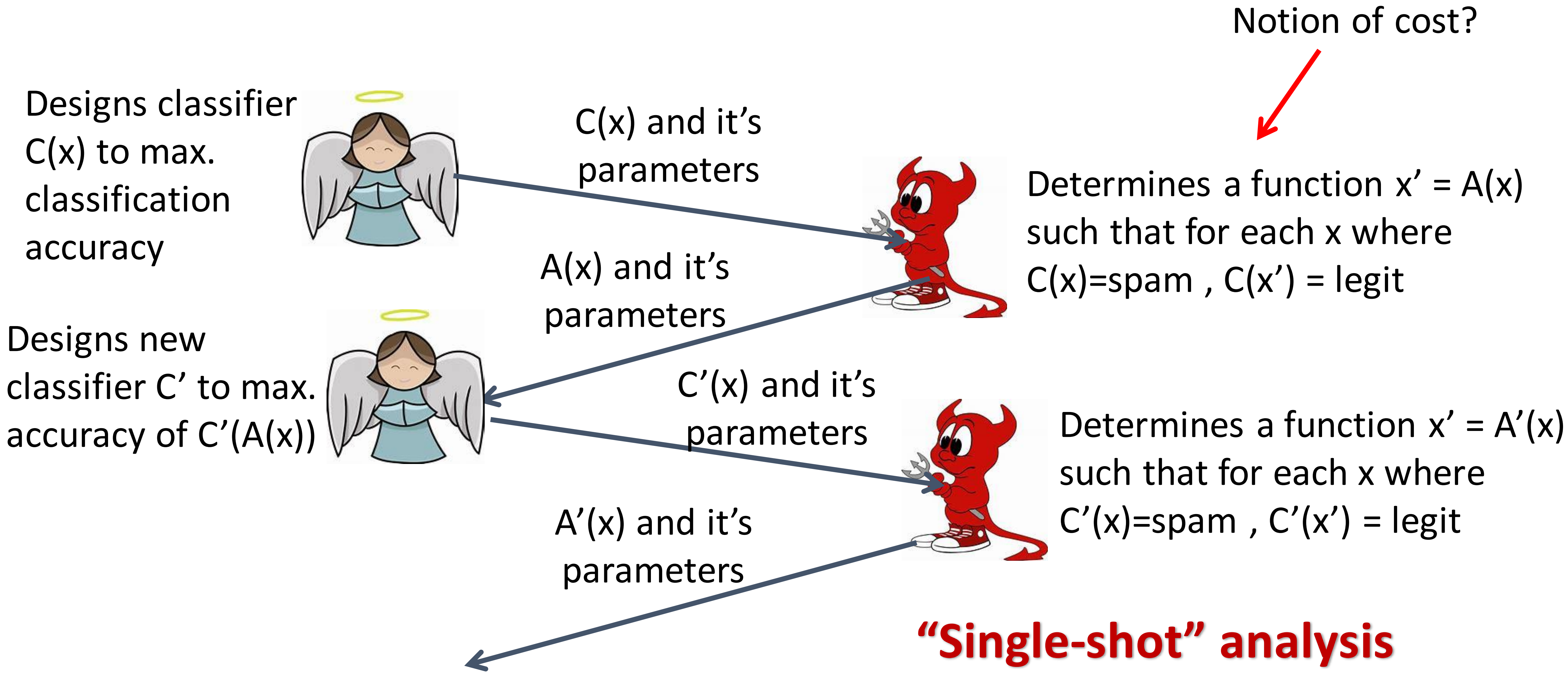
Lecture 5: Adversarial Attacks on Spam Filter + Intro To DL

Siddharth Garg

sg175@nyu.edu

“Game” Between Attacker and Defender



Notion of cost?

Designs classifier C(x) to max. classification accuracy

C(x) and it’s parameters

Determines a function x’ = A(x) such that for each x where C(x)=spam , C(x’) = legit

A(x) and it’s parameters

Designs new classifier C’ to max. accuracy of C’(A(x))

C’(x) and it’s parameters

Determines a function x’ = A’(x) such that for each x where C’(x)=spam , C’(x’) = legit

A’(x) and it’s parameters

“Single-shot” analysis

Do best response dynamics converge to a NE?

Attacker's Utility

- Attacker changes x to x'
  - Incurs a cost $c(x,x') = \sum_i c_i(x_i, x_i')$ for making modifications
  - Why do we need to account for the attacker's costs?
  - Example: #words added, #words modified etc.

- Receives a utility $U_A(y_C, y) \in \{-1,0,1\}$ where
  - y_C: predicted class by classifier *C*
  - y : true class

Implies that attacker only modifies spam emails

$U_A(y_C = legit, y = legit)$ =0

$U_A(y_C = legit, y = spam)$ =1

$U_A(y_C = spam, y = legit)$ =0

$U_A(y_C = spam, y = spam)$ =0

Attacker’s Strategy

- Assume a spam message x classified by *C* as spam. Then:

$$\frac{P\{spam \mid x\}}{P\{legit \mid x\}} = \frac{P\{x \mid spam\} * P\{spam\}}{P\{x \mid legit\} * P\{legit\}} > 1$$

$$\Rightarrow \log\left(\frac{P\{spam \mid x\}}{P\{legit \mid x\}}\right) = \log\left(\frac{P\{x \mid spam\}}{P\{x \mid legit\}}\right) + \log\left(\overset{\text{Assume=0}}{\cancel{\frac{P\{spam\}}{P\{legit\}}}}\right) > 0$$

$$\Rightarrow \log\left(\frac{P\{x \mid spam\}}{P\{x \mid legit\}}\right) = \sum_i \underbrace{\log\left(\frac{P\{x_i \mid spam\}}{P\{x_i \mid legit\}}\right)}_{LO(x_i)} > 0$$

Attacker’s Strategy

$$\Rightarrow \log\left(\frac{P\{x \mid spam\}}{P\{x \mid legit\}}\right) = \sum_i \underbrace{\log\left(\frac{P\{x_i \mid spam\}}{P\{x_i \mid legit\}}\right)}_{LO(x_i)} > 0$$

- The attacker wants to change x to x’ such that x’ is classified as legit

$$\sum_i \log\left(\frac{P\{x_i' \mid spam\}}{P\{x_i' \mid legit\}}\right) = \sum_i LO(x_i') < 0$$

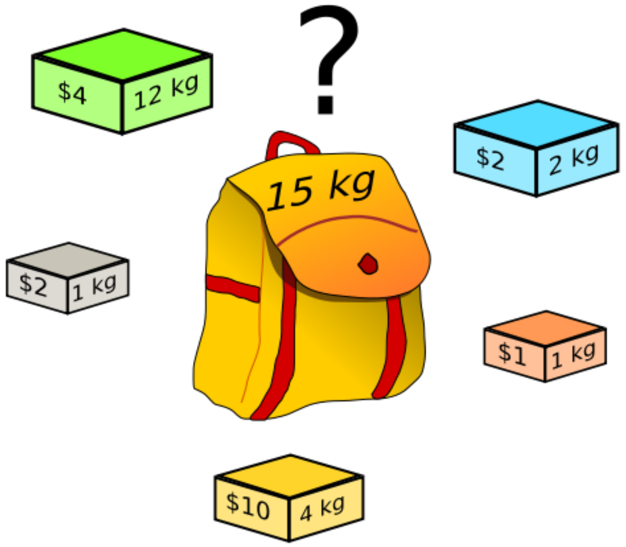
$$\Delta = \sum_i LO(x_i)$$

Desired total reduction in sum-LO

$$\delta_i = \max(LO(x_i) - LO(1 - x_i), 0)$$

Reduction obtained by adding/removing term *i*

Reduction to Knapsack Problem



- Knapsack of maximum weight 15 Kg
- N items, each of have a weight and reward
- Which items to put in knapsack such that:
  1. Weight of bag is less than 15 Kgs
  2. Reward is maximized

Attacker's Problem

- N terms, each term has "weight" δ_i and "cost" c_i if it is added/removed
- We need to add/remove enough terms such that the net weight exceeds
- While minimizing cost

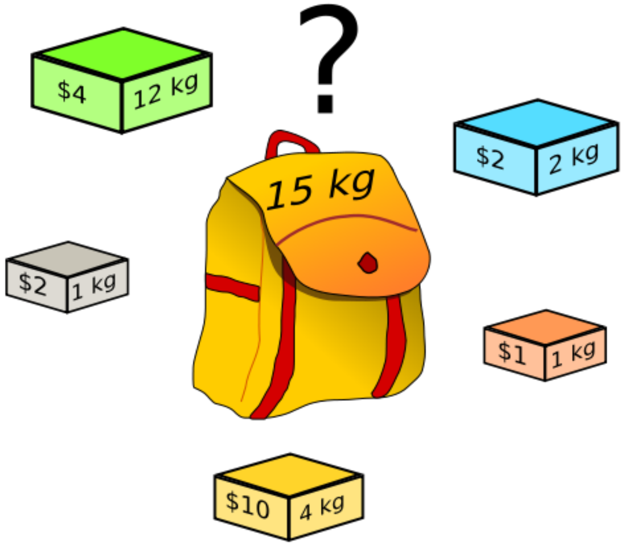
What if all terms cost the same?

Reduction to Knapsack Problem

Attacker's Problem

- N terms, each term has "weight" δ_i and "cost" c_i if it is added/removed
- We need to add/remove enough terms such that the net weight exceeds
- While minimizing cost

What if all terms cost the same?



How Does the Classifier Respond?

