Classification via gradient descent Logistic regression

James G. Shanahan [1,2]

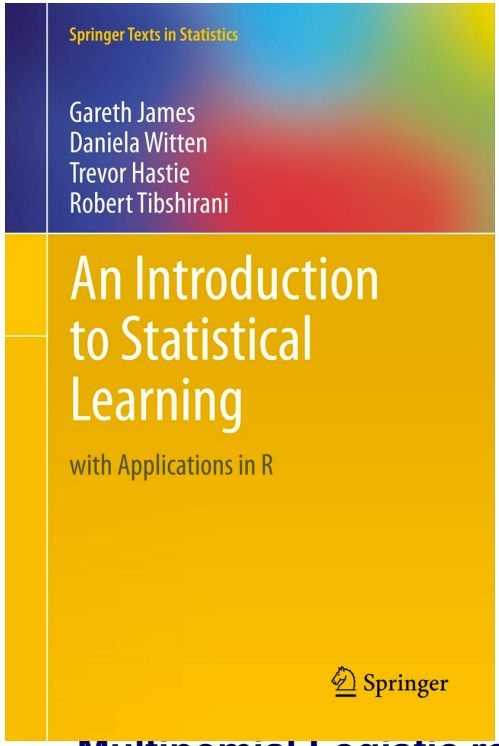
[1]Church and Duncan Group,

[2]School of Informatics, Computing and Engineering, Indiana University

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

Reading material

- **See Free PDF**
  - Please read chapter 4and 6 for a very good intro on linear regression and regularization
  - An Introduction to Statistical Learning: with Applications in R (Springer Texts in Statistics) 1st ed. 2013, Corr. 6th printing 2016 Edition by Gareth James (Author), Daniela Witten (Author), Trevor Hastie (Author), Robert Tibshirani (Author)
  - http://www-bcf.usc.edu/~gareth/ISL/ISLR%20Sixth%20Printing.pdf
  - http://cs231n.github.io/linear-classify/



- **Multinomial Logistic regression**
  - http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/

Outline

- Introduction
- Binomial logistic regression
- Multinomial Logistic regression
- Linear Classifier via separating hyperplane
- Multinomial Logistic regression Classifier
- Linear classifier demo
- Learning Softmax Classifiers via optimization
  - Implementation (SoftMax Classifier)
- Regressions in matrix terms and in graph terms
- Summary

Outline

- Introduction
- Binomial logistic regression
- Multinomial Logistic regression
- Linear Classifier via separating hyperplane
- Multinomial Logistic regression Classifier
- Linear classifier demo
- Learning Softmax Classifiers via optimization
  - Implementation (SoftMax Classifier)
- Regressions in matrix terms and in graph terms
- Summary

Build linear Classifiers

- Qualitative variables take values in an unordered set $\mathcal{C}$, such as:
  $\texttt{eye color} \in \{\texttt{brown}, \texttt{blue}, \texttt{green}\}$
  $\texttt{email} \in \{\texttt{spam}, \texttt{ham}\}$.
- Given a feature vector X and a qualitative response Y taking values in the set $\mathcal{C}$, the classification task is to build a function $C(X)$ that takes as input the feature vector X and predicts its value for Y; i.e. $C(X) \in \mathcal{C}$.
- Often we are more interested in estimating the *probabilities* that X belongs to each category in $\mathcal{C}$.

For example the SPAMiness of an email

Basic operation on vectors in $\mathbb{E}^n$

1. Multiplication by a scalar

Consider a vector $\vec{a} = (a_1, a_2, \ldots, a_n)$ and a scalar c

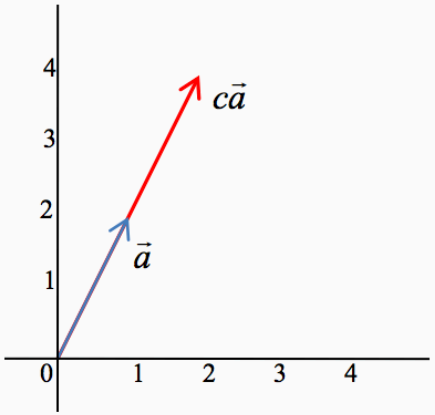
Define: $c\vec{a} = (ca_1, ca_2, \ldots, ca_n)$

When you multiply a vector by a scalar, you "stretch" it in the same or opposite direction depending on whether the scalar is positive or negative.

$\vec{a} = (1,2)$

$c = 2$

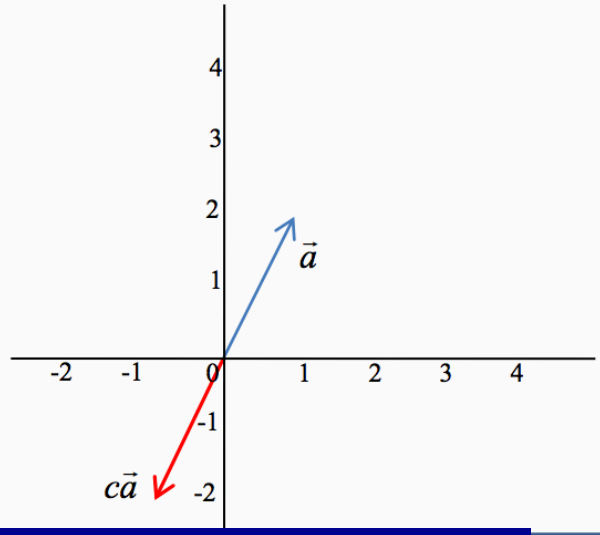
$\vec{c}a = (2,4)$



$\vec{a} = (1,2)$

$c = -1$

$\vec{c}a = (-1,-2)$



Adopted from http://www.med.nyu.edu/chibi/sites/default/files/chibi/Final.pdf

Basic operation on vectors in $\mathbb{E}^n$

2. Addition

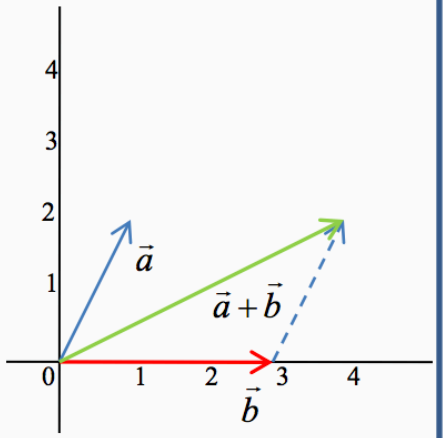
Consider vectors $\vec{a} = (a_1, a_2, \ldots, a_n)$ and $\vec{b} = (b_1, b_2, \ldots, b_n)$

Define: $\vec{a} + \vec{b} = (a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n)$

$\vec{a} = (1,2)$

$\vec{b} = (3,0)$

$\vec{a} + \vec{b} = (4,2)$



Recall addition of forces in classical mechanics.

Basic operation on vectors in $\mathbb{E}^n$

3. Subtraction

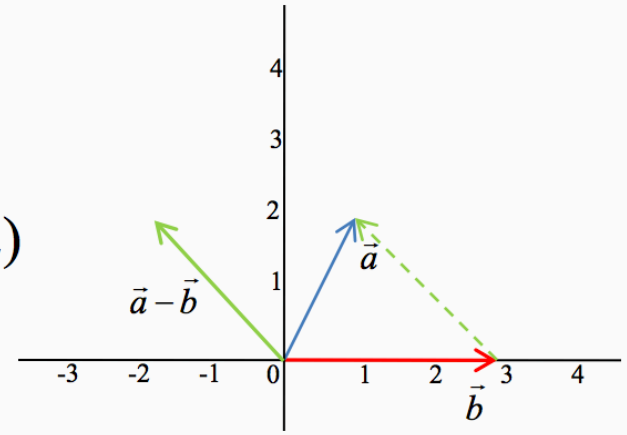
Consider vectors $\vec{a} = (a_1, a_2, \ldots, a_n)$ and $\vec{b} = (b_1, b_2, \ldots, b_n)$

Define: $\vec{a} - \vec{b} = (a_1 - b_1, a_2 - b_2, \ldots, a_n - b_n)$

$\vec{a} = (1,2)$

$\vec{b} = (3,0)$

$\vec{a} - \vec{b} = (-2,2)$



What vector do we need to add to $\vec{b}$ to get $\vec{a}$? I.e., similar to subtraction of real numbers.

Basic operation on vectors in $\mathbb{E}^n$

3. Subtraction

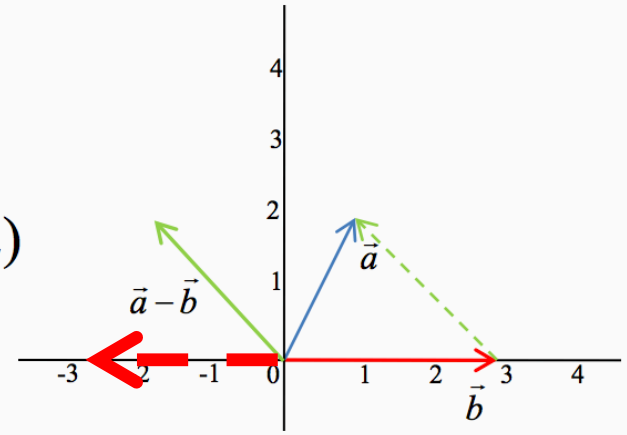
Consider vectors $\vec{a} = (a_1, a_2, \ldots, a_n)$ and $\vec{b} = (b_1, b_2, \ldots, b_n)$

Define: $\vec{a} - \vec{b} = (a_1 - b_1, a_2 - b_2, \ldots, a_n - b_n)$

$\vec{a} = (1,2)$

$\vec{b} = (3,0)$

$\vec{a} - \vec{b} = (-2,2)$



What vector do we need to add to $\vec{b}$ to get $\vec{a}$? I.e., similar to subtraction of real numbers.

Basic operation on vectors in $\mathbb{E}^n$

3. Subtraction

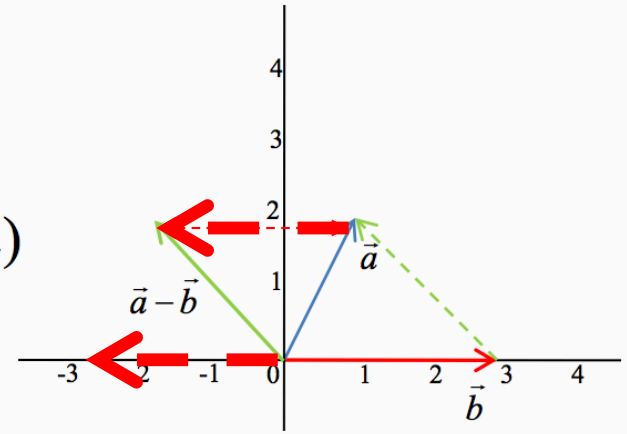
Consider vectors $\vec{a} = (a_1, a_2, \ldots, a_n)$ and $\vec{b} = (b_1, b_2, \ldots, b_n)$

Define: $\vec{a} - \vec{b} = (a_1 - b_1, a_2 - b_2, \ldots, a_n - b_n)$

$\vec{a} = (1,2)$

$\vec{b} = (3,0)$

$\vec{a} - \vec{b} = (-2,2)$



What vector do we need to add to $\vec{b}$ to get $\vec{a}$? I.e., similar to subtraction of real numbers.

Basic operation on vectors in $\mathbb{E}^n$

4. Euclidian length or L2-norm

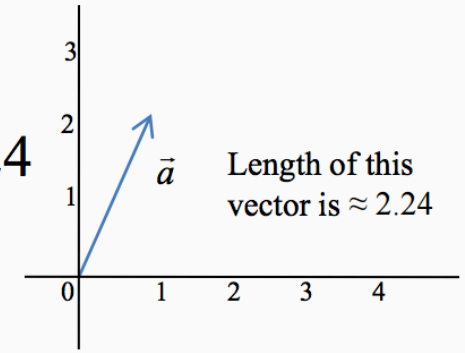
Consider a vector $\vec{a} = (a_1, a_2, \ldots, a_n)$

Define the L2-norm: $\|\vec{a}\|_2 = \sqrt{a_1^2 + a_2^2 + \ldots + a_n^2}$

We often denote the L2-norm without subscript, i.e. $\|\vec{a}\|$

$\vec{a} = (1,2)$

$\|\vec{a}\|_2 = \sqrt{5} \approx 2.24$



L2-norm is a typical way to measure length of a vector; other methods to measure length also exist.

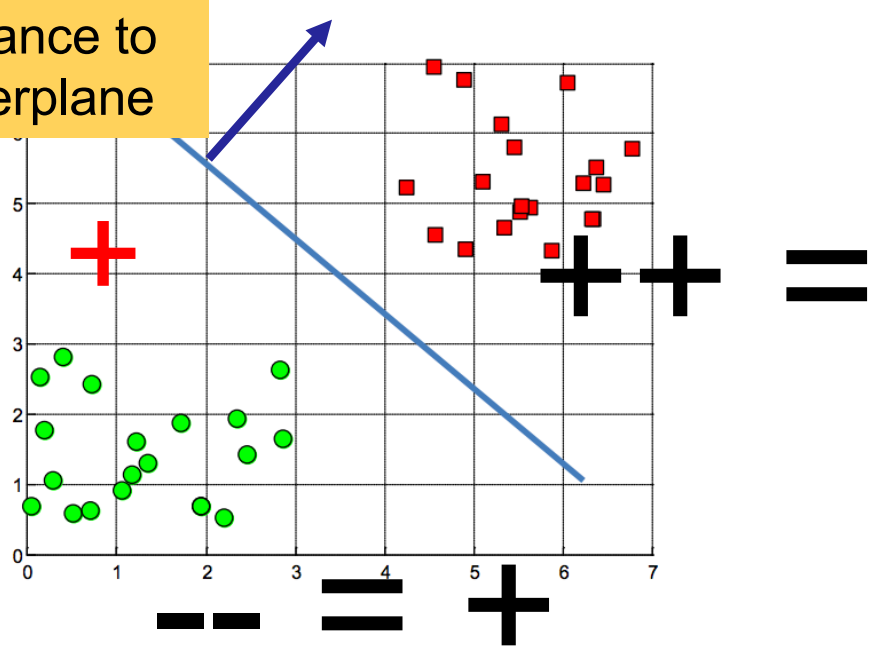
Adopted from http://www.med.nyu.edu/chibi/sites/default/files/chibi/Final.pdf

Hyperplanes as decision surfaces

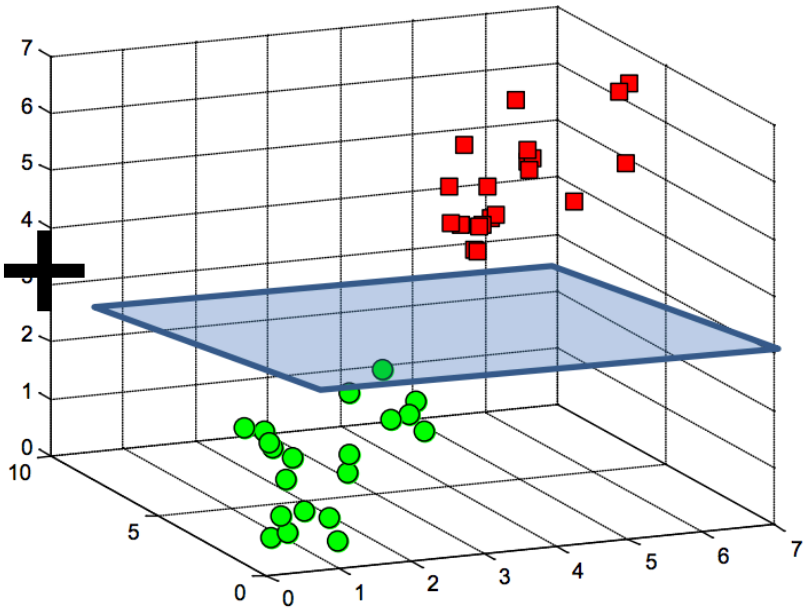
- A hyperplane is a linear decision surface that splits the space into two parts;
- It is obvious that a hyperplane is a binary classifier.

A hyperplane in $\mathbb{E}^2$ is a line

Distance to hyperplane



A hyperplane in $\mathbb{E}^3$ is a plane



A hyperplane in $\mathbb{E}^n$ is an n-1 dimensional subspace

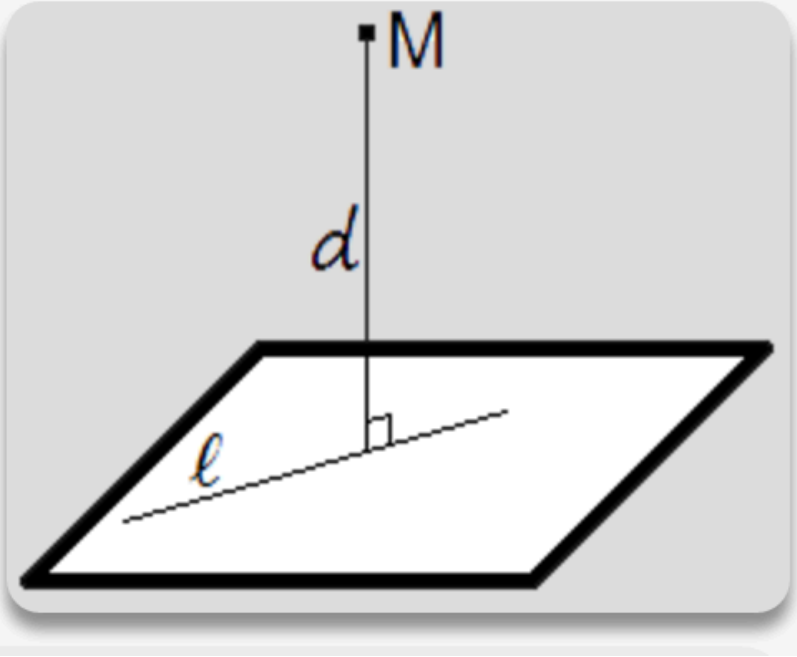
Perpendicular distance from a point to a plane

The distance from a point to a plane — is equal to length of the perpendicular lowered from a point on a plane.

If $Ax + By + Cz + D = 0$ is a plane equation, then distance from point $M(M_x, M_y, M_z)$ to plane can be found using the following formula

$$d = \frac{A \cdot M_x + B \cdot M_y + C \cdot M_z + D}{\sqrt{A^2 + B^2 + C^2}}$$

Vector that is normal



Quiz

- **How far from the origin is the line 3x + 4y = 10?**
  - (a) 1.5 units
  - (b) 2 units
  - (c) 2.5 units
  - (d) 2.75 units
  - (e) None of the above

Shortcut: Pdist from (0,0) to 3x + 4y = 10

$$\frac{(3(0) + 4(0) - 10)}{\text{SQRT}((3 * 3) + (4*4))} = \frac{-10}{5} = -2$$

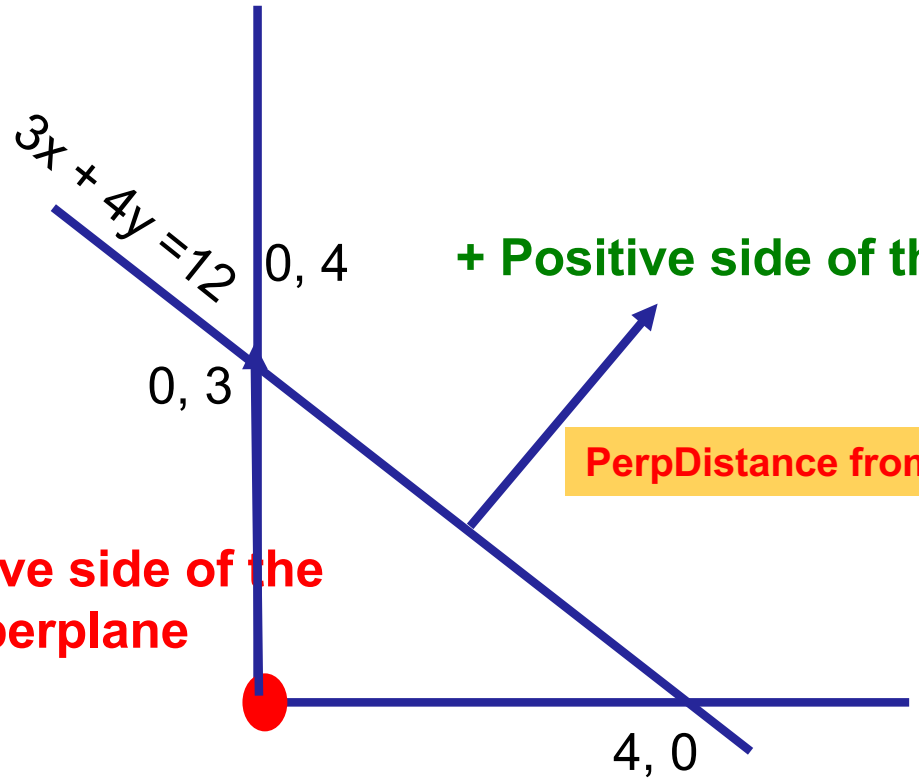
The distance from a point to a plane — is equal to length of the perpendicular lowered from a point on a plane.

If $Ax + By + Cz + D = 0$ is a plane equation, then distance from point $M(M_x, M_y, M_z)$ to plane can be found using the following formula

$$d = \frac{A \cdot M_x + B \cdot M_y + C \cdot M_z + D}{\sqrt{A^2 + B^2 + C^2}}$$

HyperPlane 3x + 4y =12

QUESTION: What far is (0,4) from the hyperplane?



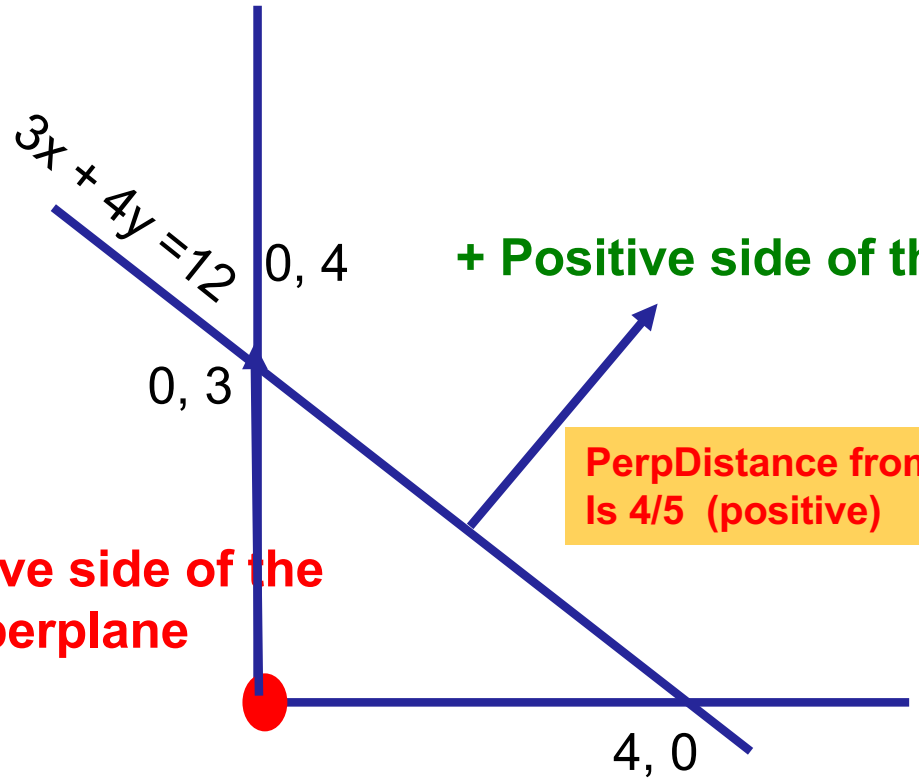
+ Positive side of the hyperplane

PerpDistance from point (0,4) to the hyperplane 3x + 4y =12

- Negative side of the hyperplane

HyperPlane 3x + 4y =12

QUESTION: What side of the hyperplane does the origin lie?



+ Positive side of the hyperplane

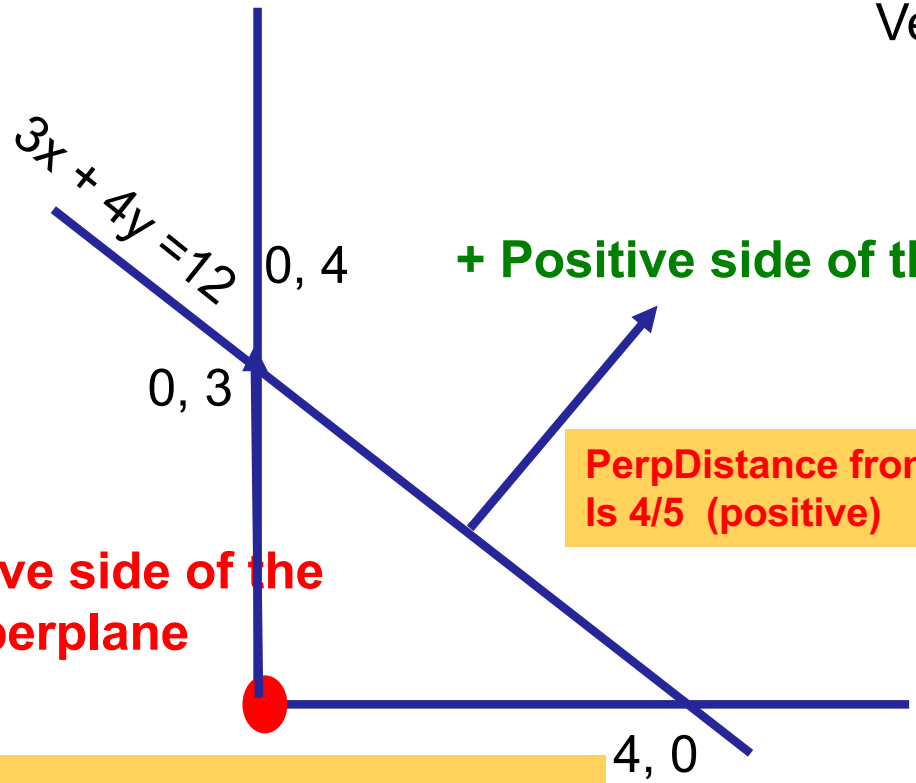
PerpDistance from point (0,4) to the hyperplane 3x + 4y =12 **Is 4/5 (positive)**

- Negative side of the hyperplane

HyperPlane 3x + 4y =12

QUESTION: What side of the hyperplane does the origin lie?

Shortcut: 3(0) + 4(0) -10 / SQTR((3 * 3) (4*4)) = -10/5=-2



PerpDistance from point (0,4) to the hyperplane 3x + 4y =12 **Is 4/5 (positive)**

The distance from a point to a plane — is equal to length of the perpendicular lowered from a point on a plane.

If $Ax + By + Cz + D = 0$ is a plane equation, then distance from point $M(M_x, M_y, M_z)$ to plane can be found using the following formula

$$d = \frac{|A \cdot M_x + B \cdot M_y + C \cdot M_z + D|}{\sqrt{A^2 + B^2 + C^2}}$$

Shortcut:

$$\frac{(3(0) + 4(0) - 12)}{\text{SQTR}((3 * 3) + (4*4))} = -12/5 = -2.4$$

Distance is signed

- **+ distance means the point is the same side where the normal is pointing**

Note that the distance formula looks like inserting P_2 into the plane equation, then dividing by the length of the normal vector. For example, the distance from a point (-1, -2, -3) to a plane x + 2y + 2z - 6 = 0 is;

$$D = \frac{1x + 2y + 2z - 6}{\sqrt{1^2 + 2^2 + 2^2}}, \qquad P_2 = (-1, -2, -3)$$

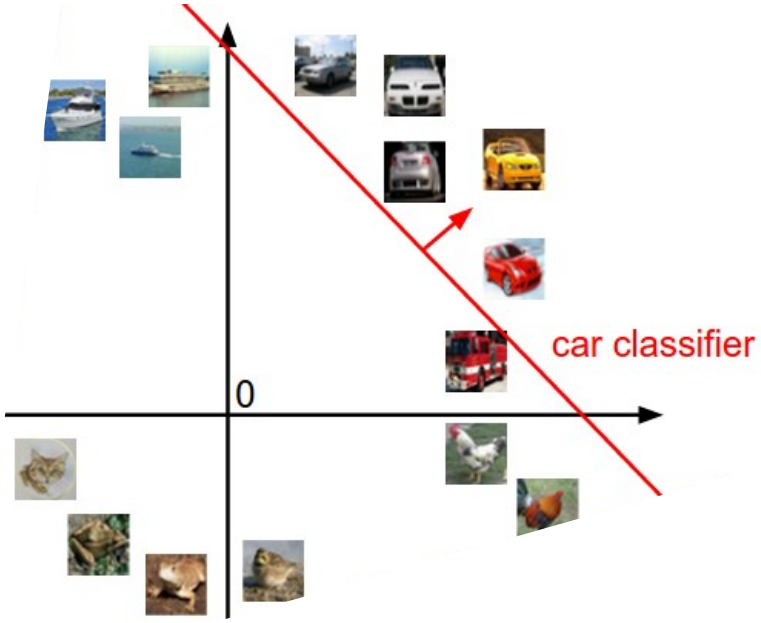
$$= \frac{1(-1) + 2(-2) + 2(-3) - 6}{\sqrt{1^2 + 2^2 + 2^2}}$$

$$= \frac{-1 - 4 - 6 - 6}{3} = \frac{-17}{3}$$

Notice this distance is signed; can be negative value. It is useful to determine the direction of the point. For example, if the distance is positive, the point is in the same side where the normal is pointing to. And, a negative distance means the point is in opposite side.

