349:Machine Learning

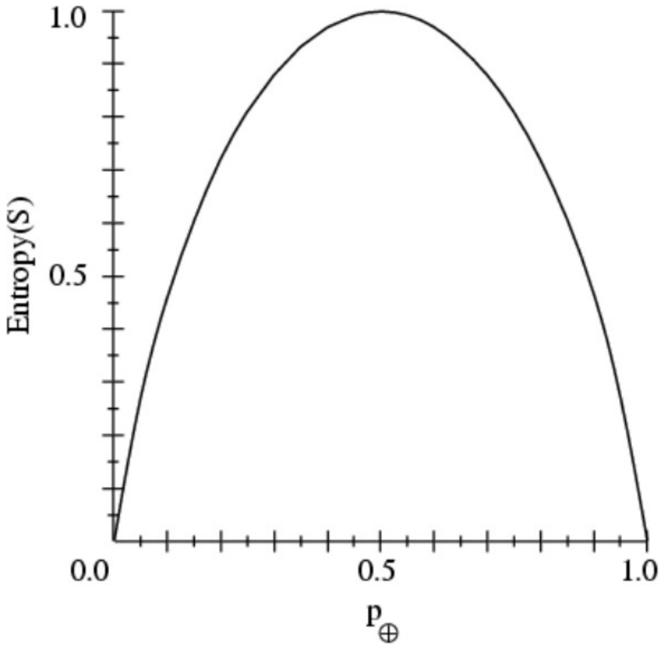
Fall 2024

Final Exam Review

Guidance

- Review slides are not necessarily exhaustive -- you are responsible for all material covered in class (including slides and commentary)
- You are not responsible for derivations
- You may not use a calculator, phone of other electronic devices
- Final exam is an individual assessment
- Final exam is a “closed-book” and “closed-notes” exam
- Any questions about grading must be made one week after scores are posted

Sample Entropy



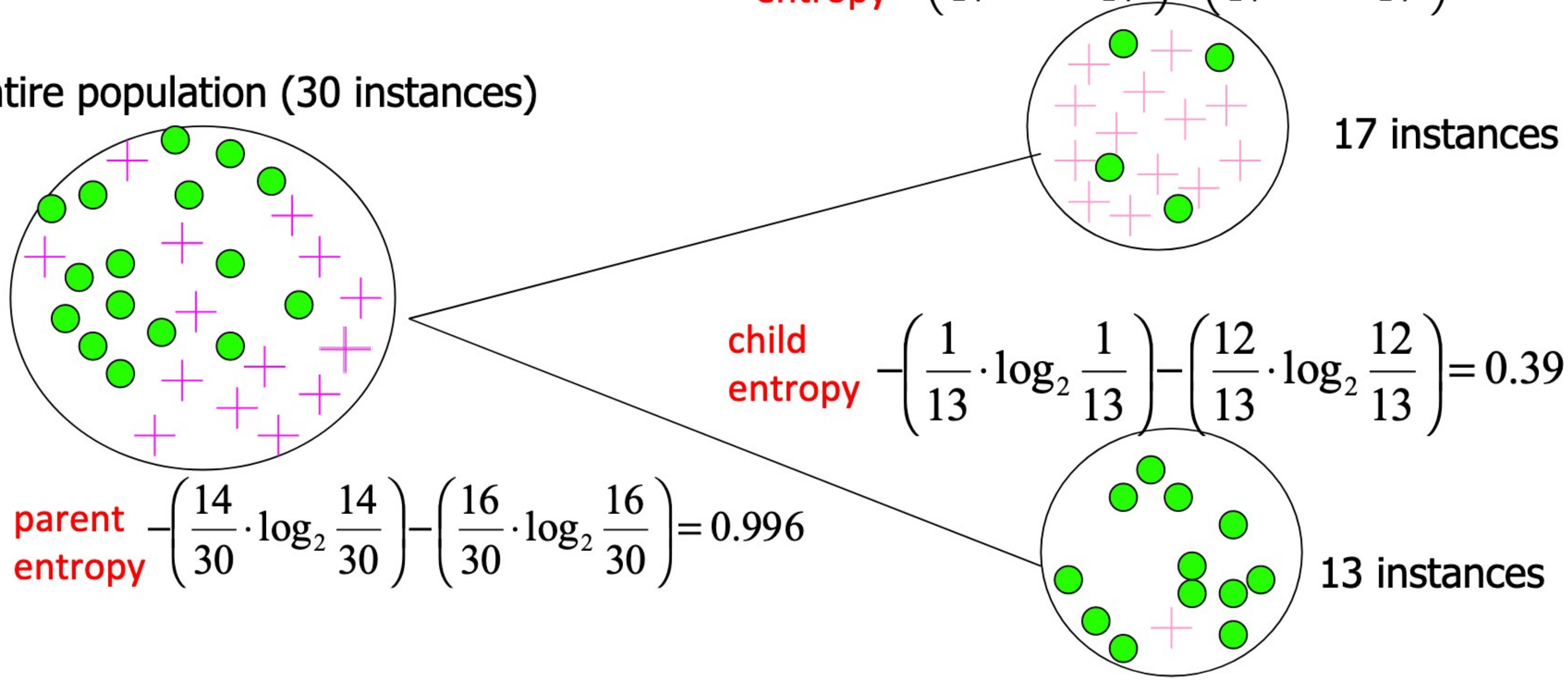
- S is a sample of training examples
- $p_\oplus$ is the proportion of positive examples in S
- $p_\ominus$ is the proportion of negative examples in S
- Entropy measures the impurity of S

$$H(S) \equiv -p_\oplus \log_2 p_\oplus - p_\ominus \log_2 p_\ominus$$

Information Gain Example

Information Gain = entropy(parent) – [average entropy(children)]

child entropy $-\left(\frac{13}{17}\cdot\log_2\frac{13}{17}\right)-\left(\frac{4}{17}\cdot\log_2\frac{4}{17}\right)=0.787$



child entropy $-\left(\frac{1}{13}\cdot\log_2\frac{1}{13}\right)-\left(\frac{12}{13}\cdot\log_2\frac{12}{13}\right)=0.391$

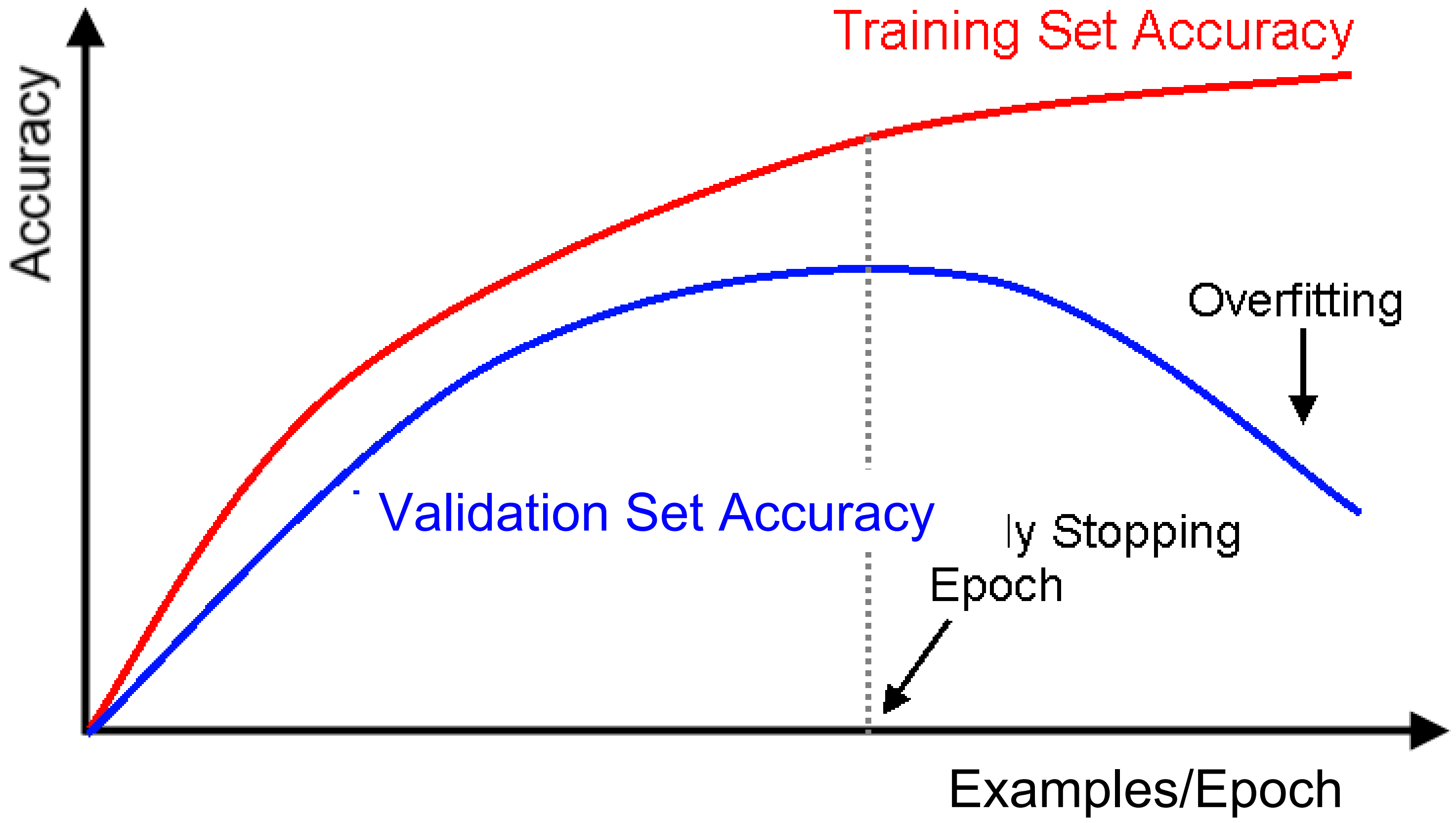
parent entropy $-\left(\frac{14}{30}\cdot\log_2\frac{14}{30}\right)-\left(\frac{16}{30}\cdot\log_2\frac{16}{30}\right)=0.996$

(Weighted) Average Entropy of Children = $\left(\frac{17}{30}\cdot 0.787\right)+\left(\frac{13}{30}\cdot 0.391\right)=0.615$

Information Gain= 0.996 - 0.615 = 0.38

38

What the learning curve tells us

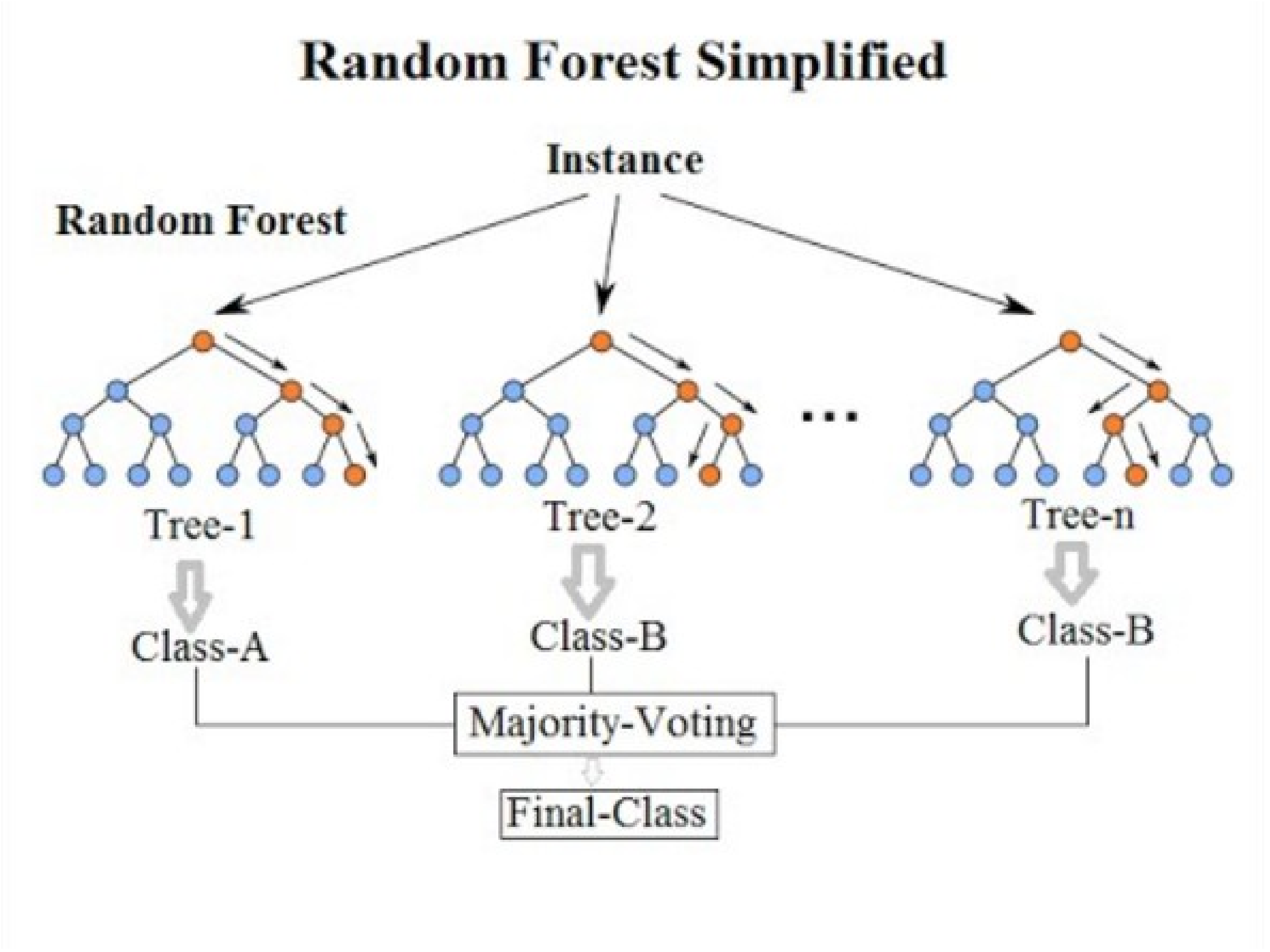


Avoid Overfitting

There are three approaches to preventing overfitting in decision trees:

1. **Early stopping:** Build the decision tree while applying some criterion that stops the decision trees growth before it overfits to the training data.
2. **Pruning:** Build the decision tree and allow it to overfit to the training data, then prune it back to remove the elements causing overfitting.
3. **Data preprocessing:** Making some changes to the initial data before ever building the tree

Random Forests



What you need to know about decision trees

Advantages to using decision trees:

1. Easy to interpret and make for straightforward visualizations
2. The internal workings are capable of being observed and thus make it possible to reproduce work
3. Can handle both numerical and categorical data
4. Can be used for classification and regression
5. Perform well on large datasets
6. Are extremely fast

Disadvantages to using decision trees:

1. Require algorithms capable of determining an optimal choice at each node
2. Prone to over-fitting, especially when a tree is particularly deep.

