Lecture 4: Softmax, gradient descent, and neural networks

Announcements:

- HW #1 is due tonight, uploaded to Gradescope by 11:59pm. To submit your Jupyter Notebook, print the notebook to a pdf with your solutions and plots filled in. You must also submit your .py files as pdfs. → HW #2 and onwards.
- HW #2 is due Monday, January 29, uploaded to Gradescope.

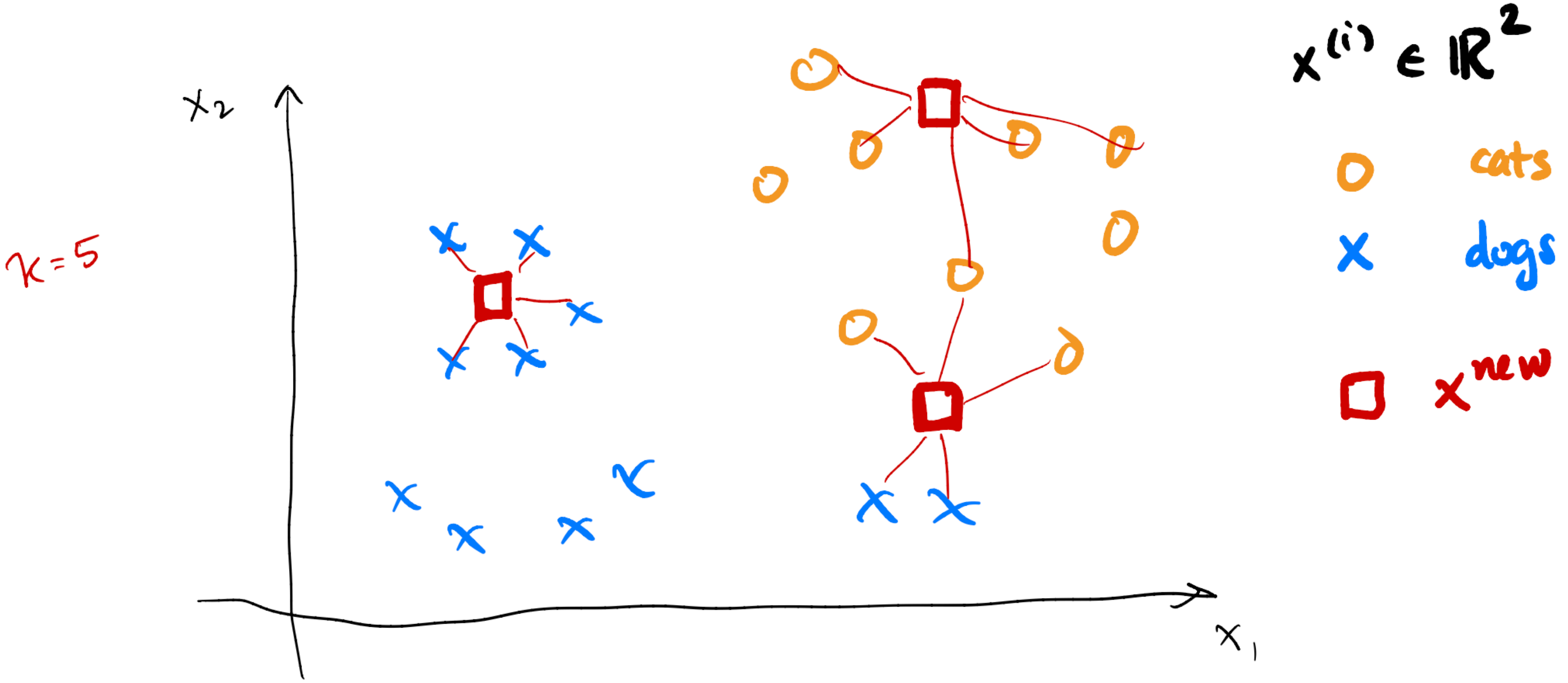
  It will be uploaded tonight.
- Moving forward, the assignments have a good amount of Python coding. Get started early. This assignment will cover k-nearest neighbors and the softmax classifier.

k-nearest neighbors

$x^{(i)} \in \mathbb{R}^{3072}$

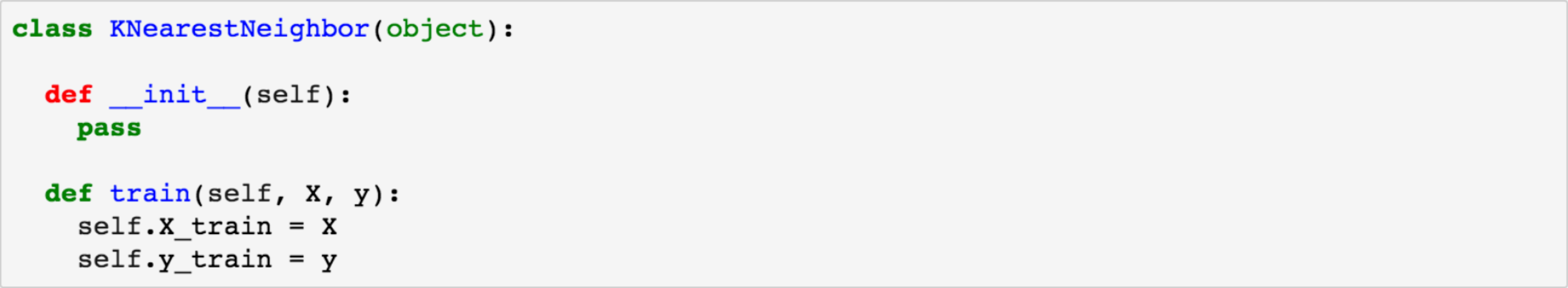
k-nearest neighbors

Intuitively, k-nearest neighbors says to find the k closest points (or nearest neighbors) in the training set, according to an appropriate metric. Each of its k nearest neighbors then vote according to what class it is in, and $\mathbf{x}^{\text{new}}$ is assigned to be the class with the most votes.



k-nearest neighbors

How do we train the classifier?



```
class KNearestNeighbor(object):

  def __init__(self):
    pass

  def train(self, X, y):
    self.X_train = X
    self.y_train = y
```

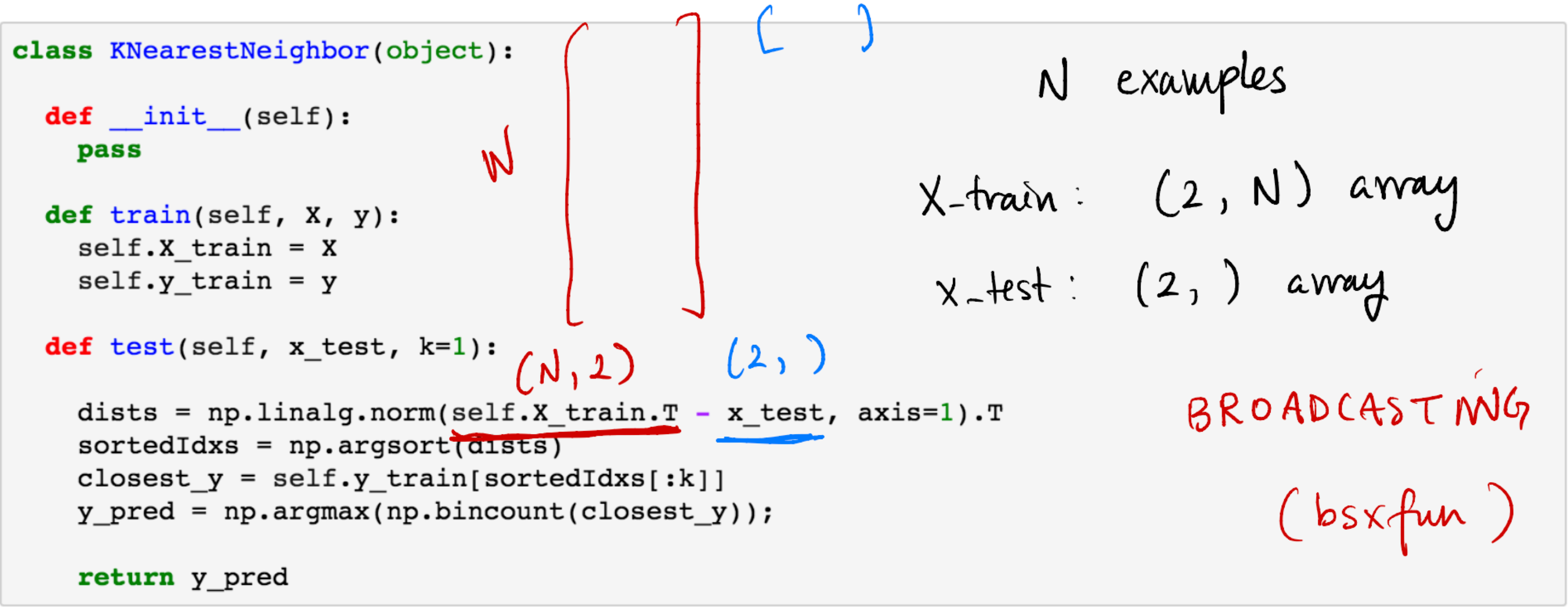
Pros? Simple, fast.