Interpreting a Linear Classifier

Learn θ which is W, b the parameters of a separating hyperplane



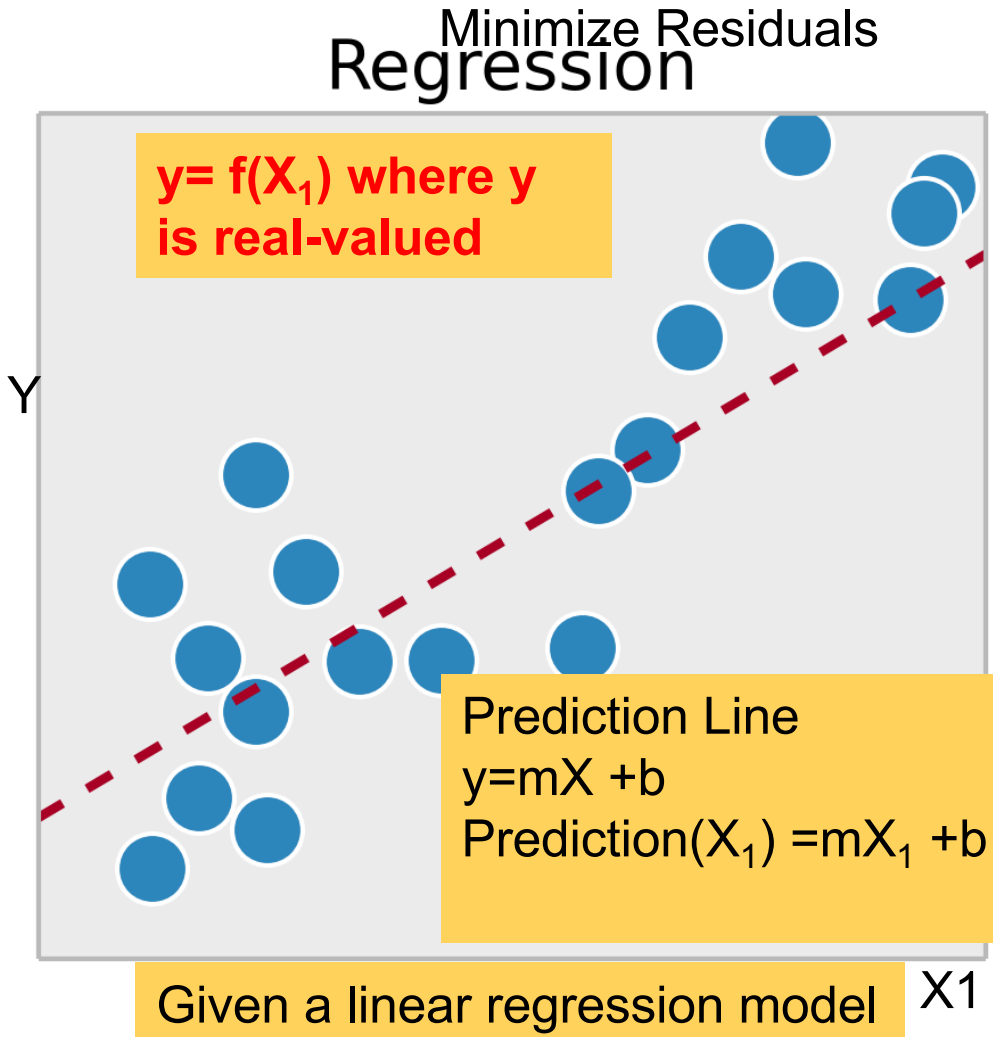
$$f(x_i, W, b) = Wx_i + b$$

[32x32x3]
array of numbers 0...1
(3072 numbers total)

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}log\left(\hat{p}^{(i)}\right) + \left(1-y^{(i)}\right)log\left(1-\hat{p}^{(i)}\right)\right]$$

where $\hat{p} = h_\theta(\mathbf{x}) = \sigma\left(\theta^T \cdot \mathbf{x}\right)$

Linear Regression Loss Function

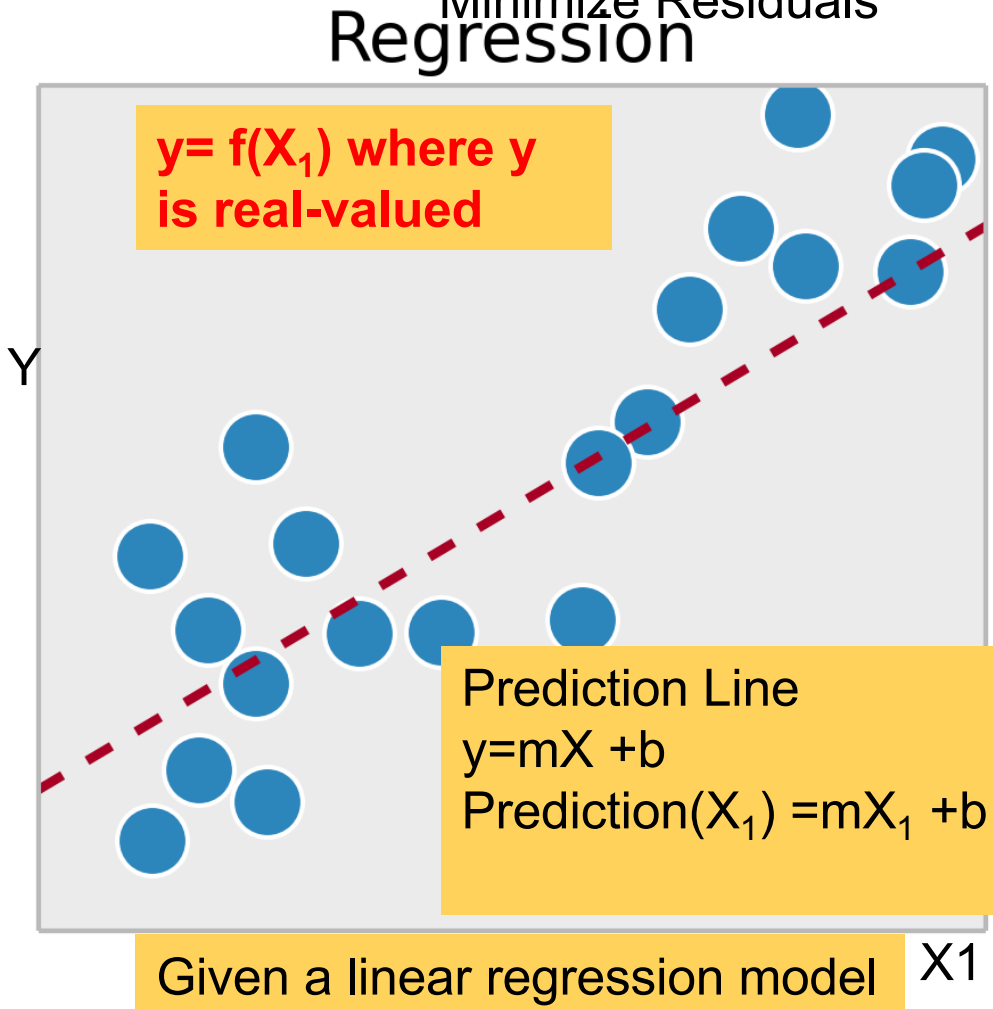


Given a linear regression model W, Please type in the loss function for linear regression

Linear Regression Loss Function

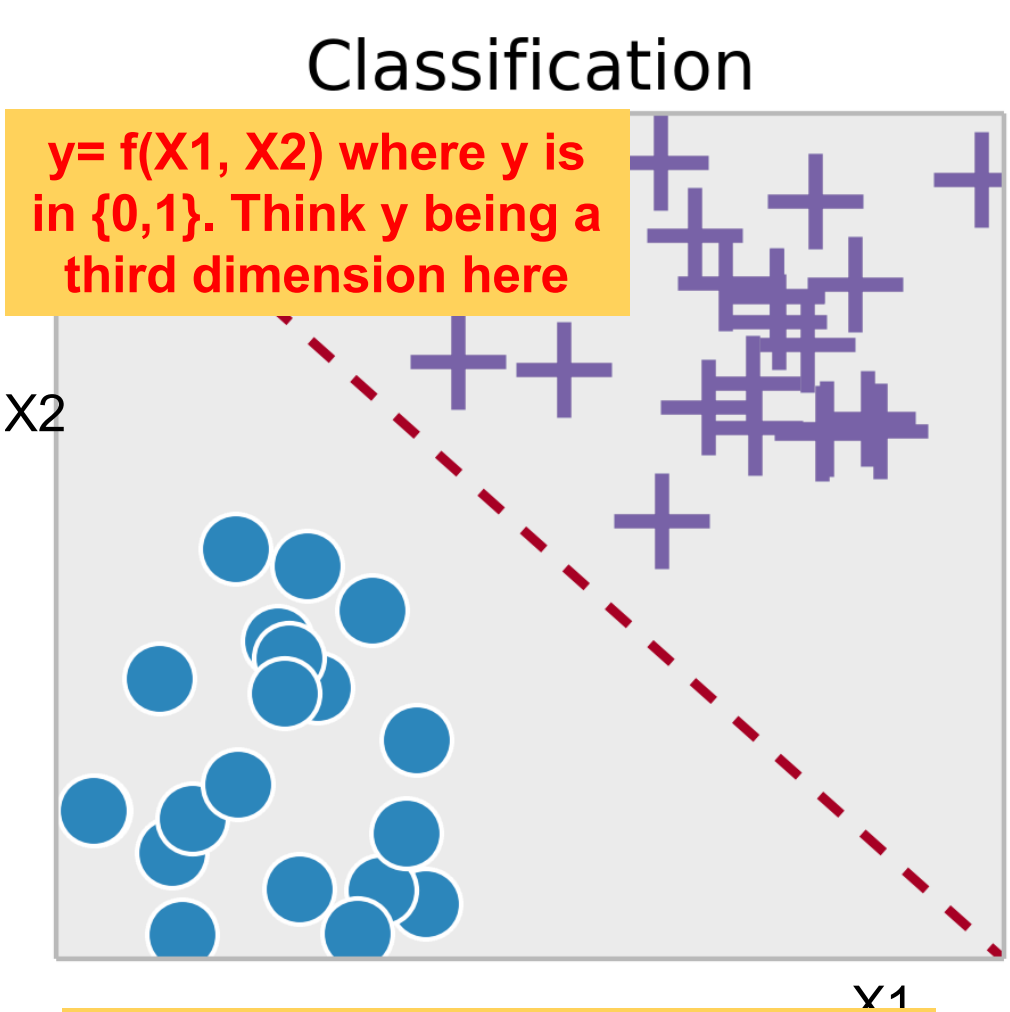
$$MSE(W) = \frac{1}{n}\sum (y_i - W^T X_i)^2$$

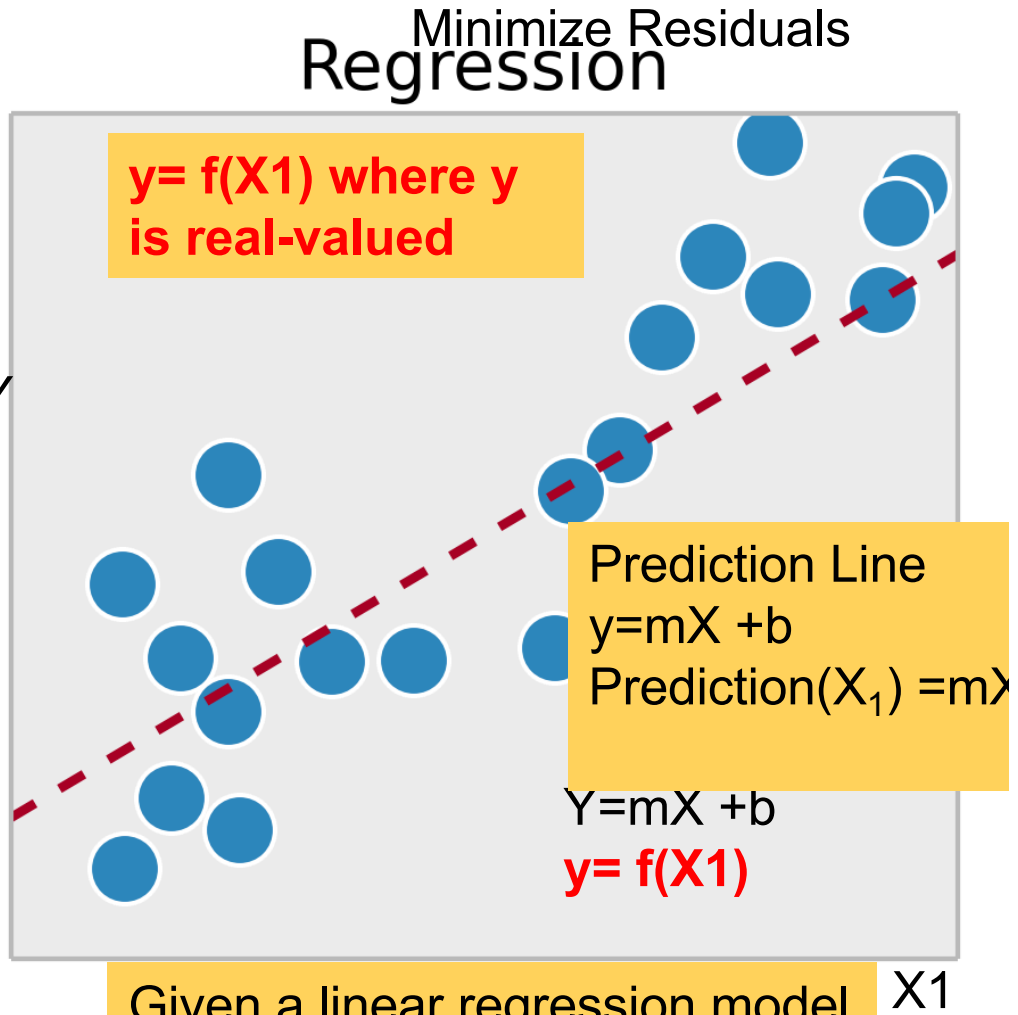
Minimize Residuals



Given a linear regression model W, Please type in the loss function for linear regression

Convex optimization in ML: Loss Functions



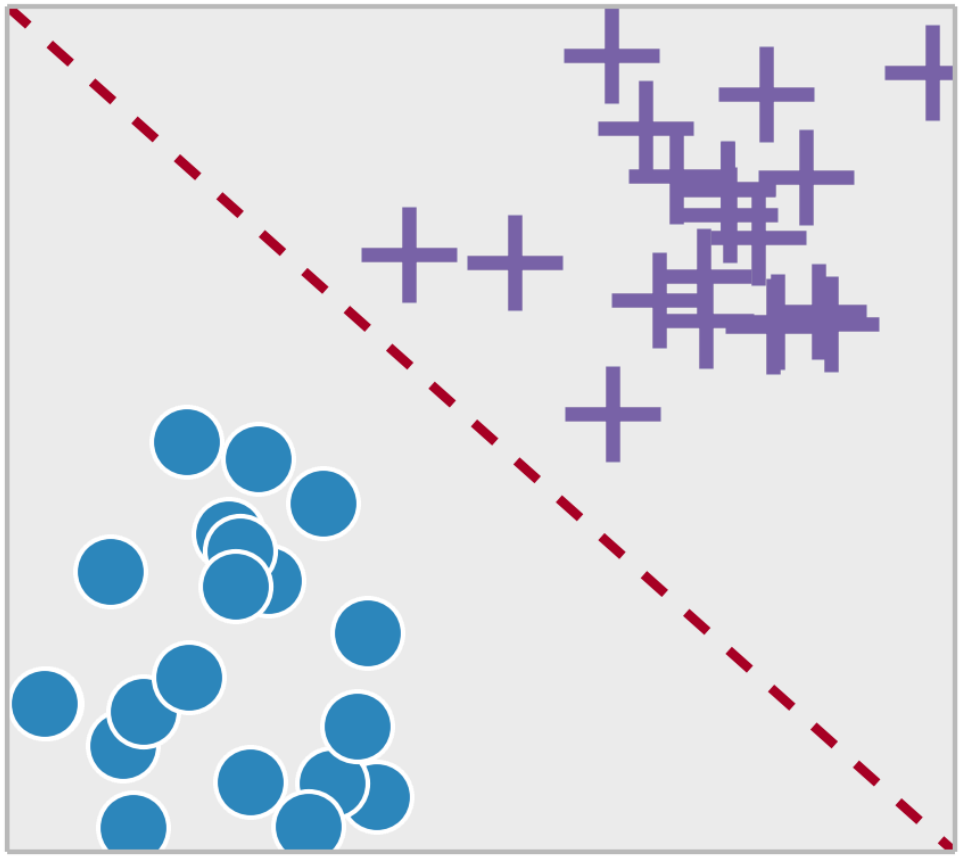


Convex optimization in ML

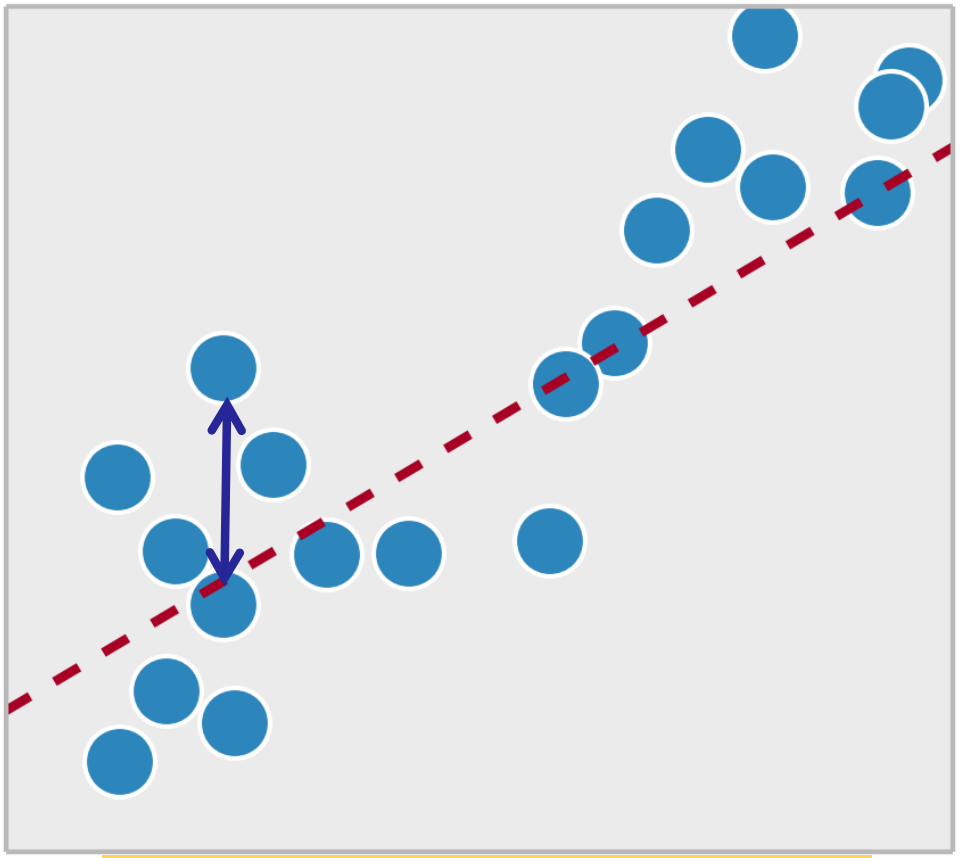
$$MSE(W) = \frac{1}{n}\sum (y_i - W^T X_i)^2$$

Minimize Residuals

Classification



Regression



Please type in the loss function for linear regression (W is the model)

Convex optimization in ML

$$J_P(W, X_1^L) = \sum_{\{X_i | y_i \langle W, X_i \rangle < 0\}} (W^T X_i y_i)$$

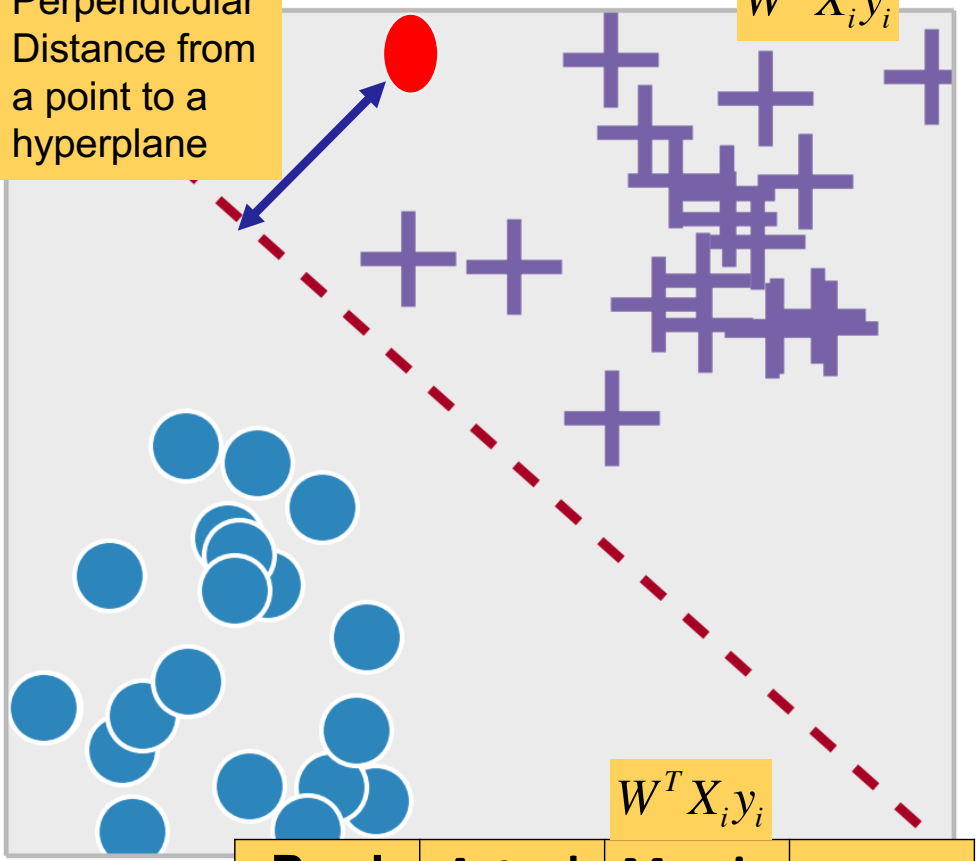
$$MSE(W) = \frac{1}{n} \sum (y_i - W^T X_i)^2$$

Minimize Residuals

Classification

Perpendicular Distance from a point to a hyperplane

$W^T X_i y_i$

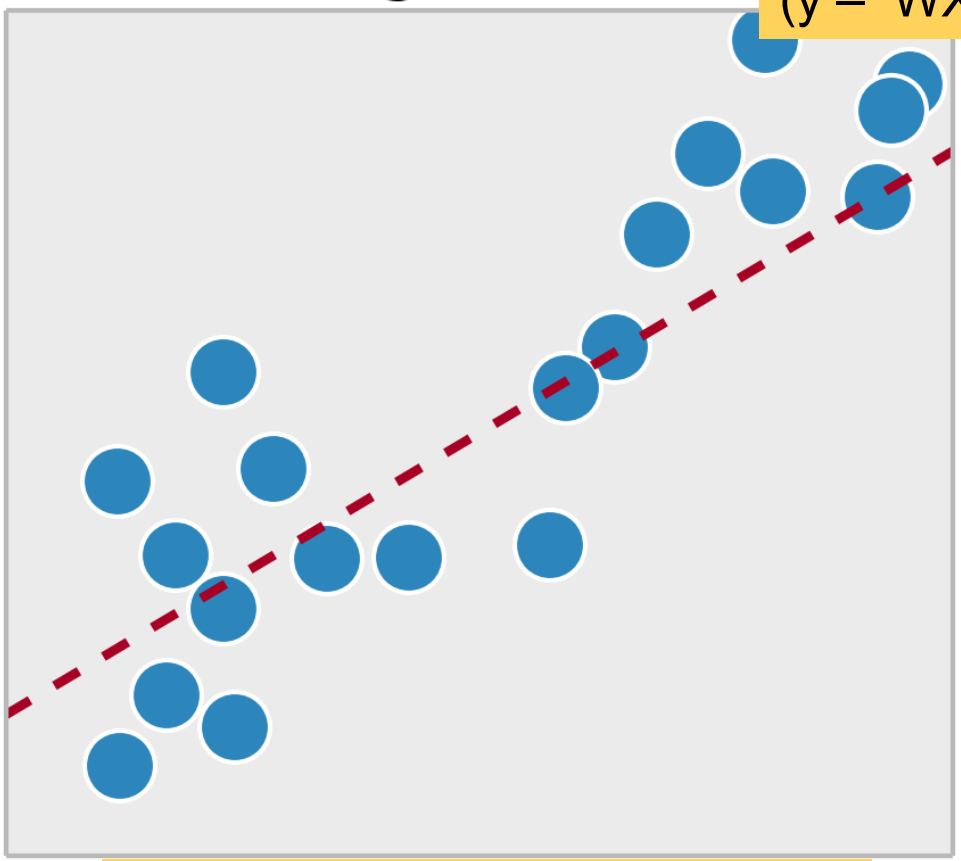


$W^T X_i y_i$

Pred	Actual	Margin	
+	+	+	(GOOD)
+	-	-	LOSS
-	+	-	LOSS
-	-	+	(GOOD)

Regression

$(y - WX)^2$

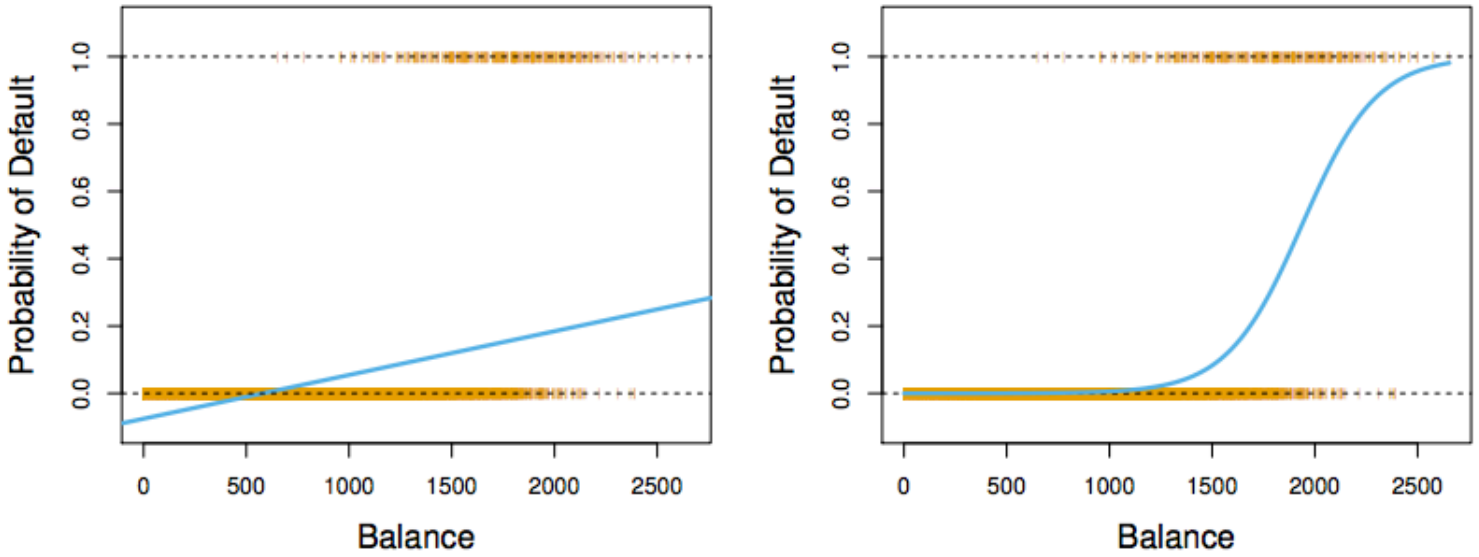


Please type in the loss function for linear regression (W is the model)

Customer will default based on current balance

Linear versus Logistic Regression

One-Dim-Input → Class



The orange marks indicate the response Y, either 0 or 1. Linear regression does not estimate $\Pr(Y = 1|X)$ well. Logistic regression seems well suited to the task.

Multiple classes

Now suppose we have a response variable with three possible values. A patient presents at the emergency room, and we must classify them according to their symptoms.

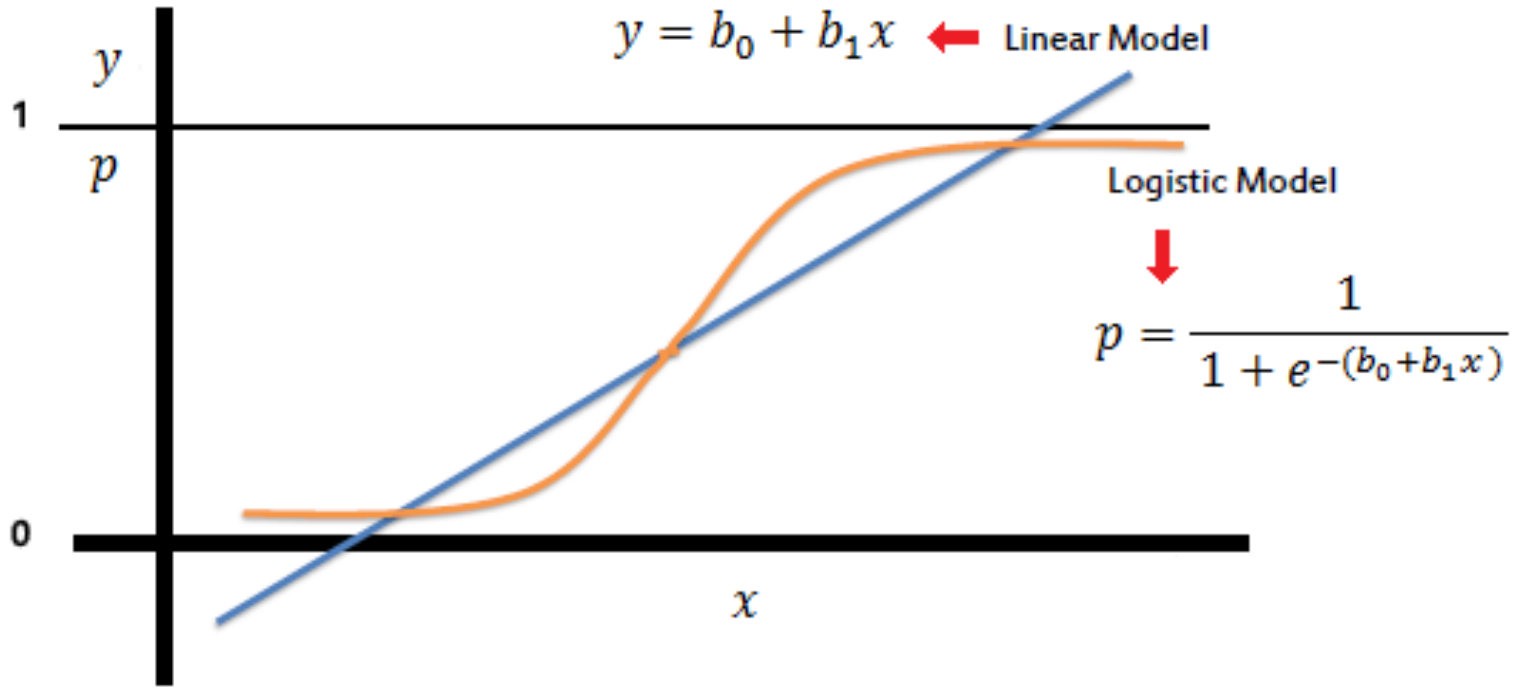
$$Y = \begin{cases} 1 & \text{if } \texttt{stroke}; \\ 2 & \text{if } \texttt{drug overdose}; \\ 3 & \text{if } \texttt{epileptic seizure}. \end{cases}$$

This coding suggests an ordering, and in fact implies that the difference between `stroke` and `drug overdose` is the same as between `drug overdose` and `epileptic seizure`.

Linear regression is not appropriate here.
Multiclass Logistic Regression or *Discriminant Analysis* are more appropriate.

Logistic regression

- Logistic regression was developed by statistician David Cox in 1958.[2][3]
- AKA Logit regression
- The binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features).
- It allows one to say that the presence of a risk factor increases the probability of a given outcome by a specific percentage.


$y = b_0 + b_1 x$
Linear Model
y
1
p
Logistic Model
$p = \frac{1}{1 + e^{-(b_0 + b_1 x)}}$
0
x

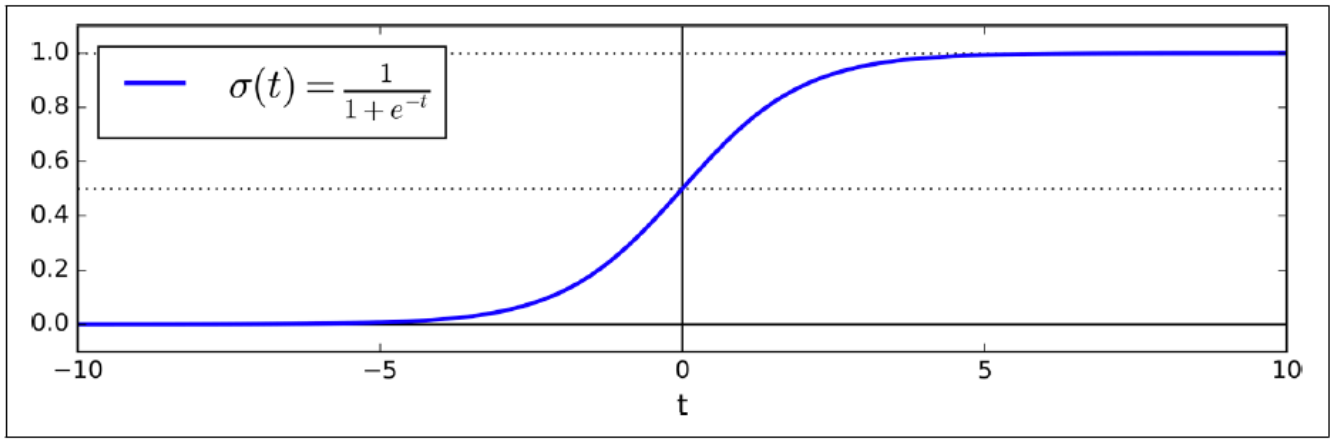
Logistic Regression

- **Logistic Regression model estimated probability (vectorized form)**

$$\hat{p} = h_\theta(\mathbf{x}) = \sigma\left(\theta^T \cdot \mathbf{x}\right)$$

Logistic

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



Classification

$$\hat{y} = \begin{cases} 0 \text{ if } \hat{p} < 0.5, \\ 1 \text{ if } \hat{p} \geq 0.5. \end{cases}$$

Notice that $\sigma(t) < 0.5$ when $t < 0$, and $\sigma(t) \geq 0.5$ when $t \geq 0$, so a Logistic Regression model predicts 1 if $\theta^T \cdot \mathbf{x}$ is positive, and 0 if it is negative.

