DS 4400 Final Exam Review

David Liu

April 12, 2024

Logistics

Exam on Tuesday April 23 from 1 – 3pm in Snell Engineering Center Room 108.

Exam will cover lectures from February 16 – April 2

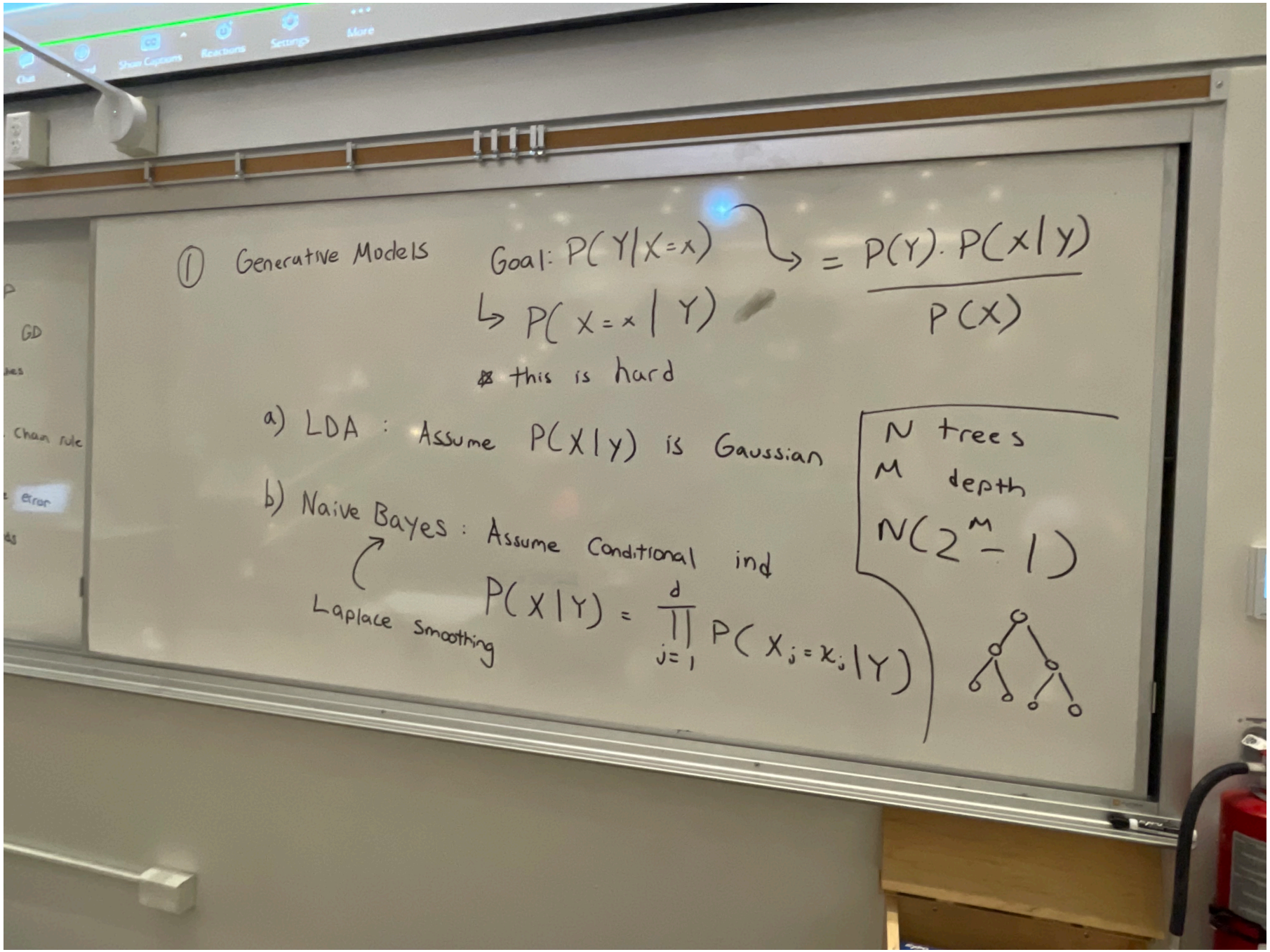
- Ethics in AI lecture not included.

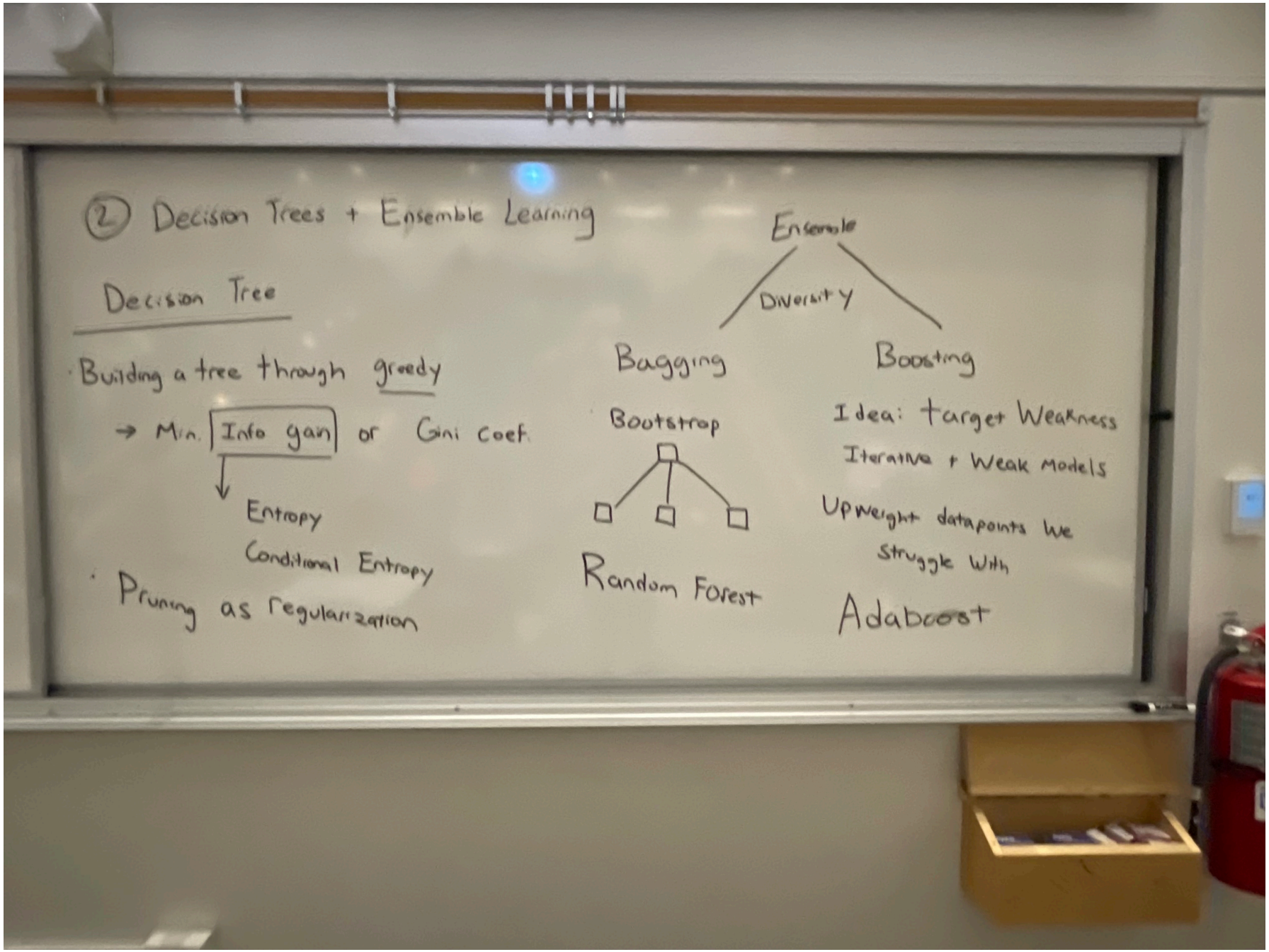
1-page cheat sheet is allowed. No restrictions on handwritten vs printed.

