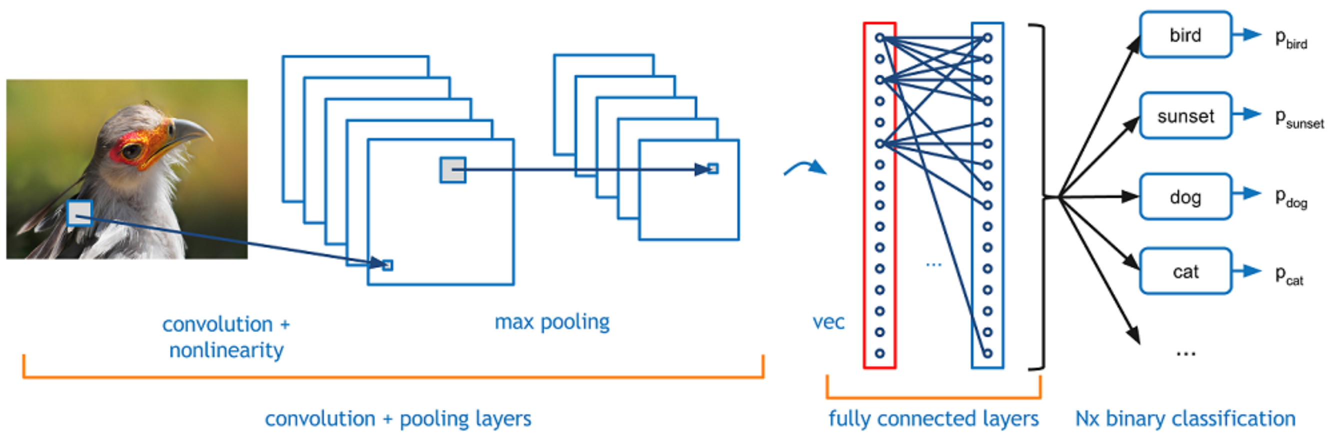
CS-7495 Computer Vision

Deep Neural Networks



Slide set prepared by A. Agarwal

Credits

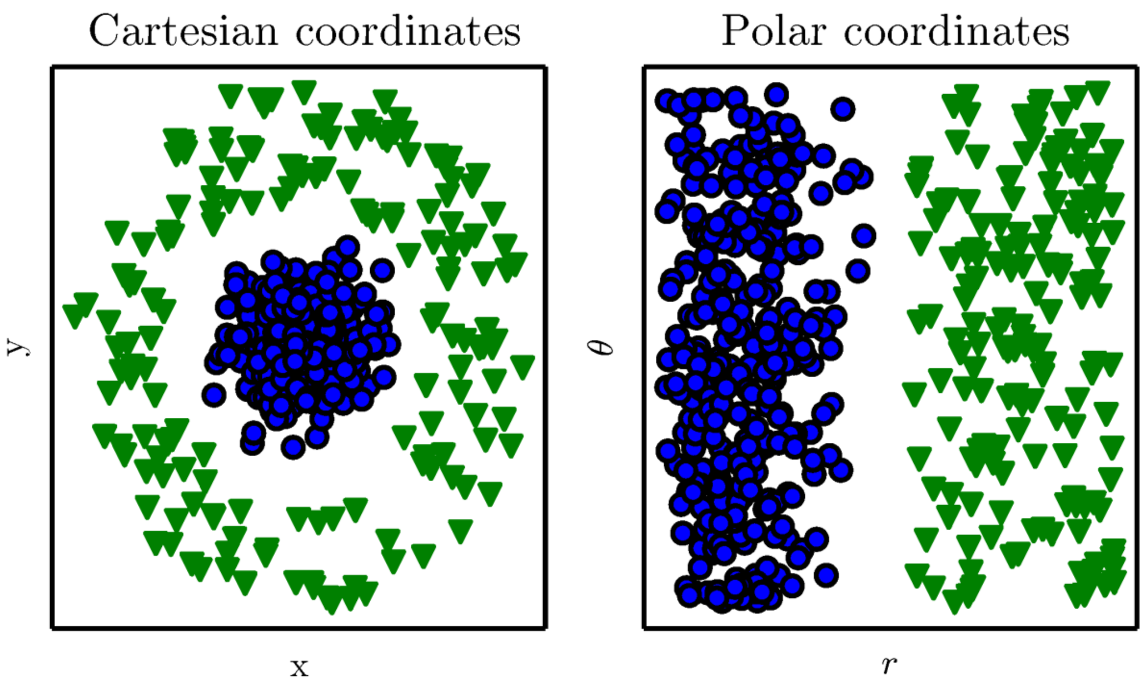
- Yoshua Bengio, Geoff Hinton, Yann LeCun, Andrew Ng, Marc'Aurelio Ranzato, …

Quick Overview

- Early AI successes relied on hard-coded knowledge representation
  - IBM's Deep Blue – chess board and rule representation was not challenging
- AI systems needed to acquire their own knowledge
- Machine Learning – extracting patterns from raw data
  - ML Algorithm *Logistic Regression* can recommend cesarean delivery
  - Another ML Algorithm *naïve Bayes* can separate spam email
- Performance of these simple ML algorithms depends heavily on the data representation

Data Representation Problem

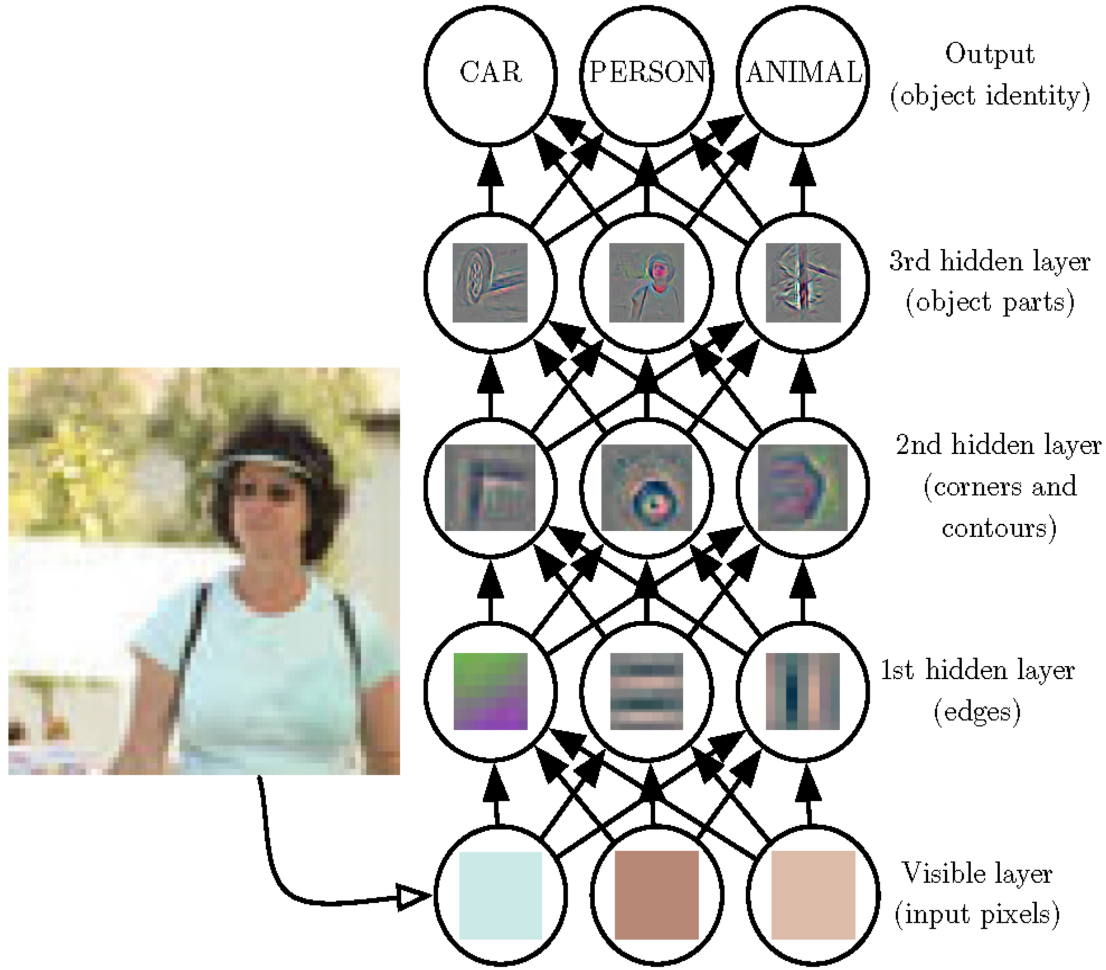
- Trying to separate two categories of data by drawing a line between them on a scatterplot



- Representation Learning – ML to learn not only data to output but also learn the representation itself
  - *Autoencoders*
- Factors of Variation – Unobserved factors that affect observable quantities
  - While observing image of a car
    - Position
    - Color
    - Angle
    - Illumination Conditions etc.
- Deep Learning tries to solve the representation problem by introducing representations that are expressed in other simpler representations

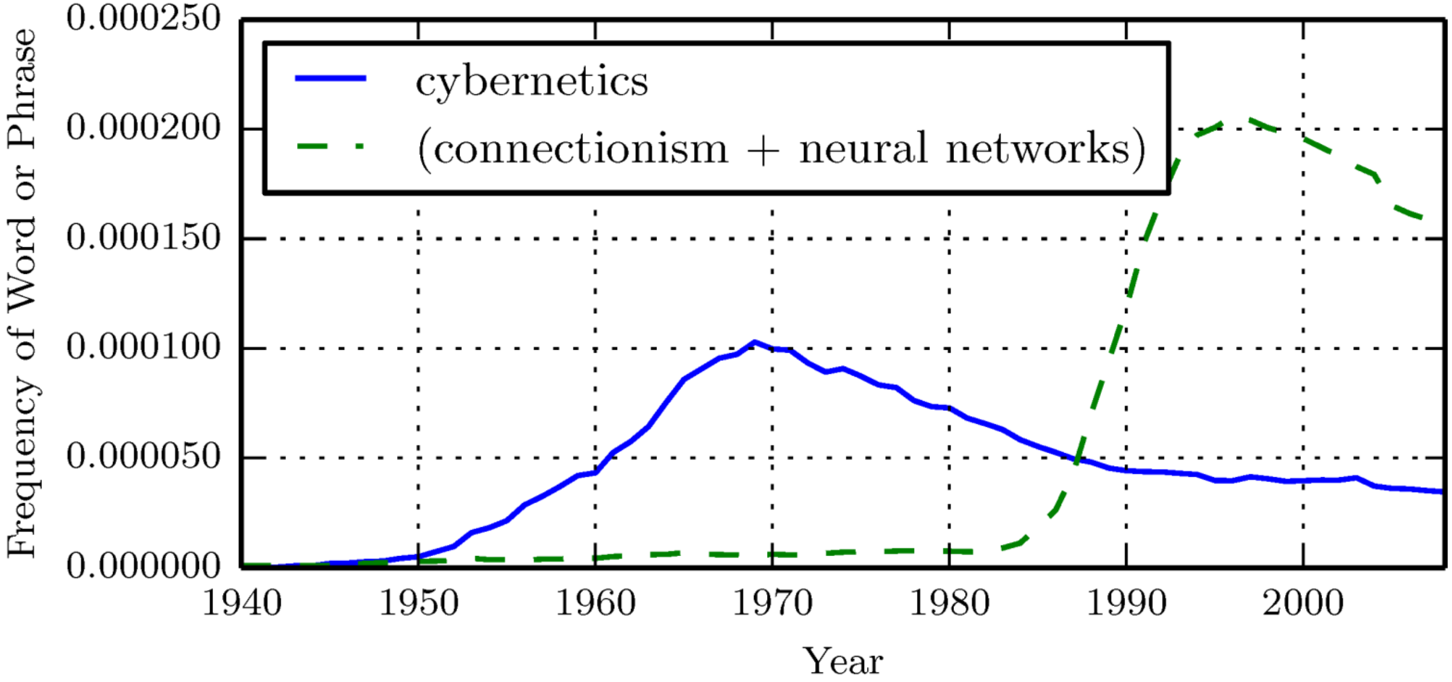
Deep Learning Model

- Input is presented at the Visible Layer
- Series of Hidden Layers extract increasingly abstract features from the image