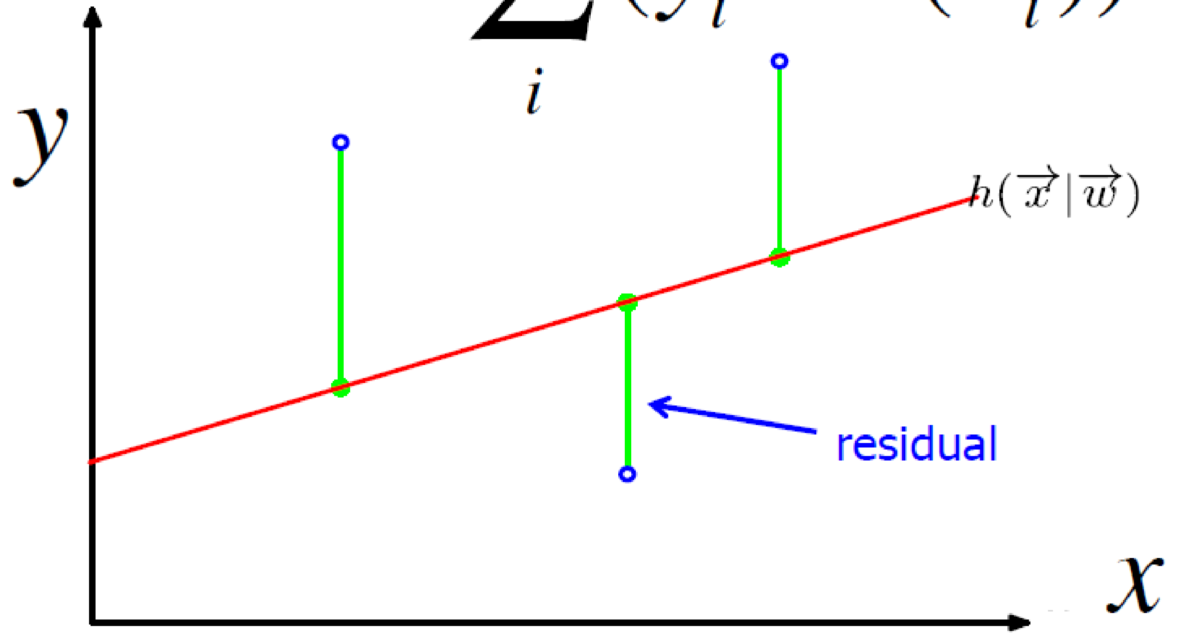
Random forests can be more accurate by reducing bias and variance

The Error Criterion

Typically estimate parameters by minimizing sum of squared residuals (RSS)…also known as the Sum of Squared Errors (SSE)

number of training examples

$$RSS = \sum_{i}^{\boxed{n}} (y_i - h(\mathbf{x}_i))^2$$



There Is a Closed-Form Solution!

Our goal is to find the weights of a function….

$$h(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + \ldots w_k x_k$$

…that minimizes the sum of squared residuals:

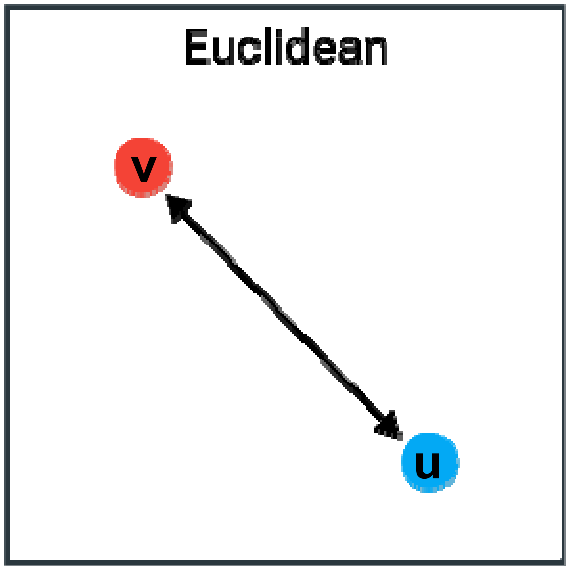
$$RSS = \sum_{i}^{n} (y_i - h(\mathbf{x}_i))^2$$

It turns out that there is a close-form solution to this problem!

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Just plug your training data into the above formula and the best hyperplane comes out!

Euclidean Distance



Source: Maarten Grootendorst

What people intuitively think of as "distance"

Most commonly used distance metric

Straight forward to calculate

Not scale invariant $\rightarrow$ normalization (in most cases)

"Curse of Dimensionality"

- every point is "equally" distant in high dimensions

Typical use cases include

- K-nearest neighbors, K-means, etc.
- Models with low-dimensional data

$$d(\mathbf{u},\mathbf{v}) = \sqrt{(u_1 - v_1)^2 + (u_2 - v_2)^2}$$

Euclidean Distance -- Generalized Formula

n = the number of dimensions

$$d(\mathbf{u},\mathbf{v}) = \left[\sum_{i}^{n} |u_i - v_i|^2\right]^{1/2}$$

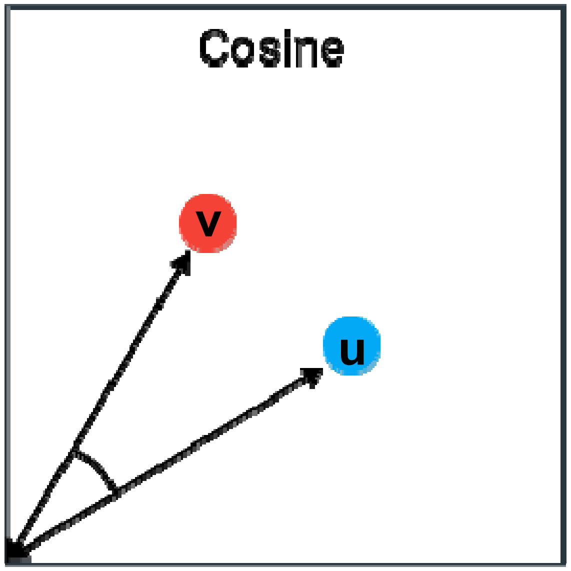
Where…

$$\mathbf{u} = [u_1, u_2, u_3, \dots, u_n]$$

$$\mathbf{v} = [v_1, v_2, v_3, \dots, v_n]$$

$$\mathbf{u}, \mathbf{v} \in \mathcal{R}^n$$

Cosine Similarity



Source: Maarten Grootendorst

Measures similarity using angle between two vectors

Not impacted by dimensionality of feature space

Straight forward to calculate

May be more intuitive in high dimensions

Scale invariant -- vector magnitude is irrelevant

Ranges from -1.0 to 1.0 (*e.g.,* [-1,1]

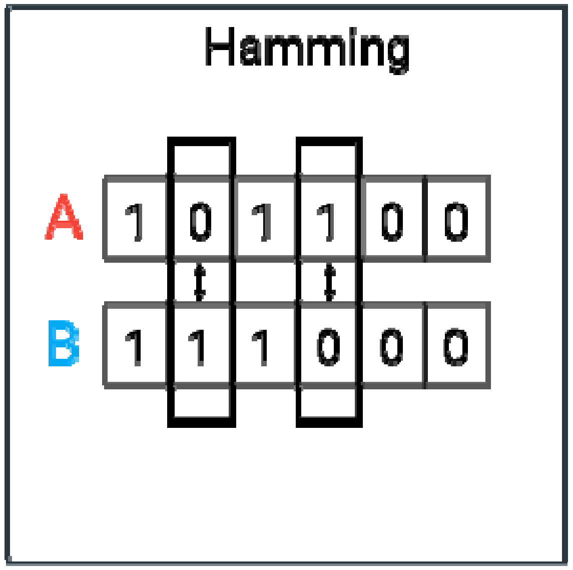
Typical use cases include

- High-dimensional data
- Natural language processing
- Google user vectors?

$$d(u,v) = \cos(\theta) = \frac{v \cdot u}{\|v\|_2 \, \|u\|_2}$$

$$\|u\|_2 = d(u,0) = \sqrt{\sum_{i=1}^{n} (u_i - 0)^2}$$

Hamming Distance



Source: Maarten Grootendorst

Features are mapped to binary vectors

Measures number of attributes that are different

Does not take magnitude into account

Does not (explicitly) take similarities into account

Typical use cases include

- Error detection and correction
- Categorical variables

$$d(\vec{x},\vec{y}) = \sum^{n} | x_i - y_i |$$

$$\text{where } \vec{x} =< x_1, x_2, \ldots, x_n >,$$
$$\vec{y} =< y_1, y_2, \ldots, y_n \}$$
$$\text{and } \quad \forall i (x_i, y_i \in \{0,1\})$$

Hamming Distance - Example

Blues	Jazz	Rock	Zydeco	Vocals
0	0	1	0	1
0	1	0	0	0
0	0	1	0	0

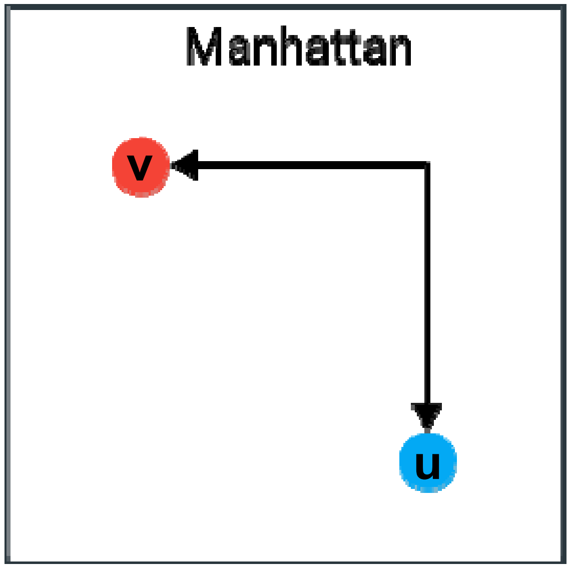
s1 = {rock, vocals}

s2 = {jazz, no_vocals}

s3 = { rock, no_vocals}

Hamming Distance = number of bits different between binary vectors

Manhattan Distance



Source: Maarten Grootendorst

Sometimes known as "City Block" distance

Less intuitive than Euclidean or Cosine Similarity

Does not suffer from the "Curse of Dimensionality"

- values change by "quanta"

