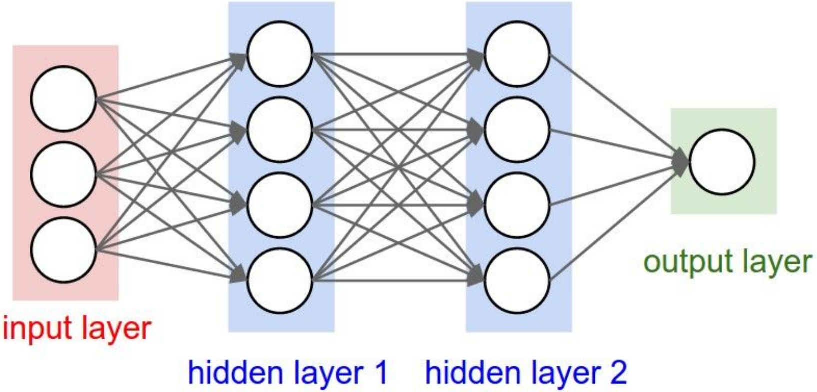


Image from CS231N

A Simple Neural Network

Sigmoid function



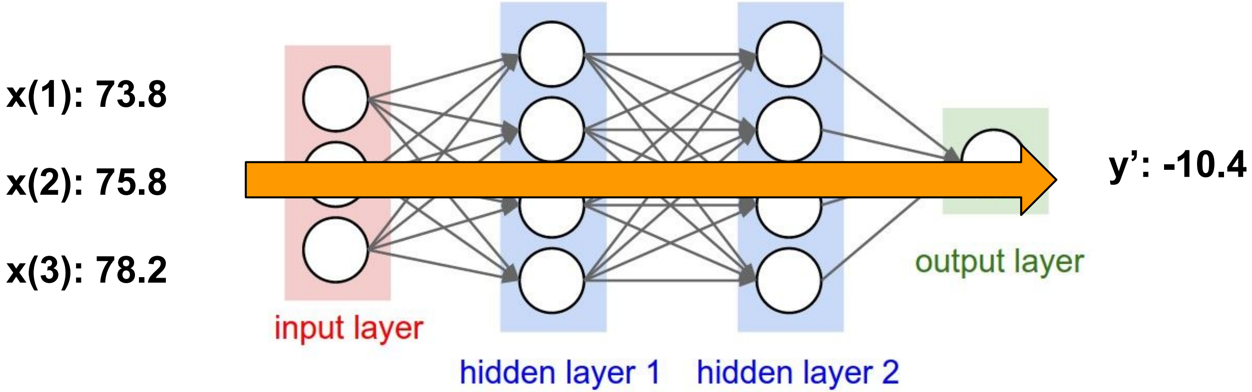


W1, b1, W2, b2, W3, b3 are network parameters that need to be learned.

```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

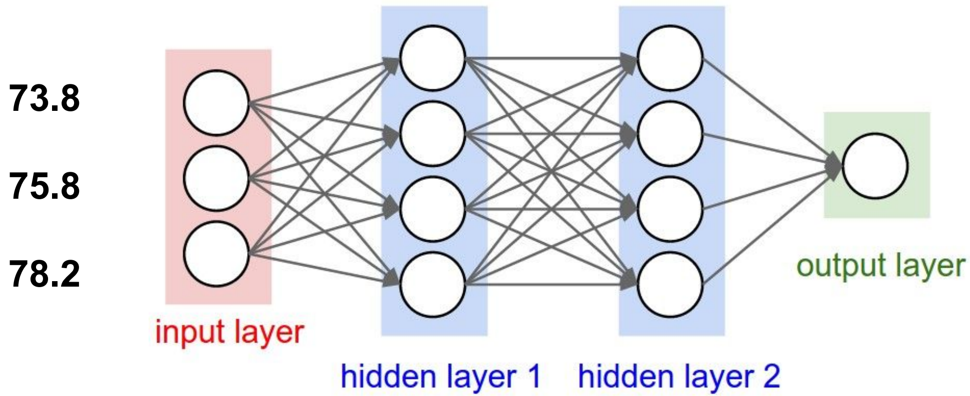
From CS231N

Neural Network: Forward Pass



$$y' = W_3 f(W_2 f(W_1 x + b_1) + b_2) + b_3)$$

Neural Network: Backward Pass



Update Network Parameters

Minimize: $L(x, y; W, b) = \sum_{i=1}^{N} (W_3 f(W_2 f(W_1 x_i + b_1) + b_2) + b_3) - y_i)^2$

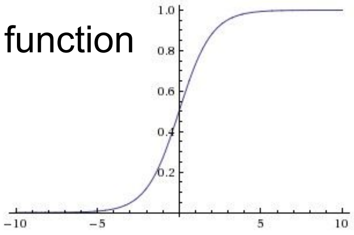
Given N training pairs: $\{x_i, y_i\}_{i=1}^{N}$

Neural Network: Backward Pass

Minimize: $L(x, y; W, b) = \sum_{i=1}^{N}(W_3 f(W_2 f(W_1 x_i + b_1) + b_2) + b_3) - y_i)^2$

Given N training pairs: $\{x_i, y_i\}_{i=1}^{N}$

Sigmoid function



Non-convex optimization :(

Neural Network: Backward Pass

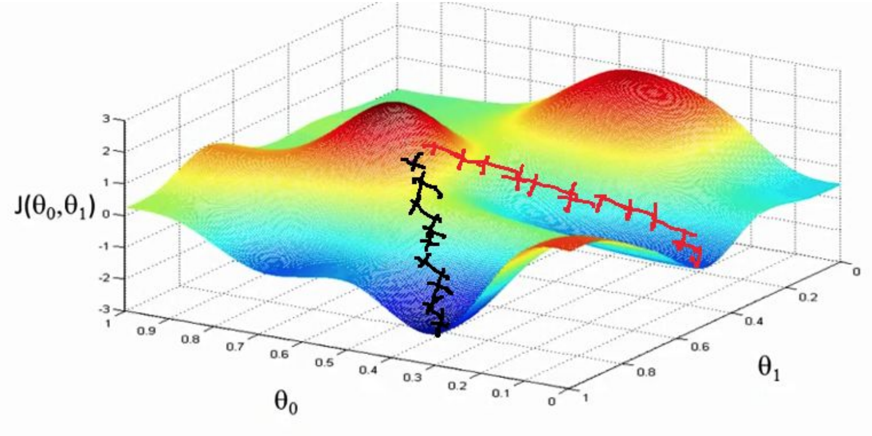
Minimize: $L(x, y; W, b) = \sum_{i=1}^{N}(W_3 f(W_2 f(W_1 x_i + b_1) + b_2) + b_3) - y_i)^2$

Given N training pairs: $\{x_i, y_i\}_{i=1}^{N}$

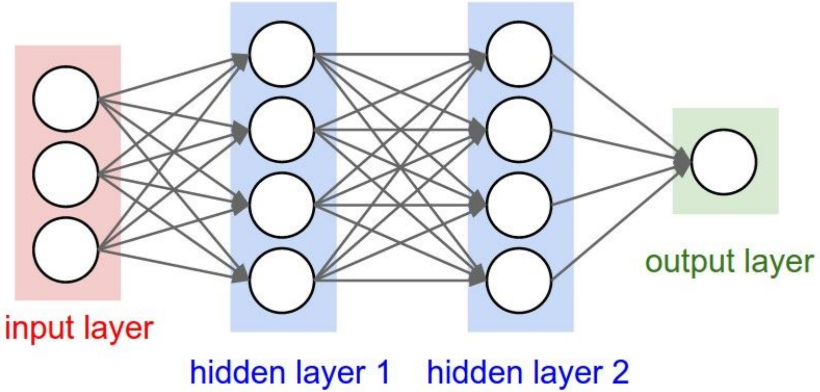
Non-convex optimization :(
Use gradient descent!

Parameter update example:

$$W_3 = W_3 - \eta \frac{\partial L}{\partial W_3}$$



A Simple Neural Network



Model: Multi-Layer Perceptron (MLP) $y' = W_3 f(W_2 f(W_1 x + b_1) + b_2) + b_3)$

Loss function: L2 loss $l(y, y') = (y - y')^2$

Optimization: Gradient descent $W = W - \eta \frac{\partial L}{\partial W}$

Outline

- Motivation
- A Simple Neural Network
- **Ideas in Deep Net Architectures**
- Ideas in Deep Net Optimization
- Practicals and Resources

What people think I am doing when I “build a deep learning model”

What I actually do...

Contents

Building blocks: fully connected, ReLU, conv, pooling, upconv, dilated conv

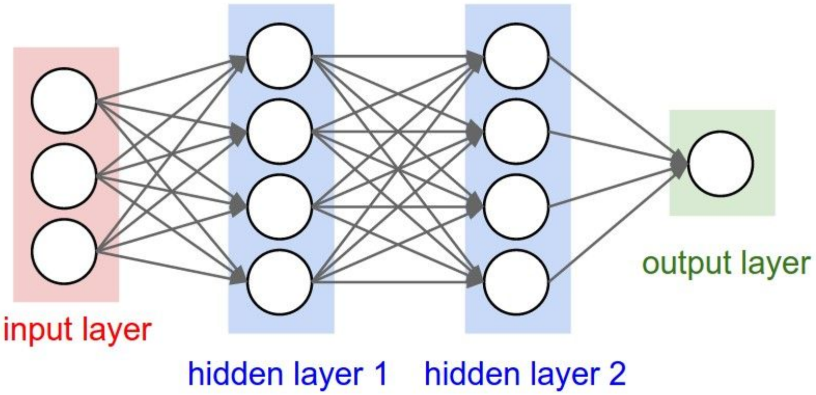
Classic architectures: MLP, LeNet, AlexNet, NIN, VGG, GoogleNet, ResNet, FCN

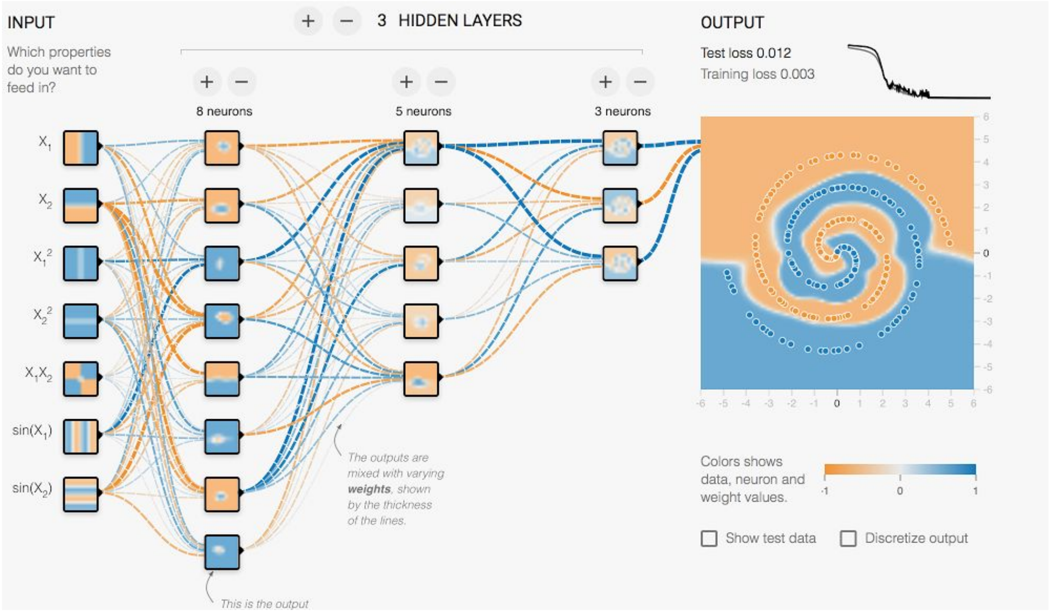
Multi-Layer Perceptron

http://playground.tensorflow.org/

Fully Connected

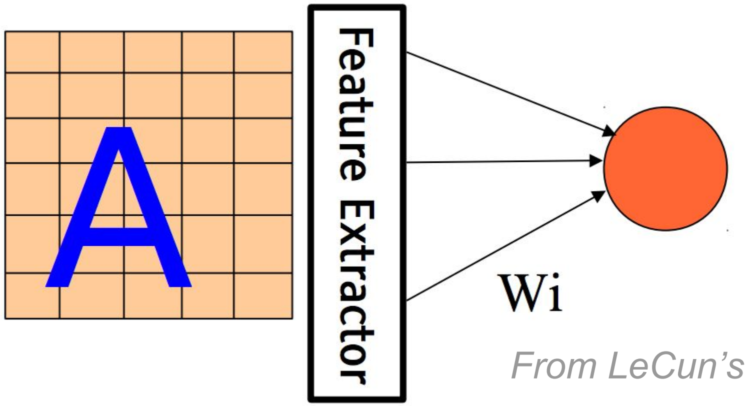
Non-linear Op



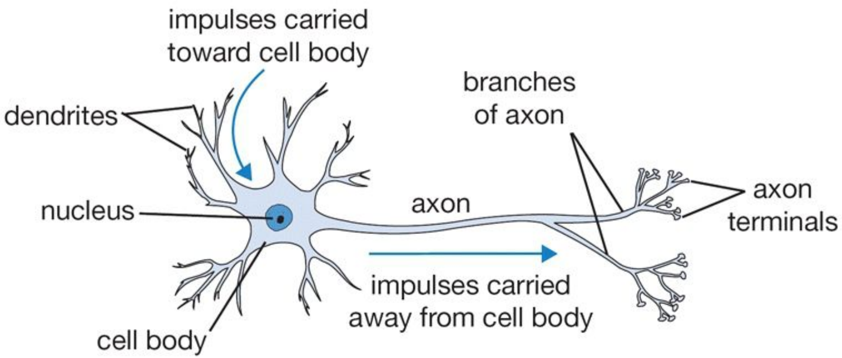


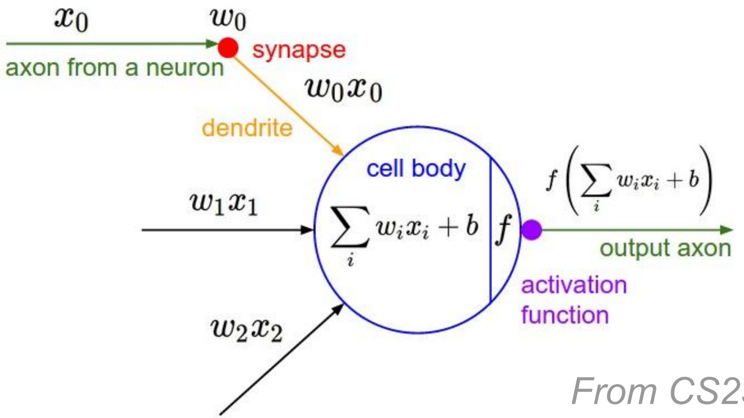
- The first learning machine: the **Perceptron** Built at Cornell in 1960