Cons? Memory intensive, b/c we need to store all the input data.

k-nearest neighbors

How do we test a new data point?

$x \in \mathbb{R}^2$



```
class KNearestNeighbor(object):

  def __init__(self):
    pass

  def train(self, X, y):
    self.X_train = X
    self.y_train = y

  def test(self, x_test, k=1):

    dists = np.linalg.norm(self.X_train.T - x_test, axis=1).T
    sortedIdxs = np.argsort(dists)
    closest_y = self.y_train[sortedIdxs[:k]]
    y_pred = np.argmax(np.bincount(closest_y));

    return y_pred
```

$$\begin{array}{cc} & 2 \\ N & \begin{bmatrix} \quad \\ \\ \\ \end{bmatrix} \end{array} \qquad \begin{bmatrix} \quad & \quad \end{bmatrix}$$

$(N, 2)$ $(2,)$

N examples

X_train : $(2, N)$ array

x_test : $(2,)$ array

BROADCASTING

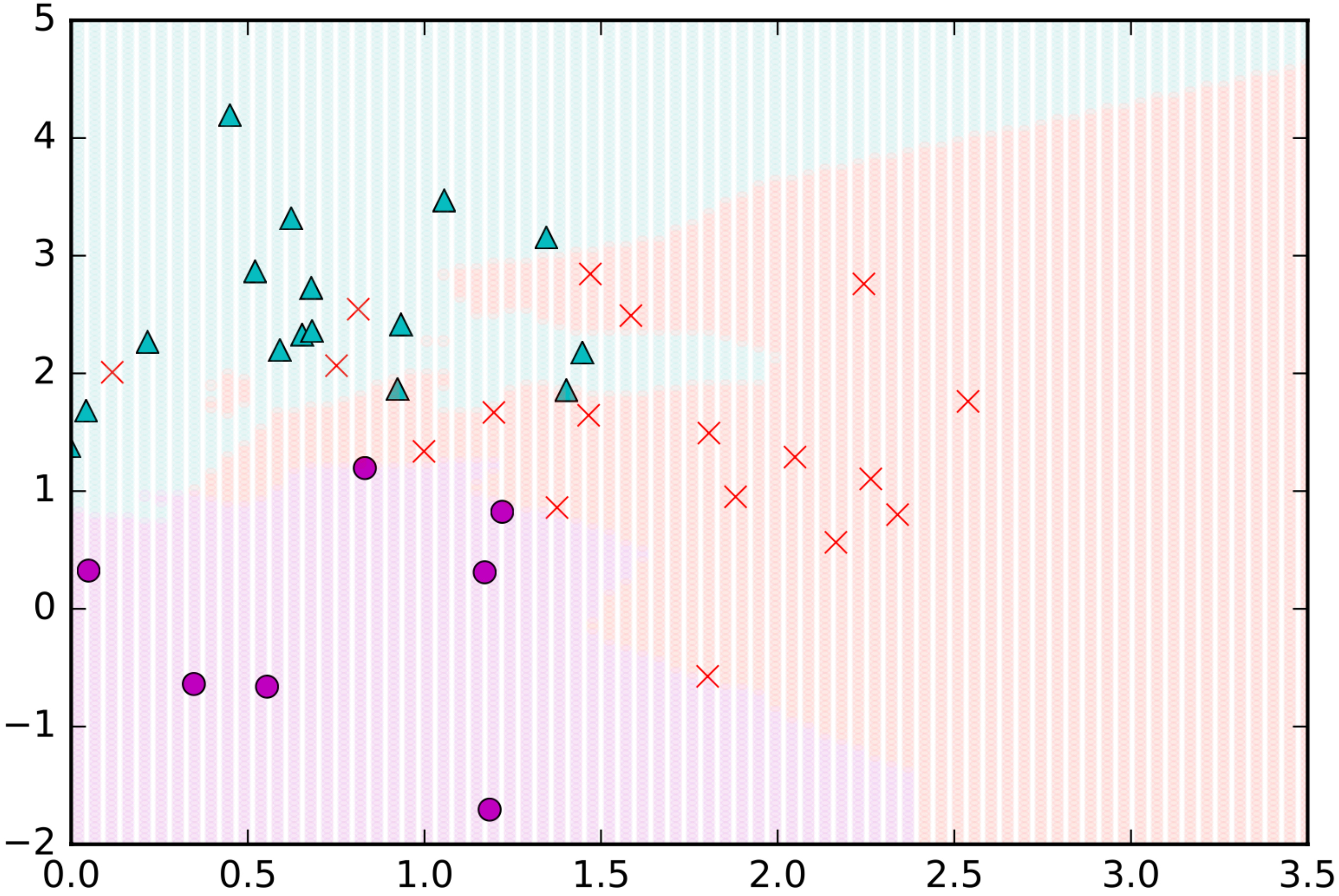
(bsxfun)

Pros? simple

Cons? slow, scale w/ the amount of training data.

k-nearest neighbors

What a solution looks like for k=3 neighbors:

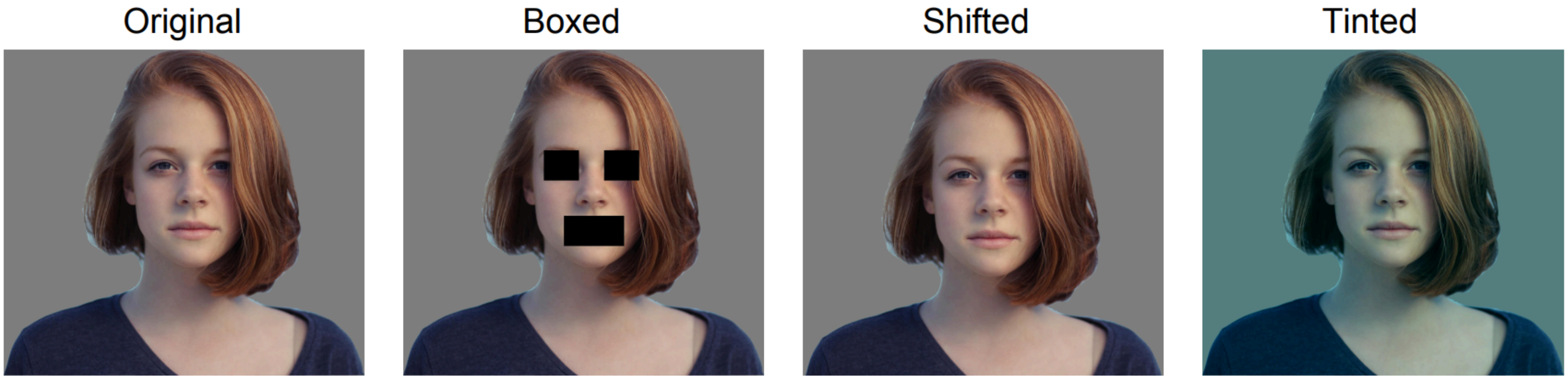


k-nearest neighbors

Why might k-nearest neighbors not be a good idea for image classification?

k-nearest neighbors

Why might k-nearest neighbors not be a good idea for image classification?



Original

Boxed

Shifted

Tinted

Original image is CC0 public domain

(all 3 images have same L2 distance to the one on the left)

Credit: CS231n, Stanford University

k-nearest neighbors

Why might k-nearest neighbors not be a good idea for image classification?

Curse of dimensionality:

- Images are very high-dimensional vectors, e.g., each CIFAR-10 image is a 3072 dimensional vector (and these are small images).
- Notions of "distance" become less intuitive in higher dimensions.
  - Distances in some dimensions matter more than others.
  - In higher-dimensional space, the volume increases exponentially.
  - This leaves a lot of empty space — and so the nearest neighbors may not be so near.

Classifiers based on linear classification

Perhaps a better way would be to develop a “score” for an image coming from each class, and then pick the class that achieves the highest score.

Linear classifiers are a major building block for neural networks. In particular, each layer of a neural network is composed of a linear classifier, followed by a nonlinear function.