Not continuous $\rightarrow$ may not be differentiable

Typical use cases include

- Discrete (or binary) values

$$d(u,v) = \sum_{i=1}^{n} \left| u_i - v_i \right|$$

Single Nearest Neighbor

Given some set of training data…

$$D = \{<\vec{x}_1, f(\vec{x}_1)>, \ldots <\vec{x}_m, f(\vec{x}_m)>\}$$

...and query point $\vec{x}_q$, predict $f(\vec{x}_q)$

distance function

1. Find the nearest member of the data set to the query

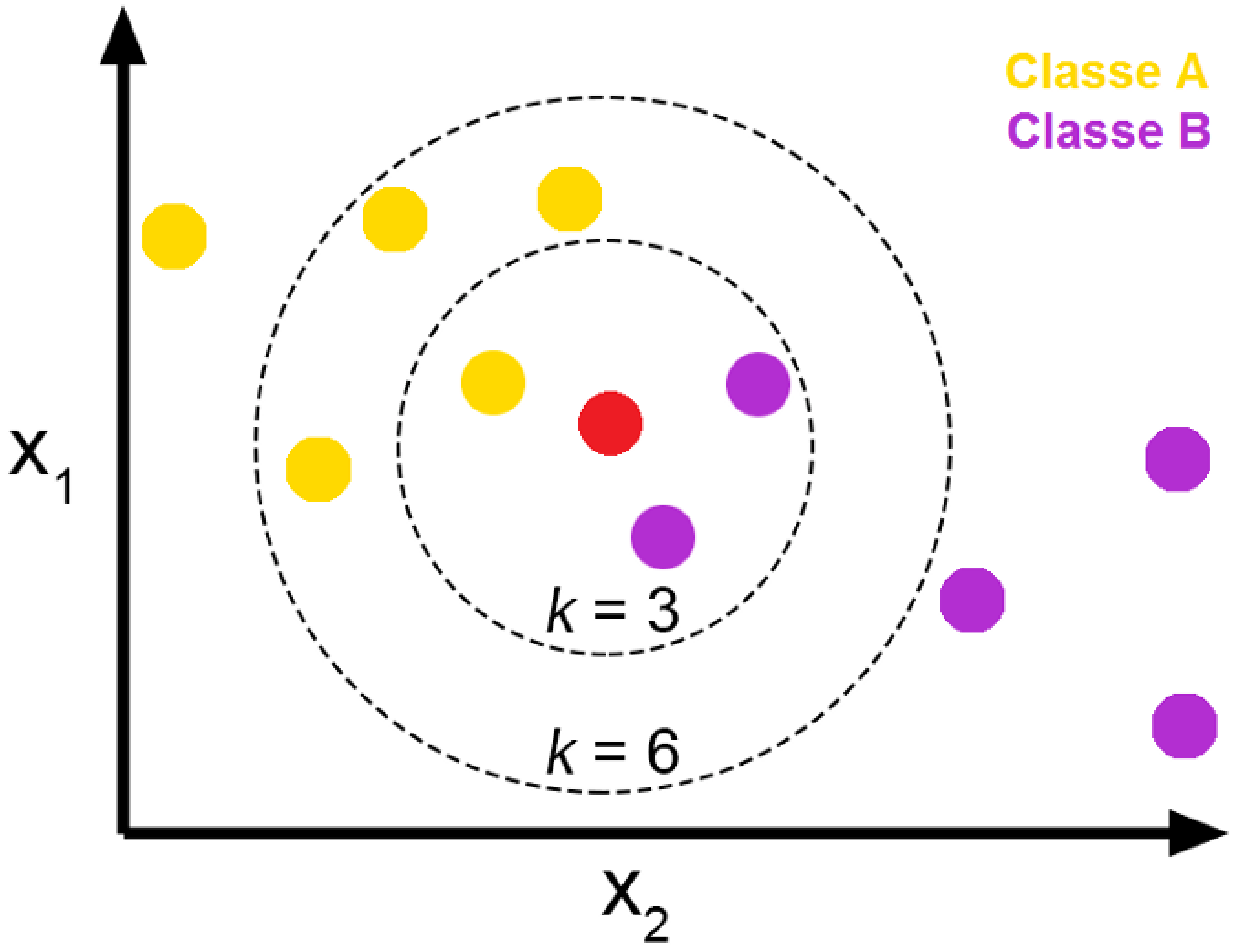
$$\vec{x}_{nn} = \underset{\vec{x} \in D}{\arg\min}(d(\vec{x}, \vec{x}_q))$$

2. Assign the nearest neighbor's output to the query

$$h(\vec{x}_q) = f(\vec{x}_{nn})$$

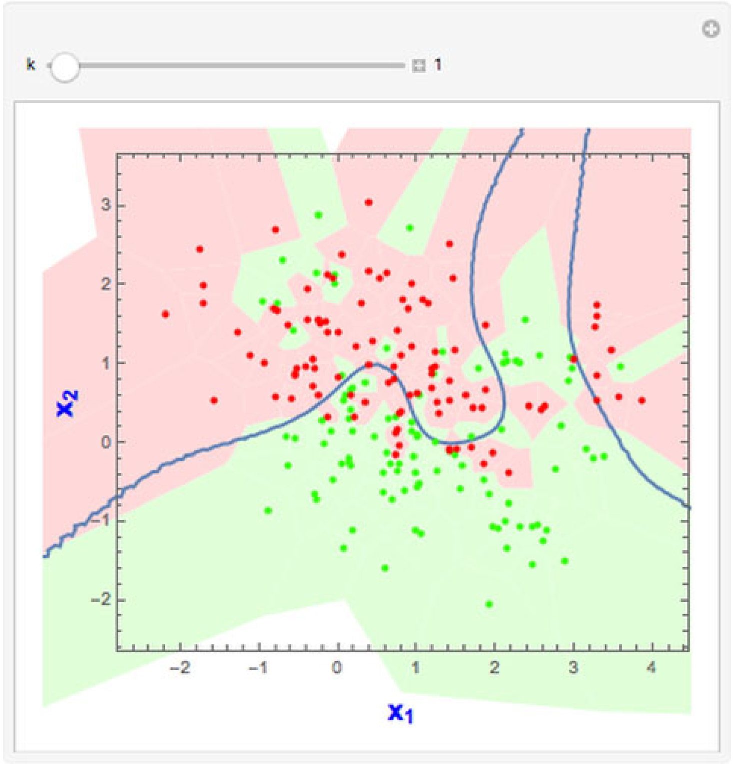
Our hypothesis

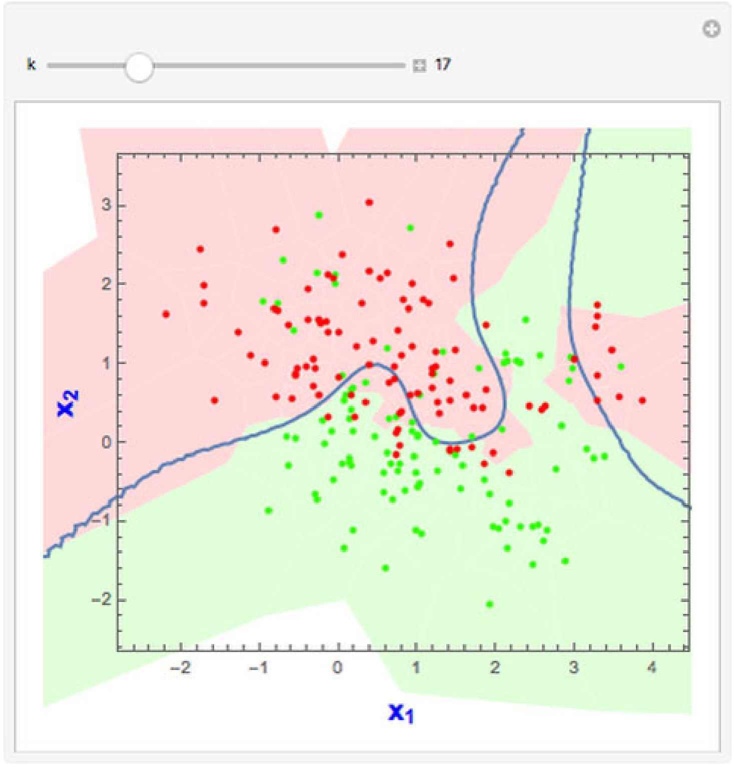
More Concrete Example



Choosing K

- Making K too small fits the output to the noise in the dataset (overfitting)
- Too large of K can make decision boundaries in classification indistinct (underfitting)
- Choose K empirically using cross-validation





Properties of k-Nearest Neighbors

Pros:

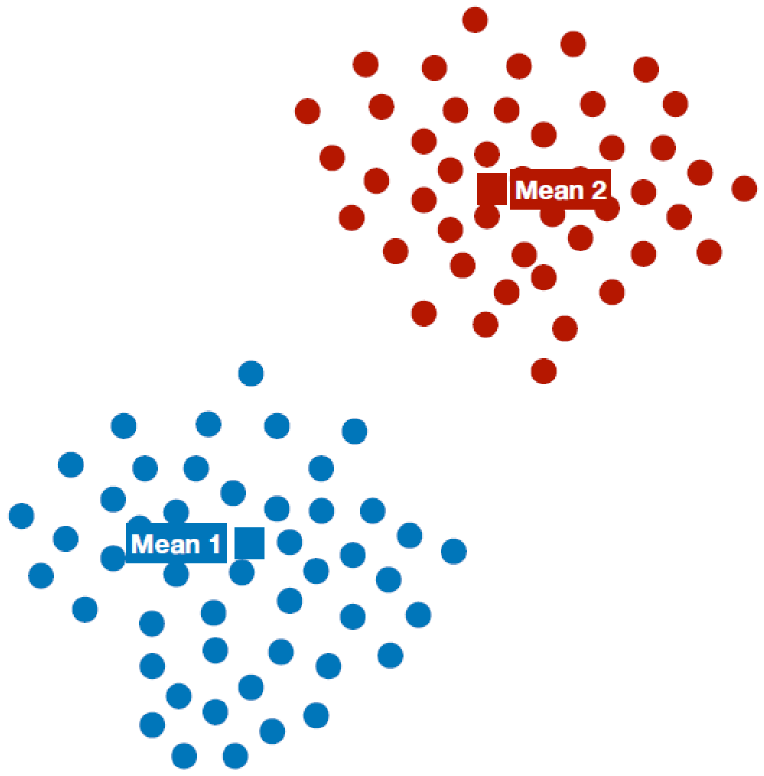
- Robust to noise
- Very effective when training data is sufficient
- Customized to each query
- Can handle complex decision boundaries and functions by considering the query instance when deciding how to generalize
- Easily adapts when new training data is added

Cons:

- How to weight different dimensions or identify irrelevant dimensions?
- Computationally expensive to label a new query
- Evaluation time grows with the dataset size
- High space requirements (must retain each training example)
- No parameterized model

k-Means Algorithm

- We start with unlabeled data
    1. Randomly select means
    2. Calculate distance to means for every data point
    3. Assign class labels based upon shortest distance
    4. Update means and repeat until ***convergence***



k-Means Algorithm

- Assign class labels based upon distance to the means

$$c_i \equiv \arg\min_m \left\| x_i - \mu_m \right\|_2$$

- Update the means:

$$\mu_{m,a} \equiv \frac{\sum_i \mathbf{1}_{c \in m} \cdot x_{i,a}}{\sum_i \mathbf{1}_{c \in m}}$$

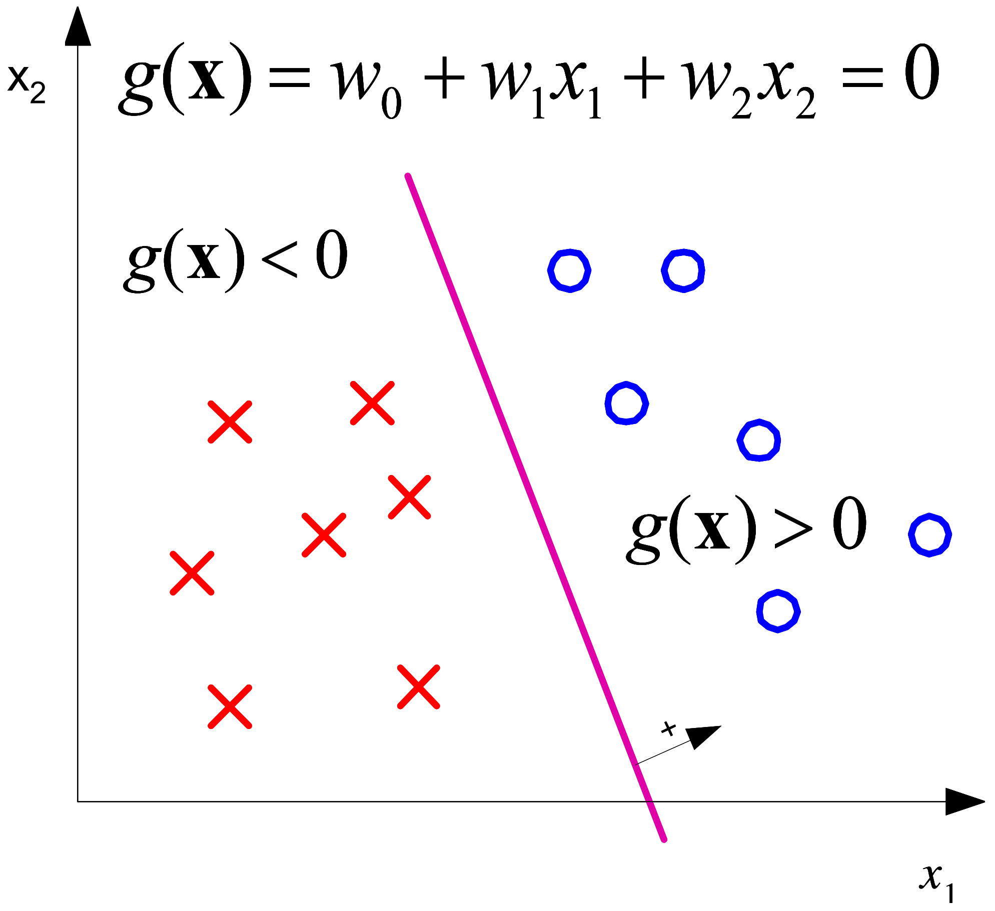
where:

a is the attribute, m is the class label, and

$\mu = (a_1, a_{2,} \ldots a_A)$ is a vector in the attribute-space

Two-Class Classification

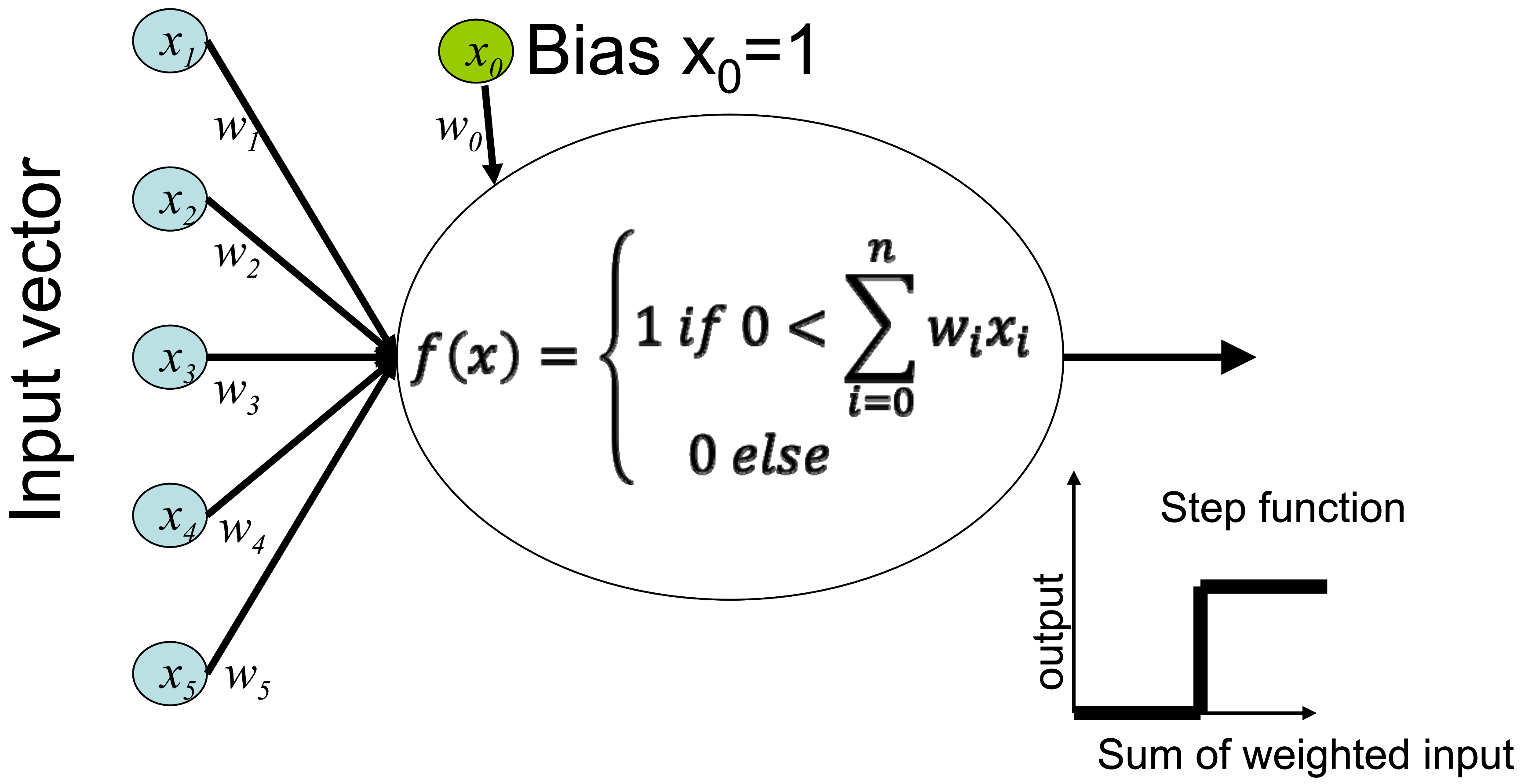
$g(\mathbf{x}) = 0$ defines a decision boundary that splits the space in two



If a line exists that does this without error, the classes are *linearly separable*

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } g(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

A Single Perceptron



Perceptron Algorithm

The decision boundary

$$0 = g(x) = \mathbf{w}^{\mathbf{T}}\mathbf{x}$$

$$m = |D| = \textit{size of data set}$$

The classification function

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } g(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

The weight update algorithm

```
$\mathbf{w} = \textit{some random setting}$
Do
    $k = (k+1)\mathrm{mod}(m)$
    if $h(\mathbf{x}_k) != y_k$
        $\mathbf{w} = \mathbf{w} + \mathbf{x}y$
$\boxed{\text{Until } \forall k,\ h(\mathbf{x}_k) = y_k}$
```

Warning: Only guaranteed to terminate if classes are linearly separable!

This means you have to add another exit condition for when you've gone through the data too many times and suspect you'll never terminate.

Precision vs Recall

Classifiers are often evaluated with an eye towards their being search engines. (e.g. labeling documents as either relevant or not to a search query). In this case people often use the following measures:

$$precision \qquad p = \frac{tp}{tp + fp}$$

$$recall \qquad r = \frac{tp}{tp + fn}$$

$$F - measure \qquad F = 2\frac{p \cdot r}{p + r}$$

True Classification

Machine's Classification

	True	False
True	**True positive (tp)**	**False positive (fp)**
False	**False negative (fn)**	**True negative (tn)**

Gradient Descent Pseudocode

```
Initialize $\theta^{(0)}$
Repeat until convergence:
    $\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_{\theta^{(t)}} L(X, Y; \theta^{(t)})$
Return $\theta^{(t_{max})}$
```

Design Choices:

- Initialization of θ
- Convergence criteria
- Choosing a loss function
- How much data to use during each step (batch size)
- Step size (η)

Stochastic, Batch, Mini-Batch Descent

- In **batch gradient descent**, the loss is a function of both the parameters and the set of all training data **D.**
  (What if if |**D**| > memory?)

- In **stochastic gradient descent**, los is a function of the parameters and a different single random training sample at each iteration. (How does the gradient change when you change **D** at every step?)

- In **mini-batch gradient descent**, random subsets of the data (e.g. 100 examples) are used at each step in the iteration.

Loss Functions

A good objective (loss) function $L(\underset{\substack{\uparrow \\ \text{data}}}{\mathrm{X}}, \underset{\substack{\uparrow \\ \text{labels}}}{\mathrm{Y}}; \underset{\substack{\nwarrow \\ \text{parameters}}}{\theta})$

Required
- $L(\mathrm{X}, \mathrm{Y}; \theta) \geq 0$
- $L(\mathrm{X}, \mathrm{Y}; \theta)$ decreases as performance improves

Required for gradient descent
- $L(\mathrm{X}, \mathrm{Y}; \theta)$ is differentiable*, with respect to θ

helpful For gradient descent
- The gradient of L is bounded … $\mathbf{0} < |\nabla L| \ll \infty$

*or subdifferentiable

Revisiting Overfitting: Regularization

- Big idea (**Occam's Razor**) – Given two models with equal performance, prefer the *simpler* model.
  - E.g., models with fewer parameters or smaller coefficients

- Regularization can be applied to any loss function

$$L_R(X, Y; \theta) = L(X, Y; \theta) + \lambda R(\theta)$$

- The amount of regularization is controlled by the hyperparameter λ

L1- and L2-regularization

- Recall the l_p-norm:

$$\ell_p(\theta) = \sqrt[p]{\sum_{i=1}^{d} |\theta_i|^p}$$

- l_1-regularization penalizes high values of the l_1-norm of the model parameters:

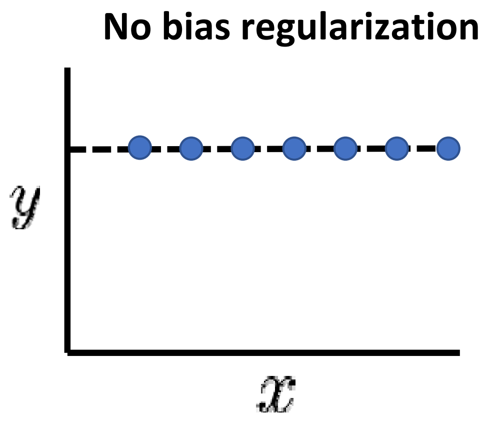
$$R_1(\theta) = \sum_{i=1}^{d} |\theta_i|$$

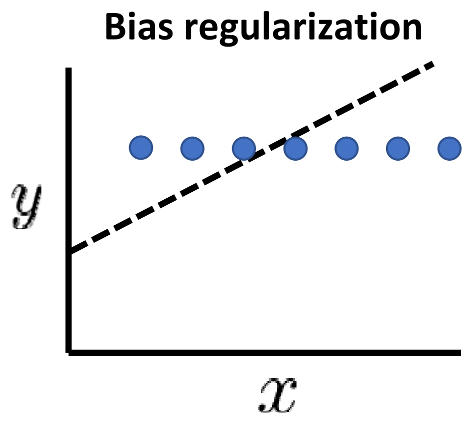
- l_2-regularization penalizes high values of the l_2-norm:

$$R_2(\theta) = \frac{1}{2} \sum_{i=1}^{d} |\theta_i|^2$$

Regularization and offset (aka bias)

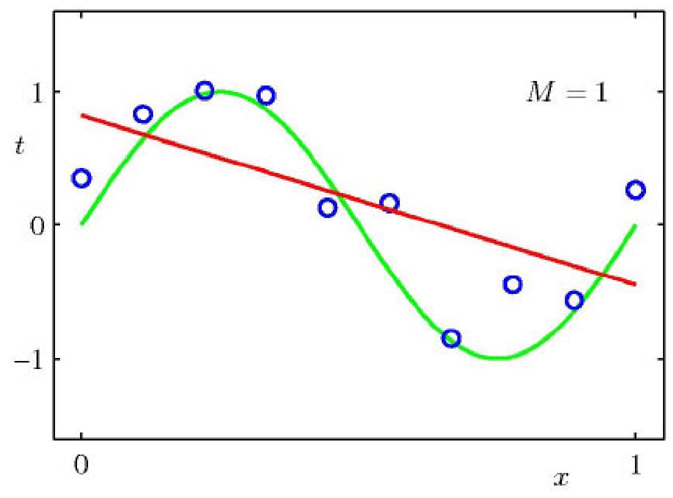
- Recall that "regularizing" a model parameter means encouraging that model parameter to tend towards 0.
- How would a linear model represent horizontal line?
- How does shrinking the bias affect its ability to do so?

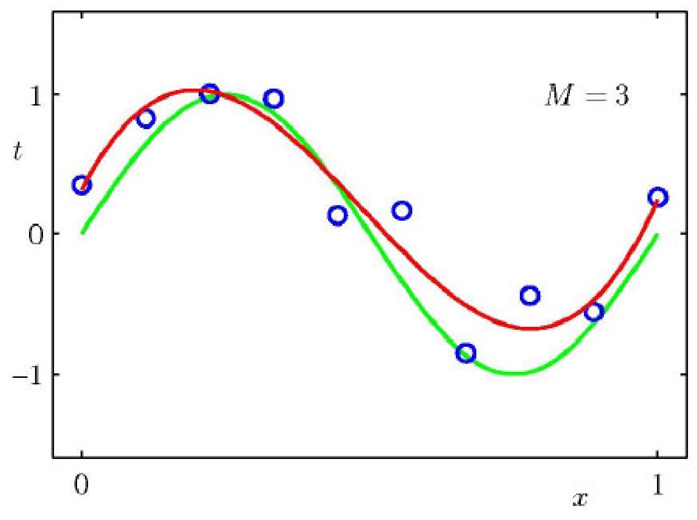


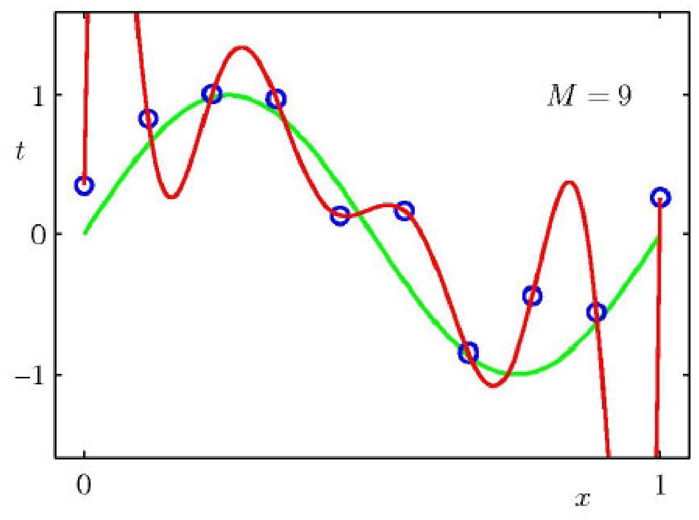


Don't regularize the bias term!

