CY 7790

Special Topics in Security and Privacy: Machine Learning Security and Privacy
Fall 2021

Alina Oprea
Associate Professor
Khoury College of Computer Science

September 20 2021

Logistical Issues

- HW1 will be released today
  - It will be due on Friday, October 1
  - You will receive a Gradescope invitation
- Will release the presentation schedule later today
- Several resources on Piazza on reading papers
  - Paper template

Outline: Review of ML and Deep Learning

- Ensemble learning
  - Bagging, boosting
- Feed-forward neural networks
- Convolutional networks
  - Common architectures
- Regularization
- Backpropagation
- Comparing classifiers

Ensemble Learning

Consider a set of classifiers $h_1, \ldots, \mathrm{h}_\mathrm{L}$

Idea: construct a classifier $H(\mathbf{x})$ that combines the individual decisions of $h_1, \ldots, \mathrm{h}_\mathrm{L}$

- e.g., could have the member classifiers vote, or
- e.g., could use different members for different regions of the instance space

Successful ensembles require **diversity**

- Classifiers should make different mistakes
- Can have different types of base learners

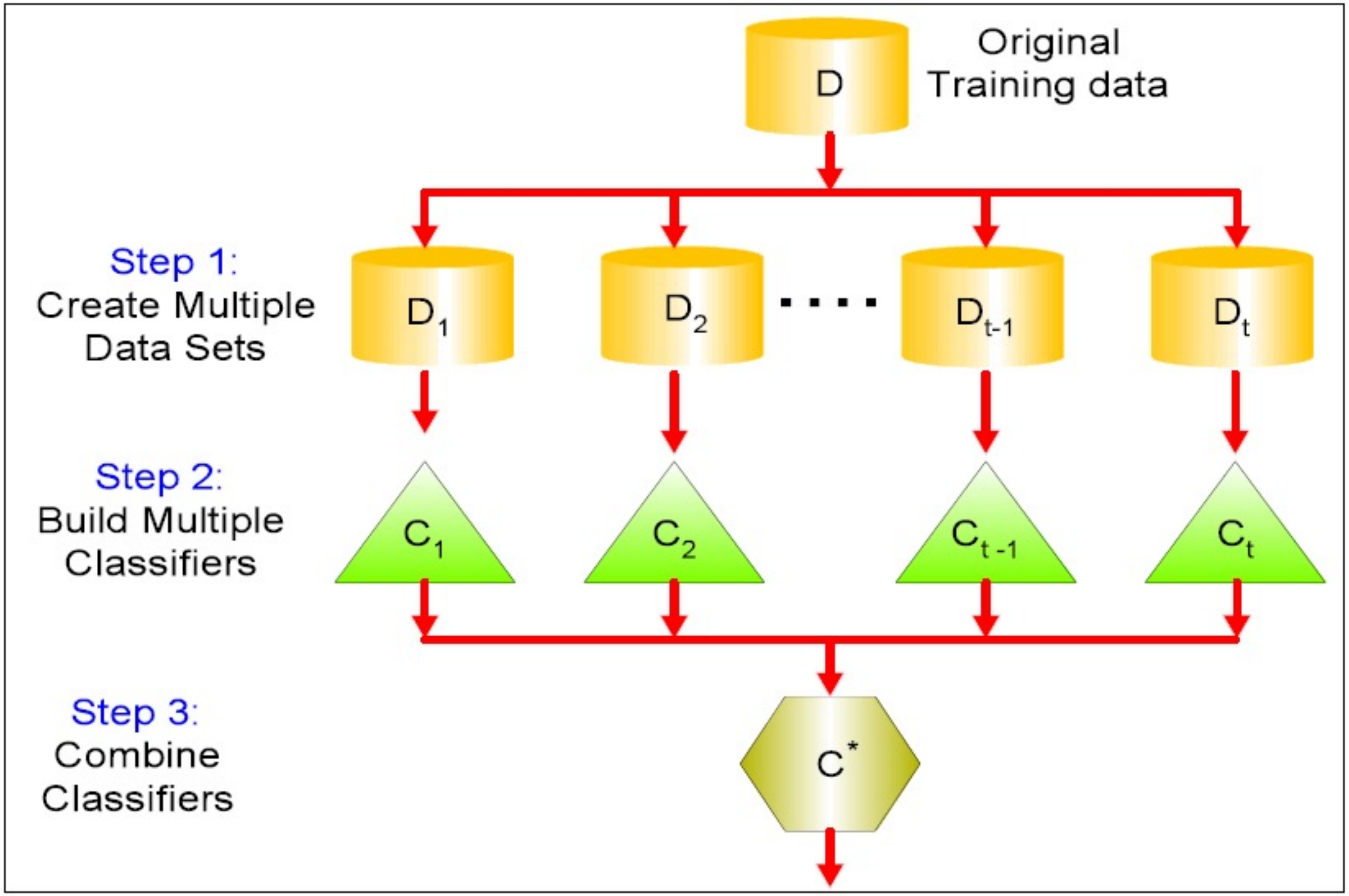
How to Achieve Diversity

- Avoid overfitting
  - Vary the training data
- Features are noisy
  - Vary the set of features

Two main ensemble learning methods

- Bagging
- Boosting

Bagging



Majority Votes

Random Forest Algorithm

1. For $b = 1$ to B:
   (a) Draw a $\boxed{\text{bootstrap sample}}$ $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select $\boxed{m \text{ variables at random}}$ from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

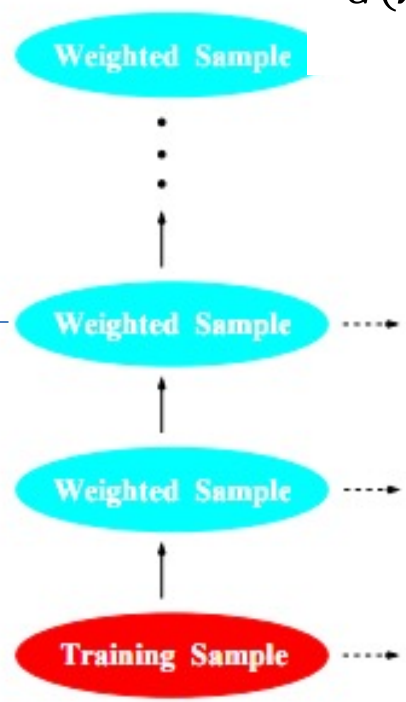
Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote } \{\hat{C}_b(x)\}_1^B$.

Overview of AdaBoost

$$G(x) = sign(\sum_{i=1}^{T} \beta_t h_t(x))$$

Better classifiers will get higher weights

- Mis-classified examples get higher weights
- Correct examples get lower weights