- The Perceptron was a (binary) linear classifier on top of a simple feature extractor

$$y = sign\left(\sum_{i=1}^{N} W_i F_i(X) + b\right)$$



From LeCun's Slides



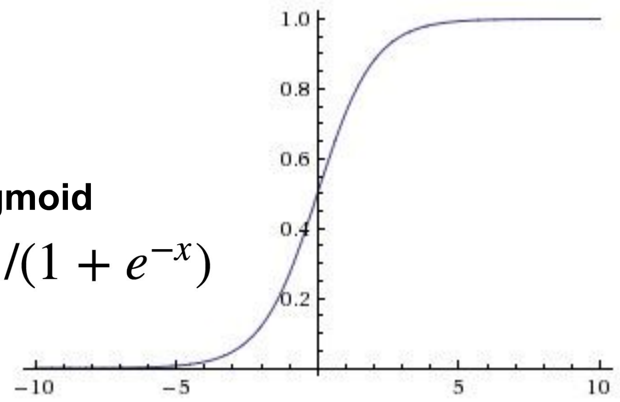


From CS231N

Non-linear Op

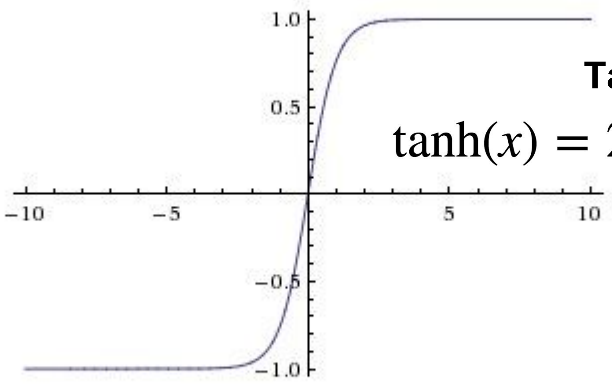
Sigmoid

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



Tanh

$$\tanh(x) = 2\sigma(2x) - 1$$

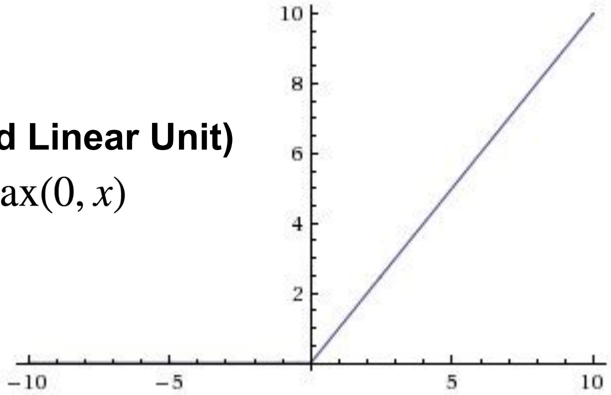


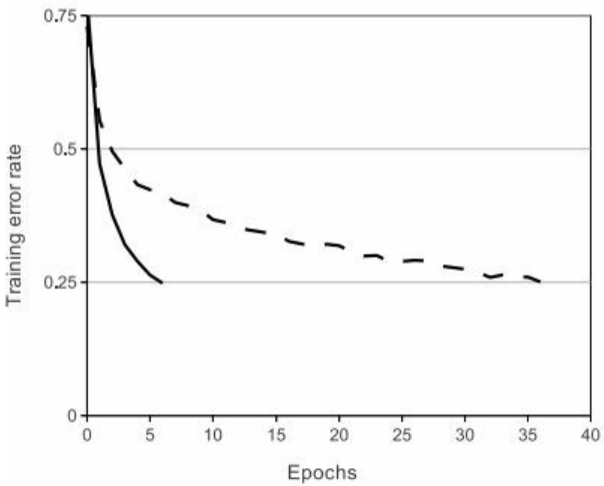
Major drawbacks: Sigmoids saturate and kill gradients

From CS231N

ReLU (Rectified Linear Unit)

$$f(x) = \max(0, x)$$





A plot from Krizhevsky et al. paper indicating **the 6x improvement in convergence** with the ReLU unit compared to the tanh unit.

+ Cheaper (linear) compared with Sigmoids (exp)
+ No gradient saturation, faster in convergence
- "Dead" neurons if learning rate set too high

Other Non-linear Op:

Leaky ReLU, $f(x) = \mathbb{1}(x < 0)(\alpha x) + \mathbb{1}(x >= 0)(x)$

MaxOut $\max(w_1^T x + b_1, w_2^T x + b_2)$

From CS231N

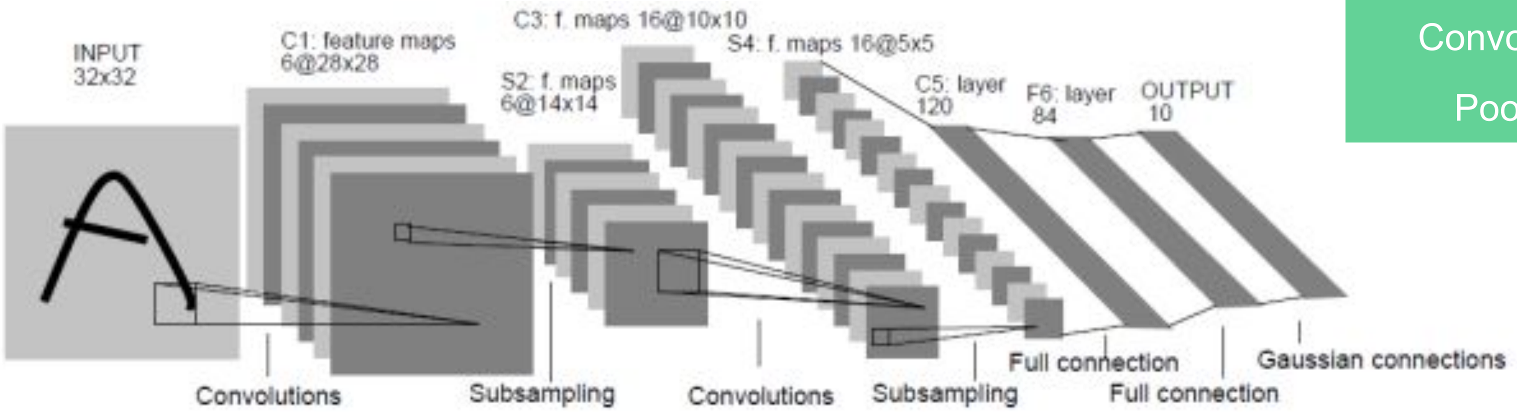
Convolutional Neural Network : LeNet (1998 by LeCun et al.)

Fully Connected

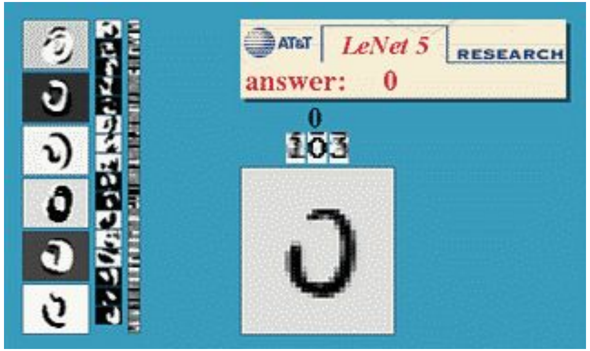
Non-linear Op

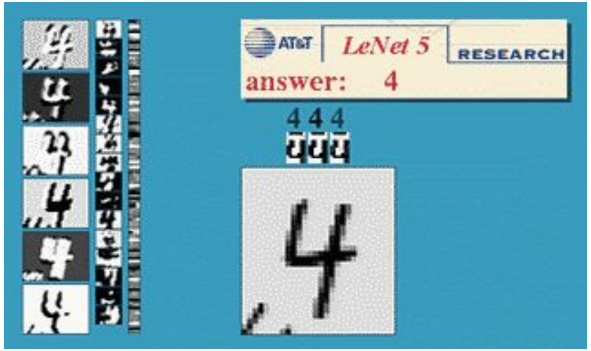
Convolution

Pooling



(pooling)

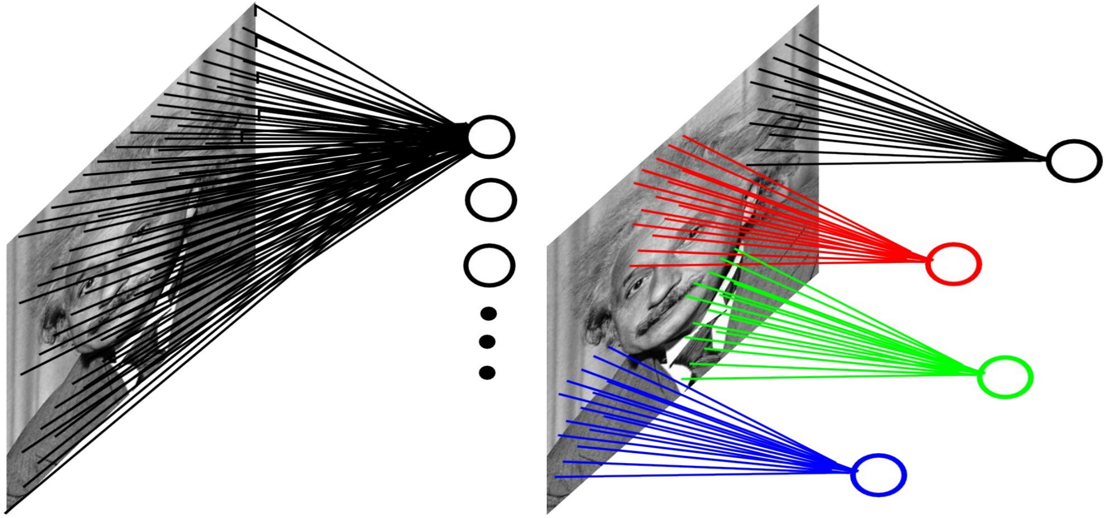




One of the first successful applications of CNN.

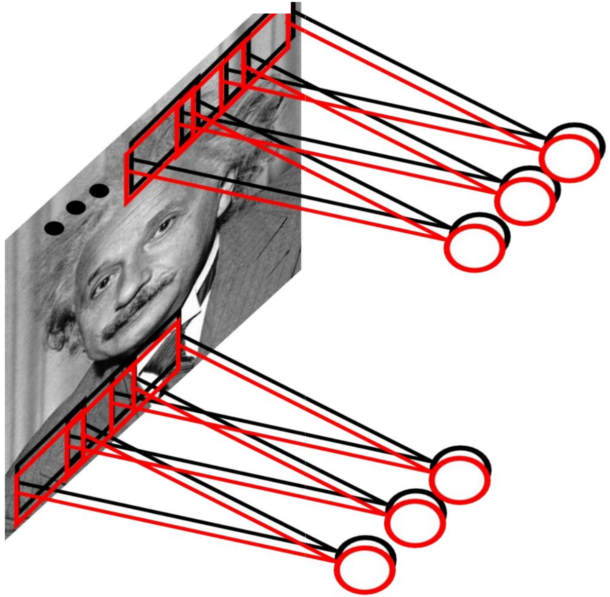
Fully Connected NN in high dimension

- Example: 200x200 image
  - Fully-connected, 400,000 hidden units = 16 billion parameters
  - Locally-connected, 400,000 hidden units 10x10 fields = 40 million params
  - Local connections capture local dependencies



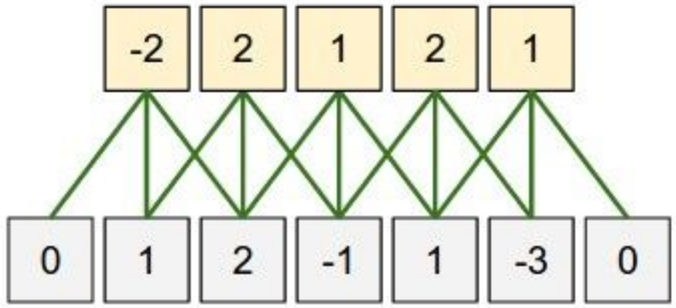
Shared Weights & Convolutions: Exploiting Stationarity

- Example: 200x200 image
  - 400,000 hidden units with 10x10 fields = 1000 params
  - 10 feature maps of size 200x200, 10 filters of size 10x10

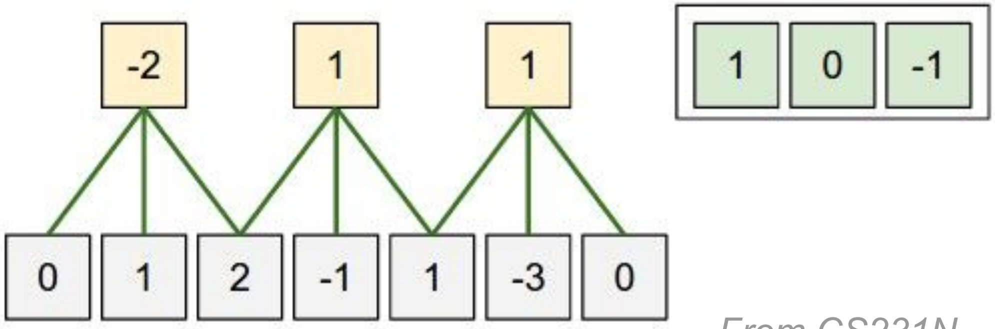


Convolution

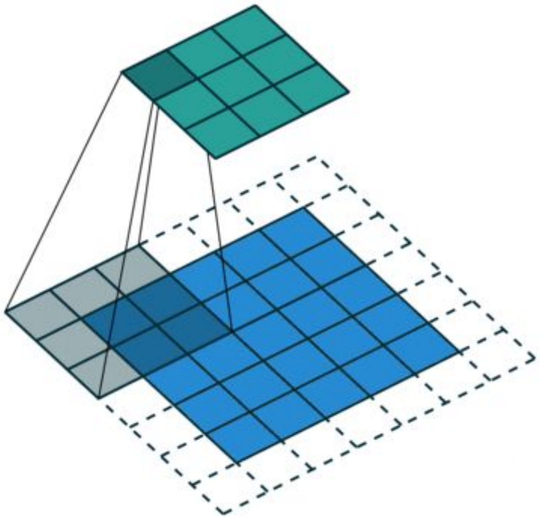
Stride 1



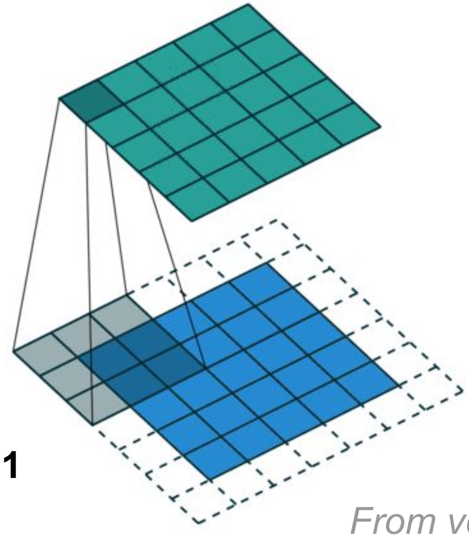
Stride 2



From CS231N



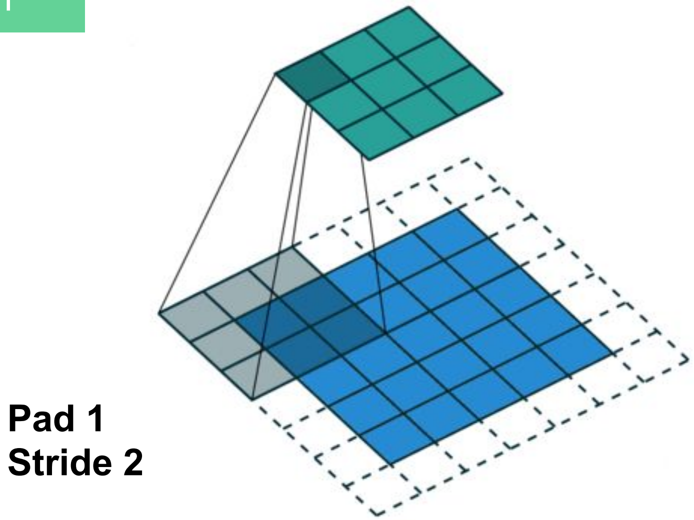
Pad 1
Stride 2

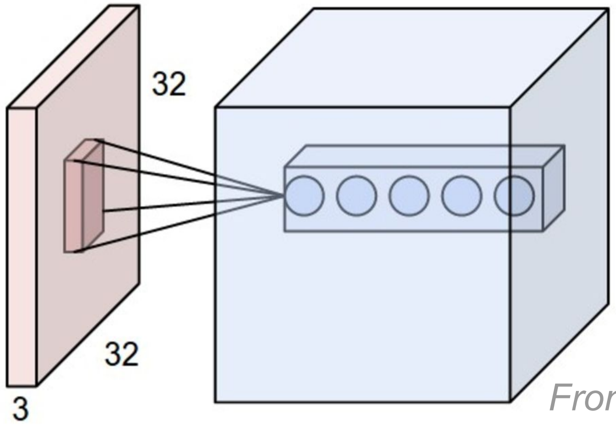


Pad 1
Stride 1

From vdumoulin/conv_arithmetic

Convolution





From CS231N

5x5 RGB Image
5x5x3 array

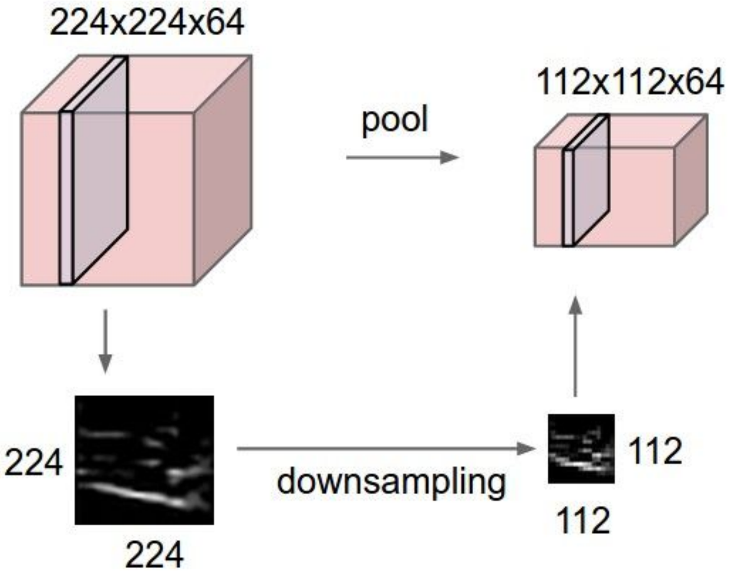