LogReg cost function(binary case)

- **The objective of training is to set the parameter vector θ so that the model estimates high probabilities for positive instances (y = 1) and low probabilities for negative instances (y = 0). This idea is captured by the cost function shown here for a single training instance x.**

Cost function for a single training example

Focus on the class of interest

$$c(\theta) = \begin{cases} -\log\left(\hat{p}\right) & \text{if } y = 1, \\ -\log\left(1 - \hat{p}\right) & \text{if } y = 0. \end{cases}$$

0 cost if p is 1

- **When y=1 then cost is 0 if p is 1**
  - otherwise →∞ as p→0
  - This cost function makes sense because – log(t) grows very large when t approaches 0, so the cost will be large if the model estimates a probability close to 0 for a positive

Logistic Regression cost function (log loss)

Log of Cost function for all training examples

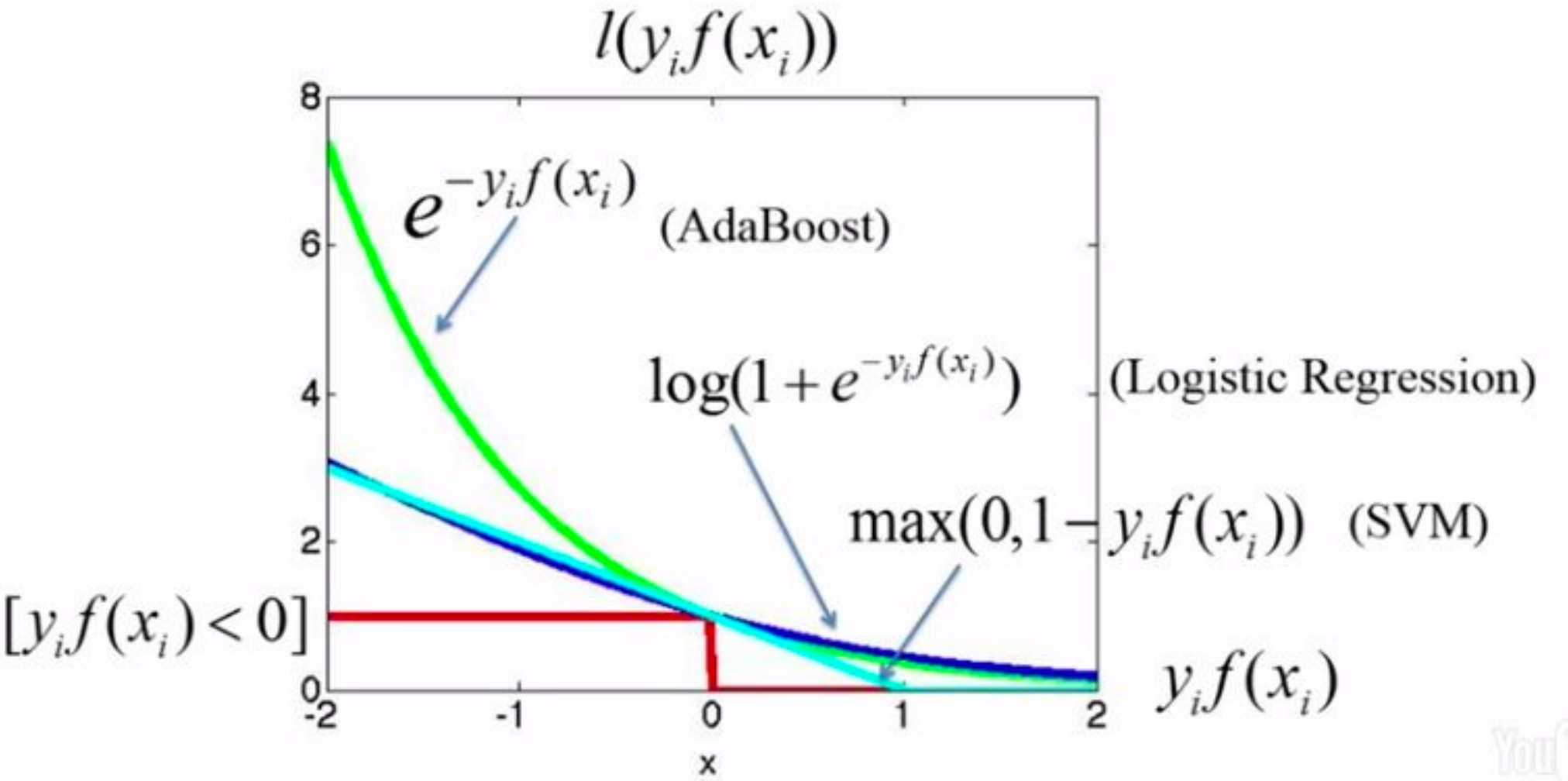
$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right)log\left(1 - \hat{p}^{(i)}\right)\right]$$

$$= \log(1 + e^{-y_i f(x_i)})$$

Expressed in terms of margin $f(x_i) = wx_i$;

- **No known closed-form equation**
  - The bad news is that there is no known closed-form equation to compute the value of θ that minimizes this cost function (there is no equivalent of the Normal Equation in linear reg.).
- **Convex cost function → GD**
  - But the good news is that this cost function is convex, so Gradient Descent (or any other optimization algorithm) is guaranteed to find the global minimum (if the learning rate is not too large and you wait long enough).

Loss Functions for Classification



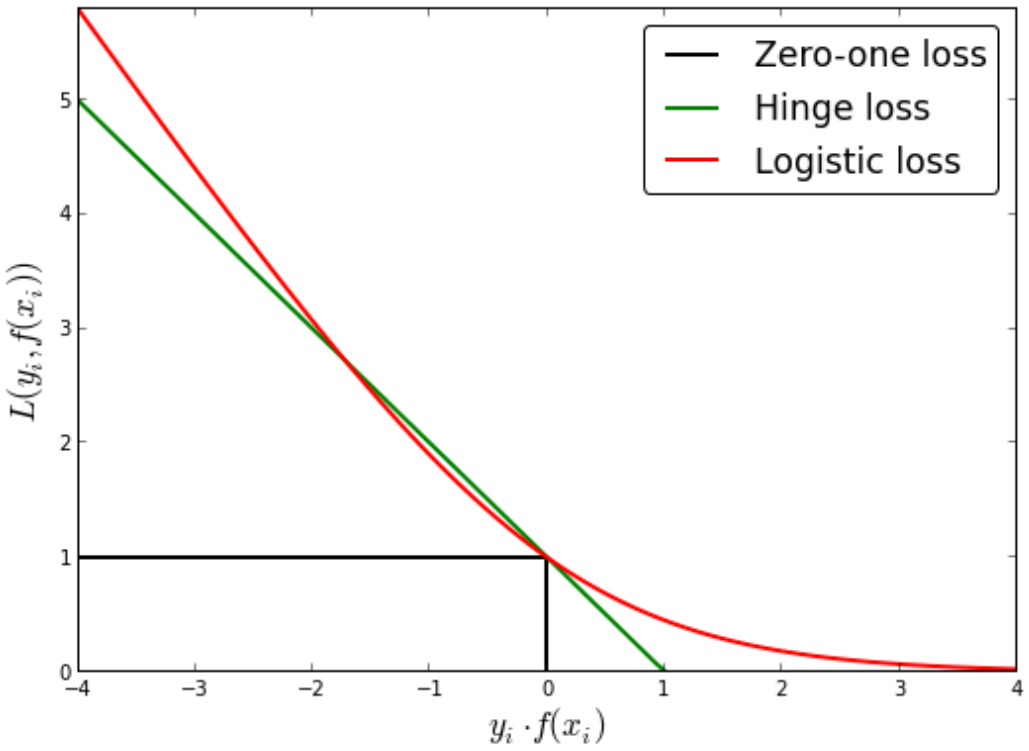
In machine learning it is common to formulate the classification task as a regression loss function minimization problem over a given loss function. Given data input data $(x_1, \ldots, x_n)$ and associated labels $(y_1, \ldots, y_n), y_i \in \{-1, 1\}$, the problem becomes to find a function $f(x)$ that minimizes

$$L(x, y) = \sum_{i}^{n} \text{loss}(f(x_i), y_i)$$

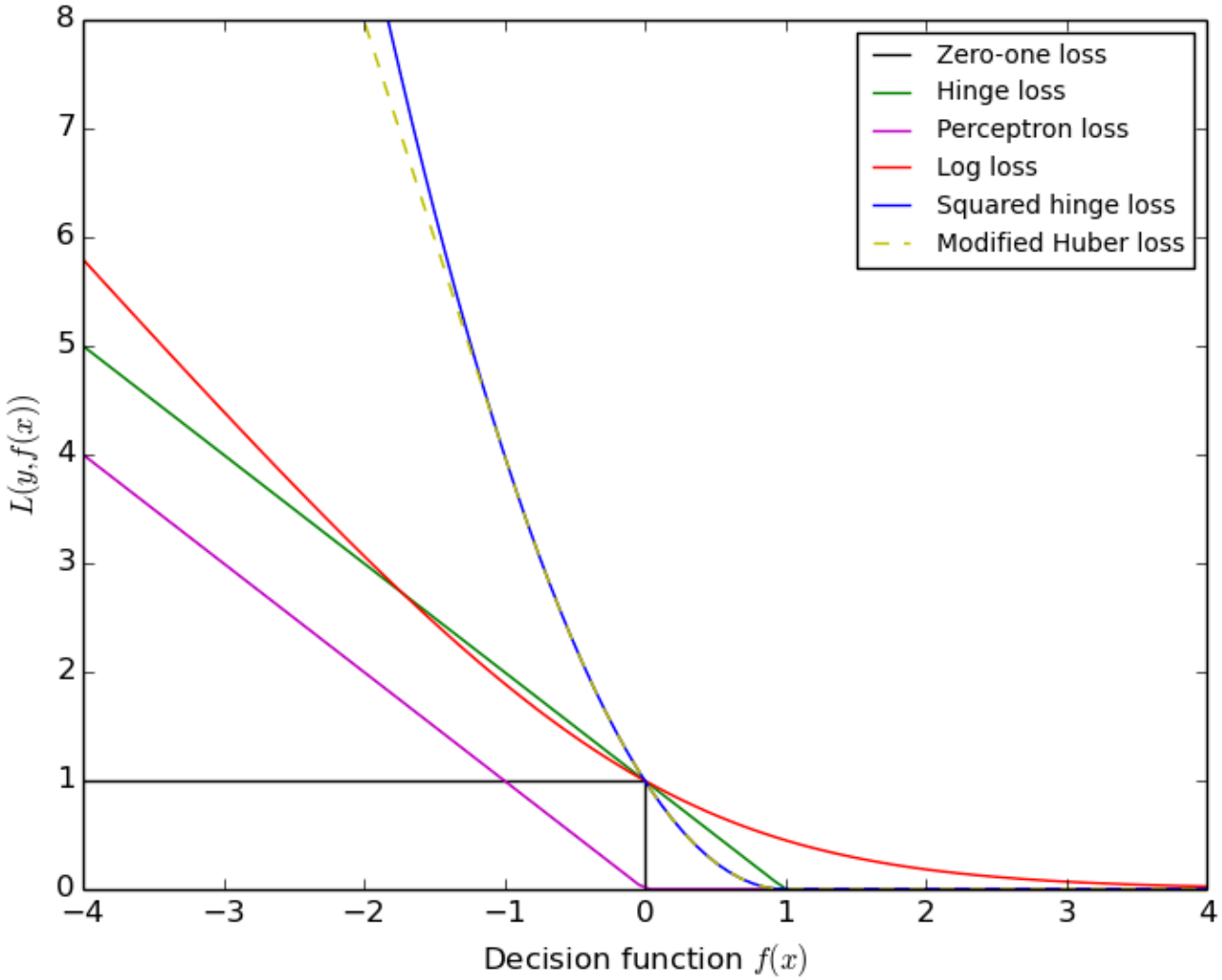
where loss is any loss function. These are usually functions that become close to zero when $f(x_i)$ agrees in sign with y_i and have a non-negative value when $f(x_i)$ have opposite signs. Common choices of loss functions are:

- Zero-one loss, $I(f(x_i) = y_i)$, where I is the indicator function.
- Hinge loss, $\max(0, 1 - f(x_i)y_i)$
- Logistic loss, $\log(1 + \exp f(x_i)y_i)$

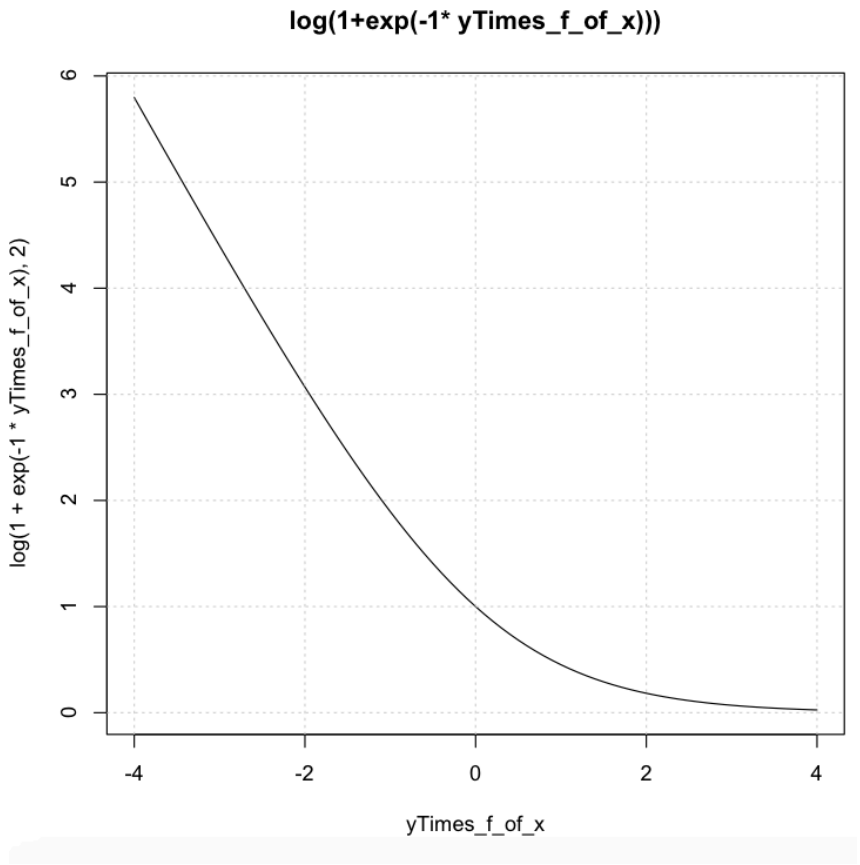
Logit space -2sigma, 2sigma



If y =1 P,
If y =-1 1-P


Zero-one loss
Hinge loss
Perceptron loss
Log loss
Squared hinge loss
Modified Huber loss
$L(y, f(x))$
Decision function $f(x)$
8
7
6
5
4
3
2
1
0
−4
−3
−2
−1
0
1
2
3
4

Log loss function (base 2)



Convex so the loss function is also convex (sum of convex functions)

```
> yTimes_f_of_x = seq(-4, 4, 0.1)
> plot(yTimes_f_of_x, log(1+exp(-1* yTimes_f_of_x),2), main="log(1+exp(-1* yTimes_f_of_x)))",
type="l")
> grid()
>
```

Negative Log Likelihood of Learnt Model W

- **Negative log-likelihood of our model**

$$\hat{\beta} = \text{argmin}_\beta \sum_{i=1}^{n} -\log p(y_i|x_i;\beta) + \mu \sum_{j=0}^{d} \beta_j^2.$$

Minimize $-2l(W) = -2\sum_{i=1}^{m} y^i \log p - (1-y^i)\log(1-p)$

Actual Predicted
-1 * log(0.5)
-1 * -0.7 = 0.7 Error
Bigger error

vs

-1 * log(0.9)
-1*-0.1 = 0.1 Error

- **AKA Residual Deviance**
- **R Code for negative log likelihood for MROZ data**

```
library("car") #Mroz
attach(Mroz)
mod.mroz.glm = glm(lfp ~ k5 + k618 + age +wc + hc + lwg + inc,
    family=binomial))
 coefficients(mod.mroz.glm)
 phat=mod.mroz.glm$fitted.values
 y= ifelse(lfp=='yes', 1,0)
minusTwoTimesLogLik = -2 * sum(y*log(phat) + (1-y)*log(1-phat) )
```

```
mod.mroz.glm  =glm(lfp ~ k5 + k618 + age, family=binomial)
phat=mod.mroz.glm$fitted.values
 y= ifelse(lfp=='yes', 1,0)
minusTwoTimesLogLik = -2 * sum(y*log(phat) + (1-y)*log(1-phat
> minusTwoTimesLogLik
[1] 960.7074
```

905.266 is the minusTwoTimesLogLikelihood of the data given the learnt model (lower is better!)

LR Model Summary

- **summary(mod.mroz.glm)**
- Call:

```
glm(formula = lfp ~ k5 + k618 + age + wc + hc + lwg + inc, family = binomial)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.1062  -1.0900   0.5978   0.9709   2.1893

Coefficients:
             Estimate   Std. Error  z value     Pr(>|z|)
(Intercept)  3.182140   0.644375    4.938     7.88e-07 ***
k5          -1.462913   0.197001   -7.426     1.12e-13 ***
k618        -0.064571   0.068001   -0.950     0.342337
age         -0.062871   0.012783   -4.918     8.73e-07 ***
wc           0.807274   0.229980    3.510     0.000448 ***
hc           0.111734   0.206040    0.542     0.587618
lwg          0.604693   0.150818    4.009     6.09e-05 ***
inc         -0.034446   0.008208   -4.196     2.71e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
    Null deviance: 1029.75  on 752  degrees of freedom
Residual deviance:  905.27  on 745  degrees of freedom
AIC: 921.27
Number of Fisher Scoring iterations: 4
```

Residual distribution in logit space
Use summary(predict(mod.mroz.glm, Mroz)) to recover the probabilities

Feature Significance

Null Deviance where phat is #successes/#events
Residual Deviance= -2*LogLikelihood
See later slides for calulcation

AIC = 905+ 2* 8 #7 variables + bias term

Loss functions have similar structure for regression and classification

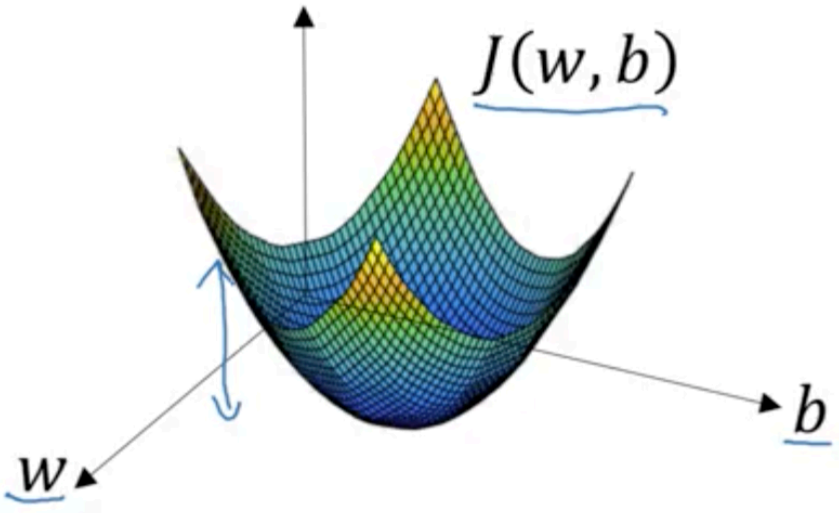
Convex Loss: but no closed form sol^n

Gradient Descent

Recap: $\hat{y} = \sigma(w^T x + b),\ \sigma(z) = \frac{1}{1+e^{-z}}$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

Want to find w, b that minimize $J(w,b)$



Learning: Logistic Regression gradient

Log of Cost function for all training examples

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}log\left(\hat{p}^{(i)}\right) + \left(1-y^{(i)}\right)log\left(1-\hat{p}^{(i)}\right)\right]$$

Avg cost wrt training data

- **Logistic cost function partial derivatives wrtt jth model parameter**

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma\left(\theta^T\cdot\mathbf{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

- **Gradient**
  - for each instance it computes the prediction error and multiplies it by the jth feature value, and then it computes the average over all training instances.
- **Use Gradient Descent algorithm.**

See here for detailed derivation
https://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf

Loss functions have similar structure

- **Linear regression**
  - Objective function : $\text{MSE}(\mathbf{X}, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}\right)^2$
  - Gradient for Linear Regression:

$$\frac{\partial}{\partial \theta_j}\text{MSE}(\theta) = \frac{2}{m}\sum_{i=1}^{m}\left(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}\right)x_j^{(i)}$$

- **Classification**
  - Gradient for Binomial Logistic Regression
    - $(\Pr(y_i|\, X_i; W) - y_i)\, X_i$

$$\frac{\partial}{\partial \theta_j}\text{J}(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma\left(\theta^T \cdot \mathbf{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

  - Gradient for Perceptron (for errors):
    - $y_i\, X_i$

▽ the gradient chorus/mantra

- **What is the gradient for linear regression?**

$$\frac{\partial E}{\partial W} = \underbrace{(O - t)}_{\text{weight}}\underbrace{X}_{\text{example}}$$

$$W = W - \alpha \times \frac{\partial E}{\partial W}$$

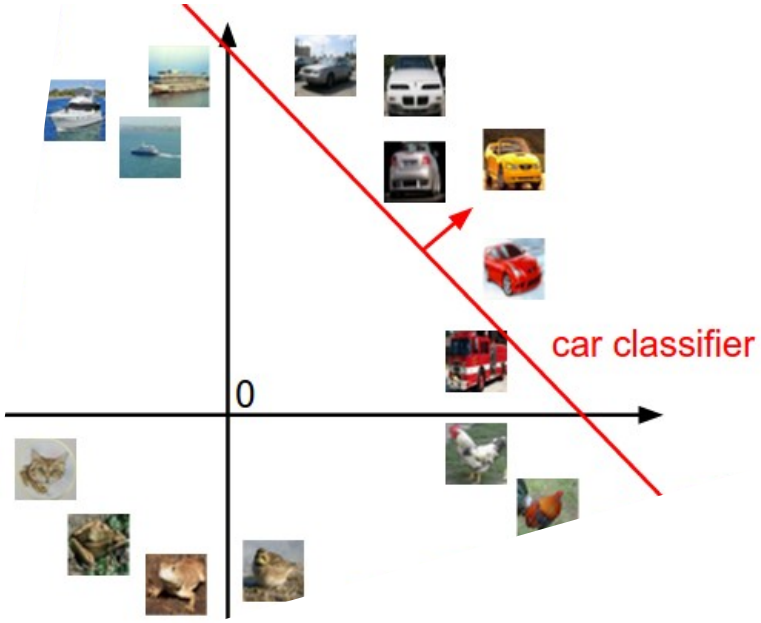
$$\begin{bmatrix} 4.32 \\ 3.98 \\ 7.92 \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \\ 8 \end{bmatrix} - 0.01 \times 2 \times \begin{bmatrix} \mathbf{34} \\ \mathbf{1} \\ \mathbf{4} \end{bmatrix}$$

$$4.32 = 5 - 0.01 \times (2 \times 34) \text{ for } i = 1$$

- **Chorus**
  - The gradient is the weighted sum of the training data, where the weights are proportional to the error (for each example) !

Interpreting a Linear Classifier

Learn θ which is W, b the parameters of a separating hyperplane



$$f(x_i, W, b) = Wx_i + b$$

[32x32x3]
array of numbers 0...1
(3072 numbers total)

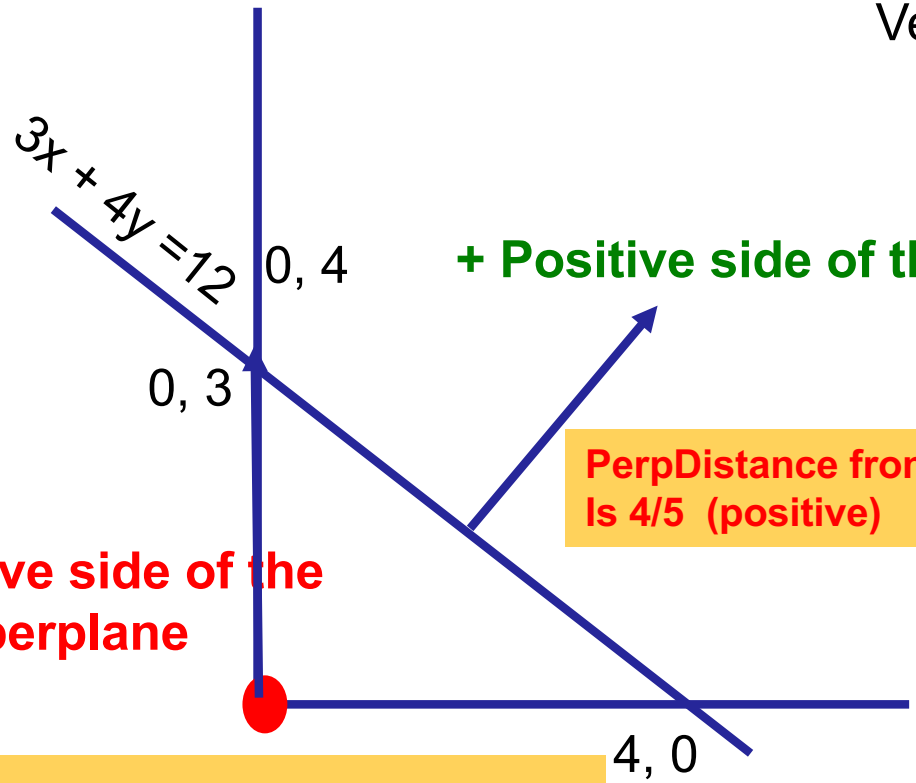
$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}log\left(\hat{p}^{(i)}\right) + \left(1-y^{(i)}\right)log\left(1-\hat{p}^{(i)}\right)\right]$$

where $\hat{p} = h_\theta(\mathbf{x}) = \sigma\left(\theta^T \cdot \mathbf{x}\right)$

REFRESH: HyperPlane 3x + 4y =12

QUESTION: What side of the hyperplane does the origin lie?

Shortcut: 3(0) + 4(0) -10 / SQTR((3 * 3) (4*4)) = -10/5=-2



Vector that normal to Hyperpane (3, 4)

+ Positive side of the hyperplane

PerpDistance from point (0,4) to the hyperplane 3x + 4y =12 **Is 4/5 (positive)**

- Negative side of the hyperplane

The distance from a point to a plane — is equal to length of the perpendicular lowered from a point on a plane.

If $Ax + By + Cz + D = 0$ is a plane equation, then distance from point $M(M_x, M_y, M_z)$ to plane can be found using the following formula

$$d = \frac{|A \cdot M_x + B \cdot M_y + C \cdot M_z + D|}{\sqrt{A^2 + B^2 + C^2}}$$

Shortcut:

$$\frac{(3(0) + 4(0) - 12)}{\text{SQTR}((3 * 3) + (4*4))} = -12/5 = -2.4$$

Outline

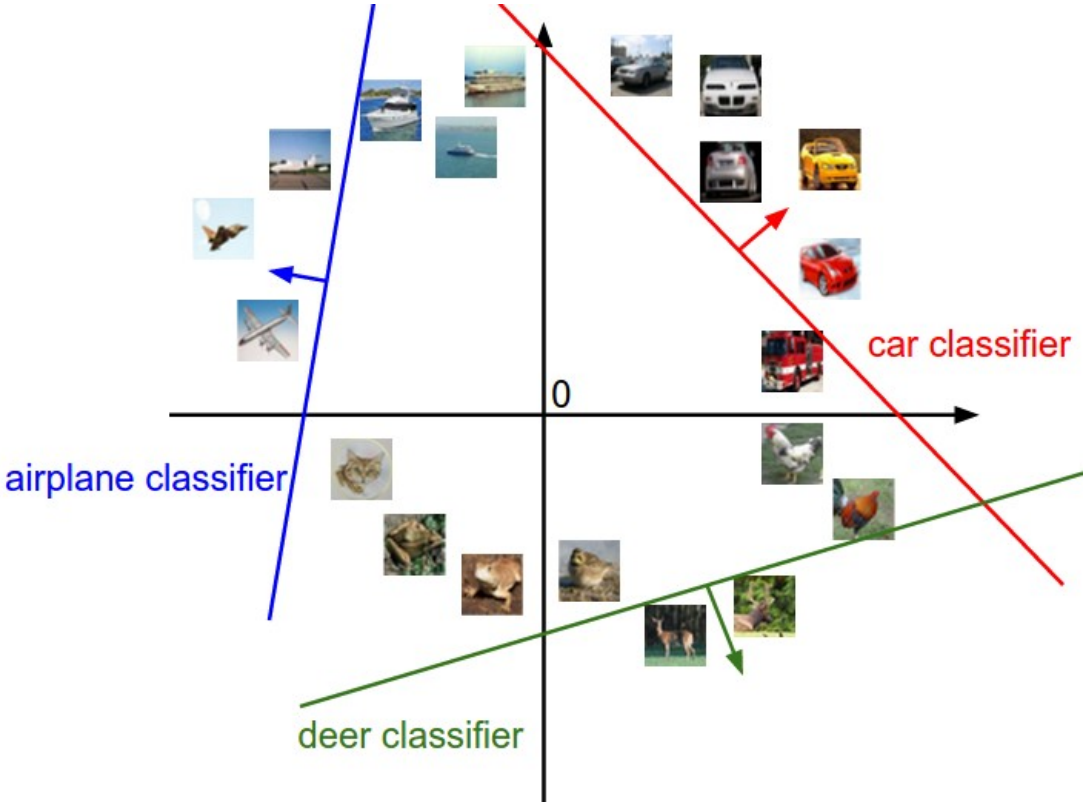
- Introduction
- Binomial logistic regression
- Multinomial Logistic regression
- Linear Classifier via separating hyperplane
- Multinomial Logistic regression Classifier
- Linear classifier demo
- Learning Softmax Classifiers via optimization
  - Implementation (SoftMax Classifier)
- Regressions in matrix terms and in graph terms
- Summary

binomial, ordinal or multinomial LR

- Logistic regression can be binomial, ordinal or multinomial.
  - Binomial or binary logistic regression deals with situations in which the observed outcome for a dependent variable can have only two possible types, "0" and "1" (which may represent, for example, "dead" vs. "alive" or "win" vs. "loss").
  - Multinomial logistic regression deals with situations where the outcome can have three or more possible types (e.g., "disease A" vs. "disease B" vs. "disease C") that are not ordered.
  - Ordinal logistic regression deals with dependent variables that are ordered. In binary logistic regression, the outcome is usually coded as "0" or "1", as this leads to the most straightforward interpretation.

SoftMax Classifier

Interpreting a Linear Classifier



$$f(x_i, W, b) = Wx_i + b$$

[32x32x3]
array of numbers 0...1
(3072 numbers total)

From binomial LR to Softmax regression AKA multinomial logistic regression

- The idea is quite simple: when given an instance x, the Softmax Regression model first computes a score $s_k(x)$ for each class k, then estimates the probability of each class by applying the softmax function (also called the normalized exponential) to the scores.
- The equation to compute $s_k(x)$ should look familiar, as it is just like the equation for Linear Regression prediction

$$s_k(\mathbf{x}) = \theta_k^T \cdot \mathbf{x}$$

Perpendicular distance

Logistic Loss → Cross entropy

Cross-entropy is defined as:

$$H(p, q) = \mathrm{E}_p[-\log q] = H(p) + D_{\mathrm{KL}}(p \| q) = -\sum_x p(x) \log q(x)$$

Where, p and q are two distributions and using the definition of K-L divergence. Now if $p \in \{y, 1-y\}$ and $q \in \{\hat{y}, 1-\hat{y}\}$, we can re-write cross-entropy as:

CXE

$$H(p, q) = -\sum_x p_x \log q_x = -y \log \hat{y} - (1-y) \log(1-\hat{y})$$

$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathbf{x})\right)}$$

Pr(Y=k|X)

- K is the number of classes.
- $\mathbf{s}(\mathbf{x})$ is a vector containing the scores of each class for the instance $\mathbf{x}$.
- $\sigma(\mathbf{s}(\mathbf{x}))_k$ is the estimated probability that the instance $\mathbf{x}$ belongs to class k given the scores of each class for that instance.

Max Likelihood →Cross Entropy

Thus, the extrema of $\mathcal{L}$ are equivalent to the extrema of $\log \mathcal{L}$:

$$\log \mathcal{L}(\theta \mid x_1, \ldots, x_n) = \sum_{i=1}^{n} \log f(x_i|\theta)$$

From which the maximum likelihood estimator $\hat{\theta}_{\text{MLE}}$ is defined as:

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta} \sum_{i=1}^{n} \log f(x_i|\theta)$$

As an aside, Bayesians will remind us we can generalized into a MAP estimator, given uniform prior $g(\theta)$:

$$\arg\max_{\theta} \sum_{i=1}^{n} \log f(x_i|\theta) = \arg\max_{\theta} \log(f|\theta) = \arg\max_{\theta} \log(f|\theta)g(\theta) = \hat{\theta}_{\text{MAP}}$$

From which optimization and real analysis reminds us of the following equivalence, for all x:

$$\arg\max_{x}(x) = \arg\min_{x}(-x)$$

Thus, the following are equivalent:

$$\arg\max_{\theta} \sum_{i=1}^{n} \log f(x_i|\theta) = \arg\min_{\theta} - \sum_{i=1}^{n} \log f(x_i|\theta) = \hat{\theta}_{\text{MLE}}$$

From this, we technically have an answer to the above two questions on equivalence. Yet, from here lies the opportunity to continue and *uncover the relationship between MLE/MAP and both entropy and loss via* ***Kullback-Leibler divergence*** (KL). To get there, c[illegible] above:

https://quantivity.wordpress.com/2011/05/23/why-minimize-negative-log-likelihood/

$$\arg\min_{\theta}(\frac{1}{n} \sum_{i=1}^{n} - \log f(x_i|\theta))$$

Which converges, by the strong law of large numbers, to the expectation:

$$E[-\log f(x|\theta)]$$

differences between maximum likelihood and cross entropy as a loss function

- MLE is usually used in more traditional statistical models, while cross entropy is used in machine learning methods notably neural networks and gradient boosted trees etc.
- The main difference between these methods is, MLE tries to get the most of the data points right, it doesn't care about optimizing how more different the right prediction is from the wrong prediction.
- getting the right class VERSUS getting more confidence in the prediction

Logistic Regression

log loss for binary versus cross-entropy for K-classes

	Binomial LR	Multinomial LR
	$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}log\left(\hat{p}^{(i)}\right) + \left(1-y^{(i)}\right)log\left(1-\hat{p}^{(i)}\right)\right]$	$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$
Gradient	$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma\left(\theta^T\cdot\mathbf{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$	$\nabla_{\theta_k} J(\Theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{p}_k^{(i)} - y_k^{(i)}\right)\mathbf{x}^{(i)}$
Class	$\hat{y} = \begin{cases} 0 \text{ if } \hat{p} < 0.5, \\ 1 \text{ if } \hat{p} \geq 0.5. \end{cases}$	$\hat{y} = \underset{k}{\text{argmax}}\ \sigma(\mathbf{s}(\mathbf{x}))_k = \underset{k}{\text{argmax}}\ s_k(\mathbf{x}) = \underset{k}{\text{argmax}}\left(\left(\theta^{(k)}\right)^T\cdot\mathbf{x}\right)$
Prob	$\hat{p} = h_\theta(\mathbf{x}) = \sigma\left(\theta^T\cdot\mathbf{x}\right)$	$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K}\exp\left(s_j(\mathbf{x})\right)}$ • K is the number of classes. • $\mathbf{s}(\mathbf{x})$ is a vector containing the scores of each class for the instance $\mathbf{x}$. • $\sigma(\mathbf{s}(\mathbf{x}))_k$ is the estimated probability that the instance $\mathbf{x}$ belongs to class k given the scores of each class for that instance.