$h_3(x)$

$h_2(x)$

$h_1(x)$

Uniform weights

FIGURE 10.1. *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

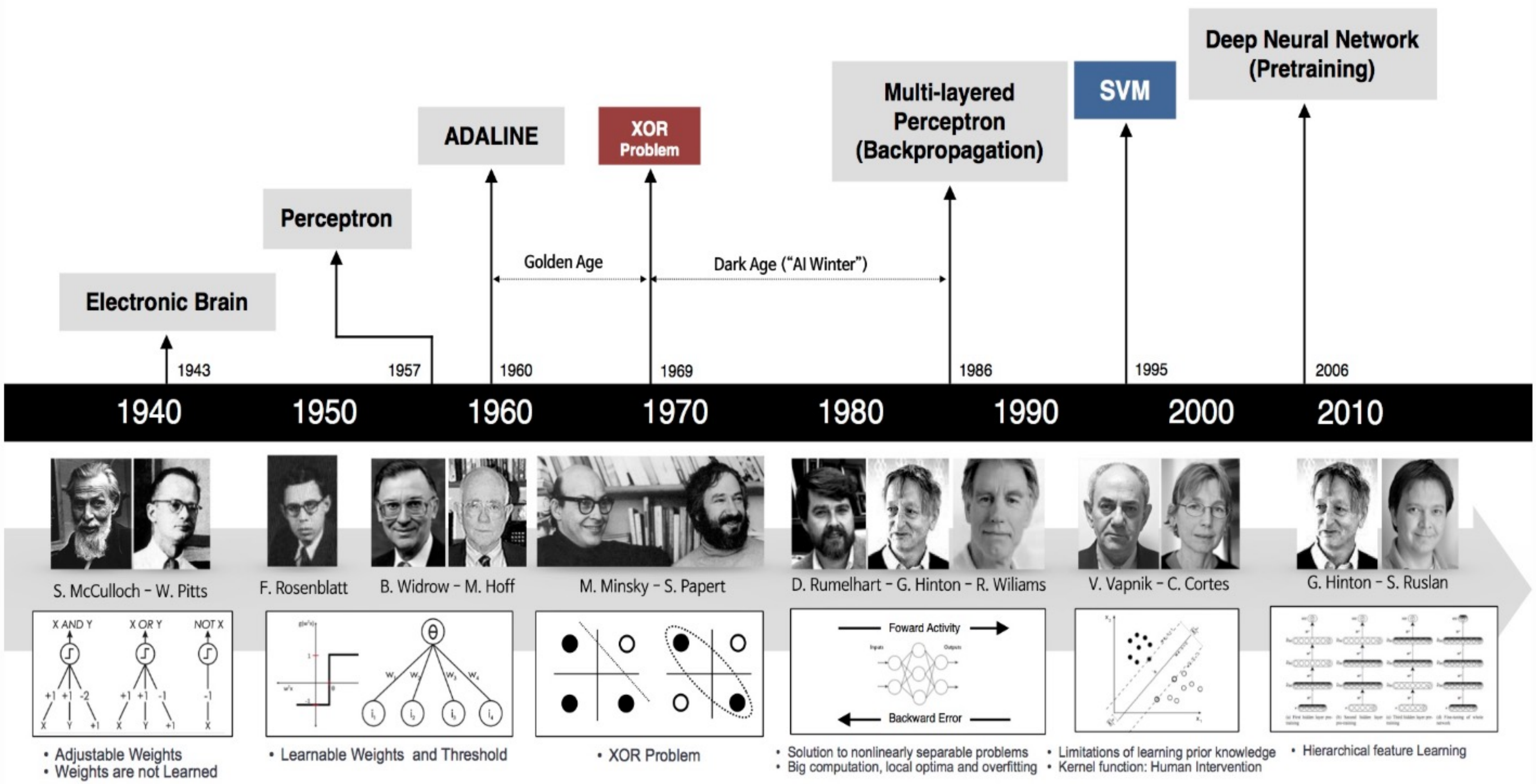
Bagging vs Boosting

Bagging	vs.	Boosting
Resamples data points		Reweights data points (modifies their distribution)
Weight of each classifier is the same		Weight is dependent on classifier's accuracy
Only variance reduction		Both bias and variance reduced – learning rule becomes more complex with iterations

Applicable to complex models with low bias, high variance

Applicable to weak models with high bias, low variance

Timeline

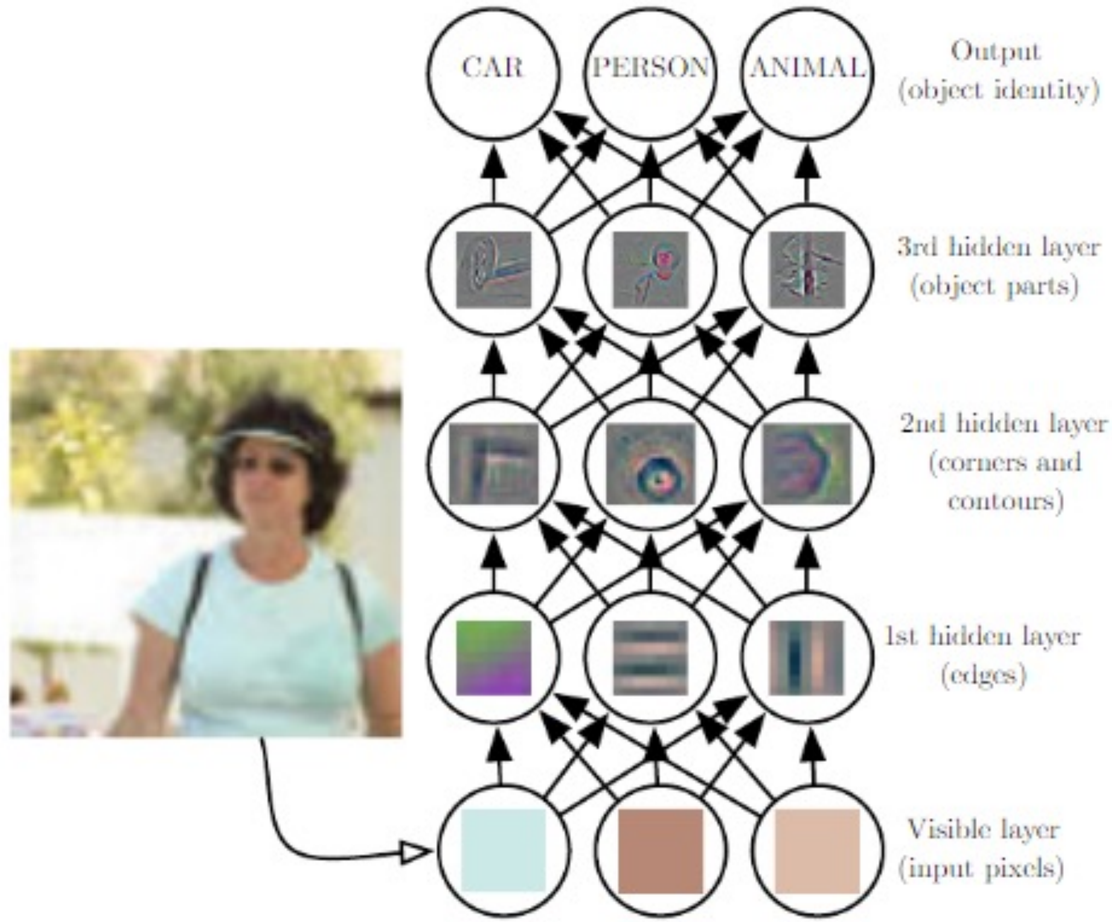


Deep Neural Network (Pretraining)
SVM
Multi-layered Perceptron (Backpropagation)
XOR Problem
ADALINE
Perceptron
Electronic Brain
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton - R. Wiliams
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

Deep Learning References

- Chapter 10 from Introduction to Statistical Learning
  - https://www.statlearning.com/
- Deep Learning books
  - https://d2l.ai/ (D2L)
  - https://www.deeplearningbook.org/ (advanced)
- Stanford notes on deep learning
  - http://cs229.stanford.edu/summer2020/cs229-notes-deep_learning.pdf
- Stanford tutorial on training Multi-Layer Neural Networks
  - http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/
- Notes on backpropagation by Andrew Ng
  - http://cs229.stanford.edu/notes-spring2019/backprop.pdf

Learning Representations

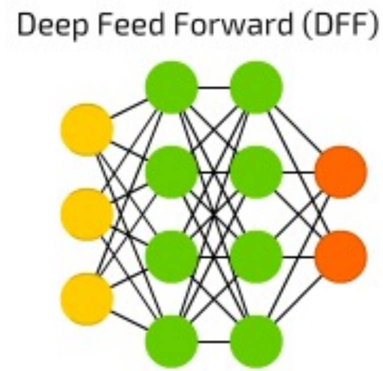


Deep Learning addresses the problem of learning hierarchical representations

Neural Network Architectures

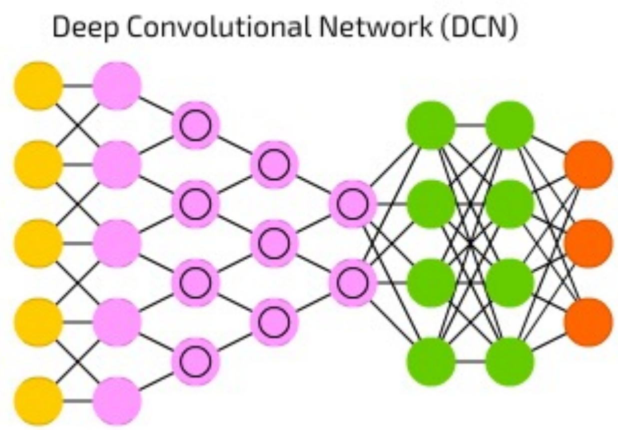
Feed-Forward Networks

- Neurons from each layer connect to neurons from next layer



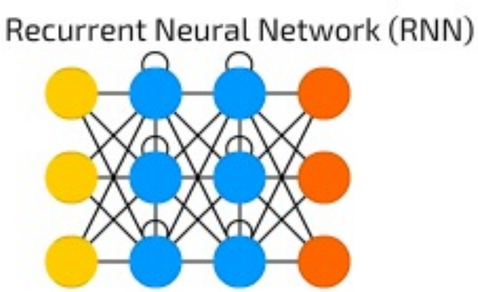
Convolutional Networks

- Includes convolution layer for feature reduction
- Learns hierarchical representations