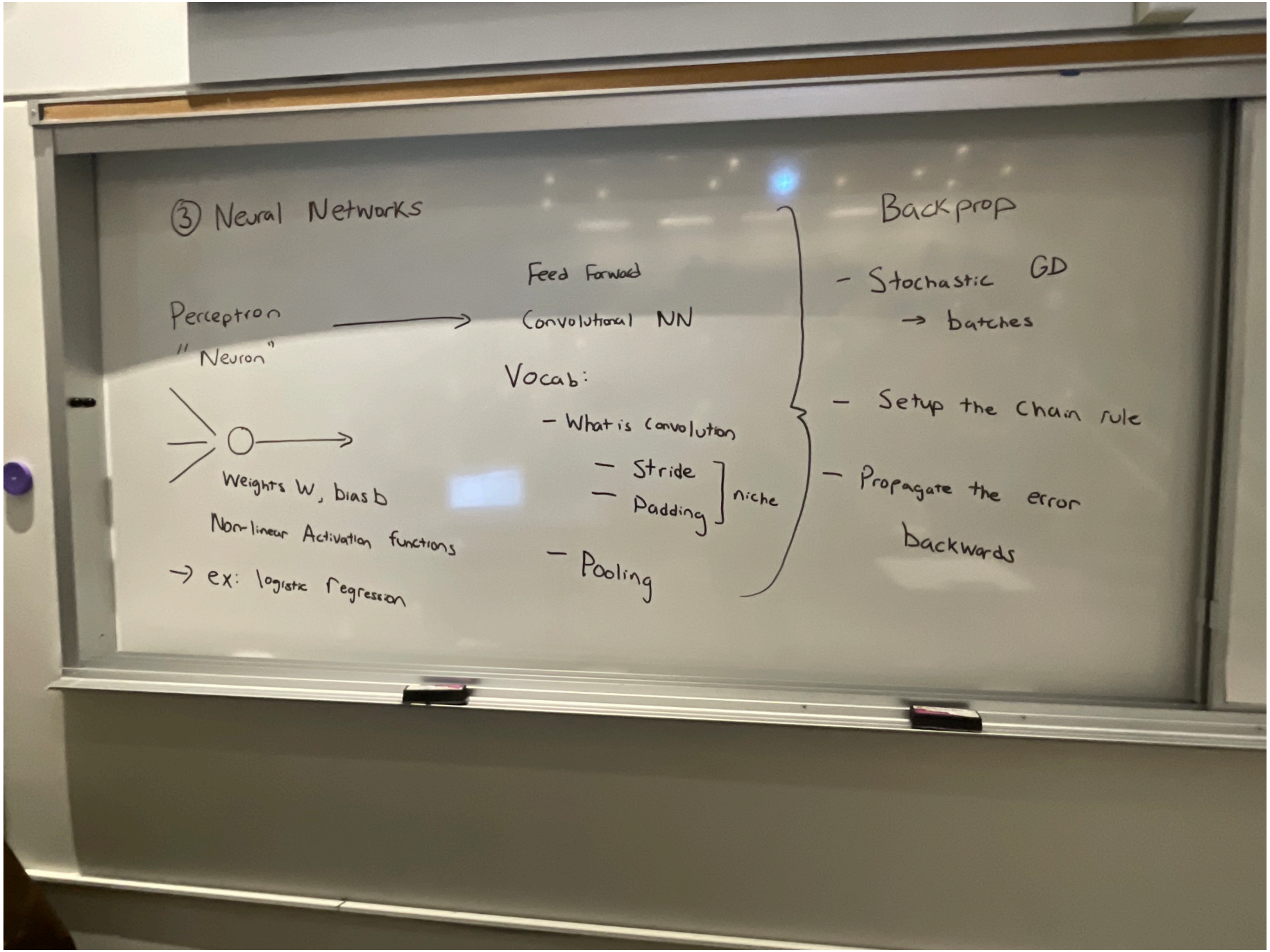
Exam will be five questions. One question will be conceptual and remaining four will be analytical.

List of Topics

- Generative Modeling
  - LDA
  - Naïve Bayes
    - Laplace Smoothing
- Decision Trees
  - Entropy, Conditional Entropy, Information Gain
- Ensemble Learning
  - Bagging
    - Random Forests
  - Boosting
    - Adaboost
- Neural Networks
  - Perceptrons
  - Convolutional Neural Networks
    - Convolutions, strides, pooling
  - Backpropagation
  - Regularization of neural networks


Chat
Show Captions
Reactions
Settings
More
GD
Chain rule
Error
① Generative Models
Goal: $P(Y|X=x) = \frac{P(Y) \cdot P(x|y)}{P(X)}$
↳ $P(X=x|Y)$
* this is hard
a) LDA : Assume $P(X|y)$ is Gaussian
b) Naive Bayes : Assume Conditional ind
$P(X|Y) = \prod_{j=1}^{d} P(X_j = x_j|Y)$
Laplace Smoothing
N trees
M depth
$N(2^M - 1)$


② Decision Trees + Ensemble Learning
Decision Tree
· Building a tree through greedy
→ Min. Info gain or Gini coef.
Entropy
Conditional Entropy
· Pruning as regularization
Ensemble
Diversity
Bagging
Boosting
Bootstrap
Random Forest
Idea: target Weakness
Iterative + Weak Models
Upweight datapoints we struggle with
Adaboost


③ Neural Networks
Perceptron
" Neuron"
Weights W, bias b
Non-linear Activation functions
→ ex: logistic regression
Feed Forward
Convolutional NN
Vocab:
- What is convolution
- Stride
- Padding
niche
- Pooling
Backprop
- Stochastic GD
→ batches
- Setup the chain rule
- Propagate the error backwards

Generative Modeling - LDA

Generative vs Discriminative

- Generative model
  - Given X and Y, learns the joint probability $P(X, Y)$
  - Can generate more examples from distribution
  - Examples: LDA, Naïve Bayes, language models (GPT-2, GPT-3, BERT)

- Discriminative model
  - Given X and Y, learns a decision function for classification

Generative classifiers based on Bayes Theorem

$$P(A \mid B) = \frac{P(B \mid A) \times P(A)}{P(B)}$$

- Exactly the process we just used
- The most important formula in probabilistic machine learning

(Super Easy) Derivation:

$$P(A \wedge B) = P(A \mid B) \times P(B)$$
$$P(B \wedge A) = P(B \mid A) \times P(A)$$

these are the same

Just set equal...

$$P(A \mid B) \times P(B) = P(B \mid A) \times P(A)$$

and solve...

