Data Mining & Machine Learning

CS57300
Purdue University

April 26, 2018

Review Topics

- Elements of data mining
- Controlling the Model Space
  - Hypothesis testing
- Multi-armed Bandits
- Logistic Regression (LR)
- Decision trees
  - Gradient Boosted Decision Trees
- Naïve Bayes
- Ensemble Methods
- Model Assessment & Validation (+ Hypothesis test + Learning curves)
- Dimensionality reduction
- Collaborative Filtering
- Neural Networks
- Neural networks for structured data

Warning: These slides are not a comprehensive review of the course material

Elements of data mining

Principles (Lecture 5)

- Define your task and choose your data.

- Decisions:
  - **Knowledge representation/model space** (i.e., set of possible models or patterns considered by the algorithm)
  - **Data**: What is your data like? What do values represent?
  - **Model complexity**: Is the model too complex for your data? Is the model too simple?
  - **Validation**: Is your algorithm doing a good job?
  - **Testing**: Formulate and test a **hypothesis** about why your algorithmic choices will outperform other methods for your data/task.

Hypothesis Testing

From claims to testable hypotheses (Lectures 18+19)

- Ever since 1980, when Ronald Reagan inspired more men than women, the difference in the way men and women vote has been a significant part of American politics.

$$X := gender$$
$$Y := \%\ voted\ Democrat$$

- How would you test this hypothesis?
- Define null hypothesis, define alternative hypothesis, …

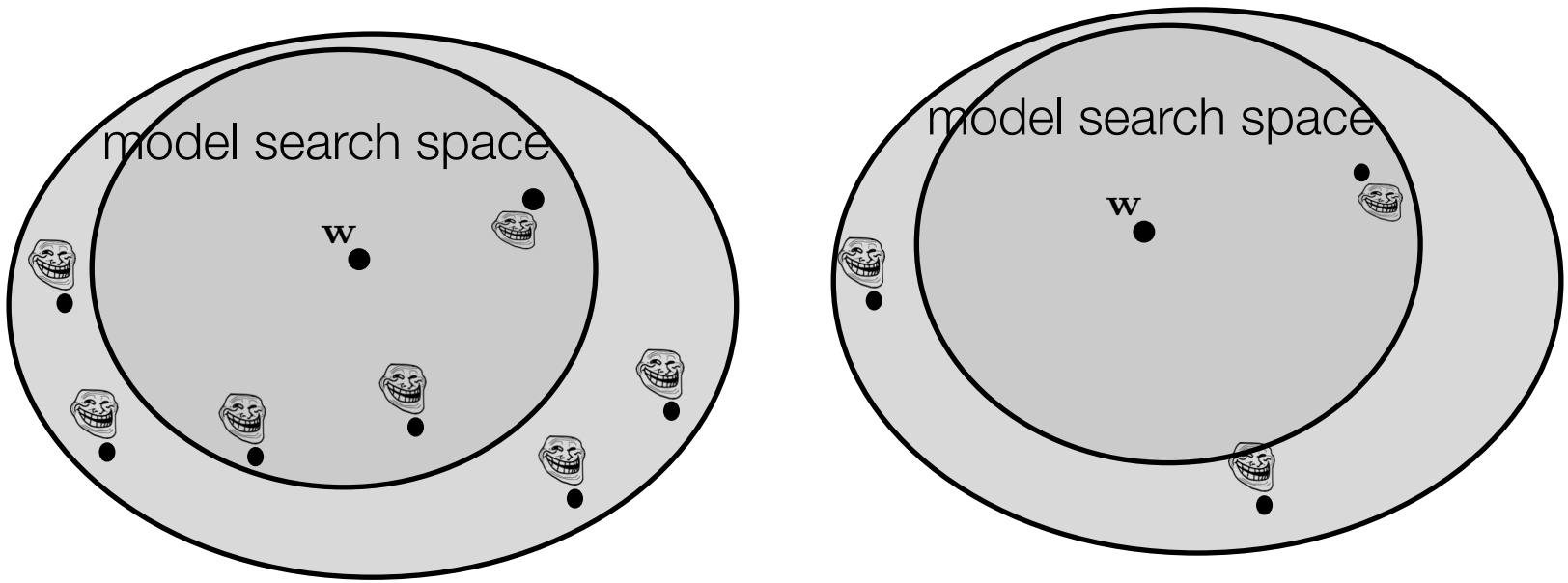
- Are we testing multiple hypotheses?
  - If yes, then we need to apply a correction
    - Type of correction depends on whether or not we are OK with rejecting null hypotheses that are true
  - Please study hypothesis testing

Controlling the Model Space

- To avoid model overfitting, models like logistic regression and neural networks, we often include parameter priors. Under which conditions would you consider using a strong or a weak prior.
- Describe why adding more data decreases the chances of overfitting
  - A good answer here will talk about hypotheses. Say, why does increasing the training data decreases the chances of have a wrong hypothesis (a troll parameter) be declared “good”?
  - Think of our class on hypothesis testing and the relationship between the number of observations and the likelihood a false hypothesis is declared true.

Controlling the Model Space (Lectures 7 + 18 + 19)

- When testing too many hypotheses, even false hypotheses will turn out true
  - This is the nature of multiple hypothesis tests
  - Adding more data helps reduce the chances of declaring a false hypothesis to be true (why?)
    - Fewer data points
    - More data points



Probabilistic interpretation:
Each of these points in the model search is a hypothesis… model search seeks a likely hypothesis given the data

Multi-armed Bandits [Lecture 20]

- Play-the-winner (issues)
- Epsilon-greedy (issues with multiple arms)
- UCB1
  - Describe the algorithm and what it is trying to accomplish.
  - Be mindful of the meaning behind simple questions like, “why not use the lower bound rather than the upper bound”.
  - What happens when the arms have similar or very different rewards?

MABs: Perpetual state of hypothesis testing (lecture 20)

- Often we don't care which hypothesis is correct
  - We really want to use the best-looking hypothesis at any point in time
- Exploration-exploitation trade-off of Multi-armed Bandits
  - Play arm with highest (empirical) average reward so far?
  - Play arms just to get a better estimate of expected reward?

Types of MABs (Lecture 20)

- Types of MABs seen in class:
  - Play-the-winner
  - UCB1
  - ε-greedy

- What are the issues and benefits of each of these methods?

Classification (Discriminative) & Supervised Learning

Linear Discriminant Function (Two Classes) [Lect 5]

- Two classes
- x is a D-dimensional real-valued vector (set of features)
- y is the car class

$$y_c = \begin{cases} 1 & \text{, if car } c \text{ is “small”} \\ -1 & \text{, if car } c \text{ is “luxury”} \end{cases}$$

- Find linear discriminant weights $\mathbf{w}$

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0$$

- Score function is the **squared error**

$$\sum_{c \in \mathrm{TestDataCars}} (y_c - y(x_c))^2$$

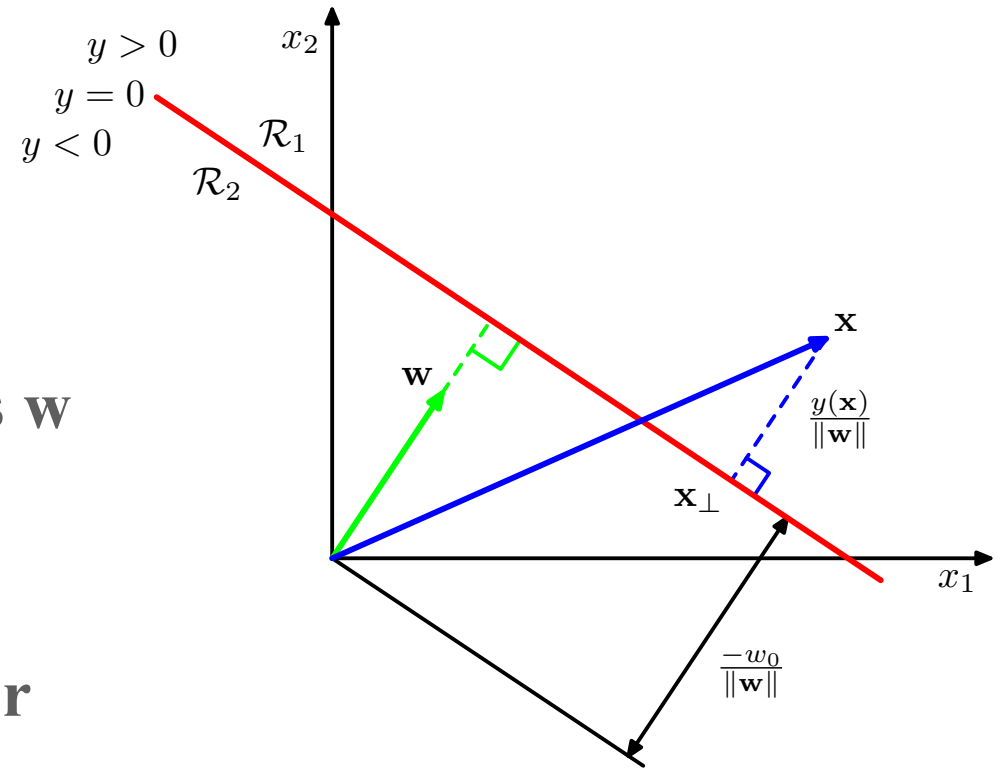
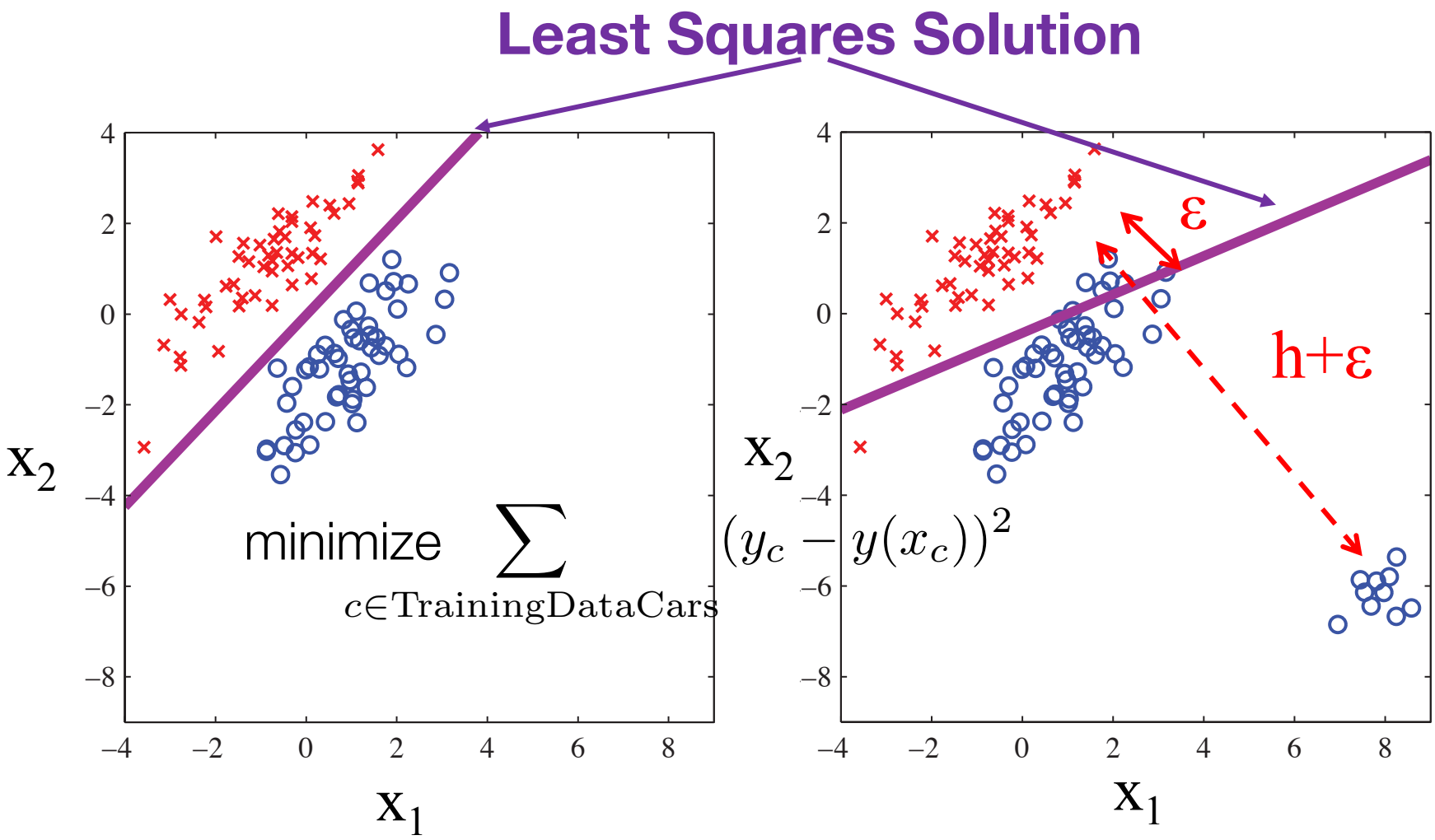