The softmax classifier is the most common classifier at the end of a neural network.

Classifiers based on linear classification

Example 2: Consider a matrix, $\mathbf{W}$, defined as:

$x \in \mathbb{R}^n$

$$\begin{bmatrix} - \mathbf{w}_1^T - \\ \vdots \\ - \mathbf{w}_c^T - \end{bmatrix}$$

$c = 10$

$\hat{y}^{(i)} = W x^{(i)} + b$

Then, $\mathbf{W} \in \mathbb{R}^{c \times N}$. Let $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$, where $\mathbf{b}$ is a vector of bias terms. Then $\mathbf{y} \in \mathbb{R}^c$ is a vector of scores, with its ith element corresponding to the score of $\mathbf{x}$ being in class i. The chosen class corresponds to the index of the highest score in $\mathbf{y}$.

score of the image x belonging to class 1

$$y = \underbrace{\begin{bmatrix} w_1^T x + b_1 \\ w_2^T x + b_2 \\ \vdots \\ w_c^T x + b_c \end{bmatrix}}_{10 \times 1} = \underbrace{\overset{W}{\begin{bmatrix} — w_1^T — \\ — w_2^T — \\ \vdots \\ — w_c^T — \end{bmatrix}}}_{10 \times 3072} \underbrace{\begin{bmatrix} \\ x \\ \\ \end{bmatrix}}_{3072 \times 1} + \underbrace{\begin{bmatrix} \\ b \\ \\ \end{bmatrix}}_{10 \times 1}$$

score being in class c

CIFAR-10

$x \in \mathbb{R}^{3072}$

$y \in \mathbb{R}^{10}$

$c = 10$

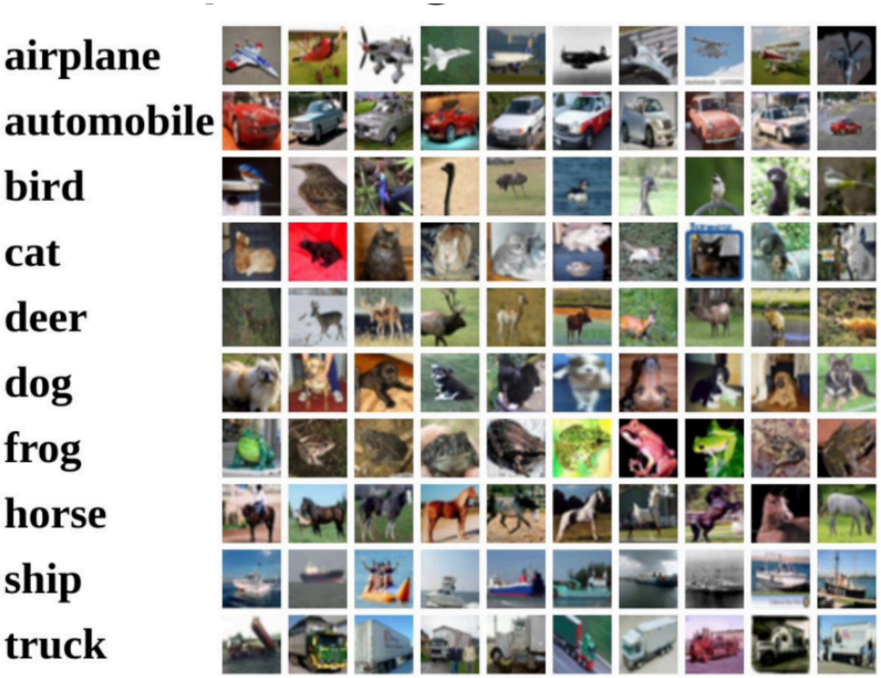
$b \in \mathbb{R}^{10}$

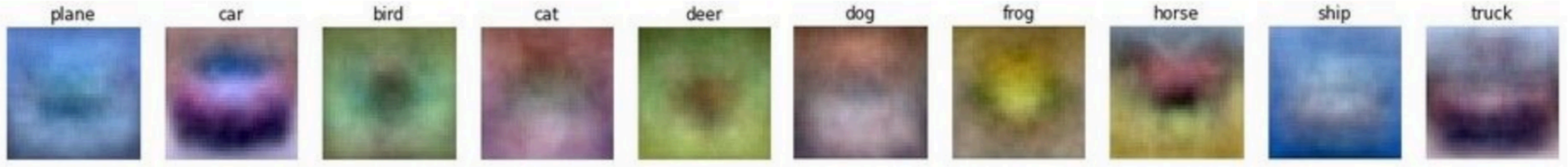
$$\hat{y}^{(i)} = \begin{bmatrix} 300 \\ 500 \\ -150 \\ \vdots \\ -23 \end{bmatrix}$$

The ith example belongs to class 2, automobiles.

Classifiers based on linear classification

Each row of **W** can be thought of as a template.

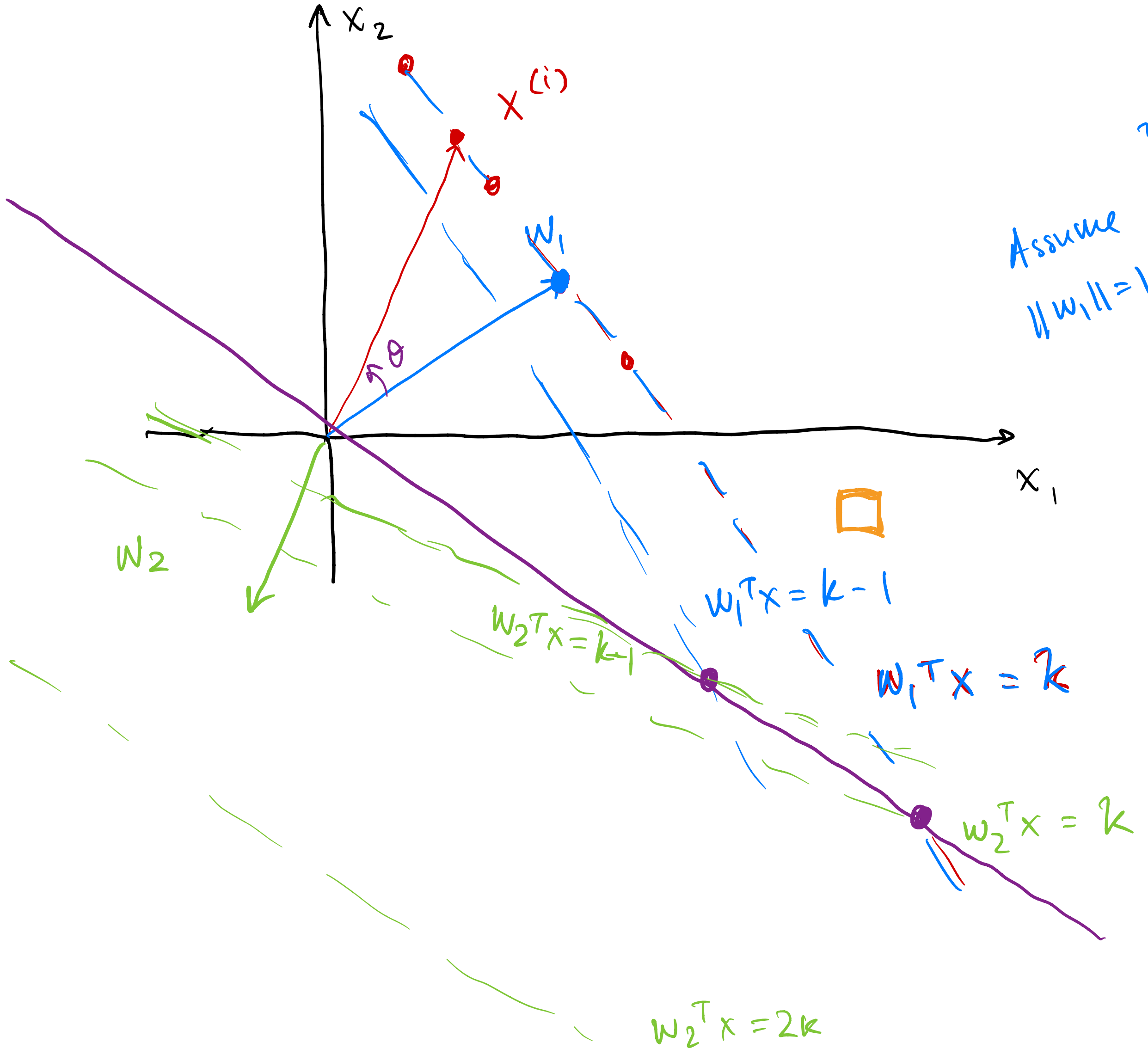




Classifiers based on linear classification

What is a linear classifier doing?