Bayes, Thomas (1763) An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society of London,* **53:370-418**

LDA

$$\Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}.$$

Assume $f_k(x)$ is Gaussian!
Unidimensional case (d=1)

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{1}{2\sigma_k^2}(x - \mu_k)^2\right)$$

$$p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x - \mu_k)^2\right)}{\sum_{l=1}^{K} \pi_l \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x - \mu_l)^2\right)}.$$

Assumption: $\sigma_1 = \ldots \sigma_k = \sigma$

LDA Training and Testing

Given training data $(x_i, y_i), i = 1, \ldots, n, y_i \in \{1, \ldots, K\}$

1. Estimate sample mean and variance

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i$$

$$\hat{\sigma}^2 = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2$$

2. Estimate prior

$$\hat{\pi}_k = n_k / n.$$

Given testing point x, predict k that maximizes:

$$p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x-\mu_k)^2\right)}{\sum_{l=1}^{K} \pi_l \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x-\mu_l)^2\right)}.$$

LDA vs Logistic Regression

- Logistic regression computes directly $\Pr[Y = 1|X = x]$ by assuming sigmoid function
  - Uses Maximum Likelihood Estimation
  - Discriminative Model
- LDA uses Bayes Theorem to estimate it
  - Estimates mean, co-variance, and prior from training data
  - Generative model
  - Assumes Gaussian distribution for $f_k(x) = \Pr[X = x|Y = k]$
- Which one is better?
  - LDA can be sensitive to outliers
  - LDA works well for Gaussian distribution
  - Logistic regression is more complex to solve, but more expressive

Generative Modeling – Naïve Bayes

Naïve Bayes Classifier

Idea: Use the training data to estimate

$P(X \mid Y)$ and $P(Y)$.

Then, use Bayes rule to infer $P(Y|X_{\text{new}})$ for new data

---

Easy to estimate from data

Impractical, but necessary

$$\mathrm{P}[Y = k|X = x] = \frac{\mathrm{P}[Y = k]\mathrm{P}[X_1 = x_1 \wedge \cdots \wedge X_d = x_d|Y = k]}{\mathrm{P}[X_1 = x_1 \wedge \cdots \wedge X_d = x_d]}$$

Unnecessary, as it turns out

Using the Naïve Bayes Classifier

- Now, we have

$$\mathrm{P}[Y = k|X = x] = \frac{\mathrm{P}[Y = k]\mathrm{P}[X_1 = x_1 \wedge \cdots \wedge X_d = x_d|Y = k]}{\mathrm{P}[X_1 = x_1 \wedge \cdots \wedge X_d = x_d]}$$

This is constant for a given instance, and so irrelevant to our prediction

  - In practice, we use log-probabilities to prevent underflow

- To classify a new point $\mathbf{x}$,

$$h(\mathbf{x}) = \arg\max_{y_k} P(Y = k) \prod_{j=1}^{d} P(X_j = x_j \mid Y = k)$$

x_j: j^{th} attribute value of $\mathbf{x}$

Naïve Bayes Classifier

TRAIN

- For each class label k
  1. Estimate prior $\pi_k = P[Y = k]$ from the data
  2. For each value v of attribute X_j
     - Estimate $\mathrm{P}[X_j = v | Y = k]$

TEST on INPUT $x = (x_1, \ldots, x_d)$

- For every k, compute the probabilities
  - $p_k = P[Y = k] \prod_{j=1}^{d} P[X_j = x_j \,|Y = k]$
- Classify x to the class k that maximizes p_k

Laplace Smoothing

- Notice that some probabilities estimated by counting might be zero
  - Possible overfitting!

- Fix by using Laplace smoothing:
  - Adds 1 to each count

$$P(X_j = v \mid Y = k\ ) = \frac{\mathrm{c}_v + 1}{\sum_{v' \in \mathrm{values}(X_j)} \mathrm{c}_{v'} + |\mathrm{values}(X_j)|}$$

where

- c_v is the count of training instances with a value of v for attribute j and class label k
- $|\mathrm{values}(X_j)|$ is the number of values X_j can take on

Decision Trees

Learning Decision Trees

- Learning the simplest (smallest) decision tree is an NP-complete problem [Hyafil & Rivest '76]
- Resort to a greedy heuristic:
  - Start from empty decision tree
  - Split on **next best attribute (feature)**
  - Recurse

Information Gain

X = College Major

Y = Likes "Gladiator"

X	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
History	No
Math	Yes

Definition of Information Gain:

IG(Y|X) = **I must transmit** *Y.* **How many bits on average would it save me if both ends of the line knew** *X*?

$IG(Y|X) = H(Y) - H(Y|X)$

Example:

- **H(Y) = 1**
- **H(Y|X) = 0.5**
- **Thus IG(Y|X) = 1 – 0.5 = 0.5**

Relevance for decision trees

- Multiple features $X_1, \ldots, X_d$
- Label Y: Initial entropy $H(Y)$
- How much each feature X_i helps explain uncertainty in Y
  - Compute Information gain

  $IG(Y|X_i) = H(Y) - H(Y|X_i)$
- Select feature that maximizes IG
- Then recurse on the remaining set of features

Learning Decision Trees

- Start from empty decision tree
- Split on **next best attribute (feature)**
  - Use, for example, information gain to select attribute:

$$\arg\max_i IG(X_i) = \arg\max_i H(Y) - H(Y \mid X_i)$$

- Recurse

ID3 algorithm uses Information Gain
Information Gain reduces uncertainty on Y

Ensemble Learning

Ensemble Learning

Consider a set of classifiers h_1, ..., h_L

Idea: construct a classifier $H(\mathbf{x})$ that combines the individual decisions of h_1, ..., h_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

Bagging

- Leo Breiman (1994)
- Take repeated bootstrap samples from training set *D*
- *Bootstrap sampling*: Given set *D* containing *N* training examples, create *D'* by drawing *N* examples at random with replacement from *D*.

- Bagging:
  - Create *k* bootstrap samples D_1 … D_k.
  - Train distinct classifier on each D_i.
  - Classify new instance by majority vote / average.

Random Forest Algorithm

1. For $b = 1$ to B:

   (a) Draw a 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 m 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 $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

If m=p, this is equivalent to Bagging with Decision Trees as base learner

AdaBoost

- A meta-learning algorithm with great theoretical and empirical performance
- Turns a base learner (i.e., a “weak hypothesis”) into a high performance classifier
- Creates an ensemble of weak hypotheses by repeatedly emphasizing mispredicted instances

Adaptive Boosting
Freund and Schapire 1997

Boosting [Shapire '89]

- **Idea:** given a weak learner, run it multiple times on (reweighted) training data, then let learned classifiers vote
- On each iteration t:
  - weight each training example by how incorrectly it was classified
  - Learn a weak hypothesis – h_t
  - A strength for this hypothesis – β_t
- Final classifier: $H(x) = sign(\sum \beta_t h_t(x))$

Convergence bounds with minimal assumptions on weak learner

If each weak learner h_t is slightly better than random guessing ($\varepsilon_t < 0.5$), then training error of AdaBoost decays exponentially fast in number of rounds T.