Classifier computes

$$\frac{P_A\{spam \mid x'\}}{P_A\{legit \mid x'\}} = \frac{P_A\{x' \mid spam\} * P_A\{spam\}}{P_A\{x' \mid legit\} * P_A\{legit\}}$$

Since the adversary only modifies the feature vector for spam emails, P_A\{x’|spam\} is the only term that changes

$$P_A\{x' \mid spam\} = \sum_{x} P_A\{x' \mid x, spam\} P\{x \mid spam\} = \sum_{x:x'=A(x)} P\{x \mid spam\}$$

Sum over all possible feature vectors

Sum over all x which when modified by adversary yield x’

Relative cost of classifying legit emails as spam (FPs)

Results

ADD WORDS (AW): Unit cost per word
ADD LENGTH (AL): Cost prop. To word length
SYNONYM (SYN): Unit cost per word

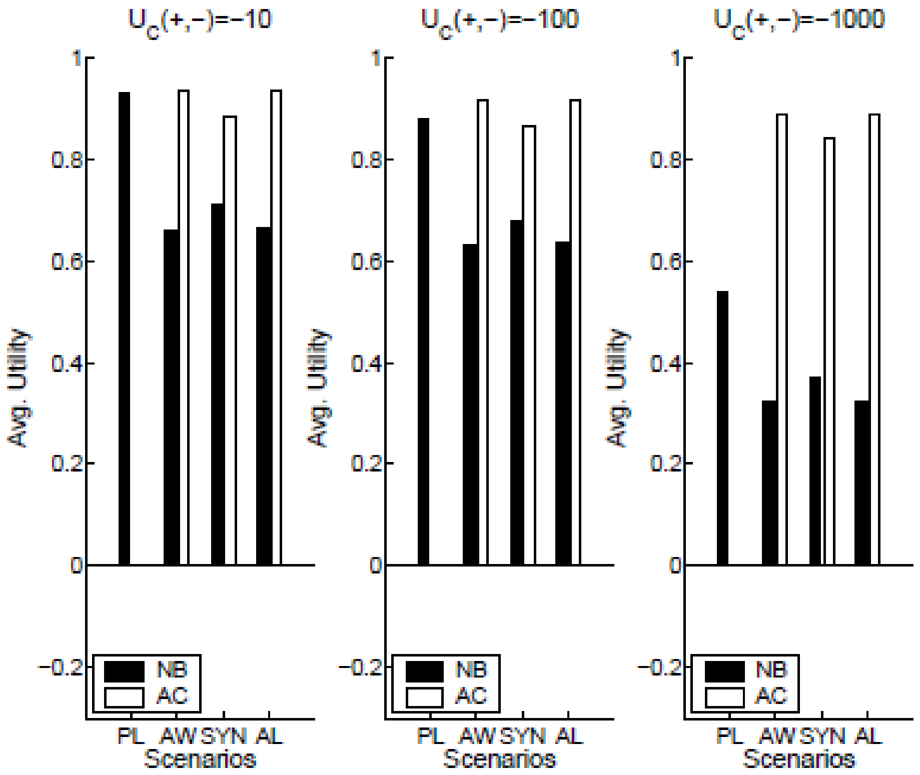
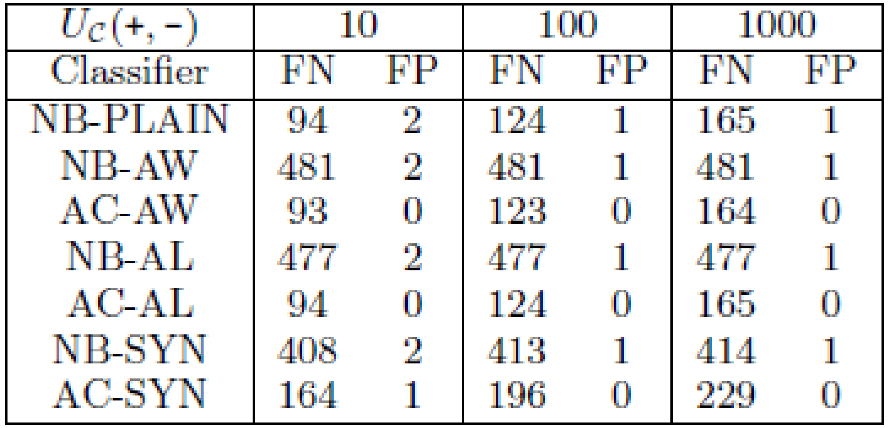


Figure 1: Utility results on the Ling-Spam dataset for different values of $U_C(+,-)$.

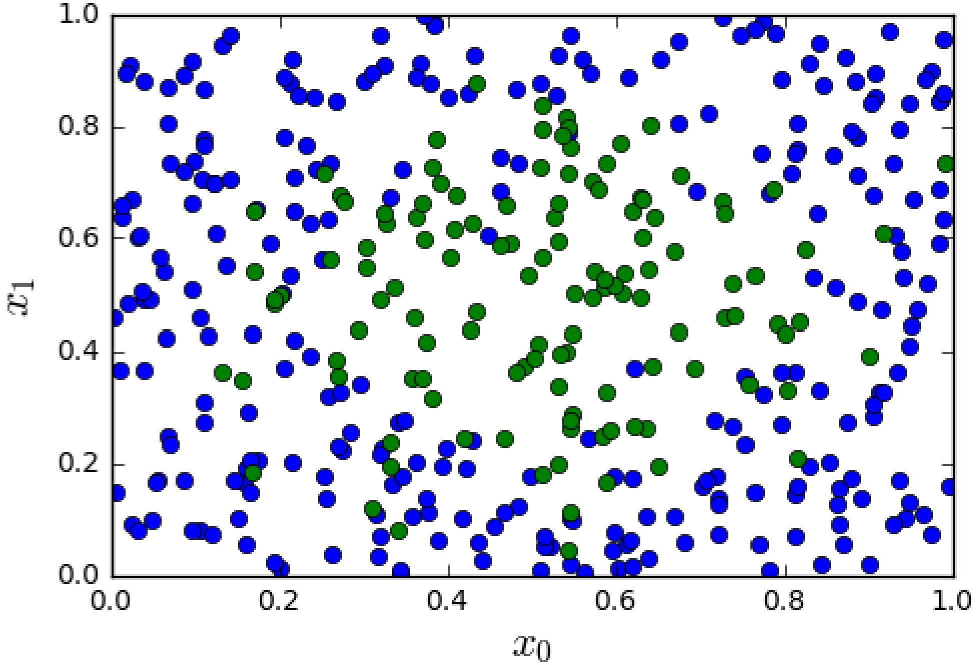


$U_C(+,-)$	10		100		1000	
Classifier	FN	FP	FN	FP	FN	FP
NB-PLAIN	94	2	124	1	165	1
NB-AW	481	2	481	1	481	1
AC-AW	93	0	123	0	164	0
NB-AL	477	2	477	1	477	1
AC-AL	94	0	124	0	165	0
NB-SYN	408	2	413	1	414	1
AC-SYN	164	1	196	0	229	0

Table 3: False positives and false negatives for naive Bayes and the adversary-aware classifier on the Ling-Spam dataset. The total number of positives in this dataset is 481, and the total number of negatives is 2412.

NB: Naïve Bayes, AC: Adversarially Trained

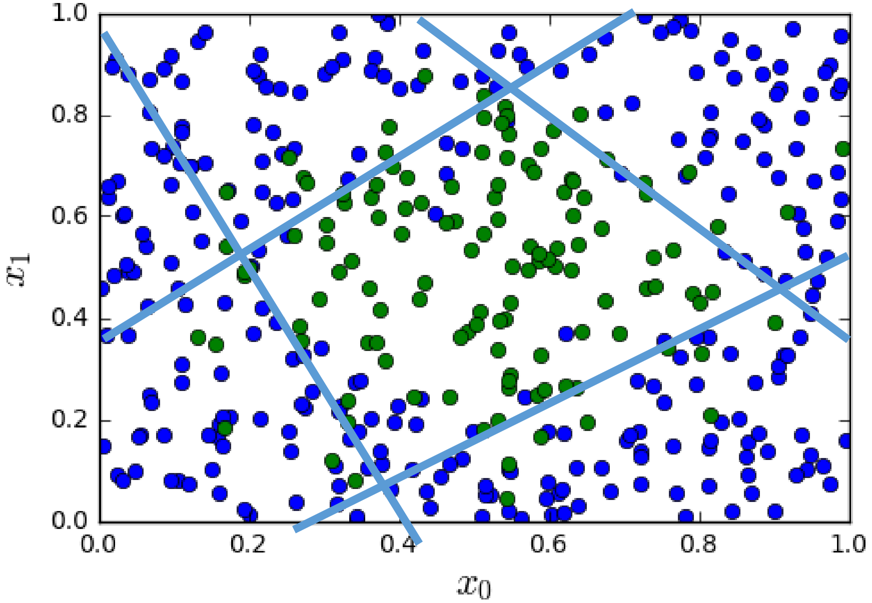
Most Datasets are not Linearly Separable



- Consider simple synthetic data
  - See figure to the left
  - 2D features
  - Binary class label
- Not separated linearly

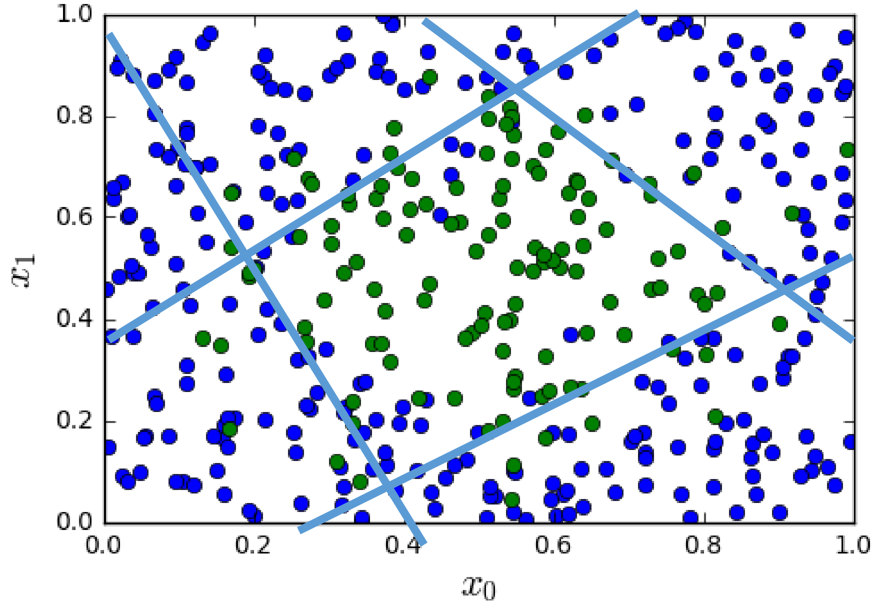
All code in https://github.com/sdrangan/introml/blob/master/neural/synthetic.ipynb