Historical Trends

- Deep Learning dates back to 1940s
  - Cybernatics 1940s – 1960s
  - Connectionism 1980s – 1990s
  - Deep Learning 2006 - …



Historical Trends

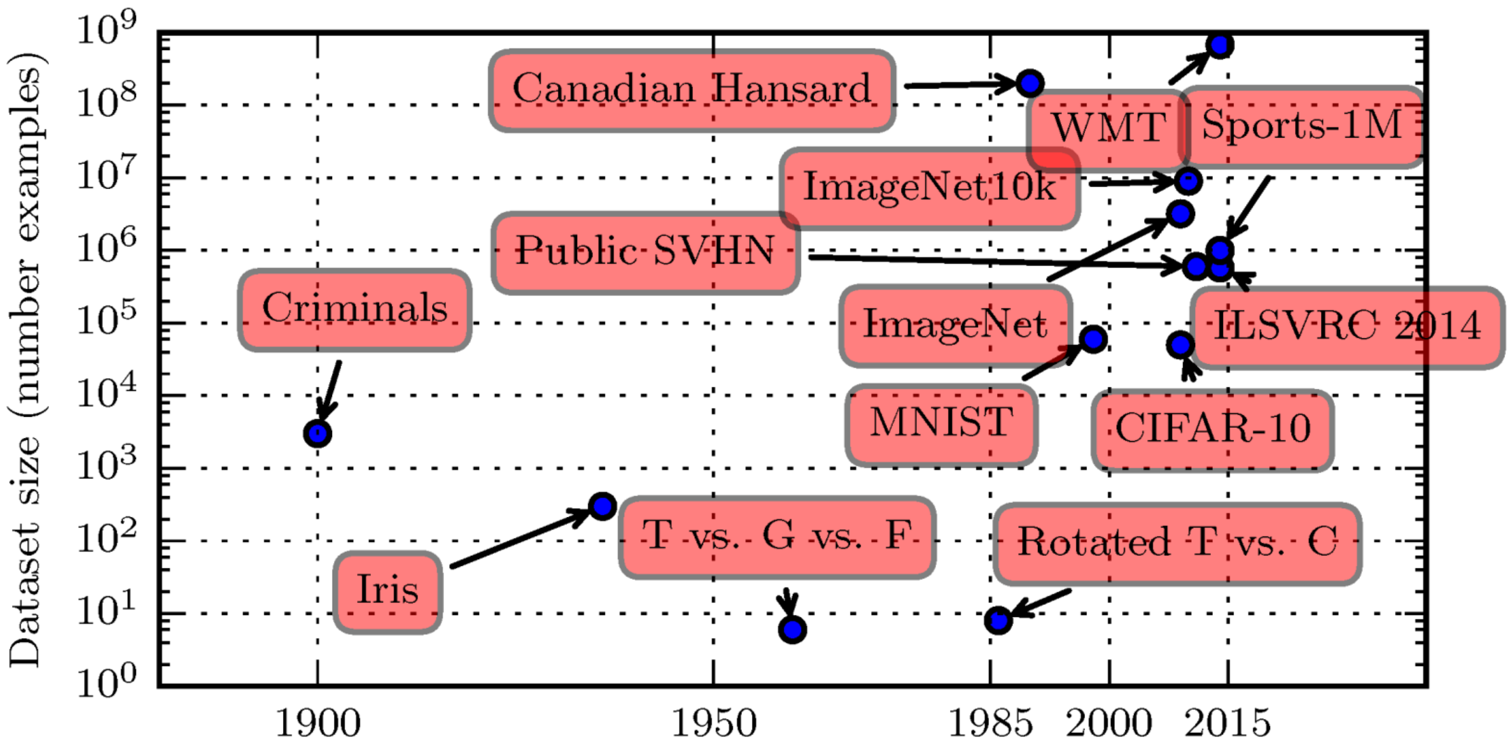
- McCulloch-Pitts neuron (1943) – Early model of brain function
  - Linear model that could recognize two different categories of inputs
- Perceptron (Rosenblatt 1958, 1962)
  - The first model that could learn the weights that defined the categories
- Adaptive Linear Element ADALINE (Widrow and Hoff 1960)
  - The training algorithm used to adapt weights was a special case of Stochastic Gradient Descent
- These Linear Models however have limitations
  - eg. can not learn XOR Function

Historical Trends

- Connectionism or Parallel Distributed Processing (1980s)
  - Large number of simple computational units can achieve intelligent behavior when connected together
- Distributed Representation (Hinton 1986)
  - Each input to a system should be represented by many features and each feature should be involved in representation of many possible inputs
- Deep Belief Network (Hinton 2006)
  - A type of NN that could be trained efficiently using greedy layer-wise pre-training.
- Most neurons today are based on modern neuron called rectified Linear Unit reLU

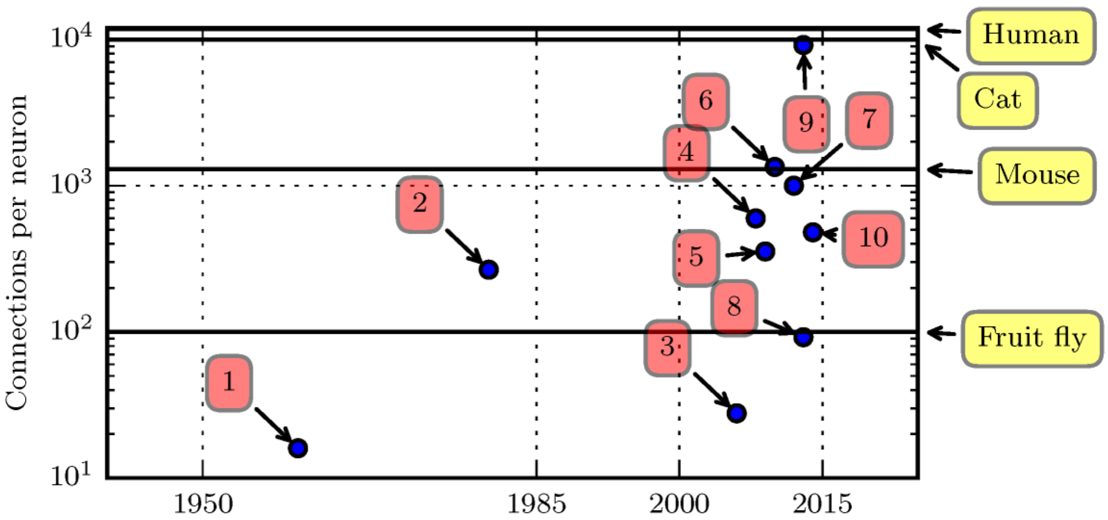
Increasing Dataset Sizes

- Deep learning has been successfully used in commercial applications since 1990s but was often regarded as art more than technology



Increasing Model Sizes

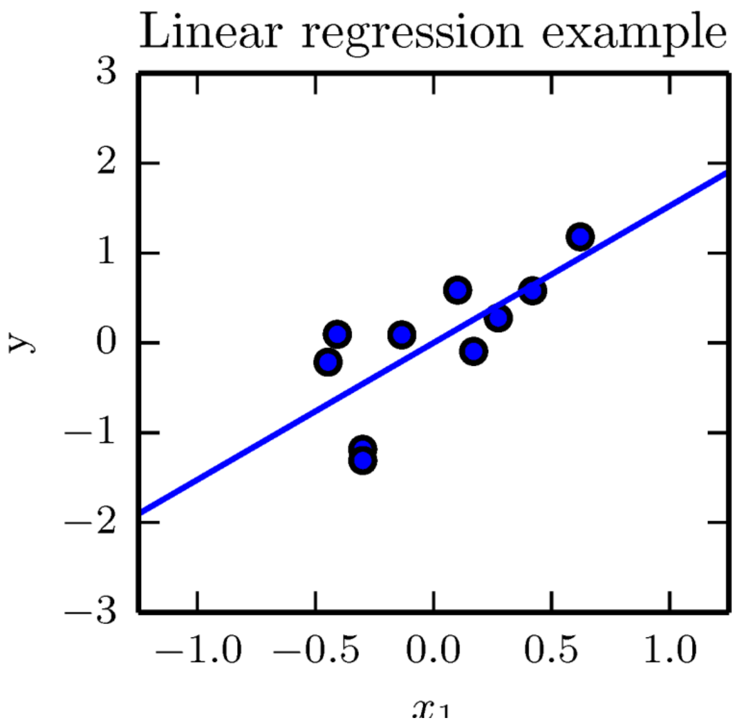
- Computational resources now allow much larger model to run
  - Animals become intelligent when many of their neurons work together



1. Adaptive Linear Element
2. Neocognition
3. GPU accelerated CNN
4. Deep Boltzmann machine
5. Unsupervised CNN
6. GPU accelerated multilayer perceptron
7. Distributed Autoencoder
8. Multi-GPU CNN
9. COTS HPC Unsupervised CNN
10. GoogLeNet

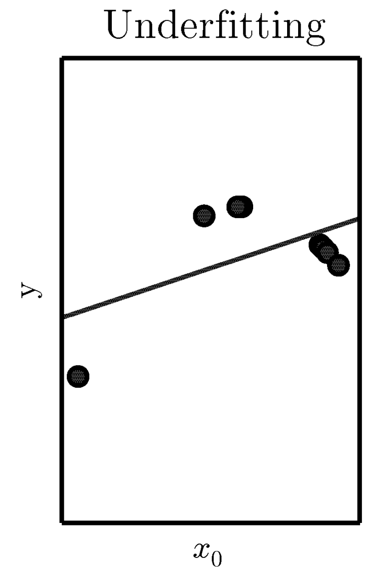
Machine Learning Basics

- Learning (Mitchell 1997)
  - *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E*
  - Example: Linear Regression

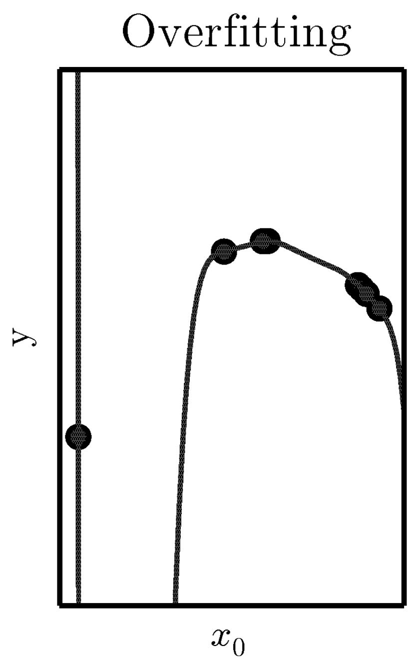


Capacity, Overfitting and Underfitting

- ML algorithm should perform well on new, previously unseen inputs



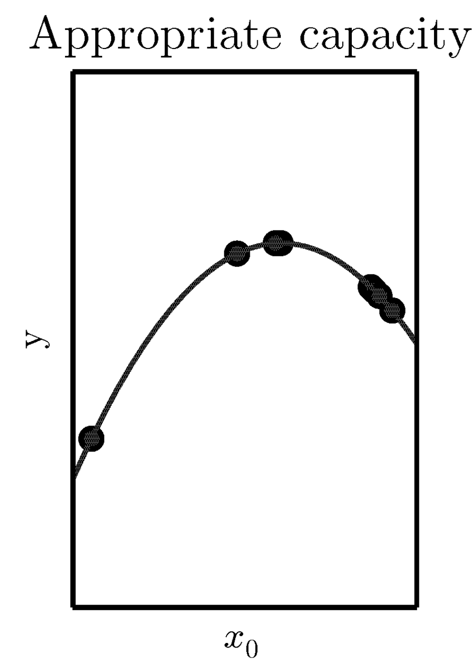
- *Underfitting – when the model is not able to obtain sufficiently low error value on the training set*