Figure: C. Bishop

- **Search algorithm: least squares algorithm**

Issues with Least Squares Classification (Lect 5)



With square loss (score), optimization cares too much about reducing distance to boundary of well separable items…

...solution is to change the score function

Naïve Bayes & Logistic Regression (LR)

- Explain why LR is considered a linear classifier
  - Understand through equations
- Naïve Bayes
  - What is the Naïve Bayes main assumption?
  - What happens if the assumption is violated?
  - Describe the algorithm

Logistic Regression (for Classification) [Lec 5]

- Back to two classes, C_1 and C_2
- Logistic regression is often used for two classes

$$p(\mathcal{C}_1|\boldsymbol{\phi}) = y(\boldsymbol{\phi}) = \sigma\left(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}\right)$$

where

$$\sigma(a) = \frac{\exp(a)}{1+\exp(a)}$$

Logistic function

and $\phi \equiv \phi(x)$

Transformation of feature vector (possibly non-linear)

- But can be easily generalized to K classes

Finding Logistic Regression Parameters (binary classification) [Lect 5]

- For a dataset $\{\phi_c, t_c\}$, where ϕ_c is transformed feature vector of car c, $t_c \in \{0,1\}$ is the class of car $c \in \text{TrainingDataCars}$
- The likelihood function over the training data is

$$p(\mathbf{t}|\mathbf{w}) = \prod_{c \in \text{TrainingDataCars}} y(\phi_c)^{t_c}(1 - y(\phi_c))^{1-t_c}$$

  where $\mathbf{t} = (t_1, \ldots, t_N)^{\mathrm{T}}$, $y(\phi_n) = \sigma(\mathbf{w}^T \phi_n)$

- The log of the likelihood function (log-likelihood) is

$$\log p(\mathbf{t}|\mathbf{w}) = \sum_{c \in \text{TrainingDataCars}} t_c \log y(\phi_c) + (1 - t_c) \log(1 - y(\phi_c))$$

- To solve the above equation, we maximize the log-likelihood over parameters $\mathbf{w}$
- The above equation is also described as *cross-entropy loss, logistic loss, log loss,* on Tensorflow, Sklearns, and pyTorch.

Solving Logistic Regression via Maximum Likelihood

- The above equations give the following gradient

$$\nabla_{\mathbf{w}} \log p(\mathbf{t}|\mathbf{w}) = \sum_{c \in \text{TrainingDataCars}} (t_c - y(\phi_c))\phi_c = \mathbf{\Phi}^T(\mathbf{t} - \mathbf{y})$$

Exercise: what are the dimensions of $\mathbf{\Phi}$?

- The logistic function has derivative

$$\frac{d}{da}\sigma(a) = \sigma(a)(1 - \sigma(a))$$

- The second derivative (Hessian) is then (verify!)

$$\mathbf{H} = \mathbf{\Phi}^T \mathbf{R} \mathbf{\Phi}$$

where $R_c = y(\phi_c)(1 - y(\phi_c))$ and $\mathbf{H}$ is a positive definite matrix. Because $\mathbf{H}$ is positive definite the optimization is concave on $\mathbf{w}$ and has a unique maximum.

Training Logistic Regression with Data Selection Bias (Lecture 5)

- Let π_i be the probability of sampling example i in the training data
- We say a data sample is biased when π_i is **not** uniform
- Correcting for bias in the likelihood function:

$$\log p(\mathbf{t}|\mathbf{w}) = \sum_{c \in \text{TrainingDataCars}} \frac{1}{\pi_c} \left(t_c \log y(\phi_c) + (1 - t_c) \log(1 - y(\phi_c)) \right)$$

  where C_c is the class of car c.

- Generally, it is a good idea to test training data for different weights
  - The weights can be used to protect against sampling designs which could cause selection bias.
  - The weights can be used to protect against misspecification of the model.
- Unbalanced datasets:
  - Suppose there are more males than females in dataset. What will happen to decision boundary? How to fix it?

Multiclass Logistic Regression (MLR) [Lecture 5]

- Consider K classes and N observations
- Let C_i be the class of the i-th example with feature vector ϕ_i

$$P(C_c = t_c|\phi_c) = \frac{\exp(\mathbf{w}_k^T \phi_c)}{\sum_{h=1}^{K} \exp(\mathbf{w}_h^T \phi_c)}$$

a.k.a. **softmax**

- If we assume one-hot encoding of target variable t_c, the log-likelihood function is

$$\sum_{c \in \text{TrainingDataCars}} \sum_{k=1}^{K} t_{c,k} \log \frac{\exp(\mathbf{w}_k^T \phi_c)}{\sum_{h=1}^{K} \exp(\mathbf{w}_h^T \phi_c)}$$

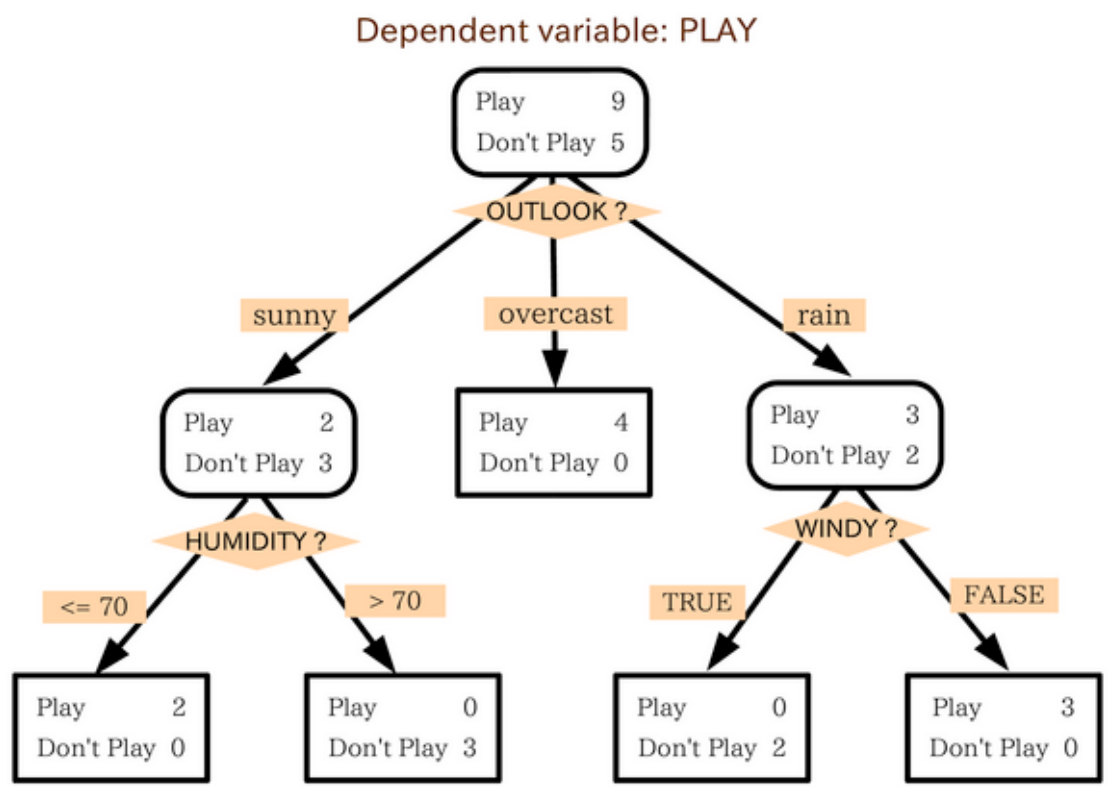
$$= \sum_{c \in \text{TrainingDataCars}} \sum_{k=1}^{K} t_{c,k} \mathbf{w}_k^T \phi_c - \sum_{c \in \text{TrainingDataCars}} \log \sum_{h=1}^{K} \exp(\mathbf{w}_h^T \phi_c)$$

Decision trees

- Know how to build decision trees (DTs)
- Why our DT optimization is greedy, i.e., why is it impractical to do an exhaustive search over the model space?
- Why binary do attributes only need to be considered once in the recursive decision tree call? Why do continuous-valued attributes must be considered at every split?
- Is it easy for a decision tree to overfit the training data? Why?

Decision Trees [Lecture 9]

- Easy to understand knowledge representation
- Can handle mixed variables
- Recursive, divide and conquer learning method
- Efficient inference



Amount of Information in the Tree

- How do we know how much information the tree encodes?