$x \in \mathbb{R}^2$ $\qquad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$



$$y_1 = w_1^T x$$

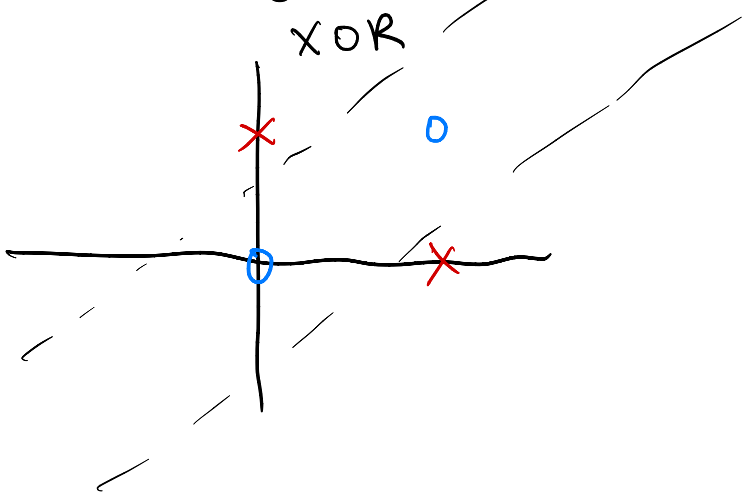
$$= \|w_1\| \cdot \|x\| \cos\theta$$

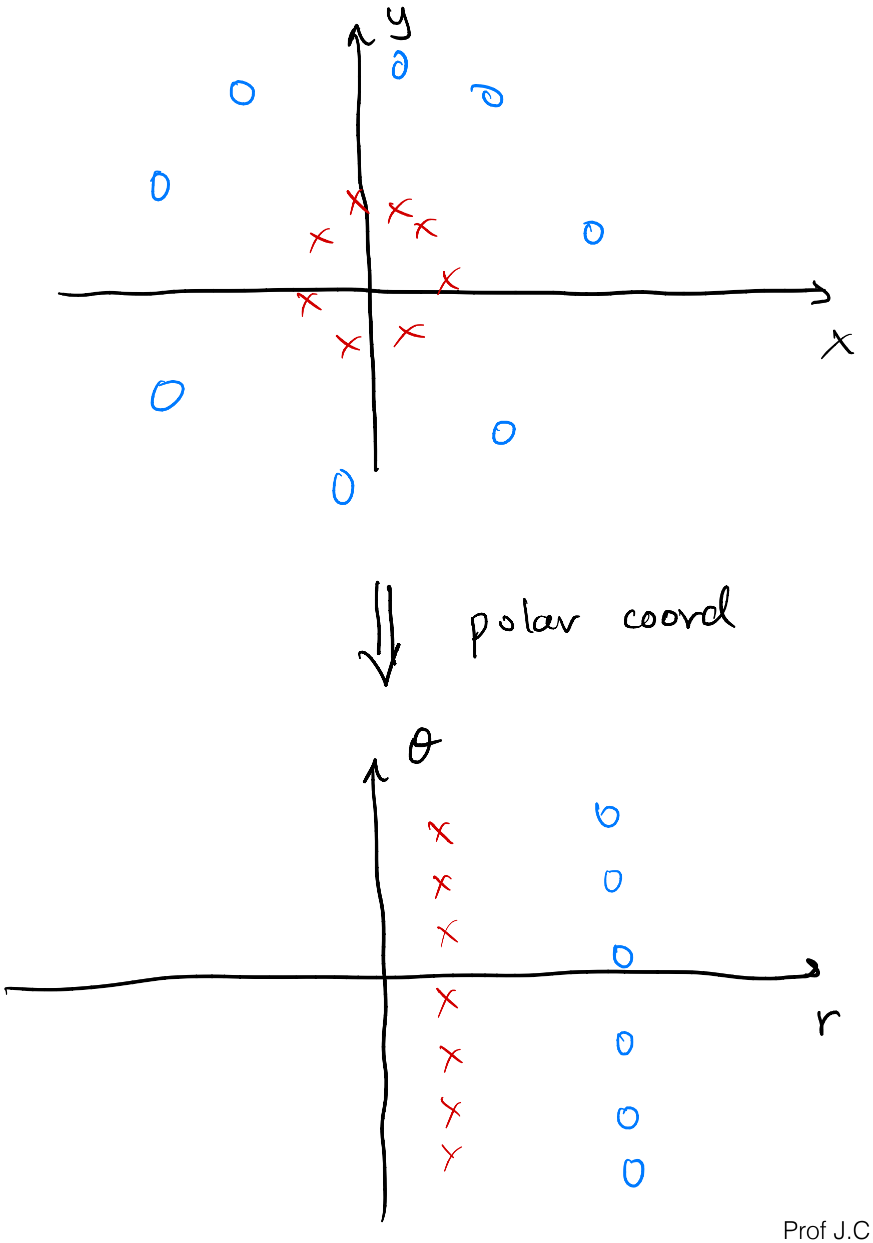
Assume $\|w_1\| = 1$

$$= \|x\| \cdot \cos\theta$$

Classifiers based on linear classification

Where might linear classifiers fail?





Classifiers based on linear classification

Next, how do we take the scores we receive (which are analog in value) and turn them into an appropriate **loss function** for us to optimize, so we can learn **W** and **b** appropriately?

Other types of optimization

We've talked about examples where we want to *minimize* a mean-square error or distance metric.

Another metric that we may want to ~~minimize~~ maximize is the *probability of having observed the data*. In this framework, the data is modeled to have some distribution with parameters. We choose the parameters to maximize the probability of having observed our training data.

Data: H T H H T T H T

Model: $x^{(i)} = \begin{cases} H \ (1), & \text{w.p. } \theta \\ T \ (0), & \text{w.p. } 1-\theta \end{cases}$

Likelihood:

Model 1: $\theta = 1$

Model 2: $\theta = 0.75$

Model 3: $\theta = 0.5$

Model 1: $1 \cdot 0 \cdot 1 \cdot 1 \cdot 0 \cdot 0 \cdot 1 \cdot 0 = 0$

Model 2: $(0.75)^4 (0.25)^4 = 0.00124$

Model 3: $(0.5)^4 (0.5)^4 = 0.0039$

Maximum-likelihood introduction

$$\mathcal{L} = \text{likelihood} = \theta^4 (1-\theta)^4$$

$$\frac{\partial \mathcal{L}}{\partial \theta} = 0 \qquad \frac{\partial \log \mathcal{L}}{\partial \theta} = 0$$

$$\Rightarrow \theta = \frac{1}{2}$$

Chain rule for probability

$$\Pr(A = a) = p_A(a) = p(a)$$

$$\Pr(B = b) = p_B(b) = p(b)$$

$$\Pr(A = a, B = b) = p_{A,B}(a,b) = p(a,b)$$

$$\Pr(A = a, B = b) = \Pr(A = a)\Pr(B = b \text{ given } A = a)$$

$$p(a,b) = p(a)\,p(b \mid a)$$

$$= p(b)\,p(a \mid b)$$

$$p(a,b,c) = p(c) \cdot p(a \mid c) \cdot p(b \mid a, c)$$

$$= p(a) \cdot p(b \mid a) \cdot p(c \mid a, b)$$

$$= p(a,c) \cdot p(b \mid a, c)$$

Chain rule for probability

$$p(b,c \mid d,e) = \frac{\boxed{?}}{\underbrace{p(d)\,p(e \mid d)}_{p(d,e)}}$$

$$\boxed{?} = p(b,c,d,e)$$

$$\underbrace{p(d,e)\cdot\boxed{?}}_{p(b,c,d,e)} = \frac{p(a,b,c,d,e)}{p(a \mid b,c,d,e)}$$

$$\boxed{?} = p(b,c \mid d,e)$$

Turn the scores into a probability

$$\frac{|a_i(x)|}{\sum_j |a_j(x)|}$$

CIFAR-10

A first thought is to turn the scores into probabilities.

$$\frac{2^{a_i(x)}}{\sum 2^{a_j(x)}}$$

Softmax function

There are several instances when the scores should be normalized. This occurs, for example, in instances where the scores should be interpreted as probabilities. In this scenario, it is appropriate to apply the *softmax* function to the scores.