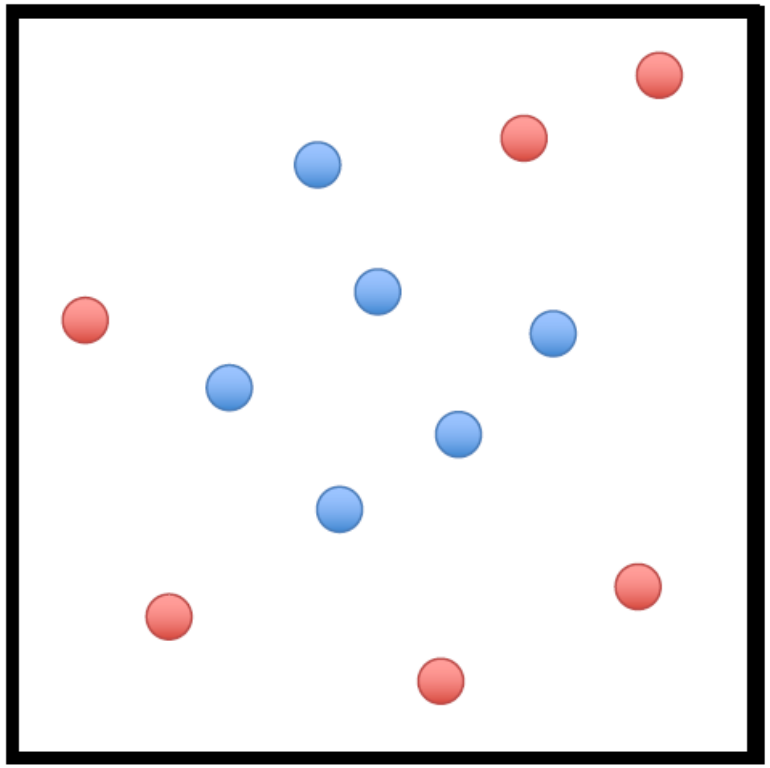
AdaBoost

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$



- Size of point represents the instance's weight

Bagging vs Boosting

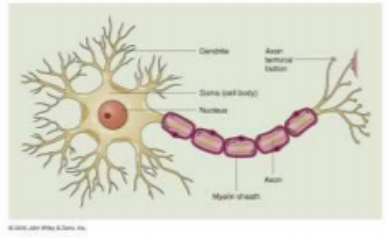
Bagging	vs.	Boosting
Resamples data points		Reweights data points (modifies their distribution)
Weight of each classifier is the same		Weight is dependent on classifier's accuracy
Only variance reduction		Both bias and variance reduced – learning rule becomes more complex with iterations

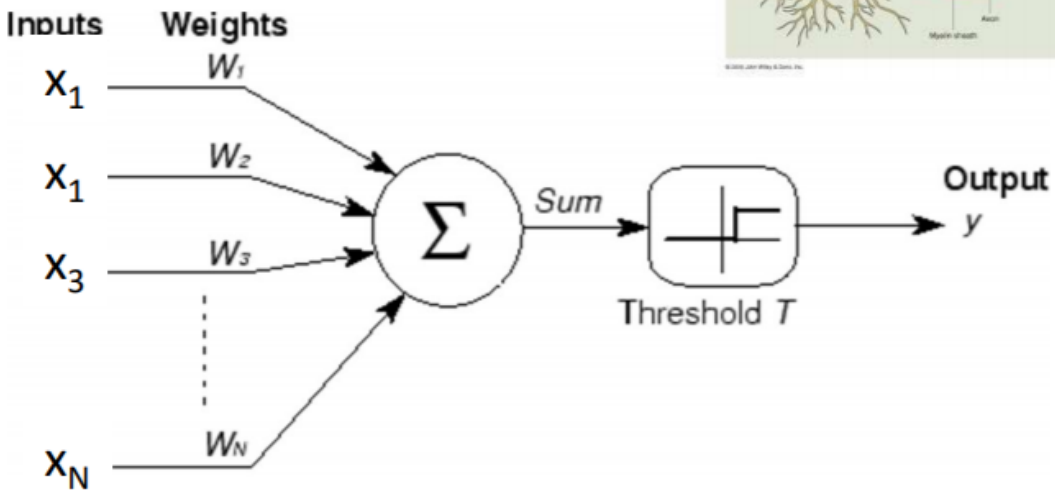
Applicable to complex models with low bias, high variance