From Linear to Nonlinear



- Idea: Build nonlinear region from linear decisions
- Possible form for a classifier:
    - Step 1: Classify into small number of linear regions
    - Step 2: Predict class label from step 1 decisions

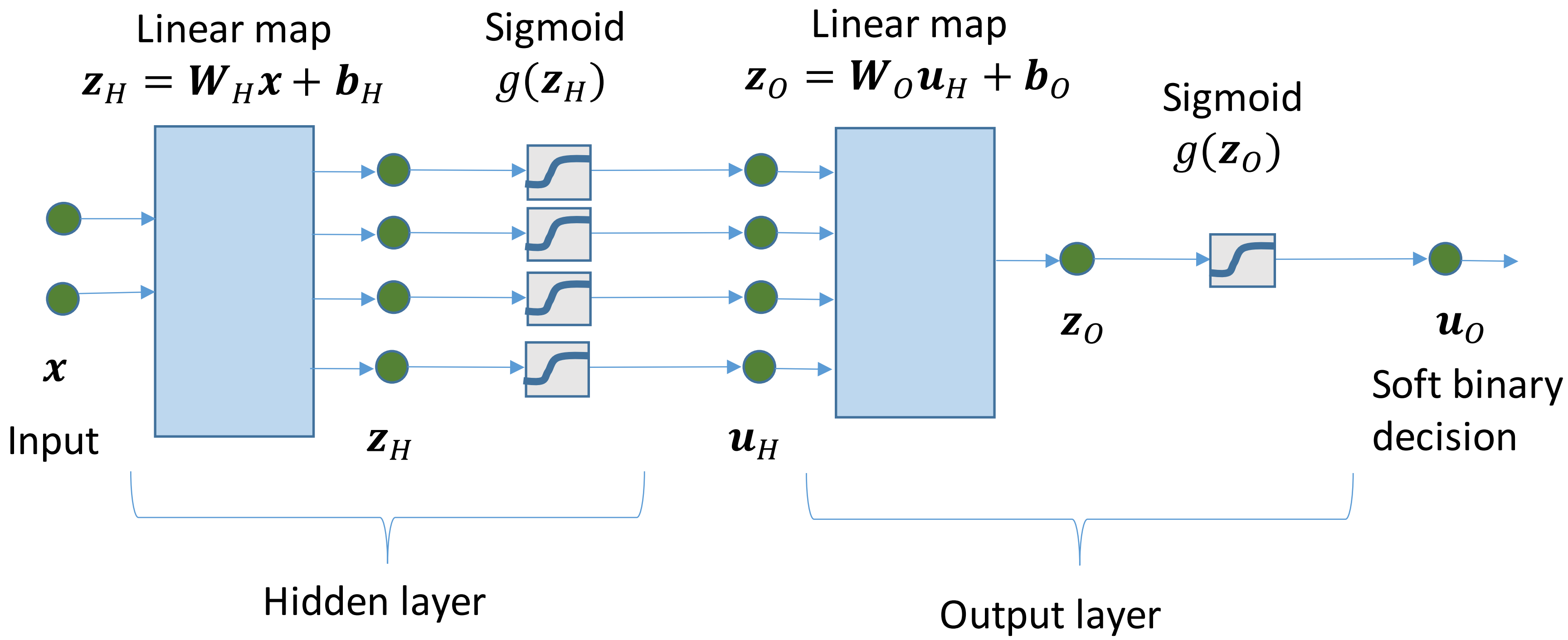
A Possible Two Stage Classifier



- Input sample: $\boldsymbol{x} = (x_1, x_2)^T$
- First step: Hidden layer
  - Take $N_H = 4$ linear discriminants
    $$z_{H,1} = \boldsymbol{w}_{H,1}^T x + b_{H,1}$$
    $$\vdots$$
    $$z_{H,N_H} = \boldsymbol{w}_{H,M}^T x + b_{H,M}$$
  - Make a soft decision on each linear region
    $u_{H,m} = g\left(z_{H,m}\right) = 1/(1 + e^{-z_{H,m}})$
- Second step: Output layer
  - Linear step $z_O = w_O^T u_H + b_O$
  - Soft decision: $u_O = g(z_O)$

Model Block Diagram

- Hidden layer: $\boldsymbol{z}_H = \boldsymbol{W}_H\boldsymbol{x} + \boldsymbol{b}_H, \quad \boldsymbol{u}_H = g(\boldsymbol{z}_H)$
- Output layer: $\boldsymbol{z}_O = \boldsymbol{W}_O\boldsymbol{u}_H + \boldsymbol{b}_O,\ u_O = g(\boldsymbol{z}_O)$



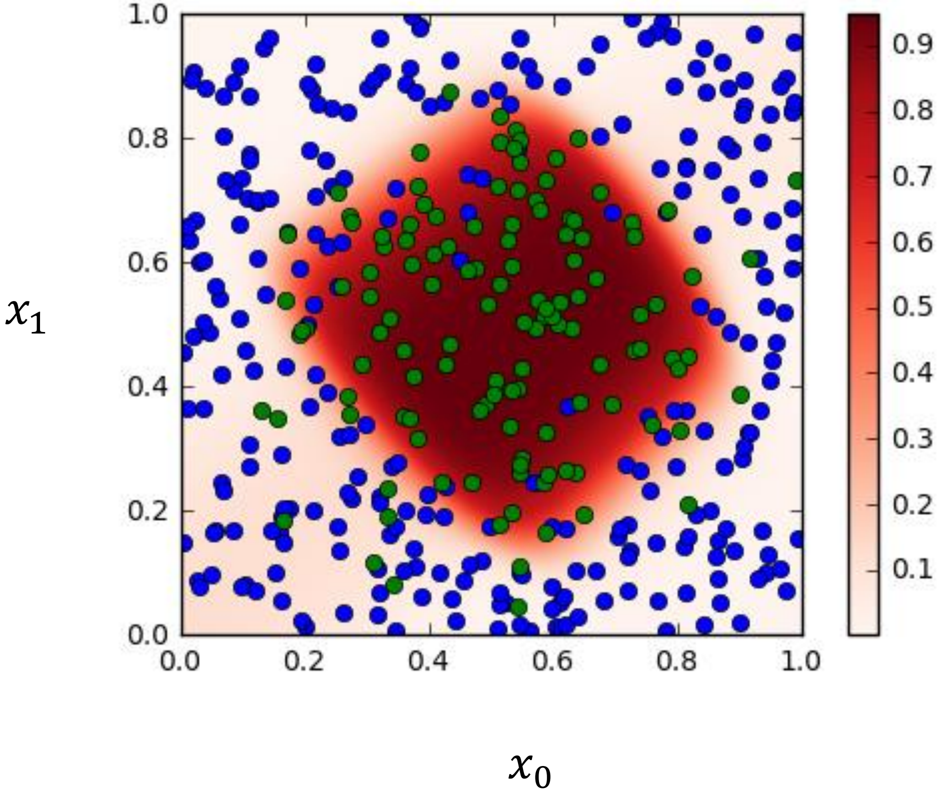
Training the Model

- Model in matrix form:
  - Hidden layer: $\boldsymbol{z}_H = \boldsymbol{W}_H \boldsymbol{x} + \boldsymbol{b}_H, \quad \boldsymbol{u}_H = g(\boldsymbol{z}_H)$
  - Output layer: $z_O = \boldsymbol{W}_O \boldsymbol{u}_H + \boldsymbol{b}_O,\ u_O = g(z_O)$
- $z_O = F(\boldsymbol{x}, \theta)$: Linear output from final stage
  - Parameters: $\theta = (\boldsymbol{W}_H, \boldsymbol{W}_O, b_H, b_O)$
- Get training data $(\boldsymbol{x}_i, y_i), i = 1, \ldots, N$
- Define loss function: $L(\theta) \coloneqq \sum_{i=1}^{N} \ln[1 + e^{-y_i z_{O,i}}]$, $z_{O,i} = F(x_i,, \theta)$ (logistic loss)
- Pick parameters to minimize loss:

$$\hat{\theta} = \arg\min_{\theta} L(\theta)$$

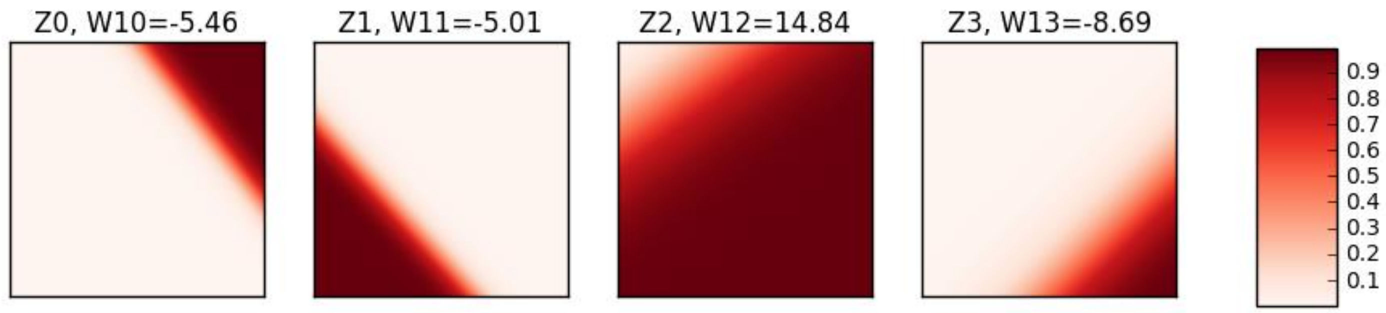
  - Will discuss how to do this minimization later