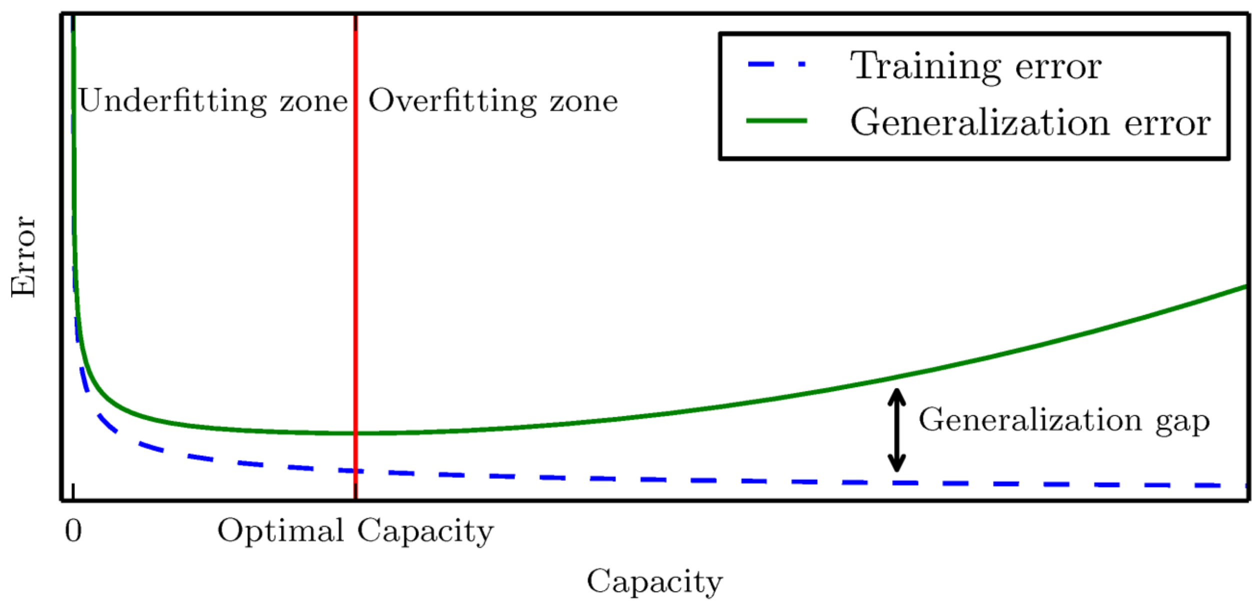


- *Overfitting– when the gap between the training error and test error is too large*

Capacity, Overfitting and Underfitting

- *Capacity – a model's ability to fit wide variety of functions*

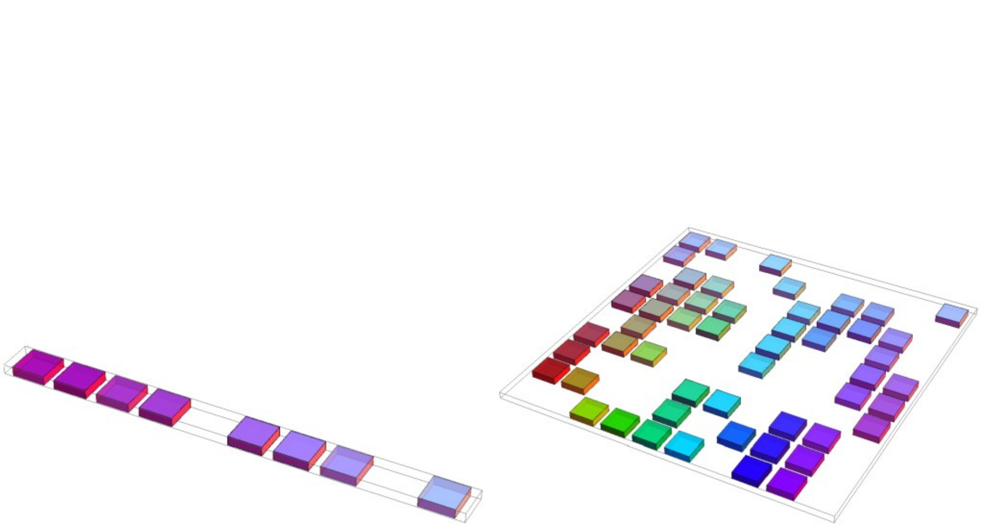




Typical relationship between capacity and error

Challenges Motivating Deep Learning

- The Curse of Dimensionality
  - The number of possible distinct configurations of a set of variables increases exponentially as the number of variables increase



Challenges Motivating Deep Learning

- Local Constancy and Smoothness Regularization
  - To generalize well, ML algorithms need to be guided by prior beliefs about the kind of function they should learn
  - Smoothness or Local constancy prior states that the function we learn should not change very much within a small region

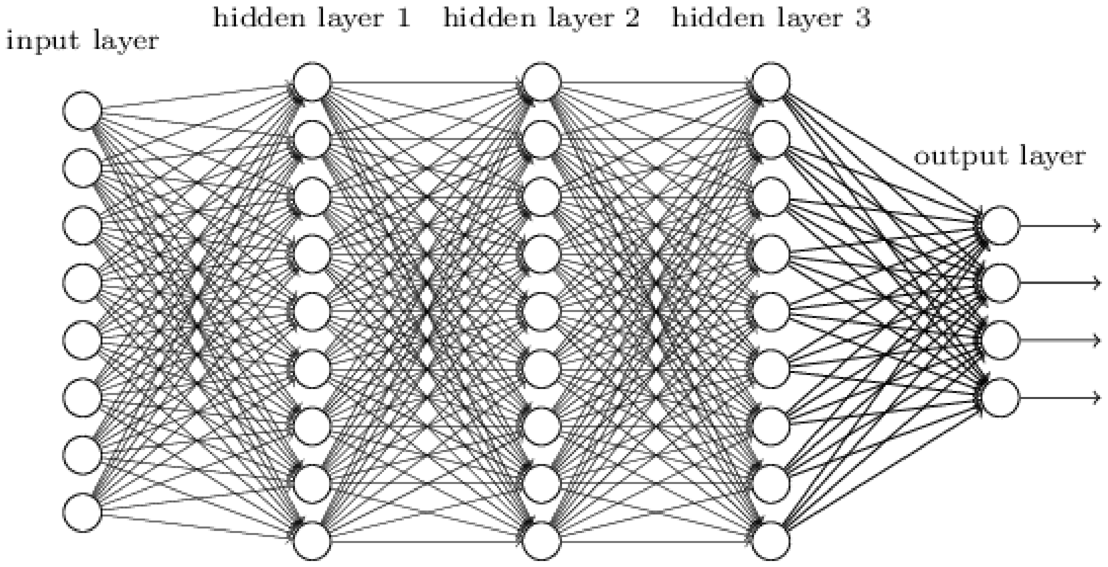
$$f^*(x) \approx f^*(x+\varepsilon)$$

for most configurations x and small change ε

Challenges Motivating Deep Learning

- Manifold Learning
  - Manifold is a connected region or set of points associated with a neighborhood around each point
  - Manifold learning algorithms assume that most $\mathbb{R}^n$ consists of invalid inputs, and that interesting inputs occur only along a collection of manifolds containing a small subset of points

Deep Feedforward Networks



Deep Feedforward Networks aka Feedforward Neural Networks aka Multilayer Perceptrons (MLPs)

$$y = f(x;\theta)$$

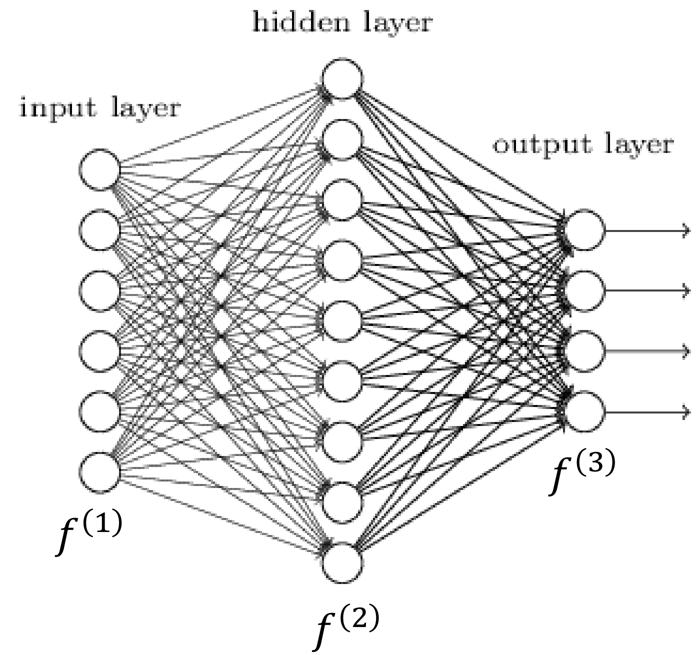
- Feedforward network learns the value of parameters θ that results in the best function approximation
- Information flows thorough the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y.
- There are no feedback connections in which the outputs of the model are fed back into itself

Deep Feedforward Networks

- Typically represented by composing together many different functions
- Model is associated with a directed acyclic graph describing how the functions are composed together

$$f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$$

- The overall length of the chain gives the depth of the model



Deep Feedforward Networks

- To extend linear models to represent nonlinear functions of x, the linear model can be applied to transformed input $\phi(x)$ where ϕ is a nonlinear transformation.
- How to choose mapping ϕ
  - Use a very generic ϕ such as *infinite-dimensional* ϕ
  - Manually engineer ϕ
  - Learn ϕ (Deep Learning Approach)

$$y = f(x;\theta,w) = \phi(x;\theta)^T w$$

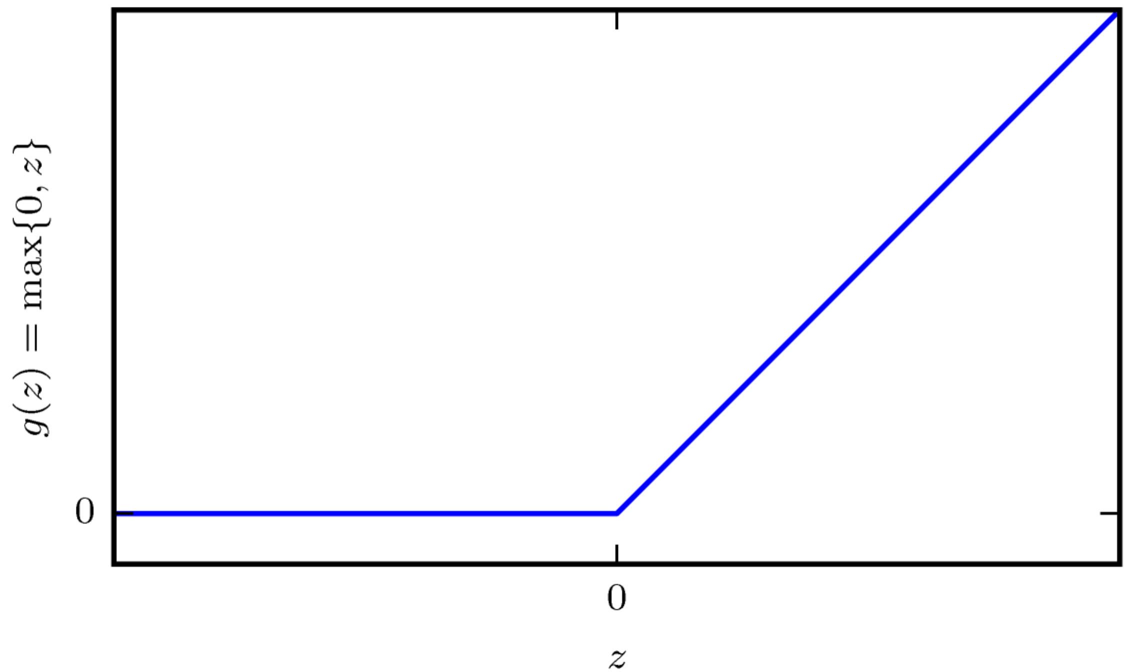
    - Parameters θ are used to learn ϕ from a broad class of functions
    - Parameters w map from $\phi(x)$ to the desired output.