The softmax function transforms the class score, $\text{softmax}_i(\mathbf{x})$, so that:

$$= \tilde{w}_i^T \tilde{x}$$

$$a_i(x) = y_i = w_i^T x + b_i$$

$$= \begin{bmatrix} \underset{1\times}{w_i^T} & \underset{1\,1}{b_i} \end{bmatrix} \underbrace{\begin{bmatrix} x \\ 1 \end{bmatrix}}_{3073}$$

$$\text{softmax}_i(\mathbf{x}) = \frac{e^{a_i(\mathbf{x})}}{\sum_{j=1}^{c} e^{a_j(\mathbf{x})}}$$

for $a_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{x} + b_i$ and c being the number of classes.

$$0 \le \text{softmax}_i(x) \le 1$$

Two classes:

$$a_1(x) = \tilde{w}_1^T x$$

$$\text{softmax}_1(x) = \frac{e^{\tilde{w}_1^T x}}{e^{\tilde{w}_1^T x} + e^{\tilde{w}_2^T x}}$$

$$a_2(x) = \tilde{w}_2^T x$$

$$\text{softmax}_2(x) = \frac{e^{\tilde{w}_2^T x}}{e^{\tilde{w}_1^T x} + e^{\tilde{w}_2^T x}}$$

Softmax classifier

$$\text{softmax}_i(\mathbf{x}) = \frac{e^{a_i(\mathbf{x})}}{\sum_{j=1}^{c} e^{a_j(\mathbf{x})}}$$

for $a_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{x} + b_i$ and c being the number of classes.

If we let $\theta = \{\mathbf{w}_j, b_j\}_{j=1,\ldots,c}$, then $\text{softmax}_i(\mathbf{x})$ can be interpreted as the probability that $\mathbf{x}$ belongs to class i. That is,

$$\Pr(y^{(j)} = i | \mathbf{x}^{(j)}, \theta) = \text{softmax}_i(\mathbf{x}^{(j)})$$

Example j

⇓

Prob that image $x^{(j)}$ belongs to class i

$p(x^{(1)}, y^{(1)}, x^{(2)}, y^{(2)} | \theta)$

$= p(x^{(1)}, y^{(1)} | \theta) \cdot p(x^{(2)}, y^{(2)} | \theta)$

Softmax classifier

$\Pr(\mathbb{X}^{(1)} = x^{(1)}, \mathbb{Y}^{(1)} = y^{(1)}, \mathbb{X}^{(2)} = x^{(2)}, \mathbb{Y}^{(2)} = y^{2} | \theta)$

Softmax classifier

$y^{(j)} = \{1, 2, \ldots, 10\}$

Although we know the softmax function, how do we specify the *objective* to be optimized with respect to θ?

One intuitive heuristic is that we should choose the parameters, θ, so as to maximize the likelihood of having seen the data. Assuming the samples, $(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(m)}, y^{(m)})$ are iid, this corresponds to maximizing:

image $\in \mathbb{R}^{3072}$ dog image cat

$\{W, b\}$

$$p(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}, y^{(1)}, \ldots, y^{(m)} | \theta) = \prod_{i=1}^{m} p(\mathbf{x}^{(i)}, y^{(i)} | \theta)$$

$$= \prod_{i=1}^{m} p(\mathbf{x}^{(i)} | \theta) \underbrace{p(y^{(i)} | \mathbf{x}^{(i)}, \theta)}_{\text{softmax}}$$

$$= \prod_{i=1}^{m} p(y^{(i)} | \theta) \cdot p(x^{(i)} | y^{(i)}, \theta)$$

Softmax classifier

$p(x^{(i)} | \underset{\{w, b\}}{\underset{\downarrow}{\theta}}) = p(x^{(i)})$

$$\arg\max_{\theta} \prod_{i=1}^{m} p(\mathbf{x}^{(i)}|\theta)p(y^{(i)}|\mathbf{x}^{(i)},\theta) \quad = \quad \arg\max_{\theta} \prod_{i=1}^{m} p(y^{(i)}|\mathbf{x}^{(i)},\theta)$$

$$\theta^* = \arg\max_{\theta} \sum_{i=1}^{m} \log \text{softmax}_{y^{(i)}}(x^{(i)})$$

$$= \arg\max_{\theta} \sum_{i=1}^{m} \log \left[\frac{e^{w_{y^{(i)}}^T x^{(i)} + b_{y^{(i)}}}}{\sum_{j=1}^{c} e^{w_j^T x^{(i)} + b_j}} \right]$$

$$= \arg\max_{\theta} \frac{1}{m} \sum_{i=1}^{m} \left[a_{y^{(i)}}(x^{(i)}) - \log \sum_{j=1}^{c} e^{a_j(x^{(i)})} \right]$$

$$= \arg\min_{\theta} \boxed{\frac{1}{m} \sum_{i=1}^{m} \left[\log \left(\sum_{j=1}^{c} e^{a_j(x^{(i)})} \right) - a_{y^{(i)}}(x^{(i)}) \right]}$$

$$\arg\max_{\theta} f(\theta) = \arg\min_{\theta} -f(\theta)$$

ith ex. $x^{(i)}, y^{(i)} = 2$

kth ex. $x^{(k)}, y^{(k)} = 4$

$$\boxed{a_j(x^{(i)}) = w_j^T x^{(i)} + b_j}$$

Softmax classifier

Now we have our softmax loss function.

$$\arg\min_{\theta} \sum_{i=1}^{m} \left(\log \sum_{j=1}^{c} e^{a_j(\mathbf{x})} - a_{y^{(i)}}(\mathbf{x}^{(i)}) \right)$$

Note, we haven't figured out yet how to get the optimal parameters.

W, b

(That'll be later.)

Softmax classifier

Simple sanity check:

Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2

Score check: $-a_{y^{(i)}}(\mathbf{x}^{(i)}) + \log \sum_{j=1}^{c} \exp(a_j(\mathbf{x}^{(i)}))$

Softmax classifier

Simple sanity check:

Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2

Cat: -2.1 + log(exp(2.1) + exp(3.4) + exp(-2.0)) = 1.54

Softmax classifier

Simple sanity check:

Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2
Loss:	1.54		

Car: -5.1 + log(exp(0.2) + exp(5.1) + exp(1.7)) = 0.04

Softmax classifier

Simple sanity check:

Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2
Loss:	1.54	0.04	

Bird: 1.2 + log(exp(2.3) + exp(3.1) + exp(-1.2)) = 4.68

Softmax classifier

Simple sanity check:

Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2
Loss:	1.54	0.04	4.68

$$-a_{y^{(i)}}(\mathbf{x}^{(i)}) + \log \sum_{j=1}^{c} \exp(a_j(\mathbf{x}^{(i)}))$$

Softmax classifier

A few additional notes on the softmax classifier:

$$\frac{e^{100}}{e^{100} + e^{20} + e^{-1}}$$

Softmax classifier: intuition

When optimizing likelihoods, we typically work with the "log likelihood." When applying the softmax, we interpret its output as the probability of a class.

$$\begin{aligned} \log \Pr(y = y^{(i)}|\mathbf{x}) &= \log \text{softmax}_i(\mathbf{x}) \\ &= a_{y^{(i)}}(\mathbf{x}) - \log \sum_{j=1}^{c} \exp(a_j(\mathbf{x})) \end{aligned}$$

3 classes:

100

20

-1

When maximizing this, the term $a_{y^{(i)}}(\mathbf{x})$ is made larger, and the term $\log \sum_j \exp(a_j(\mathbf{x}))$ is made smaller. The latter term can be approximated by $\max_j a_j(\mathbf{x})$. (Why?)

We consider two scenarios:

- If $a_{y^{(i)}}(\mathbf{x})$ produces the largest score, then the log likelihood is approximately 0.
- If $a_j(\mathbf{x})$ produces the largest score for $j \neq i$, then $a_{y^{(i)}}(\mathbf{x}) - a_j(\mathbf{x})$ is negative, and thus the log likelihood is negative.

Softmax classifier

A few additional notes on the softmax classifier:

Overflow of softmax

A potential problem when implementing a softmax classifier is overflow.

- If $a_i(\mathbf{x}) \gg 0$, then $e^{a_i(\mathbf{x})}$ may be very large, and numerically overflow and / or result to numerical impre~~s~~cision.
- Thus, it is standard practice to normalize the softmax function as follows:

$$
\begin{aligned}
\text{softmax}_i(\mathbf{x}) &= \frac{e^{a_i(\mathbf{x})}}{\sum_{j=1}^{c} e^{a_j(\mathbf{x})}} \\
&= \frac{k e^{a_i(\mathbf{x})}}{k \sum_{j=1}^{c} e^{a_j(\mathbf{x})}} \\
&= \frac{e^{a_i(\mathbf{x}) + \log k}}{\sum_{j=1}^{c} e^{a_j(\mathbf{x}) + \log k}}
\end{aligned}
$$

3 classes

$$\begin{bmatrix} 400 \\ 500 \\ 450 \end{bmatrix} \overset{-500}{\Rightarrow} \begin{bmatrix} -100 \\ 0 \\ -50 \end{bmatrix}$$

- A sensible choice of k is so that $\log k = -\max_i a_i(\mathbf{x})$, making the maximal argument of the exponent 0.