- How to decide each split?
  - Information Gain (Entropy)
  - Gini gain
  - Chi-Square score

Gradient Boosted Decision Trees [Lecture 12]

- Describe roughly how it works
  - Optimization objective:
    - Discrete optimization: tree building based on gradient scores
    - Continuous optimization: learning leaf weights and tree weight parameters

NBC (Naïve Bayes Classifier) [Lecture 10]

CPD: Conditional Probability Distribution

$$P(BC|A,I,S,CR) = \frac{P(A,I,S,CR|BC)P(BC)}{P(A,I,S,CR)}$$
$$= \frac{P(A|BC)P(I|BC)P(S|BC)P(CR|BC)P(BC)}{P(A,I,S,CR)}$$
$$\propto P(A|BC)P(I|BC)P(S|BC)P(CR|BC)P(BC)$$

NBC parameters = CPDs+prior

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

CPDs: P(A|BC)
P(I|BC)
P(S|BC)
P(CR|BC)
Prior:P(BC)

Ack: J. Neville

Naïve Bayes Classifier: Likelihood

- NBC likelihood uses the NBC probabilities for each data instance (i.e., probability of the class given the attributes)

$$L(\theta|D) = \prod_{i=1}^{n} p(i|\theta)$$ **General likelihood**

$$\propto \prod_{i=1}^{n} p(\mathbf{x}_i|c_i,\theta)P(c_i|\theta)$$ **Bayes rule**

$$\propto \prod_{i=1}^{n}\prod_{j=1}^{m} p(x_{ij}|c_i,\theta)P(c_i|\theta)$$ **Naive assumption**

Ack: J. Neville

NBC: Zero counts are a problem

- If an attribute value does not occur in training example, we assign **zero** probability to that value
- How does that affect the conditional probability $\mathrm{P}(\, f(x) \mid x \,)$?
- It equals 0!!!
- Why is this a problem?
- Adjust for zero counts by “smoothing” probability estimates

Ack: J. Neville

Ensemble Methods [Lecture 11]

- When and why should bagging be used? Give one condition where it would it have little to no effect.
- Describe how Random Forests (RFs) work
  - Why for small datasets or for datasets with lots of features, RFs tend to work better than a single decision tree?
- When and why should boosting be used?
  - Describe the Adaboost algorithm
  - Describe a condition where it will fail

Boosting Example

Given training set $(x_1, x_1), \ldots, (x_m, y_m)$

$y_i \in \{-1, +1\}$ is the correct label of instance $x_i \in X$

For t = 1, …, T

– Construct a distribution D_t on {1,…m}

– Find weak hypothesis ("*rule of thumb*")

$$h_t : X \to \{-1, +1\}$$

with small error ϵ_t on D_t:

$$\epsilon_t = P_{D_t}\left[h_t(x_i) \neq y_i\right]$$

Error is measured accord to the distribution!

Output: final hypothesis H_{final}

Adaboost

Constructing D_t on {1,..,m}:

$$D_1(i) = \frac{1}{m}$$

Given D_t and h_t:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot \begin{cases} e^{-\alpha_t} & \text{if} \quad y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if} \quad y_i \neq h_t(x_i) \end{cases}$$

$$= \frac{D_t(i)}{Z_t} \cdot e^{(-\alpha_t y_i h_t(x_i))}$$

Where:

Z_t = Normalization constant

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t}\right) > 0$$

Note that the weight of each hypothesis correlates with its error

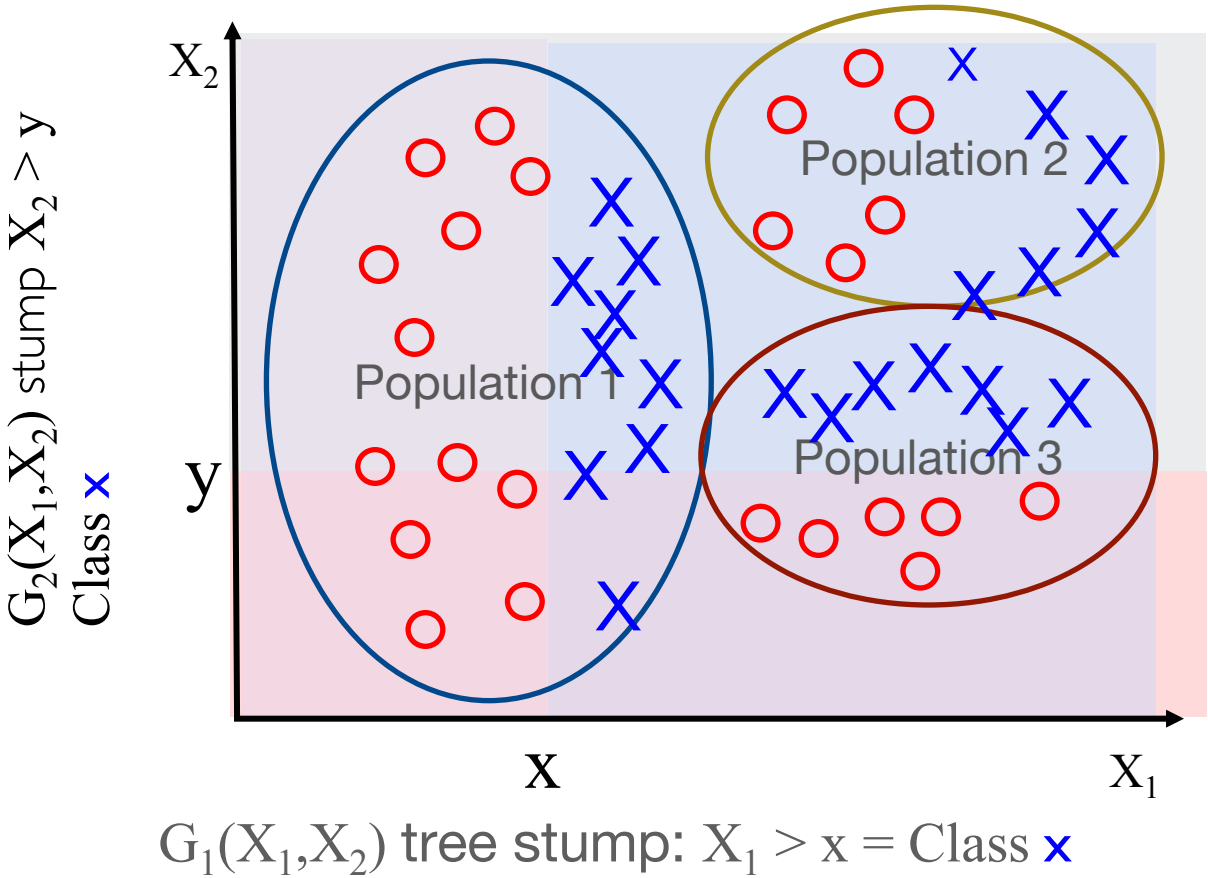
Final Hypothesis:

$$H_{\text{final}}(x) = \text{sign}\left(\sum_t \alpha_t h_t(x)\right)$$

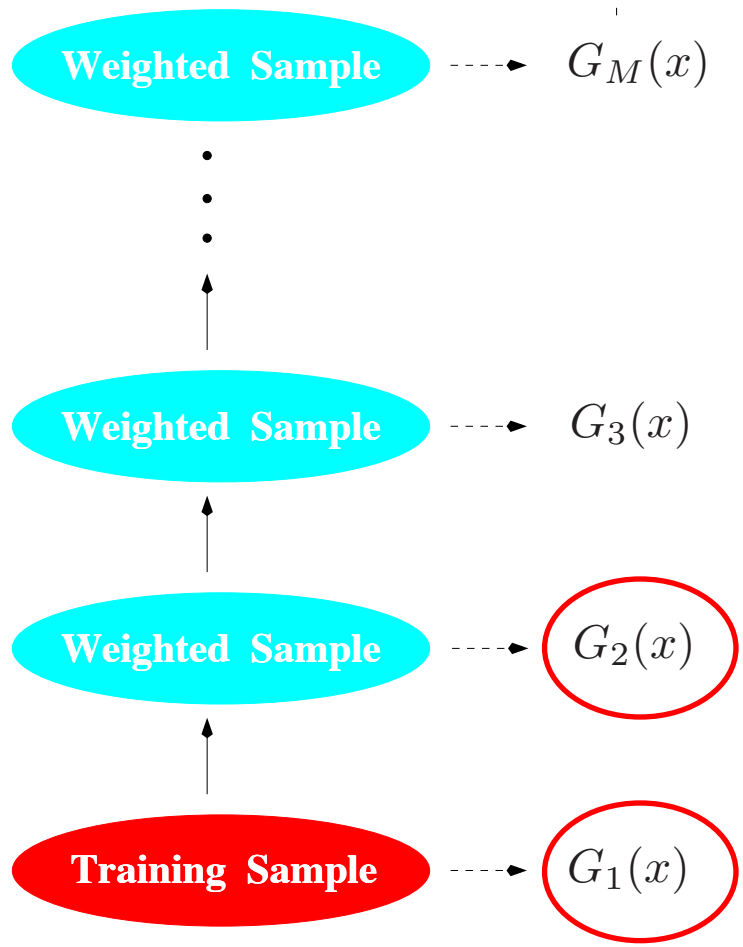
40

Boosting Idea

Our Data



$$G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$$



Boosting **greedily** fits simple classifiers to data (different populations)

But a more "thorough search" could do better

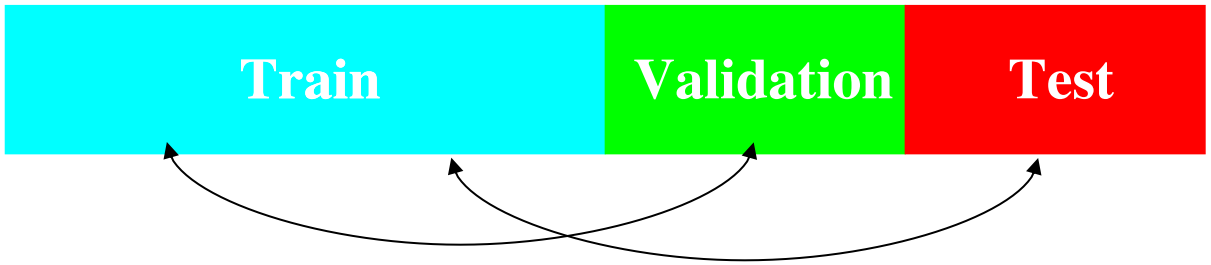
Model Selection & Assessment

Model Validation & Assessment [Lectures 17 – 19]

- Learning curves (training and validation accuracies as training data increases)
  - What do you expect to see?
  - Identify problems with model search (optimization) and whether or not we have enough data (or the model is too simple)
  - Peculiarities of neural network models (see homework 3)
- ROC curves and AUC score
- K-fold cross validation
- Bootstrapping
- Hypothesis testing
  - Formalizing the hypotheses (null and alternative)
    - One-tailed and two-tailed tests
  - Testing multiple hypotheses
    - What are the issues one faces when testing multiple hypotheses
    - Nested hypothesis tests (as in the homework)
  - Paired t-tests