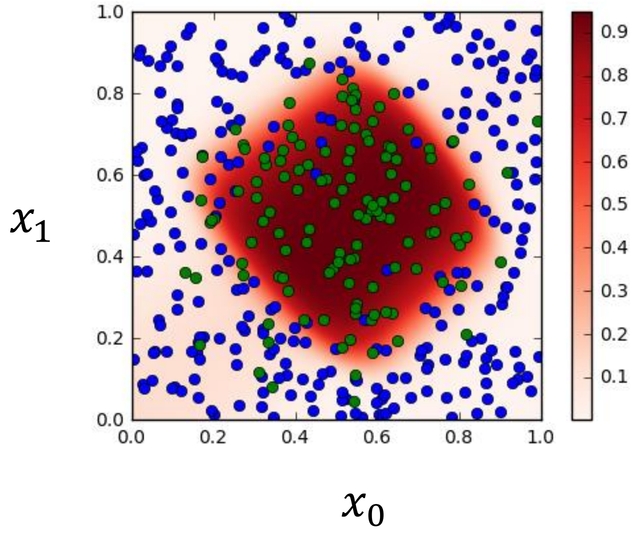
Results



- Neural network finds a nonlinear region
- Plot shows:
  - Blue circles: Negative samples
  - Greed circles: Positive samples
  - Red color: Classifier soft probability $g(z_O)$

Visualizing the Hidden Layer Weights

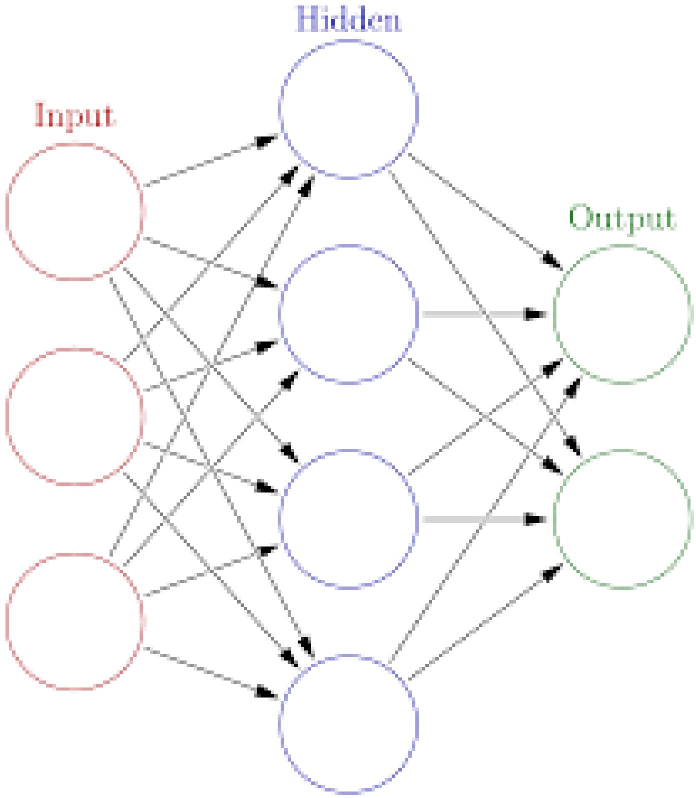




- Hidden weights finds lower layer features

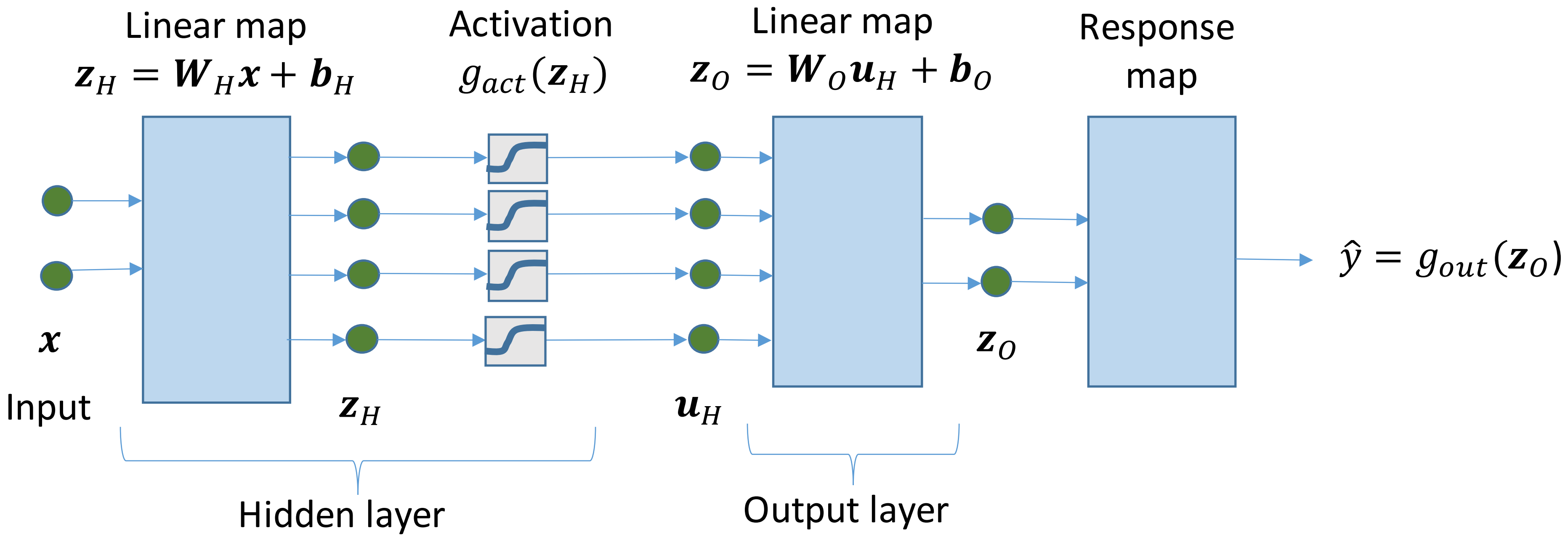
General Structure

- Input: $\boldsymbol{x} = (x_1, \cdots, x_d)$
  - d = number of features
- Hidden layer:
  - Linear transform: $\boldsymbol{z}_H = \boldsymbol{W}_H \boldsymbol{x} + \boldsymbol{b}_H$
  - Soft decision: $\boldsymbol{u}_H = g(\boldsymbol{z}_H)$
  - Dimension: M hidden units
- Output layer:
  - Linear transform: $\boldsymbol{z}_O = \boldsymbol{W}_O \boldsymbol{u}_H + \boldsymbol{b}_O$
  - Dimension: K = number of classes / outputs
- Can be used for classification or regression



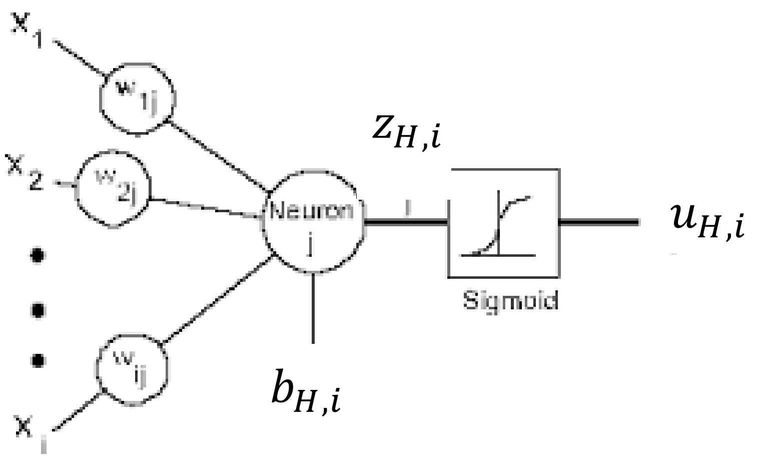
General Neural Net Block Diagram

- Hidden layer: $\boldsymbol{z}_H = \boldsymbol{W}_H\boldsymbol{x} + \boldsymbol{b}_H, \quad \boldsymbol{u}_H = g_{act}(\boldsymbol{z}_H)$
- Output layer: $\boldsymbol{z}_O = \boldsymbol{W}_O\boldsymbol{u}_H + \boldsymbol{b}_O$
- Response map: $\hat{y} = g_{out}(\boldsymbol{z}_O)$


Linear map
$\boldsymbol{z}_H = \boldsymbol{W}_H\boldsymbol{x} + \boldsymbol{b}_H$
Activation
$g_{act}(\boldsymbol{z}_H)$
Linear map
$\boldsymbol{z}_O = \boldsymbol{W}_O\boldsymbol{u}_H + \boldsymbol{b}_O$
Response
map
$\hat{y} = g_{out}(\boldsymbol{z}_O)$
$\boldsymbol{x}$
Input
$\boldsymbol{z}_H$
$\boldsymbol{u}_H$
$\boldsymbol{z}_O$
Hidden layer
Output layer

Terminology

- Hidden variables: the variables $\boldsymbol{z}_H, \boldsymbol{u}_H$
  - These are not directly observed
- Hidden units: The functions that compute:
  - $\mathrm{z}_{H,i} = \sum_j W_{H,ij} x_j + b_{H,i}$, $u_{H,i} = g(z_{H,i})$
  - The function $g(z)$ called the activation function
- Output units: The functions that compute
  - $z_{O,i} = \sum_j W_{O,ij} u_{H,j} + b_{O,i}$

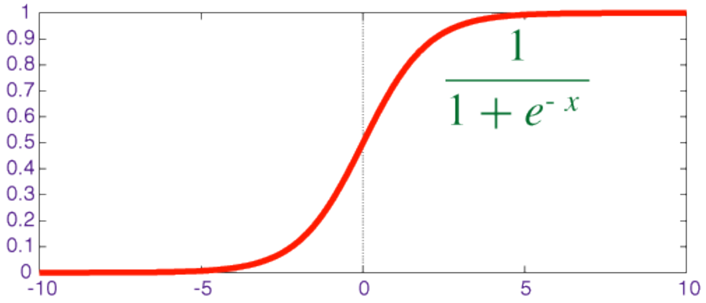


Response Map or Output Activation

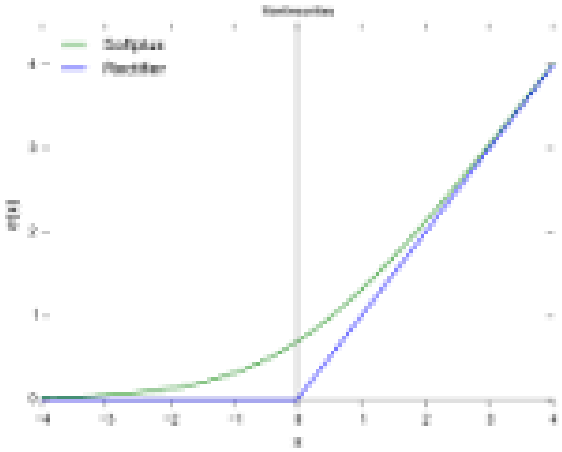
- Last layer depends on type of response
- Binary classification: $y = \pm 1$
  - z_O is a scalar
  - Hard decision: $\hat{y} = \mathrm{sign}(z_O)$
  - Soft decision: $P(y = 1|x) = 1/(1 + e^{-z_O})$
- Multi-class classification: $y = 1, \ldots, K$
  - $\boldsymbol{z}_O = \left[z_{O,1}, \cdots, z_{O,K}\right]^T$ is a vector
  - Hard decision: $\hat{y} = \arg\max_k z_{O,k}$
  - Soft decision: $P(y = k|x) = S_k(\boldsymbol{z}_O)$, $S(\boldsymbol{z}_O) =$ softmax
- Regression: $y \in R^d$
  - $\hat{y} = z_O$