3x3 kernel, 2 output channels, pad 1, stride 2
weights: 2x3x3x3 array
bias: 2x1 array

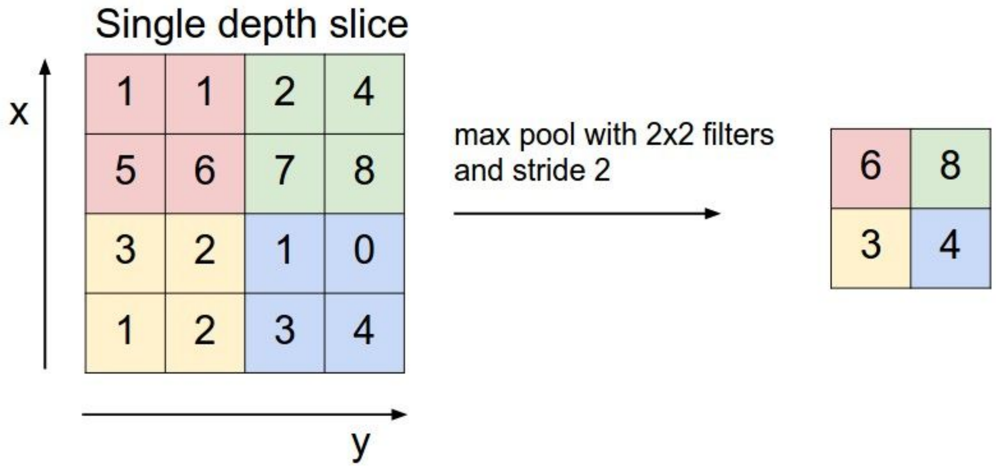
Output
3x3x2 array

H’ = (H - K)/stride_H + 1
= (7-3)/2 + 1 = 3

Pooling

Pooling layer (usually inserted in between conv layers) is used to reduce spatial size of the input, thus reduce number of parameters and overfitting.





Discarding pooling layers has been found to be important in training good generative models, such as variational autoencoders (VAEs) or generative adversarial networks (GANs).
It seems likely that future architectures will feature very few to no pooling layers.

From CS231N

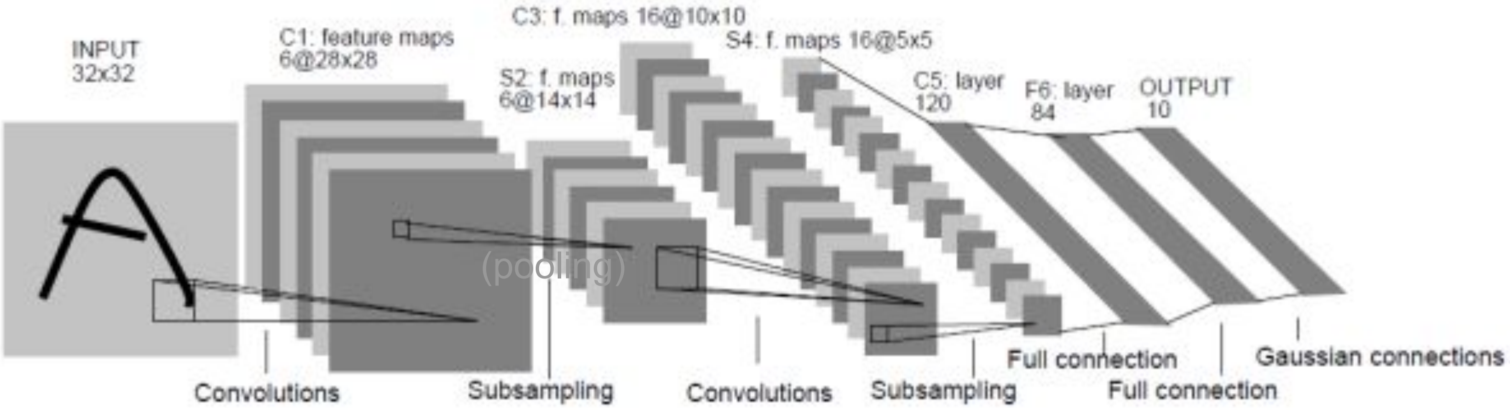
LeNet (1998 by LeCun et al.)

Fully Connected

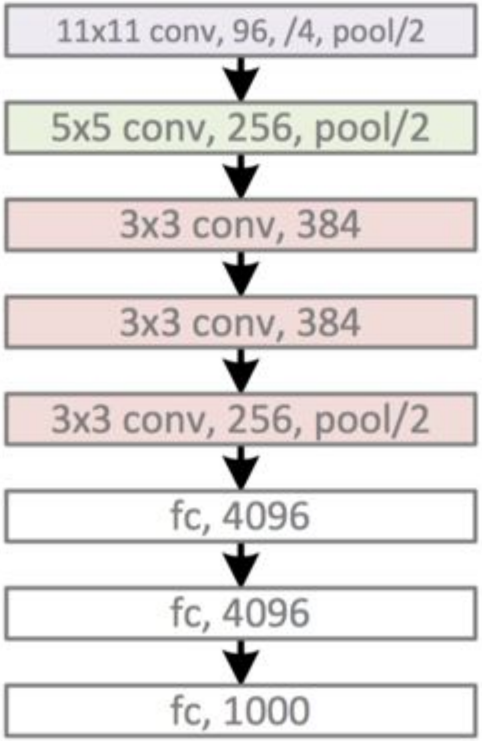
Non-linear Op

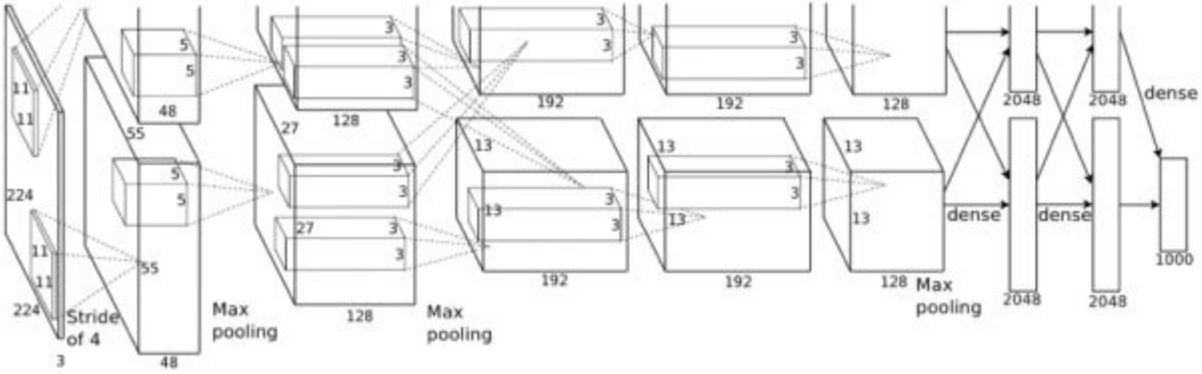
Convolution

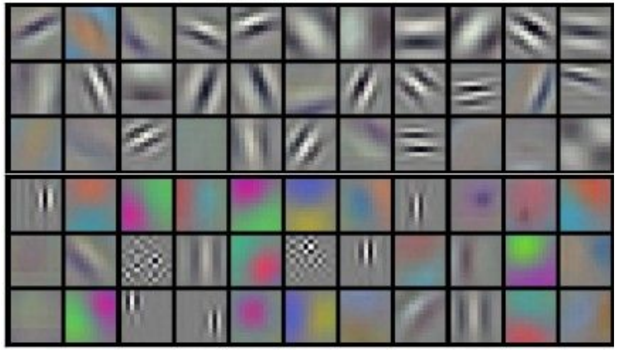
Pooling



AlexNet (2012 by Krizhevsky et al.)



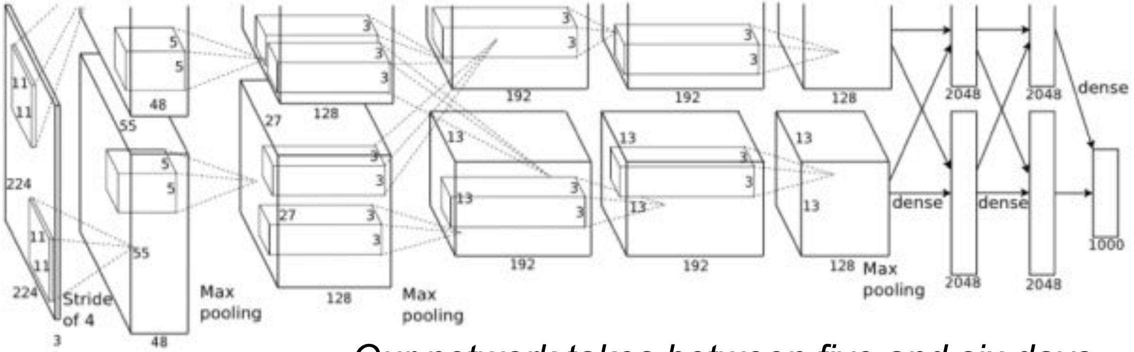




The first work that popularized Convolutional Networks in Computer Vision

What's different?

AlexNet (2012 by Krizhevsky et al.)

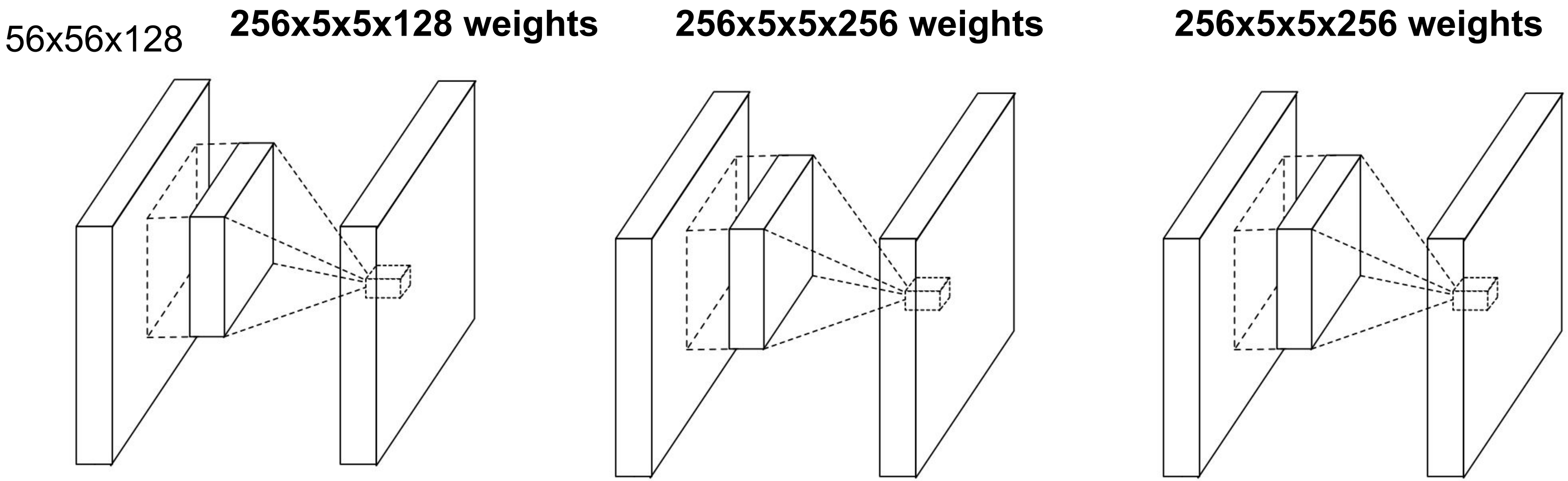


Our network takes between five and six days to train on two GTX 580 3GB GPUs. -- Alex

What's different?

- Big data: ImageNet
- GPU implementation: more than 10x speedup
- Algorithm improvement: deeper network, data augmentation, ReLU, dropout, normalization layers etc.

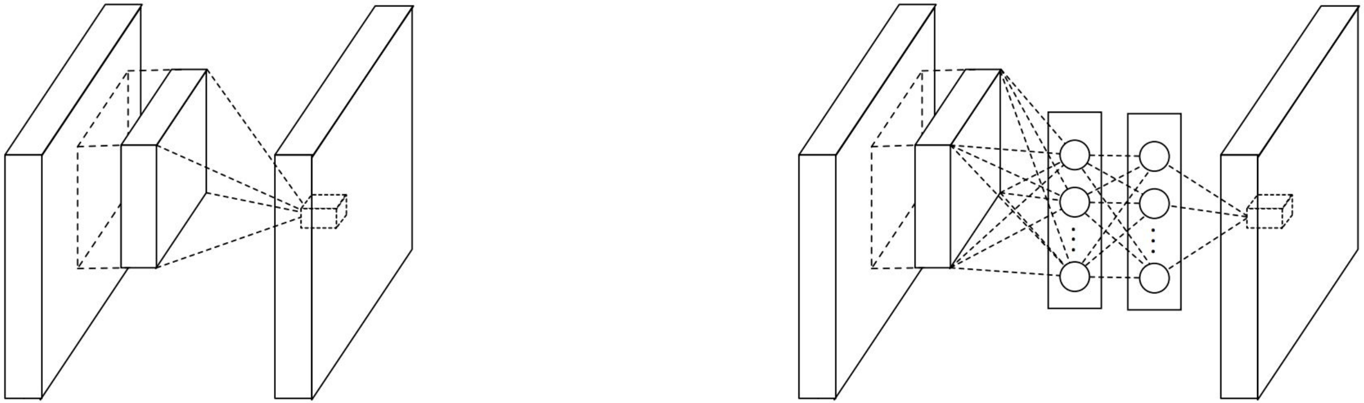
Network in Network (2013 by Min Lin et al.)



Network in Network (2013 by Min Lin et al.)

56x56x128 **256x5x5x128 weights**

256x5x5x128 weights
+ 1x1 conv (256x256 weights)
+ 1x1 conv (256x256 weights)



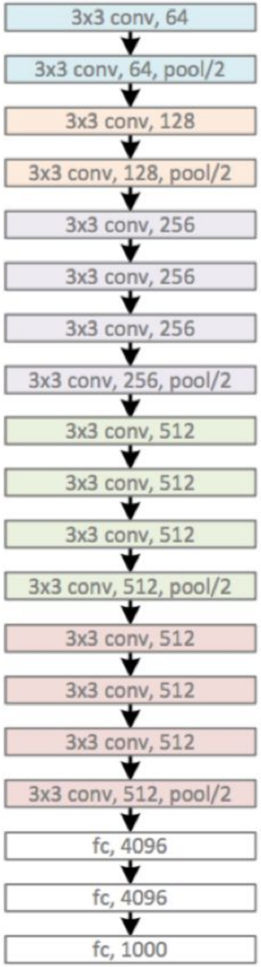
(a) Linear convolution layer

(b) Mlpconv layer

1x1 convolution: MLP in each pixel's channels
Use very little parameters for large model capacity.

VGG (2014 by Simonyan and Zisserman)

The runner-up in ILSVRC 2014



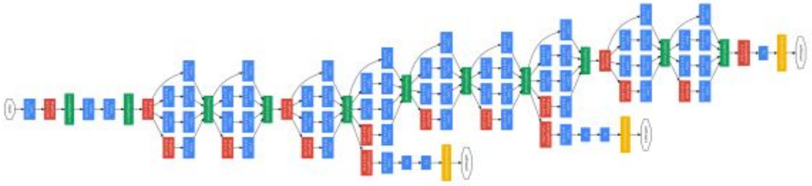
Karen Simonyan, Andrew Zisserman: **Very Deep** Convolutional Networks for Large-Scale Image Recognition.

- Its main contribution was in showing that the depth of the network is a critical component for good performance.
- Their final best network contains 16 CONV/FC layers and, appealingly, features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end.