▽ the gradient chorus/mantra

- **What is the gradient for linear regression?**

$$\frac{\partial E}{\partial W} = \underbrace{(O - t)}_{\text{weight}}\underbrace{X}_{\text{example}}$$

$$W = W - \alpha \times \frac{\partial E}{\partial W}$$

$$\begin{bmatrix} 4.32 \\ 3.98 \\ 7.92 \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \\ 8 \end{bmatrix} - 0.01 \times 2 \times \begin{bmatrix} \mathbf{34} \\ \mathbf{1} \\ \mathbf{4} \end{bmatrix}$$

$$4.32 = 5 - 0.01 \times (2 \times 34) \text{ for } i = 1$$

- **Chorus**
  - The gradient is the weighted sum of the training data, where the weights are proportional to the error (for each example) !

Softmax Regression: Argmax Classifier

$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathbf{x})\right)}$$

- K is the number of classes.
- $\mathbf{s}(\mathbf{x})$ is a vector containing the scores of each class for the instance $\mathbf{x}$.
- $\sigma(\mathbf{s}(\mathbf{x}))_k$ is the estimated probability that the instance $\mathbf{x}$ belongs to class k given the scores of each class for that instance.

Just like the Logistic Regression classifier, the Softmax Regression classifier predicts the class with the highest estimated probability (which is simply the class with the highest score)

$$\hat{y} = \underset{k}{\operatorname{argmax}}\ \sigma(\mathbf{s}(\mathbf{x}))_k = \underset{k}{\operatorname{argmax}}\ s_k(\mathbf{x}) = \underset{k}{\operatorname{argmax}}\ \left(\left(\theta^{(k)}\right)^T \cdot \mathbf{x}\right)$$

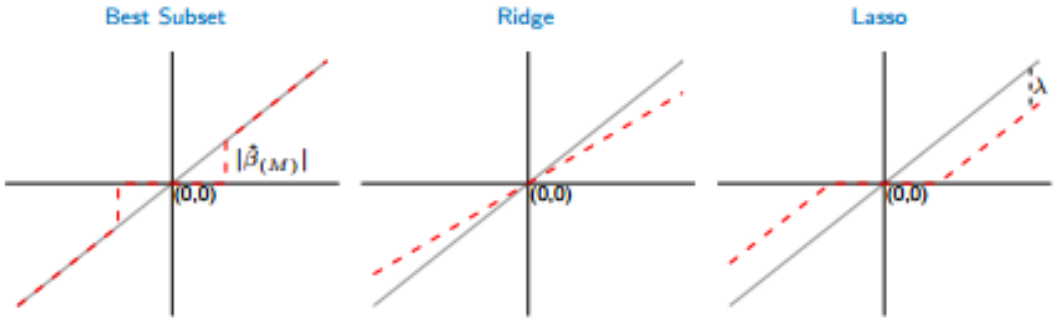
- The *argmax* operator returns the value of a variable that maximizes a function. In this equation, it returns the value of k that maximizes the estimated probability $\sigma(\mathbf{s}(\mathbf{x}))_k$.

Logistic Regression and Regularized LR

- **Penalized logistic regression**
  - Laplace (L1) versus gaussian (L2)

LogReg Regularization: Constrained optimization

- **Add regularization to prevent over-fitting**
- **Lasso**
  - L1 Norm regularization: **|w|**
- **Ridge**
  - L2 Norm regularization: **w^2**
- **shrinkage of w**
- **Elastic net**



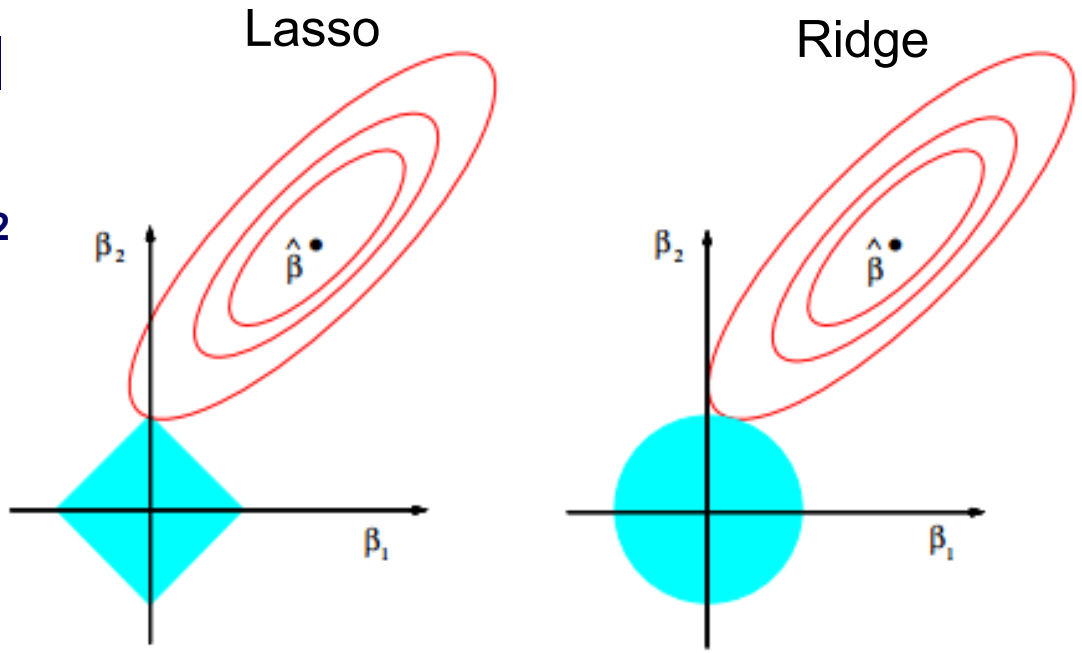


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

From The Elements of Statistical Learning

Priors on Logistic Regression

- **Penalized logistic regression**
  - Laplace (L1) versus gaussian (L2)

- https://www.dropbox.com/s/he10166scbsmncz/logisticRegression-L1-L2-Laplace-versus-Gaussian.pdf?dl=0

- https://cs.brown.edu/courses/archive/2006-2007/cs195-5/lectures/lecture13.pdf

Notebooks for logistic Regression

- **Weights?**
- **Third party notebooks**
  - http://ipython-books.github.io/featured-04/
  - The rest of the chapter contains the following recipes:
    - Predicting who will survive on the Titanic with logistic regression
    - Learning to recognize handwritten digits with a K-nearest neighbors classifier
    - Learning from text: Naive Bayes for Natural Language Processing
    - Using Support Vector Machines for classification tasks
    - Using a random forest to select important features for regression
    - Reducing the dimensionality of a data with a Principal Component Analysis
    - Detecting hidden structures in a dataset with clustering
    - You'll find the rest of the chapter in the full version of the IPython Cookbook, byCyrille Rossant, Packt Publishing, 2014.

Outline

- Introduction
- Binomial logistic regression
- Multinomial Logistic regression
- Linear Classifier via separating hyperplane
- Multinomial Logistic regression Classifier
- Linear classifier demo
- Learning Softmax Classifiers via optimization
  - Implementation (SoftMax Classifier)
- Regressions in matrix terms and in graph terms
- Summary

Softmax classification rule

$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathbf{x})\right)}$$

- K is the number of classes.
- $\mathbf{s}(\mathbf{x})$ is a vector containing the scores of each class for the instance $\mathbf{x}$.
- $\sigma(\mathbf{s}(\mathbf{x}))_k$ is the estimated probability that the instance $\mathbf{x}$ belongs to class k given the scores of each class for that instance.

Just like the Logistic Regression classifier, the Softmax Regression classifier predicts the class with the highest estimated probability (which is simply the class with the highest score), as shown in Equation 4-21.

Equation 4-21. Softmax Regression classifier prediction

$$\hat{y} = \underset{k}{\operatorname{argmax}}\ \sigma(\mathbf{s}(\mathbf{x}))_k = \underset{k}{\operatorname{argmax}}\ s_k(\mathbf{x}) = \underset{k}{\operatorname{argmax}}\ \left(\left(\theta^{(k)}\right)^T \cdot \mathbf{x}\right)$$

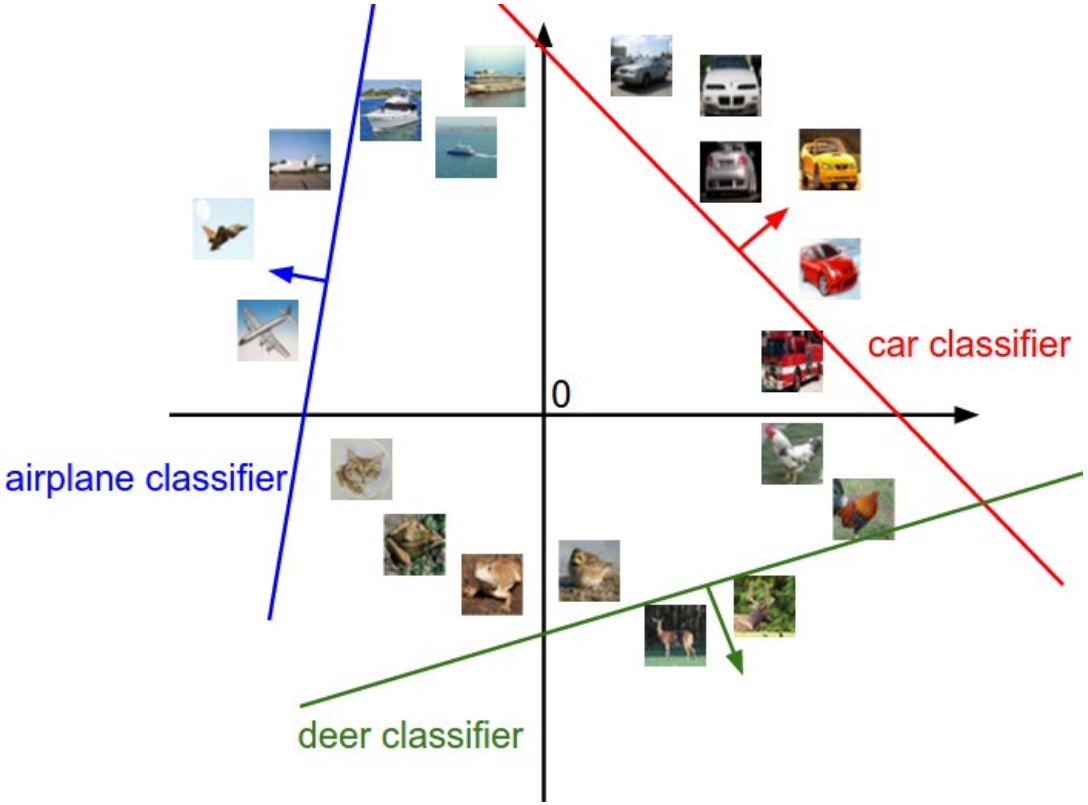
- The *argmax* operator returns the value of a variable that maximizes a function. In this equation, it returns the value of k that maximizes the estimated probability $\sigma(\mathbf{s}(\mathbf{x}))_k$.

Logistic Regression

log loss for binary versus cross-entropy for K-classes

	Binomial LR	Multinomial LR
	$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}log\left(\hat{p}^{(i)}\right) + \left(1-y^{(i)}\right)log\left(1-\hat{p}^{(i)}\right)\right]$	$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$
Gradient	$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma\left(\theta^T\cdot\mathbf{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$	$\nabla_{\theta_k} J(\Theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{p}_k^{(i)} - y_k^{(i)}\right)\mathbf{x}^{(i)}$
Class	$\hat{y} = \begin{cases} 0 \text{ if } \hat{p} < 0.5, \\ 1 \text{ if } \hat{p} \geq 0.5. \end{cases}$	$\hat{y} = \underset{k}{\text{argmax}}\ \sigma(\mathbf{s}(\mathbf{x}))_k = \underset{k}{\text{argmax}}\ s_k(\mathbf{x}) = \underset{k}{\text{argmax}}\left(\left(\theta^{(k)}\right)^T\cdot\mathbf{x}\right)$
Prob	$\hat{p} = h_\theta(\mathbf{x}) = \sigma\left(\theta^T\cdot\mathbf{x}\right)$	$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K}\exp\left(s_j(\mathbf{x})\right)}$ • K is the number of classes. • $\mathbf{s}(\mathbf{x})$ is a vector containing the scores of each class for the instance $\mathbf{x}$. • $\sigma(\mathbf{s}(\mathbf{x}))_k$ is the estimated probability that the instance $\mathbf{x}$ belongs to class k given the scores of each class for that instance.

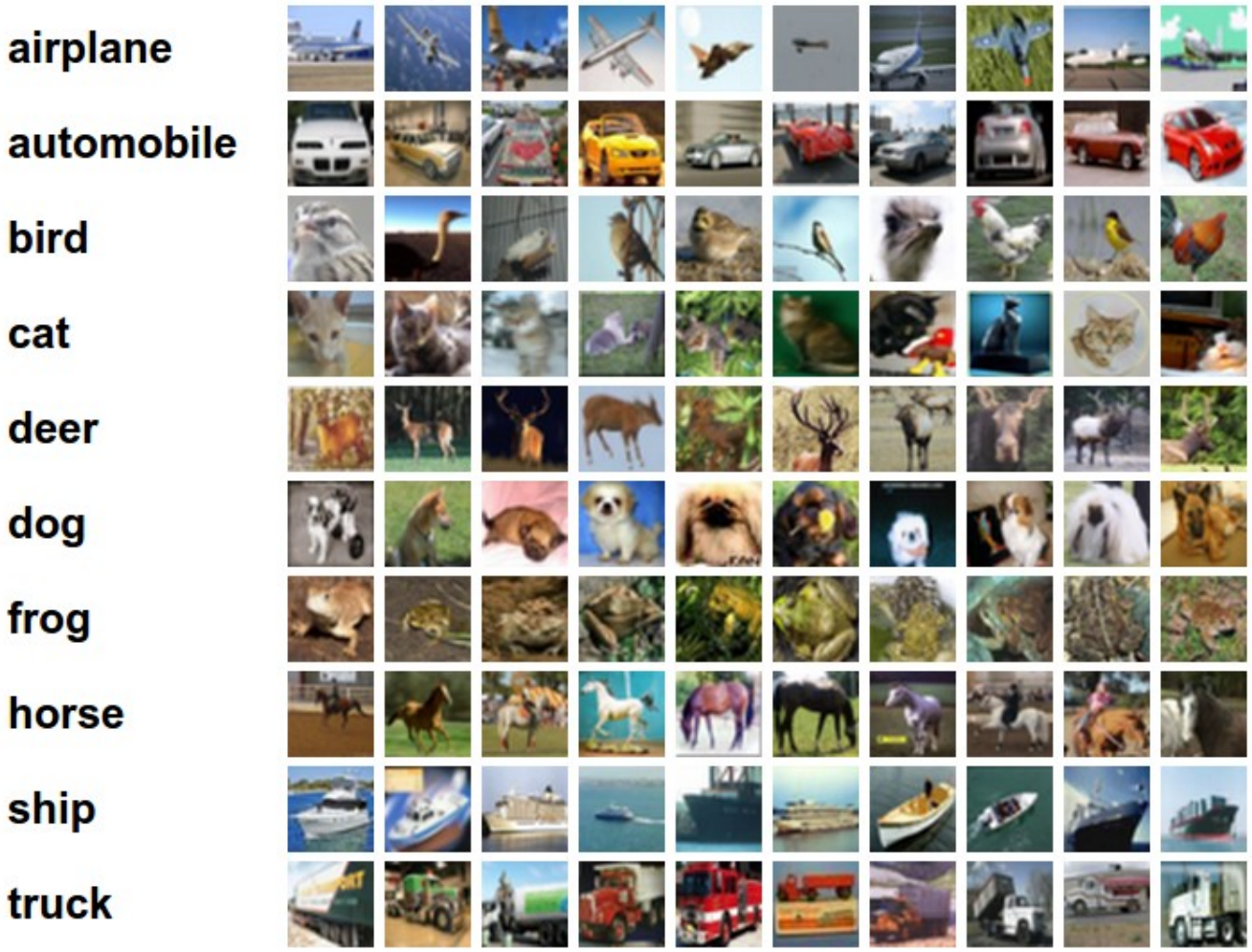
Interpreting a Linear Classifier



$$f(x_i, W, b) = Wx_i + b$$

[32x32x3]
array of numbers 0...1
(3072 numbers total)

CIFAR-10



Example dataset: **CIFAR-10**
10 labels
50,000 training images
each image is **32x32x3**
10,000 test images.

Parametric approach (flatten data)

image parameters

f(**x**,**W**)

10 numbers, indicating class scores

[32x32x3]
array of numbers 0...1
(3072 numbers total)

CIFAR Dataset: 50,000 Train; 10,000 Test

Let's also look at how we might implement the classifier in code.

First, let's load the CIFAR-10 data into memory as 4 arrays: the training data/labels and the test data/labels. In the code below, Xtr (of size 50,000 x 32 x 32 x 3) holds all the images in the training set, and a corresponding 1-dimensional array Ytr (of length 50,000) holds the training labels (from 0 to 9)

```
Xtr, Ytr, Xte, Yte = load_CIFAR10('data/cifar10/') # a magic function we provide
# flatten out all images to be one-dimensional
Xtr_rows = Xtr.reshape(Xtr.shape[0], 32 * 32 * 3) # Xtr_rows becomes 50000 x 3072
Xte_rows = Xte.reshape(Xte.shape[0], 32 * 32 * 3) # Xte_rows becomes 10000 x 3072
```

Parametric approach: Linear classifier

$$f(x, W) = Wx$$

[32x32x3]
array of numbers 0...1

10 numbers, indicating class scores

STEP1: Calculate the perpendicular distance between the example and each class hyperplane

Parametric approach: Linear classifier

$$f(x, W) = Wx$$

3072x1

+ bias. #10 values

10x1

10x3072

[32x32x3]
array of numbers 0...1

10 numbers, indicating class scores

parameters, or “weights”

Parametric approach: Linear classifier

[32x32x3]
array of numbers 0...1

$$f(x, W) = Wx + b$$

10x1 10x3072 3072x1 (+b) 10x1

10 numbers, indicating class scores

parameters, or “weights”

STEP1: Calculate the perpendicular distance between the example and each class hyperplane

Simplify to 3 classes

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

STEP1: Calculate the perpendicula distance between the example and class hyperplane

STEP1: Calculate the perpendicular distance between the example and class hyperplane

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Flatten image (32 X 32 X 3) to 4 values

stretch pixels into single column

input image

0.2	-0.5	0.1	2.0
1.5	1.3	2.1	0.0
0	0.25	0.2	-0.3

W

56
231
24
2

x_i

+

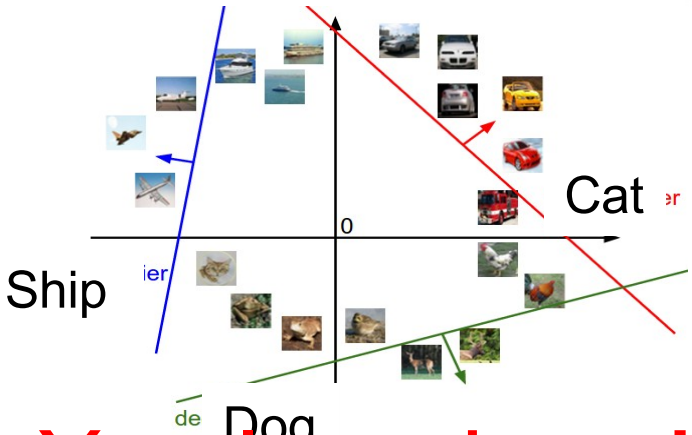
1.1
3.2
-1.2

b

→

-96.8	cat score
437.9	dog score
61.95	ship score

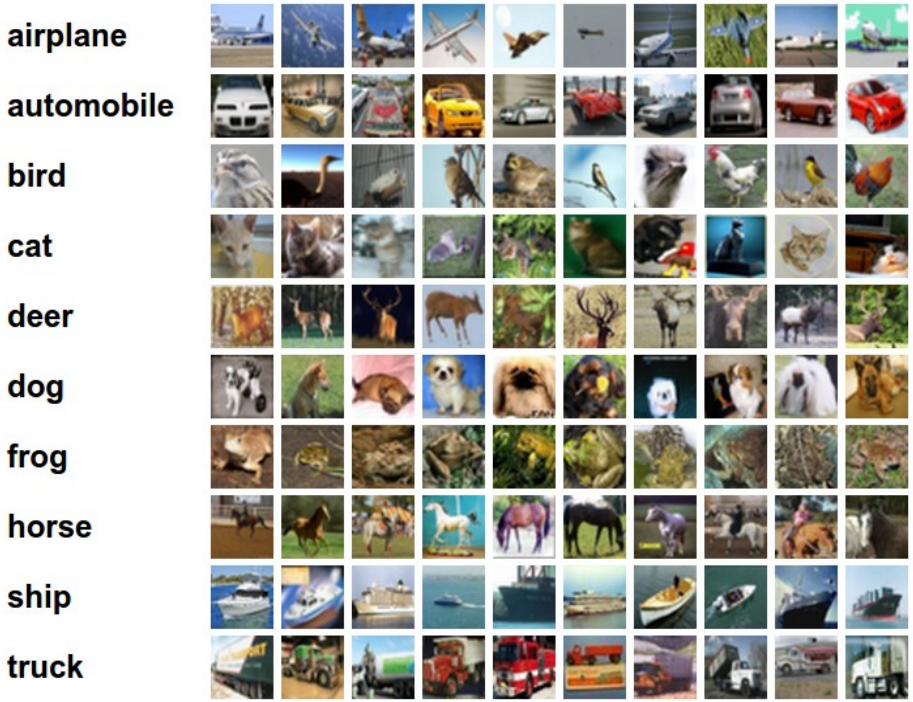
$f(x_i; W, b)$



Where is this example?

X--- I am here! See me?

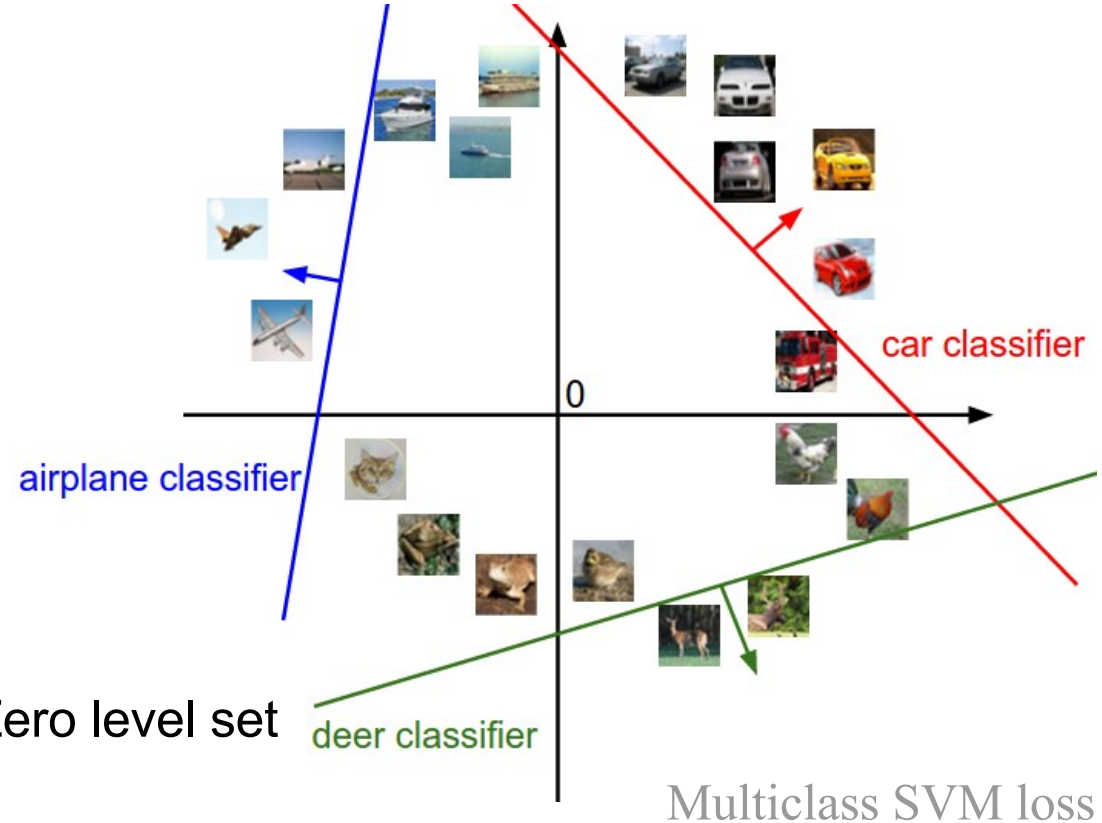
Interpreting a Linear Classifier



$$f(x_i, W, b) = Wx_i + b$$

Example trained weights of a linear classifier trained on CIFAR-10:
Visualize the weight vectors

Interpreting a Linear Classifier



Zero level set

$$f(x_i, W, b) = Wx_i + b$$

[32x32x3]
array of numbers 0...1
(3072 numbers total)

Multiclass SVM loss formulation:
Weston Watkins 1999
One vs. All
Structured SVM
Softmax

Given a random model with 10 classes

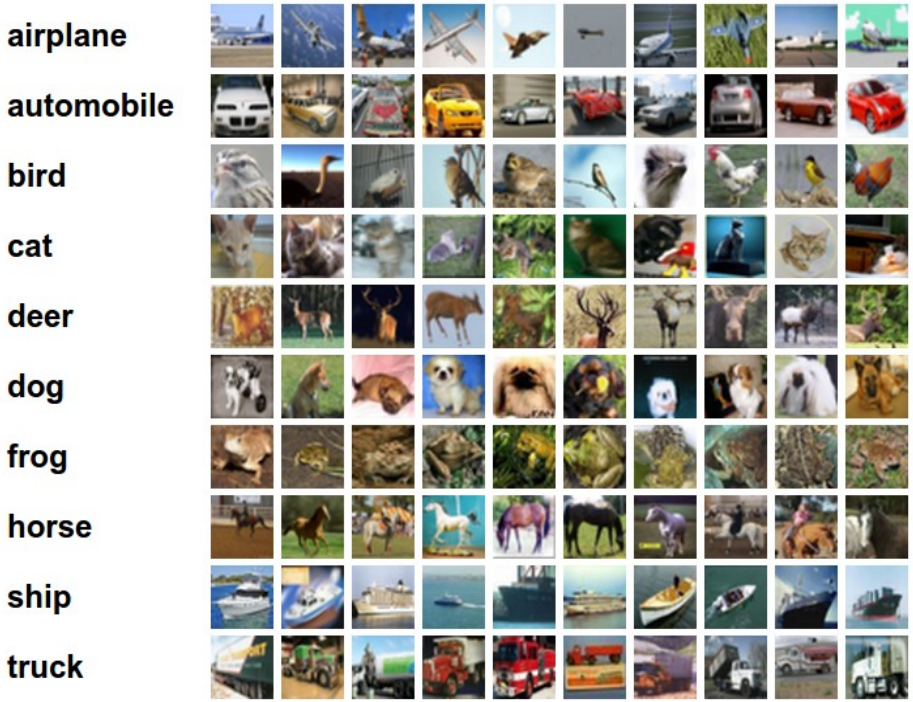
So far: We defined a (linear) **score function**: $f(x_i, W, b) = Wx_i + b$

Example class scores for 3 images, with a random W:



airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

Interpreting a Linear Classifier



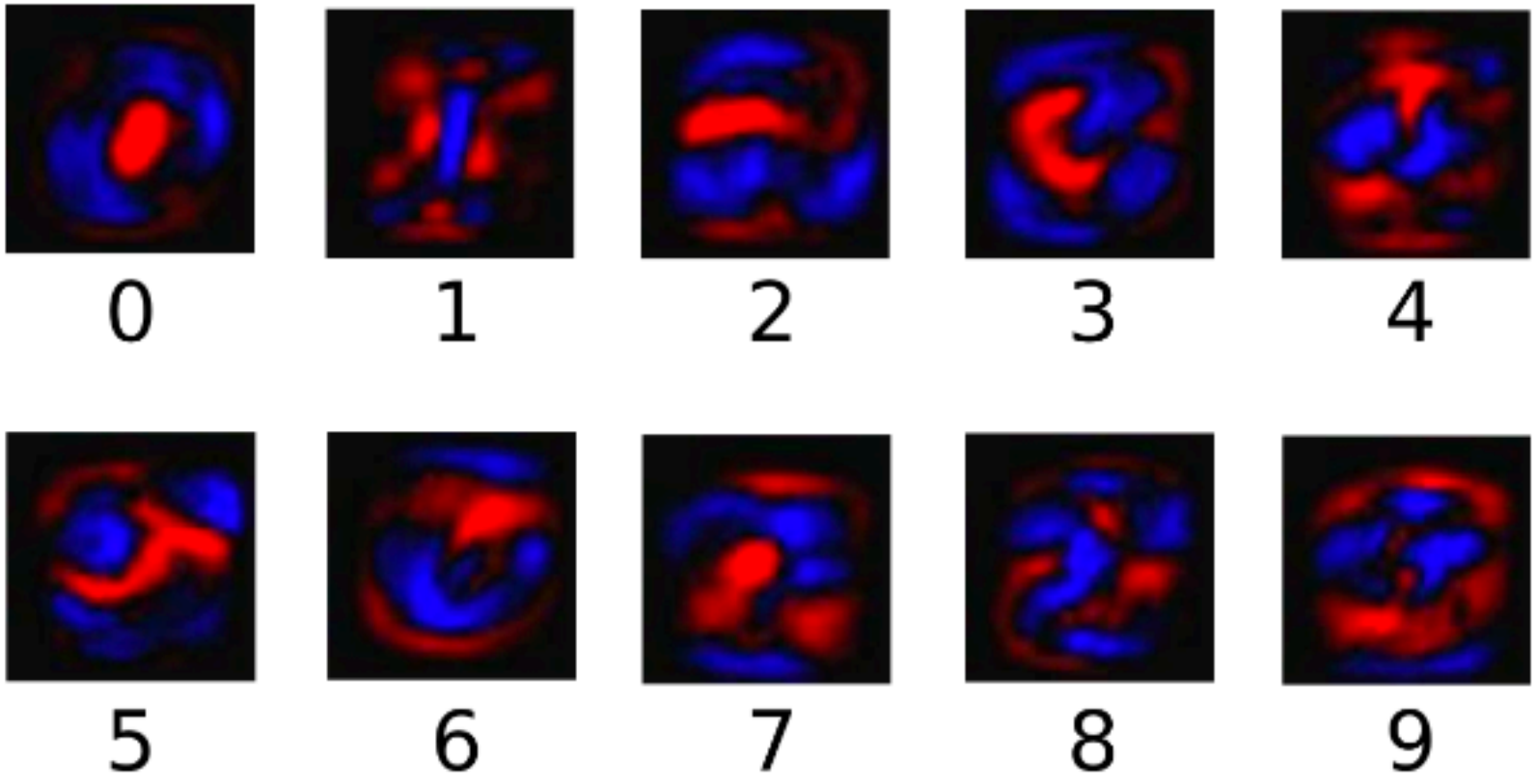
$$f(x_i, W, b) = Wx_i + b$$

Example trained weights of a linear classifier trained on CIFAR-10:
Visualize the weight vectors

Logistic Regression Weights for MNIST

- **The weights one model learned for each of these classes.**
  - Red represents negative weights
  - Blue represents positive weights