Model Selection and Assessment

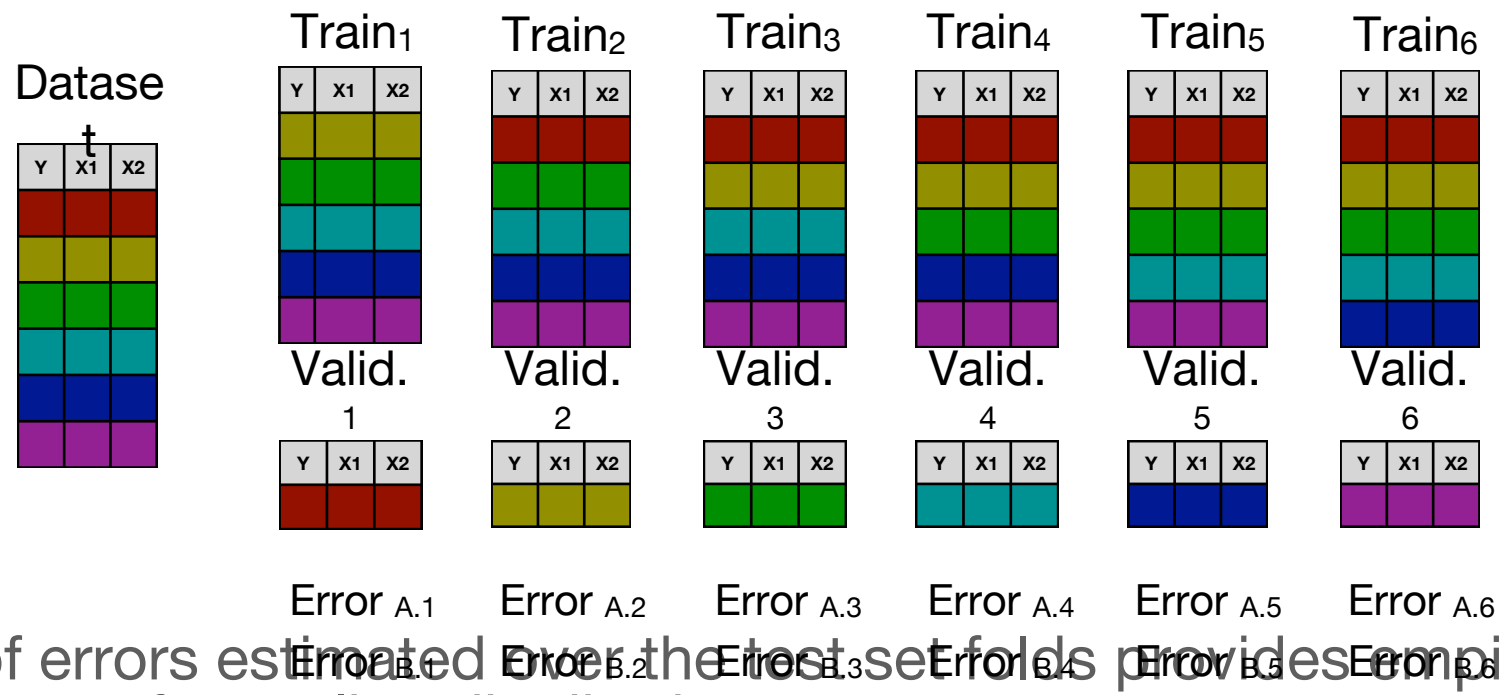
- In data-rich scenarios split the data:



- In data-poor scenarios: *approximate validation step*
  - analytically
    - AIC, BIC, MDL
  - via sample re-use
    - cross- validation
      - Leave-one-out
      - K-fold
    - bootstrap

Evaluating classification algorithms A and B

- Use k-fold cross-validation to get k estimates of error for algorithms A and B



- Set of errors estimated over the test set folds provides empirical estimate of sampling distribution
- Mean is estimate of expected error
- Is the mean error significant between A and B?
  - Hypothesis testing over the output of cross-validation

Decomposing Test Error

- Bias
  - Often related to size of model space
  - More complex models tend to have lower bias
- Variance
  - Often related to size of dataset and quality of information
  - When data is large enough to estimate true parameters (θ^*) well we get lower variance
- With big data simple models can perform surprisingly well due to lower variance

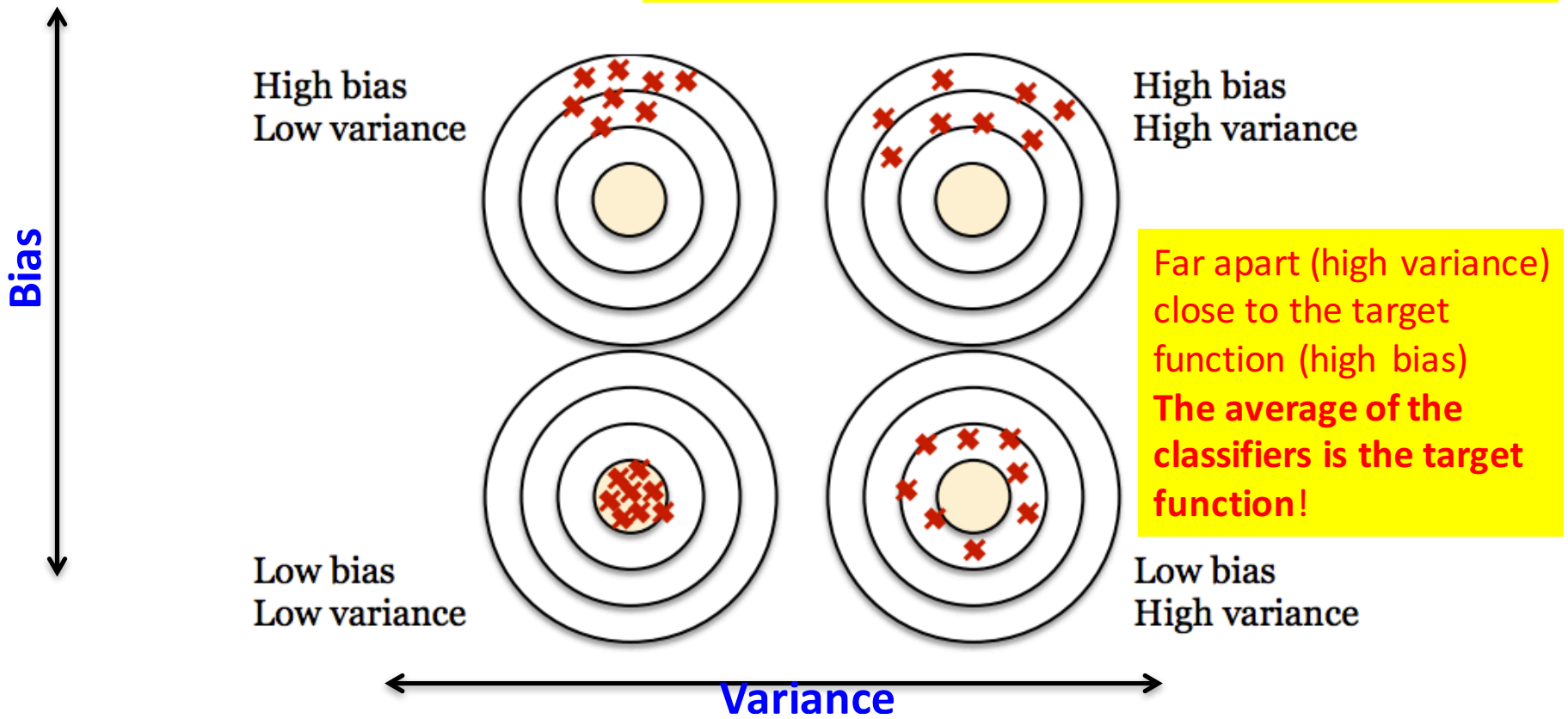
$$
\begin{aligned}
\mathrm{Err}(x_0) &= E[(t - f(x_0;\hat{\theta}))^2 | X = x_0] \\
&= \sigma_\epsilon^2 + \left(Ef(x_0;\hat{\theta}) - f(x_0;\theta^\star)\right)^2 + E\left[f(x_0;\hat{\theta}) - Ef(x_0;\hat{\theta})\right]^2 \\
&= \sigma_\epsilon^2 + \mathrm{Bias}^2(f(x_0;\hat{\theta})) + \mathrm{Var}(f(x_0;\hat{\theta}))
\end{aligned}
$$

Bias-Variance Tradeoff

Sample different sets from P(x,y) and train (each x is a learned model)

Dartboard = hypothesis space
Bullseye = target function
Darts = learned models

How would the learned models "behave" when trained over different datasets from the distribution?



Far apart (high variance) close to the target function (high bias)
The average of the classifiers is the target function!

Learning Curves

As we increase the training data while maintaining the same model complexity, we eventually find model parameters that are optimal for the underlying population.

Assumptions:

Assumption 1: the training data is representative of the underlying population

Assumption 2: parameters are free to assume their optimal value, i.e., have no priors

Dimensionality Reduction [Lectures 26 & 27]

- Problems with PCA's requirement of orthogonality
- Describe how we can use a neural network for dimensionality reduction

Dimensionality Reduction Examples:

- Principal Component Analysis (PCA) is a linear projection that maximizes the variance of the projected data
  - The output of PCA is a set of k orthogonal vectors in the original p-dimensional feature space, the k *principal components,* $k \leq p$
  - PCA gives us uncorrelated components, which are generally not independent components; for that you need independent component analysis

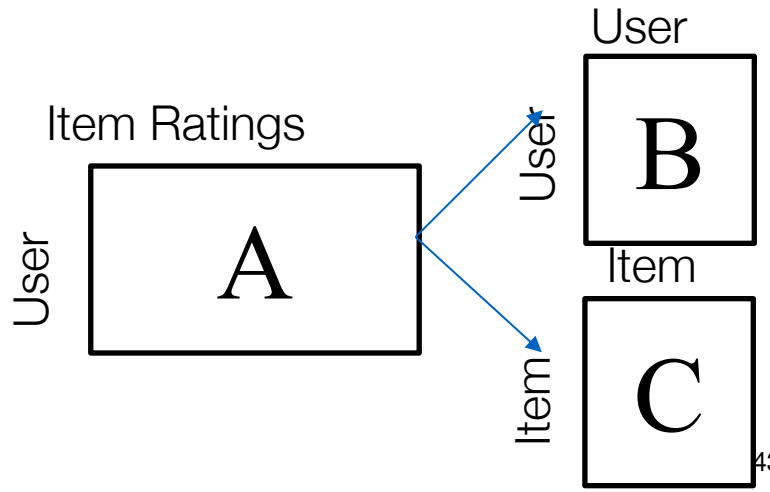
- Independent Component Analysis (ICA)
  - A weaker form of independence is uncorrelatedness. Two random variables y and y are said to be uncorrelated if their covariance is zero
  - But uncorrelatedness does not imply independence (nonlinear dependencies)
  - We would like to learn hidden independence in the data

Collaborative Filtering [Lecture 22]