Hidden Activation Function

- Two common activation functions
- Sigmoid:
  - $g_{act}(z) = 1/1 + e^{-z}$
  - Benefits: Values are bounded
  - Often used for small networks



- Rectified linear unit (ReLU):
  - $g_{act}(z) = \max(0, z)$
  - Can add sparsity (more on this later)
  - Often used for larger networks
  - Esp. in combination with dropout



Training a Neural Network

- Given data: $(\boldsymbol{x}_i, y_i), i = 1, \ldots, N$
- Learn parameters: $\theta = (W_H, b_H, W_o, b_o)$
  - Weights and biases for hidden and output layers
- Will minimize a loss function: $L(\theta)$

$$\hat{\theta} = \arg\min_{\theta} L(\theta)$$

  - $L(\theta)$ = measures how well parameters θ fit training data $(\boldsymbol{x}_i, y_i)$

Selecting the Right Loss Function

- Depends on the problem type
- Always compare final output z_{Oi} with target y_i

Problem	Target y_i	Output z_{Oi}	Loss function	Formula
Regression	y_{i} = Scalar real	z_{Oi} = Prediction of y_i Scalar output / sample	Squared / L2 loss	$\sum_i (y_i - z_{Oi})^2$
Regression with vector samples	$\boldsymbol{y}_i = (y_{i1}, \dots, y_{iK})$	z_{Oik} = Prediction of y_{ik} K outputs / sample	Squared / L2 loss	$\sum_{ik} (y_{ik} - z_{Oik})^2$
Binary classification	$y_{\mathrm{i}} = \{0,1\}$	z_{Oi} = "logit" score Scalar output / sample	Binary cross entropy	$\sum_i -y_i z_{Oi} + \ln(1 + e^{y_i z_i})$
Multi-class classification	$y_{\mathrm{i}} = \{1, \dots, K\}$	z_{Oik} = "logit" scores K outputs / sample	Categorical cross entropy	$\sum_i \ln\left(\sum_k e^{z_{Oik}}\right) - \sum_k r_{ik} z_{Oik}$

Loss Function: Regression

- Regression case:
  - y_i = scalar target variable for sample i
  - Typically continuous valued
- Output layer:
  - z_{Oi} = estimate of y_i
- Loss function: Use L2 loss

$$L(\theta) = \sum_{i=1}^{N} (y_i - z_{Oi})^2$$

- For vector $\boldsymbol{y}_i = (y_{i1}, \ldots, y_{iK})$, use vector L2 loss

$$L(\theta) = \sum_{i=1}^{N} \sum_{j=1}^{K} (y_{ik} - z_{Oik})^2$$

Loss Function: Binary Classification

- Binary classification: $y_i = \{0,1\}$ = class label
- Loss function = negative log likelihood

$$L(\theta) = -\sum_{i=1}^{N} \ln P(y_i|x_i,\theta)\,, \qquad P(y_i = 1|x_i,\theta) = \frac{1}{1 + e^{-z_{oi}}}$$

  - Output z_{Oi} called the logit score
  - z_{Oi} scalar.
- From lecture on logistic regression:

$$-\ln P(y_i|x_i,\theta) = \ln[1 + e^{y_i z_{oi}}] - y_i z_{oi}$$

  - Called the binary cross-entropy

Loss Function: Multi-Class Classification 1

- $y_i = \{1, \ldots, K\}$ = class label
- Output: $\boldsymbol{z}_{Oi} = (z_{O,i1}, \ldots, z_{O,iK})$
  - K outputs. One per class
  - Also called the logit score
- Likelihood given by softmax:

$$P(y_i = k | \boldsymbol{x}_i, \theta) = g_k(z_{Oi}), \qquad g_k(z_{Oi}) = \frac{e^{z_{O,ik}}}{\sum_\ell e^{z_{O,ik}}}$$

  - Assigns class highest probability with highest logit score

Loss Function: Multi-Class Classification 2

- $y_i = \{1, \ldots, K\}$ = class label
- Define one-hot coded response

$$r_{ik} = \begin{cases} 1 & y_i = k \\ 0 & y_i \neq k \end{cases}$$

  - $\boldsymbol{r}_i = (r_{i1}, \ldots, r_{iK})$ is K-dimensional
- Negative log-likelihood given by:

$$L(\theta) = \sum_i \ln\left(\sum_k e^{z_{Oik}}\right) - \sum_k r_{ik} z_{O,ik}$$

  - Called the categorical cross-entropy

Problems with Standard Gradient Descent

- Neural network training (like all training): Minimize loss function

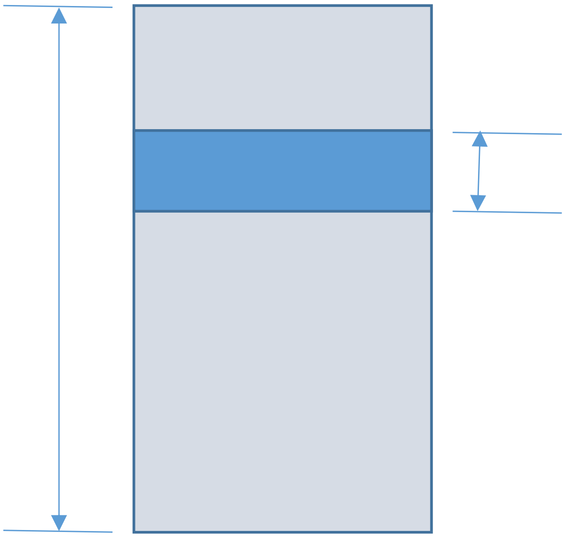
$$\hat{\theta} = \arg\min_{\theta} L(\theta)\,, \qquad L(\theta) = \sum_{i=1}^{N} L_i(\theta, \boldsymbol{x}_i, y_i)$$

  - $L_i(\theta, \boldsymbol{x}_i, y_i)$ = loss on sample i for parameter θ
- Standard gradient descent:

$$\theta^{k+1} = \theta^k - \alpha \nabla L\left(\theta^k\right) = \theta^k - \alpha \sum_{i=1}^{N} \nabla L_i(\theta^k, \boldsymbol{x}_i, y_i)$$

  - Each iteration requires computing N loss functions and gradients
  - Will discuss how to compute later
  - But, gradient computation is expensive when data size N large

Stochastic Gradient Descent



Full batch of training records

e.g. 50,000 in MNIST

Randomly selected mini-batch

e.g. 100 records

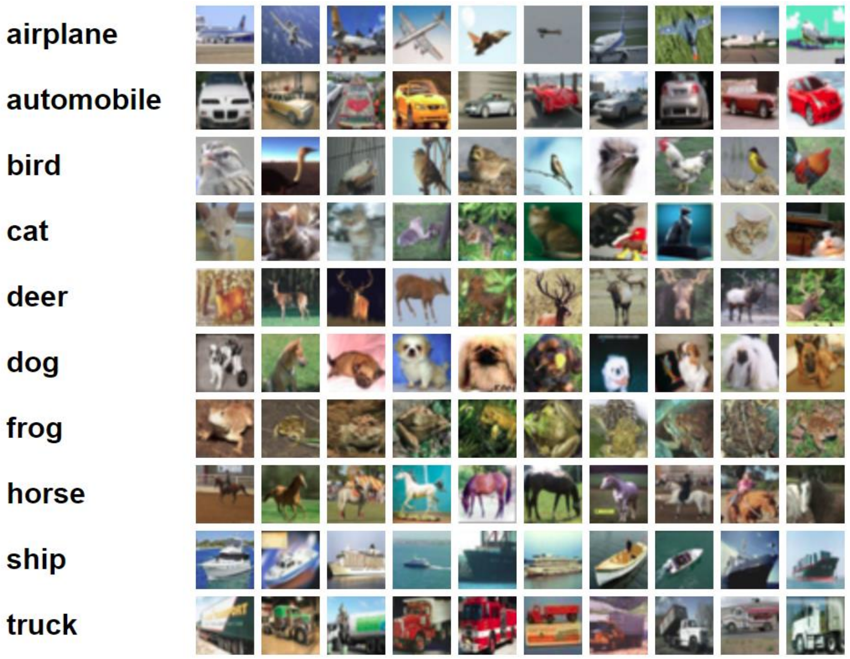
- In each step:
  - Select random small “mini-batch”
  - Evaluate gradient on mini-batch

- For $t = 1$ to N_{steps}
  - Select random mini-batch $I \subset \{1, \ldots, N\}$
  - Compute gradient approximation: $$g^t = \frac{1}{|I|} \sum_{i \in I} \nabla L(x_i, y_i, \theta)$$
  - Update parameters: $$\theta^{t+1} = \theta^t - \alpha^t g^t$$