Deep Feedforward Networks – Design Decisions

- Gradient Based Learning Design Choices
  - The cost function
  - The form of the output units
- Activation Functions for Hidden Layers
- Architecture of the network
  - Number of layers
  - Connection between layers
  - Number of units in each layer

Rectified Linear Units ReLU

- They use activation function $g(z) = max\{0, z\}$
- The only difference between linear unit and ReLU is that ReLU outputs zero across half its domain



- The ReLU however cannot learn via gradient based methods for examples where activation is zero

Generalizations of Rectified Linear Units ReLU

- Generalizations of ReLU based on using a nonzero slope α_i when $z_i < 0$: $h_i = g(z, \alpha)_i = \max(0, z_i) + \alpha_i \min(0, z_i)$
  - Absolute value rectification fixes $\alpha_i = -1$ to obtain $g(z) = |z|$
    - Used for object recognition from images (Jarrett 2009)
  - Leaky ReLU (Maas 2013) fixes α_i to a small value like 0.01
  - Parametric ReLU or PReLU treats α_i as a learnable parameter
- Maxout Units (Goodfellow 2013a) instead of applying an element-wise function $g(z)$, they divide z into groups of k values. Each maxout unit then outputs the maximum element of one of these groups:

$$g(z)_i = \max_{j \in G^{(i)}} z_i$$

Other Hidden Units

- Logistic sigmoid activation function $g(z) = \sigma(z)$
- Hyperbolic tangent activation function $g(z) = tanh(z)$
- Radial basis function (RBF) $h_i = \exp(-\frac{1}{\sigma_i^2}\left\|W_{:,i} - x\right\|^2)$
- Softplus $g(a) = \zeta(a) = \log(1 + e^a)$
- Hard tanh $g(a) = \max(-1, \min(1, a))$

Architecture of the Network

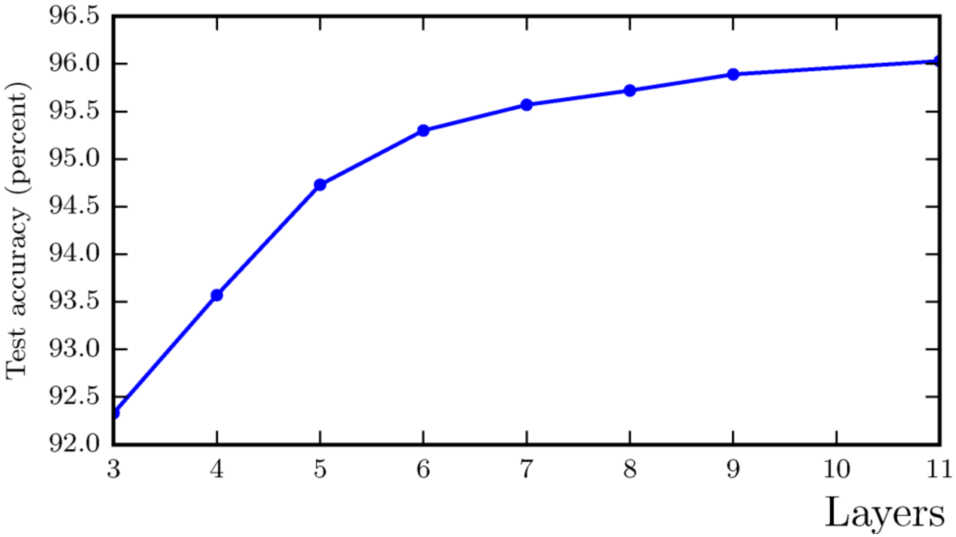
- Most neural networks are organized into groups of units called layers arranged in a chain structure
  - Each layer being a function f the layer that precedes it
  - First layer is given by
  $$h^{(1)} = g^{(1)}(W^{(1)T}x + b^{(1)})$$
  - Second layer is given by
  $$h^{(2)} = g^{(2)}(W^{(2)T}h^{(1)} + b^{(2)})$$
  - …

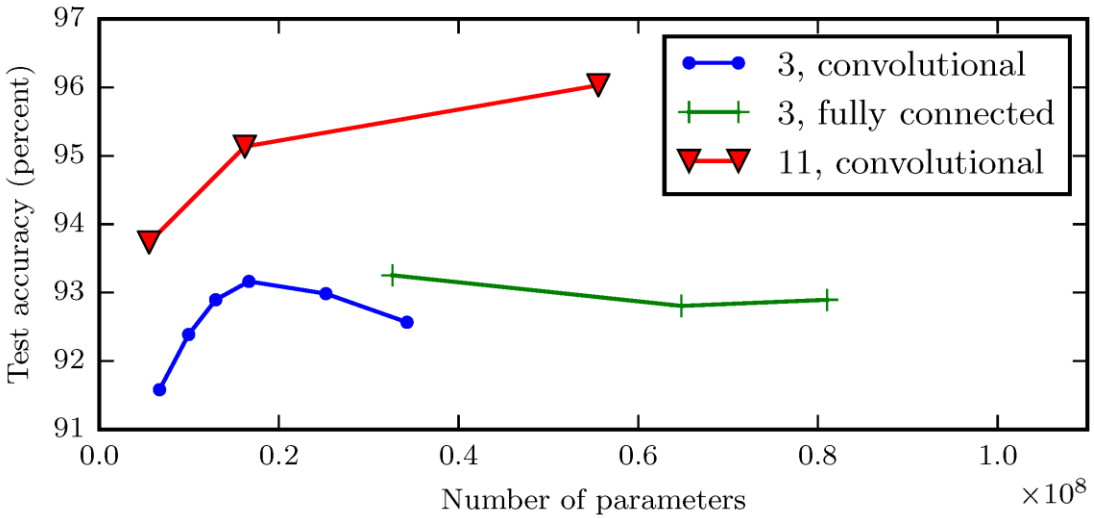
Architecture of the Network

- Feedforward networks with hidden layers provide a universal approximation framework.
- Universal Approximation Theorem (Hornik 1989, Cybenko 1989)
  - *A feedforward network with a linear output layer and atleast one hidden layer with any 'squashing' activation function can approximate any Borel measurable function from one finite-dimensional space to another with any desired nonzero amount of error, provided that the network is given enough hidden units.*

Other Architectural Considerations

- *Empirical results showing that deeper networks generalize better when used to transcribe multidigit numbers.*





- *Effect of number of parameters. Deeper models tend to perform better.*

Regularization for Deep Learning

Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.

Parameter Norm Penalties

- Limiting the capacity of models by adding a parameter norm penalty $\Omega(\theta)$ to the objective function J.
- Regularized objective function:

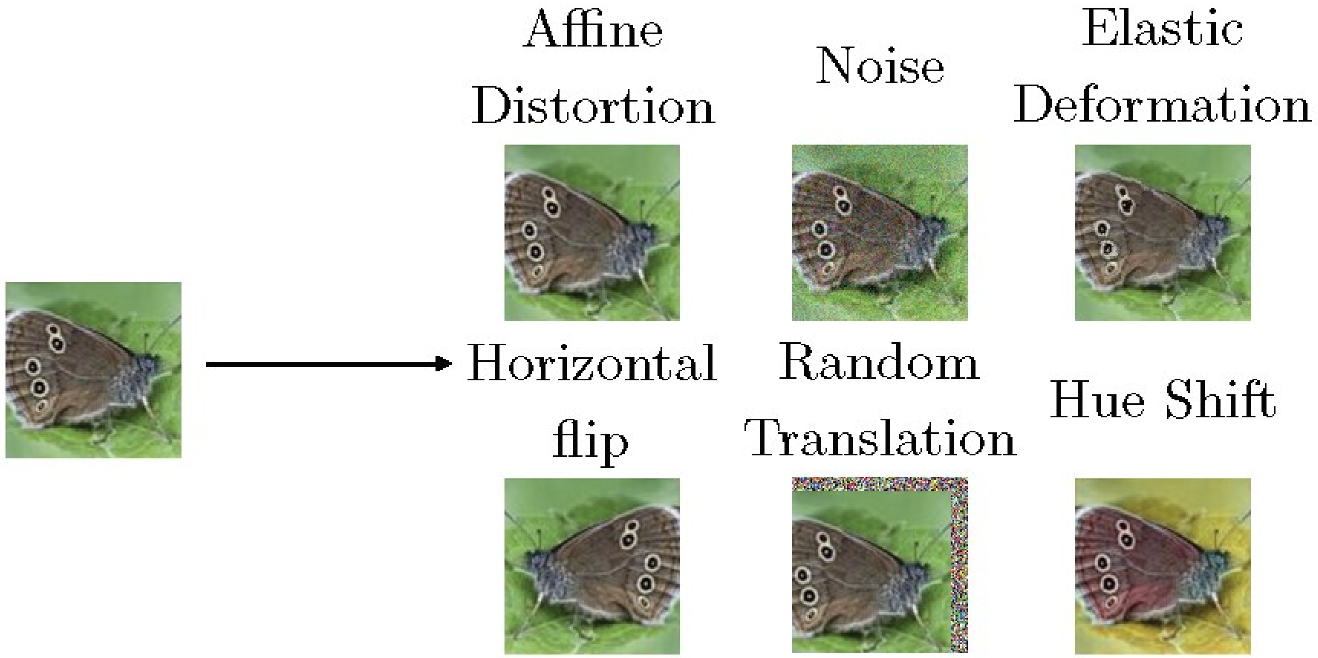
$$\bar{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta) \text{ where } \alpha \in [0, \infty)$$

- L^2 Parameter Regularization
  - Commonly known as weight decay, it drives the weights closer to the origin by adding a regularization term $\Omega(\theta) = \frac{1}{2}\|w\|_2^2$ to the objective function.
- L^1 Parameter Regularization on the model parameter w is defined as:

$$\Omega(\theta) = \|w\|_1 = \sum_i |w_i|_1$$

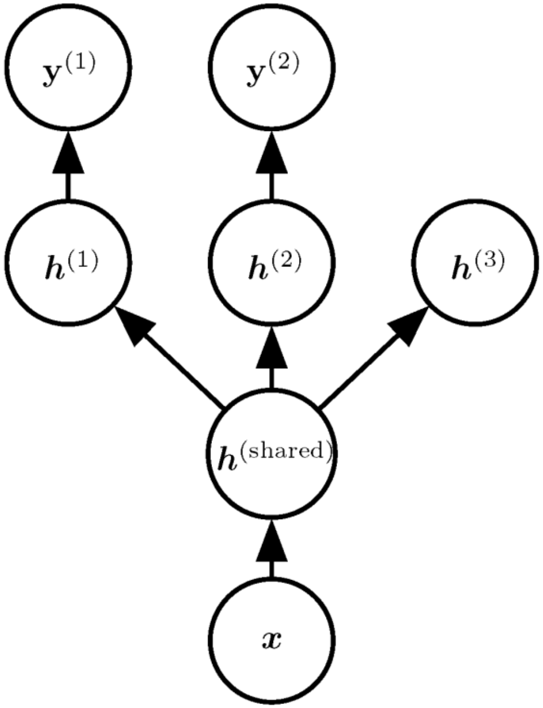
Dataset Augmentation

- The best way to make ML model generalize better is to train it on more data
- One way to augment data is by creating fake data and adding it to dataset.