Basic Idea

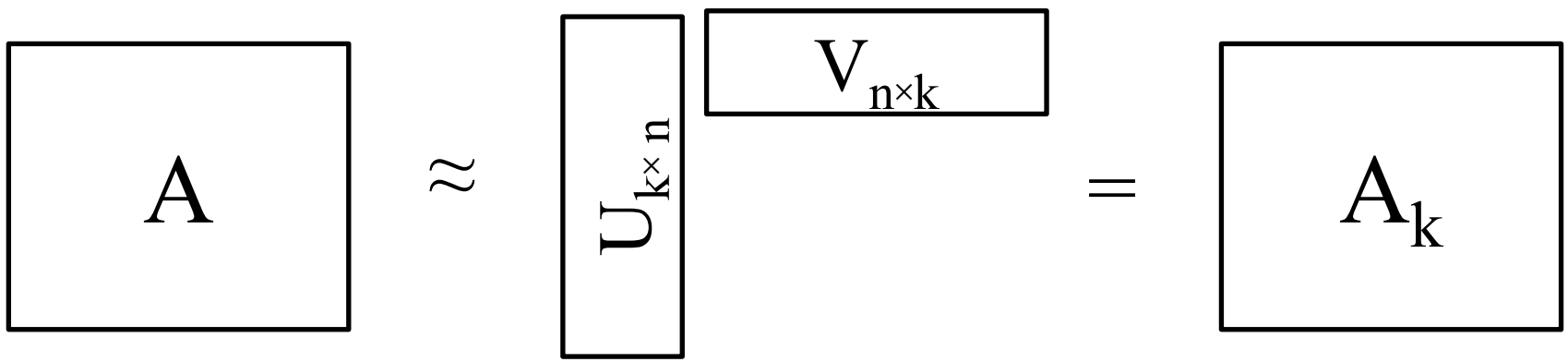
- (user,user) similarity to recommend items
  - B = AA^T/<normalization factor>
  - good if item base is smaller than user base
  - good if item base changes rapidly

- (item,item) similarity to recommend new items that were also liked by the same users
  - C = A^TA /<normalization factor>
  - good if the user base is small



Low Rank Reconstruction

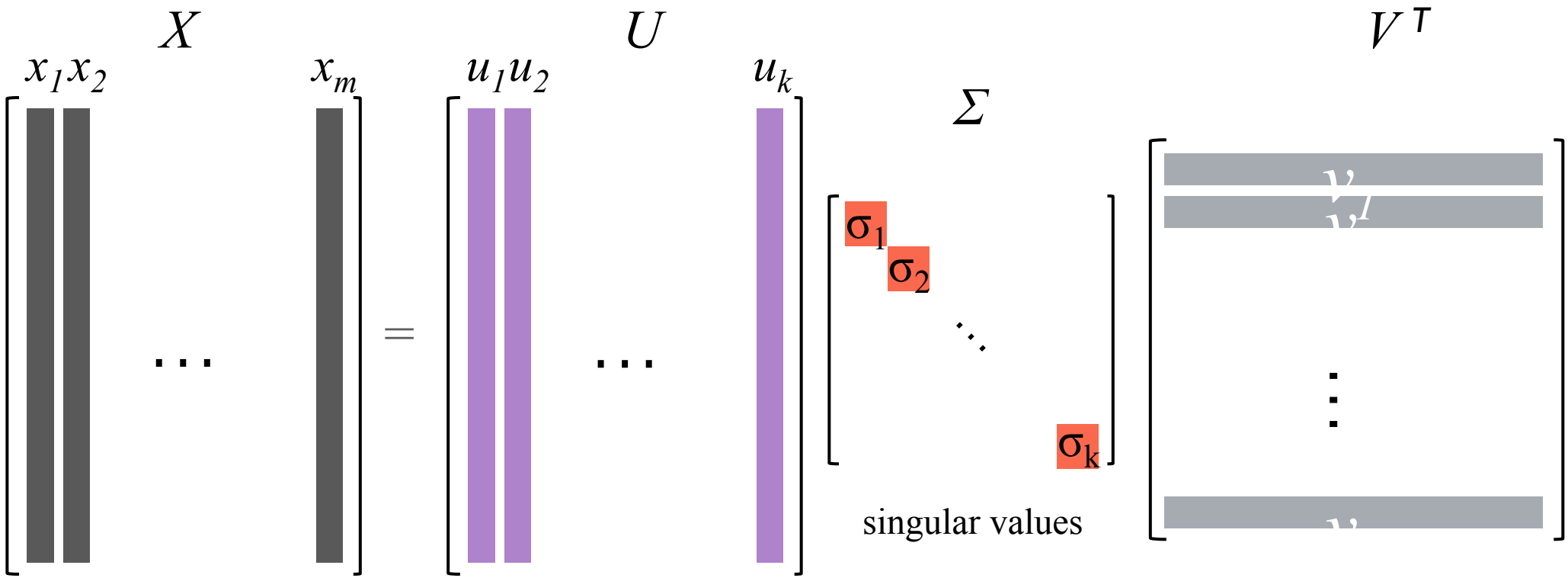
- Represent the adjacency matrix A with a lower rank matrix A_k.



- If $A_k(u,v)$ has large value for a missing $A(u,v)=0$, then recommend link (u,v)

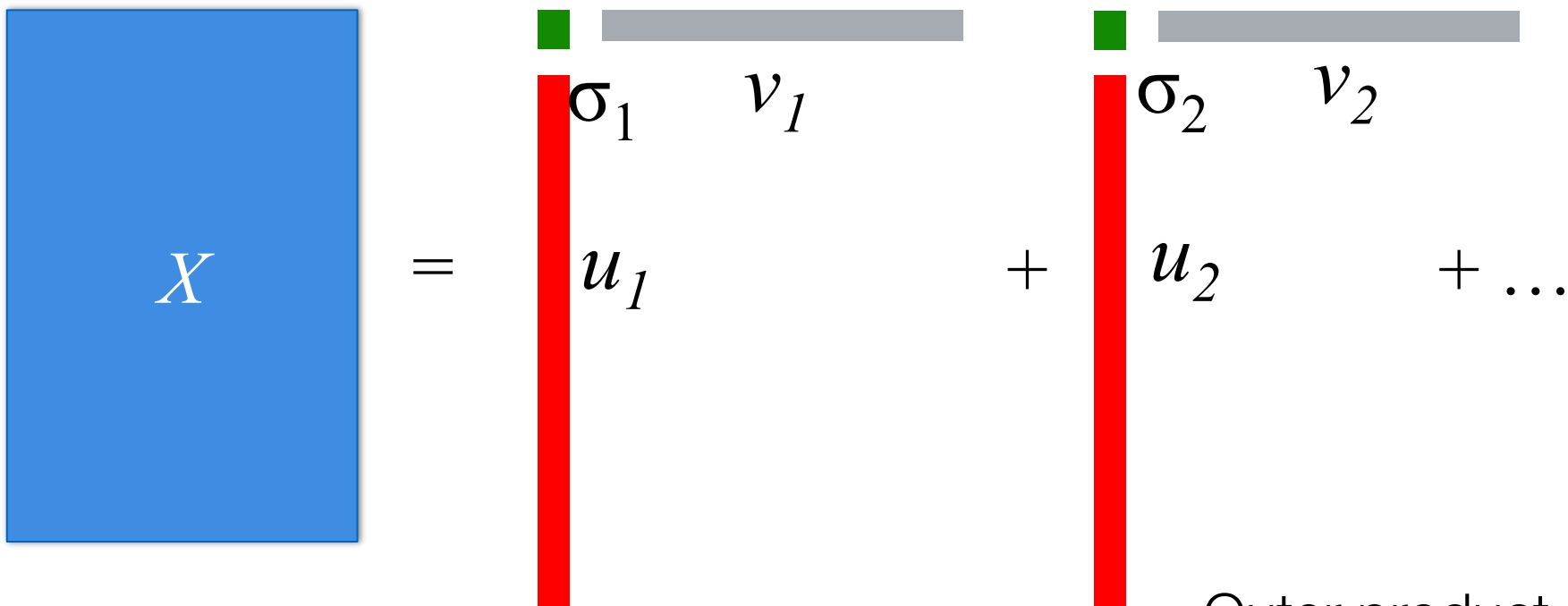
SVD Dimensions

$$X = U\Sigma V^T$$



SVD Definition

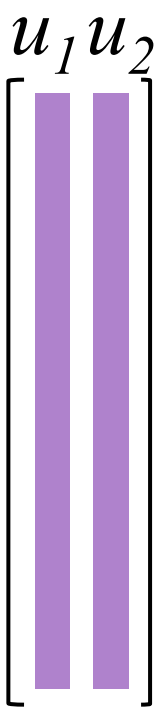
- SVD gives best rank-k approximation of X in L_2 and Frobenius norm