Hand written digit recognition



https://www.tensorflow.org/tutorials/mnist/beginners/

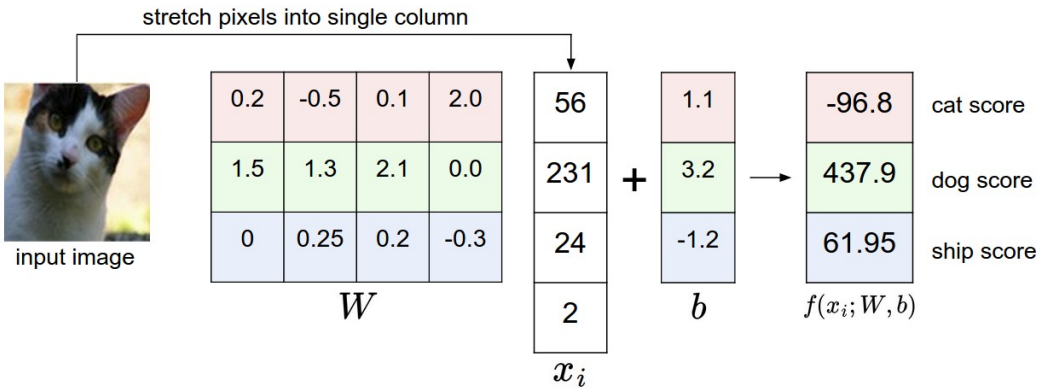
Linear classifier

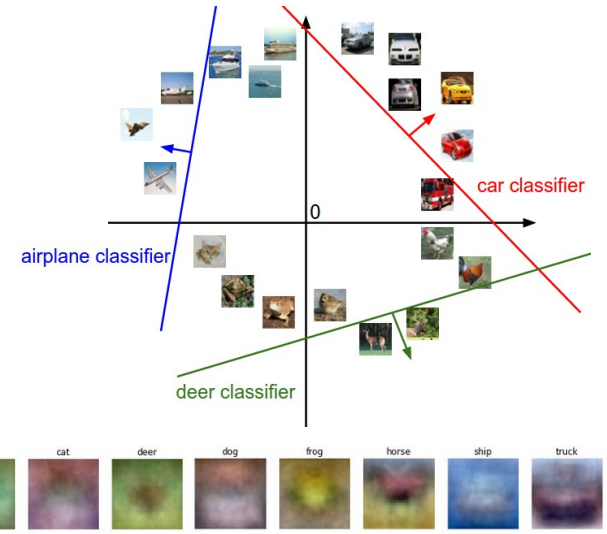
[32x32x3]
array of numbers 0...1
(3072 numbers total)

image parameters

$f(\mathbf{x}, \mathbf{W})$

10 numbers, indicating class scores





Learn LR: Loss function/Optimization



airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data.
2. Come up with a way of efficiently finding the parameters that minimize the loss function. **(optimization)**

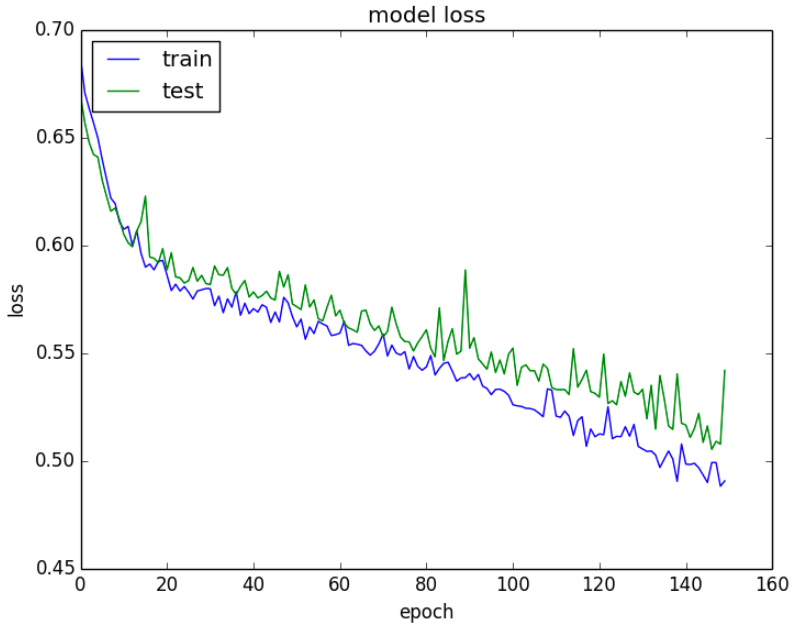
Outline

- Introduction
- Binomial logistic regression
- Multinomial Logistic regression
- Linear Classifier via separating hyperplane
- Multinomial Logistic regression Classifier (loss)
- Linear classifier demo
- Learning Softmax Classifiers via optimization
  - Implementation (SoftMax Classifier)
- Regressions in matrix terms and in graph terms
- Summary

Track Loss as a proxy for how training is going

- So far we have focused on classification based on perpendicular distances. Here we extend this probabilities and track the loss during training of the true class associated with each example. The predicted probability should be 1 it not we incur a loss.
- We track the loss over the true class over each epoch of training to see our progress. Gradient looks at all classes.

$$L_i = -\log P(Y = y_i | X = x_i)$$



Logistic Regression

log loss for binary versus cross-entropy for K-classes

	Binomial LR	Multinomial LR
	$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}log\left(\hat{p}^{(i)}\right) + \left(1-y^{(i)}\right)log\left(1-\hat{p}^{(i)}\right)\right]$	$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$
Gradient	$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma\left(\theta^T\cdot\mathbf{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$	$\nabla_{\theta_k} J(\Theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{p}_k^{(i)} - y_k^{(i)}\right)\mathbf{x}^{(i)}$
Class	$\hat{y} = \begin{cases} 0 \text{ if } \hat{p} < 0.5, \\ 1 \text{ if } \hat{p} \geq 0.5. \end{cases}$	$\hat{y} = \underset{k}{\text{argmax}}\ \sigma(\mathbf{s}(\mathbf{x}))_k = \underset{k}{\text{argmax}}\ s_k(\mathbf{x}) = \underset{k}{\text{argmax}}\left(\left(\theta^{(k)}\right)^T\cdot\mathbf{x}\right)$
Prob	$\hat{p} = h_\theta(\mathbf{x}) = \sigma\left(\theta^T\cdot\mathbf{x}\right)$	$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K}\exp\left(s_j(\mathbf{x})\right)}$ • K is the number of classes. • $\mathbf{s}(\mathbf{x})$ is a vector containing the scores of each class for the instance $\mathbf{x}$. • $\sigma(\mathbf{s}(\mathbf{x}))_k$ is the estimated probability that the instance $\mathbf{x}$ belongs to class k given the scores of each class for that instance.

Calculate probability $\widehat{p_k}$ Pr(Class = k|X)

1. **STEP1: Calculate the perpendicular distance between the example and each class hyperplane, denoted as $s_k(x)$**
2. **Take the exp($s_k(x)$)**
3. **Calculate the relative probability $\widehat{p_k}$**

$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathbf{x})\right)}$$

- K is the number of classes.
- $\mathbf{s}(\mathbf{x})$ is a vector containing the scores of each class for the instance $\mathbf{x}$.
- $\sigma(\mathbf{s}(\mathbf{x}))_k$ is the estimated probability that the instance $\mathbf{x}$ belongs to class k given the scores of each class for that instance.

Just like the Logistic Regression classifier, the Softmax Regression classifier predicts the class with the highest estimated probability (which is simply the class with the highest score), as shown in Equation 4-21.

Equation 4-21. Softmax Regression classifier prediction

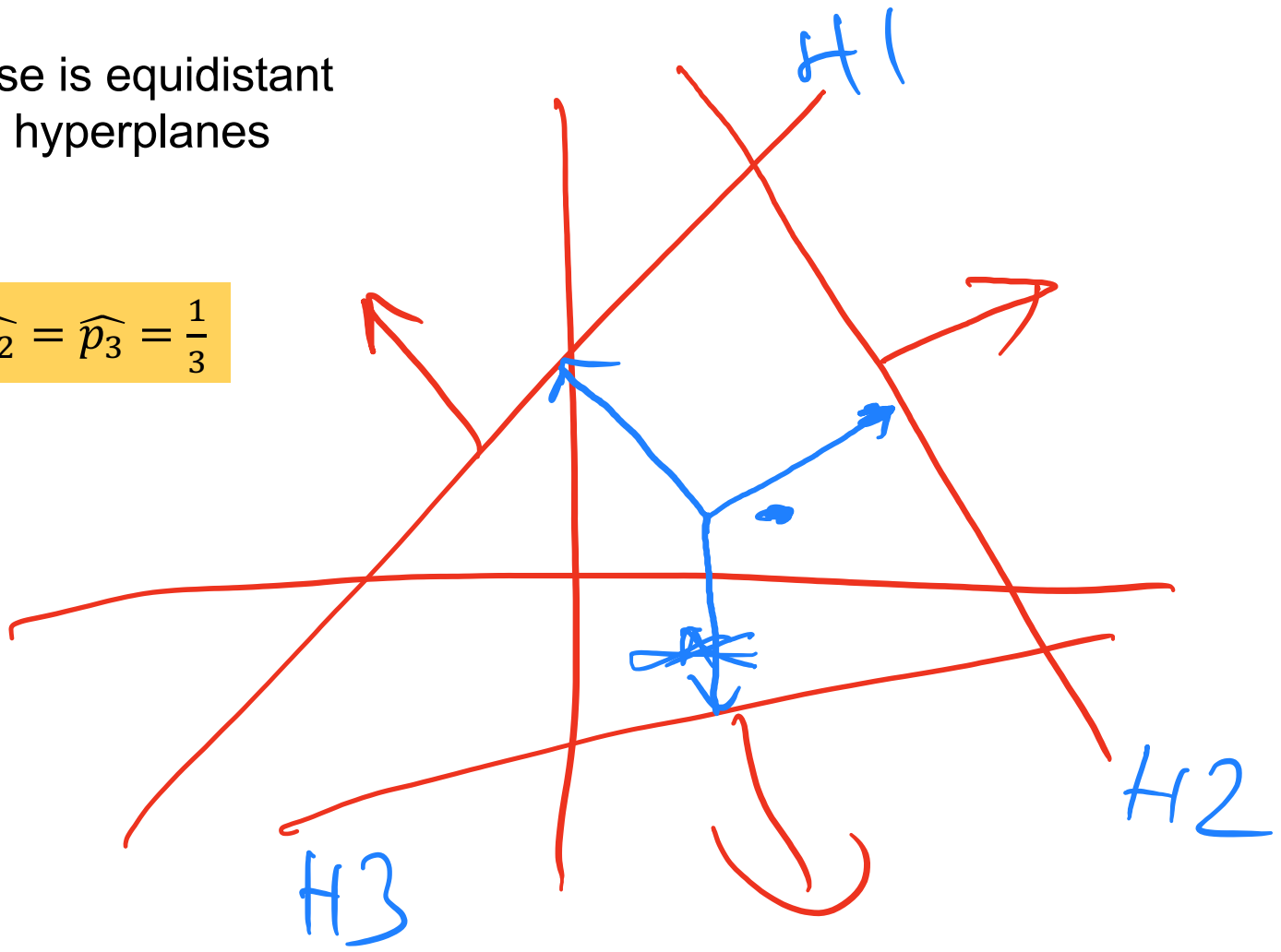
$$\hat{y} = \underset{k}{\operatorname{argmax}}\ \sigma(\mathbf{s}(\mathbf{x}))_k = \underset{k}{\operatorname{argmax}}\ s_k(\mathbf{x}) = \underset{k}{\operatorname{argmax}}\ \left(\left(\theta^{(k)}\right)^T \cdot \mathbf{x}\right)$$

- The *argmax* operator returns the value of a variable that maximizes a function. In this equation, it returns the value of k that maximizes the estimated probability $\sigma(\mathbf{s}(\mathbf{x}))_k$.

Extreme example: Test case in the centre of the negative zones of all classes

Test case is equidistant from all hyperplanes

$$\widehat{p_1} = \widehat{p_2} = \widehat{p_3} = \frac{1}{3}$$



Maximize (conditional) likelihood Pr(Class|X; W)
Minimize negative log likelihood Pr(Class|X; W)

Maximize Pr(Y=y_i|X; W) # Maximize (conditional) likelihood
Mimimize -Log(Pr(Y=y_i |X_i; W) #LOSS

cat **3.2**

car 5.1

frog -1.7

scores = unnormalized log probabilities of the classes.

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ where $$s = f(x_i; W)$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$L_i = -\log P(Y = y_i|X = x_i)$$

Loss for each example to an overall sense of progress during training

Compute loss for the target class only
Use this a tracking metric to track Loss during training

cat **3.2**

car 5.1

frog -1.7

scores = unnormalized log probabilities of the classes.

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

where

$$s = f(x_i; W)$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$L_i = -\log P(Y = y_i|X = x_i)$$

---

in summary:

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

Loss for each example to an overall sense of progress during training

cat	**3.2**
car	5.1
frog	-1.7

Compute loss for the target class only
Use this a tracking metric to track Loss during training

NOTE: Gradient calculation is related but a little different; it focuses on all classes for each example and not just the true actual class

$$L_i = -\log P(Y = y_i | X = x_i)$$

in summary: $$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

Example: PerpDist → Probabilites

Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

	unnormalized log probabilities	exp	unnormalized probabilities	normalize	probabilities
cat	**3.2**	→	**24.5**	→	**0.13**
car	5.1	→	164.0	→	0.87
frog	-1.7	→	0.18	→	0.00

Numerical stability for SoftMax function

Shift the logit values so the highest value is zero

Practical issues: Numeric stability. When you're writing code for computing the Softmax function in practice, the intermediate terms $e^{f_{y_i}}$ and $\sum_j e^{f_j}$ may be very large due to the exponentials. Dividing large numbers can be numerically unstable, so it is important to use a normalization trick. Notice that if we multiply the top and bottom of the fraction by a constant C and push it into the sum, we get the following (mathematically equivalent) expression:

$$\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} = \frac{Ce^{f_{y_i}}}{C\sum_j e^{f_j}} = \frac{e^{f_{y_i}+\log C}}{\sum_j e^{f_j+\log C}}$$

Set C = – Max(WX)

We are free to choose the value of C. This will not change any of the results, but we can use this value to improve the numerical stability of the computation. A common choice for C is to set $\log C = -\max_j f_j$. This simply states that we should shift the values inside the vector f so that the highest value is zero. In code:

```
f = np.array([123, 456, 789]) # example with 3 classes and each having large scores
p = np.exp(f) / np.sum(np.exp(f)) # Bad: Numeric problem, potential blowup

# instead: first shift the values of f so that the highest number is 0:
f -= np.max(f) # f becomes [-666, -333, 0]
p = np.exp(f) / np.sum(np.exp(f)) # safe to do, gives the correct answer
```

SoftMax Classifiers provide a probabilistic output
Change loss to cross-entropy loss function

It turns out that the SVM is one of two commonly seen classifiers. The other popular choice is the **Softmax classifier**, which has a different loss function. If you've heard of the binary Logistic Regression classifier before, the Softmax classifier is its generalization to multiple classes. Unlike the SVM which treats the outputs $f(x_i, W)$ as (uncalibrated and possibly difficult to interpret) scores for each class, the Softmax classifier gives a slightly more intuitive output (normalized class probabilities) and also has a probabilistic interpretation that we will describe shortly. In the Softmax classifier, the function mapping $f(x_i; W) = Wx_i$ stays unchanged, but we now interpret these scores as the unnormalized log probabilities for each class and replace the *hinge loss* with a **cross-entropy loss** that has the form:

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \qquad \text{or equivalently} \qquad L_i = -f_{y_i} + \log\sum_j e^{f_j}$$

where we are using the notation f_j to mean the j-th element of the vector of class scores f. As before, the full loss for the dataset is the mean of L_i over all training examples together with a regularization term $R(W)$. The function $f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$ is called the **softmax function**: It takes a vector of arbitrary real-valued scores (in z) and squashes it to a vector of values between zero and one that sum to one. The full cross-entropy loss that involves the softmax function might look scary if you're seeing it for the first time but it is relatively easy to motivate.

http://cs231n.github.io/linear-classify/#softmax

SoftMax Classifier: Information Theory View

Information theory view. The *cross-entropy* between a "true" distribution p and an estimated distribution q is defined as:

$$H(p,q) = -\sum_x p(x) \log q(x)$$

The Softmax classifier is hence minimizing the cross-entropy between the estimated class probabilities ($q = e^{f_{y_i}} / \sum_j e^{f_j}$ as seen above) and the "true" distribution, which in this interpretation is the distribution where all probability mass is on the correct class (i.e. $p = [0, \ldots 1, \ldots, 0]$ contains a single 1 at the y_i -th position.). Moreover, since the cross-entropy can be written in terms of entropy and the Kullback-Leibler divergence as $H(p,q) = H(p) + D_{KL}(p||q)$, and the entropy of the delta function p is zero, this is also equivalent to minimizing the KL divergence between the two distributions (a measure of distance). In other words, the cross-entropy objective *wants* the predicted distribution to have all of its mass on the correct answer.

SoftMax Classifier: Probabilistic view

For each example i:

- *Maximum likelihood:* $P(y_i|x_i;W)$
- *Maximum log likelihood:* $log[P(y_i|x_i;W)]$
- Minimize $-log[P(y_i|x_i;W)]$

Probabilistic interpretation. Looking at the expression,

$$P(y_i \mid x_i; W) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}$$

can be interpreted as the (normalized) probability assigned to the correct label y_i given the image x_i and parameterized by W. To see this, remember that the Softmax classifier interprets the scores inside the output vector f as the unnormalized log probabilities. Exponentiating these quantities therefore gives the (unnormalized) probabilities, and the division performs the normalization so that the probabilities sum to one. In the probabilistic interpretation, we are therefore minimizing the negative log likelihood of the correct class, which can be interpreted as performing *Maximum Likelihood Estimation* (MLE). A nice feature of this view is that we can now also interpret the regularization term $R(W)$ in the full loss function as coming from a Gaussian prior over the weight matrix W, where instead of MLE we are performing the *Maximum a posteriori* (MAP) estimation. We mention these interpretations to help your intuitions, but the full details of this derivation are beyond the scope of this class.

Focus on the class of the i-th example and ignore all other classes

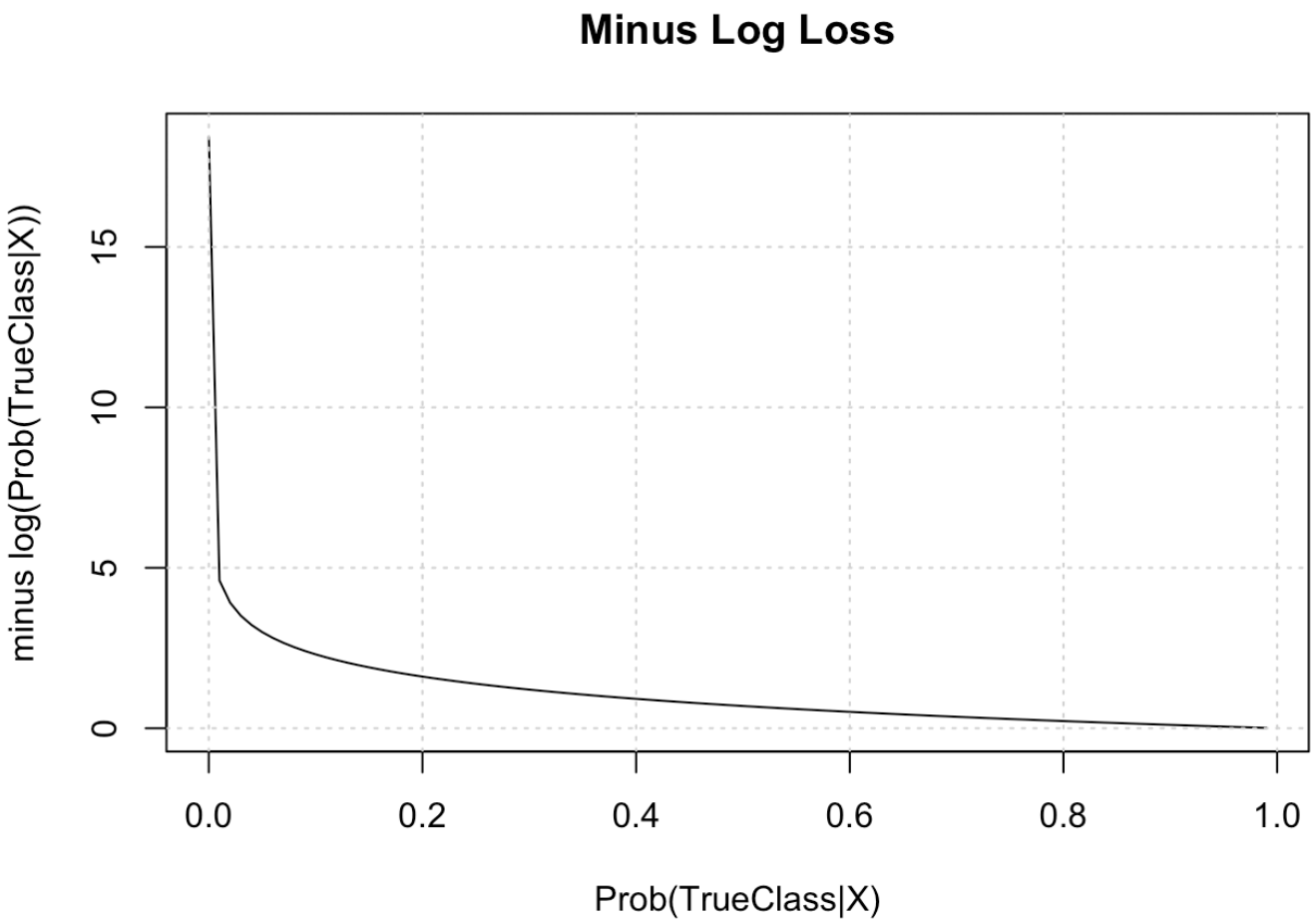
- p(x) X log(q(x))
-1 X log(q(x))

Minus Log loss of the Pr(actual class|X)

Loss for the class label only And ignore all other classes

So if predictions for the actual class is one (i.e., Pr(actual class|X)=1) then the loss will be zero

Range of log loss is [0, ∞]



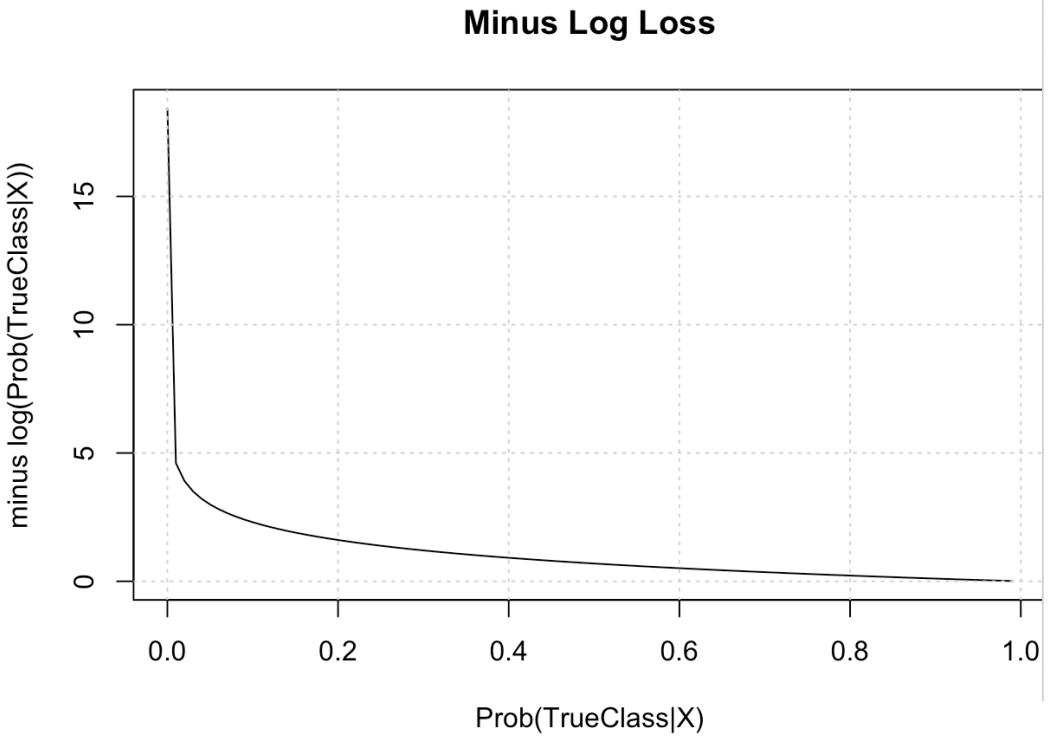

```
plot(seq(0.00000001, 0.99999999, 0.01), -log(seq(0.00000001, 0.99999999, 0.01)), type="l",
main="Minus Log Loss", xlab="Prob(TrueClass|X)", ylab="minus log(Prob(TrueClass|X))")
grid()
```

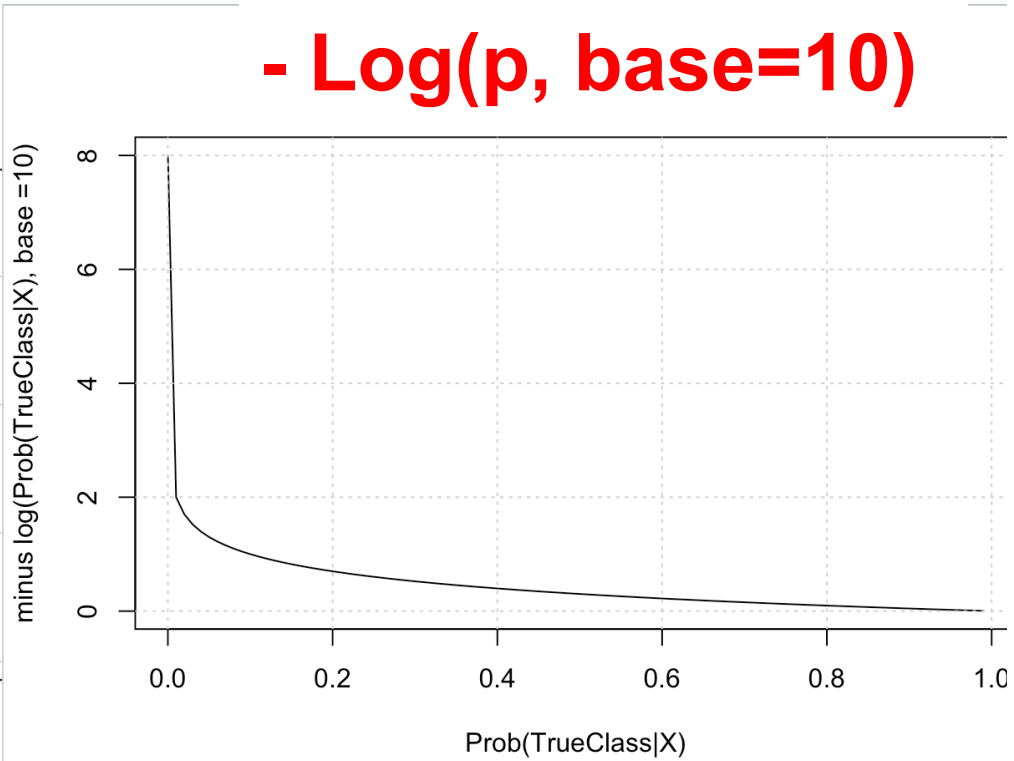
Loss for the class label only
And ignore all other classes

So if predictions for all actual classed is one (i.e., Pr(actual class|X)=1) then the loss will be zero

Minus Log loss of the Pr(actual class|X)

- Log(p, base=10)





Log(p, base = e)
Log(p, base =10)

- plot(seq(0.00000001, 0.99999999, 0.01), -log(seq(0.00000001, 0.99999999, 0.01)), type="l", main="Minus Log Loss", xlab="Prob(TrueClass|X)", ylab="minus

Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

Compute loss for the target class for tracking purposes

	unnormalized log probabilities	exp	unnormalized probabilities	normalize	probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

Base 10

L_i = -log(0.13)

= **0.89**

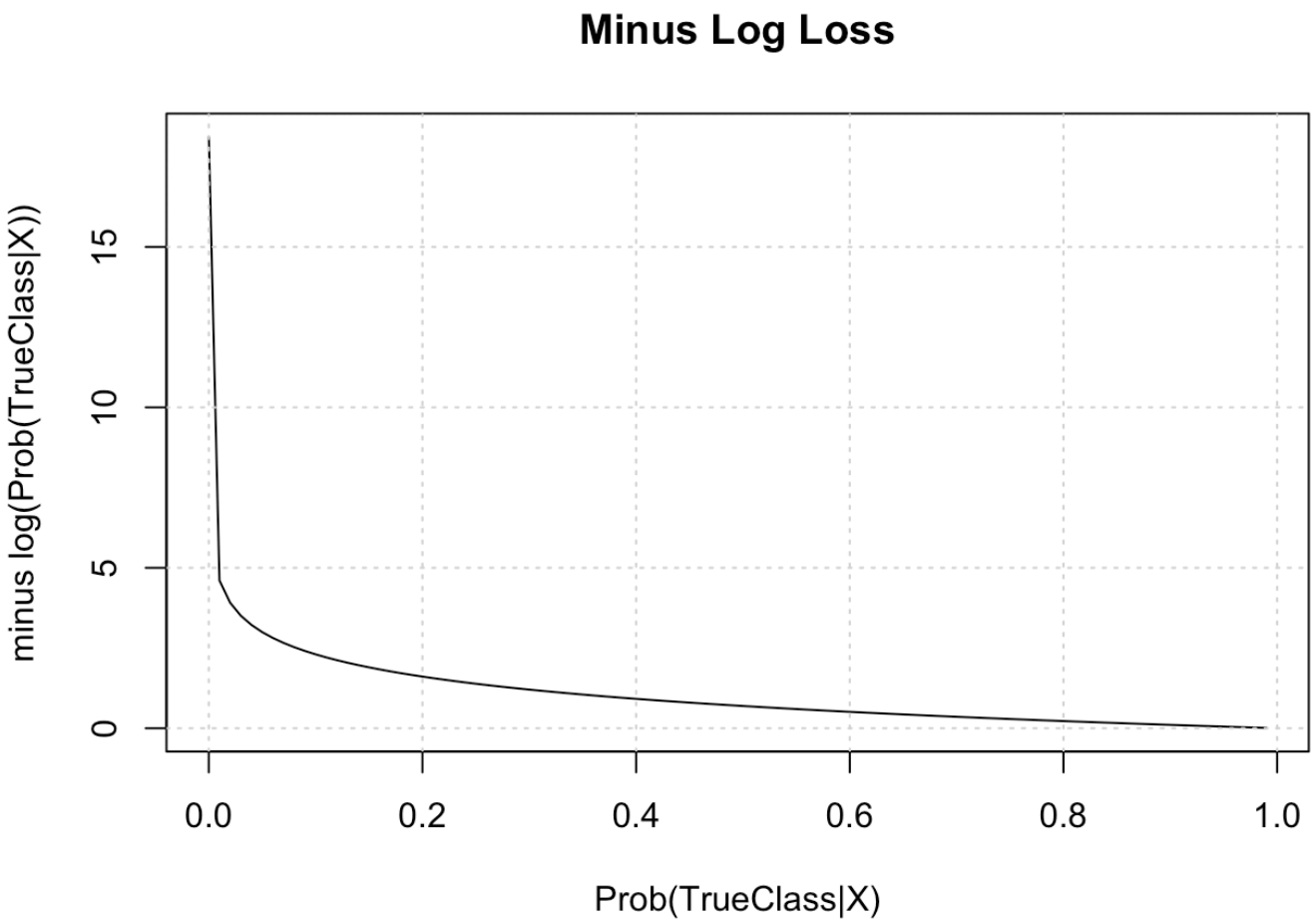
Loss for the class label only
And ignore all other classes

Minus Log loss of the Pr(actual class|X)

Loss for the class label only And ignore all other classes

So if predictions for the actual class is one (i.e., Pr(actual class|X)=1) then the loss will be zero

Range of log loss is [0, ∞]




```
plot(seq(0.00000001, 0.99999999, 0.01), -log(seq(0.00000001, 0.99999999, 0.01)), type="l",
main="Minus Log Loss", xlab="Prob(TrueClass|X)", ylab="minus log(Prob(TrueClass|X))")
grid()
```

Softmax Classifier (Multinomial Logistic Regression)

Q: What is the minimum and maximum possible loss values for L_i?

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

	unnormalized log probabilities	exp	unnormalized probabilities	normalize	probabilities	
cat	**3.2**	→	**24.5**	→	**0.13**	→ Base 10 L_i = -log(0.13) = **0.89**
car	5.1		164.0		0.87	
frog	-1.7		0.18		0.00	

Softmax Classifier (Multinomial Logistic Regression)

Q: What is the minimum and maximum possible loss values for L_i?

[0, -log(0)]

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

	unnormalized log probabilities	exp	unnormalized probabilities	normalize	probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

Base 10

L_i = -log(0.13)
= **0.89**

Softmax Classifier (Multinomial Logistic Regression)

Usually at initialization W are small numbers, so all scores (aka perpendicular distances) will be ~= 0.
What is the loss?

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

Base 10

	unnormalized log probabilities	exp	unnormalized probabilities	normalize	probabilities	
cat	**3.2**		**24.5**		**0.13**	→ L_i = -log(0.13, 10) = **0.89**
car	5.1		164.0		0.87	
frog	-1.7		0.18		0.00	

Softmax Classifier (Multinomial Logistic Regression)

Usually at initialization W are small numbers, so all scores (aka perpendicular distances) will be ~= 0.
What is the loss?