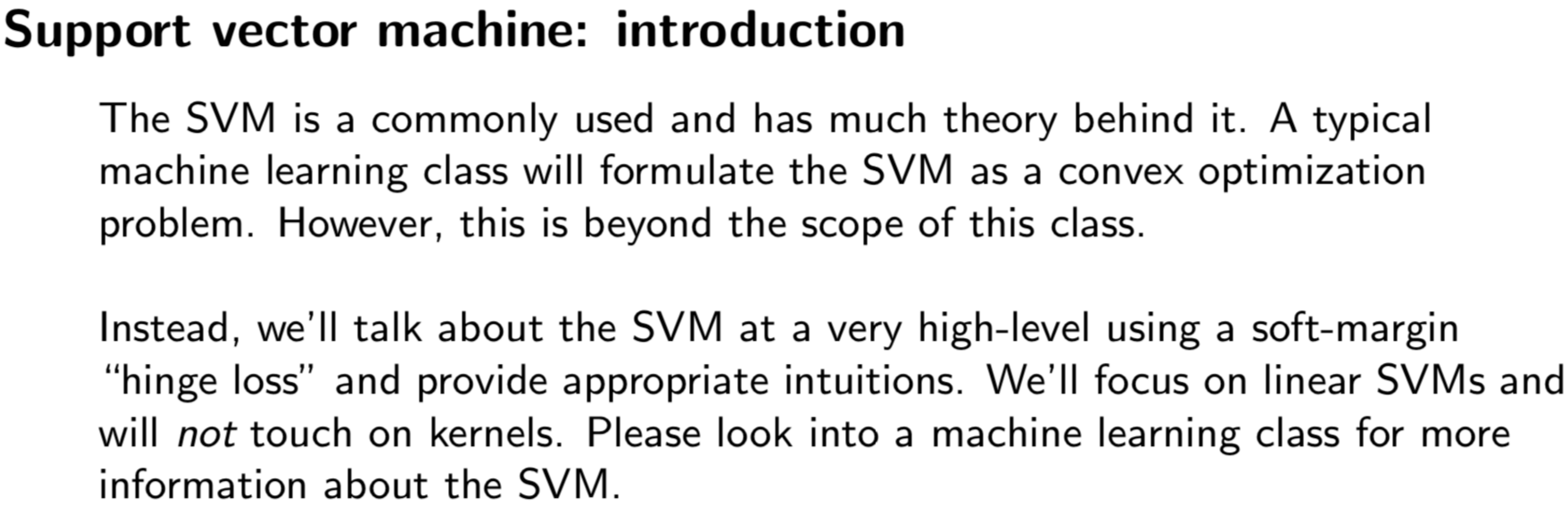
Support Vector Machine

In prior years, we also taught on the support vector machine (SVM) and the hinge loss. Since most modern neural networks today only use a softmax classifier, we have decided to remove this material, and you will not be tested on SVMs. We have kept these slides in as extra resources.

Support vector machine

NOT TESTED

Before getting to parameter fitting, we'll want to introduce one more classifier that is commonly used: the support vector machine.



Support vector machine: introduction

The SVM is a commonly used and has much theory behind it. A typical machine learning class will formulate the SVM as a convex optimization problem. However, this is beyond the scope of this class.

Instead, we'll talk about the SVM at a very high-level using a soft-margin "hinge loss" and provide appropriate intuitions. We'll focus on linear SVMs and will *not* touch on kernels. Please look into a machine learning class for more information about the SVM.

NOT TESTED

Support vector machine: introduction

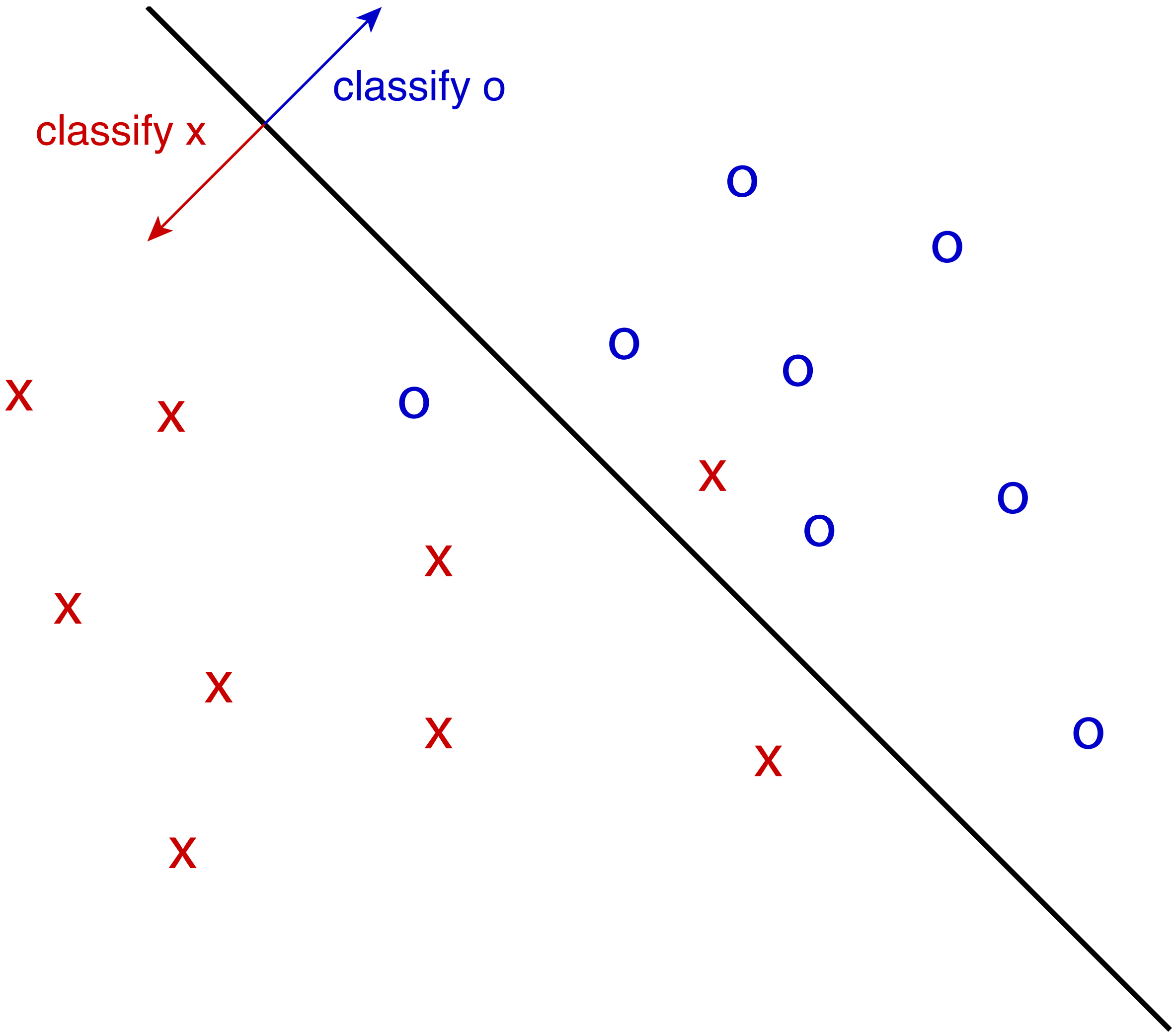
Another common decision boundary classifier is the support vector machine (SVM).

Informally, the SVM finds a boundary that maximizes the *margin*, or intuitively the "gap" between the boundary and the data points. The fundamental idea here is that if a point is further away from the decision boundary, there ought to be greater *confidence* in classifying that point.