-- quoted from CS231N

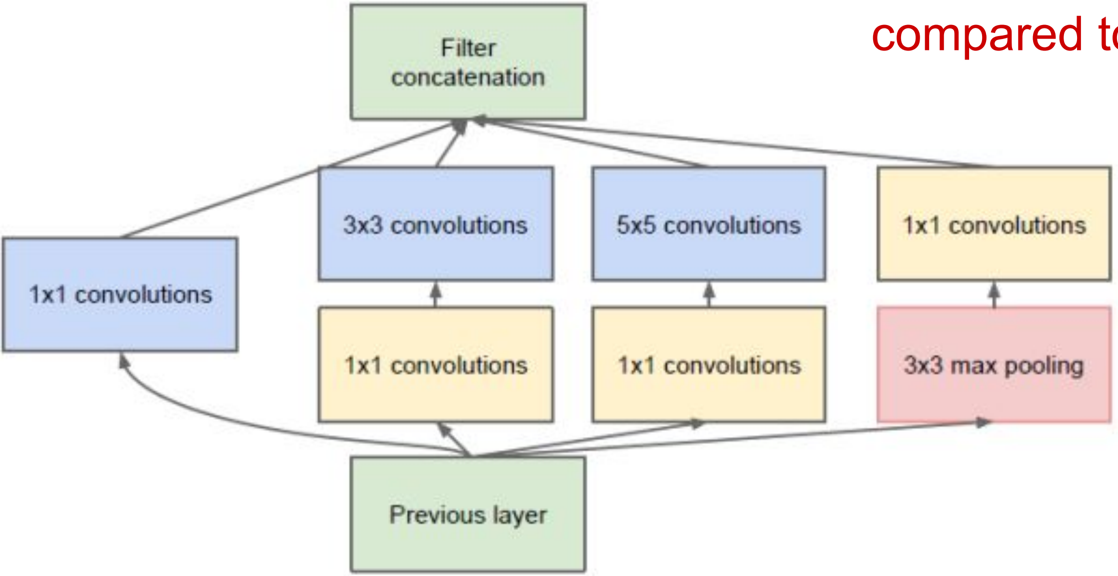
GoogleNet (2015 by Szegedy et al.)

The winner in ILSVRC 2014



Its main contribution was the development of an Inception Module and the using Average Pooling instead of Fully Connected layers at the top of the ConvNet, which dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M).

-- edited from CS231N

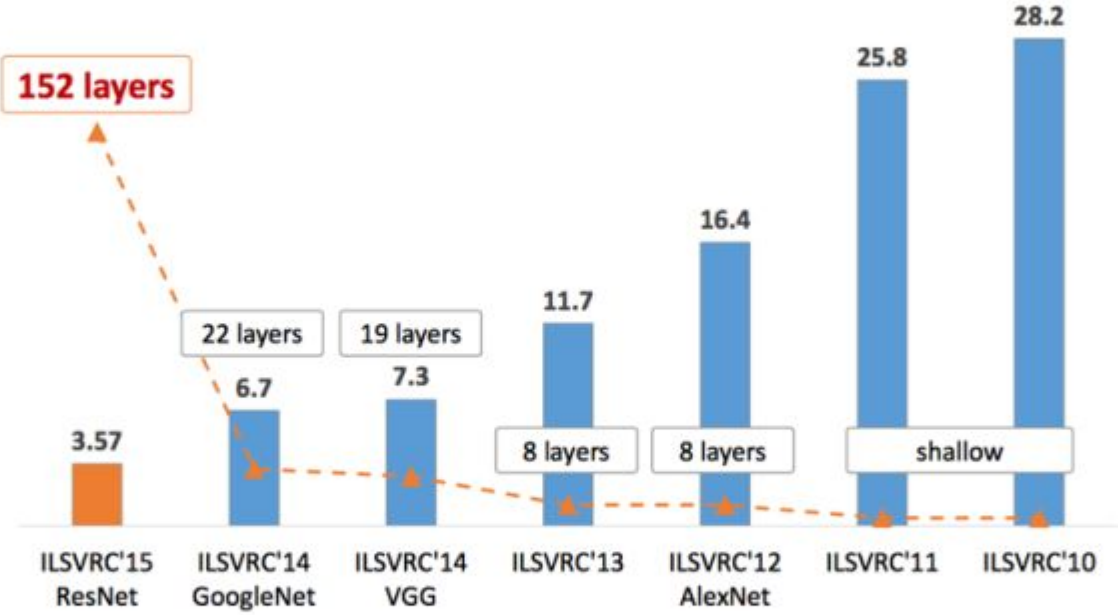


An Inception Module: a new building block..

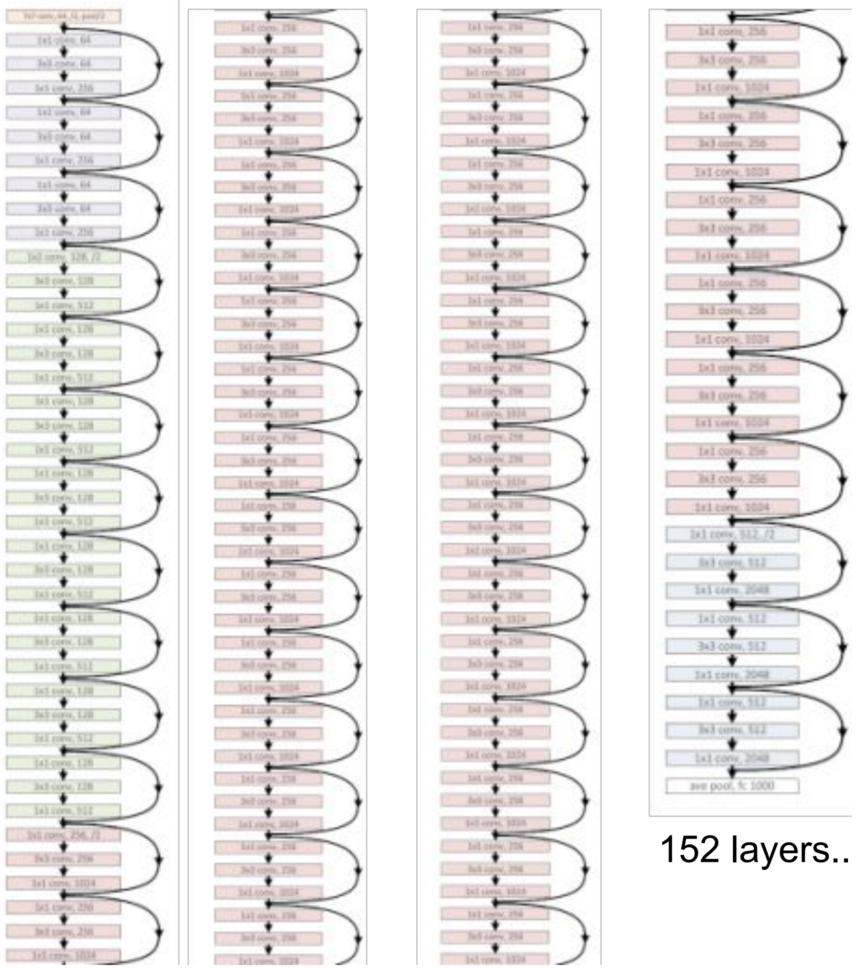
Tip on ConvNets:
Usually, most computation is spent on convolutions, while most space is spent on fully connected layers.

ResNet (2016 by Kaiming He et al.)

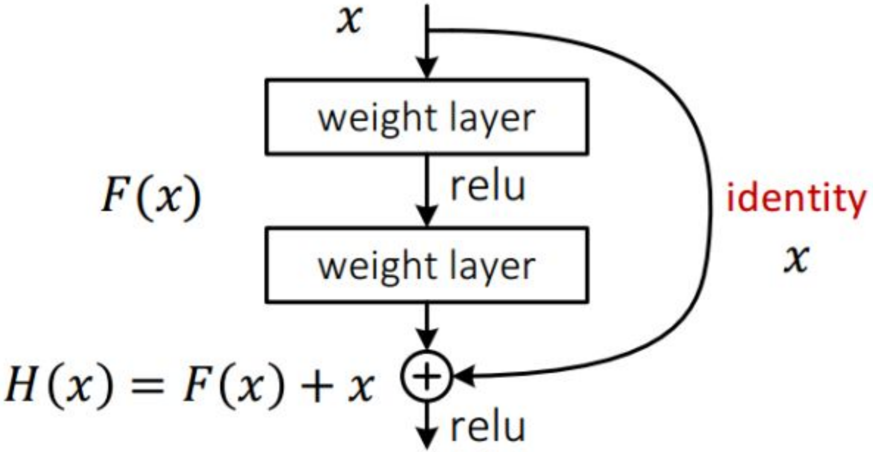
The winner in ILSVRC 2015



ResNet (2016 by Kaiming He et al.)

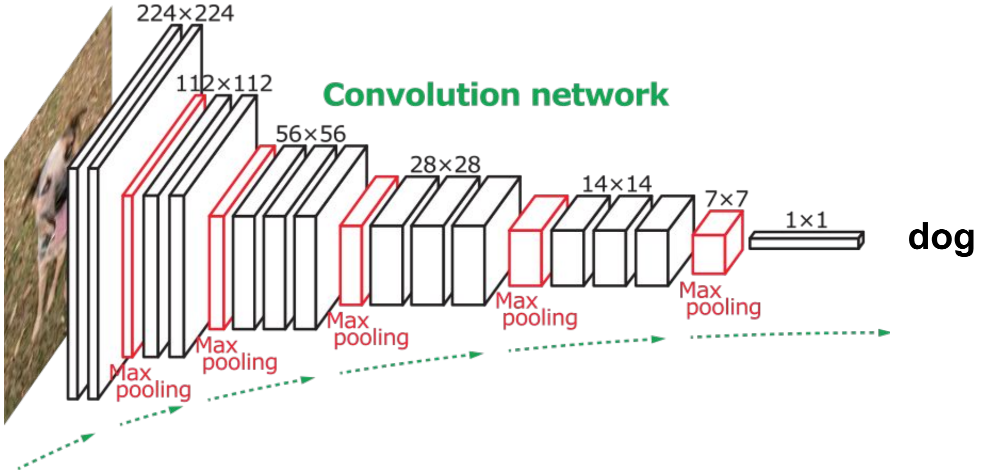


152 layers..

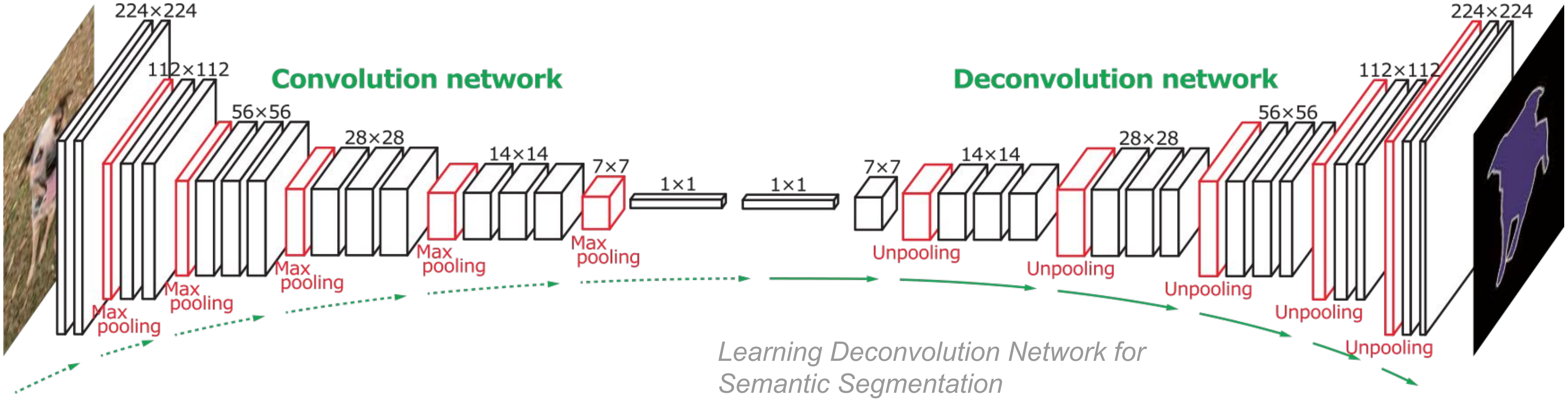


- Deeper network hard to train: Use skip connections for residual learning.
- Heavy use of batch normalization.
- No fully connected layers.

Classification:

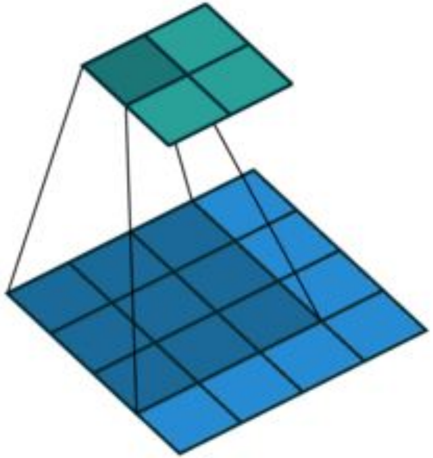


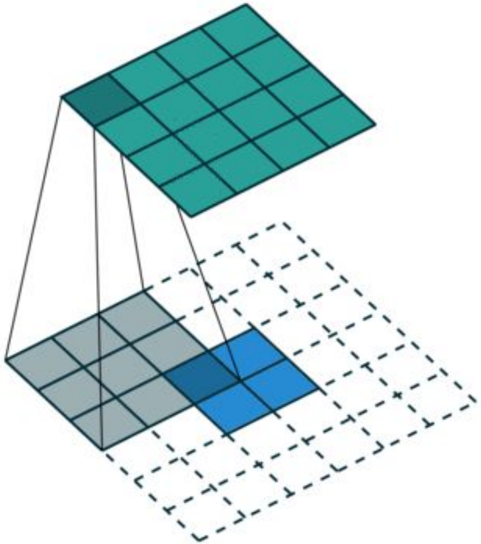
Segmentation:



Learning Deconvolution Network for Semantic Segmentation

If you know how to compute **gradients** in convolution layers, you know upconv.





Up convolution/Convolution transpose/Deconvolution

x

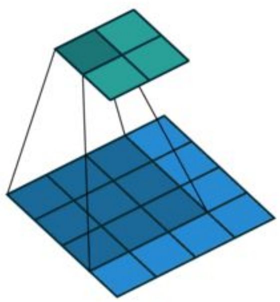
11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44

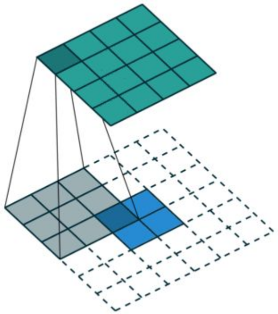
w

11	12	13
21	22	23
31	32	33

y

11	12
21	22





$$y_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{13}x_{13} + w_{21}x_{21} + w_{22}x_{22} + w_{23}x_{23} + w_{31}x_{31} + w_{32}x_{32} + w_{33}x_{33}$$

$$\frac{\partial L}{\partial x_{11}} = \sum_i \sum_j \frac{\partial L}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial x_{11}} = \frac{\partial L}{\partial y_{11}} \frac{\partial y_{11}}{\partial x_{11}} = \frac{\partial L}{\partial y_{11}} w_{11}$$

Up convolution/Convolution transpose/Deconvolution

x

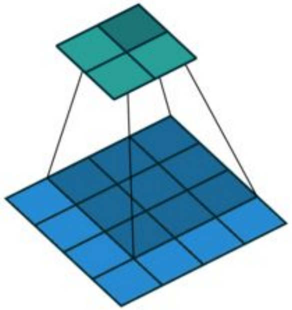
11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44

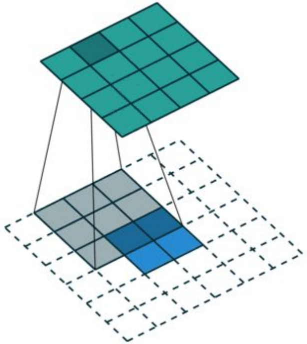
w

11	12	13
21	22	23
31	32	33

y

11	12
21	22



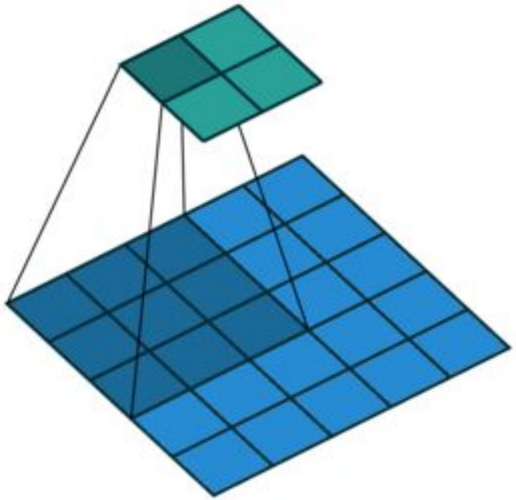


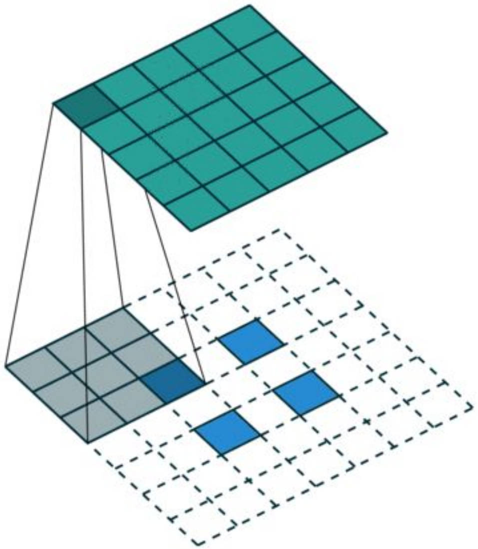