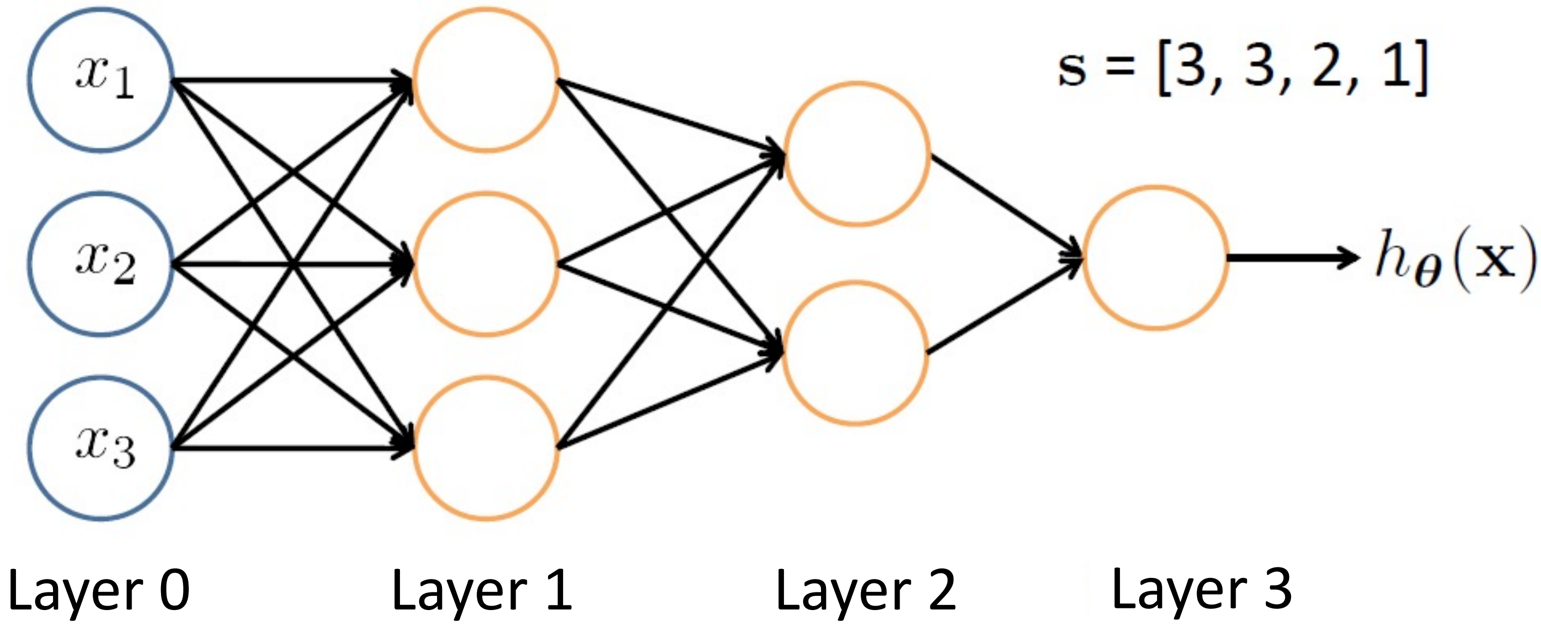


Recurrent Networks

- Keep hidden state
- Have cycles in computational graph



Feed-Forward Networks

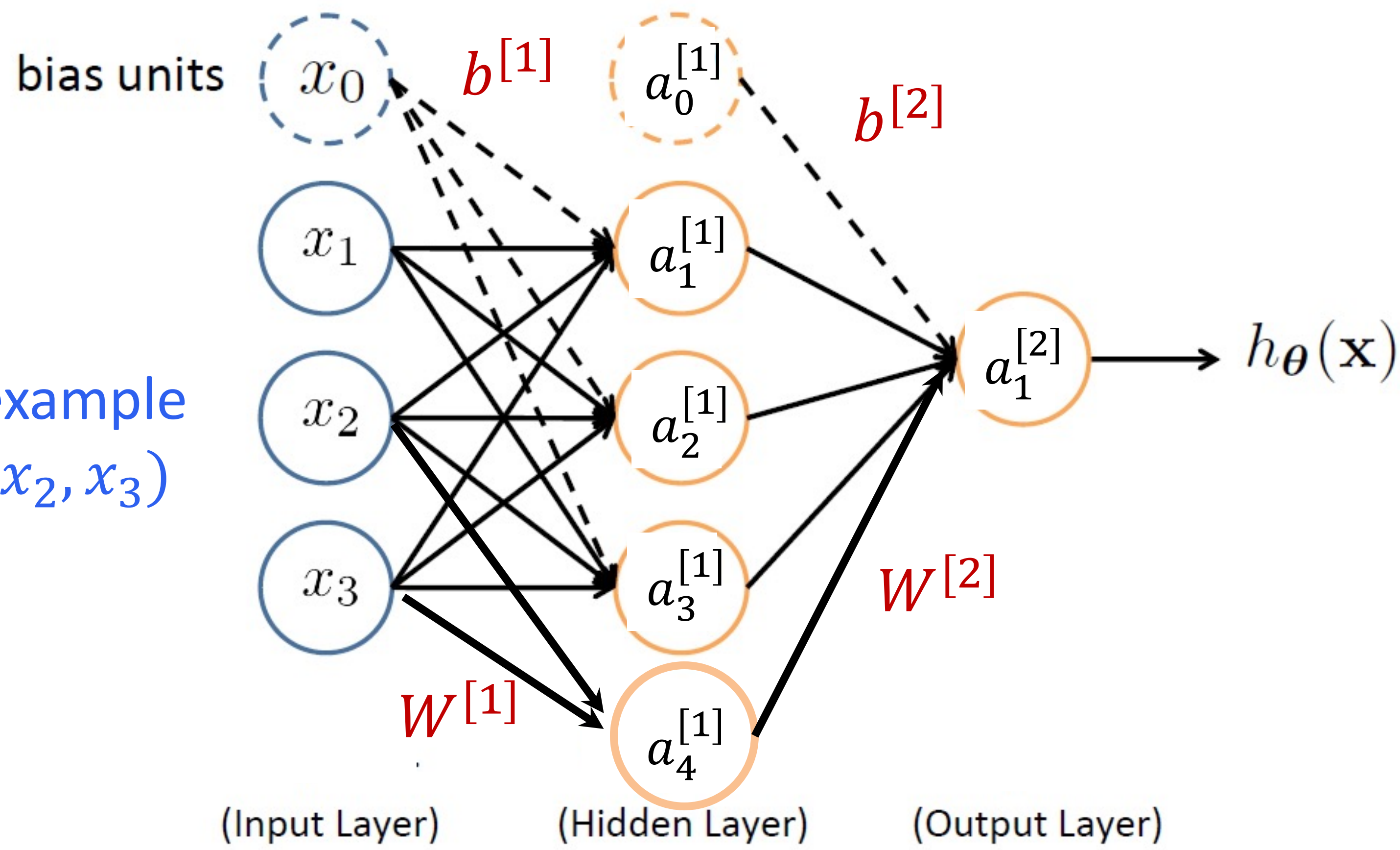


L denotes the number of layers

$\mathbf{s} \in {\mathbb{N}^{+}}^{L}$ contains the numbers of nodes at each layer

- Not counting bias units
- Typically, $s_0 = d$ (# input features) and $s_{L\text{-}1}=K$ (# classes)

Feed-Forward Neural Network



Training example

$x = (x_1, x_2, x_3)$

Layer 0 Layer 1 Layer 2

No cycles

$\theta = (b^{[1]}, W^{[1]}, b^{[2]}, W^{[2]})$

Layer Operations

$$
\begin{aligned}
z_1^{[1]} &= W_1^{[1]} x + b_1^{[1]} \quad \text{and} \quad a_1^{[1]} = g(z_1^{[1]}) \\
&\vdots \qquad\qquad\qquad\qquad \vdots \qquad\qquad\qquad \vdots \\
z_4^{[1]} &= W_4^{[1]} x + b_4^{[1]} \quad \text{and} \quad a_4^{[1]} = g(z_4^{[1]})
\end{aligned}
$$

$$
\boxed{\underbrace{\begin{bmatrix} z_1^{[1]} \\ \vdots \\ \vdots \\ z_4^{[1]} \end{bmatrix}}_{z^{[1]} \in \mathbb{R}^{4\times 1}} = \underbrace{\begin{bmatrix} — & W_1^{[1]} & — \\ — & W_2^{[1]} & — \\ & \vdots & \\ — & W_4^{[1]} & — \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{4\times 3}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x \in \mathbb{R}^{3\times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]} \in \mathbb{R}^{4\times 1}}}
$$

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

Linear

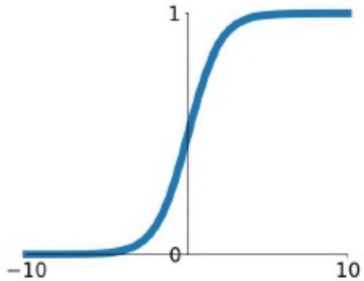
$$a^{[1]} = g(z^{[1]})$$

Non-Linear

Activation Functions

Sigmoid

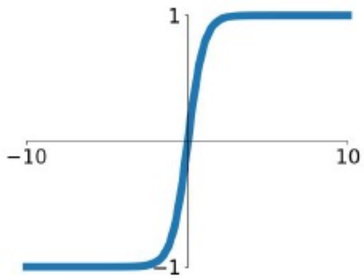
$\sigma(x) = \frac{1}{1+e^{-x}}$



Binary Classification

tanh

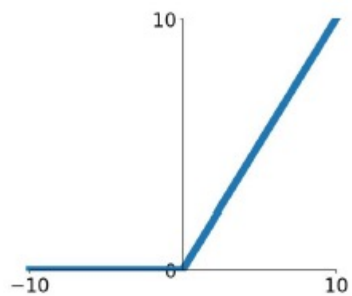
$\tanh(x)$



Regression

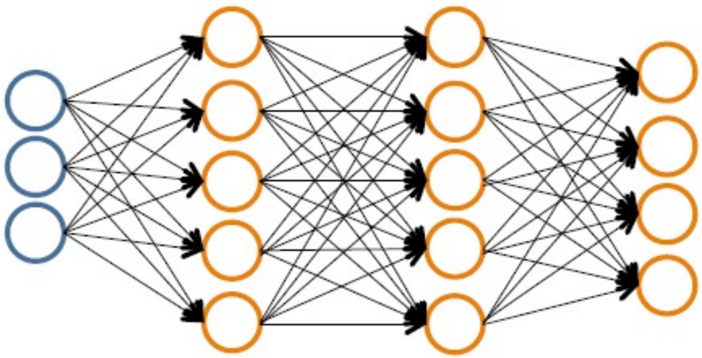
ReLU

$\max(0, x)$



Intermediary layers

Neural Network Classification



Given:

$\{(\mathbf{x}_1, y_1),\ (\mathbf{x}_2, y_2),\ \ldots,\ (\mathbf{x}_n, y_n)\}$

$\mathbf{s} \in \mathbb{N}^{+L}$ contains # nodes at each layer

- $s_0 = d$ (# features)

<u>Binary classification</u>

y = 0 or 1

1 output unit $(s_{L-1} = 1)$

Sigmoid

<u>Multi-class classification</u> (K classes)

$\mathbf{y} \in \mathbb{R}^K$ e.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

K output units $(s_{L-1} = K)$

Softmax

Softmax classifier

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

- Predict the class with highest probability
- Generalization of sigmoid/logistic regression to multi-class

Convolutional Nets

- Particular type of Feed-Forward Neural Nets
  - Invented by [LeCun 89]
- Applicable to data with natural grid topology
  - Time series
  - Images
- Use convolutions on at least one layer
  - Convolution is a linear operation that uses local information
  - Also use pooling operation
  - Used for dimensionality reduction and learning hierarchical feature representations

Convolutional Nets

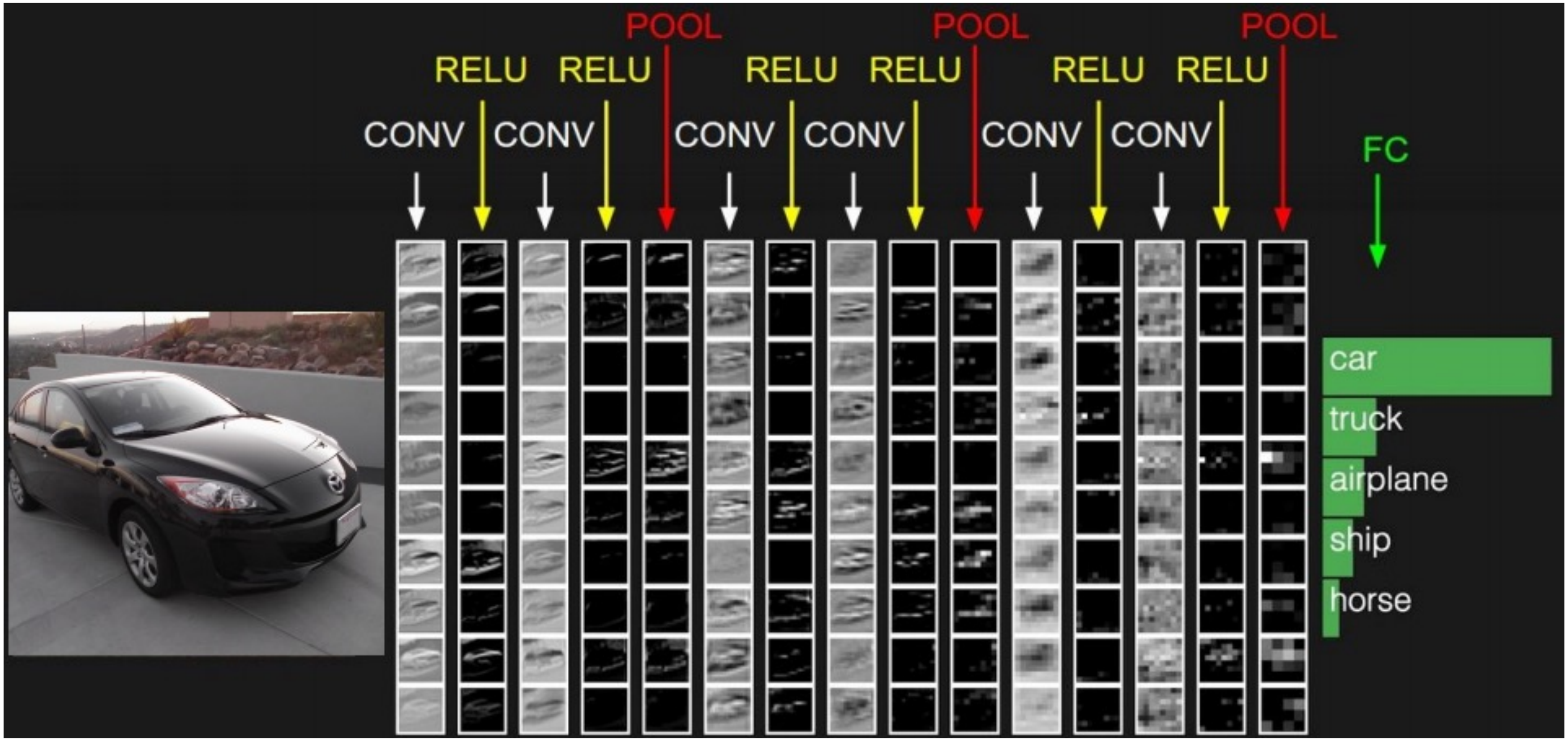
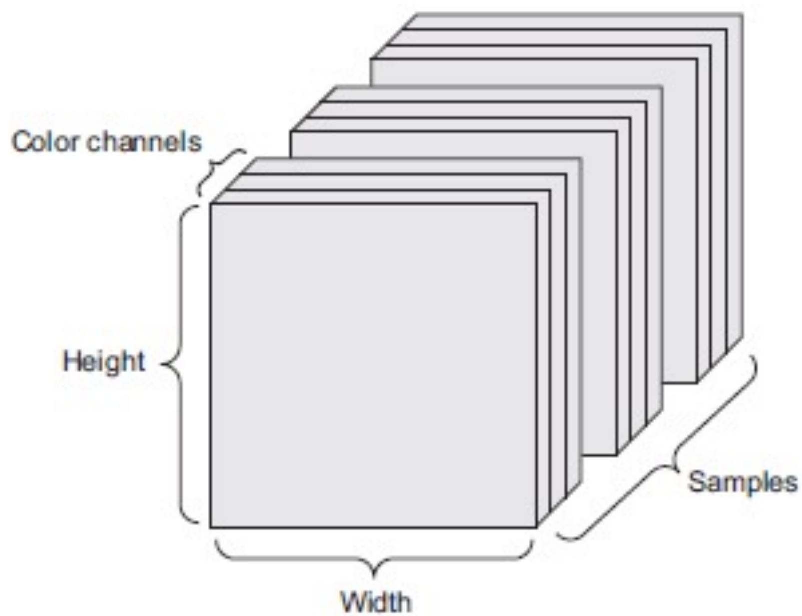


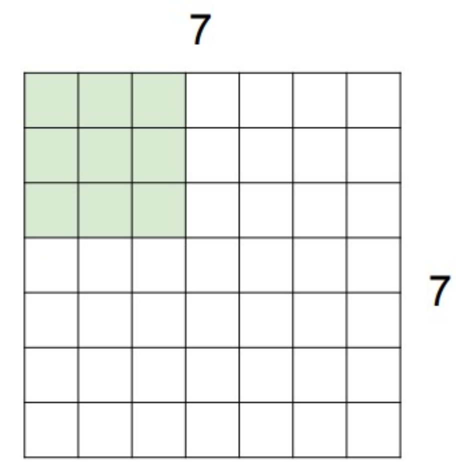
Image Representation

- Image is 3D "tensor": height, width, color channel (RGB)
- Black-and-white images are 2D matrices: height, width
  - Each value is pixel intensity

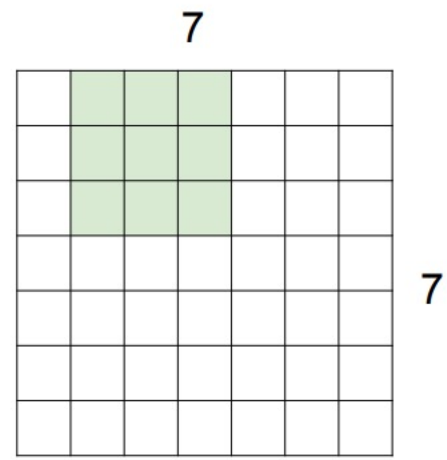


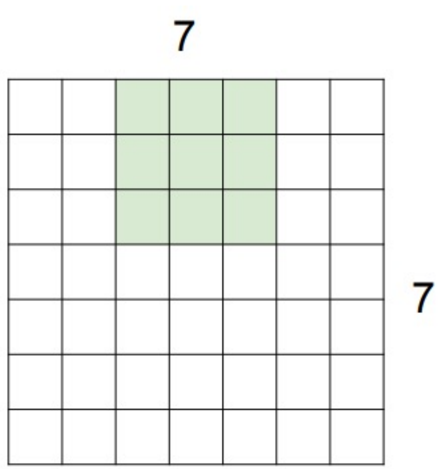
Convolutions

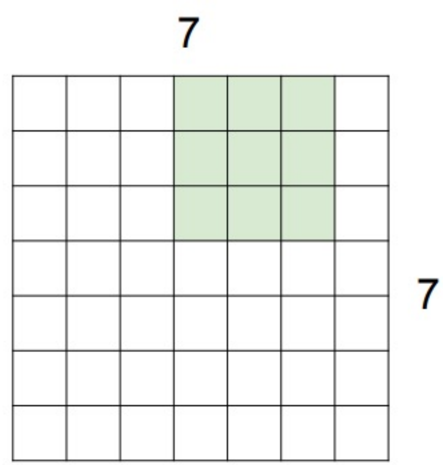
A closer look at spatial dimensions:

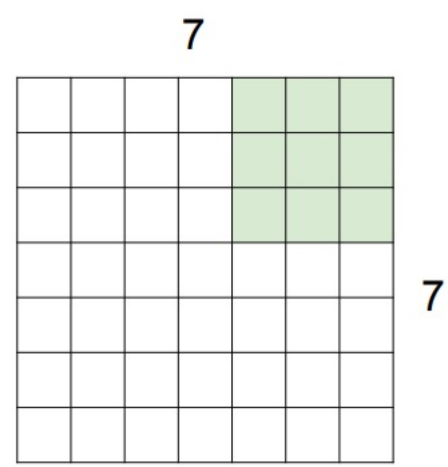


7x7 input (spatially)
assume 3x3 filter







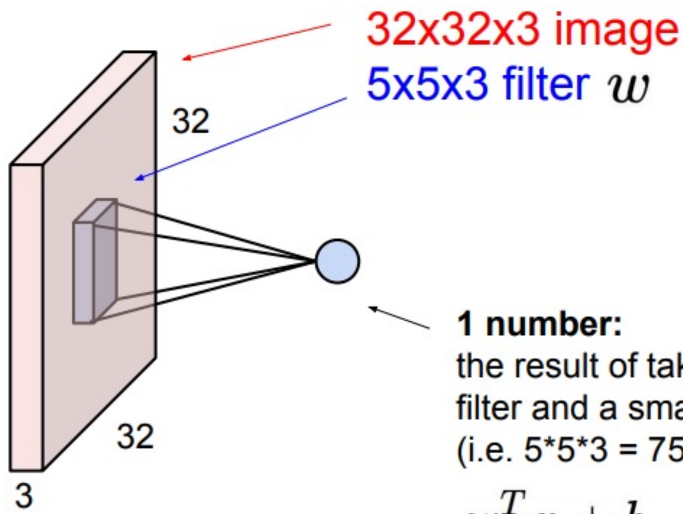


Example

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 19 & 25 \\ 37 & 43 \end{bmatrix}$$

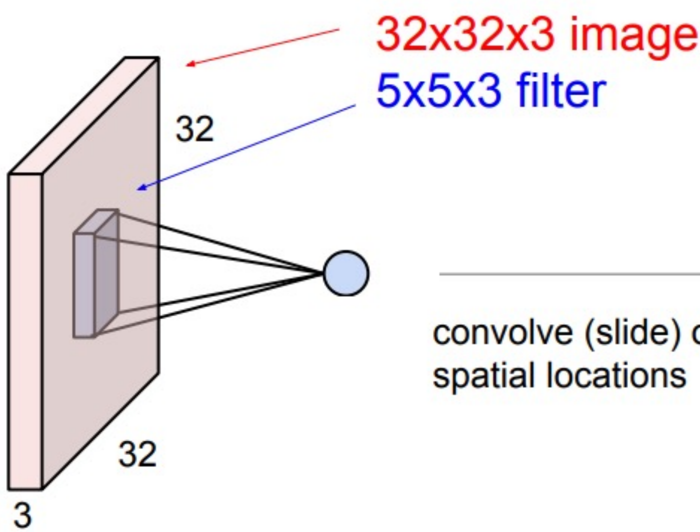
Input Filter Output

Convolution Layer



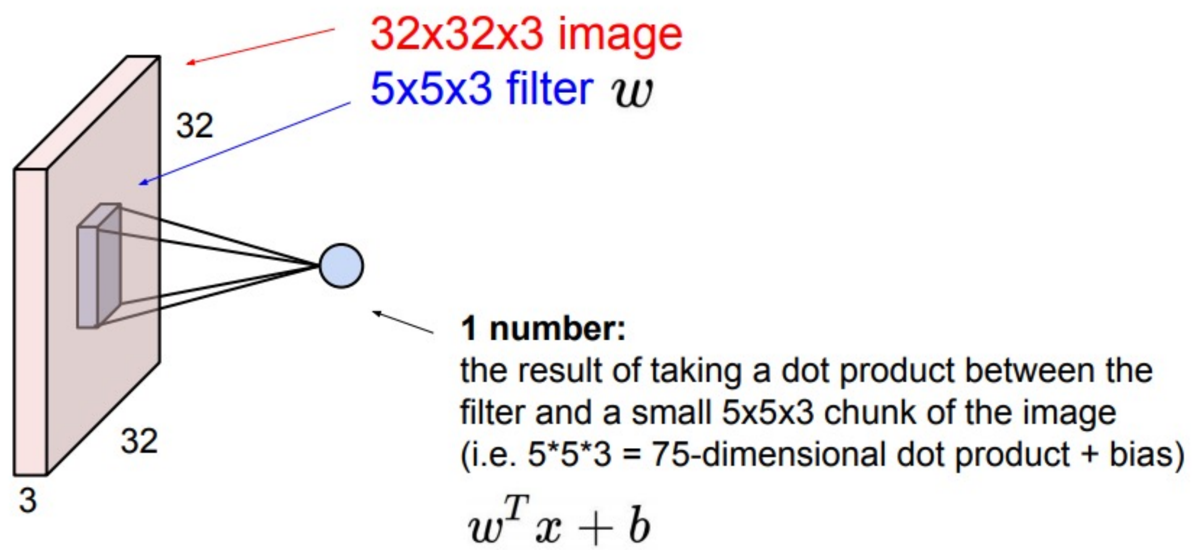
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

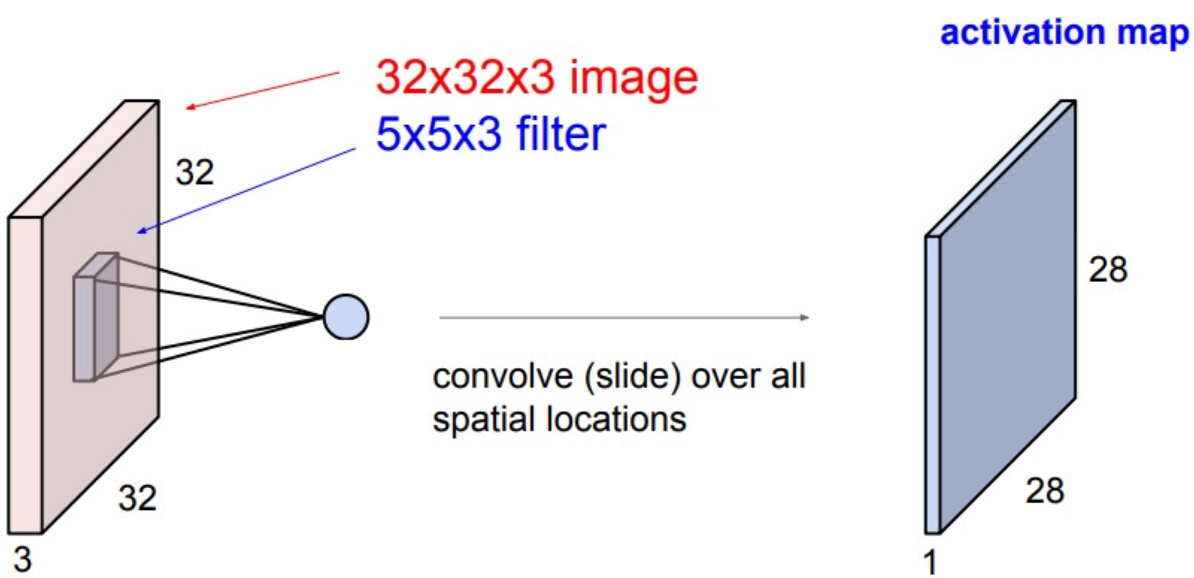
$$w^T x + b$$



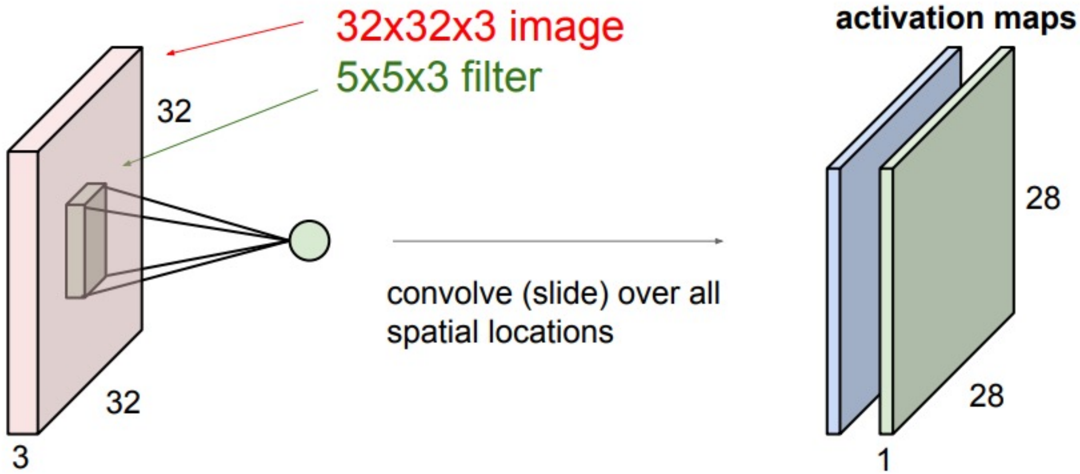
convolve (slide) over all spatial locations