Applicable to weak models with high bias, low variance

Feed Forward Neural Networks

Perceptron

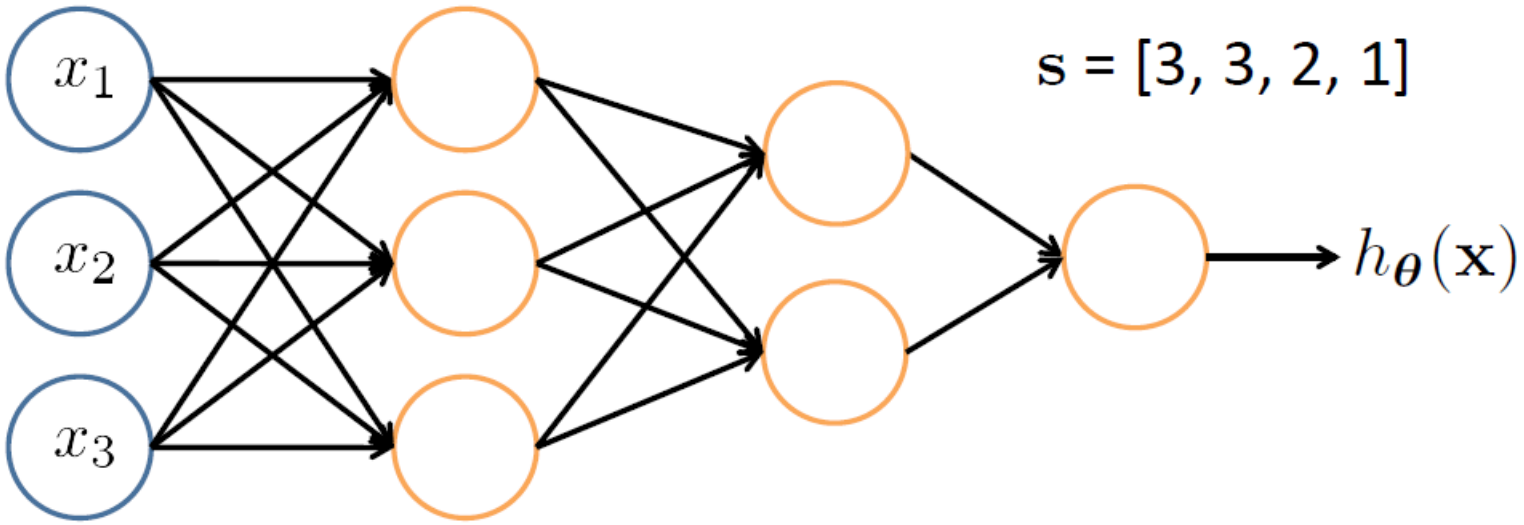




$$y = \begin{cases} 1 \; if \; \sum_i w_i x_i \geq T \\ 0 \; else \end{cases}$$

- A threshold unit
  - "Fires" if the weighted sum of inputs exceeds a threshold

Feed-Forward Networks

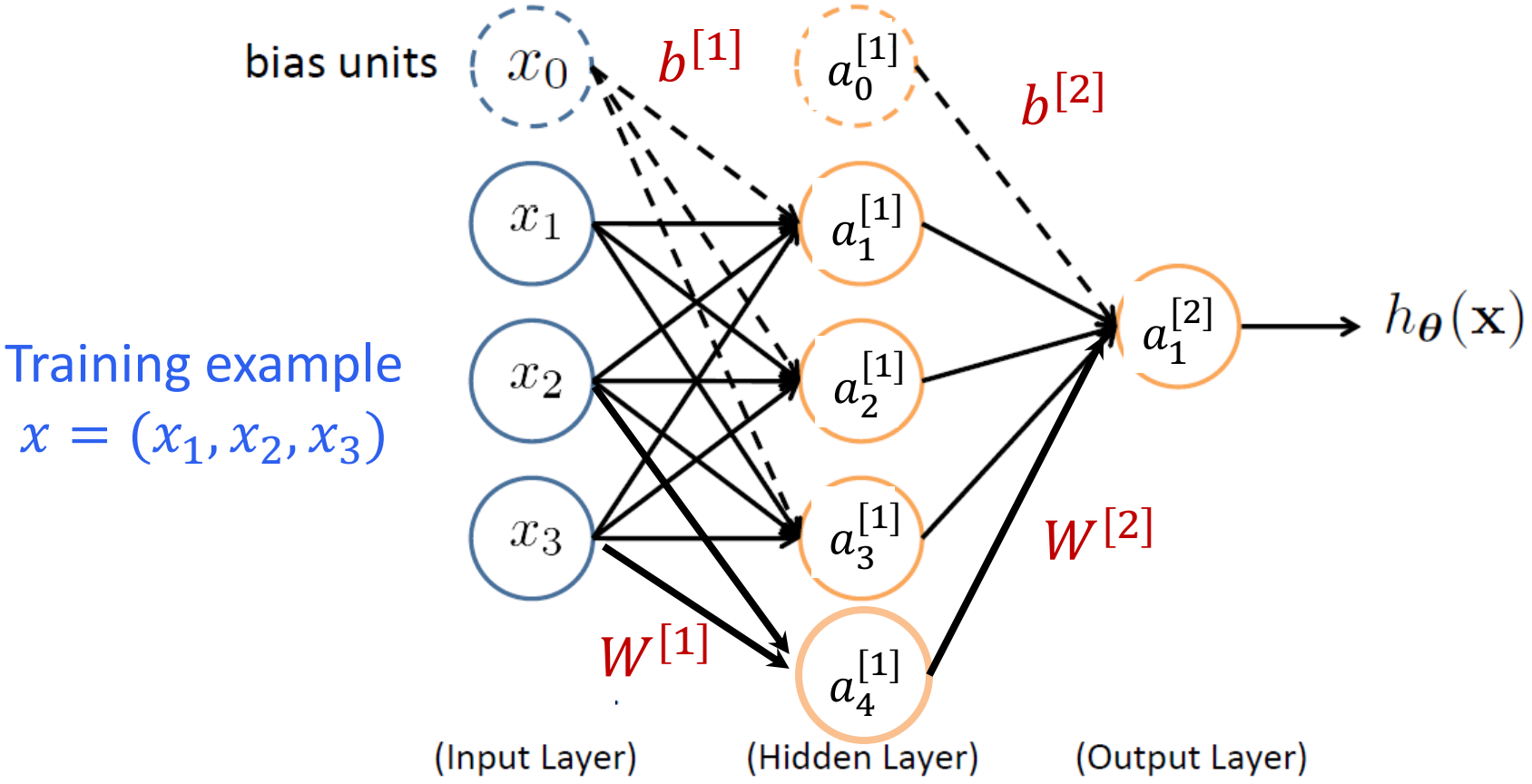


L denotes the number of layers

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

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

Feed-Forward Neural Network

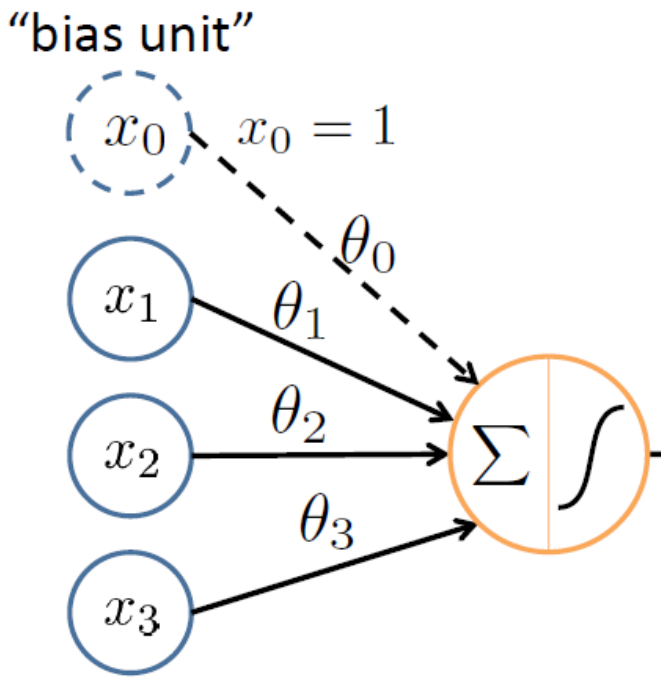


Training example
$x = (x_1, x_2, x_3)$

No cycles

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

Logistic Unit: A simple NN



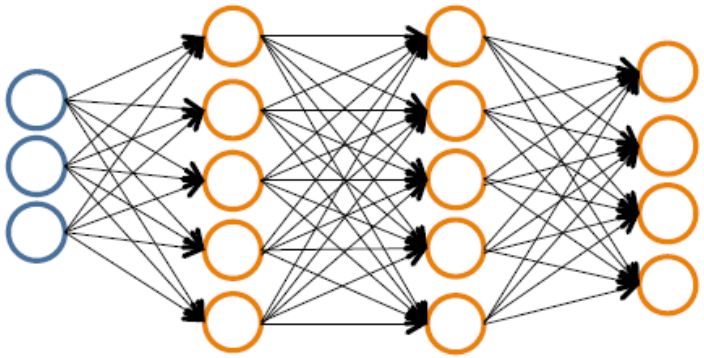
$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}\right) = g(w^T x + b)$$
$$= \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}}}$$

Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

No hidden 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)

Binary classification

y = 0 or 1

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

Sigmoid

Multi-class classification (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