$$y_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{13}x_{13} + w_{21}x_{21} + w_{22}x_{22} + w_{23}x_{23} + w_{31}x_{31} + w_{32}x_{32} + w_{33}x_{33}$$

$$y_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{13}x_{14} + w_{21}x_{22} + w_{22}x_{23} + w_{23}x_{24} + w_{31}x_{32} + w_{32}x_{33} + w_{33}x_{34}$$

$$\frac{\partial L}{\partial x_{12}} = \sum_i \sum_j \frac{\partial L}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial x_{12}} = \frac{\partial L}{\partial y_{11}} w_{12} + \frac{\partial L}{\partial y_{12}} w_{11}$$

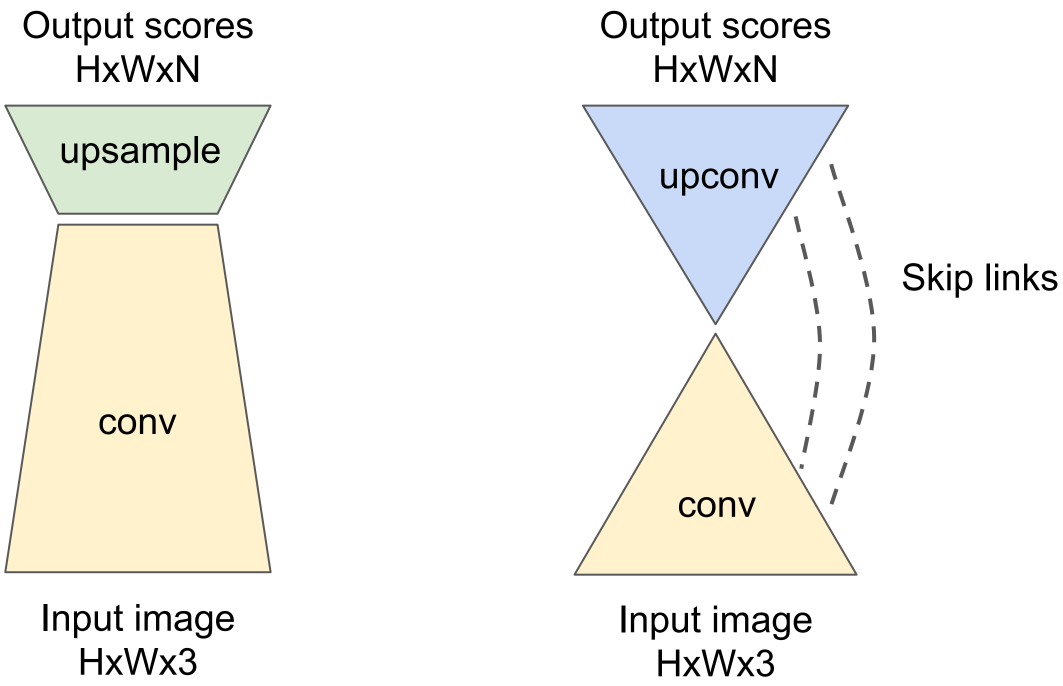
Convolution with stride =>
Upconvolution with input upsampling





See https://github.com/vdumoulin/conv_arithmetic for examples

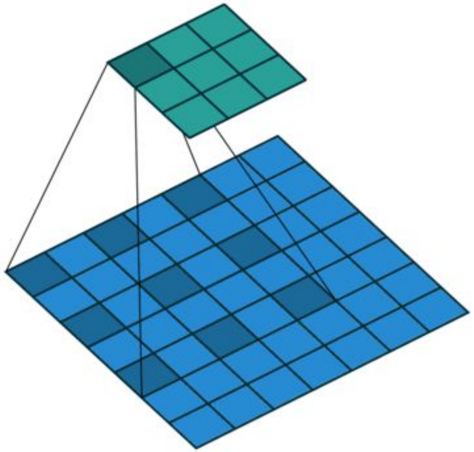
Fully convolutional network **(FCN) variations**



Issues with convolution in dense prediction (image segmentation)

- Use small kernels
    - Receptive field grows linearly with #layers: `l*(k-1)+k`
- Use large kernels
    - loss of resolution

Dilated convolutions support exponentially expanding receptive fields without losing resolution or coverage.



dilation=2

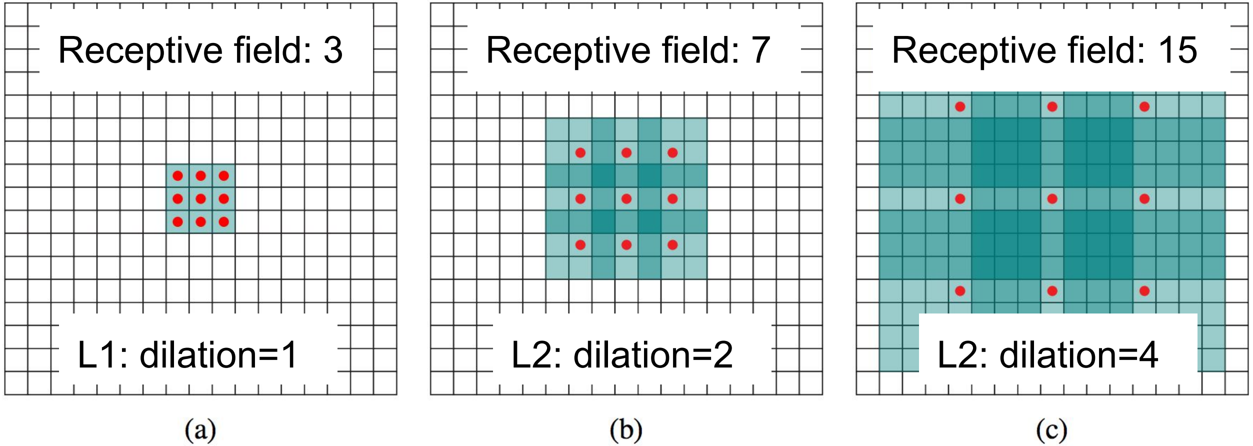


Fig from ICLR 16 paper by Yu and Koltun.

Dilated/Atrous Convolution

Baseline: conv + FC Dilated conv

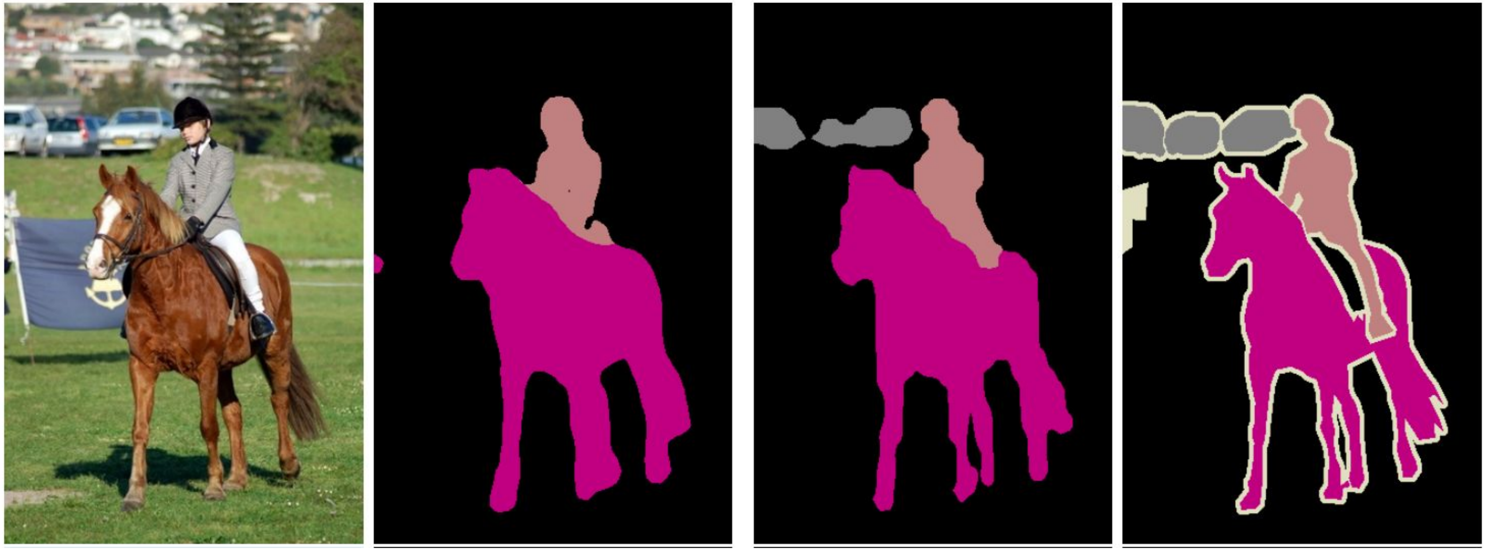


Fig from ICLR 16 paper by Yu and Koltun.

Outline

- Motivation
- A Simple Neural Network
- Ideas in Deep Net Architectures
- **Ideas in Deep Net Optimization**
- Practicals and Resources

Optimization

Basics: Gradient descent, SGD, mini-batch SGD, Momentum, Adam, learning rate decay

Other Ingredients: Data augmentation, Regularization, Dropout, Xavier initialization, Batch normalization

NN Optimization:

Back Propagation [Hinton et al. 1985]

Gradient Descent with Chain Rule Rebranded.

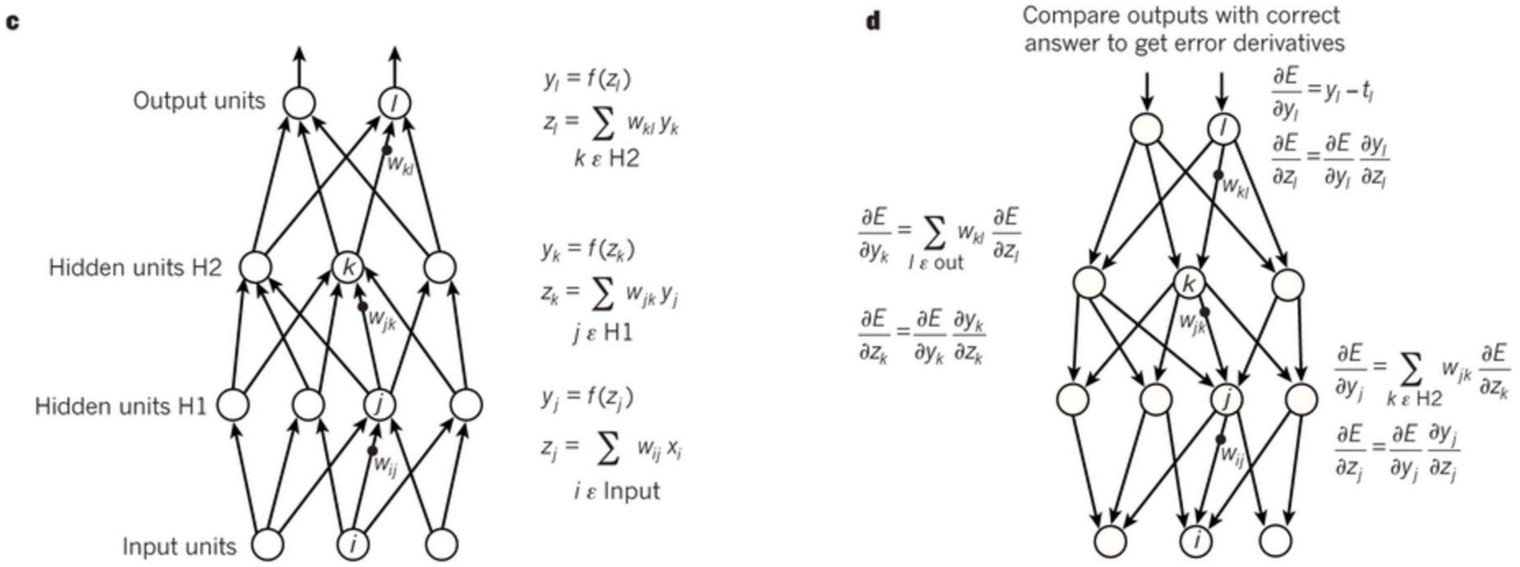


Fig from Deep Learning by LeCun, Bengio and Hinton. Nature 2015

SGD, Momentum, RMSProp, Adagrad, Adam

- **Batch gradient descent (GD):**
  - Update weights once after looking at all the training data.
- **Stochastic gradient descent (SGD):**
  - Update weights for each sample.
- **Mini-batch SGD:**
  - Update weights after looking at every “mini batch” of data, say 128 samples.

Let x be the weight/parameters, dx be the gradient of x. In mini-batch, dx is the average within a batch.

SGD (the vanilla update)