Large-Scale Image Classification

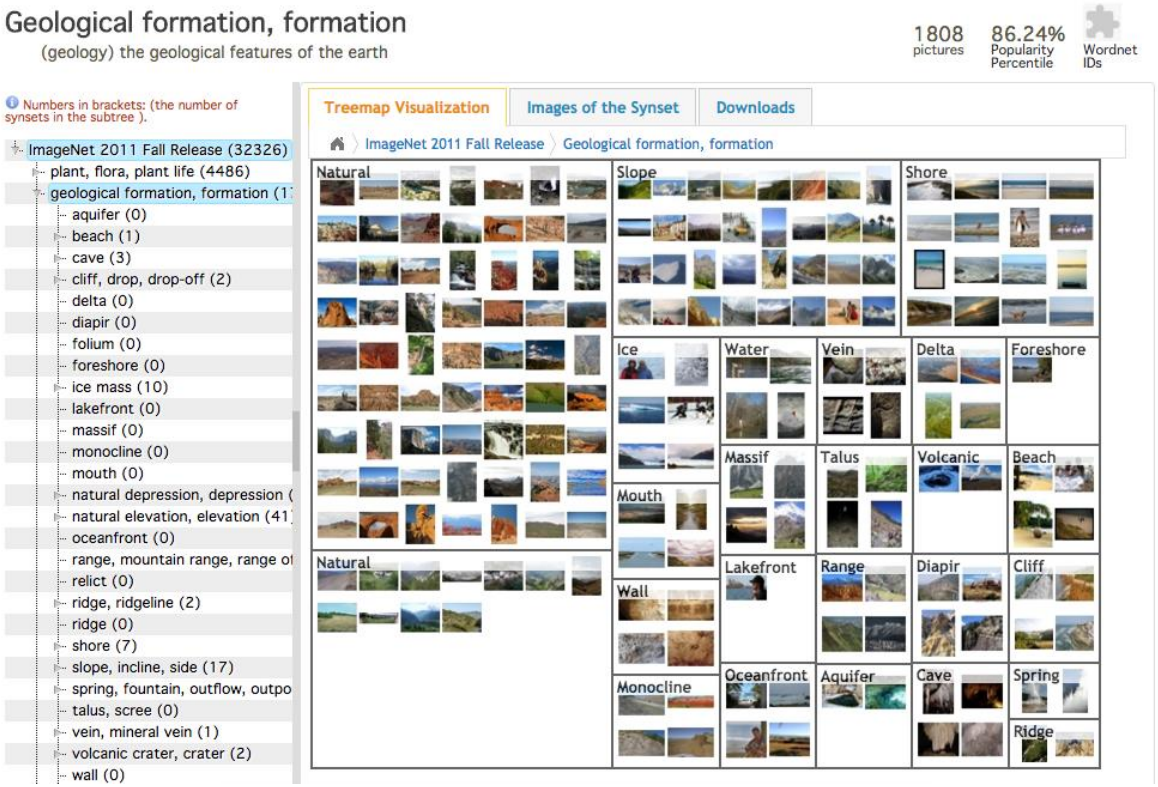
- Pre-2009, many image recognition systems worked on relatively small datasets
  - MNIST: 10 digits
  - CIFAR 10 (right)
  - CIFAR 100
  - …
- Small number of classes (10-100)
- Low resolution (eg. 32 x 32 x 3)
- Performance saturated
  - Difficult to make significant advancemen

https://www.cs.toronto.edu/~kriz/cifar.html



ImageNet (2009)

- Better algorithms need better data
- Build a large-scale image dataset
- 2009 CVPR paper:
  - 3.2 million images
  - Annotated by mechanical turk
  - Much larger scale than any previous
- Hierarchical categories



Geological formation, formation

(geology) the geological features of the earth

1808 pictures 86.24% Popularity Percentile Wordnet IDs

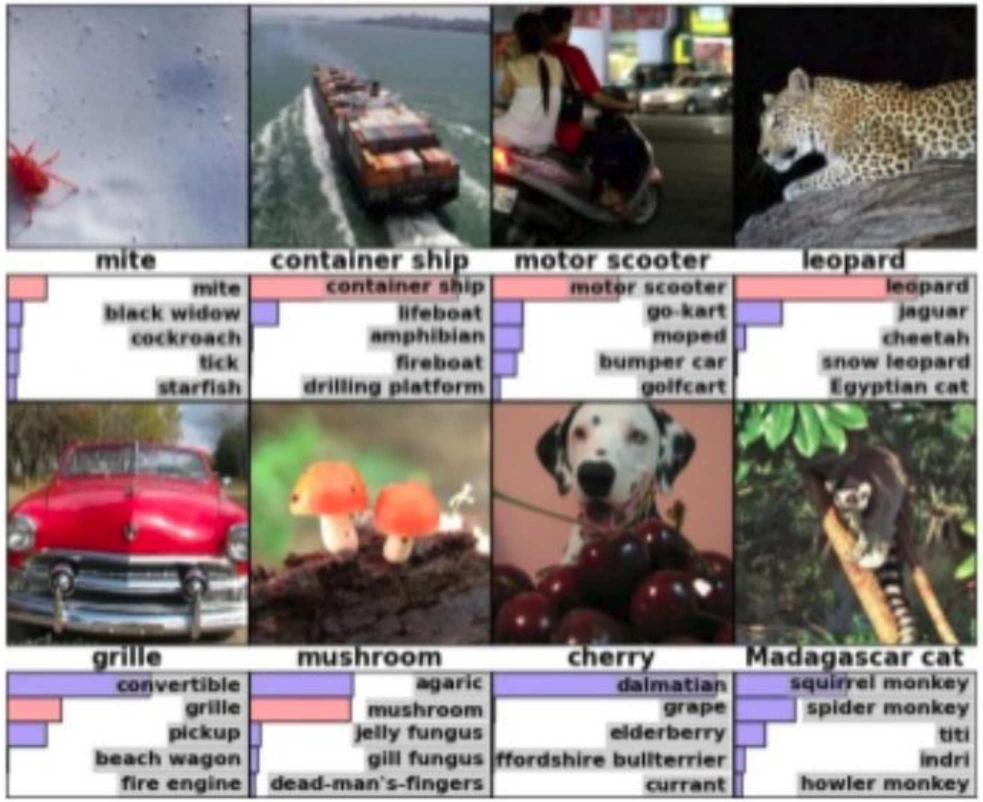
Numbers in brackets: (the number of synsets in the subtree).

- ImageNet 2011 Fall Release (32326)
  - plant, flora, plant life (4486)
  - geological formation, formation (1
    - aquifer (0)
    - beach (1)
    - cave (3)
    - cliff, drop, drop-off (2)
    - delta (0)
    - diapir (0)
    - folium (0)
    - foreshore (0)
    - ice mass (10)
    - lakefront (0)
    - massif (0)
    - monocline (0)
    - mouth (0)
    - natural depression, depression (
    - natural elevation, elevation (41
    - oceanfront (0)
    - range, mountain range, range of
    - relict (0)
    - ridge, ridgeline (2)
    - ridge (0)
    - shore (7)
    - slope, incline, side (17)
    - spring, fountain, outflow, outpo
    - talus, scree (0)
    - vein, mineral vein (1)
    - volcanic crater, crater (2)
    - wall (0)

Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* (pp. 248-255). IEEE.

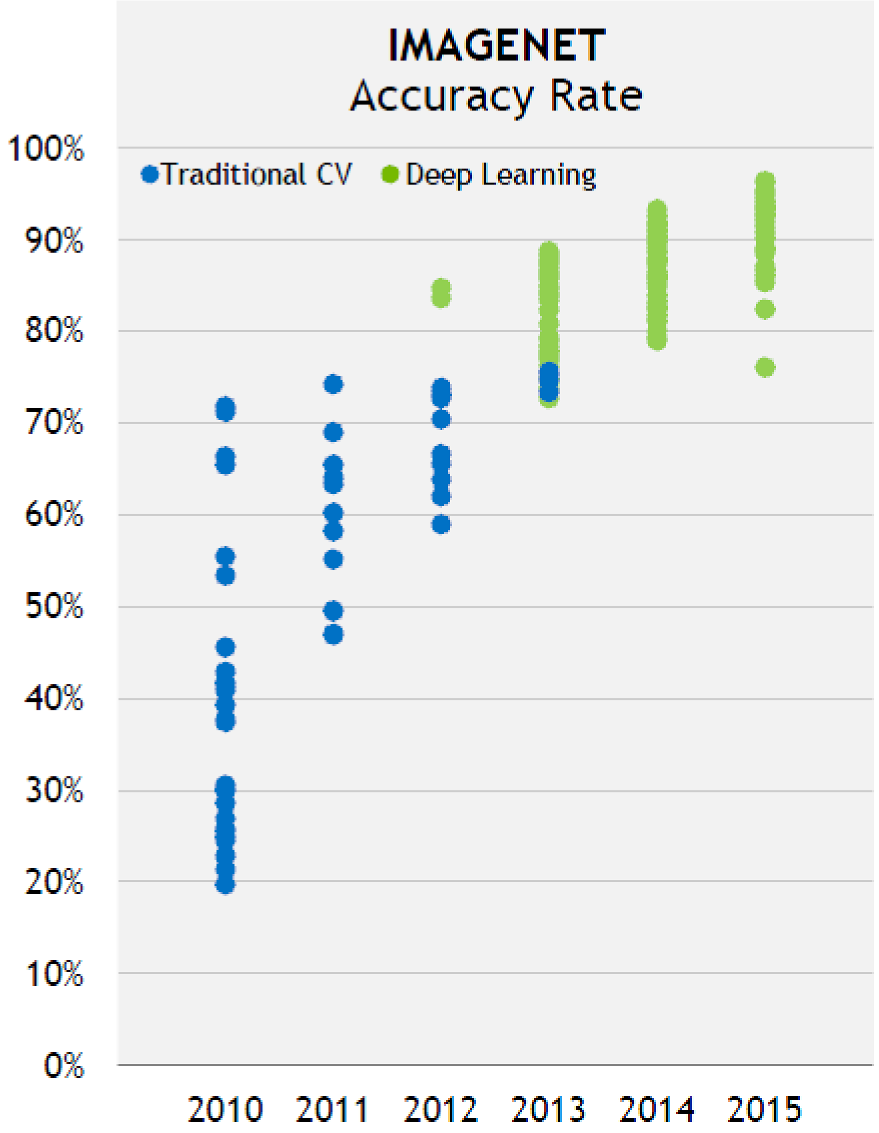
ILSVRC

- ImageNet Large-Scale Visual Recognitic
- First year of competition in 2010
- Many developers tried their algorithms
- Many challenges:
  - Objects in variety of positions, lighting
  - Occlusions
  - Fine-grained categories
    (e.g. African elephants vs. Indian elephan
  - …