Convolution Layer

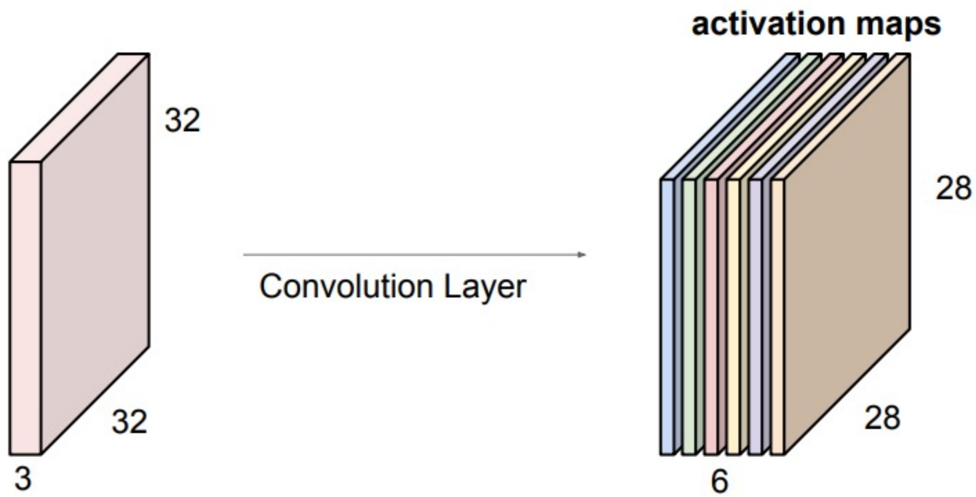




Convolution Layer

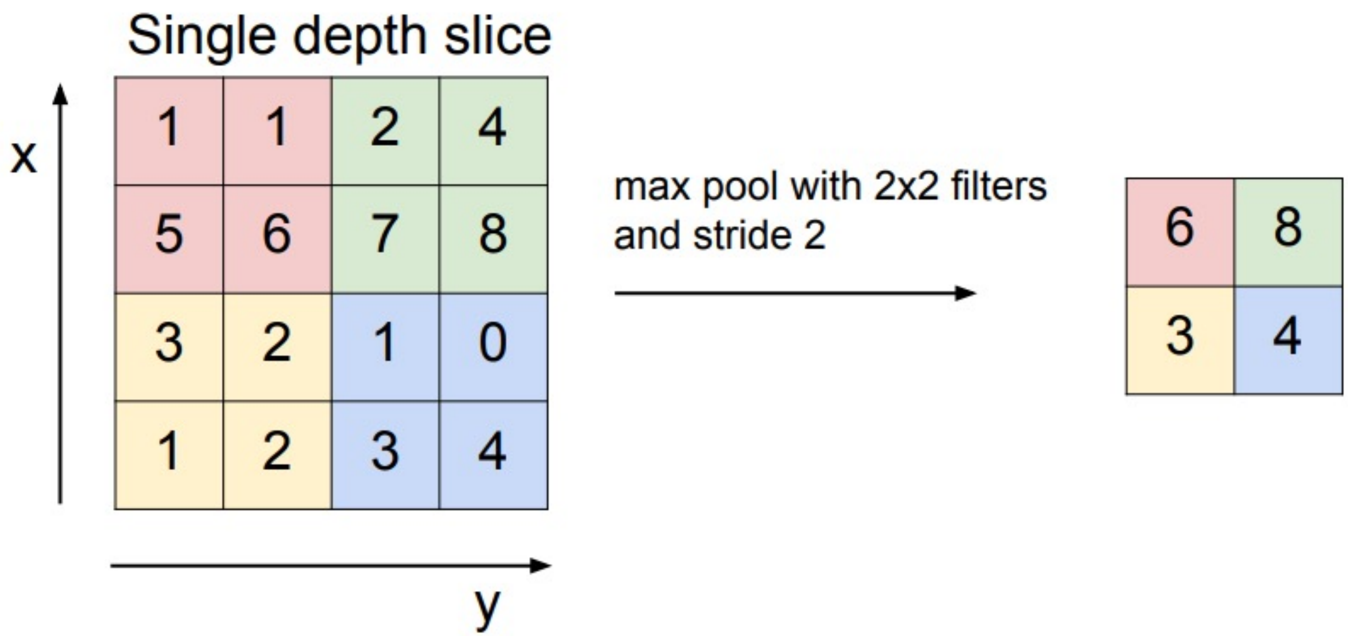


Second, green filter



6 filters

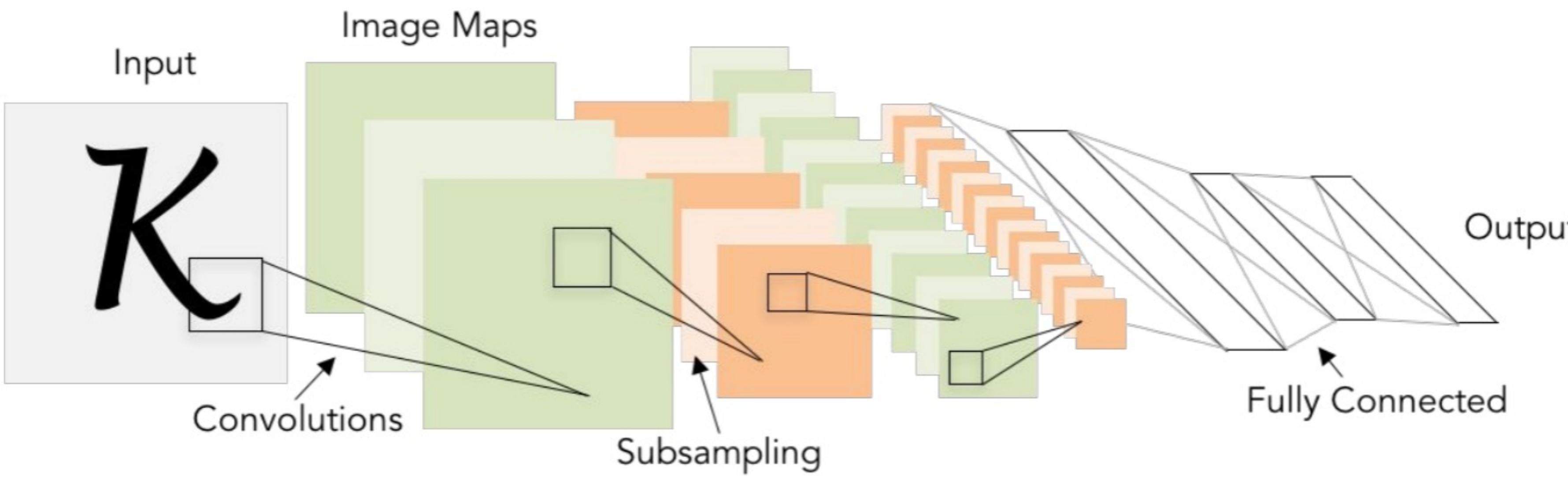
Max Pooling



- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

LeNet 5

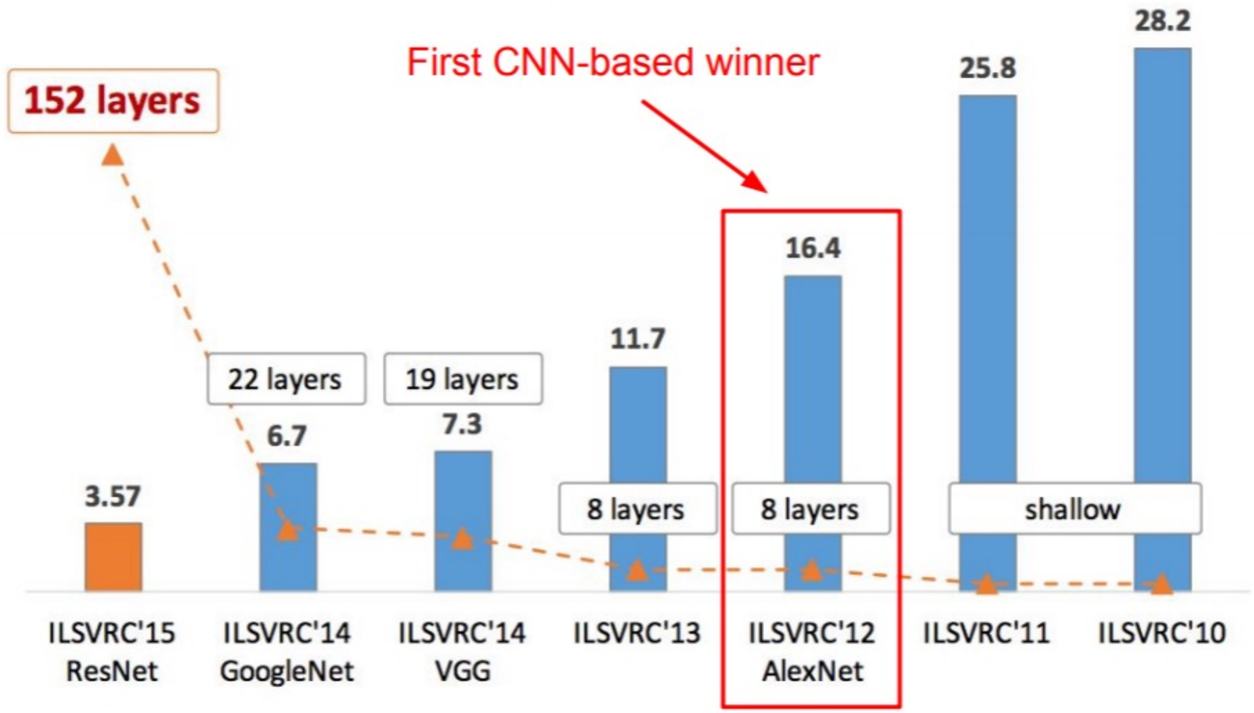
[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