```
# Vanilla update
x += - learning_rate * dx
```

where learning_rate is a hyperparameter - a fixed constant.

From CS231N

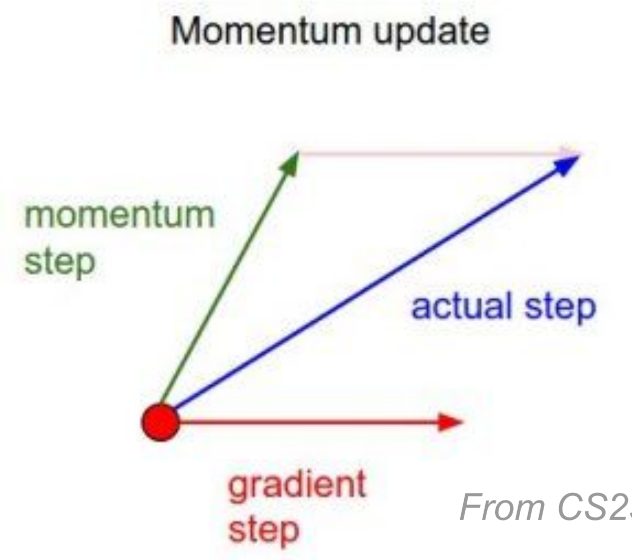
SGD, Momentum, RMSProp, Adagrad, Adam

Momentum:

```
# Momentum update
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```

Initializing the parameters with random numbers is equivalent to setting a particle with zero initial velocity at some location.

The optimization process can then be seen as equivalent to the process of **simulating the parameter vector (i.e. a particle) as rolling on the landscape.**



From CS231N

SGD, Momentum, RMSProp, Adagrad, Adam

Per-parameter adaptive learning rate methods

Adagrad by Duchi et al.:

```
# Assume the gradient dx and parameter vector x
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

weights with high gradients => effective learning rate reduced

RMSProp by Hinton:

```
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

Use moving average to reduce Adagrad’s aggressive, **monotonically** decreasing learning rate

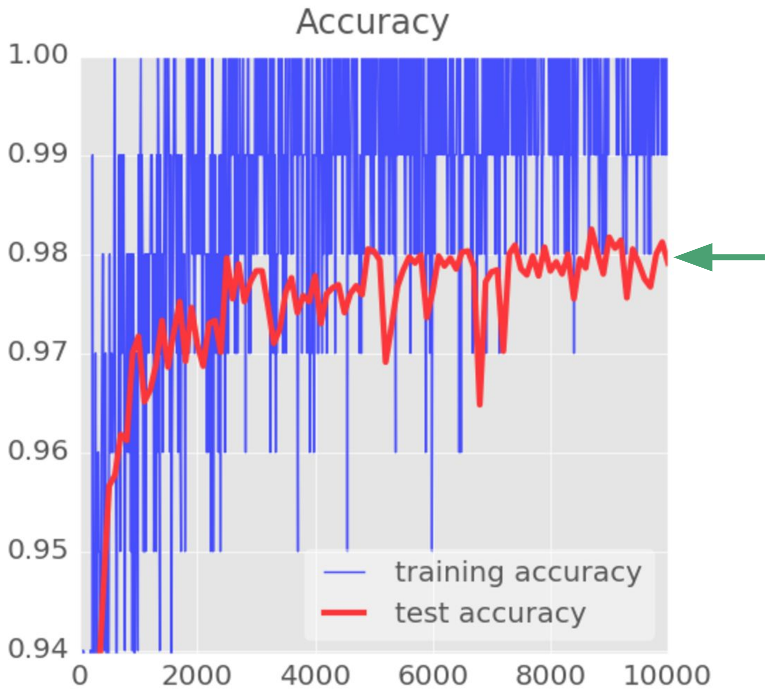
Adam by Kingma et al.:

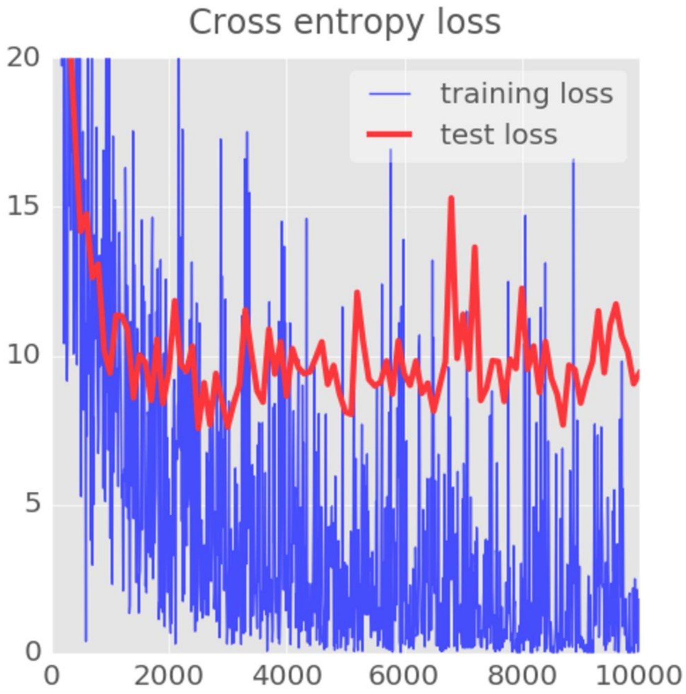
```
m = beta1*m + (1-beta1)*dx
v = beta2*v + (1-beta2)*(dx**2)
x += - learning_rate * m / (np.sqrt(v) + eps)
```

Use smoothed version of gradients compared with RMSProp. **Default optimizer (along with Momentum).**

From CS231N

Annealing the learning rate (the dark art...)

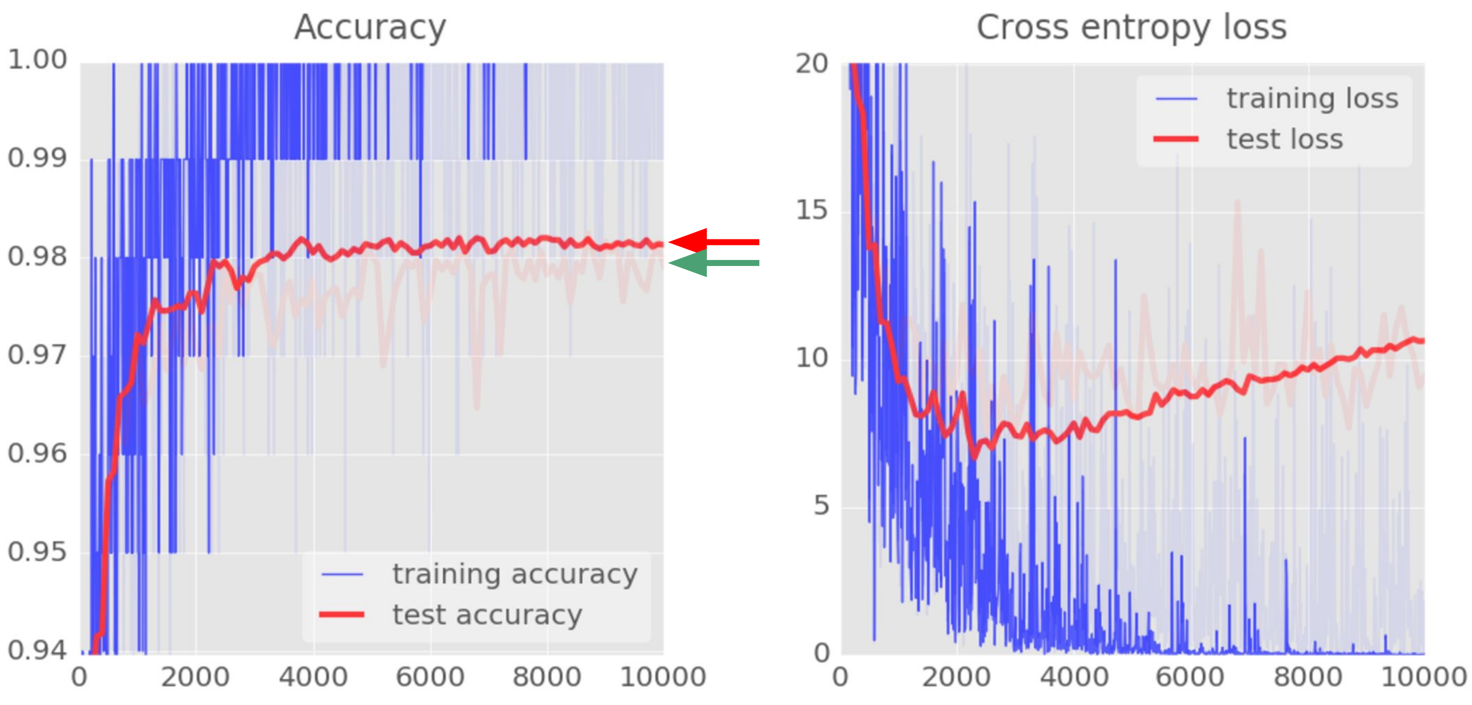




Learning rate 0.003

From Martin Gorner

Annealing the learning rate (the dark art...)



Learning rate 0.003 at start then dropping exponentially to 0.0001

From Martin Gorner

Annealing the learning rate (the dark art...)

- **Stairstep decay:** Reduce the learning rate by some factor every few epochs. E.g. half the learning rate every 10 epochs.
- **Exponential decay:** learning_rate = initial_lr * exp(-kt) where t is current step.
- **"On-demand" decay:** Reduce the learning rate when error plateaus

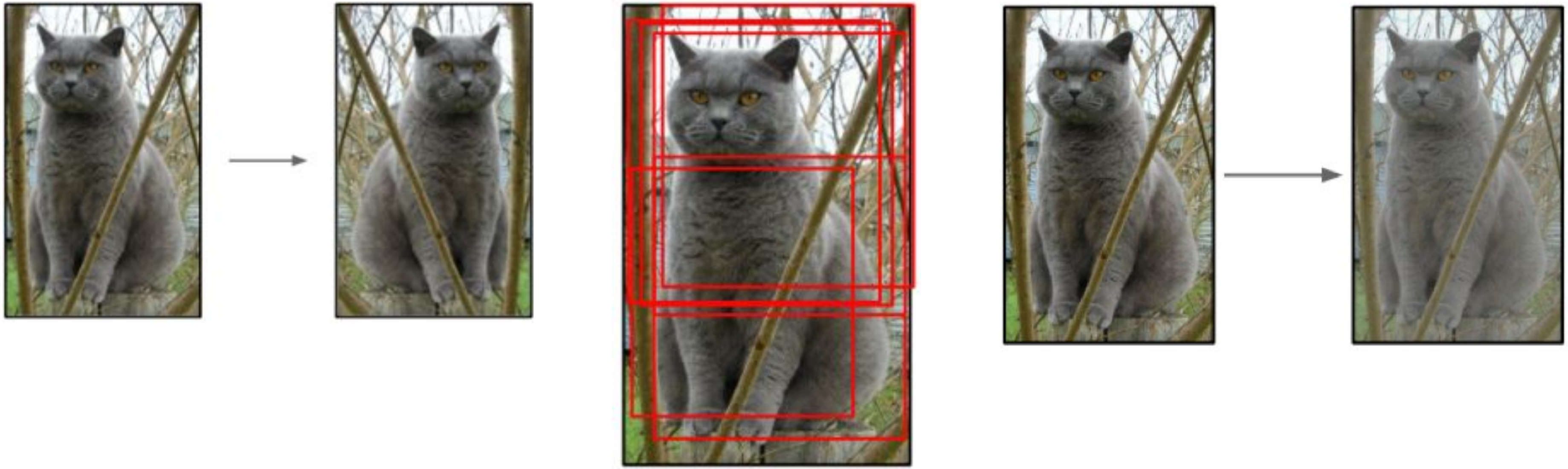
Optimization

Basics: Gradient descent, SGD, mini-batch SGD, Momentum, Adam, learning rate decay

Other Ingredients: Data augmentation, Regularization, Dropout, Xavier initialization, Batch normalization

Dealing with Overfitting: Data Augmentation

Flipping, random crop, random translation, color/brightness change, adding noise...