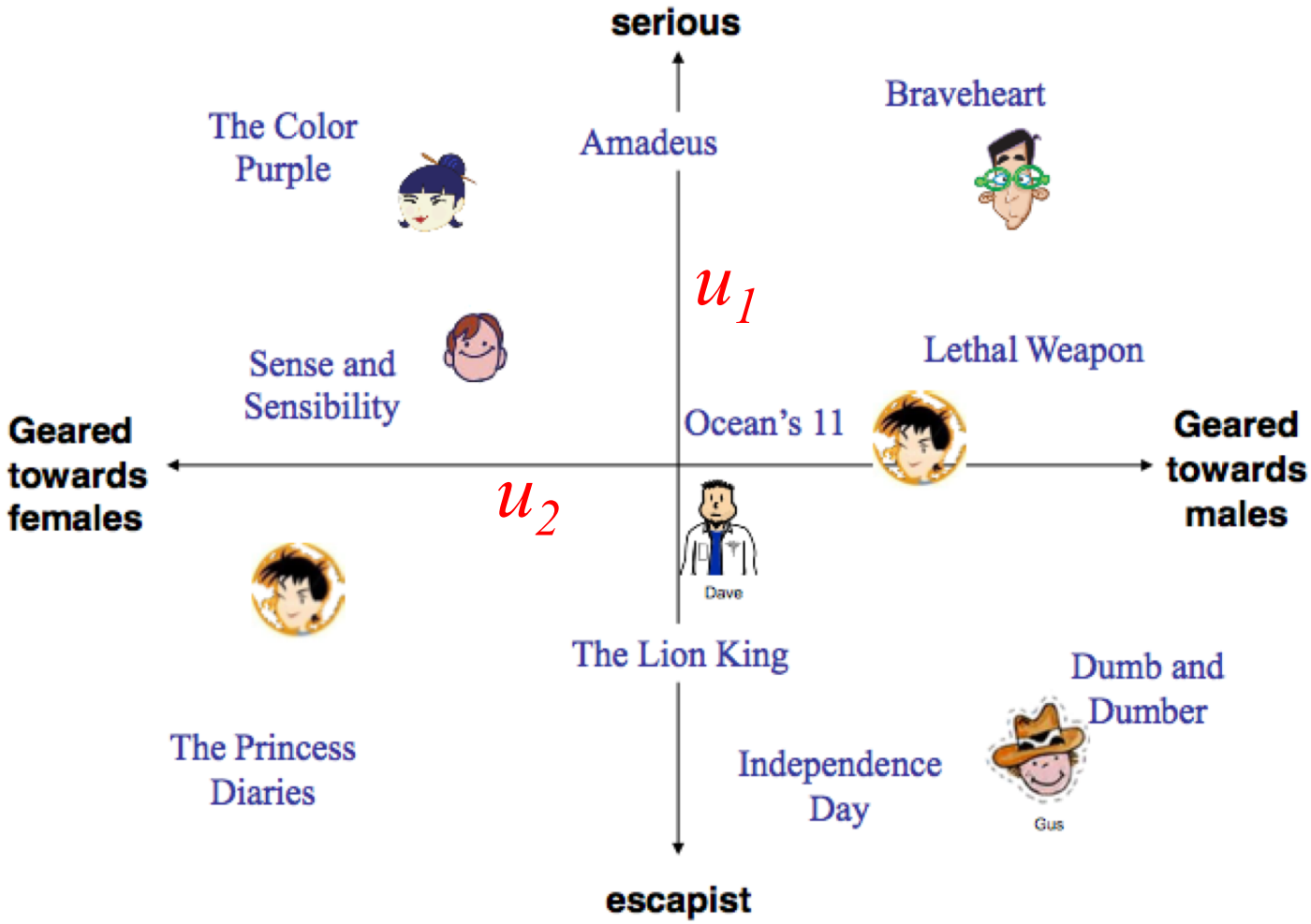


Outer product

$$X = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i u_i \otimes v_i$$

Low dimensional view of data





Neural Network Ensembles
(Deep Learning)

Neural Networks [Lectures 13, 14, 15, 24]

- Why does a fully connected neural network layer is guaranteed to outperform a not-fully-connected layer if we have enough training data. Give an example and explain why.
- Why do we need nonlinear neuron activations?
  - What would happen to a deep neural network if the activation were a linear function? Can you describe it mathematically?
- Describe the forward and backward passes used to train neural networks
  - Describe how gradients flow in and out of a network layer. Which gradients a neural network layer is expecting to receive, which gradients does it propagates back to the lower layers, and how does it update its own parameters? Pay attention to clearly index the gradients using the training data.
  - In a ReLU unit, under which conditions can the gradients be zero?

Equivalent model: Fully connected with zero weights between yacht and mansion parts of the model

Model M2

owns yacht
probability ∈ [0,1]

is rich y ∈ {0, 1}

owns mansion
probability ∈ [0, 1]

is tanned

employs a captain

pays high property taxes

employs a cook

1 x_{i1} x_{i2} 1 x_{i1} x_{i2}

$w_1^{(1)}$ $w_2^{(1)}$ $w_3^{(1)}$ $w_1^{(2)}$ $w_2^{(2)}$ $w_3^{(2)}$

$w_5^{(1)} = 0$ $w_4^{(1)} = 0$ $w_4^{(2)} = 0$ $w_5^{(2)} = 0$

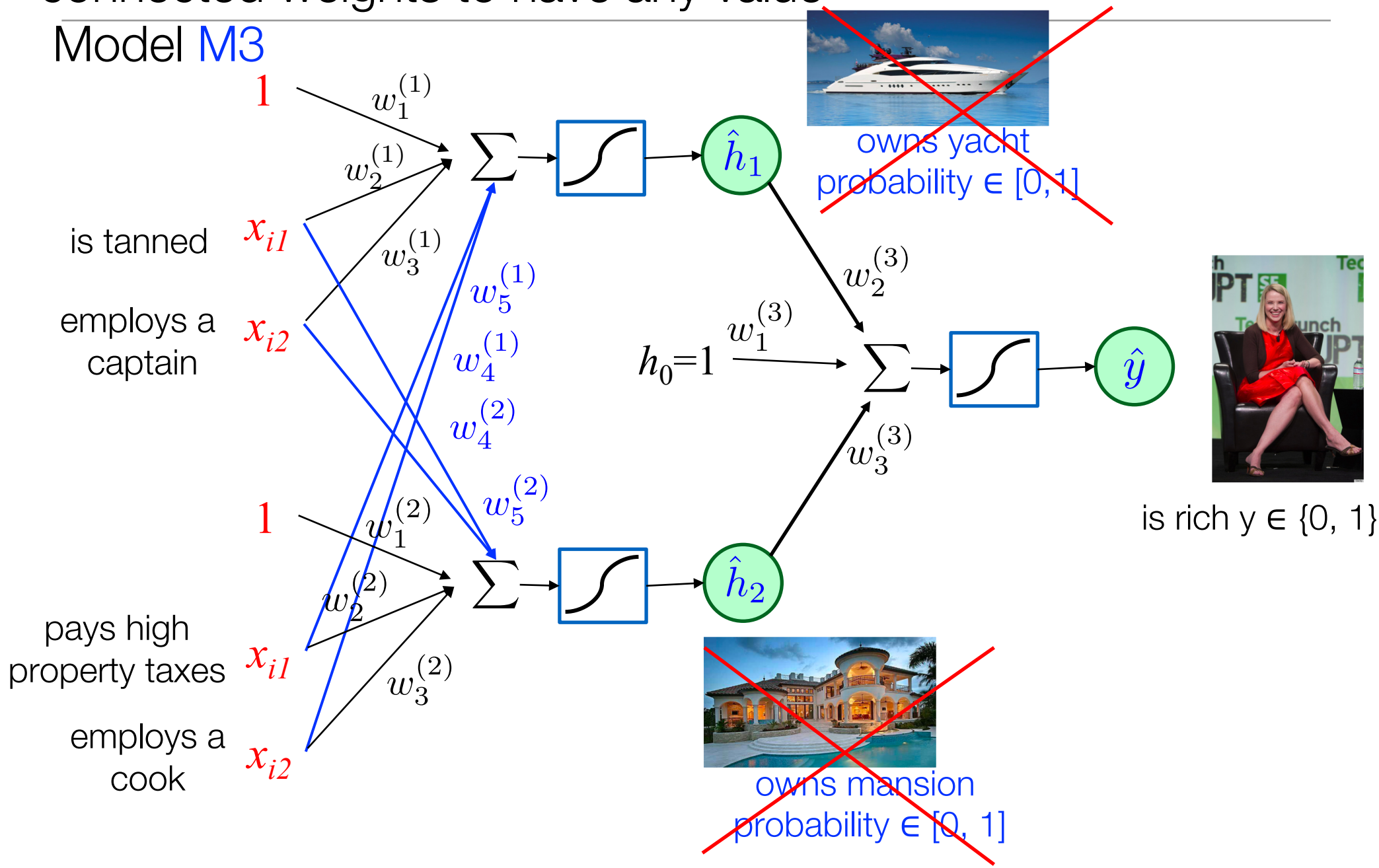
$\hat{h}_1$ $\hat{h}_2$ h_0=1

$w_2^{(3)}$ $w_1^{(3)}$ $w_3^{(3)}$

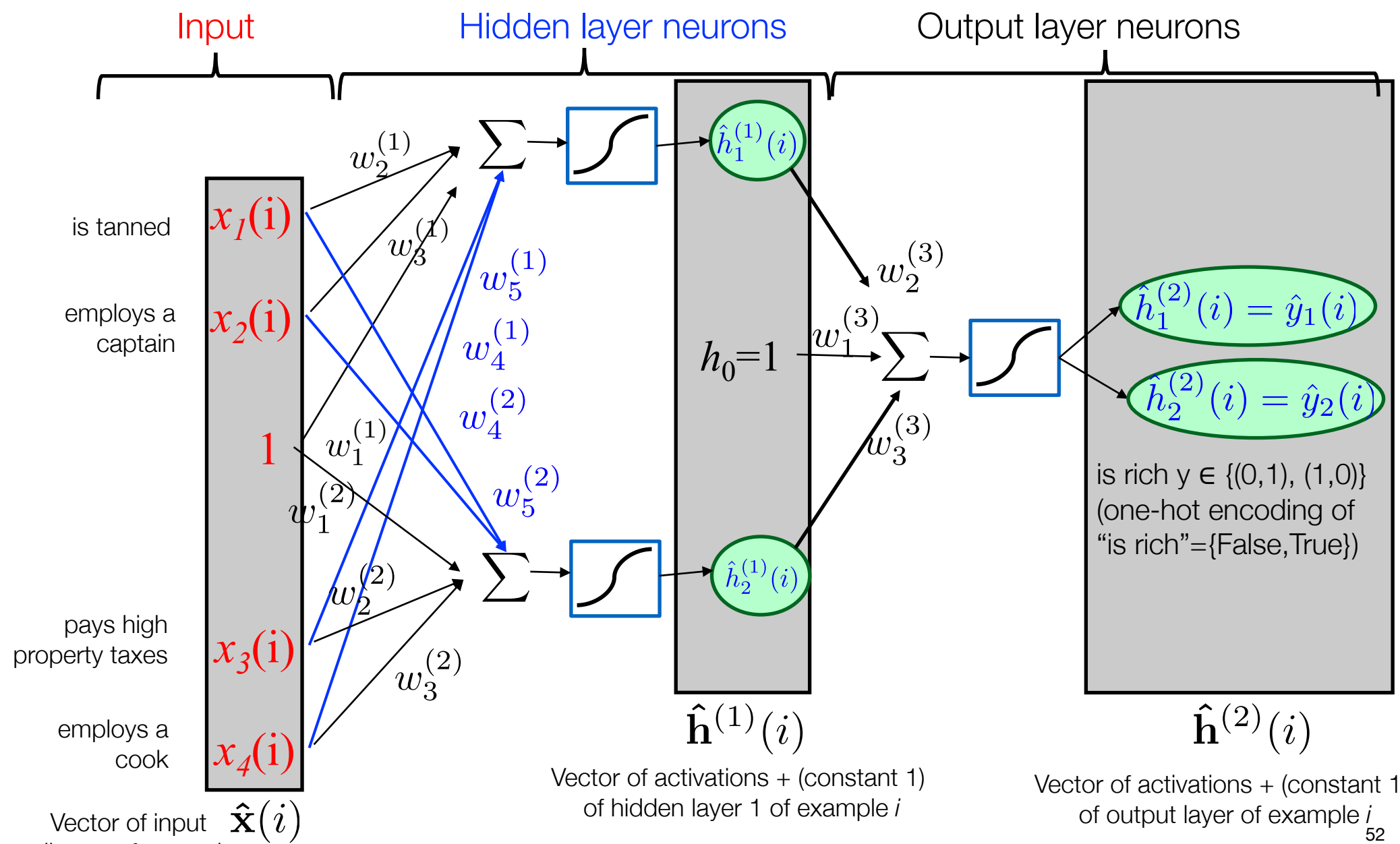
$\hat{y}$

More flexible model: Fully connected allowing inter-connected weights to have any value

Model M3



Feedforward Neural Network Example (is person rich?)