– Log(1/k)

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

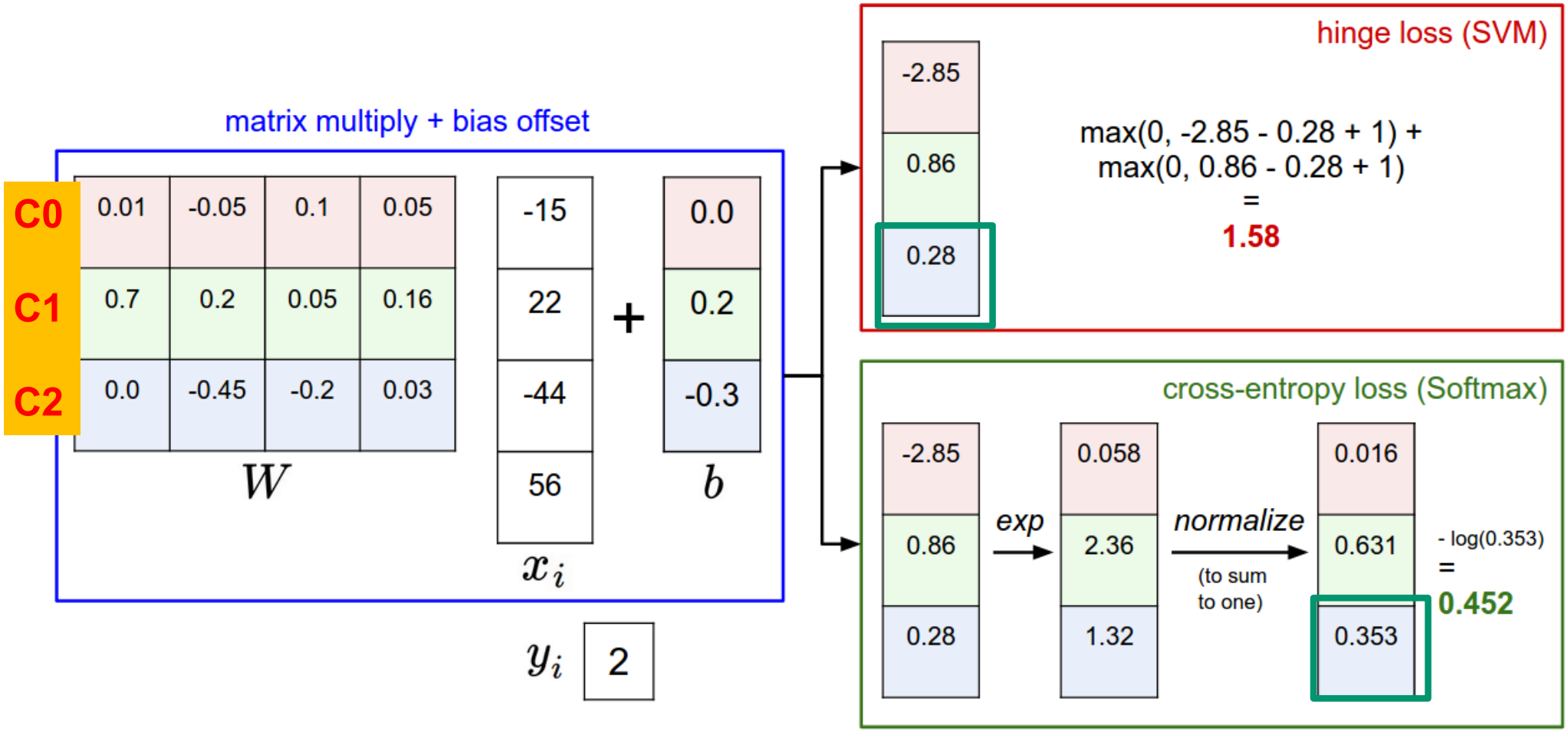
unnormalized probabilities

Base 10

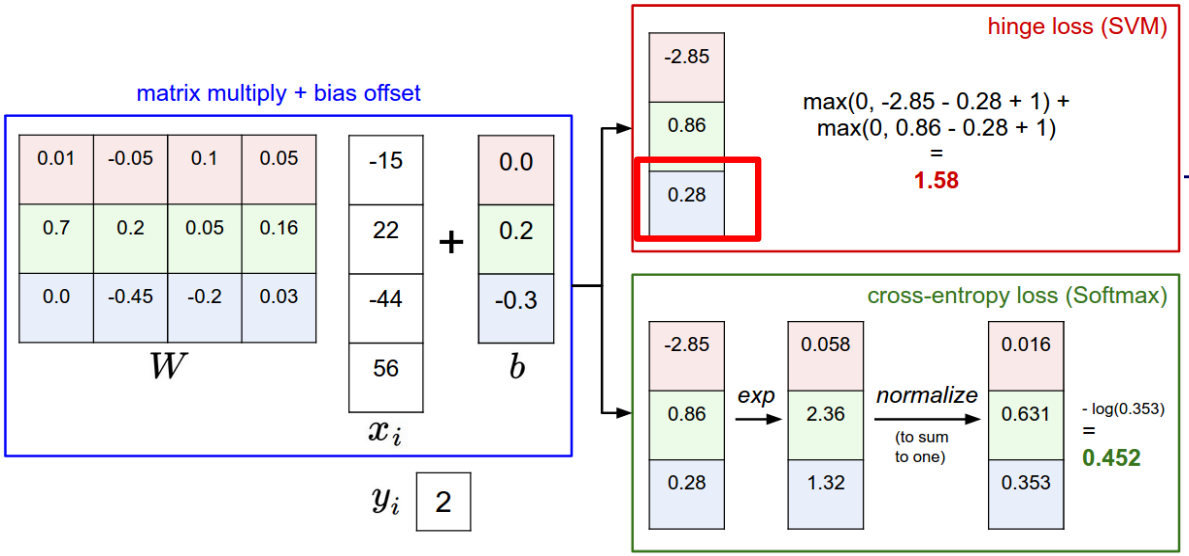
	unnormalized log probabilities	exp	unnormalized probabilities	normalize	probabilities	
cat	**3.2**	→	**24.5**	→	**0.13**	→ L_i = -log(0.13, 10) = **0.89**
car	5.1		164.0		0.87	
frog	-1.7		0.18		0.00	

- For CIFAR-10 loss after init could be
- -log(1/10)
- 2.302585

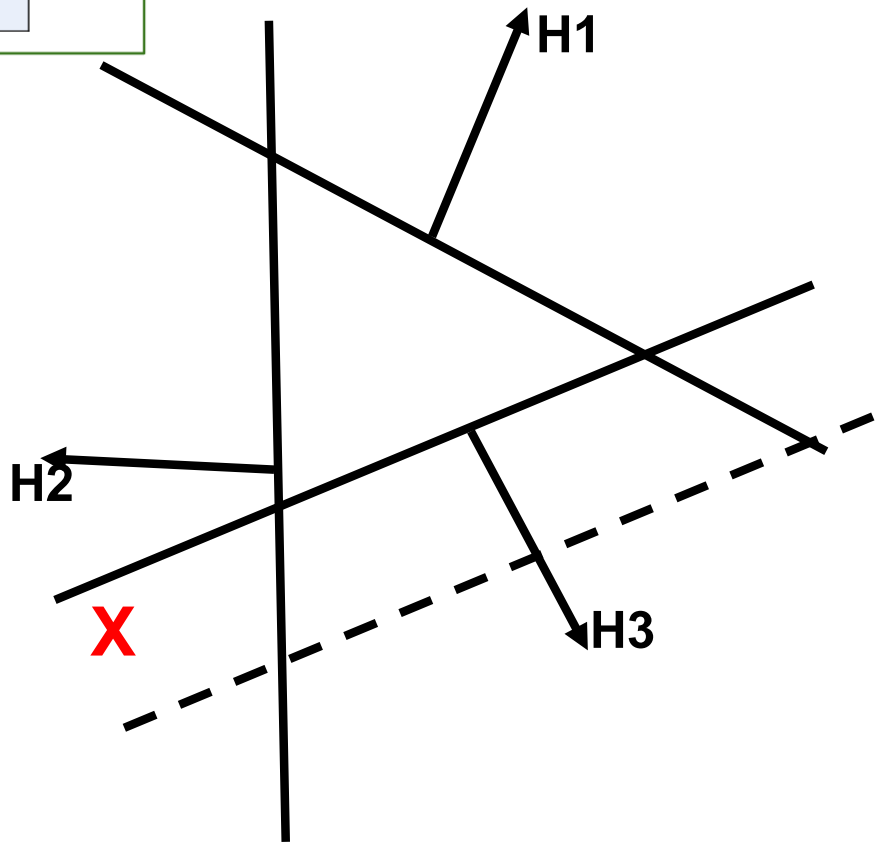
Hinge Loss vs. Cross-entropy Loss



SVM Classifier



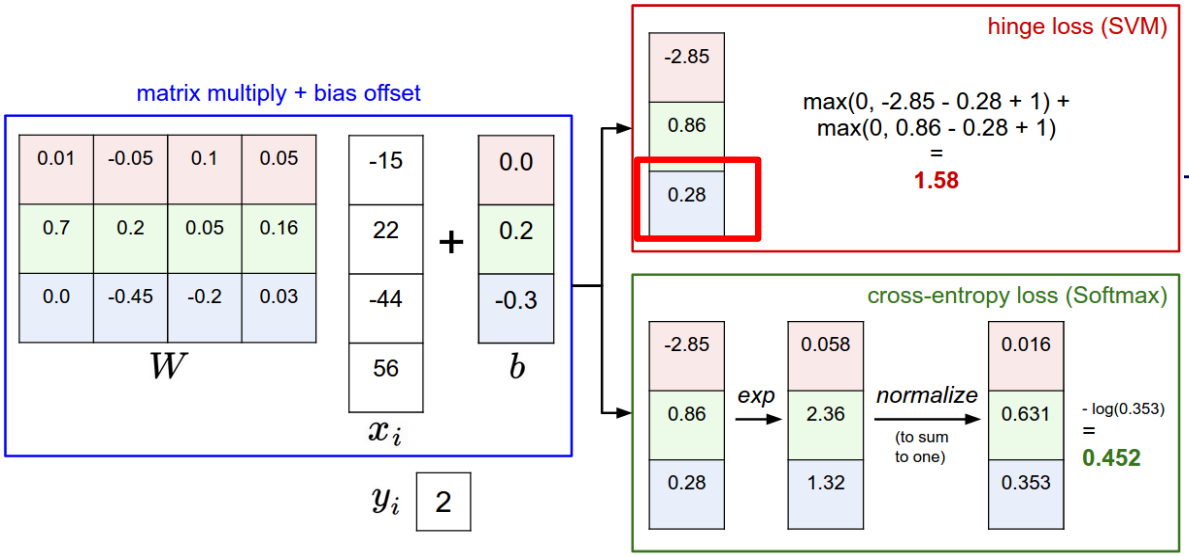
SVM loss demands the test case needs to be actual on the positive side of the true class supporting hyperplane (+1 hyperplane) and in addition the test case can not be on the positive side any other class hyperplane. It there are, the positive perpendicular distance needs to < (the perpendicular distance of the actual class +1)



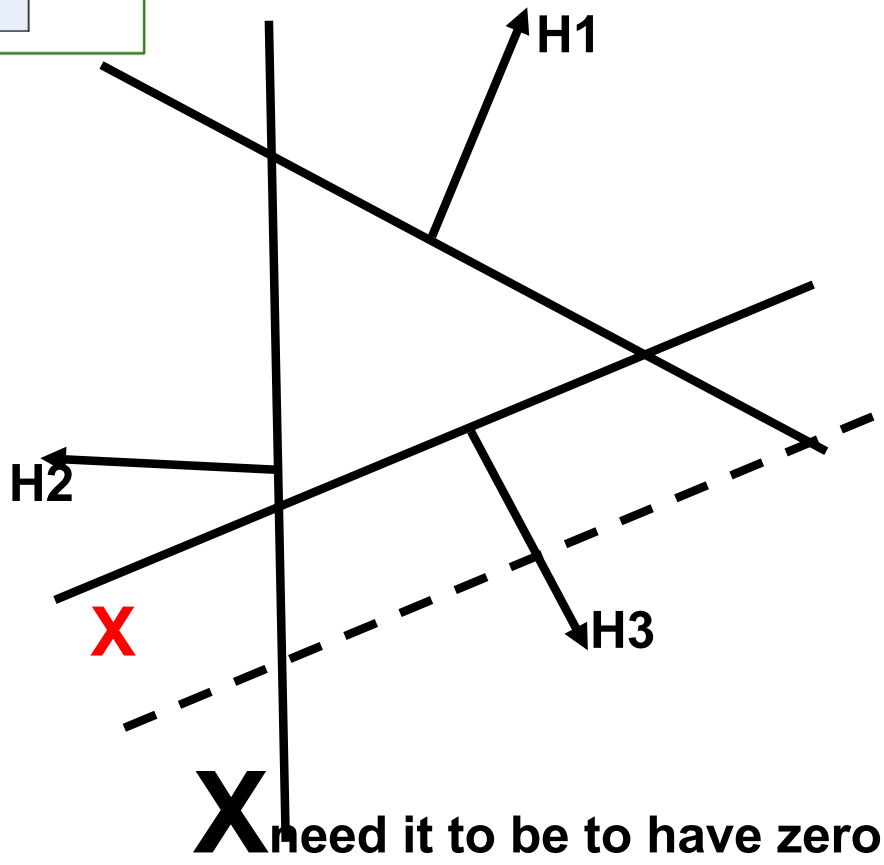
SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

SVM Classifier



SVM loss demands the test case needs to be actual on the positive side of the true class supporting hyperplane (+1 hyperplane) and in addition the test case can not be on the positive side any other class hyperplane. It there are, the positive perpendicular distance needs to < (the perpendicular distance of the actual class +1)



need it to be to have zero loss

SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Linear Classifiers in Summary

Recap

- We have some dataset of (x,y)
- We have a **score function:** $s = f(x; W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function**:

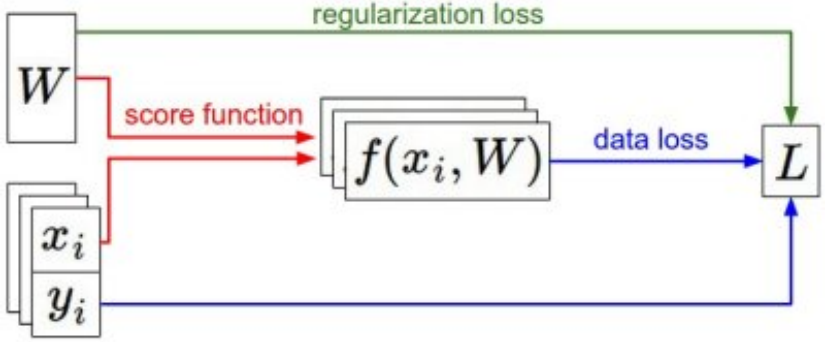
Softmax

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$ Full (regularized) loss

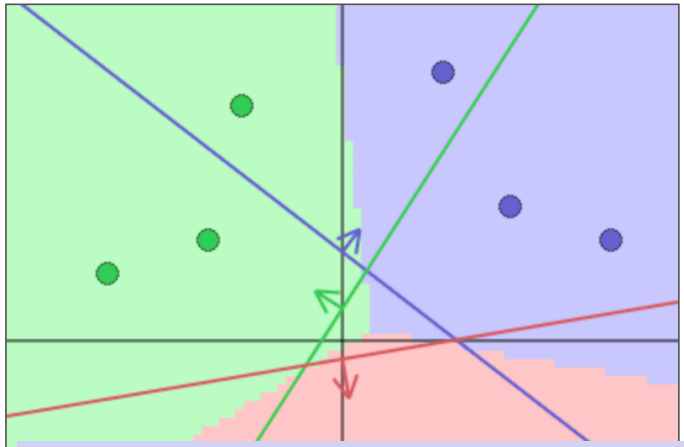


Outline

- Introduction
- Binomial logistic regression
- Multinomial Logistic regression
- Linear Classifier via separating hyperplane
- Multinomial Logistic regression Classifier
- Linear classifier demo
- Learning Softmax Classifiers via optimization
  - Implementation (SoftMax Classifier)
- Regressions in matrix terms and in graph terms
- Summary

Interactive Web Demo time

Datapoints are shown as circles colored by their class (red/gree/blue). The background regions are colored by whichever class is most likely at any point according to the current weights. Each classifier is visualized by a line that indicates its zero score level set. For example, the blue classifier computes scores as $W_{0,0}x_0 + W_{0,1}x_1 + b_0$ and the blue line shows the set of points (x_0, x_1) that give score of zero. The blue arrow draws the vector $(W_{0,0}, W_{0,1})$, which shows the direction of score increase and its length is proportional to how steep the increase is.
Note: you can drag the datapoints.



Parameters W, b are shown below. The value is in bold and its gradient (computed with backprop) is in *red, italic* below. Click the triangles to control the parameters.

W[0,0]	W[0,1]	b[0]
0.52 *-0.07*	**0.68** *-0.04*	**-0.18** *0.01*
W[1,0]	W[1,1]	b[1]
-0.82 *0.08*	**0.52** *-0.02*	**-0.05** *0.01*
W[2,0]	W[2,1]	b[2]
0.20 *-0.01*	**-1.17** *0.06*	**-0.06** *-0.02*

Visualization of the data loss computation. Each row is loss due to one datapoint. The first three columns are the 2D data x_i and the label y_i. The next three columns are the three class scores from each classifier $f(x_i; W, b) = Wx_i + b$ (E.g. s[0] = x[0] * W[0,0] + x[1] * W[0,1] + b[0]). The last column is the data loss for a single example, L_i.

x[0]	x[1]	y	s[0]	s[1]	s[2]	L
0.50	0.40	0	0.35	-0.25	-0.43	0.69
0.80	0.30	0	0.44	-0.55	-0.26	0.63
0.30	0.80	0	0.52	0.12	-0.94	0.64
-0.40	0.30	1	-0.19	0.43	-0.50	0.66
-0.30	0.70	1	0.14	0.56	-0.94	0.63
-0.70	0.20	1	-0.41	0.63	-0.44	0.53
0.70	-0.40	2	-0.08	-0.83	0.54	0.58
0.50	-0.60	2	-0.32	-0.77	0.74	0.45
-0.40	-0.50	2	-0.73	0.01	0.44	0.67
						mean:
						0.61

LOSS

Total data loss: 0.61
Regularization loss: 0.31

Maximize $Pr(Y=y_i|X; W)$ # Maximize (conditional) likelihood; should be 1
Mimimize $-Log(Pr(Y=y_i|X_i; W)$ #LOSS; should be 0.

Adapted from http://vision.stanford.edu/teaching/cs231n/linear-classify-demo/

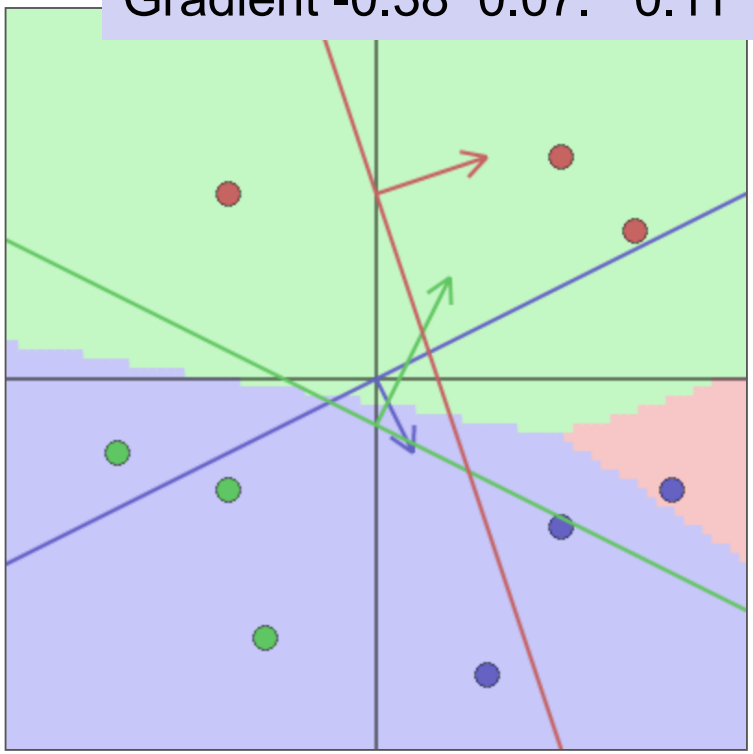
Corrected DEMO here on Dropbox
Please download everything from dropbox to your computer and run locally
https://www.dropbox.com/sh/91ivcvd7rckmfq0/AAABLQz9iIxUR8iM2d8bEdcBa?dl=0

Linear Classifier Demo

blue line shows the set of points (x_0, x_1) that give score of zero. Th
shows t
proporti
Note: yo

parameters.

Hyperplane Purple weights
Weights:1.00. 2.00 0.00
Gradient -0.38 0.07. 0.11



W[0,0]	W[0,1]	b[0]
1.00 *-0.38*	2.00 *0.07*	0.00 *0.11*
W[1,0]	**W[1,1]**	**b[1]**
2.00 *0.51*	-4.00 *-0.58*	0.50 *-0.11*
W[2,0]	**W[2,1]**	**b[2]**
3.00 *0.17*	-1.00 *0.36*	-0.50 *0.00*

Step size: 0.10000

Single parameter update

Start repeated update

Stop repeated update

Randomize parameters

a single example, L_i:

x[0]	x[1]	y	s[0]	s[1]	s[2]	L
0.50	0.40	0	1.30	-0.10	0.60	0.30
0.80	0.30	0	1.40	0.90	1.60	1.70
0.30	0.80	0	1.90	-2.10	-0.40	0.00
-0.40	0.30	1	0.20	-1.50	-2.00	3.20
-0.30	0.70	1	1.10	-2.90	-2.10	6.80
-0.70	0.20	1	-0.30	-1.70	-2.80	2.40
0.70	-0.40	2	-0.10	3.50	2.00	2.50
0.50	-0.60	2	-0.70	3.90	1.60	3.30
-0.40	-0.50	2	-1.40	1.70	-1.20	4.70
						mean:
						2.77

Total data loss: 2.77
Regularization loss: 3.50
Total loss: 6.27

L2 Regularization strength: 0.10000

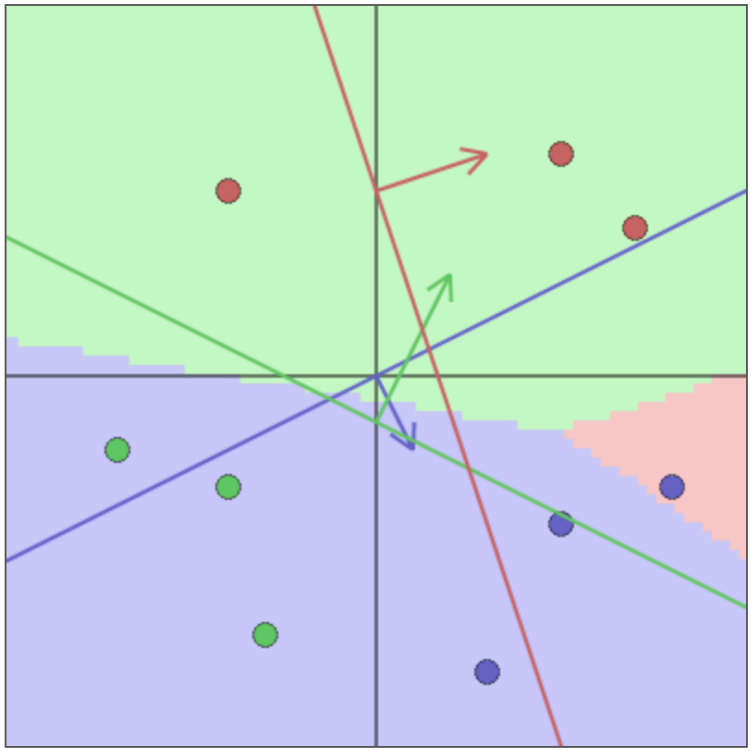
Focus on SoftMax classifier

Multiclass SVM loss formulation:
- (•) Weston Watkins 1999
- ( ) One vs. All
- ( ) Structured SVM
- ( ) Softmax

Linear

Hyperplane Purple training data
X values, target, PerpDists Loss

blue line shows the set of points (x_0, x_1) that give score of zero. The blue arrow draws the vector $(W_{0,0}, W_{0,1})$, which shows the direction of score increase and its length is proportional to how steep the increase is.
Note: you can drag the datapoints.



parameters.

W[0,0]	W[0,1]	b[0]
1.00 *-0.38*	2.00 *0.07*	0.00 *0.11*
W[1,0]	**W[1,1]**	**b[1]**
2.00 *0.51*	-4.00 *-0.58*	0.50 *-0.11*
W[2,0]	**W[2,1]**	**b[2]**
3.00 *0.17*	-1.00 *0.36*	-0.50 *0.00*

Step size: 0.10000

Single parameter update

Start repeated update

Stop repeated update

Randomize parameters

x[0]	x[1]	y	s[0]	s[1]	s[2]	L
0.50	0.40	0	1.30	-0.10	0.60	0.30
0.80	0.30	0	1.40	0.90	1.60	1.70
0.30	0.80	0	1.90	-2.10	-0.40	0.00
-0.40	0.30	1	0.20	-1.50	-2.00	3.20
-0.30	0.70	1	1.10	-2.90	-2.10	6.80
-0.70	0.20	1	-0.30	-1.70	-2.80	2.40
0.70	-0.40	2	-0.10	3.50	2.00	2.50
0.50	-0.60	2	-0.70	3.90	1.60	3.30
-0.40	-0.50	2	-1.40	1.70	-1.20	4.70
						mean:
						2.77

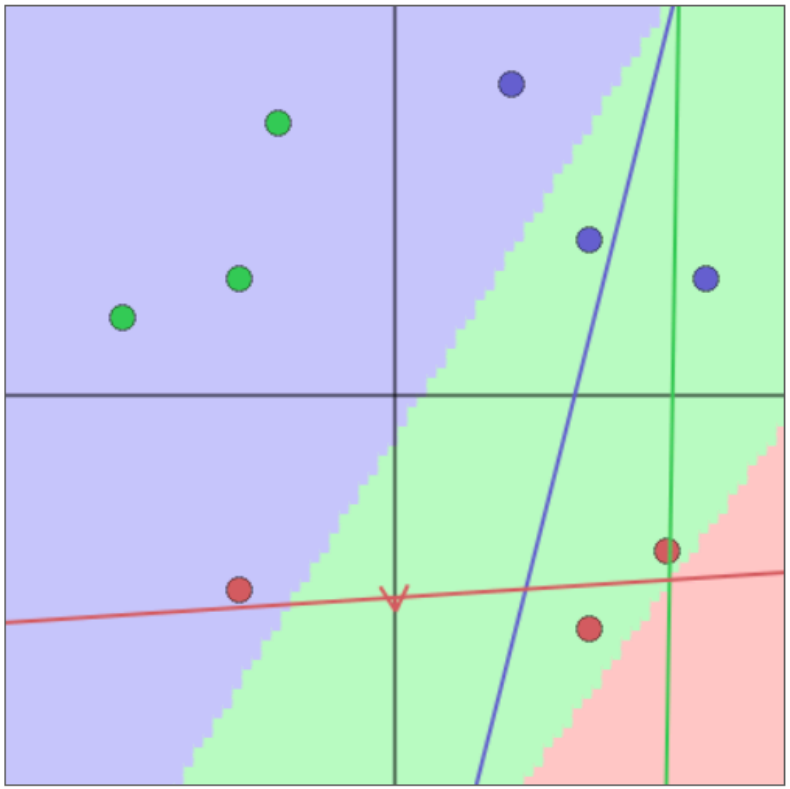
Total data loss: 2.77
Regularization loss: 3.50
Total loss: 6.27

L2 Regularization strength: 0.10000

Focus on SoftMax classifier

Multiclass SVM loss formulation:
- (•) Weston Watkins 1999
- ( ) One vs. All
- ( ) Structured SVM
- ( ) Softmax

shows the direction of score increase and its length is proportional to how steep the increase is.
Note: you can drag the datapoints.



W[1,0]	W[1,1]	b[1]
-0.78 *-0.21*	0.20 *-0.08*	0.36 *0.05*
-0.48 *0.17*	0.01 *-0.08*	0.34 *0.04*
W[2,0]	W[2,1]	b[2]
0.02 *-0.03*	-0.35 *0.16*	-0.18 *-0.10*

Step size: 0.10000

Single parameter update

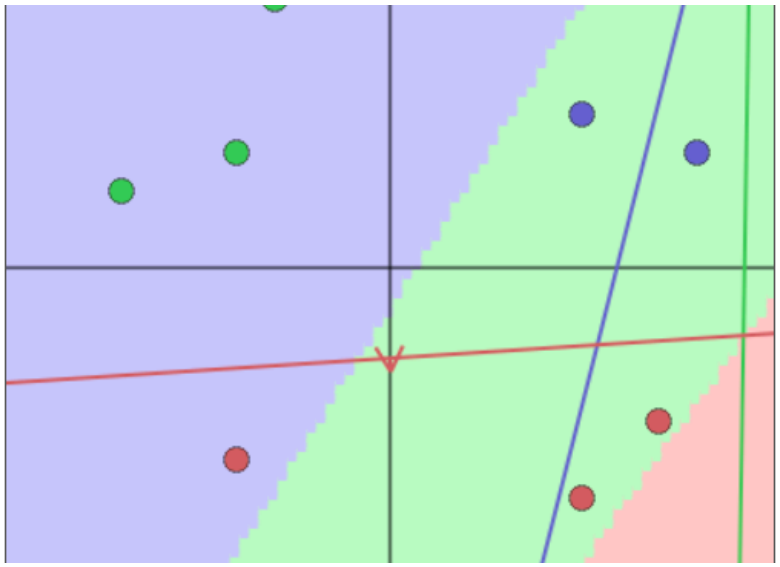
Start repeated update

0.80	0.30	0	-0.20	-0.04	-0.27	1.14
0.30	0.80	0	0.28	0.20	-0.46	0.88
-0.40	0.30	1	0.73	0.53	-0.30	0.98
-0.30	0.70	1	0.73	0.49	-0.44	0.98
-0.70	0.20	1	0.94	0.67	-0.27	0.99
0.70	-0.40	2	-0.26	0.00	-0.03	1.04
0.50	-0.60	2	-0.15	0.10	0.04	1.06
-0.40	-0.50	2	0.57	0.53	-0.02	1.51
						mean:
						1.06

Total data loss: 1.06
Regularization loss: 0.10
Total loss: 1.16

L2 Regularization strength: 0.10000

Change individual coefficients
And see what happens



W[1,0]	W[1,1]	b[1]
-0.48 *0.17*	0.01 *-0.08*	0.44 *0.04*
W[2,0]	W[2,1]	b[2]
0.02 *-0.03*	-0.35 *0.16*	-0.08 *-0.10*

Step size: 0.10000

Single parameter update

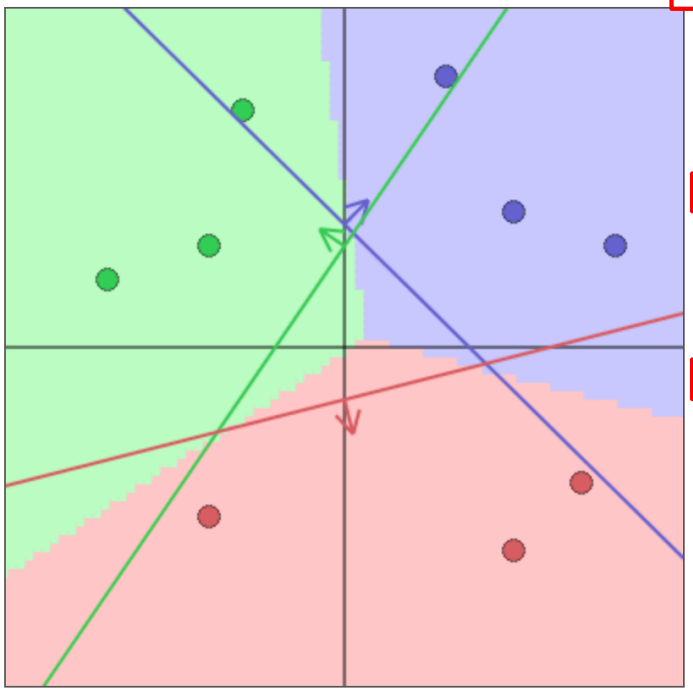
-0.70	0.20	1	1.04	0.77	-0.17	0.99
0.70	-0.40	2	-0.16	0.10	0.07	1.04
0.50	-0.60	2	-0.05	0.20	0.14	1.06
-0.40	-0.50	2	0.67	0.63	0.08	1.51
						mean:
						1.06

Total data loss: 1.06
Regularization loss: 0.10
Total loss: 1.16

L2 Regularization strength: 0.10000

Gradient (redbox); small gradient → shorter arrow

Datapoints are shown as circles colored by their class (red/gree/blue). The background regions are colored by whichever class is most likely at any point according to the current weights. Each classifier is visualized by a line that indicates its zero score level set. For example, the blue classifier computes scores as $W_{0,0}x_0 + W_{0,1}x_1 + b_0$ and the blue line shows the set of points (x_0, x_1) that give score of zero. The blue arrow draws the vector $(W_{0,0}, W_{0,1})$, which shows the direction of score increase and its length is proportional to how steep the increase is.
Note: you can drag the datapoints.



Parameters W, b are shown below. The value is in **bold** and its gradient (computed with backprop) is in *red, italic* below. Click the triangles to control the parameters.