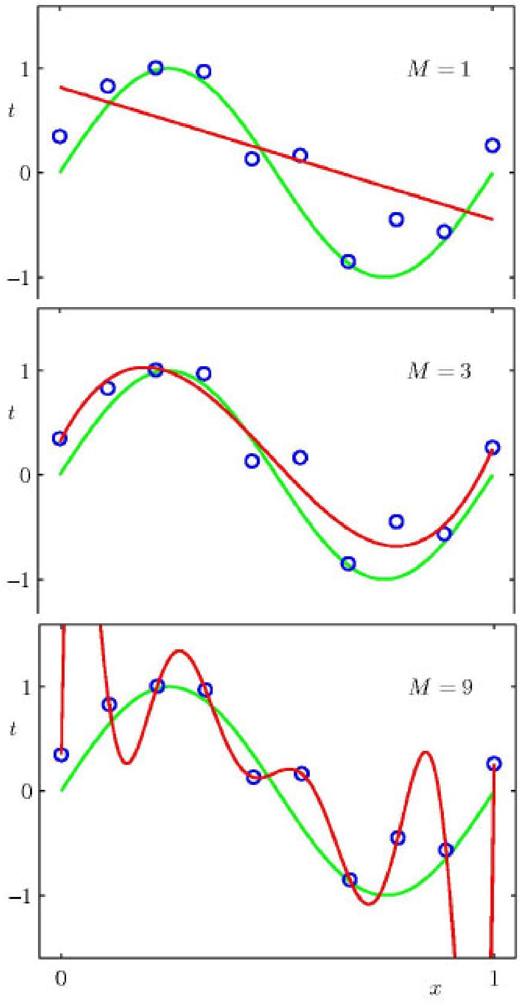
Under/Over Fitting

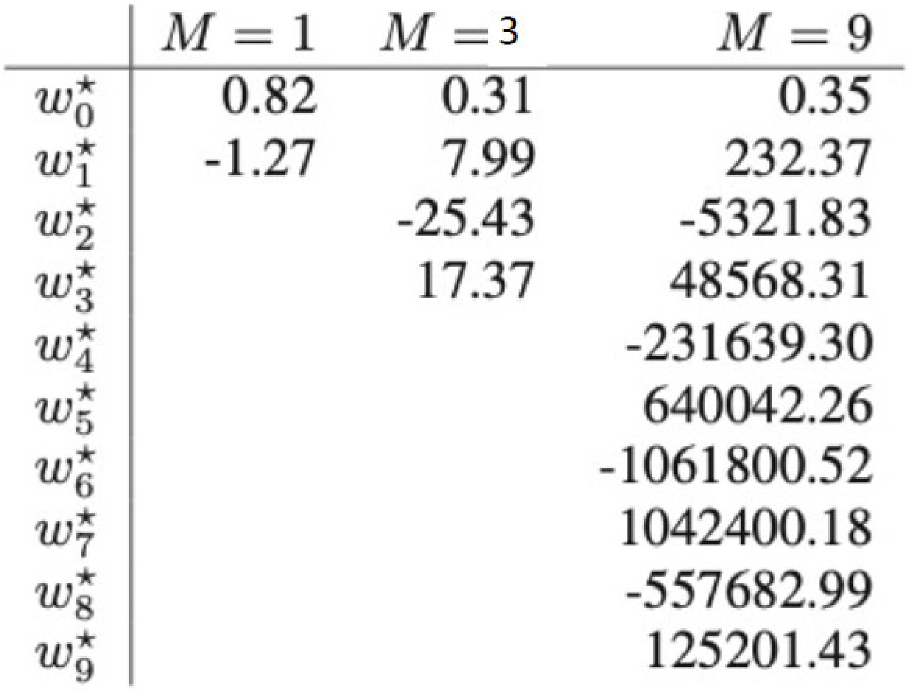






Motivating Regularization





	$M = 1$	$M = 3$	$M = 9$
$w_0^\star$	0.82	0.31	0.35
$w_1^\star$	-1.27	7.99	232.37
$w_2^\star$		-25.43	-5321.83
$w_3^\star$		17.37	48568.31
$w_4^\star$			-231639.30
$w_5^\star$			640042.26
$w_6^\star$			-1061800.52
$w_7^\star$			1042400.18
$w_8^\star$			-557682.99
$w_9^\star$			125201.43

Feed Forward Neural Network: Basic Formulation

number of outputs

number of hidden layer units

number of inputs per sample

$$\hat{y}_k = g\left(\sum_{j=0}^{M} w_{jk}\, f\left(\overbrace{\sum_{i=0}^{N} w_{ij}\, x_i + b_j}^{\text{hidden-layer activation}}\right) + b_k\right)$$

output bias term

input bias term

input data

input-to-hidden weight matrix
(i rows by j columns)

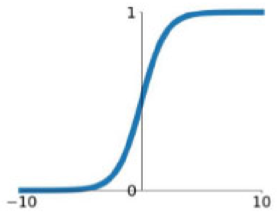
activation function: sigmoid, tanh(), ReLU, etc.

real-value network output

Activation Functions

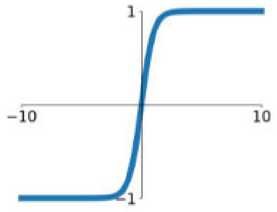
Sigmoid

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



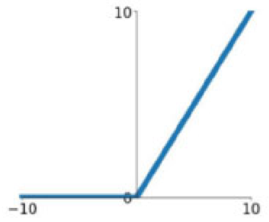
tanh

$\tanh(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$



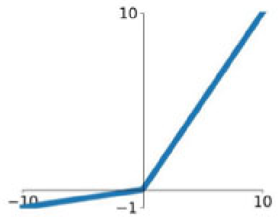
ReLU

$\max(0, x)$



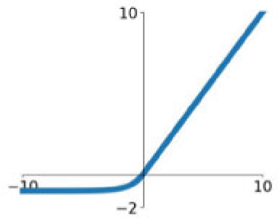
Leaky ReLU

$\max(0.1x, x)$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Output Layers

A variety of output layers can be applied to the basic neural network architecture:

- Regression: Output is a real-valued number (e.g., linear regression in statistics, or previous example).

- Binary-Classification: Applies a threshold to a regression output to decide between classes :

$$class = \begin{cases} y_k > 0.50 \rightarrow \; healthy \\ y_k \leq 0.50 \rightarrow \; unhealthy \end{cases}$$

- Multi-Class Classification: The output-layer activation is replaced with the softmax function and a probability distribution is generated across k elements:

$$\hat{p}_k = \mathtt{softmax}\left(\sum_{j=0}^{M} w_{jk} f\left(\sum_{i=0}^{N} w_{ij} x_i + b_j \right) + b_k \right) \quad \text{and} \quad \mathtt{softmax}(z_k) = \frac{e^{z_k}}{\sum_{c \in C} e^{z_c}}$$

Training

Step 1: The neural network is presented with a series input/output pairs $\langle \overline{x}_s, y_s \rangle \in S$ from the training set.

Step 2: The neural network is used to perform inference (a forward pass) to generate an estimated output $\hat{y}_s$.

Step 3: A cost function (see next slides) is used to quantify how well (or poorly) the neural network performed by comparing $\hat{y}_s$ and y_s .

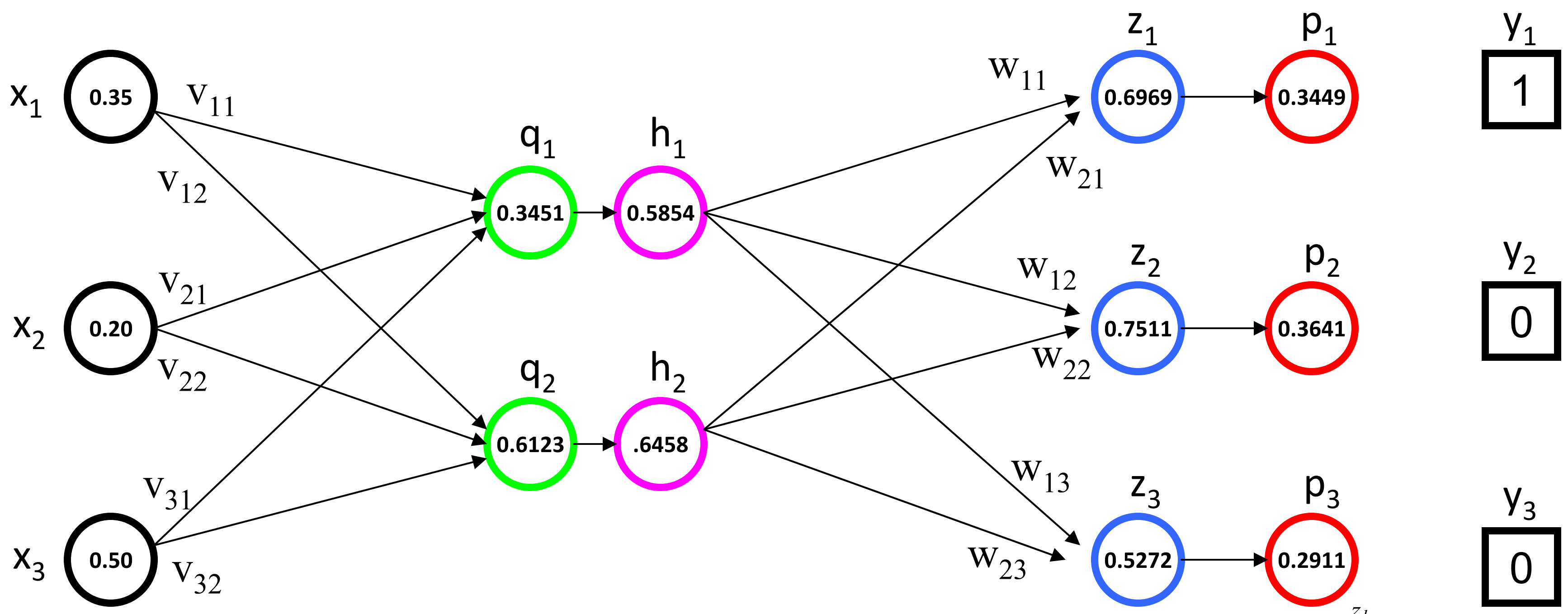
Step 4: Back propagation (see next slides) is performed by calculating gradients with respect to each trainable variable, which are used to update trainable variables:

$$\hat{y}_k = g\left(\sum_{j=0}^{M} w_{jk} f\left(\sum_{i=0}^{N} w_{ij} x_i + b_j\right) + b_k\right)$$

The updates can be performed as:

- Stochastic Gradient Descent: Network parameters are updated for each training example, or
- Batch Gradient Descent: Gradients are accumulated over the entire training set and applied all at once.

Gradient Descent -- Architecture



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

$$p_k = \frac{e^{z_k}}{\sum_{c \in C} e^{z_c}}$$

$$\begin{array}{c} 0.3500 \\ 0.2000 \\ 0.5000 \end{array}$$

Inputs X
(1 x 3)

$$\begin{array}{cc} 0.4963 & 0.7682 \\ 0.0885 & 0.1320 \\ 0.3074 & 0.6341 \end{array}$$

Weights V
(3 x 2)

$$\begin{array}{c} 0.3451 \\ 0.6123 \end{array}$$

Linear Q
(1 x 2)

$$\begin{array}{c} 0.5854 \\ 0.6485 \end{array}$$

Hidden H
(1 x 2)

$$\begin{array}{ccc} 0.4901 & 0.8964 & 0.4556 \\ 0.6323 & 0.3489 & 0.4017 \end{array}$$

Weights W
(2 x 3)

$$\begin{array}{c} 0.6969 \\ 0.7511 \\ 0.5272 \end{array}$$

Logits Z
(1 x 3)

$$\begin{array}{c} 34.5\% \\ 36.4\% \\ 29.1\% \end{array}$$

Probs P
(1 x 3)

$$\begin{array}{c} \textbf{Good} \\ \textbf{Neutral} \\ \textbf{Bad} \end{array}$$

Labels Y
(1 x 3)