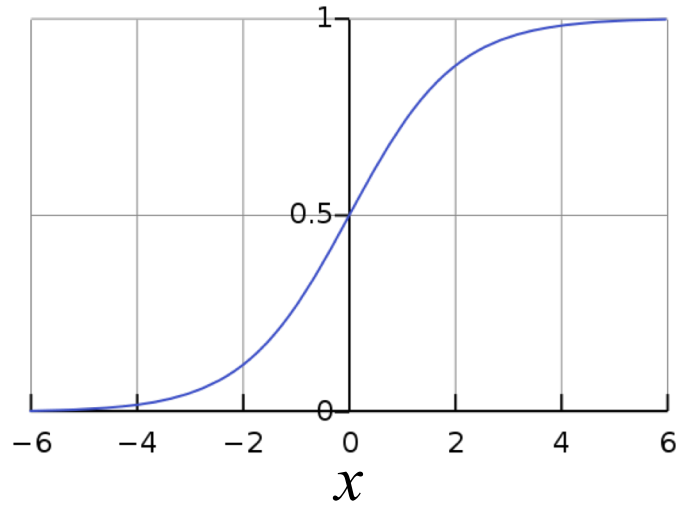


Logistic (neuron) Activation (non-linear filter)

- If input is $x = \mathbf{w}^T\mathbf{x}$, the output will look like a probability $\sigma(\mathbf{w}^T\mathbf{x}) \in [0, 1]$

- $p(y = 1 \mid \mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^T\mathbf{x}) \in [0, 1]$

$\sigma(x)$

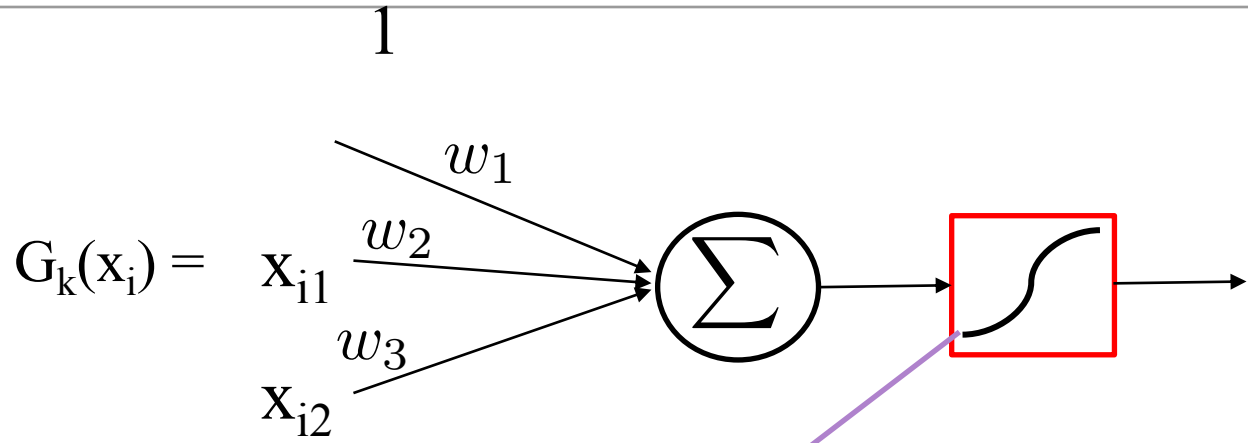


$$\sigma(x) = \frac{\exp(x)}{1 + \exp(x)}$$

- We will represent the logistic function with the symbol: 
- Very simple derivative: $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$

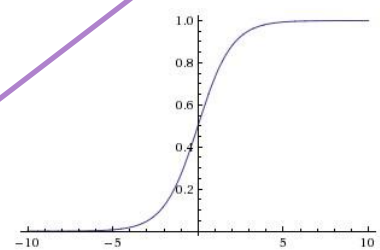
Why do we need non-linear activations?

Some Activation Functions Used (two classes)

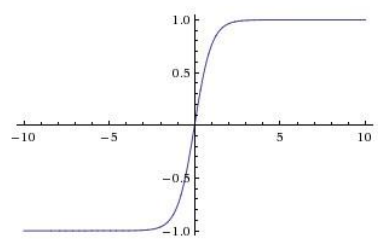


Sigmoid

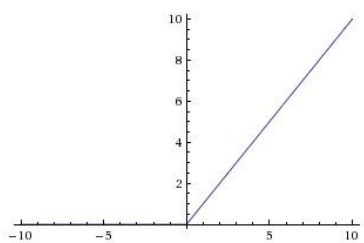
$$\sigma(x) = 1/(1+e^{-x})$$



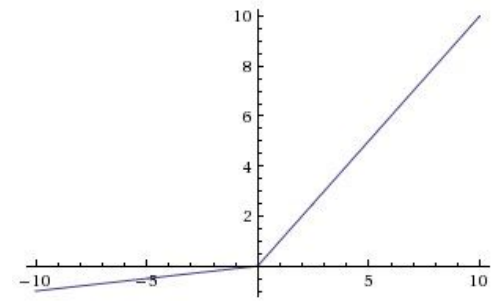
tanh tanh(x)



ReLU max(0,x)



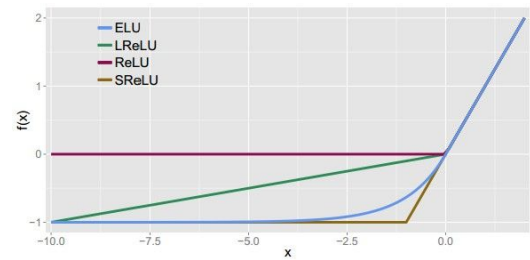
Leaky ReLU
max(0.1x, x)



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

ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$$



Ack: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Model search for our example

- Training data: $\{(x_1, y_1), \ldots, (x_n, y_n)\}$
- $\mathbf{x} = (1, x_{11}, x_{12}, x_{21}, x_{22})$
- $\mathbf{w}^{(k)} = (b, w_2^{(k)}, w_3^{(k)}, w_4^{(k)}, w_5^{(k)})$, $k = 1, 2$, where $b = w_1^{(1)} + w_1^{(2)}$
- $\mathbf{w}^{(3)} = (w_1^{(3)}, w_2^{(3)}, w_3^{(3)})$

Optimize using maximum likelihood estimation

Characteristics of user *i (tan, captain, cook,...)*

User *i* in training data: is rich $y_i \in \{0, 1\}$

$$\underset{\mathbf{w}^{(1)},\mathbf{w}^{(2)},\mathbf{w}^{(3)}}{\operatorname{argmax}} \sum_{i=1}^{n} \log p(y = y_i | \mathbf{x}_i; \mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \mathbf{w}^{(3)})$$

$$= \sum_{i=1}^{n} y_i \log \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i)) + (1 - y_i) \log(1 - \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i))),$$

where $\sigma(x) = \frac{\exp(x)}{1+\exp(x)}$, and $\mathbf{h}(\mathbf{x}) = (1, \hat{h}_1(\mathbf{x}), \hat{h}_2(\mathbf{x}))$,

$$\hat{h}_1(\mathbf{x}) = p(h_1 = 1 | \mathbf{x}) = \sigma((\mathbf{w}^{(1)})^T \mathbf{x})$$

and

$$\hat{h}_2(\mathbf{x}) = p(h_2 = 1 | \mathbf{x}) = \sigma((\mathbf{w}^{(2)})^T \mathbf{x})$$

Maximize likelihood via gradient ascent

- Model search via gradient ascent, requires computing the gradient first with respect to all parameters

Let $\mathbf{w} = (\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \mathbf{w}^{(3)})$. Learning via maximum likelihood estimation

$$\frac{\partial}{\partial \mathbf{w}} \sum_{i=1}^{n} \log p(y = y_i | \mathbf{x}_i; \mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \mathbf{w}^{(3)})$$

$$= \sum_{i=1}^{n} y_i \frac{\partial}{\partial \mathbf{w}} \log \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i)) + (1 - y_i) \frac{\partial}{\partial \mathbf{w}} \log(1 - \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i)))$$

$$= \sum_{i=1}^{n} y_i \frac{\frac{\partial}{\partial \mathbf{w}} \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i))}{\sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i))} - (1 - y_i) \frac{\frac{\partial}{\partial \mathbf{w}} \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i)))}{(1 - \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i)))}$$

$$= \sum_{i=1}^{n} \frac{\partial}{\partial \mathbf{w}} \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i)) \left(\frac{y_i}{\sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i))} - \frac{1 - y_i}{(1 - \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i)))} \right)$$

where $\sigma(x) = \frac{\exp(x)}{1+\exp(x)}$, and $\mathbf{h}(\mathbf{x}) = (1, \hat{h}_1(\mathbf{x}), \hat{h}_2(\mathbf{x}))$,

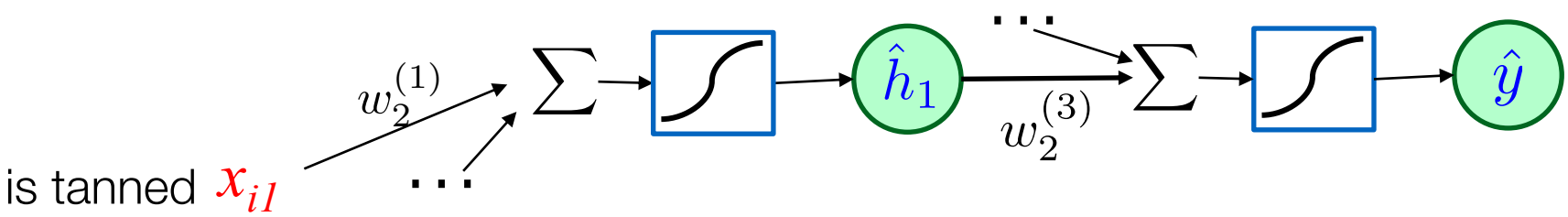
$$\hat{h}_1(\mathbf{x}) = p(h_1 = 1 | \mathbf{x}) = \sigma((\mathbf{w}^{(1)})^T \mathbf{x})$$

and

$$\hat{h}_2(\mathbf{x}) = p(h_2 = 1 | \mathbf{x}) = \sigma((\mathbf{w}^{(2)})^T \mathbf{x})$$

Computing Gradients of the Lower-Layer Parameters

- In a deep neural network each layer is a composition of previous layers



- The influence of a lower layer parameter in the final error can be recovered by the chain rule, which generally states:

$$\frac{\partial f^l(f^{l-1}(\cdots f^2(f^1(w))))}{\partial w} = \frac{\partial f^l}{\partial f^{l-1}} \cdot \frac{\partial f^{l-1}}{\partial f^{l-2}} \cdots \frac{\partial f^2}{\partial f^1} \cdot \frac{\partial f^1(x)}{\partial w}$$

$$\frac{\partial \sigma(\ldots + w_2^{(3)} \sigma(\ldots + w_2^{(1)} x_{i1}))}{\partial w_2^{(1)}} = \frac{\partial \sigma(\ldots + w_2^{(3)} \hat{h}_1)}{\partial \hat{h}_1} \cdot \frac{\partial \sigma(\ldots + w_2^{(1)} x_{i1})}{\partial w_2^{(1)}}$$