Multitask Learning

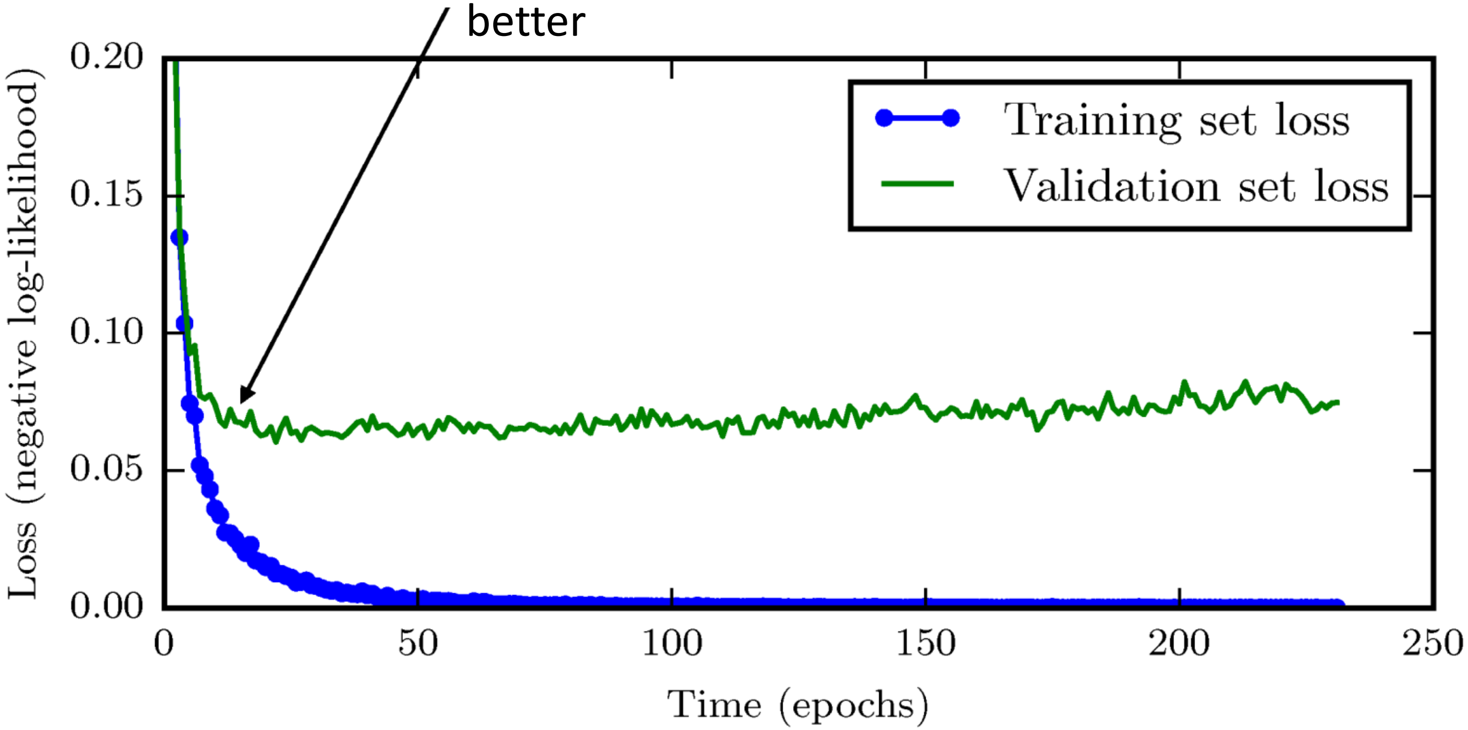
- This is a way to improve generalization by pooling the examples arising out of several tasks.
- The model can usually be divided into two kinds of parts:
  - Task specific parameters which only benefit from the examples of their task to achieve good generalization. These are the upper layer of neural network.
  - Generic parametrs, shared across all the tasks which benefit from the pooled data of all the tasks. These are the lower layers of neural network.



Early Stopping

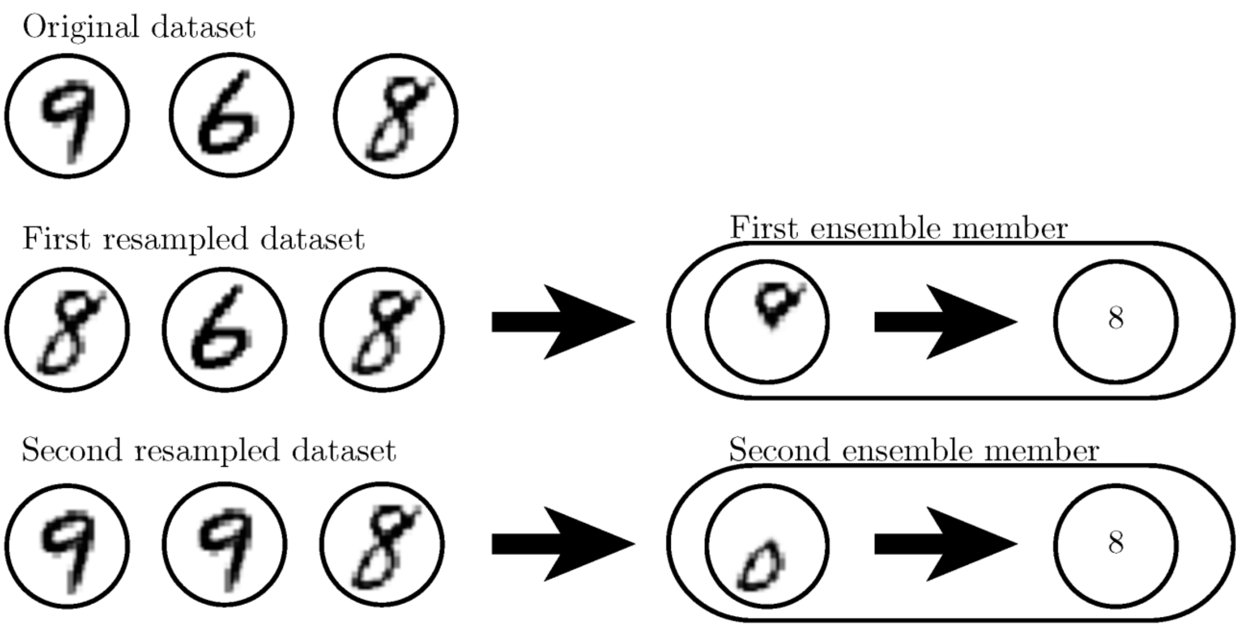
- By storing a copy of model parameters every time the error on validation set improves and return these parameters instead of the latest when training algorithm terminates.

Terminate while validation set performance is better



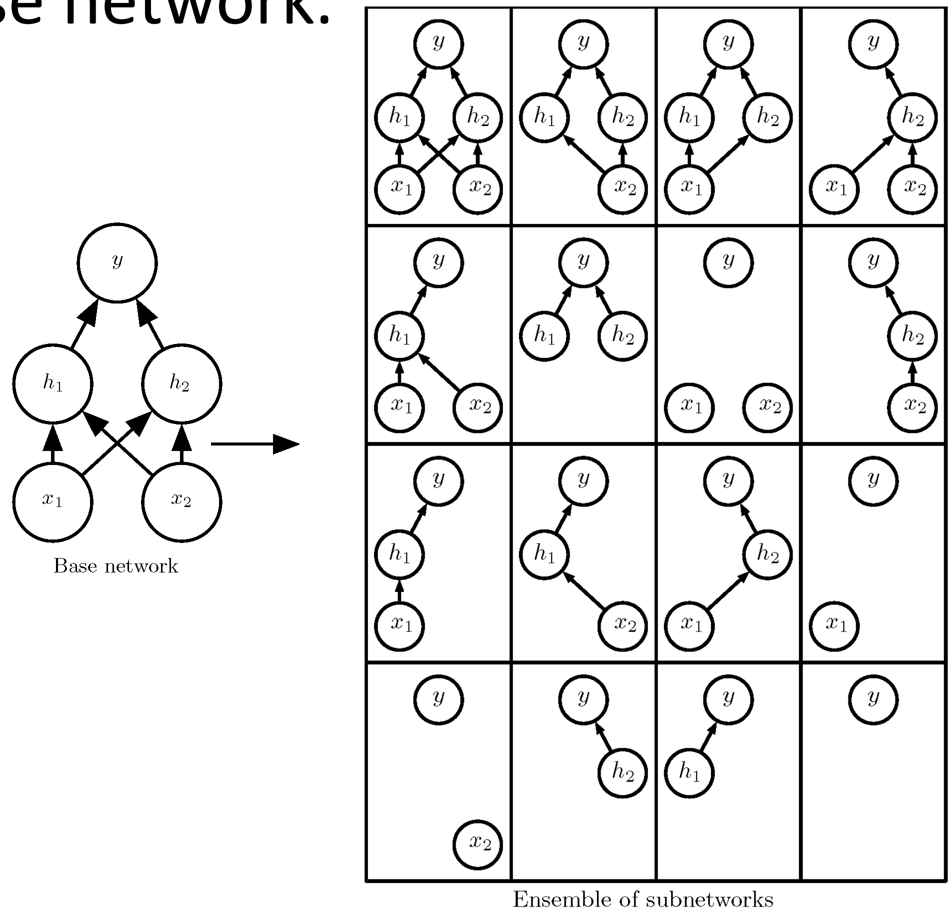
Bagging

- Bootstrap Aggregating reduces generalization error by combining several models
  - Train several different models separately, then have all the models vote on the output for test examples.

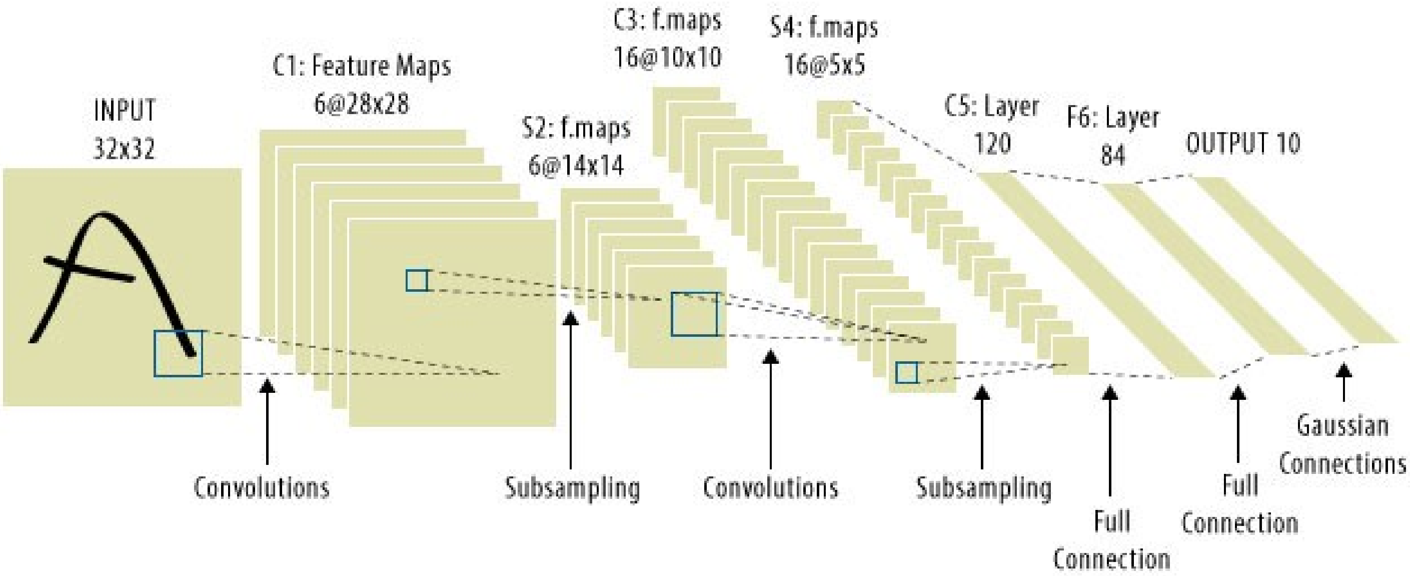


Dropout

- Dropout trains the ensemble consisting of all subnetworks that can be formed by removing non-output units from an underlying base network.