Gradient Descent -- Equations

Cost Function: Multi-Class Cross-Entropy

$$L = -\sum_{i=0}^{C} y_i \ln(p_i) + (1 - y_i)\ln(1 - p_i)$$

Activation Function: Sigmoid

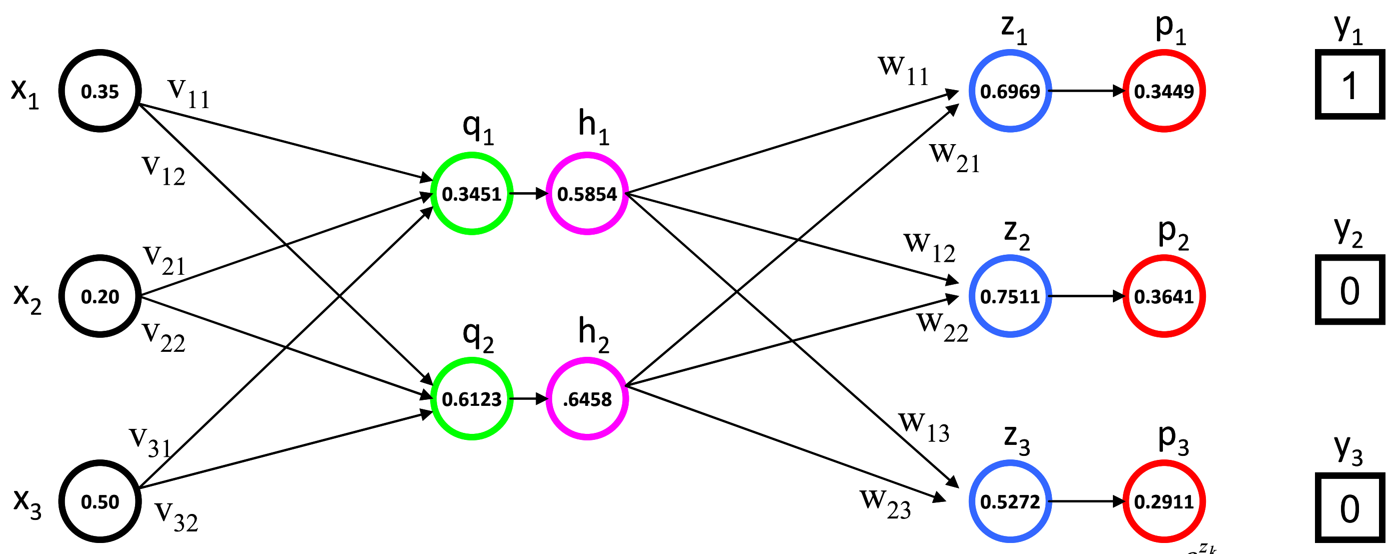
$$\sigma(q) = \frac{1}{1 + e^{-q}}$$

Softmax:

$$p_k = \frac{e^{z_k}}{\sum_{c \in C} e^{z_c}}$$

$$= \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

Gradient Descent -- Forward Pass Calculations



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

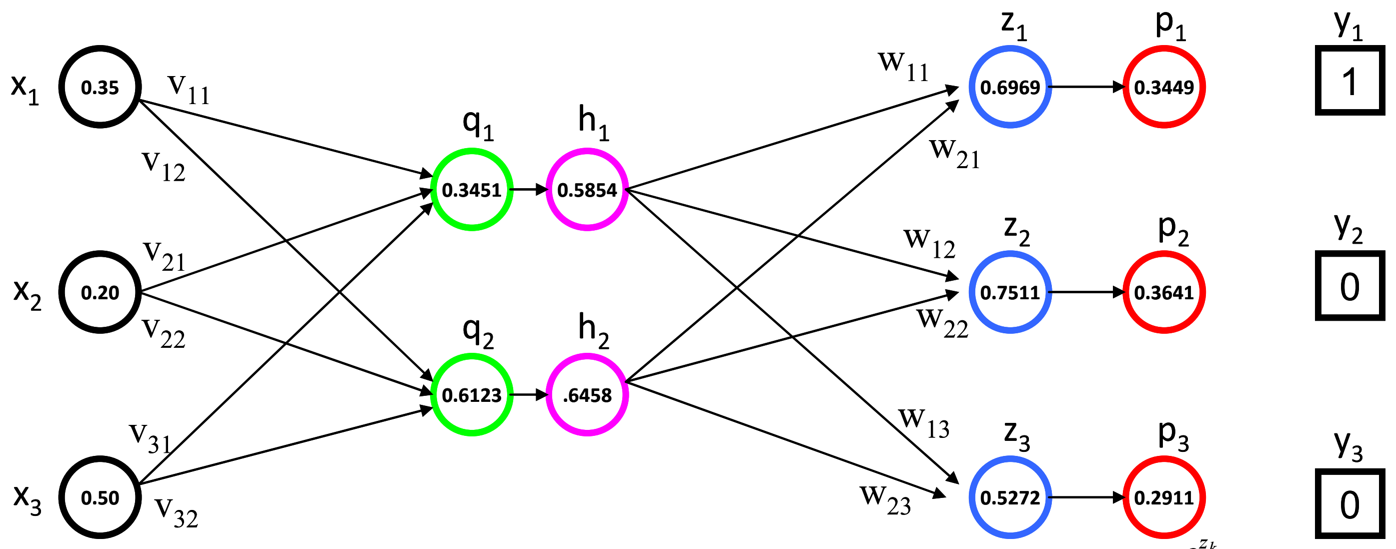
$$p_k = \frac{e^{z_k}}{\sum_{c \in C} e^{z_c}}$$

$$\begin{array}{cccccccc}
\begin{bmatrix}0.3500\\0.2000\\0.5000\end{bmatrix} &
\begin{bmatrix}0.4963 & 0.7682\\0.0885 & 0.1320\\0.3074 & 0.6341\end{bmatrix} &
\begin{bmatrix}0.3451\\0.6123\end{bmatrix} &
\begin{bmatrix}0.5854\\0.6485\end{bmatrix} &
\begin{bmatrix}0.4901 & 0.8964 & 0.4556\\0.6323 & 0.3489 & 0.4017\end{bmatrix} &
\begin{bmatrix}0.6969\\0.7511\\0.5272\end{bmatrix} &
\begin{bmatrix}34.5\%\\36.4\%\\29.1\%\end{bmatrix} &
\begin{bmatrix}\textbf{Good}\\\textbf{Neutral}\\\textbf{Bad}\end{bmatrix} \\
\textit{Inputs } X & \textit{Weights } V & \textit{Linear } Q & \textit{Hidden } H & \textit{Weights } W & \textit{Logits } Z & \textit{Probs } P & \textit{Labels } Y \\
(1 \times 3) & (3 \times 2) & (1 \times 2) & (1 \times 2) & (2 \times 3) & (1 \times 3) & (1 \times 3) & (1 \times 3)
\end{array}$$

$$\begin{bmatrix}0.3500 & 0.2000 & 0.500\end{bmatrix} \bullet \begin{bmatrix}0.4963 & 0.7682\\0.0885 & 0.1320\\0.3074 & 0.6341\end{bmatrix} = \begin{bmatrix}0.3451\\0.6123\end{bmatrix}$$

0.4963 X 0.3500 + 0.0885 X 0.2000 + 0.3074 X 0.500 = 0.3451

Gradient Descent -- Forward Pass Calculations



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

$$p_k = \frac{e^{z_k}}{\sum_{c \in C} e^{z_c}}$$

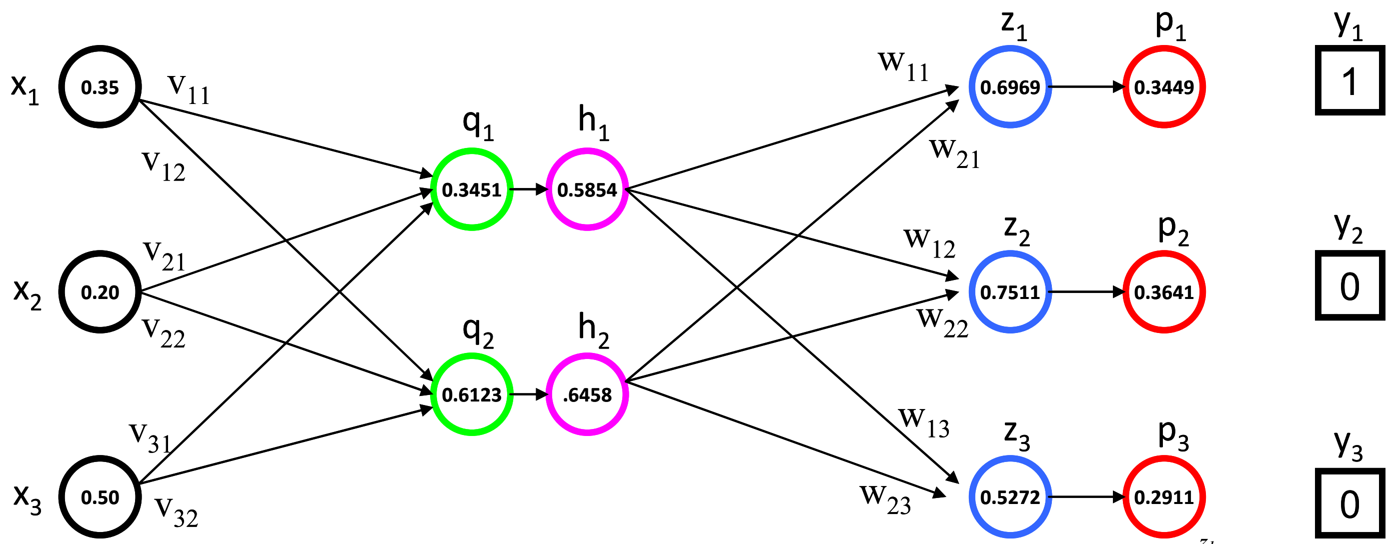
$$\begin{bmatrix} 0.3500 \\ 0.2000 \\ 0.5000 \end{bmatrix} \quad \begin{bmatrix} 0.4963 & 0.7682 \\ 0.0885 & 0.1320 \\ 0.3074 & 0.6341 \end{bmatrix} \quad \begin{bmatrix} 0.3451 \\ 0.6123 \end{bmatrix} \quad \begin{bmatrix} 0.5854 \\ 0.6485 \end{bmatrix} \quad \begin{bmatrix} 0.4901 & 0.8964 & 0.4556 \\ 0.6323 & 0.3489 & 0.4017 \end{bmatrix} \quad \begin{bmatrix} 0.6969 \\ 0.7511 \\ 0.5272 \end{bmatrix} \quad \begin{bmatrix} 34.5\% \\ 36.4\% \\ 29.1\% \end{bmatrix} \quad \begin{bmatrix} \textbf{Good} \\ \textbf{Neutral} \\ \textbf{Bad} \end{bmatrix}$$

Inputs X	*Weights V*	*Linear Q*	*Hidden H*	*Weights W*	*Logits Z*	*Probs P*	*Labels Y*
`(1 x 3)`	`(3 x 2)`	`(1 x 2)`	`(1 x 2)`	`(2 x 3)`	`(1 x 3)`	`(1 x 3)`	`(1 x 3)`

$$\sigma\left(\begin{bmatrix} 0.3451 \\ 0.6123 \end{bmatrix}\right) = \begin{bmatrix} 0.5854 \\ 0.6458 \end{bmatrix}$$

1 / (1 + exp(-0.3451) = 0.5854

Gradient Descent -- Forward Pass Calculations



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

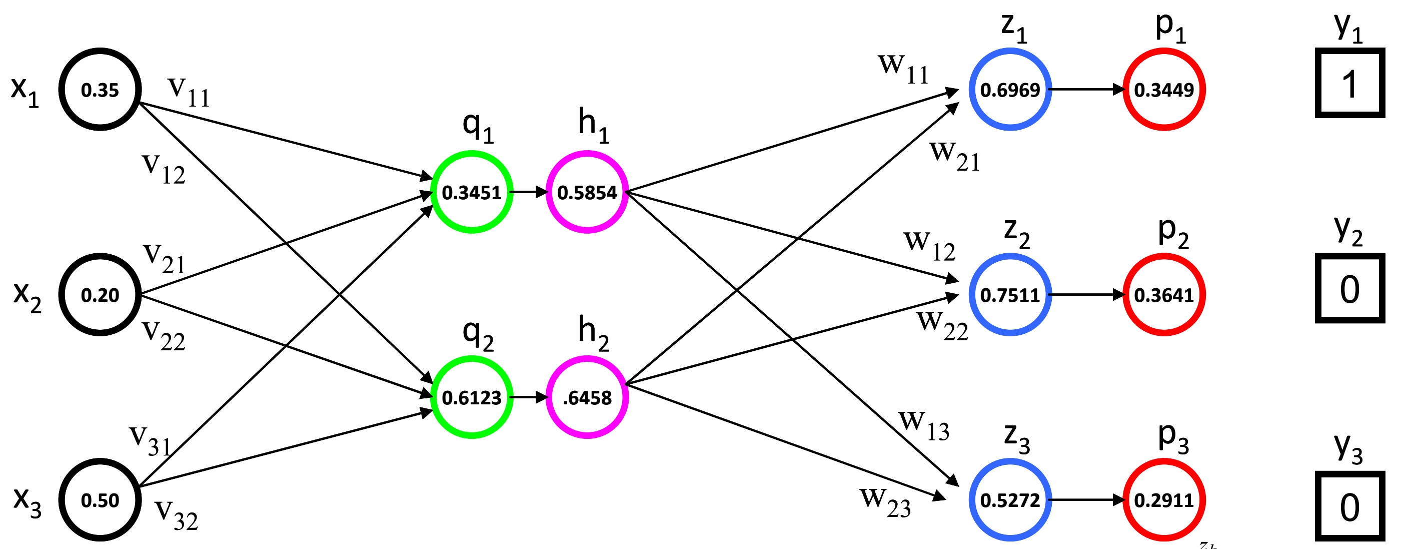
$$p_k = \frac{e^{z_k}}{\sum_{c \in C} e^{z_c}}$$

0.3500 0.2000 0.5000	0.4963 0.7682 0.0885 0.1320 0.3074 0.6341	0.3451 0.6123	0.5854 0.6485	0.4901 0.8964 0.4556 0.6323 0.3489 0.4017	0.6969 0.7511 0.5272	34.5% 36.4% 29.1%	**Good** **Neutral** **Bad**
Inputs X `(1 x 3)`	*Weights V* `(3 x 2)`	*Linear Q* `(1 x 2)`	*Hidden H* `(1 x 2)`	*Weights W* `(2 x 3)`	*Logits Z* `(1 x 3)`	*Probs P* `(1 x 3)`	*Labels Y* `(1 x 3)`

$$\begin{bmatrix} 0.5854 & 0.6485 \end{bmatrix} \bullet \begin{bmatrix} 0.4901 & 0.8964 & 0.4556 \\ 0.6323 & 0.3489 & 0.4017 \end{bmatrix} = \begin{bmatrix} 0.6969 \\ 0.7511 \\ 0.5272 \end{bmatrix}$$