- Specific to the example given above:

$$\hat{h}_1 = \sigma(\ldots + w_2^{(1)} x_{i1})$$

How Tensorflow / Pytorch search model parameters

- neural nets will be very large: no hope of writing down gradient formula for all parameters

- **backpropagation** = recursive application of the chain rule along a computational graph to compute the gradients of all inputs/parameters/intermediates

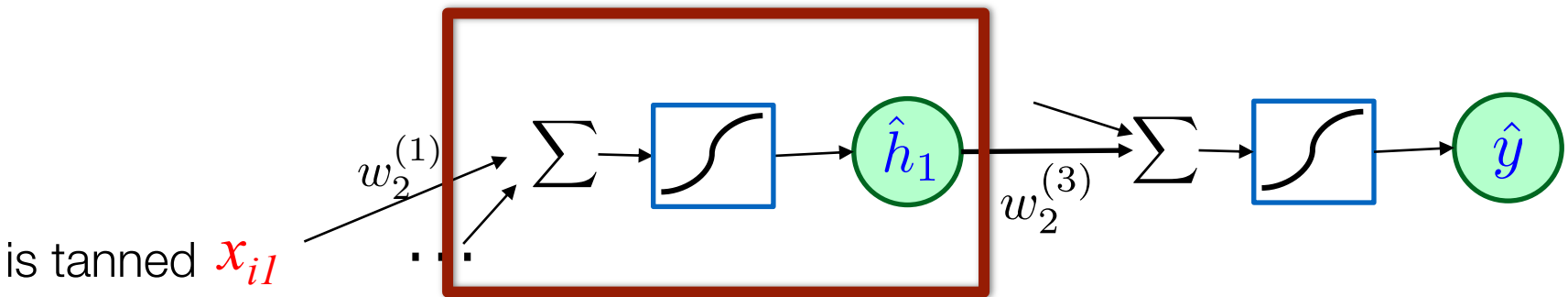
- Widely used implementations maintain a graph structure, where the nodes implement the **forward**() / **backward**() functions.

    - **forward:** compute result of an operation and save any intermediates needed for gradient computation in memory

    - **backward**: apply the chain rule to compute the gradient of the loss function with respect to the inputs.

Neural Network Gradient Ascent

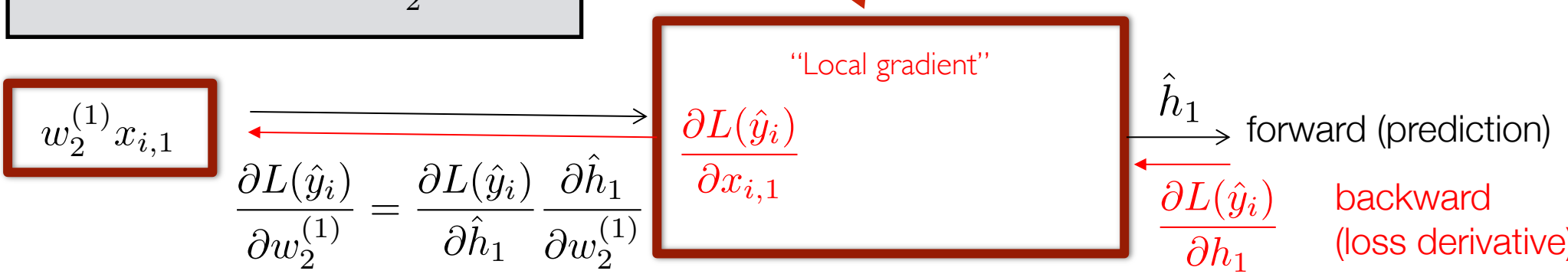
- L is the loss function with respect to final output



Gradient update step (learning rate $\varepsilon \cong 0$):

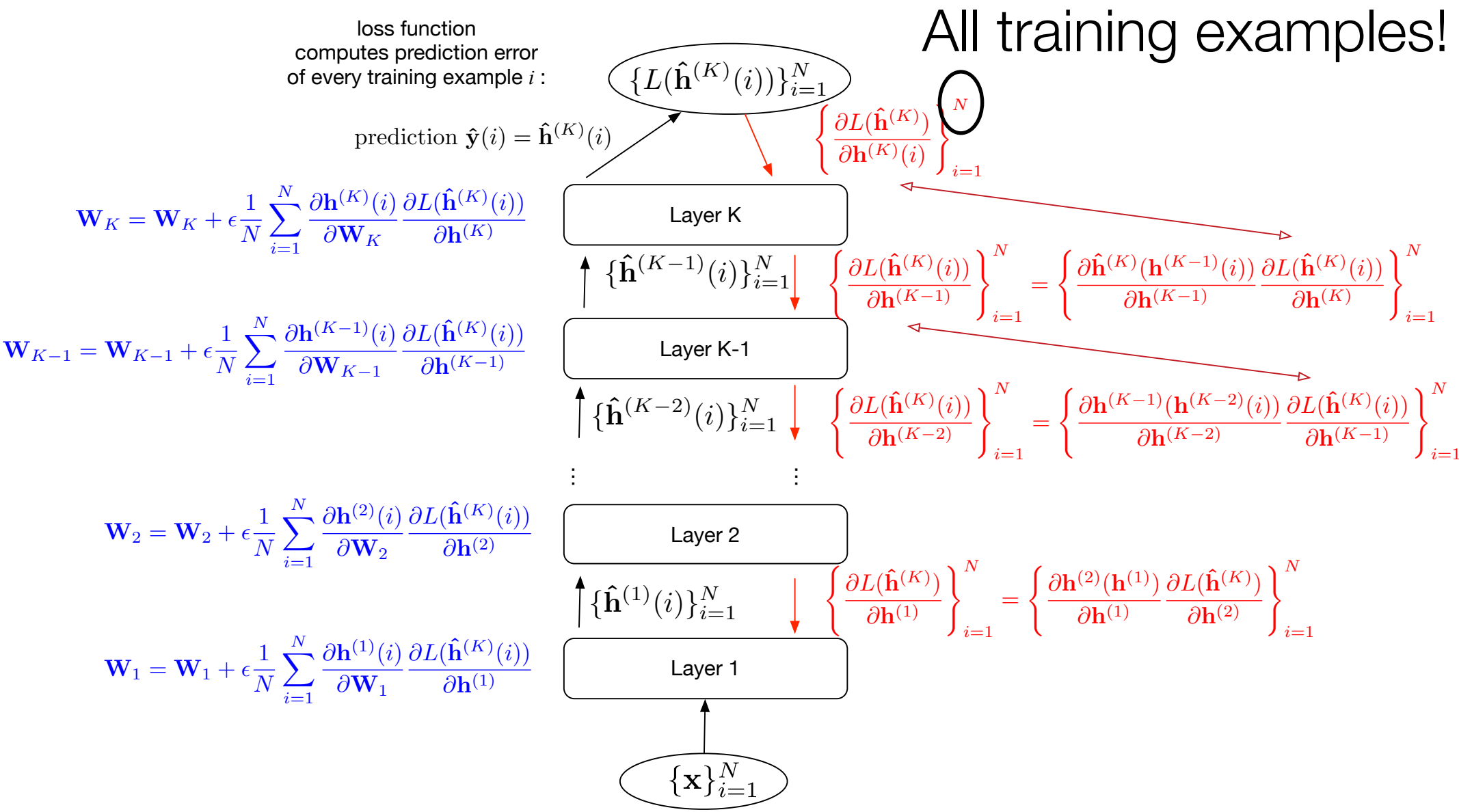
$$w_2^{(1)} = w_2^{(1)} + \epsilon \frac{\partial L(\hat{y}_i)}{\partial w_2^{(1)}}$$

- The local gradient measures how the output changes with the input



$$\frac{\partial L(\hat{y}_i)}{\partial w_2^{(1)}} = \frac{\partial L(\hat{y}_i)}{\partial \hat{h}_1} \frac{\partial \hat{h}_1}{\partial w_2^{(1)}}$$

Forward + Backward Updates (following the training data)



Stochastic Gradient Descent

- Rather than using all training examples in the gradient descent, we will use just a subset of the data at each time
    - At every gradient descent step we will just use a subset of the examples
- At every gradient update we randomly choose another set of *n* training examples
    - In practice, we do sampling **without** replacement; $\{\mathbf{x}\}_{i=1}^{n}$ where $n < N$.
      If training data is exhausted, restart sampling
- The "new" training data $\{\mathbf{x}\}_{i=1}^{n}$ is known as a mini-batch
    - The of training via gradient descent with mini-batches is called *mini-batch stochastic gradient ascent*
      (or mini-batch stochastic gradient descent if we are trying to minimize the score)

Rationalizing Mini-Batch Sizes

- Gradient computation: gradient is averaged over all training examples

$$\mathbf{W}_K = \mathbf{W}_K + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(K)}(i)}{\partial \mathbf{W}_K} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(K)}}$$

$$\mathbf{W}_{K-1} = \mathbf{W}_{K-1} + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(K-1)}(i)}{\partial \mathbf{W}_{K-1}} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(K-1)}}$$

$$\mathbf{W}_2 = \mathbf{W}_2 + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(2)}(i)}{\partial \mathbf{W}_2} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(2)}}$$

$$\mathbf{W}_1 = \mathbf{W}_1 + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(1)}(i)}{\partial \mathbf{W}_1} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(1)}}$$

- **Larger mini-batches** produce more accurate gradients
  - Would larger batches be better for training?
  - “Optimization benefits from more data because we can compute better gradients”?
- **Smaller learning rates ε** provide more accurate gradient ascent approximation, better models?

Model Search for Deep Neural Network

Q: Is it better to search for the best model (highest likelihood score) using all the training data?

A: Depends (Zhang et al. 2017)

- Deep neural network scores are nonconvex, many local maxima
- Pros of using all training data: Searching using all the training data should give a model that better fits the entire training data
- Cons: Using all training examples often works ***terribly*** in practice
  - Model found by gradient descent performs **poorly** even on the **training data** itself (due to poor local maxima)
  - Small mini-batches are often better than larger batches…
  - …but not too small…
  - …and depends on the learning rate (infinitesimal gradient ε steps)

(Zhang et al. 2017) Zhang, C., Bengio, S., Hardt, M., Recht, B. and Vinyals, O., 2016. Understanding deep learning requires rethinking generalization. ICLR 2017

Neural networks for structured data [Lecture 23]

- Why not use a feedforward network for images?
- What are the issues when trying to apply neural networks to graphs
  - Understand the importance of graph isomorphism to this problem
  - Understand what constitutes an operation that is invariant to isomorphism (sum, average, max, min)