History

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



VGGNet

Case Study: VGGNet

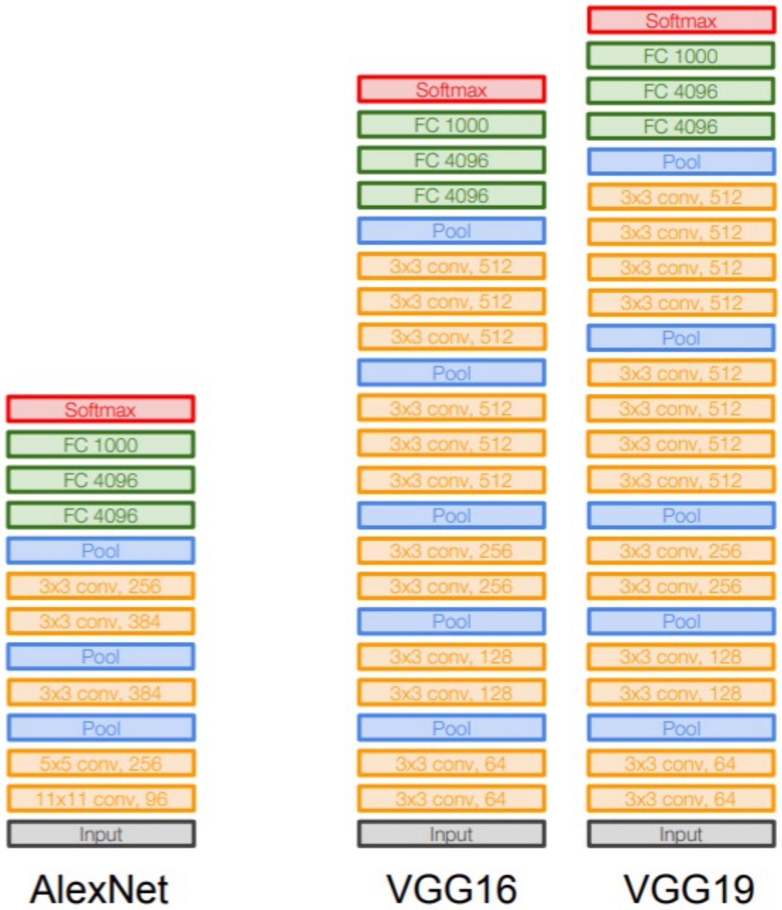
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14



138 million
parameters

How to train Neural Networks?

- Backpropagation algorithm
- David Rumelhart, Geoffrey Hinton, Ronald Williams. "Learning representations by back-propagating errors". Nature. 323 (6088): 533–536. 1986
- Applicable to both FFNN and CNN
- Extension of Gradient Descent to multi-layer neural networks

Training Neural Networks

- Training data $x_1, y_1, \dots x_N, y_N$
- One training example $x_i = (x_{i1}, \dots x_{id})$, label y_i
- One forward pass through the network
  - Compute prediction $\hat{y}_i = h(x_i)$
- Loss function for one example
  - $L(\hat{y}, y) = -[(1-y)\log(1-\hat{y}) + y\log\hat{y}]$

Cross-entropy loss

- Loss function for training data
  - $J(W, b) = \frac{1}{N}\sum_i L(\hat{y}_i, y_i)$

GD for Neural Networks

- Initialization
  - For all layers ℓ
    - Initialize $W^{[\ell]}, b^{[\ell]}$
- Backpropagation
  - Fix learning rate α
  - For all layers ℓ (starting backwards)
    - $W^{[\ell]} = W^{[\ell]} - \alpha \sum_{i=1}^{N} \frac{\partial L(\hat{y}_i, y_i)}{\partial W^{[\ell]}}$
    - $b^{[\ell]} = b^{[\ell]} - \alpha \sum_{i=1}^{N} \frac{\partial L(\hat{y}_i, y_i)}{\partial b^{[\ell]}}$

GD for Neural Networks

- Initialization
  - For all layers ℓ
    - Set $W^{[\ell]}, b^{[\ell]}$ at random
- Backpropagation
  - Fix learning rate α
  - Repeat
    - For all layers ℓ (starting backwards)
      - $W^{[\ell]} = W^{[\ell]} - \alpha \sum_{i=1}^{N} \frac{\partial L(\hat{y}_i, y_i)}{\partial W^{[\ell]}}$
      - $b^{[\ell]} = b^{[\ell]} - \alpha \sum_{i=1}^{N} \frac{\partial L(\hat{y}_i, y_i)}{\partial b^{[\ell]}}$

This is expensive!

Stochastic Gradient Descent

- Initialization
  - For all layers ℓ
    - Set $W^{[\ell]}, b^{[\ell]}$ at random
- Backpropagation
  - Fix learning rate α
  - Repeat
    - For all layers ℓ (starting backwards)
      - For all training examples x_i, y_i

$$W^{[\ell]} = W^{[\ell]} - \alpha \frac{\partial L(\hat{y}_i, y_i)}{\partial W^{[\ell]}}$$

$$b^{[\ell]} = b^{[\ell]} - \alpha \frac{\partial L(\hat{y}_i y_i)}{\partial b^{[\ell]}}$$

Incremental version of GD

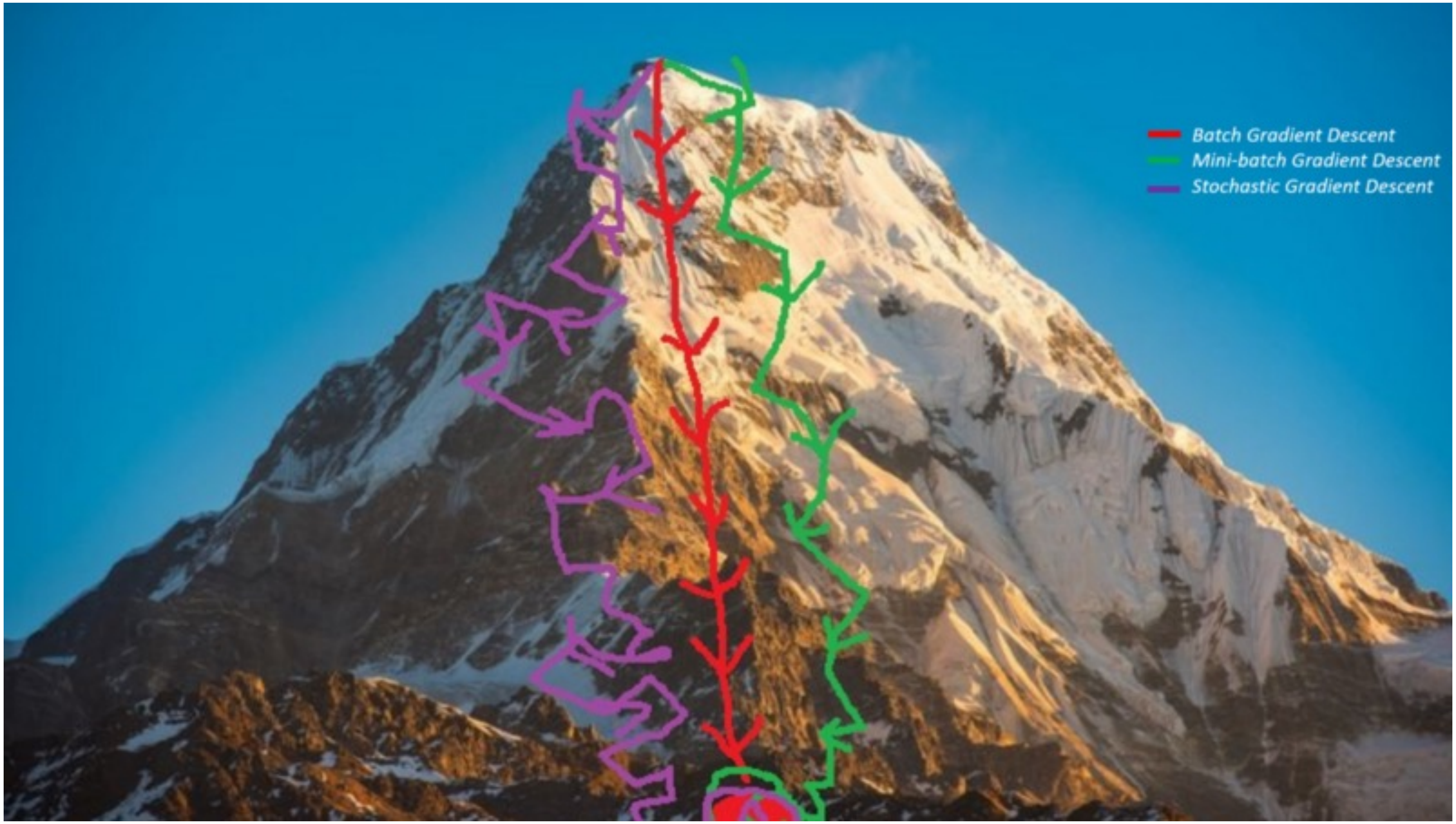
Mini-batch Gradient Descent

- Initialization
  - For all layers ℓ
    - Set $W^{[\ell]}, b^{[\ell]}$ at random
- Backpropagation
  - Fix learning rate α
  - Repeat
    - For all layers ℓ (starting backwards)
      - For all batches b of size B with training examples x_{ib}, y_{ib}

$$W^{[\ell]} = W^{[\ell]} - \alpha \sum_{i=1}^{B} \frac{\partial L(\hat{y}_{ib}, y_{ib})}{\partial W^{[\ell]}}$$

$$b^{[\ell]} = b^{[\ell]} - \alpha \sum_{i=1}^{B} \frac{\partial L(\hat{y}_{ib}, y_{ib})}{\partial b^{[\ell]}}$$