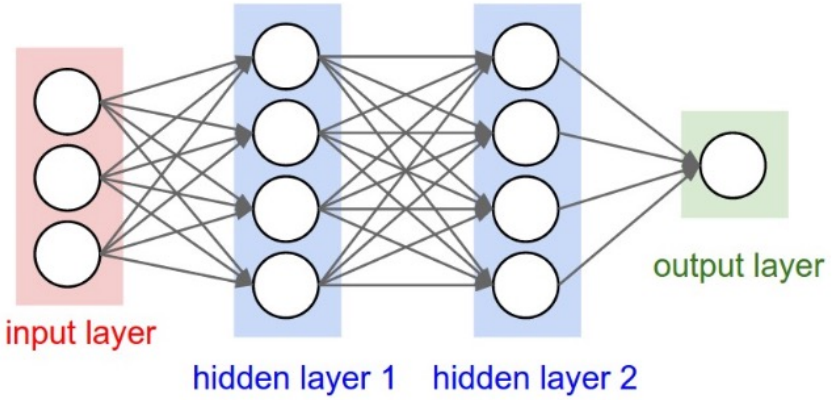
K output units $(s_{L-1} = K)$

Softmax

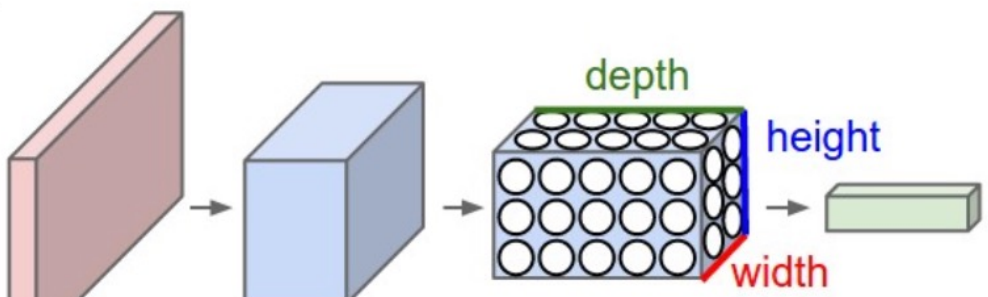
Convolutional Neural Networks

Convolutional Neural Networks

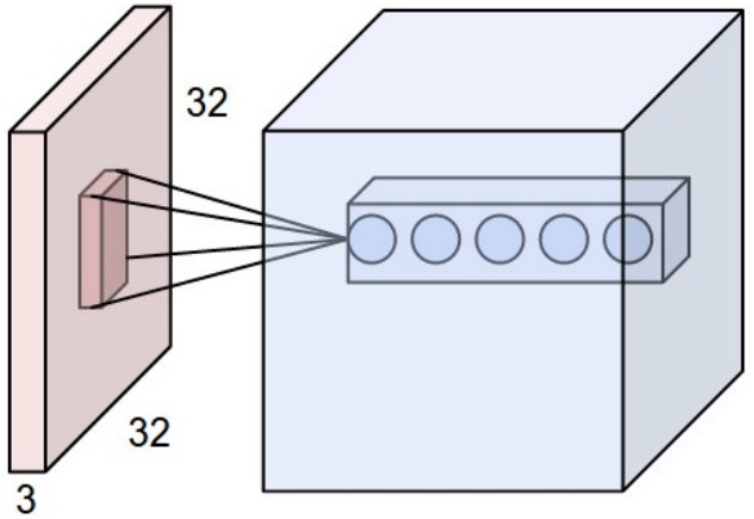
Feed-forward network



Convolutional network



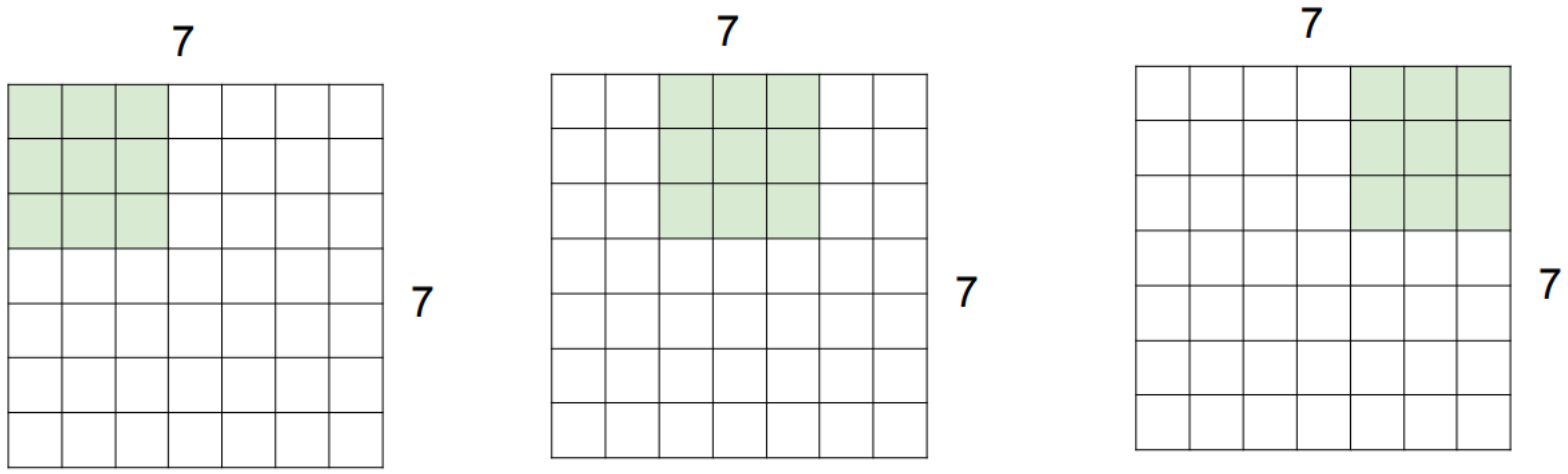
Filter



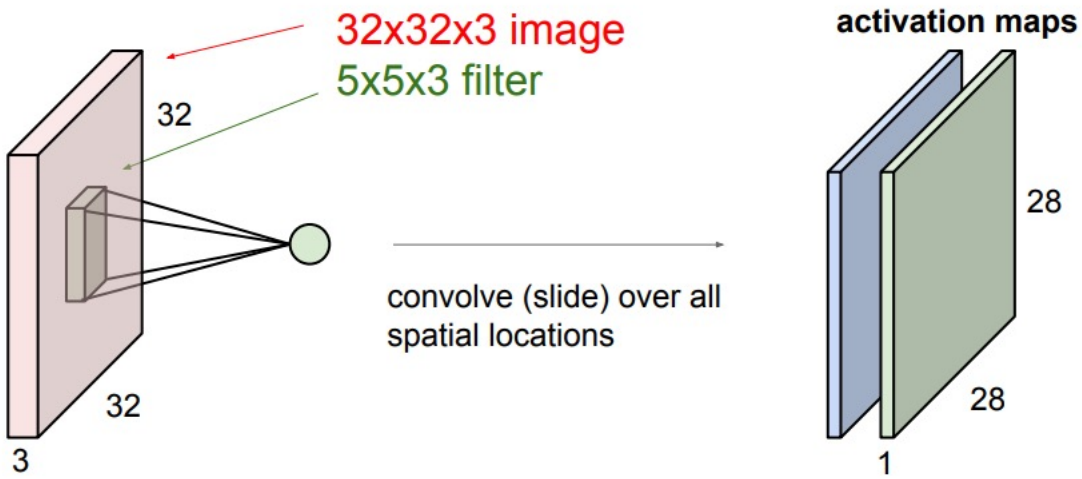
https://www.youtube.com/watch?v=f0t-OCG79-U