0.4901 X 0.5854 + 0.6323 X 0.6458 = 0.6969

Gradient Descent -- Forward Pass Calculations



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

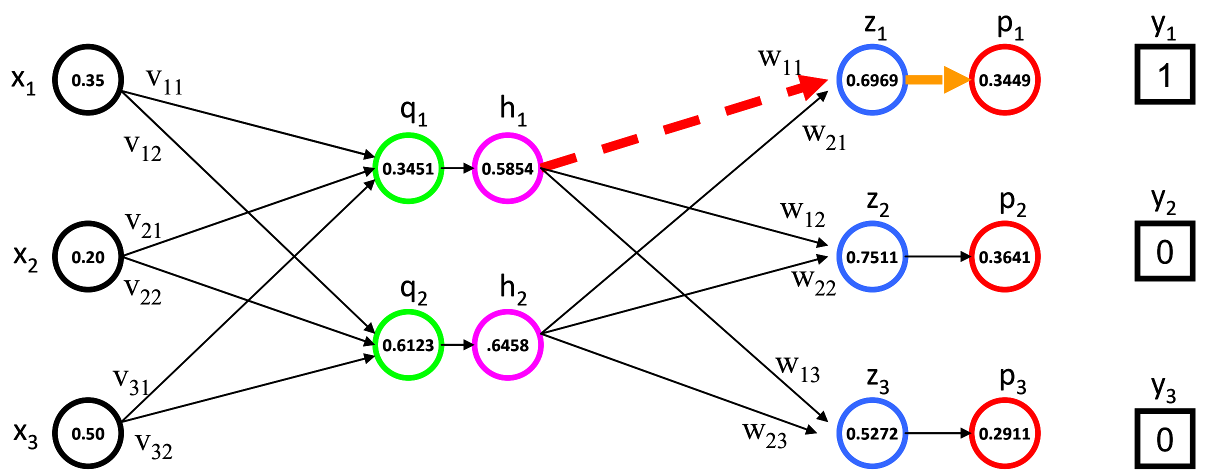
$$p_k = \frac{e^{z_k}}{\sum_{c \in C} e^{z_c}}$$

0.3500	0.4963 0.7682	0.3451	0.5854	0.4901 0.8964 0.4556	0.6969	34.5%	Good
0.2000	0.0885 0.1320	0.6123	0.6485	0.6323 0.3489 0.4017	0.7511	36.4%	Neutral
0.5000	0.3074 0.6341				0.5272	29.1%	Bad
Inputs X	*Weights V*	*Linear Q*	*Hidden H*	*Weights W*	*Logits Z*	*Probs P*	*Labels Y*
(1 x 3)	(3 x 2)	(1 x 2)	(1 x 2)	(2 x 3)	(1 x 3)	(1 x 3)	(1 x 3)

$$\text{softmax}\left(\begin{bmatrix}0.6969\\0.7511\\0.5272\end{bmatrix}\right) = \begin{bmatrix}34.5\%\\36.4\%\\29.1\%\end{bmatrix}$$

exp(0.6969)/[exp(0.6969) + exp(0.7511) + exp(0.5272)] = 0.3449

Gradient Descent -- Gradient Calculation



Chain Rule:

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x}$$

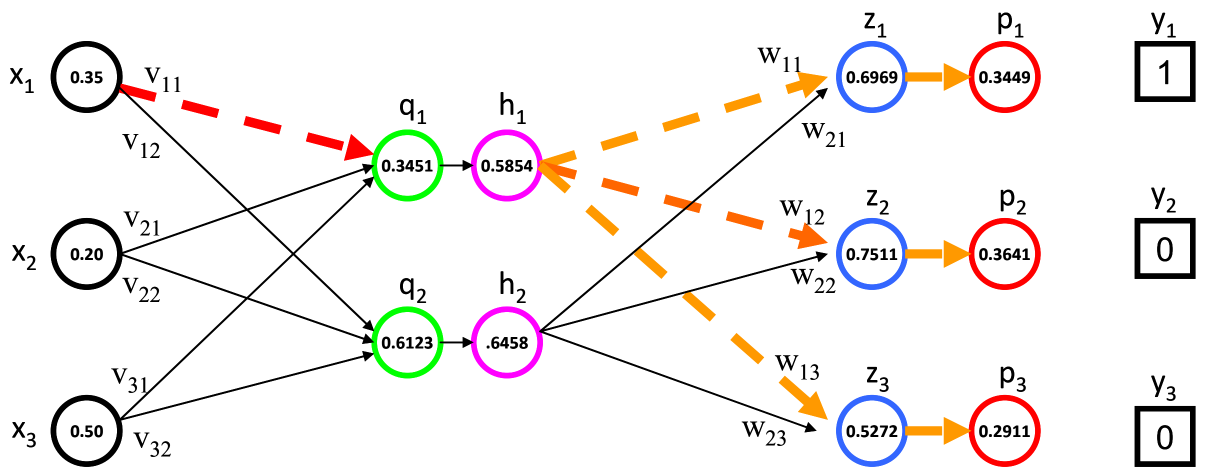
$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial p_1} \cdot \frac{\partial p_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_{11}}$$

$$= -\frac{1}{p_1} \cdot \frac{e^{z_1} \cdot (e^{z_2} + e^{z_3})}{(e^{z_1} + e^{z_2} + e^{z_3})^2} \cdot h_1$$

$$= -\frac{1}{0.3449} \cdot \frac{e^{0.6969} \cdot (e^{0.7511} + e^{0.5272})}{(e^{0.6969} + e^{0.7511} + e^{0.5272})^2} \cdot 0.5854$$

$$= -0.3835$$

Gradient Descent -- Gradient Calculation



$$\frac{\partial L}{\partial v_{11}} = \frac{\partial L}{\partial p_1}\cdot\frac{\partial p_1}{\partial z_1}\cdot\frac{\partial z_1}{\partial h_1}\cdot\frac{\partial h_1}{\partial q_1}\cdot\frac{\partial q_1}{\partial v_{11}} + \frac{\partial L}{\partial p_2}\cdot\frac{\partial p_2}{\partial z_2}\cdot\frac{\partial z_2}{\partial h_1}\cdot\frac{\partial h_1}{\partial q_1}\cdot\frac{\partial q_1}{\partial v_{11}} + \frac{\partial L}{\partial p_3}\cdot\frac{\partial p_3}{\partial z_3}\cdot\frac{\partial z_3}{\partial h_1}\cdot\frac{\partial h_1}{\partial q_1}\cdot\frac{\partial q_1}{\partial v_{11}}$$

$$= -\frac{1}{p_1}\cdot\frac{e^{z_1}\cdot(e^{z_2}+e^{z_3})}{(e^{z_1}+e^{z_2}+e^{z_3})^2}\cdot w_{11}\cdot\sigma(q_1)(1-\sigma(q_1))\cdot x_1 +$$

$$\frac{1}{(1-p_2)}\cdot\frac{e^{z_2}\cdot(e^{z_1}+e^{z_3})}{(e^{z_1}+e^{z_2}+e^{z_3})^2}\cdot w_{12}\cdot\sigma(q_1)(1-\sigma(q_1))\cdot x_1 +$$

$$\frac{1}{1-p_3}\cdot\frac{e^{z_3}\cdot(e^{z_1}+e^{z_2})}{(e^{z_1}+e^{z_2}+e^{z_3})^2}\cdot w_{13}\cdot\sigma(q_1)(1-\sigma(q_1))\cdot x_1$$

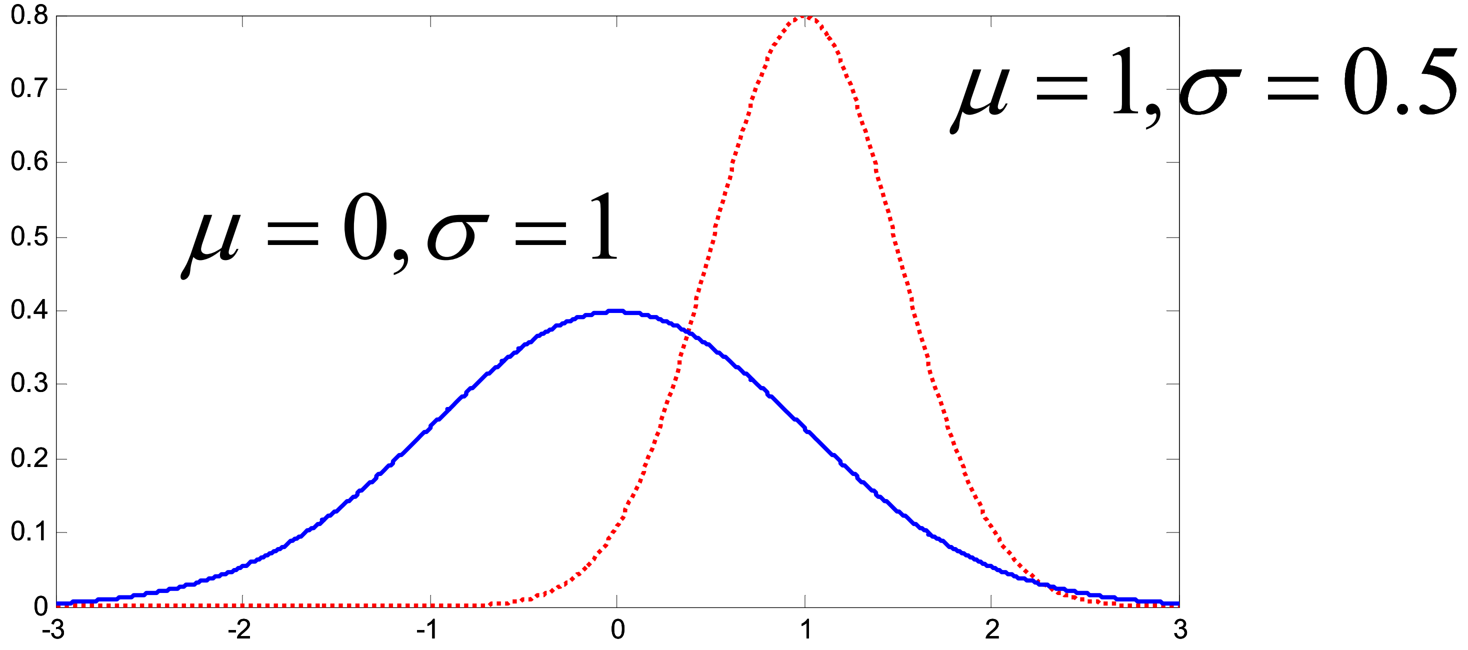
$$= -0.0118$$

The Normal (Gaussian) Distribution

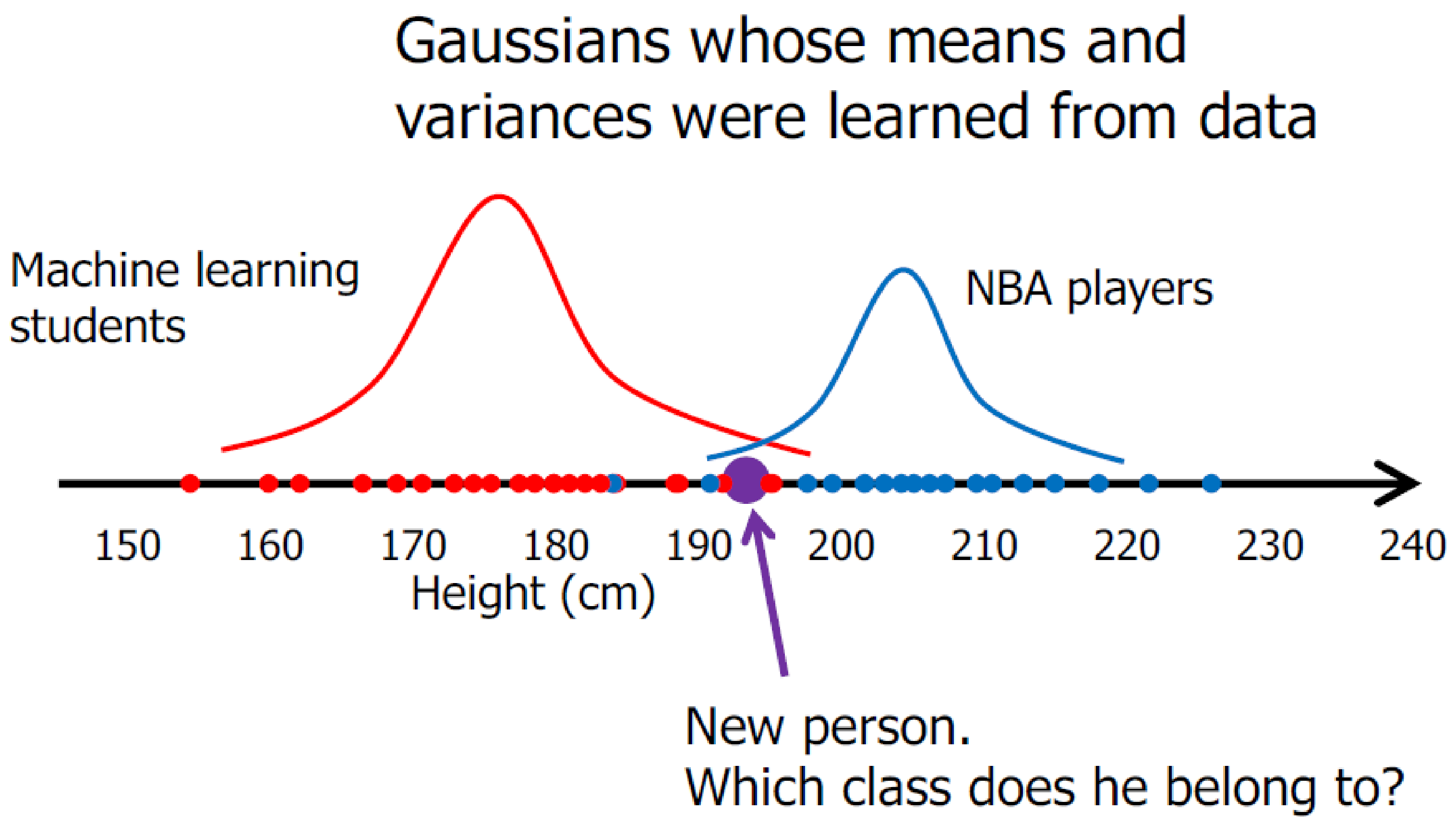
$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