Convolution Networks



Convolution Networks CNNs

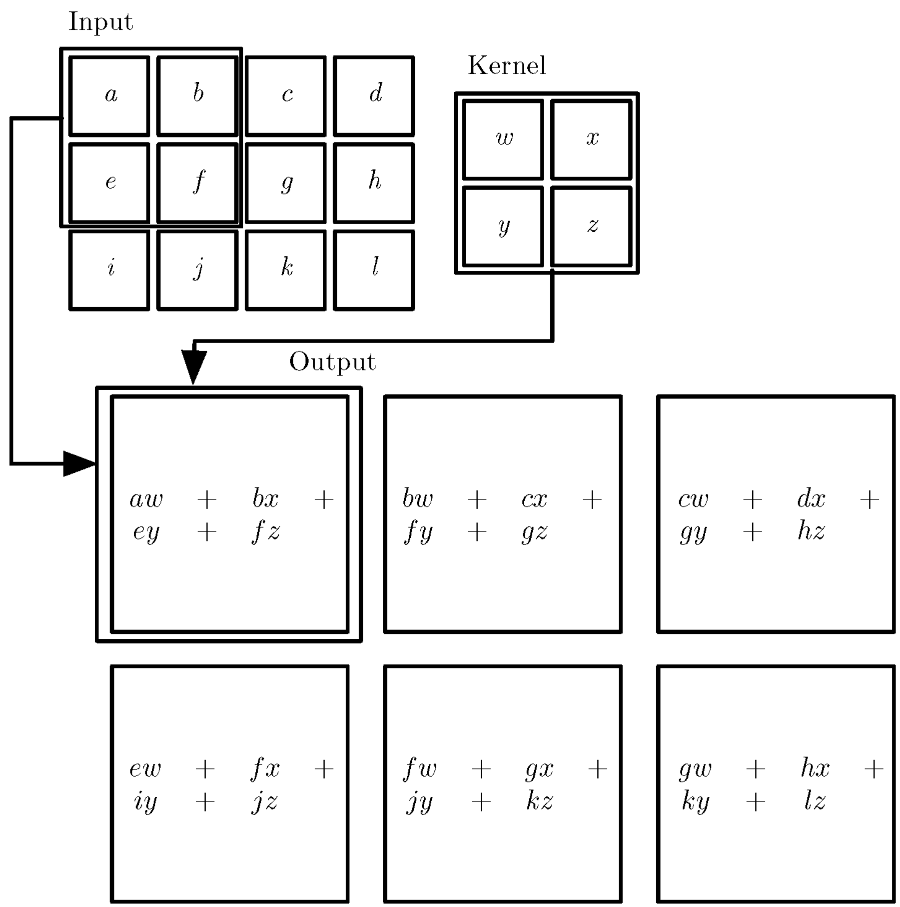
- Specialized neural network for processing data that has a known grid-like topology.
  - Time series data, 1-D grid taking samples at regular intervals
  - Image data, 2-D grid of pixels
- These network employ a mathematical operation called convolution.

- Discrete Convolution 1-D $$s(t) = (\mathrm{x} * \mathrm{w})(\mathrm{t}) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

- Discrete Convolution 2-D $$S(i,j) = (\mathrm{I} * \mathrm{K})(i,\mathrm{j}) = \sum_{m}\sum_{n} I(m,n)K(i-m,j-n)$$

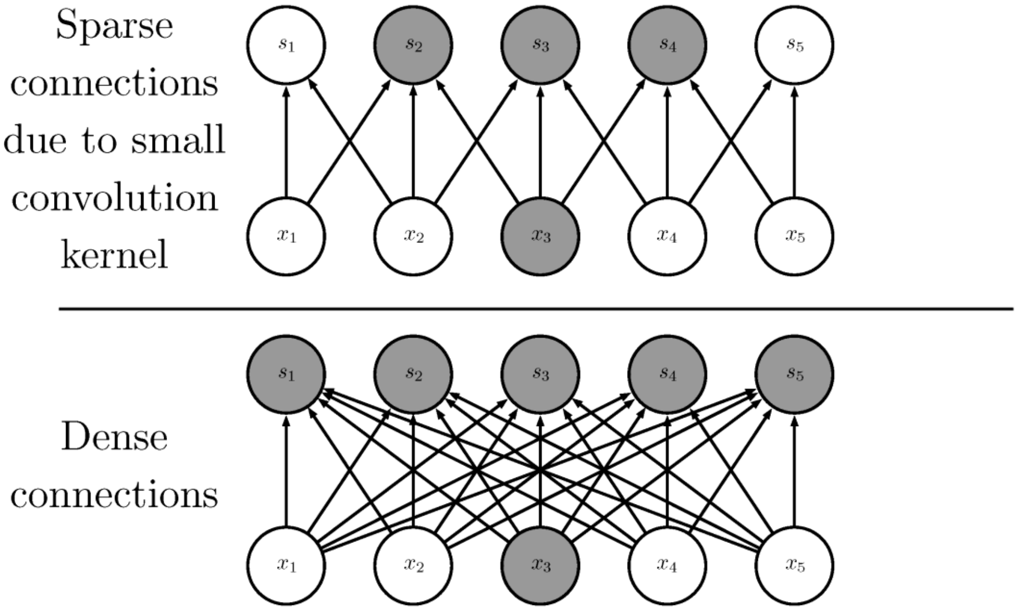
2-D Convolution

- Discrete Convolution 2-D $$S(i,j) = (\mathrm{I} * \mathrm{K})(i,\mathrm{j}) = \sum_m \sum_n I(m,n)K(i-m,j-n)$$



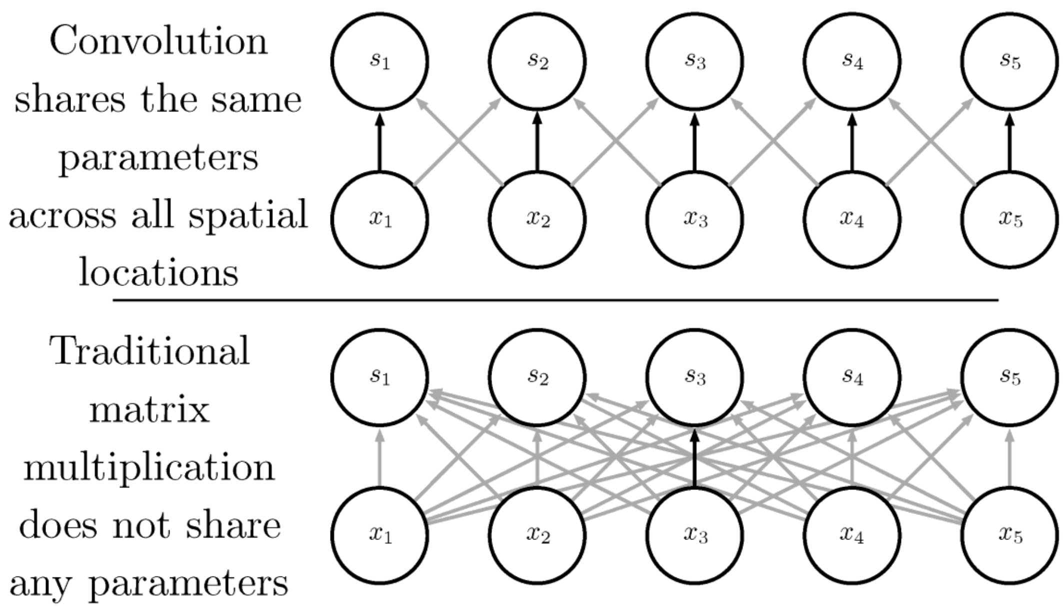
Sparse Interactions

- Traditional NN layers use matrix multiplication by a matrix of parameters with a separate parameter.
- CNNs have sparse interactions or connectivity or weights
  - Kernel size is smaller than input
  - Computing the output requires fewer operations



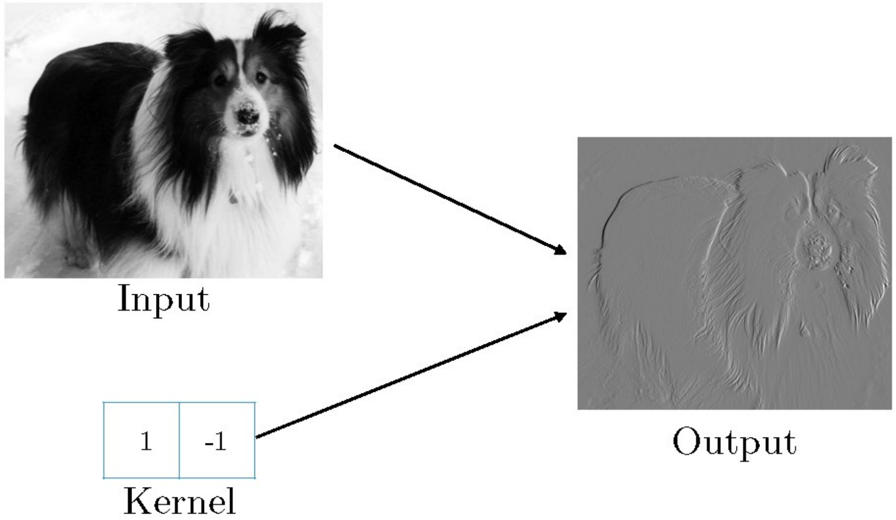
Parameter Sharing

- Using of same parameter for more than one function in a model
  - Rather than learning a separate set of parameters for every location, we learn only one set.
  - Reduces the storage requirement of the model to smaller parameters.



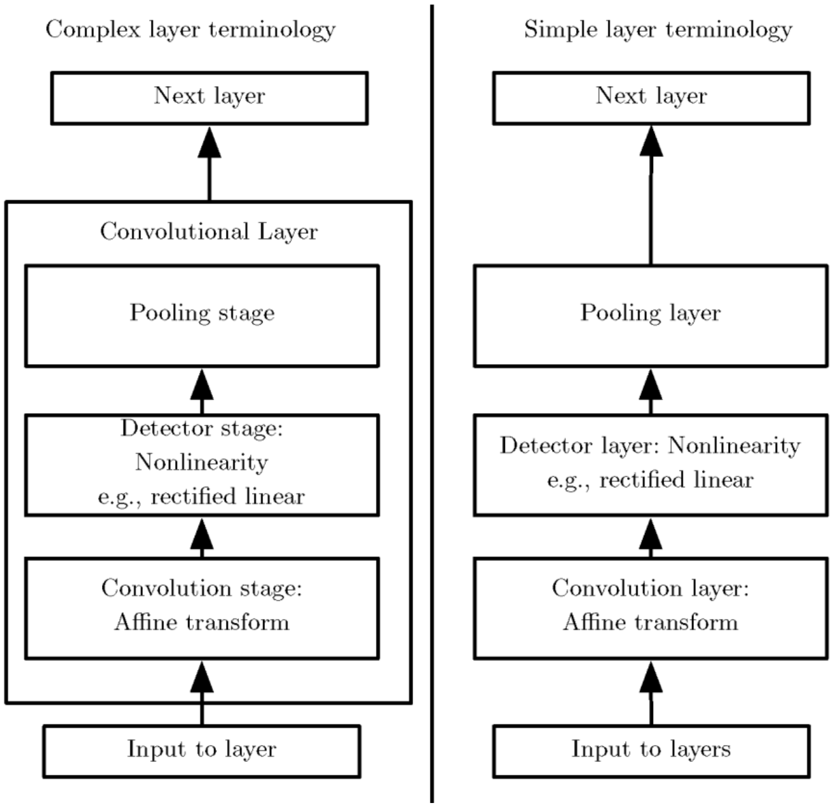
Example of Sparse Connectivity + Parameter Sharing