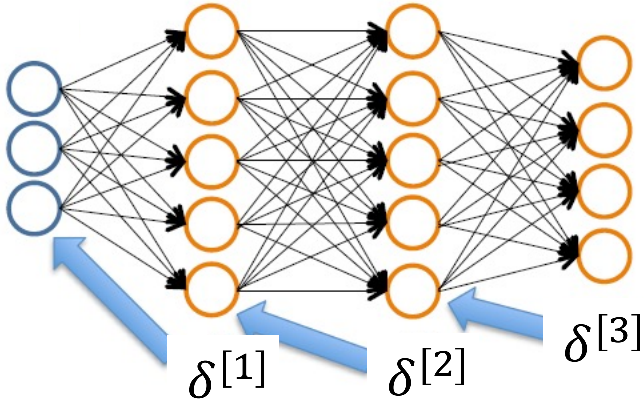
Gradient Descent Variants



Backpropagation

Let $\delta_j^{(l)}$ = "error" of node j in layer l

$$L(y,\hat{y}) = -[(1-y)\log(1-\hat{y}) + y\log\hat{y}]$$



Definitions

- $z^{[\ell]} = W^{[\ell]} a^{[\ell-1]} + b^{[\ell]}, a^{[\ell]} = g\left(z^{[\ell]}\right)$
- $\delta^{[\ell]} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}}$; Output $\hat{y} = a^{[L]} = g\left(z^{[L]}\right)$

1. For last layer L: $\delta^{[L]} = \frac{\partial L(\hat{y},y)}{\partial z^{[L]}} = \frac{\partial L(\hat{y},y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{[L]}} = \frac{\partial L(\hat{y},y)}{\partial \hat{y}} g'\left(z^{[L]}\right)$
2. For layer ℓ: $\delta^{[\ell]} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell+1]}} \frac{\partial z^{[\ell+1]}}{\partial a^{[\ell]}} \frac{\partial a^{[\ell]}}{\partial z^{[\ell]}} = \delta^{[\ell+1]} W^{[\ell+1]} g'(z^{[\ell]})$
3. Compute parameter gradients

- $\frac{\partial L(\hat{y},y)}{\partial W^{[\ell]}} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}} \frac{\partial z^{[\ell]}}{\partial W^{[\ell]}} = \delta^{[\ell]} a^{[\ell-1]T}$
- $\frac{\partial L(\hat{y},y)}{\partial b^{[\ell]}} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}} \frac{\partial z^{[\ell]}}{\partial b^{[\ell]}} = \delta^{[\ell]}$

Training NN with Backpropagation

Given training set $(x_1, y_1), \ldots, (x_N, y_N)$
Initialize all parameters $W^{[\ell]}, b^{[\ell]}$ randomly, for all layers ℓ
Loop

Set $\Delta_{ij}^{[l]} = 0$, for all layers l and indices i, j

EPOCH

For each training instance (x_k, y_k):

Compute $a^{[1]}, a^{[2]}, \ldots, a^{[L]}$ via forward propagation

Compute errors $\delta^{[L]}, \delta^{[L-1]}, \ldots \delta^{[1]}$

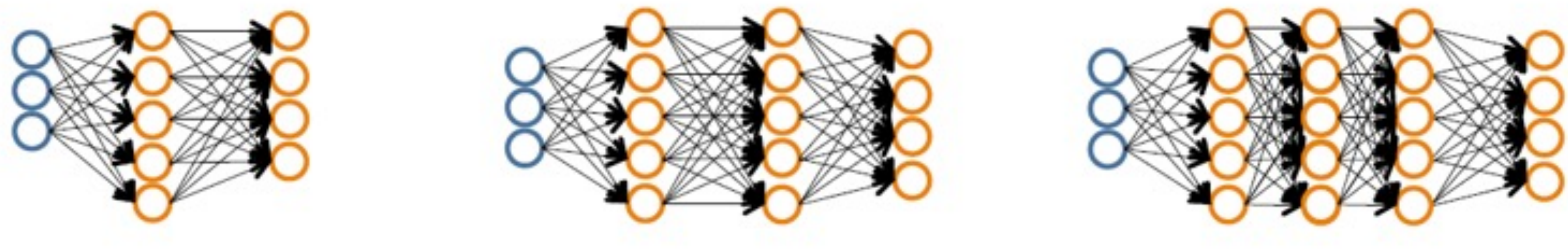
Aggregate gradients $\Delta_{ij}^{[l]} = \Delta_{ij}^{[l]} + a_j^{[l-1]} \delta_i^{[l]}$

Update weights via gradient step

- $W_{ij}^{[\ell]} = W_{ij}^{[\ell]} - \alpha \Delta_{ij}^{[\ell]}$
- Similarly for $b_{ij}^{[\ell]}$

Until weights converge or maximum number of epochs is reached

Overfitting



- The larger the network, the higher the capacity (more model parameters)
- But also more prone to overfitting!

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}} + \underbrace{\lambda R(W)}_{\text{Regularization: Prevent the model from doing } \textit{too} \text{ well on training data}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

λ = regularization strength (hyperparameter)

<u>L2 regularization:</u> $R(W) = \sum_k \sum_l W_{k,l}^2$ → Weight decay

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

- When computing gradients of loss function, regularization term needs to be taken into account

Dropout

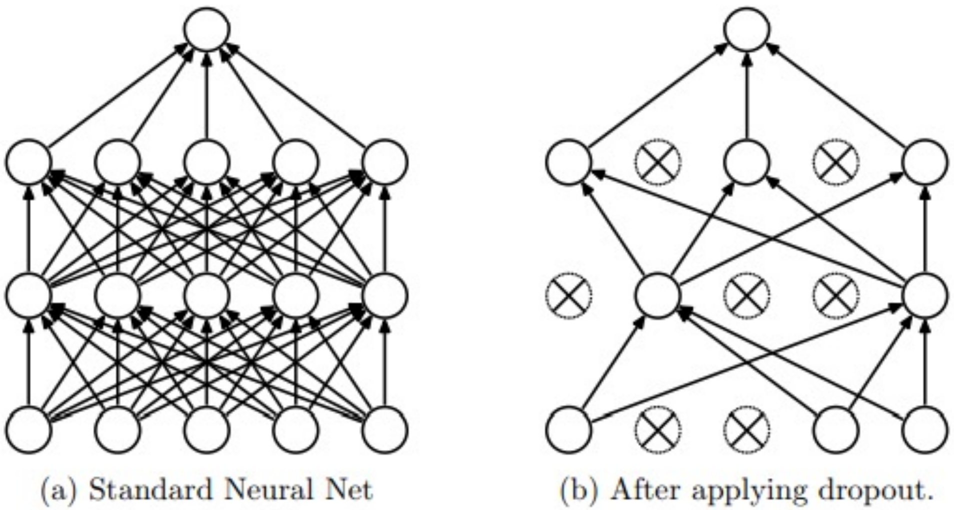


Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

- At training time, sample a sub-network per epoch (batch) and learn weights
  - Keep each neuron with probability p
- At testing time, all neurons are there, but multiply weight by a factor of p
- Srivastava et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15, 2014

Comparing classifiers

Algorithm	Interpretable	Model size	Predictive accuracy	Training time	Testing time
Logistic regression	High	Small	Lower	Low	Low
kNN	Medium	Large	Lower	No training	High
LDA	Medium	Small	Lower	Low	Low
Decision trees	High	Medium	Lower	Medium	Low
Ensembles	Low	Large	High	High	High
Naïve Bayes	Medium	Small	Lower	Medium	Low
SVM	Medium	Small	Lower	Medium	Low
Neural Networks	Low	Large	High	High	Low

Summary

- Deep Learning performs well for datasets with certain structure: images, text, speech
  - Learns hierarchical feature representations
- Multiple types of NN architectures
  - Feed-forward neural networks
  - Convolutional neural networks (applicable to images)
  - Recurrent neural networks, Transformers (text)
- Training most NN architectures via backpropagation
- Regularization (weight decay, dropout) to avoid overfitting