Support vector machine

NOT TESTED

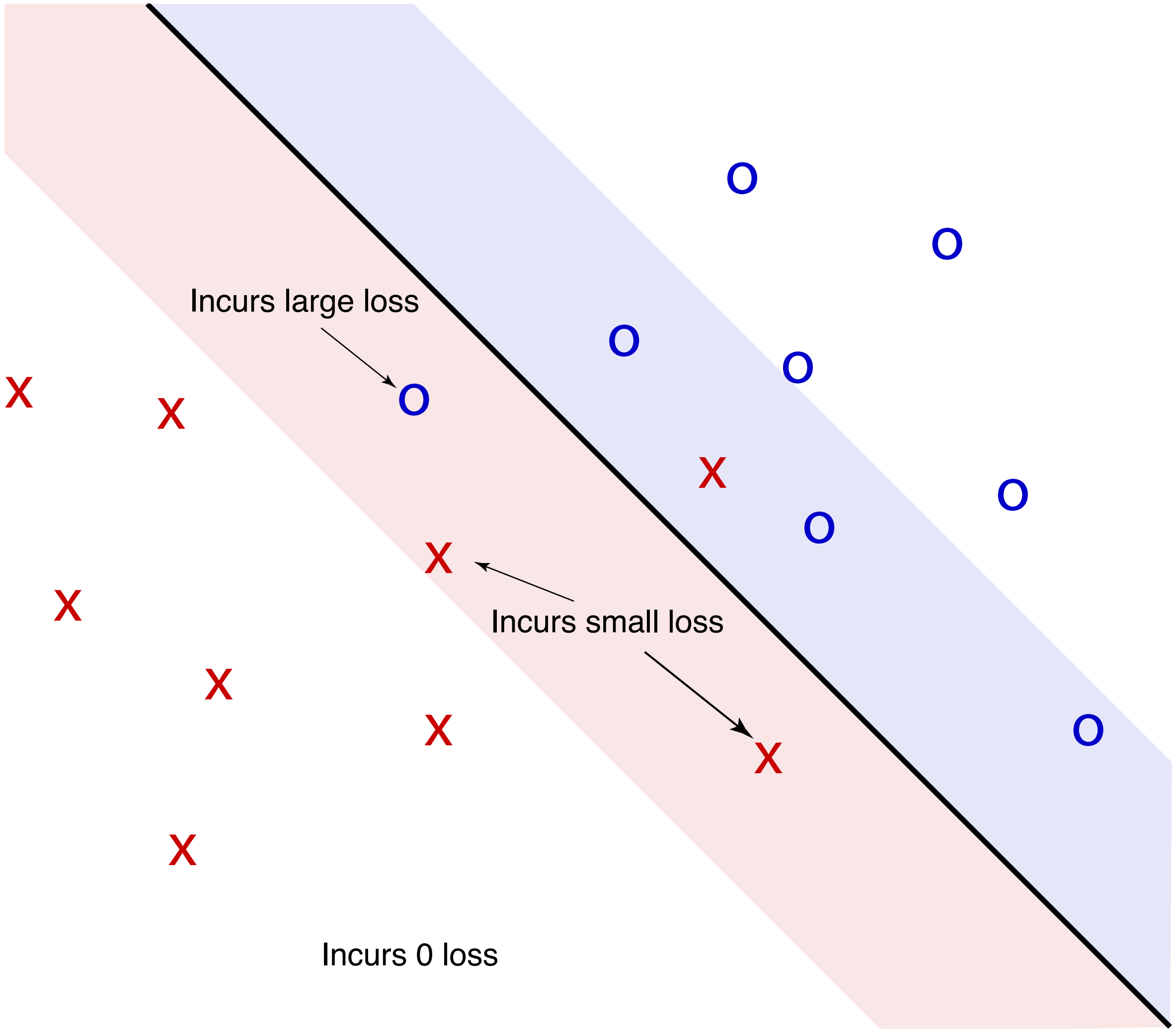
This is the picture we should have in mind:



Support vector machine

NOT TESTED

This is the picture we should have in mind:



Support vector machine

NOT TESTED

The hinge loss function

The hinge loss is standardly defined for a binary output $y^{(i)} \in \{-1, 1\}$. If $y^{(i)} = 1$, then we would like $\mathbf{w}^T\mathbf{x}^{(i)} + b$ to be large and positive. If $y^{(i)} = -1$, then we would like $\mathbf{w}^T\mathbf{x}^{(i)} + b$ to be large and negative. The larger $a_i(\mathbf{x}^{(j)})$ is in the right direction, the larger the margin.

In this scenario, the hinge loss is for $x^{(i)}$ being in class $y^{(i)}$ is:

$$\text{hinge}_{y^{(i)}}(\mathbf{x}^{(i)}) = \max(0, 1 - y^{(j)}(\mathbf{w}^T\mathbf{x}^{(i)} + b))$$

Support vector machine

NOT TESTED

$$\mathrm{hinge}_{y^{(i)}}(\mathbf{x}^{(i)}) = \max(0, 1 - y^{(j)}(\mathbf{w}^T\mathbf{x}^{(i)} + b))$$

This is a loss, and hence something we wish to minimize. There are a few things to notice about the form of this function.

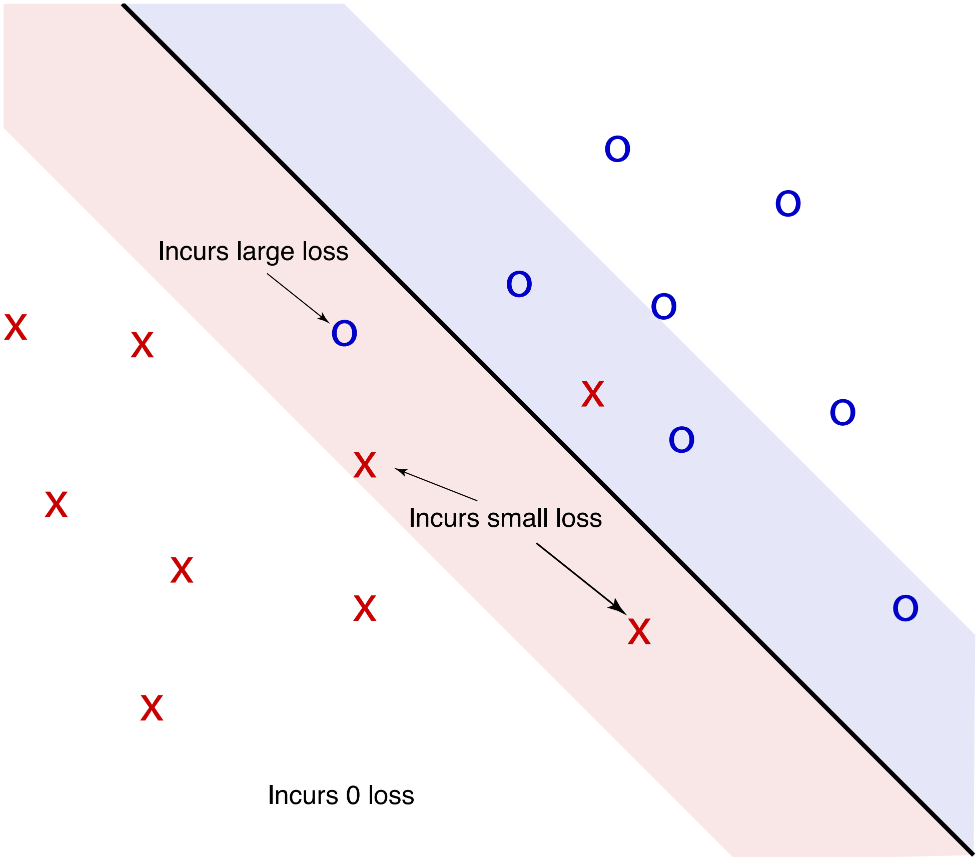
- If $\mathbf{w}^T\mathbf{x}^{(i)} + b$ and $y^{(i)}$ have the same sign, indicating a correct classification, then $0 \leq \mathrm{hinge}_{y^{(i)}}(\mathbf{x}^{(i)}) \leq 1$.
    - The error will be zero if $\mathbf{w}^T\mathbf{x}^{(i)} + b$ is large, corresponding to a large margin.
    - The error will be nonzero if $\mathbf{w}^T\mathbf{x}^{(i)} + b$ is small, corresponding to a small margin.
- If $\mathbf{w}^T\mathbf{x}^{(i)} + b$ and $y^{(i)}$ have opposite signs, then the hinge loss is non-negative, i.e., $\mathrm{hinge}_{y^{(i)}}(\mathbf{x}^{(i)}) = 1 + |\mathbf{w}^T\mathbf{x}^{(i)} + b|$.

Support vector machine

NOT TESTED

Hinge loss intuition

The intuition of the prior slide is that the hinge loss is greatest for misclassifications, and the greater the error in misclassification, the worse the loss. For correct classifications, the loss will be zero only if there is a large enough margin.