W[0,0]	W[0,1]	b[0]
0.70	**0.72**	**-0.26**
-0.05	*-0.04*	*0.02*
W[1,0]	W[1,1]	b[1]
-0.74	**0.50**	**-0.15**
0.08	*-0.03*	*0.00*
W[2,0]	W[2,1]	b[2]
0.26	**-1.02**	**-0.16**
-0.01	*0.08*	*-0.02*

Step size: 0.01256

Single parameter update

Start repeated update

Stop repeated update

Randomize parameters

Visualization of the data loss computation. Each row is loss due to one datapoint. The first three columns are the 2D data x_i and the label y_i. The next three columns are the three class scores from each classifier $f(x_i; W, b) = Wx_i + b$ (E.g. s[0] = x[0] * W[0,0] + x[1] * W[0,1] + b[0]). The last column is the data loss for a single example, L_i.

x[0]	x[1]	y	s[0]	s[1]	s[2]	L
0.50	0.40	0	0.38	-0.32	-0.44	0.66
0.80	0.30	0	0.52	-0.59	-0.26	0.58
0.30	0.80	0	0.52	0.03	-0.89	0.62
-0.40	0.30	1	-0.33	0.29	-0.57	0.67
-0.30	0.70	1	0.03	0.42	-0.95	0.66
-0.70	0.20	1	-0.61	0.47	-0.54	0.53
0.70	-0.40	2	-0.05	-0.87	0.43	0.64
0.50	-0.60	2	-0.34	-0.82	0.58	0.50
-0.40	-0.50	2	-0.90	-0.11	0.25	0.70
						mean:
						0.62

```
Total data loss: 0.62
Regularization loss: 0.29
Total loss: 0.91
```

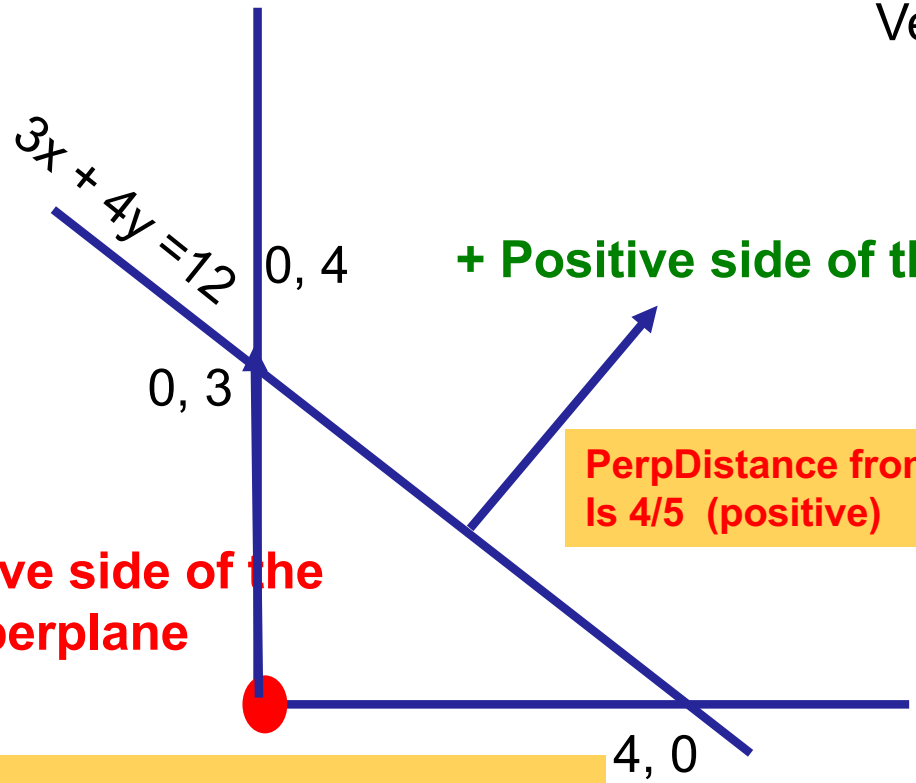
L2 Regularization strength: 0.10000

Multiclass SVM loss formulation:
- ○ Weston Watkins 1999
- ○ One vs. All
- ○ Structured SVM
- ◉ Softmax

REFRESH: HyperPlane 3x + 4y =12

QUESTION: What side of the hyperplane does the origin lie?

Shortcut: 3(0) + 4(0) -10 / SQTR((3 * 3) (4*4)) = -10/5=-2



Vector that normal to Hyperpane (3, 4)

+ Positive side of the hyperplane

PerpDistance from point (0,4) to the hyperplane 3x + 4y =12 **Is 4/5 (positive)**

- Negative side of the hyperplane

The distance from a point to a plane — is equal to length of the perpendicular lowered from a point on a plane.

If $Ax + By + Cz + D = 0$ is a plane equation, then distance from point $M(M_x, M_y, M_z)$ to plane can be found using the following formula

$$d = \frac{|A \cdot M_x + B \cdot M_y + C \cdot M_z + D|}{\sqrt{A^2 + B^2 + C^2}}$$

Shortcut:
(3(0) + 4(0) -12)
---------- = -12/5 = -2.4
SQTR((3 * 3) + (4*4))

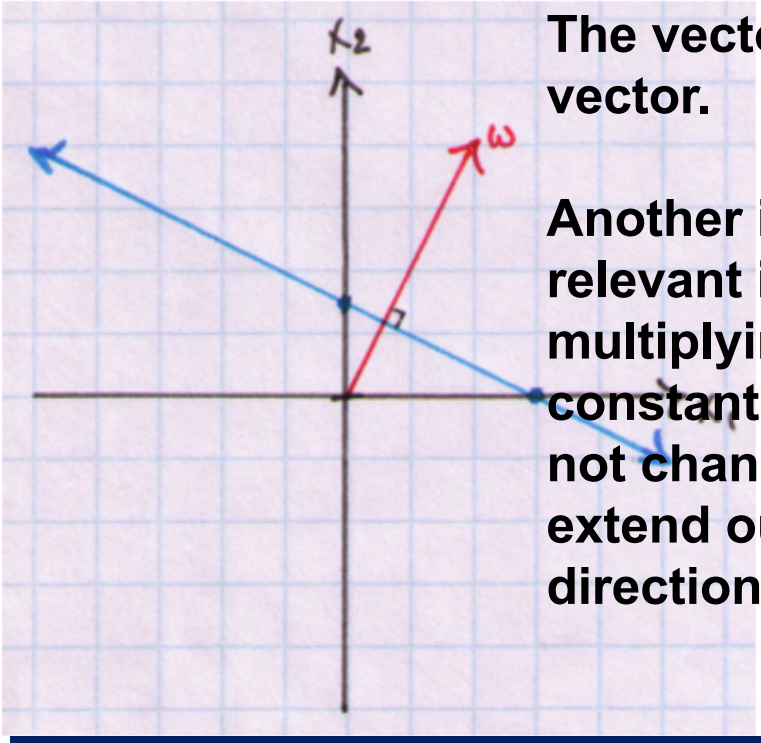
Plotting a hyperplane

$$2x_1 + 4x_2 = 6$$

Equation of a hyperplane

$$x_2 = -\frac{1}{2}x_1 + \frac{3}{2}$$

Plot the hyperplane using this equation



The vector w, [2, 4], is called the normal vector.

Another important property that will be relevant in our discussion is that multiplying vector w and b by a constant (scalar multiplication) does not change our hyperplane but does extend our normal vector in the original direction.

https://quantmacro.wordpress.com/2016/06/14/support-vector-machines-without-tears-part-1/

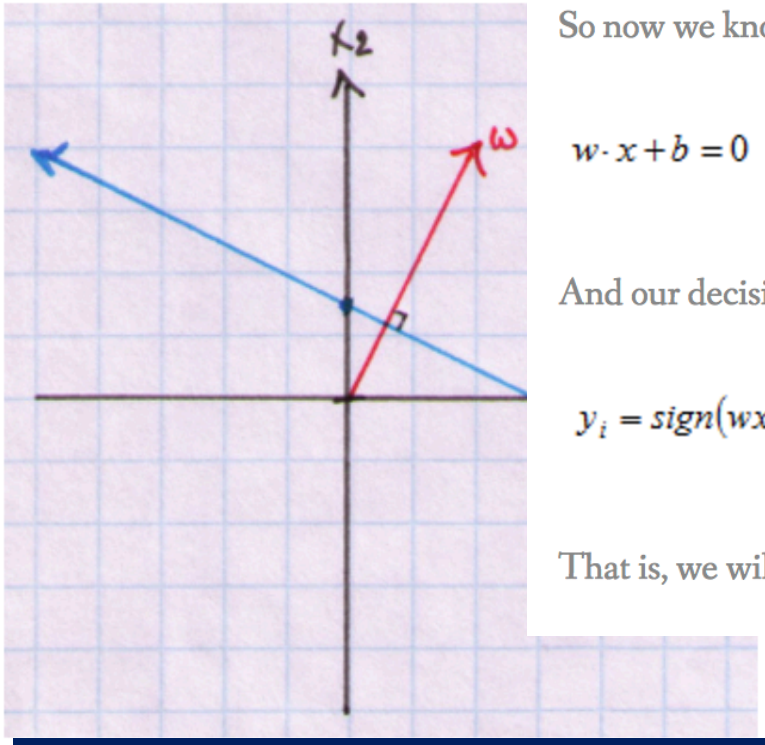
Plotting a hyperplane

$$2x_1 + 4x_2 = 6$$

Equation of a hyperplane

$$x_2 = -\frac{1}{2}x_1 + \frac{3}{2}$$

Plot the hyperplane using this equation



So now we know that our separating plane will be defined as:

$$w \cdot x + b = 0$$

And our decision rule for the model will be:

$$y_i = sign(wx_i + b)$$

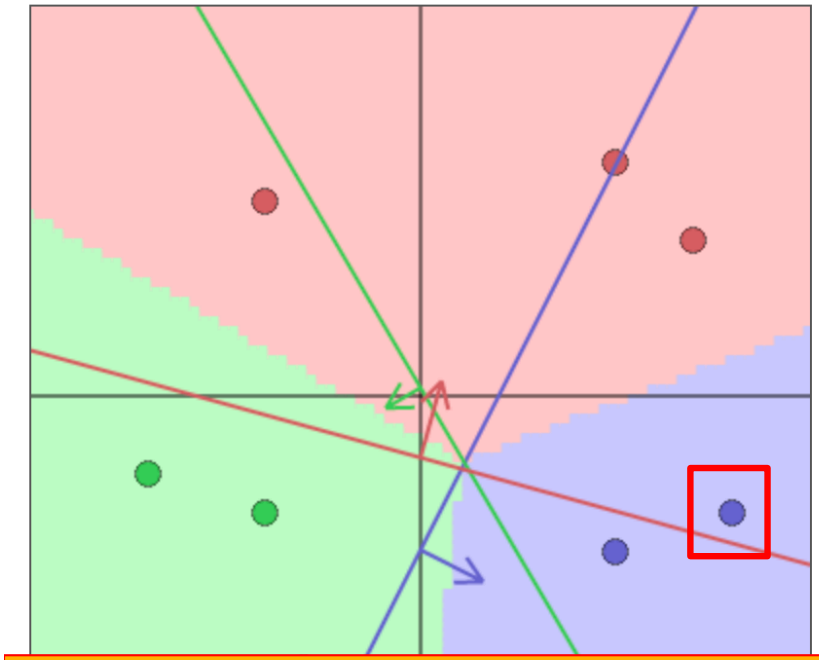
That is, we will assign observation xi to class +1 if wx+b is >0 and assign xi to class -1 if wx+b <0.

https://quantmacro.wordpress.com/2016/06/14/support-vector-machines-without-tears-part-1/

QUIZ: Calculate negative logLoss

Softmax Calculation (column L is mistakenly labeled)
Loss is
-log(exp(1.21)/sum(exp(c(1.214, -0.55, 0.151))))
0.4205871

Datapoints (red/gree/b... whichever ... current we... indicates it... classifier co... blue line shows the set of points (x_0, x_1) that give score of zero. The blue arrow draws the vector $(W_{0,0}, W_{0,1})$, which shows the direction of score increase and its length is proportional to how steep the increase is.
Note: you can drag the datapoints.



... parameters.

W[0,0]	W[0,1]	b[0]
1.61	0.82	-0.32
0.01	-0.02	-0.02
W[1,0]	W[1,1]	b[1]
-0.90	0.53	0.01
0.04	-0.00	-0.00
W[2,0]	W[2,1]	b[2]
0.54	-1.97	0.31
0.01	-0.01	0.02

Step size: 0.10000

Visualization of the data loss computation. Each row is loss due to one datapoint. The first three columns are the 2D data x_i and the label y_i. The next three columns are the three class scores from each classifier $f(x_i; W, b) = Wx_i + b$ (E.g. s[0] = x[0] * W[0,0] + x[1] * W[0,1] + b[0]). The last column is the data loss for a single example, L_i.

x[0]	x[1]	y	s[0]	s[1]	s[2]	L
0.50	0.40	0	0.81	-0.23	-0.20	0.54
0.80	0.30	0	1.21	-0.55	0.15	??
0.30	0.80	0	0.81	0.16	-1.10	0.51
-0.40	0.30	1	-0.72	0.53	-0.49	0.50
-0.30	0.70	1	-0.23	0.65	-1.23	0.45
-0.70	0.20	1	-1.28	0.74	-0.46	0.36
0.70	-0.40	2	0.48	-0.83	1.48	0.38
0.50	-0.60	2	-0.01	-0.75	1.76	0.22
-0.40	-0.50	2	-1.37	0.11	1.08	0.38
						mean:
						0.42

Total data loss: 0.42
Regularization loss: 0.85
Total loss: 1.27

L2 Regularization strength: 0.10000

Look at example in purple class
- matrix(c(-0.32, 1.61, 0.82, 0.1, -0.9, 0.53, 0.31, 0.54, -1.97), 3, 3, byrow = TRUE) %*% c(1, 0.8, 0.3)
- 1.214, -0.461, 0.151

> 1.214 #SCORE for this class 0 example

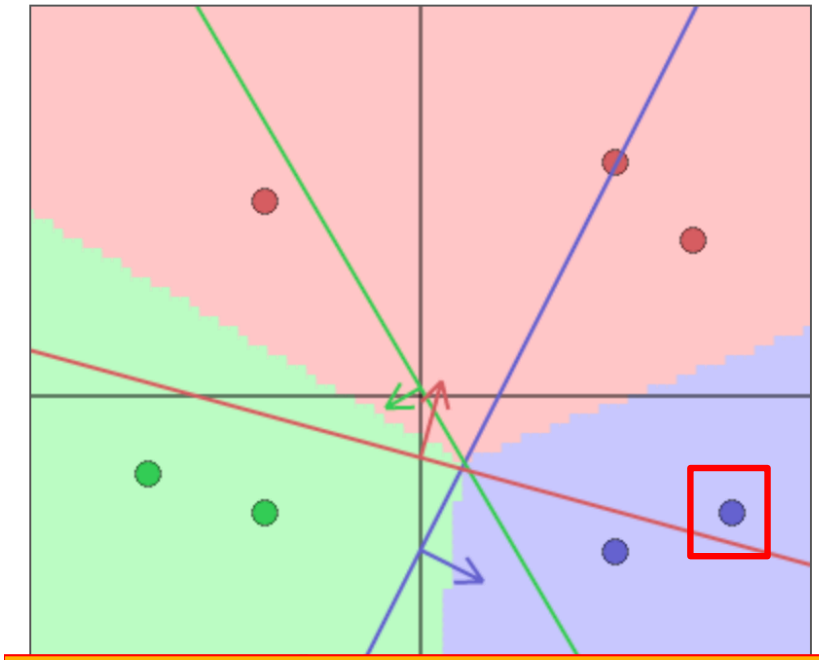
1.21 - log(exp(c(-0.32, 1.61, 0.82))

- -log(exp(1.21)/sum(exp(c(1.214, -0.55, 0.151))))
- 0.4205871. #loss on Prob(CLASS0 |example)

-Log(Pr(Y=y_i |X_i; W) calculation

Softmax Calculation (column L is mistakenly labeled)
Loss is
-log(exp(1.21)/sum(exp(c(1.214, -0.55, 0.151))))
0.4205871

Datapoints (red/gree/b... whichever ... current we... indicates it... classifier co... blue line shows the set of points (x_0, x_1) that give score of zero. The blue arrow draws the vector $(W_{0,0}, W_{0,1})$, which shows the direction of score increase and its length is proportional to how steep the increase is.
Note: you can drag the datapoints.



... parameters.

W[0,0]	W[0,1]	b[0]
1.61	0.82	-0.32
0.01	-0.02	-0.02
W[1,0]	W[1,1]	b[1]
-0.90	0.53	0.01
0.04	-0.00	-0.00
W[2,0]	W[2,1]	b[2]
0.54	-1.97	0.31
0.01	-0.01	0.02

Step size: 0.10000

Visualization of the data loss computation. Each row is loss due to one datapoint. The first three columns are the 2D data x_i and the label y_i. The next three columns are the three class scores from each classifier $f(x_i; W, b) = Wx_i + b$ (E.g. s[0] = x[0] * W[0,0] + x[1] * W[0,1] + b[0]). The last column is the data loss for a single example, L_i.

x[0]	x[1]	y	s[0]	s[1]	s[2]	L
0.50	0.40	0	0.81	-0.23	-0.20	0.54
0.80	0.30	0	1.21	-0.55	0.15	0.42
0.30	0.80	0	0.81	0.16	-1.10	0.51
-0.40	0.30	1	-0.72	0.53	-0.49	0.50
-0.30	0.70	1	-0.23	0.65	-1.23	0.45
-0.70	0.20	1	-1.28	0.74	-0.46	0.36
0.70	-0.40	2	0.48	-0.83	1.48	0.38
0.50	-0.60	2	-0.01	-0.75	1.76	0.22
-0.40	-0.50	2	-1.37	0.11	1.08	0.38
						mean:
						0.42

Total data loss: 0.42
Regularization loss: 0.85
Total loss: 1.27

L2 Regularization strength: 0.10000

Look at example in purple class
- matrix(c(-0.32, 1.61, 0.82, 0.1, -0.9, 0.53, 0.31, 0.54, -1.97), 3, 3, byrow = TRUE) %*% c(1, 0.8, 0.3)
- 1.214, -0.461, 0.151

> 1.214 #SCORE for this class 0 example

1.21 - log(exp(c(-0.32, 1.61, 0.82))

- -log(exp(1.21)/sum(exp(c(1.214, -0.55, 0.151))))
- 0.4205871. #loss on Prob(CLASS0 |example)

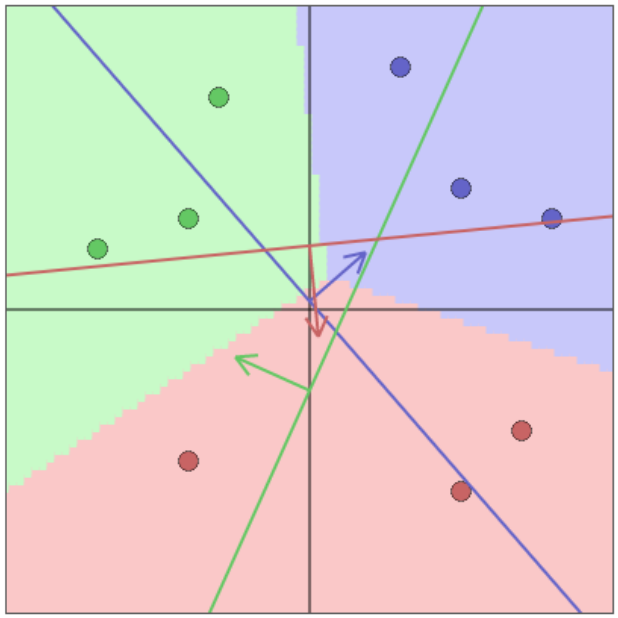
Loss for mulitnomial logistic regression (SoftMax Classifier)

QUIZ: Calculate negative logLoss

-Log(Pr(Y=y_i |X_i; W) calculation

Datapoints are shown as circles colored by their class (red/gree/blue). The background regions are colored by whichever class is most likely at any point according to the current weights. Each classifier is visualized by a line that indicates its zero score level set. For example, the blue classifier computes scores as $W_{0,0}x_0 + W_{0,1}x_1 + b_0$ and the blue line shows the set of points (x_0, x_1) that give score of zero. The blue arrow draws the vector $(W_{0,0}, W_{0,1})$, which shows the direction of score increase and its length is proportional to how steep the increase is.

Note: you can drag the datapoints.



Parameters W, b are shown below. The value is in **bold** and its gradient (computed with backprop) is in *red, italic* below. Click the triangles to control the parameters.

W[0,0]	W[0,1]	b[0]
1.85 *-0.02*	**1.61** *-0.02*	**-0.04** *0.00*
W[1,0]	W[1,1]	b[1]
-2.45 *0.02*	**1.10** *-0.01*	**0.29** *0.00*
W[2,0]	W[2,1]	b[2]
0.29 *-0.00*	**-3.00** *0.03*	**0.63** *-0.01*

Step size: 0.10000

Single parameter update

Start repeated update

Stop repeated update

Randomize parameters

Visualization of the data loss computation. Each row is loss due to one datapoint. The first three columns are the 2D data x_i and the label y_i. The next three columns are the three class scores from each classifier $f(x_i; W, b) = Wx_i + b$ (E.g. s[0] = x[0] * W[0,0] + x[1] * W[0,1] + b[0]). The last column is the data loss for a single example, L_i.

x[0]	x[1]	y	s[0]	s[1]	s[2]	L
0.50	0.40	0	1.52	-0.49	-0.42	0.24
0.80	0.30	0	1.92	-1.33	-0.04	0.17
0.30	0.80	0	1.80	0.44	-1.68	0.25
-0.40	0.30	1	-0.30	1.61	-0.39	0.25
-0.30	0.70	1	0.53	1.80	-1.56	0.27
-0.70	0.20	1	-1.02	2.23	-0.17	0.12
0.70	-0.40	2	0.61	-1.86	2.04	0.23
0.50	-0.60	2	-0.09	-1.59	2.58	0.08
-0.40	-0.50	2	-1.59	0.72	2.02	0.26
						mean:
						0.21

Total data loss: 0.21
Regularization loss: 0.22
Total loss: 0.43

L2 Regularization strength: 0.01000

$$L = -\log \hat{y}_0 = -\log \frac{\exp\{s_0\}}{\sum_{j=1}^{3} \exp\{s_j\}}$$

$$= -\log \frac{\exp\{1.52\}}{\exp\{1.52\} + \exp\{-0.49\} + \exp\{-0.42\}} = -\log 0.78 = 0.24$$

Multiclass SVM loss formulation:

- Weston Watkins 1999
- One vs. All
- Structured SVM
- Softmax

Outline

- Introduction
- Binomial logistic regression
- Multinomial Logistic regression
- Linear Classifier via separating hyperplane
- Multinomial Logistic regression Classifier
- Linear classifier demo
- Learning Softmax Classifiers via optimization
  - Implementation (SoftMax Classifier)
- Regressions in matrix terms and in graph terms
- Summary

Optimization: Random search for weight vectors Cifar-10

Strategy #1: A first very bad idea solution: **Random search**

```
# assume X_train is the data where each column is an example (e.g. 3073 x 50,000)
# assume Y_train are the labels (e.g. 1D array of 50,000)
# assume the function L evaluates the loss function

bestloss = float("inf") # Python assigns the highest possible float value
for num in xrange(1000):
  W = np.random.randn(10, 3073) * 0.0001 # generate random parameters
  loss = L(X_train, Y_train, W) # get the loss over the entire training set
  if loss < bestloss: # keep track of the best solution
    bestloss = loss
    bestW = W
  print 'in attempt %d the loss was %f, best %f' % (num, loss, bestloss)

# prints:
# in attempt 0 the loss was 9.401632, best 9.401632
# in attempt 1 the loss was 8.959668, best 8.959668
# in attempt 2 the loss was 9.044034, best 8.959668
# in attempt 3 the loss was 9.278948, best 8.959668
# in attempt 4 the loss was 8.857370, best 8.857370
# in attempt 5 the loss was 8.943151, best 8.857370
# in attempt 6 the loss was 8.605604, best 8.605604
# ... (truncated: continues for 1000 lines)
```

Optimization: Random search Cifar-10

Lets see how well this works on the test set...

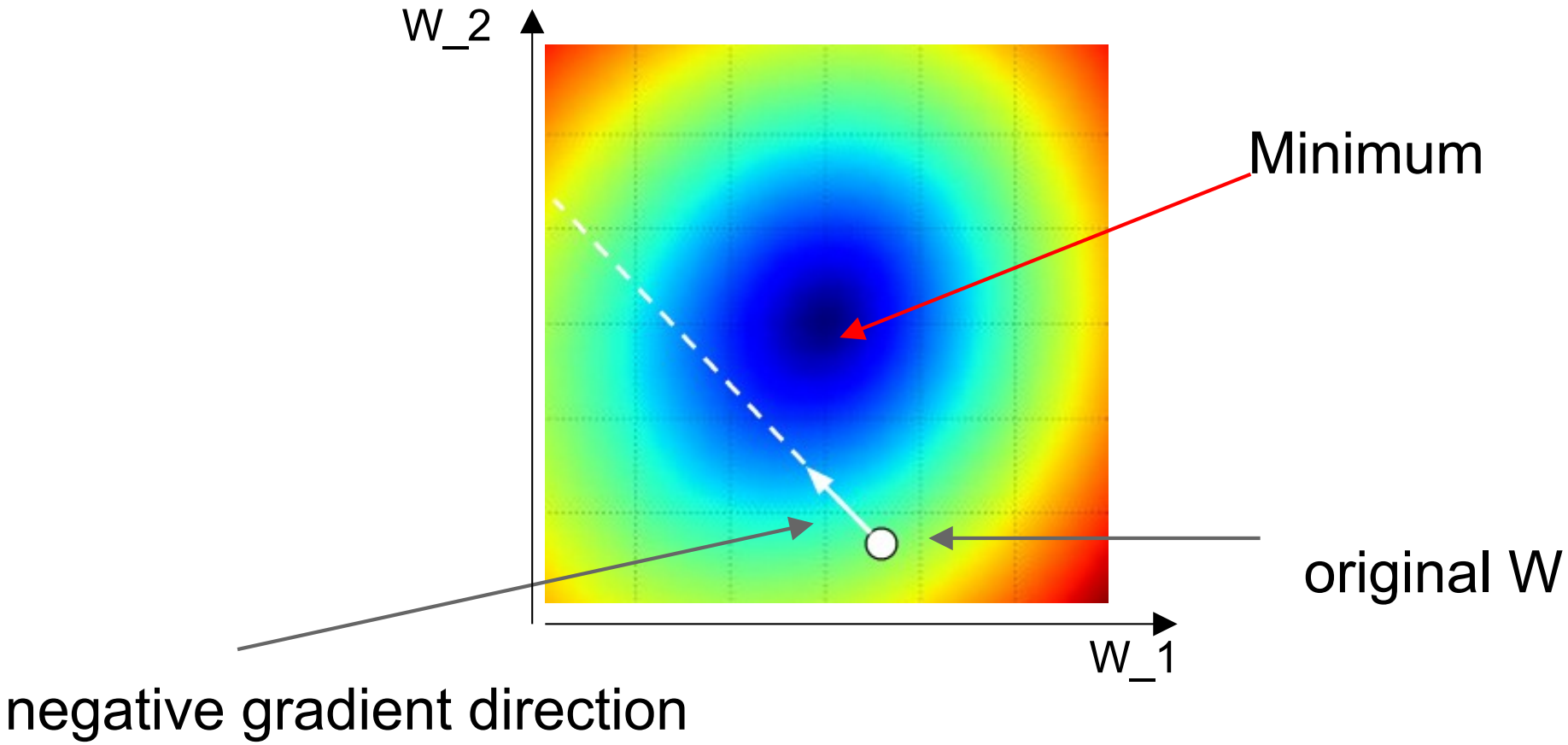
```
# Assume X_test is [3073 x 10000], Y_test [10000 x 1]
scores = Wbest.dot(Xte_cols) # 10 x 10000, the class scores for all test examples
# find the index with max score in each column (the predicted class)
Yte_predict = np.argmax(scores, axis = 0)
# and calculate accuracy (fraction of predictions that are correct)
np.mean(Yte_predict == Yte)
# returns 0.1555
```

15.5% accuracy! not bad!
(SOTA is ~95%)

Gradient Descent

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Negative gradient direction



Gradient descent numerically
I.e., Calculate gradient numerically

- In 1-dimension, the derivative of a function:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

No need for calculus!

- In multiple dimensions, the gradient is the VECTOR of (partial derivatives).

- Numerical Gradient

$$\frac{df(x)}{dx} = \frac{f(x+h) - f(x)}{h}$$

Numerical approximation of gradient

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

gradient dW:

[?,
?,
?,
?,
?,
?,
?,
?,
?,…]

Numerical approximation of gradient

current W:	**W + h** (first dim)**:**	**gradient dW:**
[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,…]	[0.34 + **0.0001**, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,…]	[?, ?, ?, ?, ?, ?, ?, ?, ?,…]
loss 1.25347	**loss 1.25322**	

Numerical approximation of gradient

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

W + h (first dim)**:**

[0.34 + **0.0001**,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25322

gradient dW:

[**-2.5**,
?,
?,

(1.25322 - 1.25347)/0.0001
= -2.5

$$\frac{df(x)}{dx} = \frac{f(x+h) - f(x)}{h}$$

?,
?,…]

Numerical approximation of gradient

current W:	**W + h** (second dim)**:**	**gradient dW:**
[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,…]	[0.34, -1.11 + **0.0001**, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,…]	[-2.5, ?, ?, ?, ?, ?, ?, ?, ?,…]
loss 1.25347	**loss 1.25353**	

Numerical approximation of gradient

current W:	**W + h** (second dim)**:**	**gradient dW:**
[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,…] **loss 1.25347**	[0.34, -1.11 + **0.0001**, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,…] **loss 1.25353**	[-2.5, **0.6**, ?, ?, ?,…]

(1.25353 - 1.25347)/0.0001
= 0.6

$$\frac{df(x)}{dx} = \frac{f(x+h) - f(x)}{h}$$

Numerical approximation of gradient

current W:	**W + h** (third dim)**:**	**gradient dW:**
[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,…]	[0.34, -1.11, 0.78 + **0.0001**, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,…]	[-2.5, 0.6, ?, ?, ?, ?, ?, ?, ?,…]
loss 1.25347	**loss 1.25347**	

Evaluation of the gradient numerically in code

$$\frac{df(x)}{dx} = \frac{f(x+h) - f(x)}{h}$$

```
def eval_numerical_gradient(f, x):
  """
  a naive implementation of numerical gradient of f at x
  - f should be a function that takes a single argument
  - x is the point (numpy array) to evaluate the gradient at
  """

  fx = f(x) # evaluate function value at original point
  grad = np.zeros(x.shape)
  h = 0.00001

  # iterate over all indexes in x
  it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
  while not it.finished:

    # evaluate function at x+h
    ix = it.multi_index
    old_value = x[ix]
    x[ix] = old_value + h # increment by h
    fxh = f(x) # evalute f(x + h)
    x[ix] = old_value # restore to previous value (very important!)

    # compute the partial derivative
    grad[ix] = (fxh - fx) / h # the slope
    it.iternext() # step to next dimension

  return grad
```

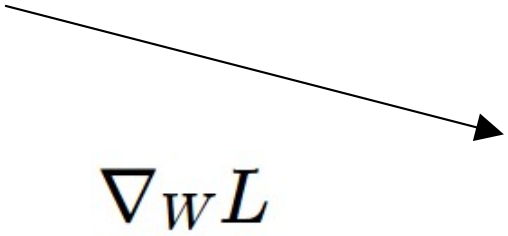
- approximate
- very slow to evaluate

dW = ...
(some function data and W)

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

dW = ...
(some function
data and W)



$\nabla_W L$

gradient dW:

[-2.5,
0.6,
0,
0.2,
0.7,
-0.5,
1.1,
1.3,
-2.1,…]

Calculate gradient via Calculus

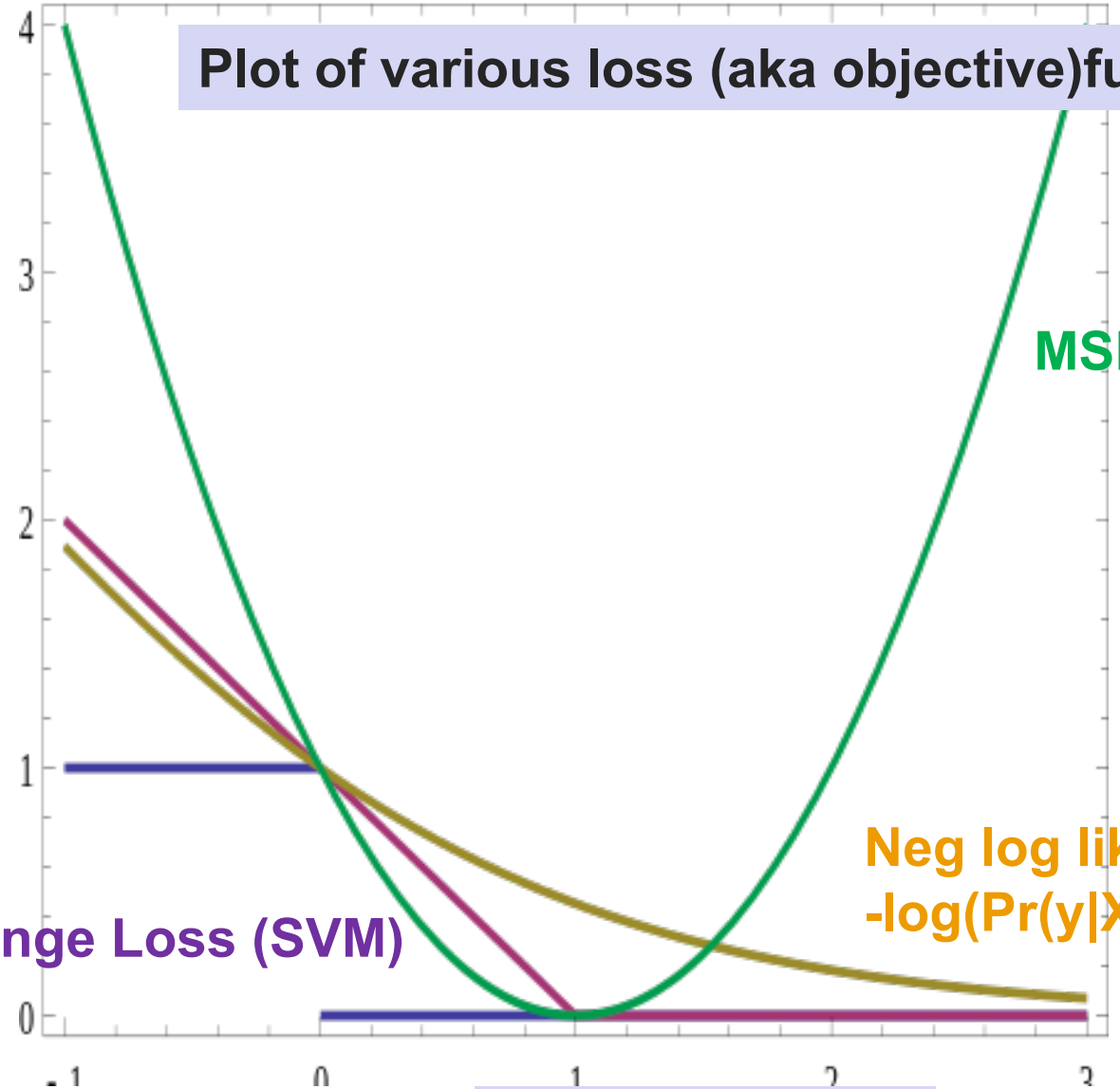
- **Numerical estimate of gradient, the derivative of a function:**

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- **In multiple dimensions, the gradient is the vector of (partial derivatives).**

- **Use calculus**

Whose line is it anyway?