- Edge detection
  - The edge detection using convolution kernel containing two elements requires $319 * 280 * 3 = 267{,}960$ floating point operations
  - Same transformation using matrix multiplication would take $320 * 280 * 319 * 280 \approx 8billion$ entries in the matrix



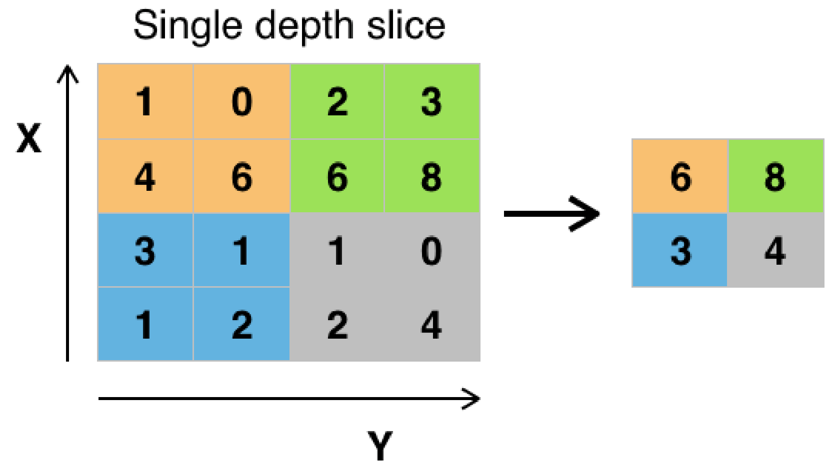
Components of Convolution Network

- Typical CNN layer has three stages
- First stage
  - Layer performs several convolutions in parallel to produce a set of linear activations
- Second Stage
  - Each linear activation is run through a nonlinear activation function such as ReLU
- Third Stage
  - Pooling Function to modify the output of the layer



Pooling

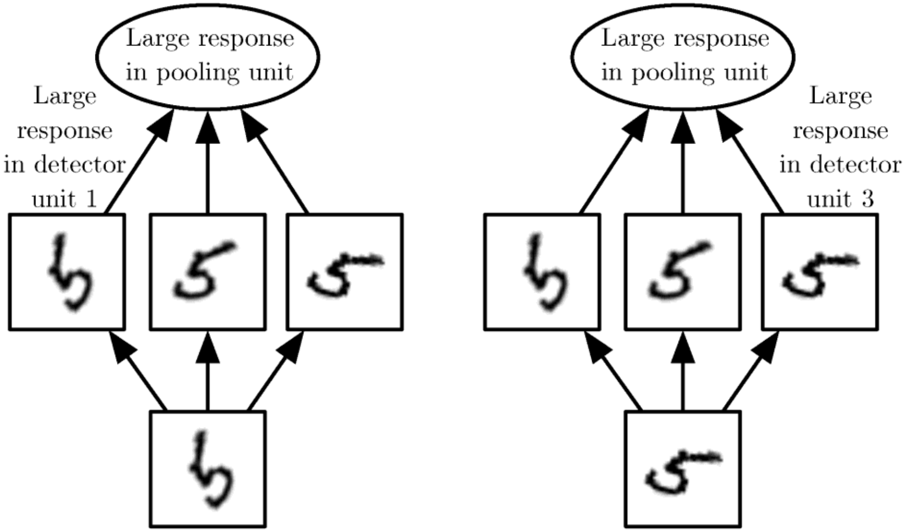
- Pooling function replaces the output of the net at a certain location with a summary statistics of the nearby outputs.
  - Max Pooling (Zhou and Chellappa 1988)
  - Average of rectangular neighborhood
  - L^2 norm of rectangular neighborhood
  - Weighted Average
- Example – Max Pooling



Example of Maxpool with a 2x2 filter and a stride of 2

Pooling Advantages

- Pooling helps to make the representation approximately invariant to small translations of the input
  - Invariance to local translation is useful property if we care more about whether some feature is present than exactly where it is
  - Pooling can be viewed as adding an infinitely strong prior that the function the layer must learn must be invariant to small translations
  - The features can learn which transformation to become invariant when pooling over the outputs of separately parameterized convolutions.



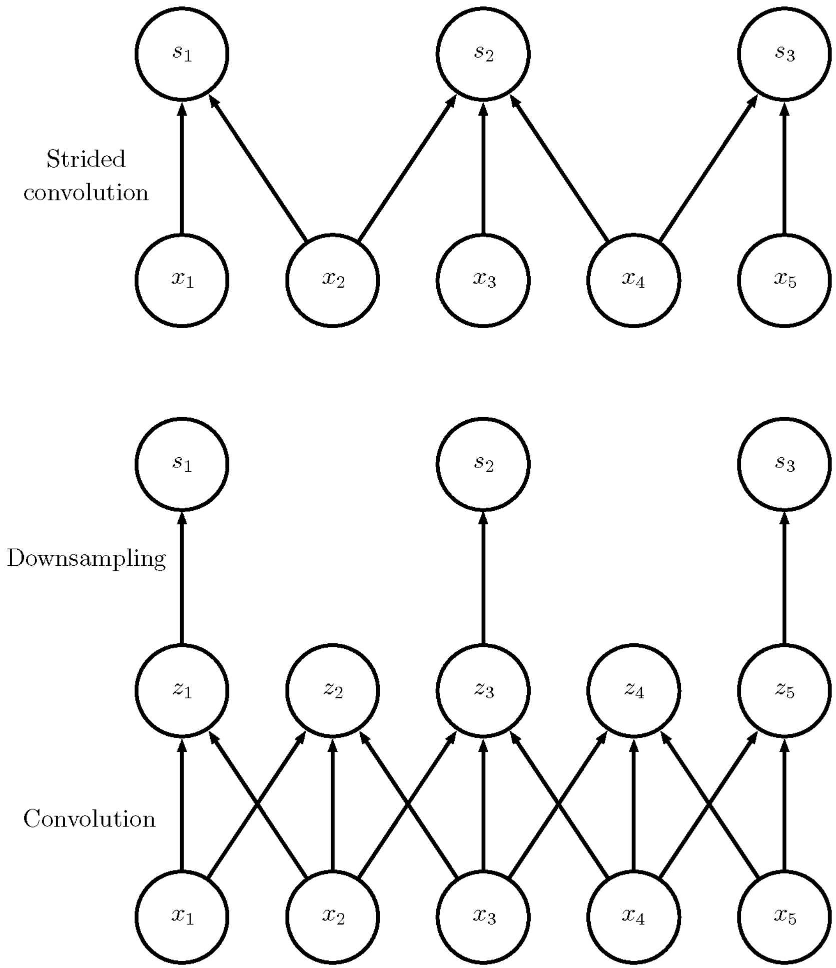
Variants of Basic Convolution Function

- Convolution in the context of NN means operation that consists of many applications of convolution in parallel
  - A single kernel extracts one type of feature. We want each layer to extract many features at many locations
- Inputs for images are usually 3-D tensors.
- To reduce computational costs we may sample only every s pixels in each direction in the output known as down-sampled convolution function

$$Z_{i,j,k} = c(K, V, s)_{i,j,k} = \sum_{l,m,n} [V_{l,(j-1)*s+m,(k-1)*s+n} K_{i,l,m,n}]$$

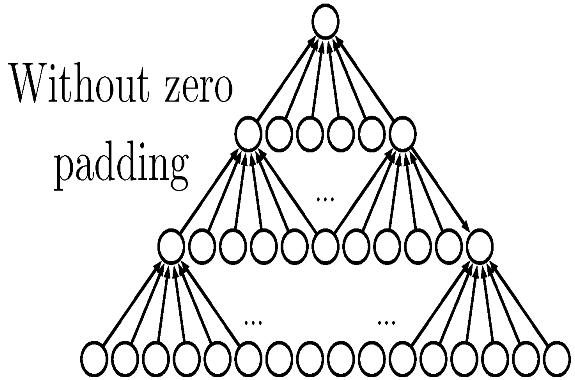
Stride of down-sampled convolution

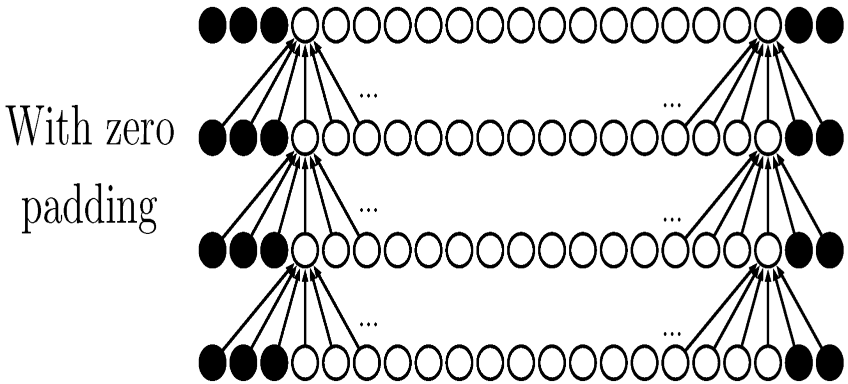
- Stride controls how the filter convolves around the input data.
- A stride of *s* means filter will convolve by shifting *s* pixels at every step.
- It is possible to define a separate stride for each direction of motion.
- Convolution with a stride greater than one pixel is mathematically equivalent to convolution followed by down-sampling



Padding

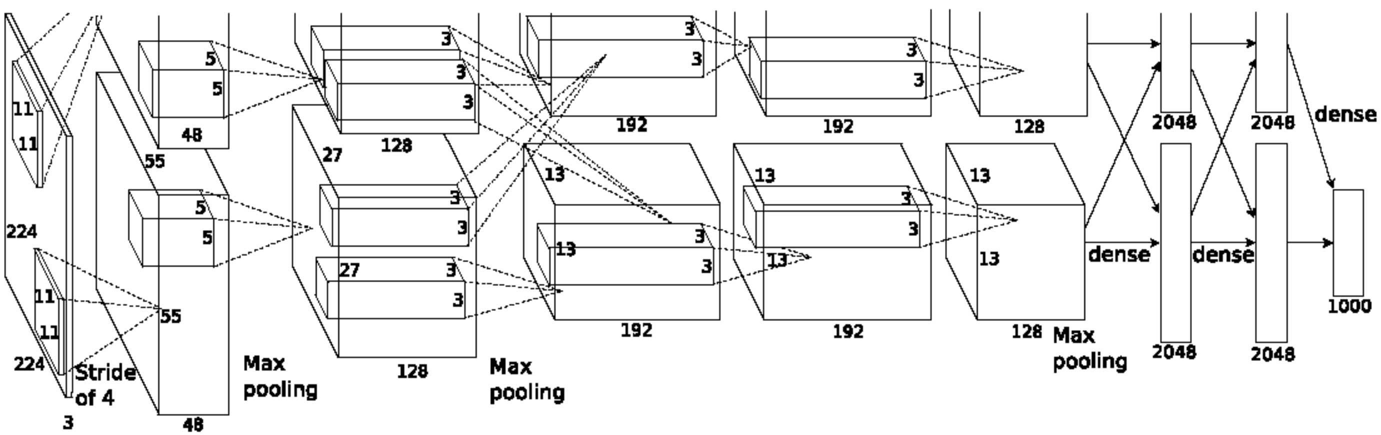
- The width of representation shrinks by one pixel less than the kernel width at each layer
- Zero padding the input allows us to control the kernel width and the size of the output independently
- Three cases of zero padding (MATLAB terminology)
  - Valid convolution
  - Same convolution
  - Full convolution