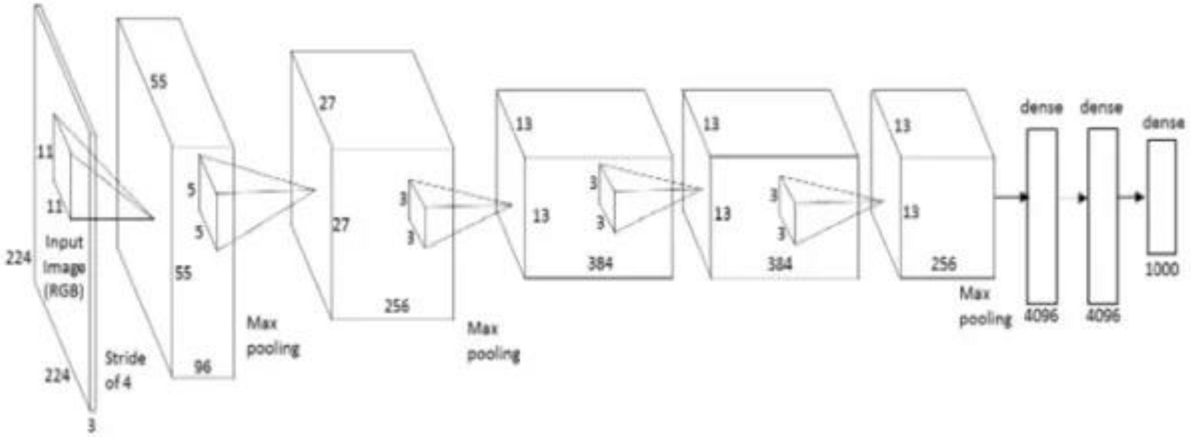
Deep Networks Enter 2012

- 2012: Stunning breakthrough by the first deep network
- “AlexNet” from U Toronto
- Easily won ILSVRC competition
  - Top-5 error rate: 15.3%, second place: 25.6%
- Soon, all competitive methods are deep networks



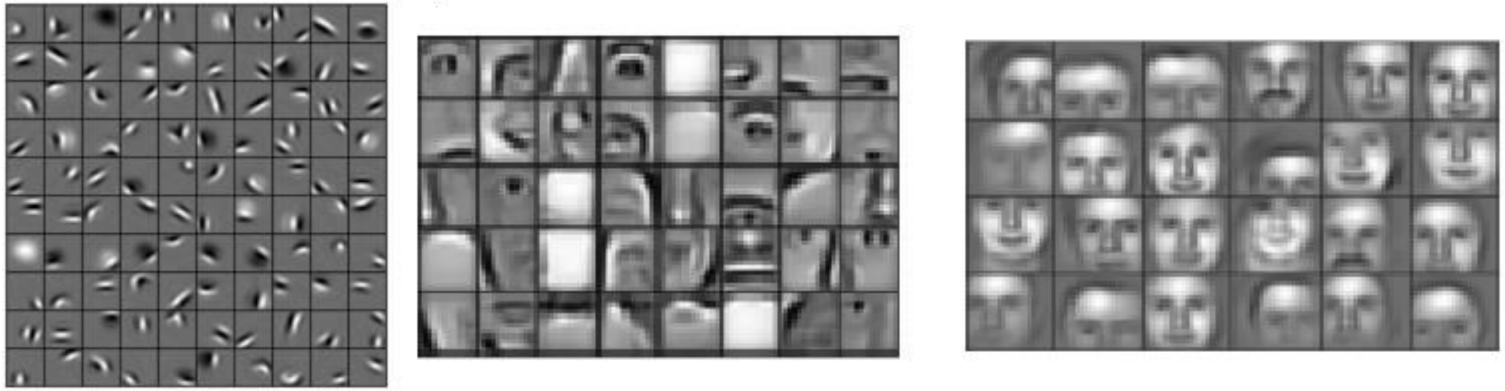
Alex Net

- Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton University of Toronto, 2012
- Key idea: Build a very deep neural network
- 60 million parameters, 650,000 neurons
- 5 conv layers + 3 FC layers
- Final is 1000-way softmax



Local Features

- Early layers in deep neural networks often find local features
- Small patterns in larger image
  - Examples: Small lines, curves, edges
- Build more complex classification from the local features

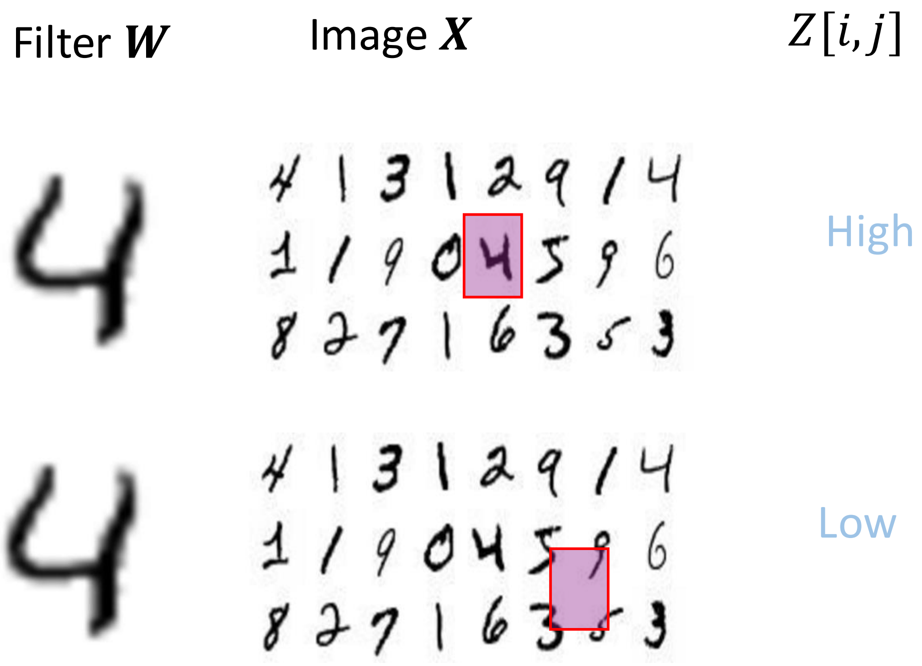


Localization via a Sliding Window

- Simple idea: Find local feature by sliding window
- Large image: $X \;\; N_1 \times N_2$ (e.g. 512 x 512)
- Small filter: $W \;\; K_1 \times K_2$ (e.g. 8 x 8)
- At each offset (i, j) compute:

$$Z[i,j] = \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} W[k_1,k_2] X[i+k_1, j+k_2$$

  - Correlation of W with image box starting at (i, j)
  - $Z[i,j]$ is large if feature is present around (i, j)

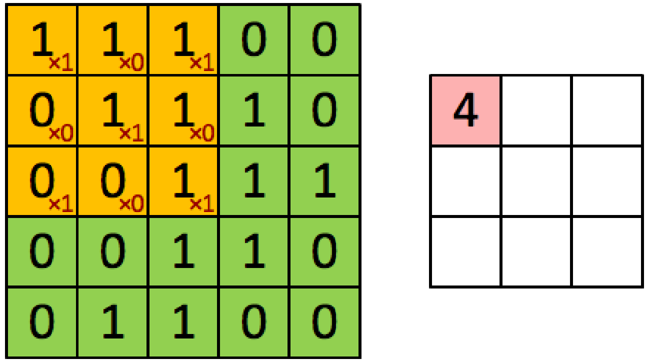


Convolution 2D Example

- Kernel

$$W = \widetilde{W} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

- Compute convolution in valid region



$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0
$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
0	0	1	1	0
0	1	1	0	0

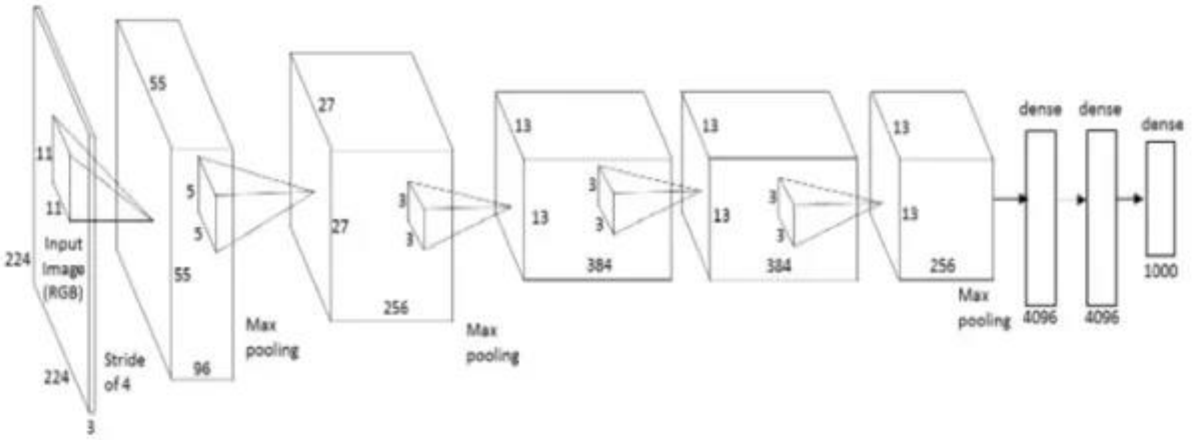
Image

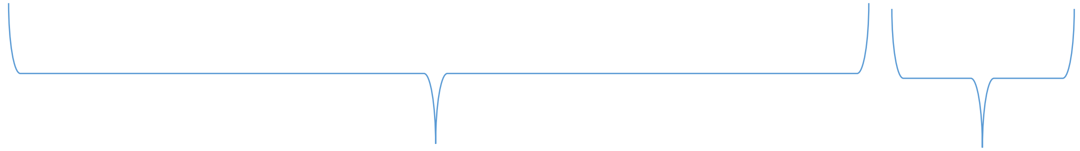
4		

Convolved Feature

https://stats.stackexchange.com/questions/199702/1d-convolution-in-neural-networks