mean

variance

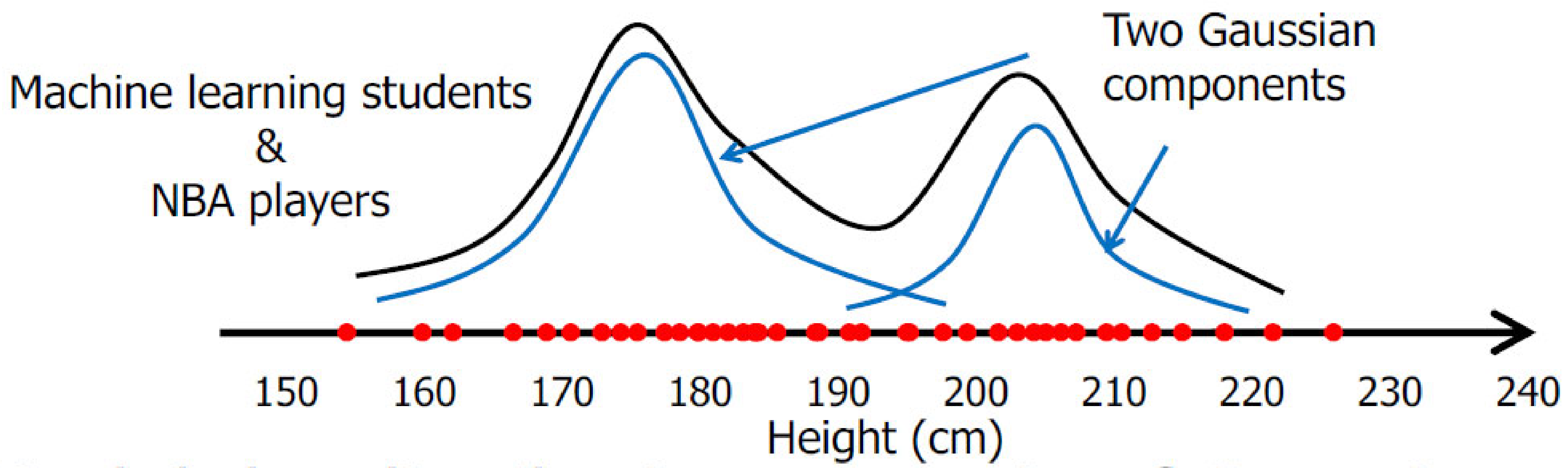


Using Generative Models for Classification


Gaussians whose means and variances were learned from data
Machine learning students
NBA players
150
160
170
180
190
200
210
220
230
240
Height (cm)
New person.
Which class does he belong to?

Answer: the class that calls him most probable.

Gaussian Mixture Model (GMM)



Model the distribution as a mix of Gaussians

$$P(x) = \sum_{j=1}^{K} P(z_j) P(x \mid z_j)$$

x is the observed value

z_j is a Boolean saying whether Gaussian j "made" x

What Are We Optimizing?

$$P(x) = \sum_{j=1}^{K} P(z_j) P(x \mid z_j)$$

Notating $P(z_j)$ as weight w_j and using the Normal (a.k.a. Gaussian) distribution $N(\mu_j, \sigma_j^2)$ gives us...

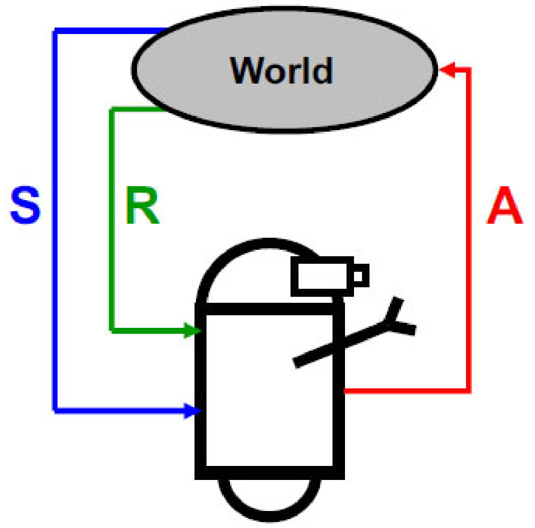
$$= \sum_{j=1}^{K} w_j N(x \mid \mu_j, \sigma_j^2) \quad \text{such that } 1 = \sum_{j=1}^{K} w_j$$

This gives 3 variables per Gaussian to optimize:

w_j, μ_j, σ_j

General Reinforcement Learning Framework

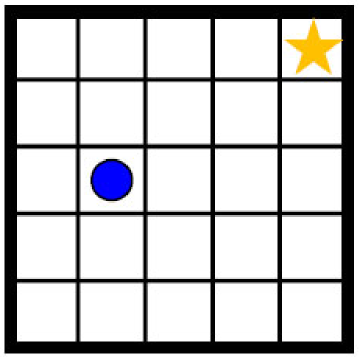
1. Observe state, s_t
2. Decide on an action, a_t
3. Perform action
4. Observe new state, s_{t+1}
5. Observe reward, r_{t+1}
6. Learn from experience
7. Repeat



Goal: Find a control policy that will maximize the observed rewards over the lifetime of the agent

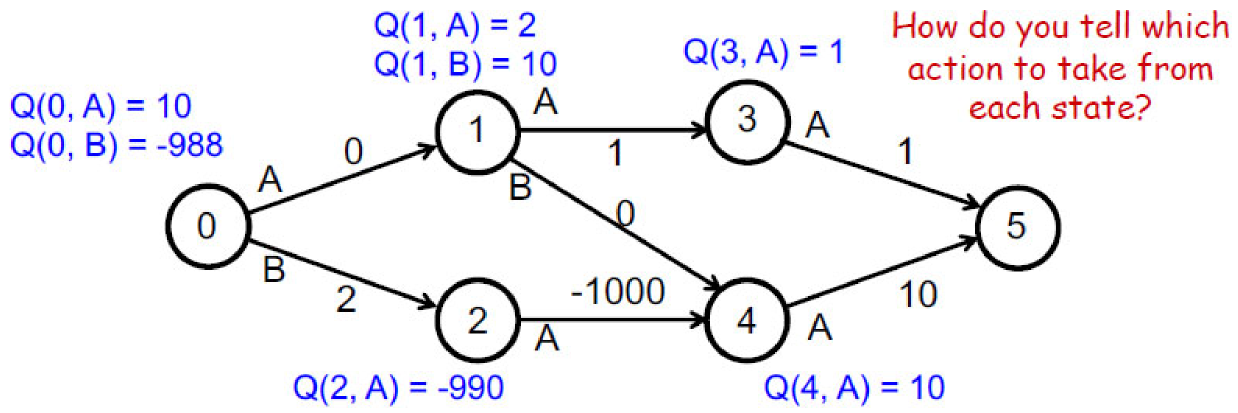
A Canonical Example: Gridworld

- States are grid cells
- 4 actions: N, S, E, W
- Suppose we want the agent to get to the top-right corner, as fast as possible.
- What should our rewards be?

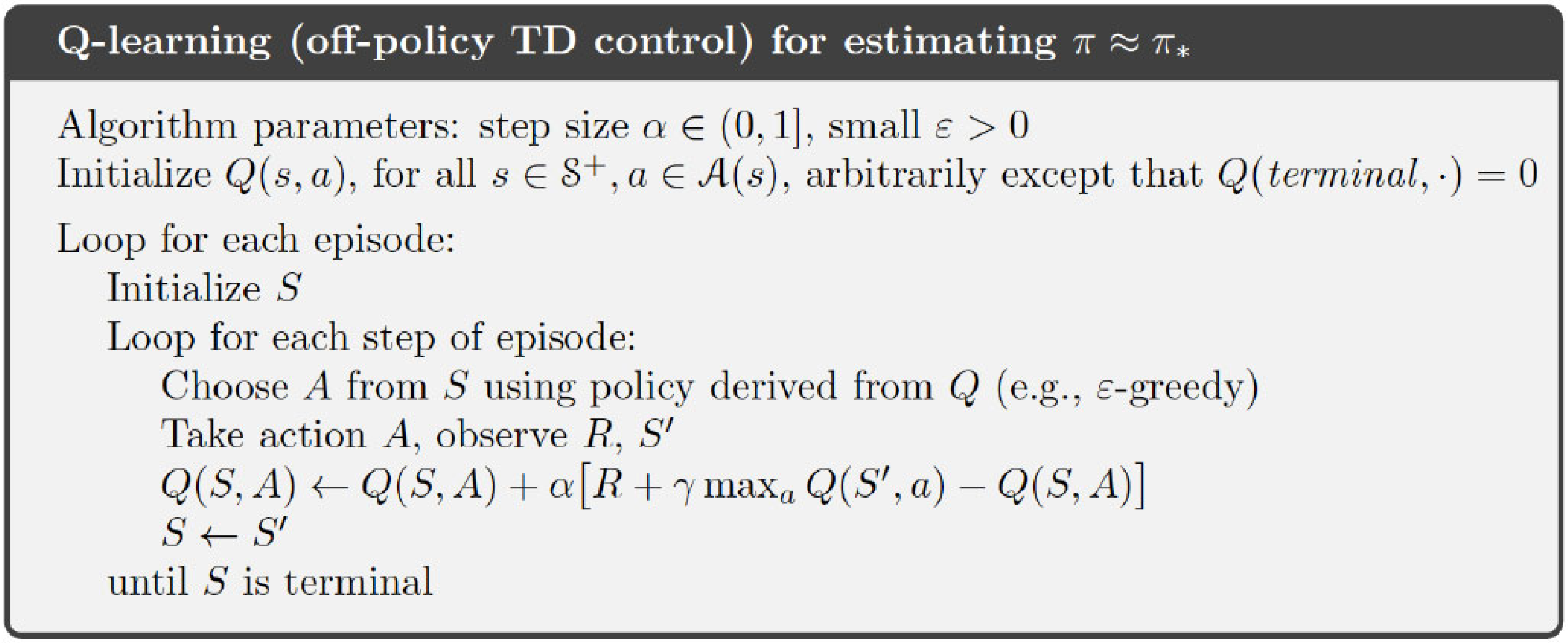


Q-Functions

- Define value without specifying the policy
  - Specify the value of taking action A from state S and then performing optimally, thereafter



Q-Learning Algorithm



Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

$0 \leq \alpha \leq 1$ is the learning rate & we should decay α, just like in TD

Machine Learning Algorithms

1. Decision Trees (ID3 Algorithm)
2. Random Forests
3. Linear Regression
4. Polynomial Regression
5. K-Nearest Neighbors
6. K-Means Clustering
7. Soft K-Means Algorithm
8. Linear Discriminants
9. Perceptrons and Perceptron Algorithm
10. Feed Forward Neural Networks
11. Gaussian Mixture Models
12. Reinforcement Learning
13. Recurrent, Convolutional and GAN Architectures
14. Support Vector Machines
15. Genetic Algorithms
16. Active Learning
17. Boosting

Machine Learning Concepts

1. Classification and Regression
2. Classification ⇔ Regression
3. Supervised and Unsupervised Learning
4. Generalized Learning Task
   (Machine Learning in a “Nutshell”)
5. Inductive Bias
6. Linear Separable and Kernel Functions
7. Entropy and Information Gain
8. Constructing Decision Boundaries
9. Train, Validation and Test Sets
10. N-Fold Cross Validation
11. Overfitting and Learning Curves
12. Early Stopping and Pruning
13. Precision-Recall and Confusion Matrices
14. Bias and Variance
15. Softmax Function
16. Data Normalization
17. Hypothesis Space and “True” Mapping Function
18. MSE and Multi-class Cross Entropy
19. Euclidean, Manhattan, Hamming
    and Cosine Distances (among others)
20. L_p Norms
21. Binary and Multi-class Classification
22. Gradient Descent and Regularization
23. Model Parameters and Hyper-parameters
24. Activation Functions
25. Weights, Biases and Matrix Operations
26. Gradients and the Chain Rule
27. Calculation Graphs and Pytorch
28. True Error vs. Sampling Error
29. Bernoulli and Gaussian Distributions
30. Student t-Tests for Hypothesis Testing
31. Discriminative vs. Generative Models
32. Expectation Maximization Algorithm
33. Reinforcement Learning General Framework
34. Reinforcement Learning Reward Schemes
35. Q-Functions and Q-Learning

Questions?

Thank You!