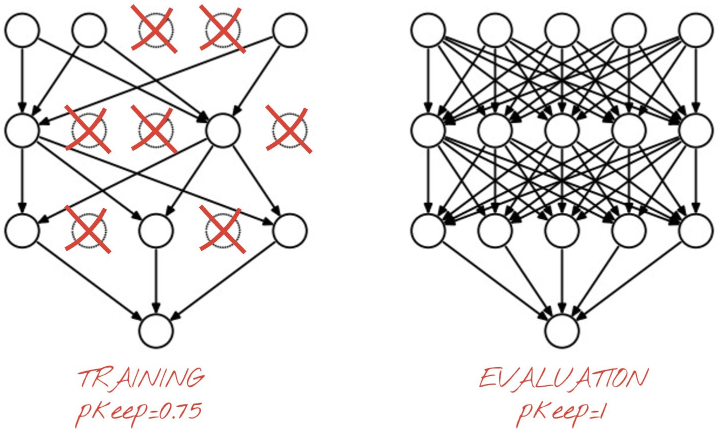


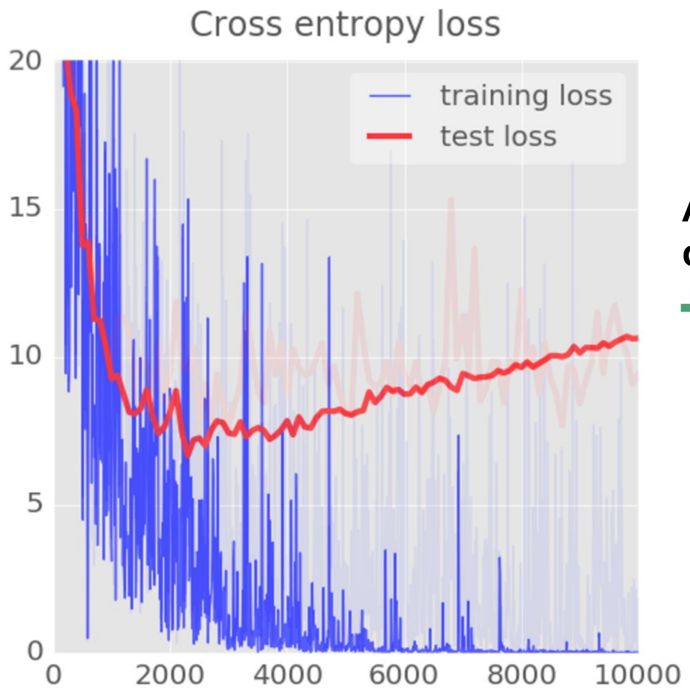
Pictures from CS231N

Dealing with Overfitting: Regularization, Dropout

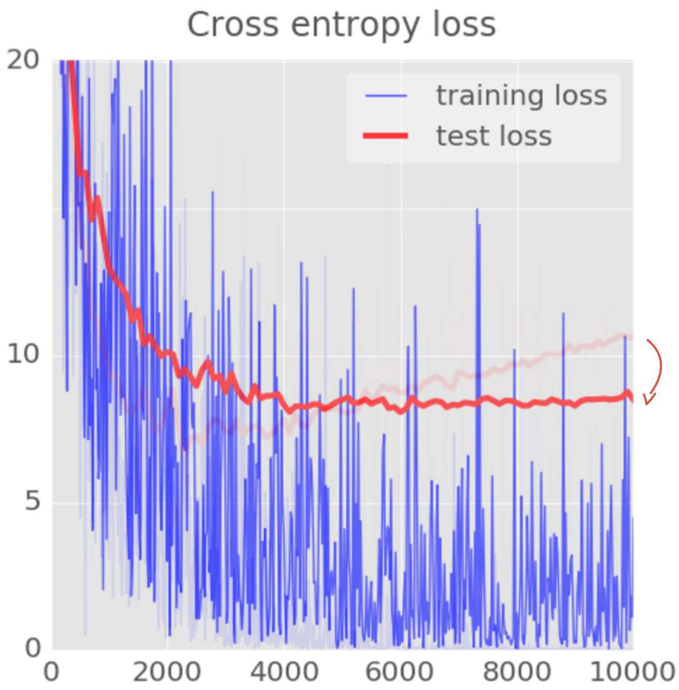
L1/L2 regularization on weights: limit the network capacity by encouraging distributed and sparse weights. When combining L1 and L2 regularization, it's called elastic net regularization: $\lambda_1 \mid w \mid + \lambda_2 w^2$

Dropout by Srivastava et al.:
During testing there is no dropout applied,
with the interpretation of evaluating an averaged
prediction across the exponentially-sized
ensemble of all sub-networks.





Applying dropout during training



Xavier and MSR Initialization

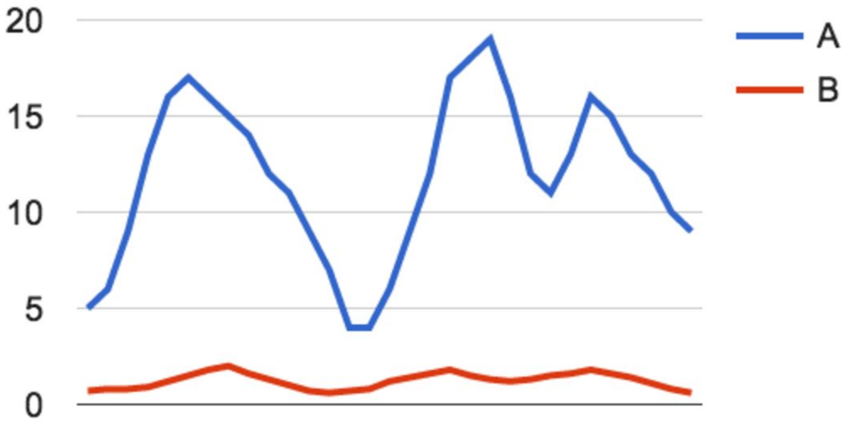
```
W = 0.01* np.random.randn(D,H)
```

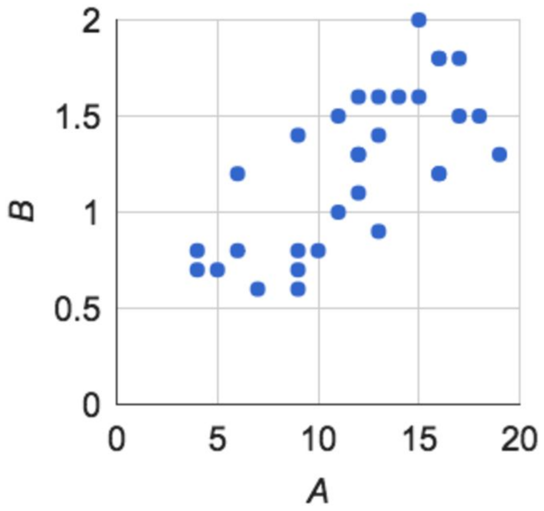
Problem with random Gaussian initialization: **the distribution of the outputs has a variance that grows with the number of inputs** => Exploding/diminishing output in very deep network.

```
w = np.random.randn(n) / sqrt(n).
```

```
w = np.random.randn(n) * sqrt(2/n).
```

Data “whitening”

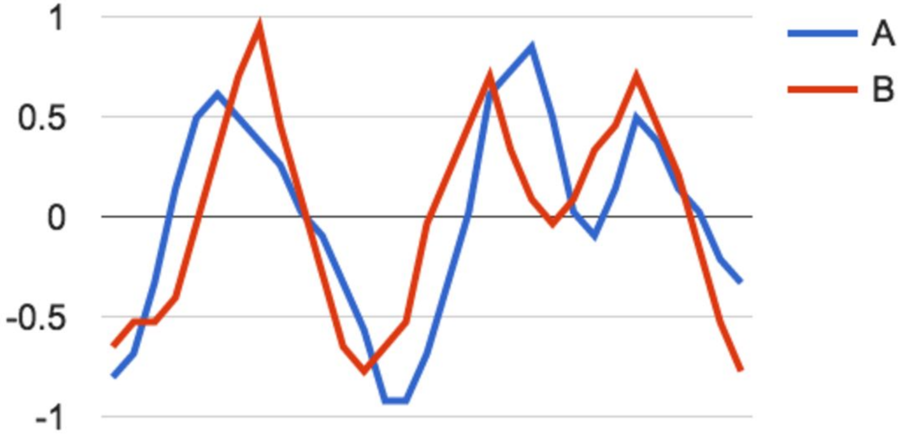


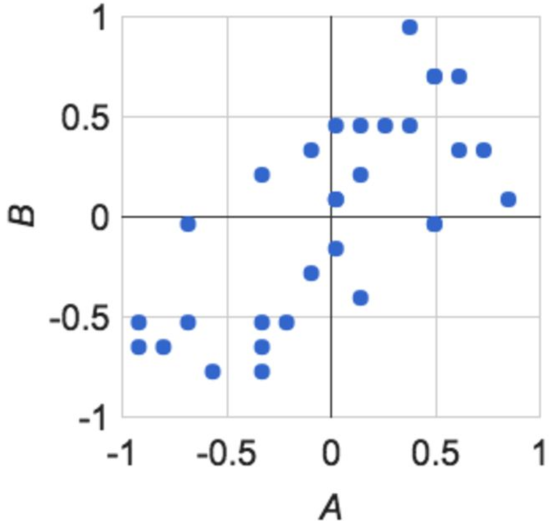


Data: large values, different scales, skewed, correlated

From Martin Gorner

Data “whitening”





Modified data: centered around zero, rescaled...

Subtract average
Divide by std dev

From Martin Gorner

Batch Normalization

"logit" = weighted sum + bias

Compute average and variance on mini-batch

Center and re-scale logits
before the activation function
(decorrelate ? no, too complex)

$$\widehat{x} = \frac{x - avg_{batch}(x)}{stdev_{batch}(x) + \epsilon}$$

From Martin Gorner

Batch Normalization

"logit" = weighted sum + bias

Compute average and variance on mini-batch

Center and re-scale logits
before the activation function
(decorrelate ? no, too complex)

$$\hat{x} = \frac{x - avg_{batch}(x)}{stdev_{batch}(x) + \epsilon}$$

one of each per neuron

Add learnable scale and offset
for each logit so as to restore expressiveness

$$BN(x) = \alpha\hat{x} + \beta$$

Try α=stdev(x) and β=avg(x) and you have BN(x) = x

From Martin Gorner

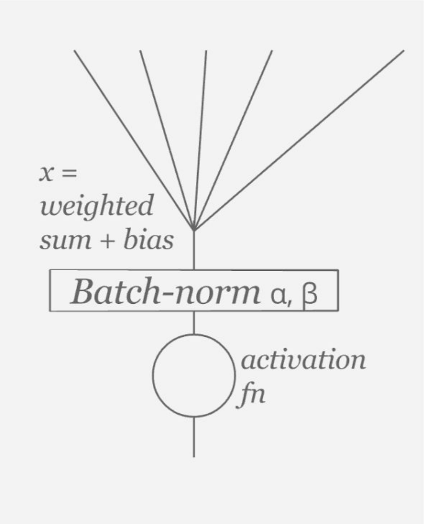
Batch Normalization

+You can go faster: use higher learning rate
+BN also regularises: lower or remove dropout

depends from:
weights, biases, images

depends from:
same weights and biases, images
only one set of weights and biases in a mini-batch

$$\hat{x} = \frac{x - avg_{batch}(x)}{stdev_{batch}(x) + \epsilon}$$



$$BN(x) = \alpha\hat{x} + \beta$$

=> BN is differentiable relatively to weights, biases, α and β
It can be used as a layer in the network, gradient calculations will still work

From Martin Gorner

Outline

- Motivation
- A Simple Neural Network
- Ideas in Deep Net Architectures
- Ideas in Deep Net Optimization
- **Practicals and Resources**

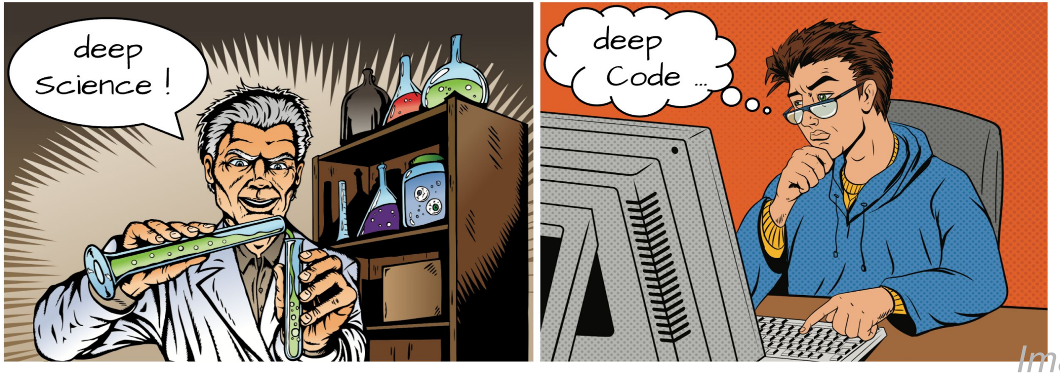


Image from Martin Gorner

Data Collecting, Cleaning, Preprocessing > 50% time

“OS” of Machine/Deep Learning

Caffe, Theano, Torch, Tensorflow, Pytorch, MXNET, …

Matlab in the earlier days. Python and C++ is the popular choice now.

Deep network debugging, Visualizations

Resources

Stanford CS231N: Convolutional Neural Networks for Visual Recognition

Stanford CS224N: Natural Language Processing with Deep Learning

Berkeley CS294: Deep Reinforcement Learning

Learning Tensorflow and deep learning, without a PhD

Udacity and Coursera classes on Deep Learning

Book by Goodfellow, Bengio and Courville: http://www.deeplearningbook.org/

Talk by LeCun 2013: http://www.cs.nyu.edu/~yann/talks/lecun-ranzato-icml2013.pdf

Talk by Hinton, Bengio, LeCun 2015:
https://www.iro.umontreal.ca/~bengioy/talks/DL-Tutorial-NIPS2015.pdf

What's not covered...

Sequential Models (RNN, LSTM, GRU)

Deep Reinforcement Learning

3D Deep Learning (MVCNN, 3D CNN, Spectral CNN, NN on Point Sets)

Generative and Unsupervised Models (AE, VAE, GAN etc.)

Theories in Deep Learning

...

Summary

- Why Deep Learning
- A Simple Neural Network
    - Model, Loss and Optimization
- Ideas in deep net architectures
    - **Building blocks:** FC, ReLU, conv, pooling, unpooling, upconv, dilated conv
    - **Classics:** MLP, LeNet, AlexNet, NIN, VGG, GoogleNet, ResNet
- Ideas in deep net optimization
    - **Basics:** GD, SGD, mini-batch SGD, Momentum, Adam, learning rate decay
    - **Other Ingredients:** Data augmentation, Regularization, Dropout, Batch normalization
- Practicals and Resources