Convolutions with stride

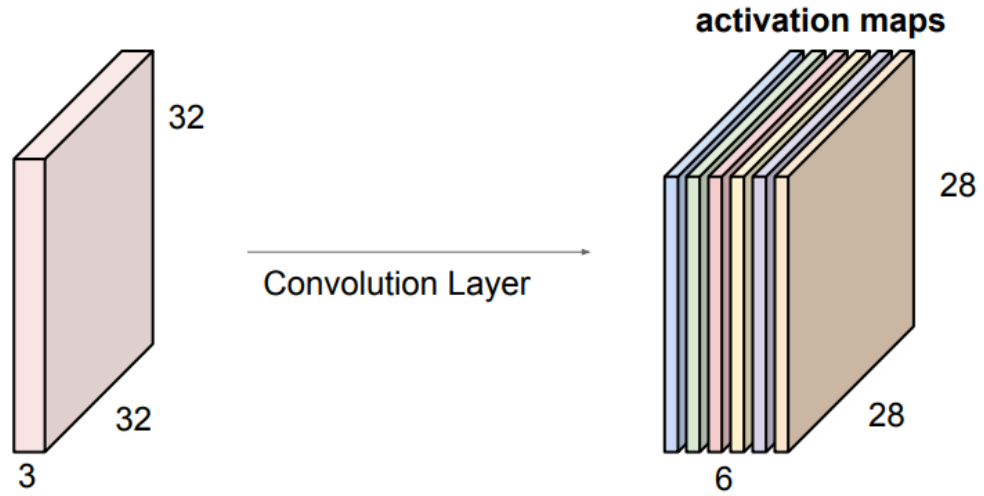
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**



Convolution Layer



Second, green filter



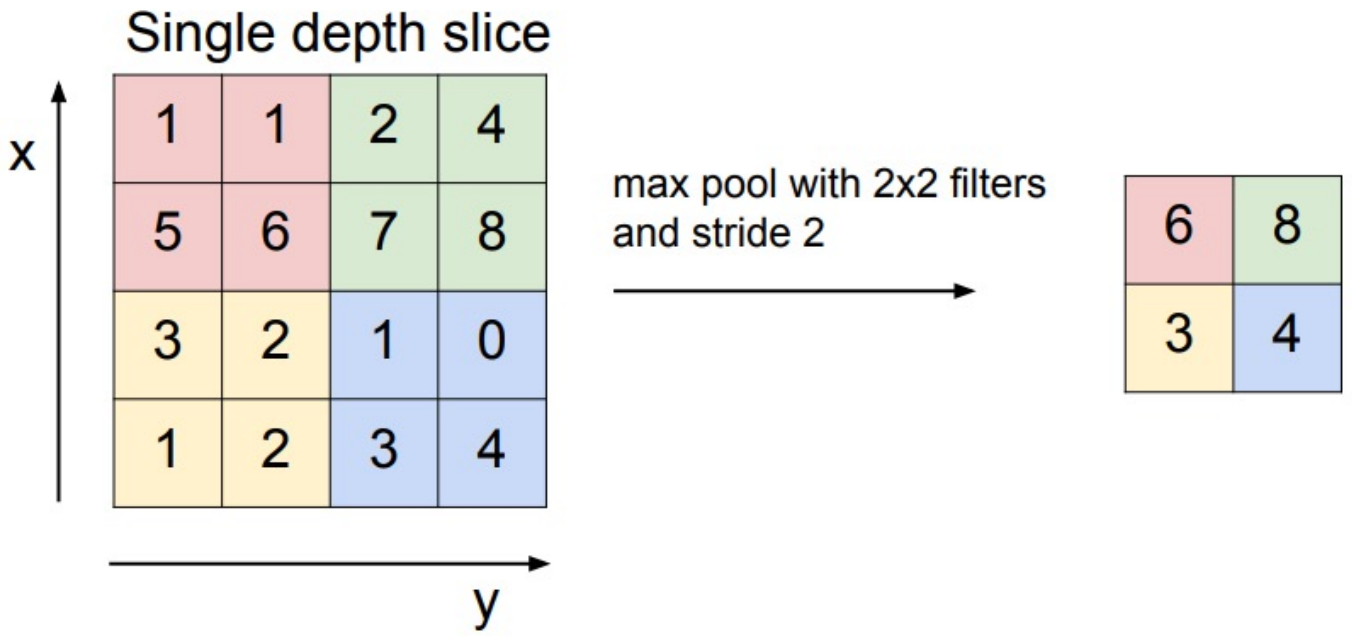
6 filters

Summary: Convolution Layer

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

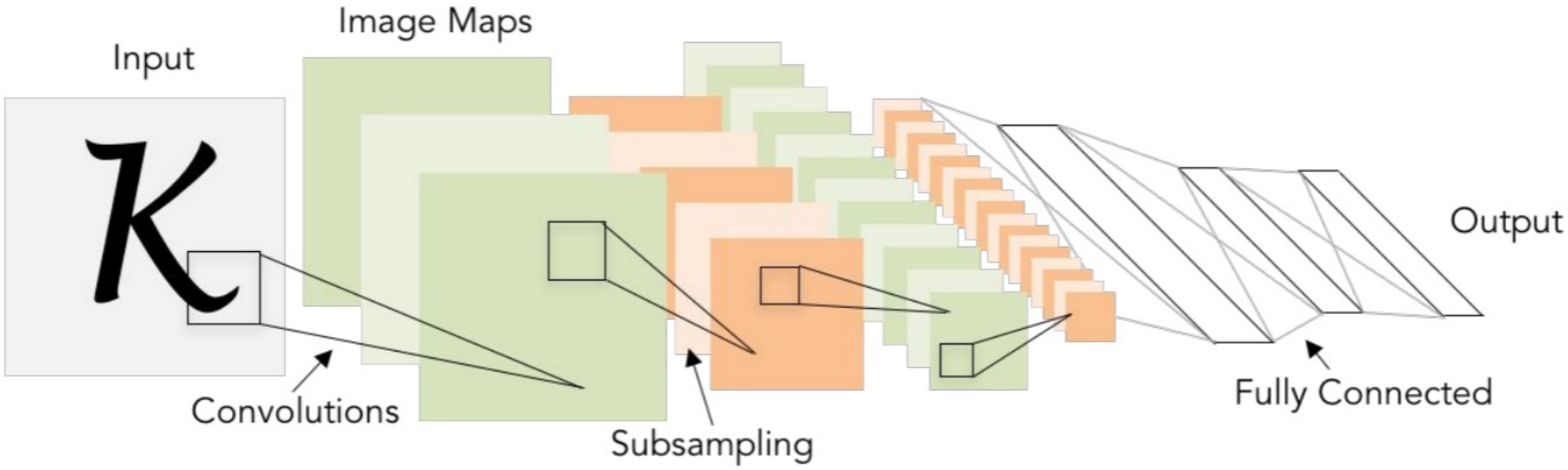
Max Pooling



- 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]