Classic CNN Structure





Convolutional layers

2D convolution with
Activation and
pooling / sub-sampling

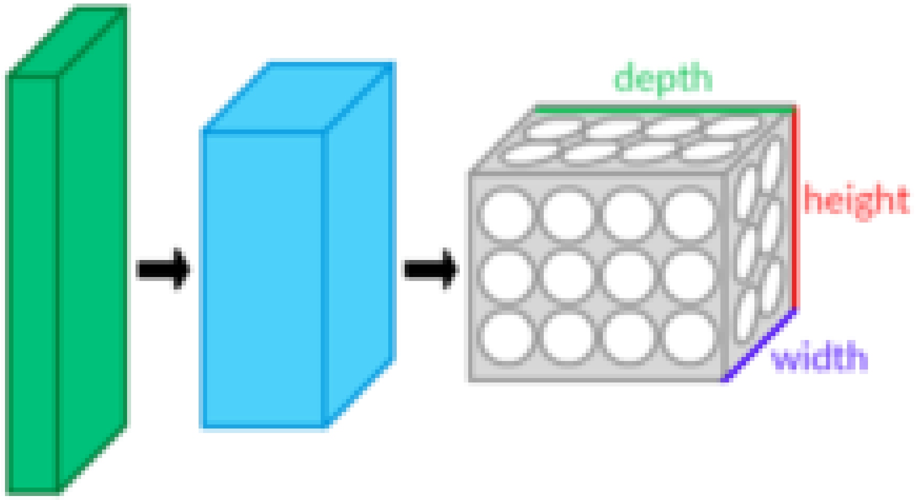
Fully connected layers

Matrix multiplication &
activation

- Alex Net example
- Each convolutional layer has:
  - 2D convolution
  - Activation (eg. ReLU)
  - Pooling or sub-sampling

Convolutional Inputs & Outputs

- Inputs and outputs are images with multiple channels
  - Number of channels also called the depth
- Can be described as tensors
- Input tensor, X shape (N_1, N_2, N_{in})
  - $N_1, N_2 = \text{input image size}$
  - N_{in} = number of input channels
- Output tensor, Z shape (M_1, M_2, N_{out})
  - $M_1, M_2 = \text{output image size}$
  - N_{out} = number of output channels



Convolutions with Multiple Channels

- Weight and bias:
  - W: Weight tensor, size $(K_1, K_2, N_{in}, N_{out})$
  - b: Bias vector, size N_{out}
- Convolutions performed over space and added over channels

$$Z[i_1, i_2, m] = \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} \sum_{n=0}^{N_{in}-1} W[k_1, k_2, n, m] X[i_1 + k_1, i_2 + k_2, n] + b[m]$$

- For each output channel m, input channel n
  - Computes 2D convolution with $W[:, :, n, m]$
  - Sums results over n

Activation and Sub-Sampling

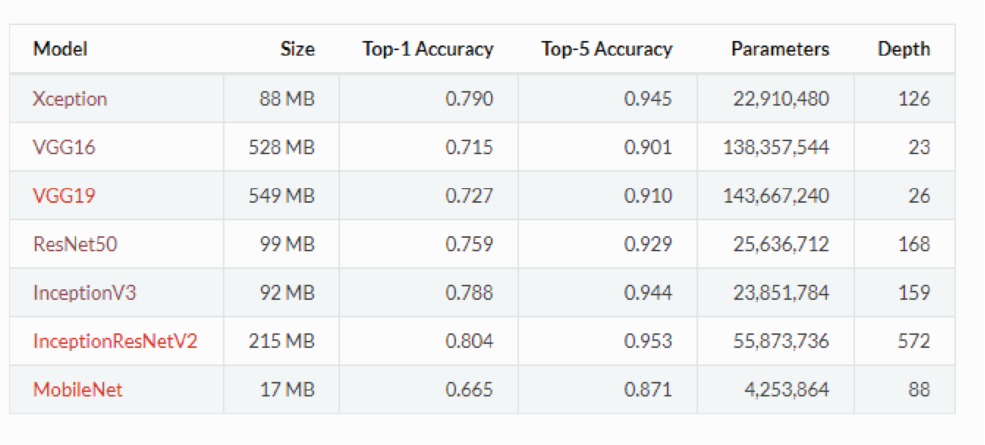
- Convolution typically followed by activation and pooling
- Activation, typically ReLU
    - Zeros out portions of image
- Sub-sampling
    - Downsample output after activation
    - Different methods (striding, sub-sampling or max-pooling)
    - Output combines local features from adjacent regions
    - Creates more complex features over wider areas
- Details for sub-sampling not covered in this class
    - See web for more info

Convolution vs Fully Connected

- Convolution exploits translational invariance
  - Same features is scanned over whole image
- Greatly reduces number of parameters
- Example Consider first layer in LeNet
  - 32 x 32 image filtered by 6 channels 5 x 5 each
  - Creates 6 x 28 x 28 outputs (edges removed in convolution)
  - Fully connected would require 32 x 32 x 6 x 28 x 28 = 4.9 million parameters!
  - Convolutional layer requires only 6 x 5 x 5 = 125 parameters (plus bias terms)
- Reserve fully connected layers for last few layers.

Pre-Trained Networks

- State-of-the-art networks take enormous resources to train
  - Millions of parameters
  - Often days of training, clusters of GPUs
  - Extremely expensive
- Pre-trained networks in Keras
  - Load network architecture and weights
  - Models available for many state-of-the-art networks



Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
InceptionV3	92 MB	0.788	0.944	23,851,784	159
InceptionResNetV2	215 MB	0.804	0.953	55,873,736	572
MobileNet	17 MB	0.665	0.871	4,253,864	88

https://keras.io/applications/

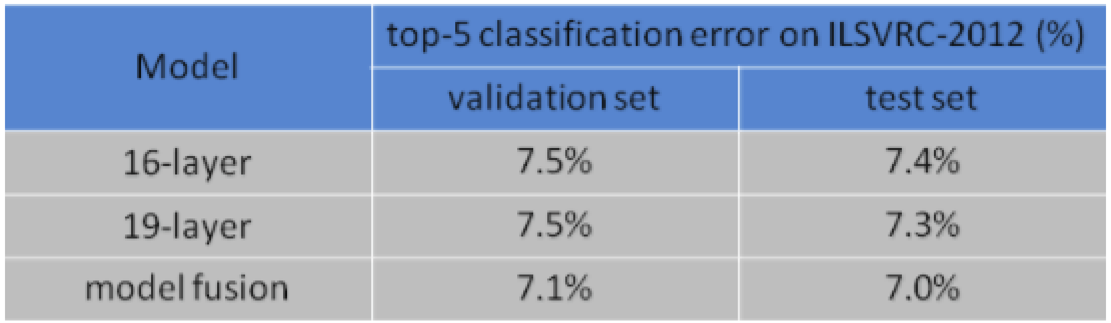
- Can be used for:
  - Making predictions
  - Building new, powerful networks (see lab)

VGG16

- From the Visual Geometry Group
  - Oxford, UK

- Won ImageNet ILSVRC-2014
- Remains a very good network

- Will load this network today



Model	top-5 classification error on ILSVRC-2012 (%)	
	validation set	test set
16-layer	7.5%	7.4%
19-layer	7.5%	7.3%
model fusion	7.1%	7.0%

http://www.robots.ox.ac.uk/~vgg/research/very_deep/

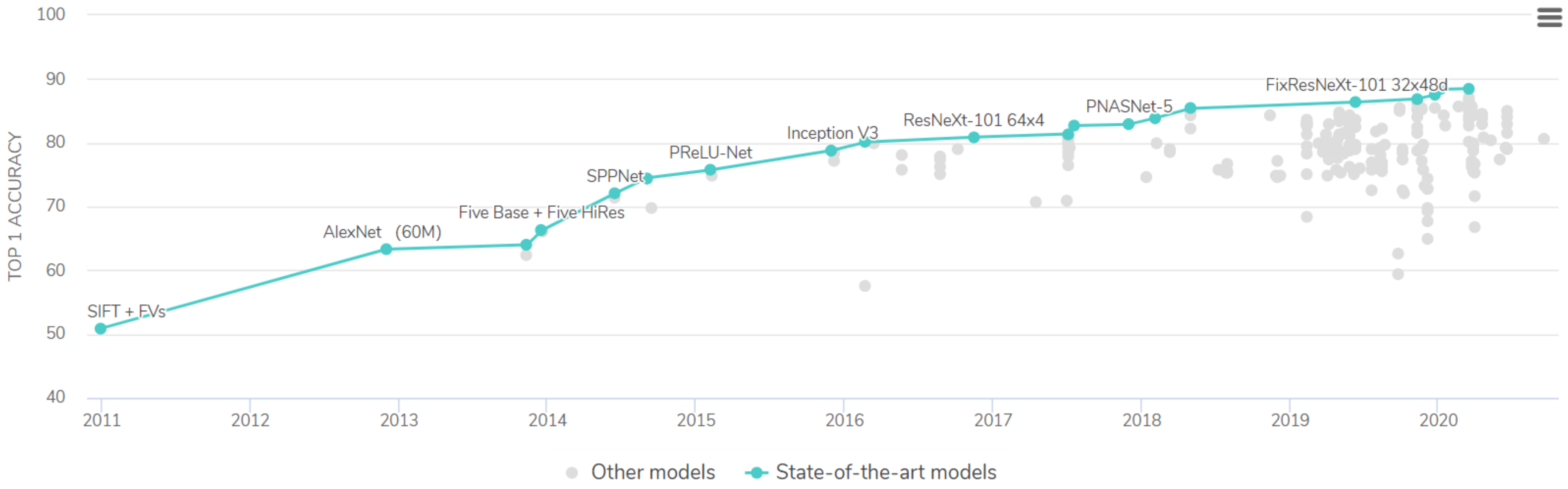
K. Simonyan, A. Zisserman
Very Deep Convolutional Networks for Large-Scale Image Recognition
arXiv technical report, 2014

State of the Art Today for Image Classification

https://kobiso.github.io/Computer-Vision-Leaderboard/imagenet.html

https://paperswithcode.com/sota/image-classification-on-imagenet

Image Classification on ImageNet


TOP 1 ACCURACY
SIFT + FVs
AlexNet (60M)
Five Base + Five HiRes
SPPNet
PReLU-Net
Inception V3
ResNeXt-101 64x4
PNASNet-5
FixResNeXt-101 32x48d
● Other models
State-of-the-art models