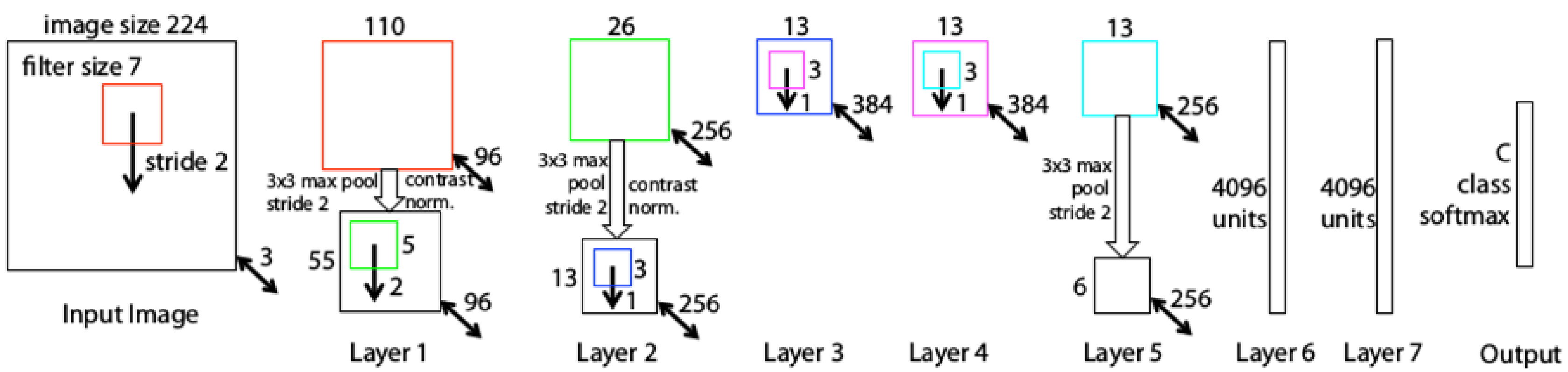
Major Architectures

- AlexNet (Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton)
  - Trained on ImageNet data
    - over 15 million annotated images
    - 22,000 categories
  - ReLU for the non-linearity function
  - Data Augmentation using translation, reflections, patch extractions
  - Dropout layers for avoiding overfitting
  - Trained using batch stochastic gradient descent
  - Trained using two GTX 580 GPUs over five to six days



Major Architectures

- ZF Net (Matthew Zeiler and Rob Fergus)
  - Trained on
    - 1.3 million annotated images
  - Filter size of 7x7 pixels
  - ReLU for the activation function
  - Trained using batch stochastic gradient descent
  - Trained using oneGTX 580 GPUs over five to six days
  - Developed a visualization technique named Deconvolutional Network



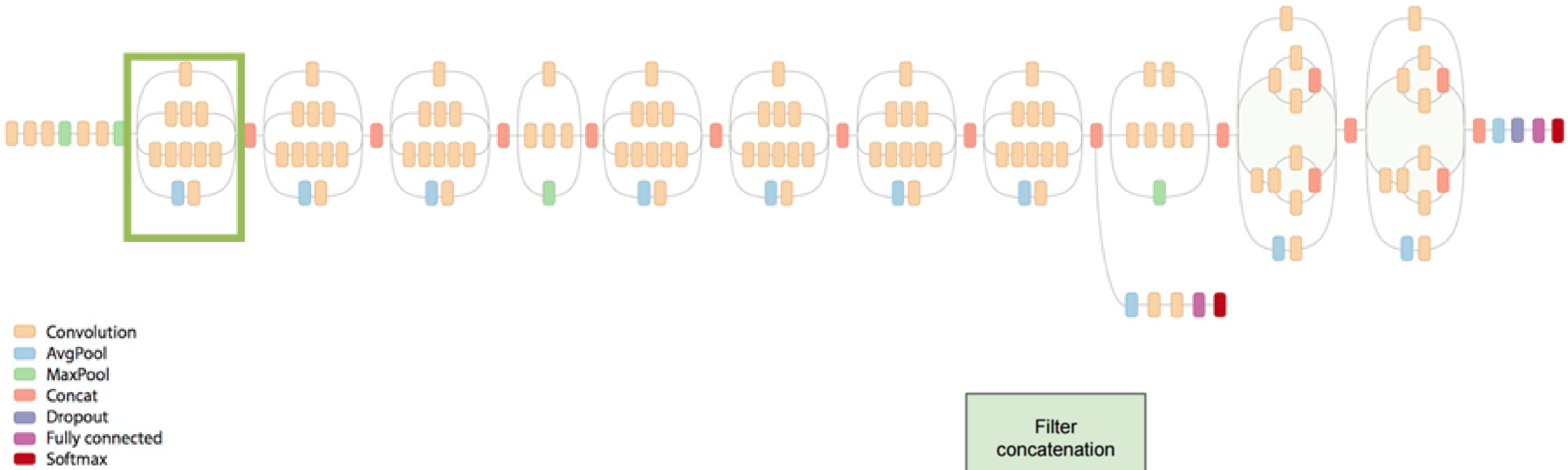
ZF Net Architecture

Major Architectures

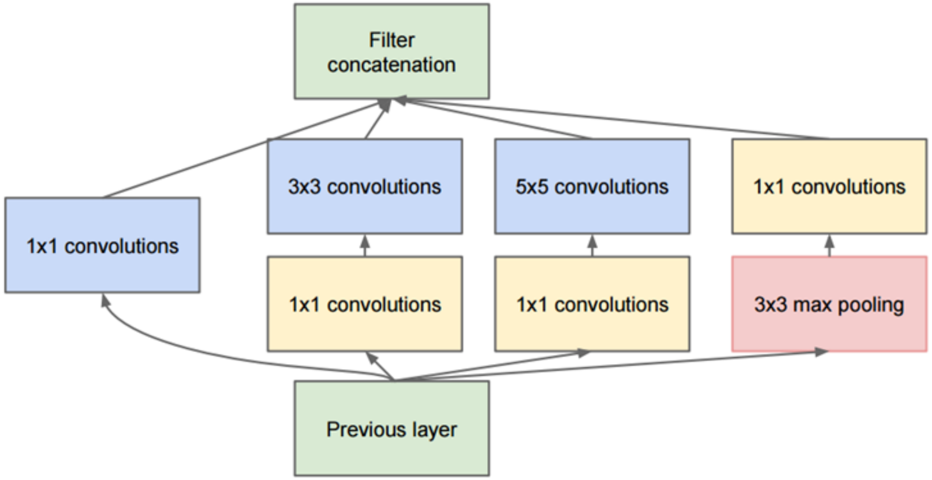
- VGG Net (Karen Simonyan and Andrew Zisserman)
  - Filter size of 3x3 pixels with two convolution layers
  - Stride and Padding of one
  - 19 weight layers
  - Scale jittering as data augmentation method
  - ReLU layer after each convolution layer
  - Trained using batch stochastic gradient descent

Major Architectures

- GoogLeNet
  - Introduced Inception Module



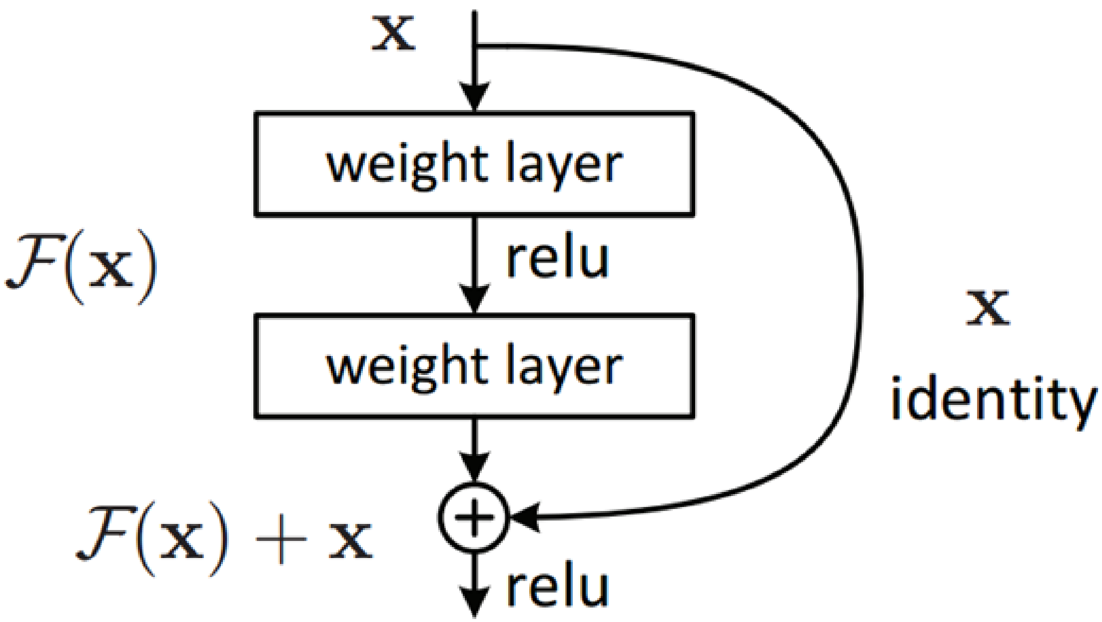
Green box shows parallel region of GoogLeNet



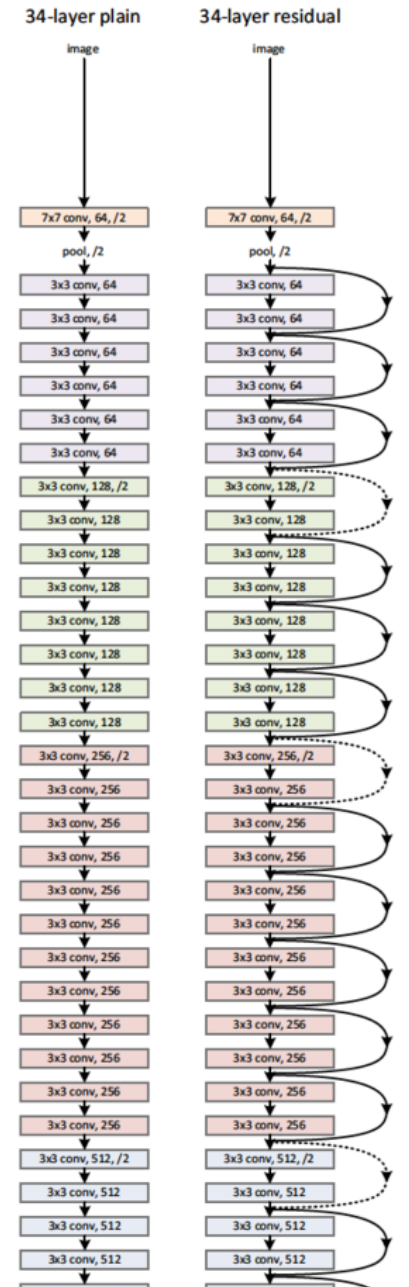
Full Inception module

Major Architectures

- Microsoft ResNet
  - Ultra Deep – Yann LeCun
  - 152 layers
  - Residul Blocks



A residual block



Deep Learning in Computer Vision

- Most Deep Learning for computer vision is used for object recognition or detection of some form
- Preprocessing
  - Requires pre-processing so that image pixels lie within same reasonable range
  - Scaling or Cropping of images
    - Some CNNs adjust pooling to accommodate varying sizes
    - Some CNNs adjust output to match the changes in incoming sizes
  - Global Contrast Normalization
    - Prevents images from having varying amount of contrast by subtracting mean from each image

$$X'_{i,j,k} = s \frac{X_{i,j,k} - \bar{X}}{\max\{\epsilon, \sqrt{\lambda + \frac{1}{3rc} \sum_{i=1}^{r} \sum_{j=1}^{c} \sum_{k=1}^{3} (X_{i,j,k} - \bar{X})^2}}$$

Conclusion

- Deep Learning is relevant for Supervised, Unsupervised and Reinforcement Learning.
- Advances in computational capabilities and GPU based architecture has given rise to deeper networks with better classification performance than humans
- Humans learn to actively perceive patterns by sequentially directing attention to relevant parts of the available data and in near future deep NNs will do so too.
- Many future deep NNs will also take into account the energy costs of activating neurons and minimize such computational costs.