Backpropagation

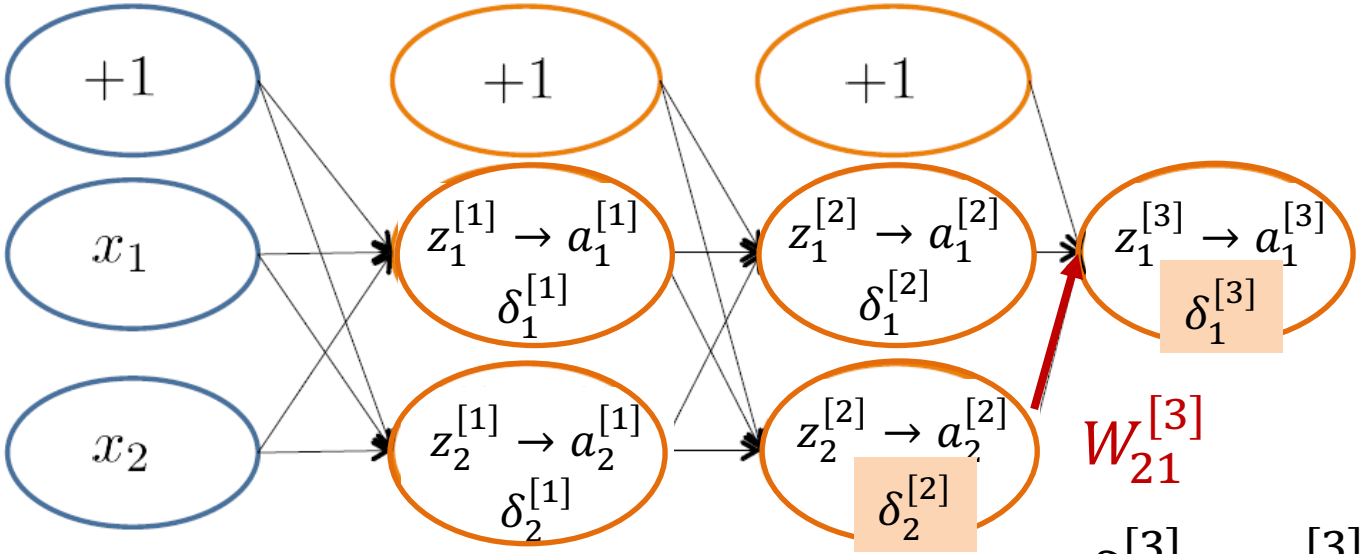
Mini-batch Gradient Descent

- 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}

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

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

Backpropagation Intuition



$W_{21}^{[3]}$

$\delta_1^{[3]} \approx a_1^{[3]} - y$

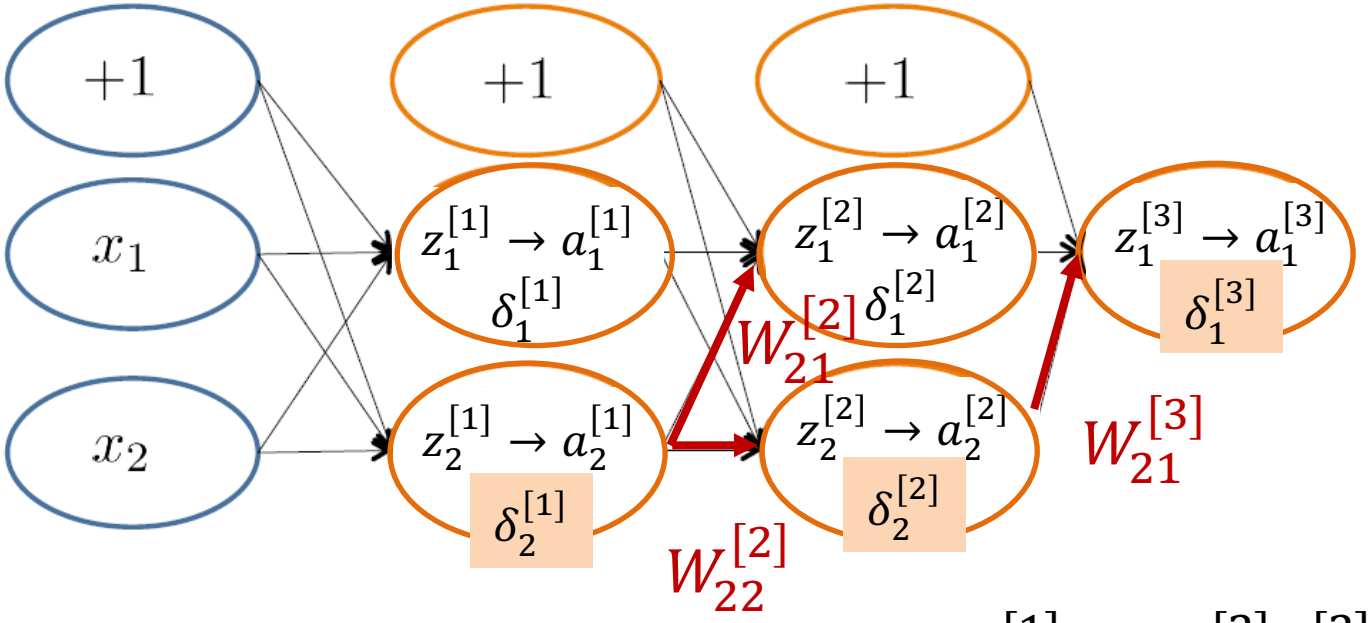
$\delta_2^{[2]} \approx \delta_1^{[3]} W_{21}^{[3]}$

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

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Backpropagation Intuition



$$\delta_2^{[1]} \approx W_{21}^{[2]}\delta_1^{[2]} + W_{22}^{[2]}\delta_2^{[2]}$$

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

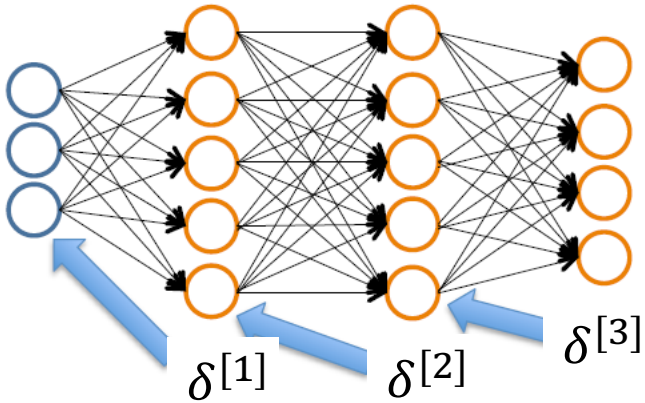
Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

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: $\delta^{[L]} = \frac{\partial L(\hat{y},y)}{\partial z^{[L]}} = \frac{\partial L(\hat{y},y)}{\hat{\partial}\,\hat{y}} \frac{\partial \hat{y}}{\hat{\partial}\, \boldsymbol{z}^{[\boldsymbol{L}]}} = \frac{\partial L(\hat{y},y)}{\hat{\partial}\,\hat{y}} g'\left(z^{[L]}\right)$
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 $\delta^{[L]} = a^{[L]} - y_k, \delta^{[L-1]}, \ldots \delta^{[1]}$

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

Update weights via gradient step

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

Until weights converge or maximum number of epochs is reached

Questions