NOT TESTED

Hinge loss extension

An extension of the hinge loss to multiple potential outputs is the following loss:

$$\text{hinge}_{y^{(i)}}(\mathbf{x}^{(i)}) = \sum_{j \neq y^{(i)}} \max(0, 1 + a_j(\mathbf{x}^{(i)}) - a_{y^{(i)}}(\mathbf{x}^{(i)}))$$

for $a_j(\mathbf{x}^{(i)}) = \mathbf{w}_j^T \mathbf{x}^{(i)}$. Some intuitions, for the scenario that there are c classes:

- When the correct class achieves the highest score, $a_{y^{(i)}}(\mathbf{x}^{(i)}) \geq a_j(\mathbf{x}^{(i)})$ for all $j \neq y^{(j)}$, then $a_j(\mathbf{x}^{(i)}) - a_{y_{(i)}}(\mathbf{x}^{(i)}) \leq 0$ and

$$0 \leq \text{hinge}_{y^{(i)}}(\mathbf{x}^{(i)}) \leq c - 1$$

- When an incorrect class, class i, achieves the highest score, then $a_j(\mathbf{x}^{(i)}) - a_{y^{(i)}}(\mathbf{x}^{(i)}) \geq 0$ and has the potential to be large.
- In both scenarios, it is still desirable to make the correct margins larger and the incorrect margins smaller.

Support vector machine

NOT TESTED

The SVM cost function

If we let $\theta = \{\mathbf{w}_j\}_{j=1,\ldots,c}$, where there are c classes, we can now formulate the SVM optimization function. In particular, we want to minimize the hinge loss across all training examples. Then, to optimize θ for a linear kernel and hinge loss, we solve the following minimization problem:

$$\arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \text{hinge}_{y^{(i)}}(\mathbf{x}^{(i)})$$

which, for the sake of completeness, can be writen as:

$$\arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \sum_{j \neq y^{(i)}} \max(0, 1 + a_j(\mathbf{x}^{(i)}) - a_{y^{(i)}}(\mathbf{x}^{(i)}))$$

Support vector machine

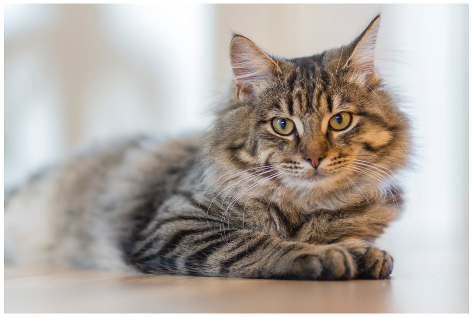
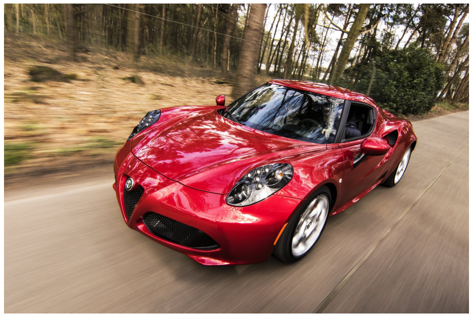
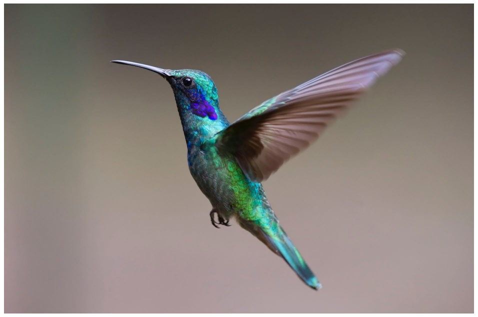
NOT TESTED

Is there a closed-form solution?

Support vector machine

NOT TESTED

Simple sanity check:

			
Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2

Score check: $$\sum_{i \neq y^{(j)}} \max(0, 1 + a_i(\mathbf{x}^{(j)}) - a_{y^{(j)}}(\mathbf{x}^{(j)}))$$

Support vector machine

NOT TESTED

Simple sanity check:

Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2

Cat: $\max(0, 1 - 2.1 + 3.4) + \max(0, 1 - 2.1 - 2.0) = \max(0, 2.3) + \max(0, -3.1) = 2.3$

Support vector machine

NOT TESTED

Simple sanity check:

Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2
Loss:	2.3		

Car: max(0, 1 - 5.1 + 0.2) + max(0, 1 - 5.1 +1.7) = max(0, -3.9) + max(0, -2.4) = 0

Support vector machine

NOT TESTED

Simple sanity check:

Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2
Loss:	0.3	0	

Bird: max(0, 1 + 1.2 + 2.3) + max(0, 1 + 1.2 + 3.1) = max(0, 4.5) + max(0, 5.3) = 9.8

Support vector machine

NOT TESTED

Simple sanity check:

Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2
Loss:	0.3	0	9.8

$$\sum_{i \neq y^{(j)}} \max(0, 1 + a_i(\mathbf{x}^{(j)}) - a_{y^{(j)}}(\mathbf{x}^{(j)}))$$

Softmax loss function

Softmax:
$$\arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \left(\log \sum_{j=1}^{c} e^{a_j(\mathbf{x})} - a_{y^{(i)}}(\mathbf{x}^{(i)}) \right)$$

Parameters?
$$\theta = \{ W, b \}$$

Softmax loss function

Softmax:

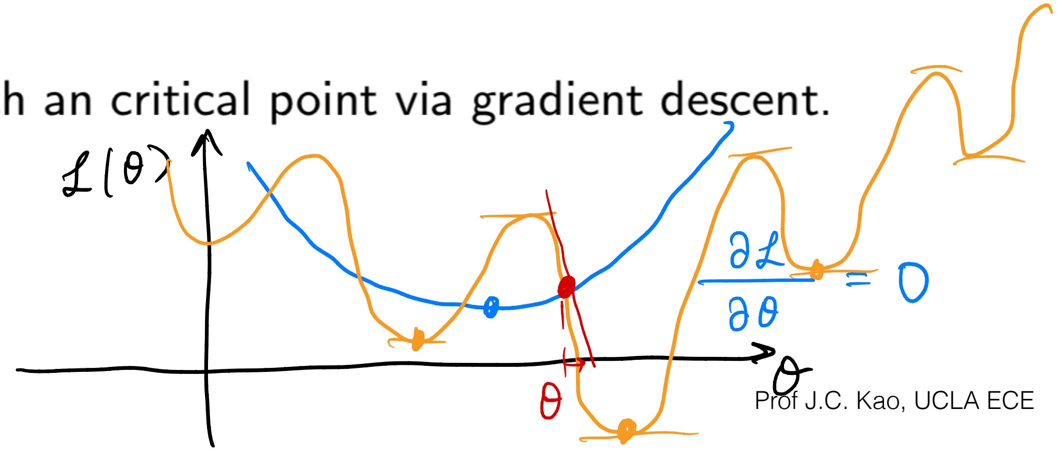
$$\arg\min_{\theta} \frac{1}{m}\sum_{i=1}^{m}\left(\log\sum_{j=1}^{c} e^{a_j(\mathbf{x})} - a_{y^{(i)}}(\mathbf{x}^{(i)})\right)$$

Parameters: $\mathbf{W} \in \mathbb{R}^{c\times n}, \mathbf{b} \in \mathbb{R}^{c}$

Big question: how do we find these parameters?

Finding the optimal weights through gradient descent

- Our goal in machine learning is to optimize an objective function, $f(x)$. (Without loss of generality, we'll consider minimizing $f(x)$. This is equivalent to maximizing $-f(x)$.)
- From basic calculus, we recall that the derivative of a function, $\frac{df(x)}{dx}$ tells us the slope of $f(x)$ at point x.
  - For small enough ϵ, $f(x+\epsilon) \approx f(x) + \epsilon f'(x)$.
  - This tells us how to reduce (or increase) $f(\cdot)$ for small enough steps.
  - Recall that when $f'(x) = 0$, we are at a stationary point or critical point. This may be a local or global minimum, a local or global maximum, or a saddle point of the function.
- In this class we will consider cases where we would like to maximize f w.r.t. vectors and matrices, e.g., $f(\mathbf{x})$ and $f(\mathbf{X})$.
- Further, often $f(\cdot)$ contains a nonlinearity or non-differentiable function. In these cases, we can't simply set $f'(\cdot) = 0$, because this does not admit a closed-form solution.
- However, we can iteratively approach an critical point via gradient descent.



Finding the optimal weights through gradient descent

To do so, we use the technique of gradient descent.