Logistic Regression

log loss for binary versus cross-entropy for K-classes

	Binomial LR	Multinomial LR
	$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}log\left(\hat{p}^{(i)}\right) + \left(1-y^{(i)}\right)log\left(1-\hat{p}^{(i)}\right)\right]$	$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$
Gradient	$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma\left(\theta^T\cdot\mathbf{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$	$\nabla_{\theta_k} J(\Theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{p}_k^{(i)} - y_k^{(i)}\right)\mathbf{x}^{(i)}$
Class	$\hat{y} = \begin{cases} 0 \text{ if } \hat{p} < 0.5, \\ 1 \text{ if } \hat{p} \geq 0.5. \end{cases}$	$\hat{y} = \underset{k}{\text{argmax}}\ \sigma(\mathbf{s}(\mathbf{x}))_k = \underset{k}{\text{argmax}}\ s_k(\mathbf{x}) = \underset{k}{\text{argmax}}\left(\left(\theta^{(k)}\right)^T\cdot\mathbf{x}\right)$
Prob	$\hat{p} = h_\theta(\mathbf{x}) = \sigma\left(\theta^T\cdot\mathbf{x}\right)$	$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K}\exp\left(s_j(\mathbf{x})\right)}$ • K is the number of classes. • $\mathbf{s}(\mathbf{x})$ is a vector containing the scores of each class for the instance $\mathbf{x}$. • $\sigma(\mathbf{s}(\mathbf{x}))_k$ is the estimated probability that the instance $\mathbf{x}$ belongs to class k given the scores of each class for that instance.

Can't solve for the min of SoftMax analytically (no closed form) BUT …SGD!

- Taking derivatives of J(θ), one can show that the gradient is:

$$\nabla_{\theta_k} J(\Theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{p}_k^{(i)} - y_k^{(i)} \right) \mathbf{x}^{(i)}$$

Y ∈{1 or 0}

Gradient is the weighted sum of training data where is the weight is the size of the error; desired the target class should be 1 and all others should be zero. Positive means a penalty and Minus means we are reducing the loss

Single EXAMPLE

p_i = [0.2, 0.3, 0.5]

y_i = [0, 1, 0]

df/dw = X_i [0.2, -0.7, 0.5]

Recall the meaning of the "$\nabla_{\theta^{(k)}}$" notation. In particular, $\nabla_{\theta^{(k)}} J(\theta)$ is itself a vector, so that its j-th element is $\frac{\partial J(\theta)}{\partial \theta_{lk}}$ the partial derivative of $J(\theta)$ with respect to the j-th element of $\theta^{(k)}$.

Armed with this formula for the derivative, one can then plug it into a standard optimization package and have it minimize $J(\theta)$.

http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/

Derivative of the Logarithmic Function

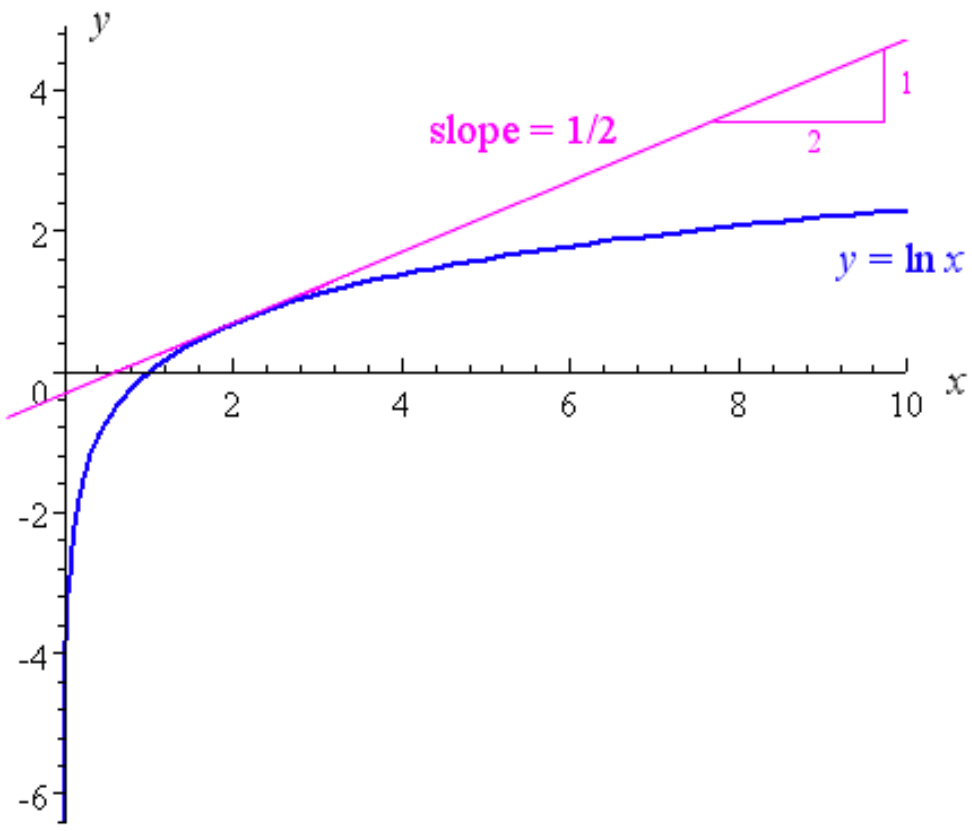
5. Derivative of the Logarithmic Function

by M. Bourne

First, let's look at a graph of the log function with base e, that is:

$f(x) = \log_e(x)$ (usually written "ln x").

The tangent at $x = 2$ is included on the graph.



The derivative of the logarithmic function $y = \ln x$ is given by:

$$\frac{d}{dx}(\ln x) = \frac{1}{x}$$

You will see it written in a few other ways as well. The following are equivalent:

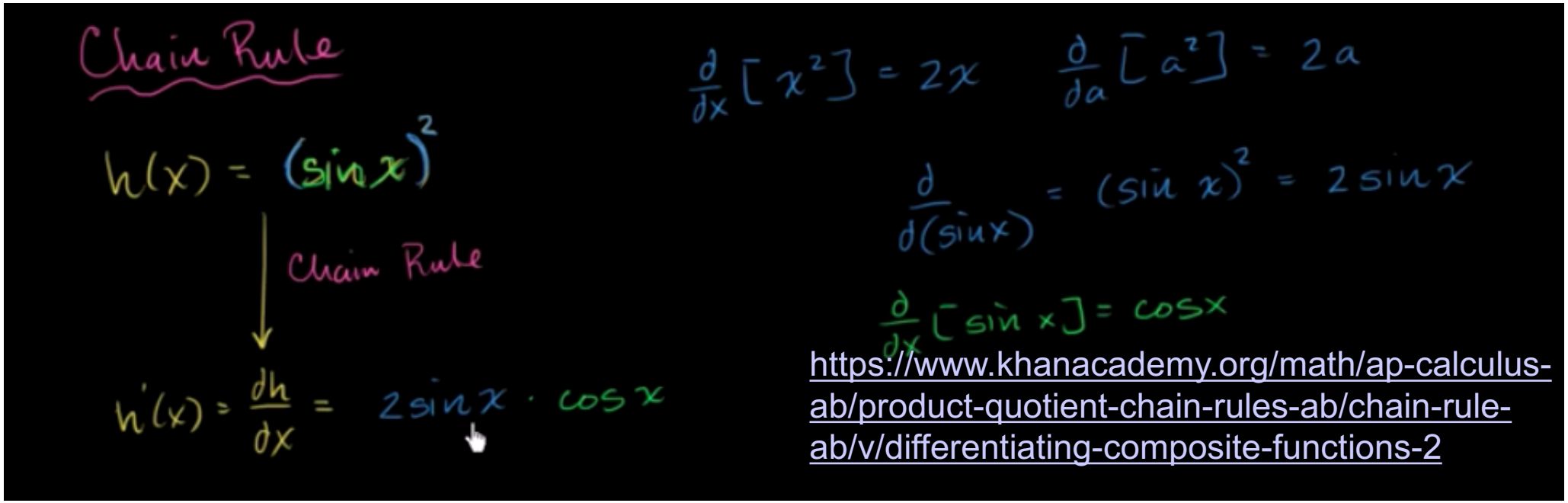
$$\frac{d}{dx}\log_e x = \frac{1}{x}$$

If $y = \ln x$, then $\frac{dy}{dx} = \frac{1}{x}$

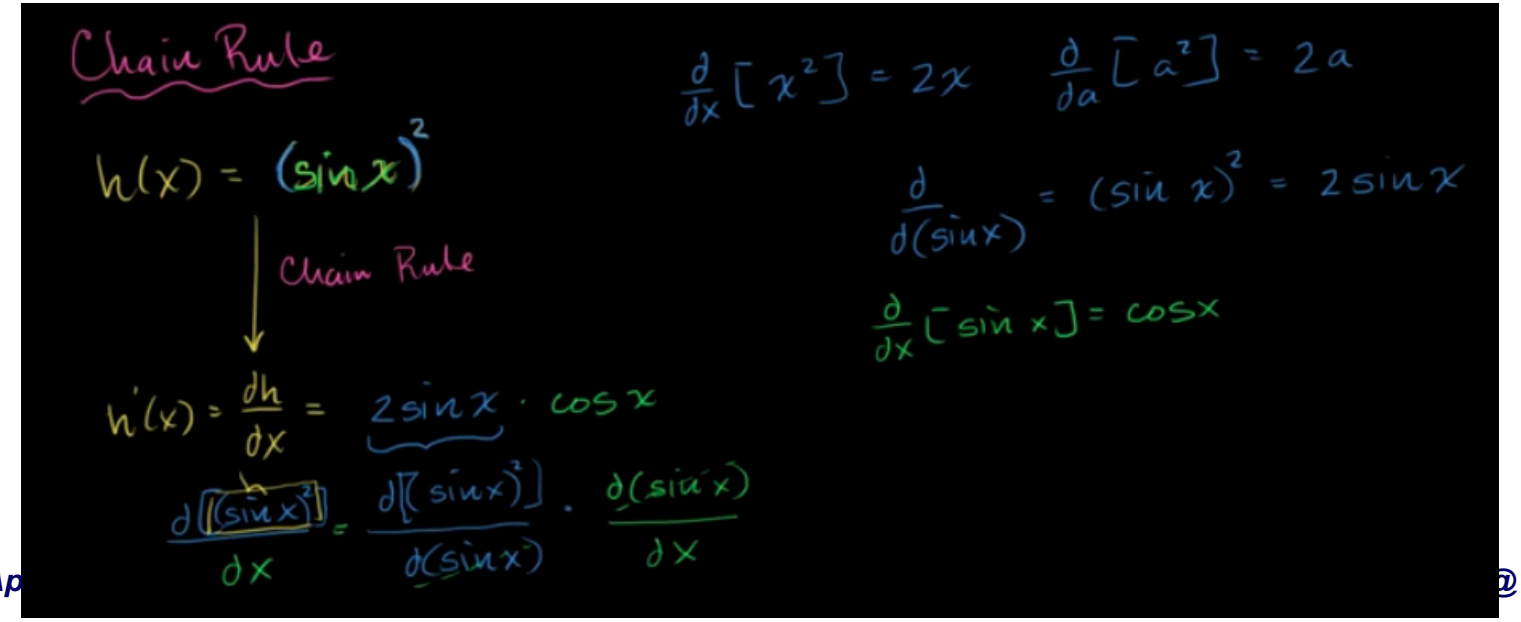
$$\frac{d\,(\ln 2x)}{dx} = \frac{2}{2x} = \frac{1}{x}$$

$$\frac{d\,(\ln x^2)}{dx} = \frac{2}{x} \text{ via chain rule}$$

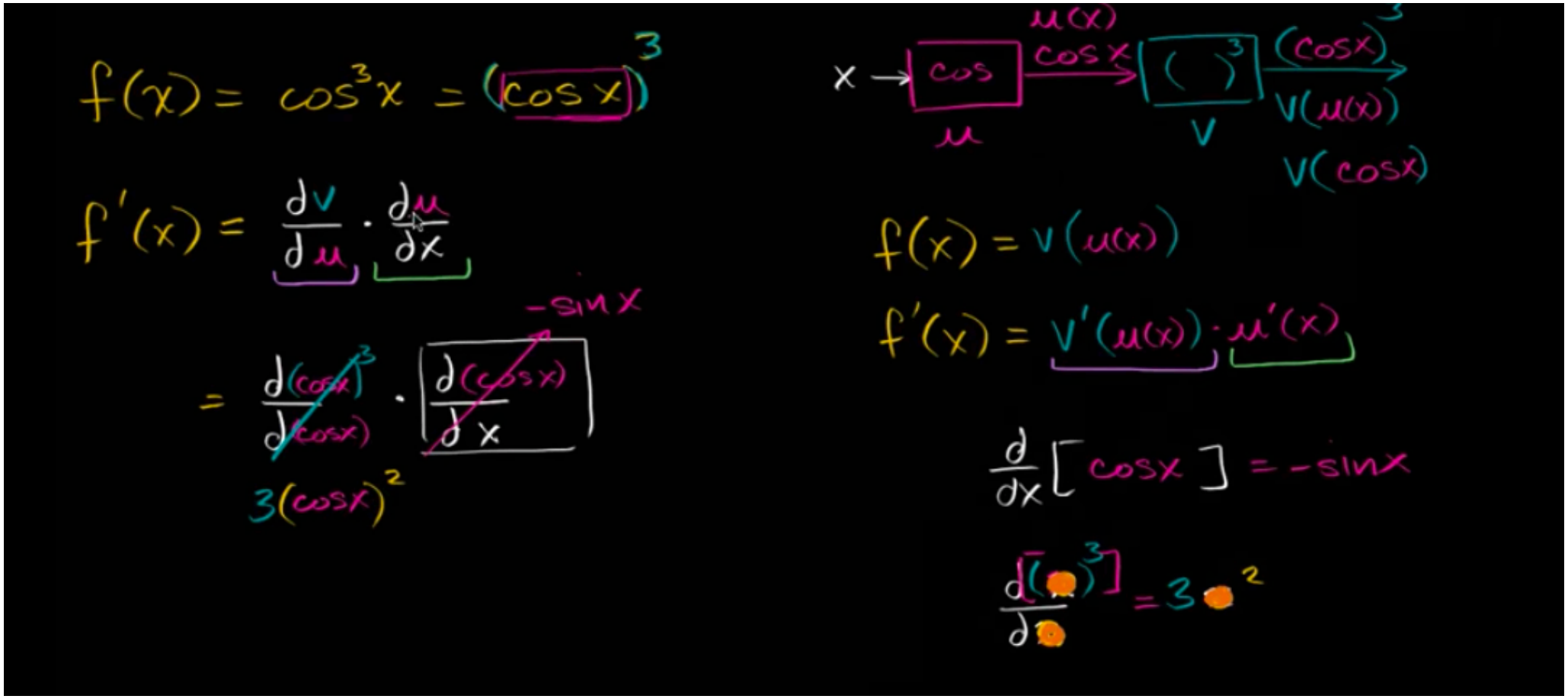
Chain Rule (Khan Academy)



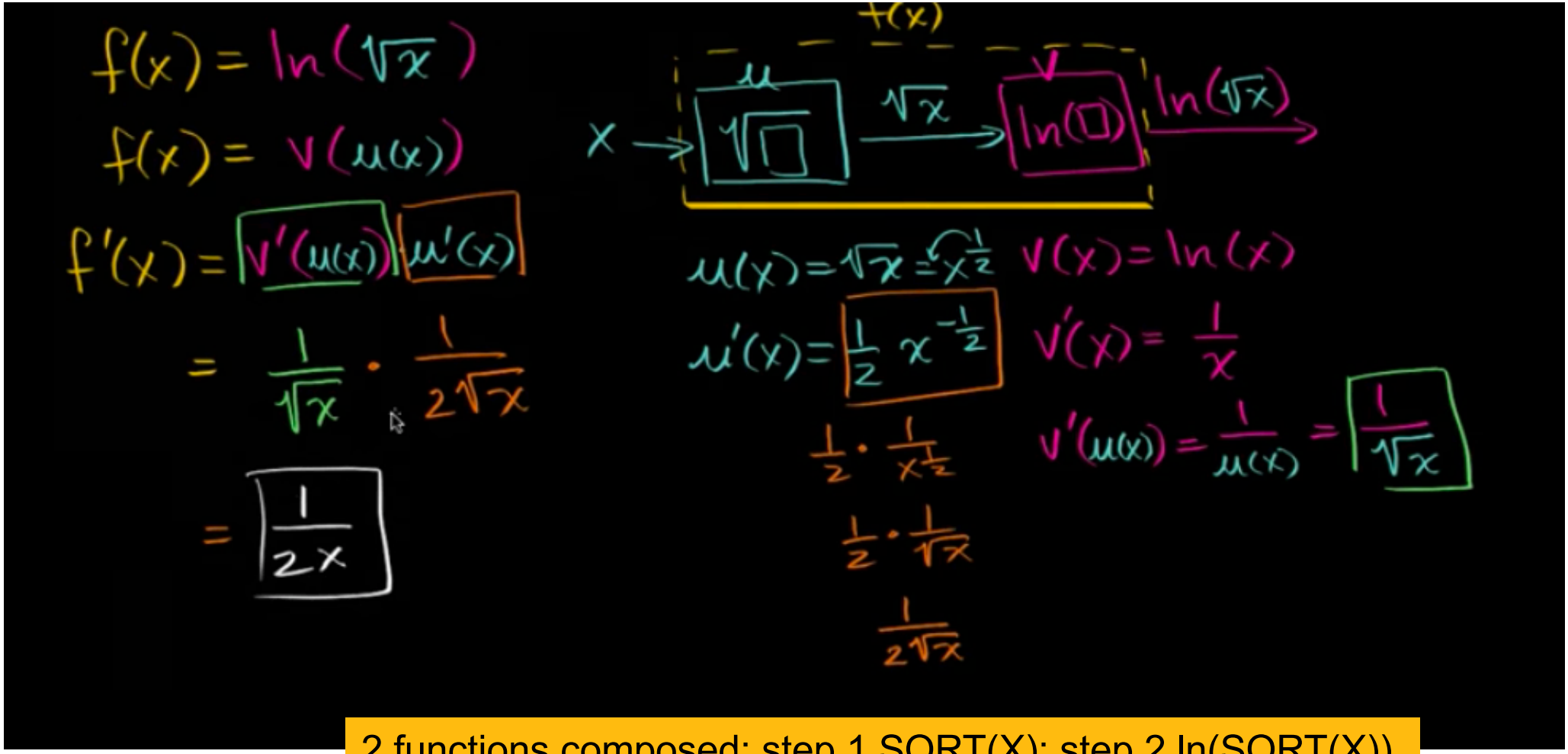
https://www.khanacademy.org/math/ap-calculus-ab/product-quotient-chain-rules-ab/chain-rule-ab/v/differentiating-composite-functions-2



Chain Rule Example 2



Chain Rule example 3



2 functions composed: step 1 SQRT(X); step 2 ln(SQRT(X))
Derivative of the outside function wrtt inside function
Derivative of the inside function wrt X

Derivative w.r.t. Input of Softmax

$$p(c_k=1|\boldsymbol{x})=\frac{e^{o_k}}{\sum_j e^{o_j}}$$

$$\frac{dL}{dw}=\frac{dL}{do}\frac{do}{dw}$$

$$\frac{dL}{do}X$$

$$(P(c|X)-y)X$$

Note: O is the perpendicular distance

$$L(\boldsymbol{x},y;\boldsymbol{\theta})=-\sum_j y_j \log p(c_j|\boldsymbol{x}) \qquad \boldsymbol{y}=[\overset{1}{0}0..0\overset{k}{1}0..0\overset{C}{0}]$$

By substituting the first formula in the second, and taking the derivative w.r.t. $\boldsymbol{o}$ we get:

$$\frac{\partial L}{\partial o}=p(c|\boldsymbol{x})-\boldsymbol{y}$$

$$\frac{dL}{dw}=\frac{dL}{do}\frac{do}{dw}$$

$$= \frac{dL}{do}X$$

$$= (P(c|X)-y)X$$

HOMEWORK: prove it!

https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/

Derivative w.r.t. Input of Softmax

$$\frac{\partial L}{\partial o_i} = -\sum_k y_k \frac{\partial \log p_k}{\partial o_i} = -\sum_k y_k \frac{1}{p_k}\frac{\partial p_k}{\partial o_i}$$

$$= -y_i(1-p_i) - \sum_{k\neq i} y_k \frac{1}{p_k}(-p_k p_i)$$

$$= -y_i(1-p_i) + \sum_{k\neq i} y_k (p_i)$$

$$= -y_i + y_i p_i + \sum_{k\neq i} y_k (p_i)$$

$$= p_i \left(\sum_k y_k\right) - y_i = p_i - y_i$$

given that $\sum_k y_k = 1$ from the slides (as y is a vector with only one non-zero element, which is 1).

$\frac{d}{dx}(\ln x^2) = \frac{2}{x}$ via chain rule

$$\frac{dL}{dw} = \frac{dL}{do}\frac{do}{dw}$$
$$= \frac{dL}{do}X$$
$$= (P(c|X) - y)X$$

Focus on class of example *i* and also on notClass

Assume p = [0.2, 0.3, 0.5]
y= [0, 1, 0]
Then dL/dw = X^T [0.2, -0.7, 0.5]
Take a weighted summed of X using three different weight (error vectors). Thereby generating the three gradients vectors.

$$y = [\overset{1}{0}\,0\,..\,0\,\overset{k}{1}\,0\,..\,0\,\overset{C}{}]$$

Notice how elegant and simple this expression is. Suppose the probabilities we computed were p = [0.2, 0.3, 0.5], and that the correct class was the middle one (with probability 0.3). According to this derivation the gradient on the scores would be df = [0.2, -0.7, 0.5]. Recalling what the interpretation of the gradient, we see that this result is highly intuitive: increasing the first or last element of the score vector f (the scores of the incorrect classes) leads to an *increased* loss (due to the positive signs +0.2 and +0.5) - and increasing the loss is bad, as expected. However, increasing the score of the correct class has *negative* influence on the loss. The gradient of -0.7 is telling us that increasing the correct class score would lead to a decrease of the loss *Li*, which makes sense.

Derivative w.r.t. Input of Softmax

$$\frac{df}{dw}$$

$$\frac{\partial L}{\partial o_i} = -\sum_k y_k \frac{\partial \log p_k}{\partial o_i} = -\sum_k y_k \frac{1}{p_k}\frac{\partial p_k}{\partial o_i}$$

$$= -y_i(1-p_i) - \sum_{k\neq i} y_k \frac{1}{p_k}(-p_k p_i)$$

$$= -y_i(1-p_i) + \sum_{k\neq i} y_k (p_i)$$

$$= -y_i + y_i p_i + \sum_{k\neq i} y_k (p_i)$$

$$= p_i \left(\sum_k y_k\right) - y_i = p_i - y_i$$

given that $\sum_k y_k = 1$ from the slides (as y is a vector with only one non-zero element, which is 1).

$$\boldsymbol{y} = [\overset{1}{0}0..0\overset{k}{1}0..\overset{C}{0}]$$

Here p and y are vectors

Think of y^k as an indicator function

EXAMPLE

p = [0.2, 0.3, 0.5]
Y = [0, 1, 0]
df/dw = X^T [0.2, -0.7, 0.5]

http://cs231n.github.io/neural-networks-case-study/

Notice how elegant and simple this expression is. Suppose the probabilities we computed were p = [0.2, 0.3, 0.5], and that the correct class was the middle one (with probability 0.3). According to this derivation the gradient on the scores would be df = [0.2, -0.7, 0.5]. Recalling what the interpretation of the gradient, we see that this result is highly intuitive: increasing the first or last element of the score vector f (the scores of the incorrect classes) leads to an *increased* loss (due to the positive signs +0.2 and +0.5) - and increasing the loss is bad, as expected. However, increasing the score of the correct class has *negative* influence on the loss. The gradient of -0.7 is telling us that increasing the correct class score would lead to a decrease of the loss *Li*Li, which makes sense.

Intuition behind example of the Derivative

- **Gradient for each class will be used to update the weight vector for that class**
  - MANTRA: the gradient is the weighted sum of the data where the wgt for each example is prop to the error for that example i (p_i -y_i)

$$W = -\alpha \nabla_W$$

$$\nabla_W = X^T(P-T)$$

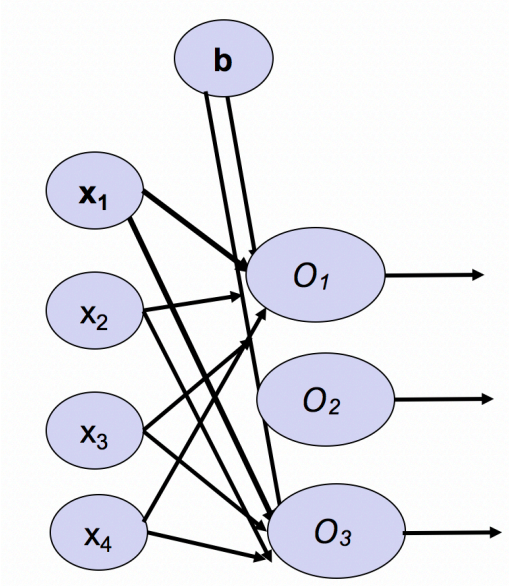
$$\nabla_W = X^T E$$

- **Example**
  - Suppose the probabilities for example i we computed were p = [0.2, **0.3**, 0.5], and that the correct class was the middle one (with probability **0.3**).
  - According to this derivation the gradient on the scores would be df = [0.2, **-0.7**, 0.5].
  - Recalling what the interpretation of the gradient, we see that this result is highly intuitive: increasing the first or last element of the score vector f (the scores of the incorrect classes) leads to an *increased* loss (due to the positive signs +0.2 and +0.5) - and increasing the loss is bad, as expected.
  - However, increasing the score of the correct class has *negative* influence on the loss. The gradient of -0.7 is telling us that increasing the correct class score would lead to a decrease of the loss *Li*, which makes sense.
- **When we take the negative of the gradient the update becomes [-0.2, 0.7, -0.5]. As a result the weights will be increased for the positive class, thereby producing a higher prediction (perpendicular distance)**
- **For each training example calculate the corresponding "weights" for each class**

Outline

- Introduction
- Binomial logistic regression
- Multinomial Logistic regression
- Linear Classifier via separating hyperplane
- Multinomial Logistic regression Classifier
- Linear classifier demo
- Learning Softmax Classifiers via optimization
  - Implementation (SoftMax Classifier)
- Regressions in matrix terms and in graph terms
- Summary

Activation matrices and model matrices: Multiple Targets



4 3 3 Class

150 **X** 4 **W** = 150 $\hat{p}$ **T**

$\nabla_W = X^T(Pr-T)$

$\nabla_W = X^T E$

W = **W** -α ∇_W

× **E** = ∇_W

Model

Data

Class API for machine learning Logistic Regression

```
class LogisticRegressionHomegrown(object):

    def __init__(self):
        self.coef_ = None        # weight vector
        self.intercept_ = None   # bias term
        self._theta = None       # augmented weight vector, i.e., bias + weights
                                 # this allows to treat all decision variables homogeneously
        self.history = {"cost": [],
                        "coef": [],
                        "intercept": [],
                        "grad": [],
                        "acc": []}

    # calculate gradient of objective function
    def _grad(self, X, y):↔

    # full gradient descent, i.e., not stochastic gd
    def _gd(self, X, y, max_iter, alpha=0.05):
        for i in range(max_iter):↔

    # public API for fitting a linear regression model
    def fit(self, X, y, max_iter=1000):↔
    # computes logloss and accuracy for (X, y)
    def score(self, X, y):↔
    # computes scores for each class and each object in X
    def _predict_raw(self, X):↔

    # predicts class for each object in X
    def predict(self, X):↔
```

Gradient calculation for LogRegression

```
# calculate gradient of objective function
def _grad(self, X, y):
    # number of training examples
    n = X.shape[0]

    # get scores for each class and exa
    # 2D matrix
    scores = self._predict_raw(X)

    # transform scores to probabilities
    # softmax
    exp_scores = np.exp(scores)
    probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)

    # error
    probs[range(n),y] -= 1

    # gradient
    gradient = np.dot(X.T, probs) / n

    return gradient
```

```
# full gradient descent, i.e., not stochastic gd
def _gd(self, X, y, max_iter, alpha=0.05):
    for i in range(max_iter):
        self.history["coef"].append(self._theta[1:])
        self.history["intercept"].append(self._theta[0
        metrics = self.score(X, y)
        self.history["cost"].append(metrics["cost"])
        self.history["acc"].append(metrics["acc"])
        # calculate gradient
        grad = self._grad(X, y)
        self.history["grad"].append(grad)
        # do gradient step
        self._theta -= alpha * grad
```

#Could do this way
#Subtract arguments, element-wise.
np.subtract(probs, classOHE)

$$E = \hat{p} - T$$

$$\nabla_W = X^T(\hat{p} - T)$$
$$\nabla_W = X^T E$$

https://github.com/cadgip/DL_course/blob/master/Labs/Unit03_Classification_IrisFlowers/LogRegrIris.ipynb

Logistic Regression

```
14      # calculate gradient of objective function
15      def _grad(self, X, y):
16          # number of training examples
17          n = X.shape[0]
18
19          # get scores for each class and example
20          # 2D matrix
21          scores = self._predict_raw(X)
22
23          # transform scores to probabilities
24          # softmax
25          exp_scores = np.exp(scores)
26          probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
27
28          # error
29          probs[range(n),y] -= 1
30
31          # gradient
32          gradient = np.dot(X.T, probs) / n
33
34          return gradient
35
36      # full gradient descent, i.e., not stochasti
37      def _gd(self, X, y, max_iter, alpha=0.05):
38          for i in range(max_iter):
39              self.history["coef"].append(self._theta[1:])
40              self.history["intercept"].append(self._theta[0])
41
42              metrics = self.score(X, y)
43              self.history["cost"].append(metrics["cost"])
44              self.history["acc"].append(metrics["acc"])
45
46              # calculate gradient
47              grad = self._grad(X, y)
48              self.history["grad"].append(grad)
49
50              # do gradient step
51              self._theta -= alpha * grad
52
```

np.dot(X, self.theta)
WX, Perpendicular distance

Compute Probs

Compute Gradient

```
92      # computes scores for each class and each object in X
93      def _predict_raw(self, X):
94          # check whether X has appended bias feature or not
95          if X.shape[1] == len(self._theta):
96              scores = np.dot(X, self._theta)
97          else:
98              scores = np.dot(X, self.coef_) + self.intercept_
99          return scores
```

Logistic Regression

```
14      # calculate gradient of objective function
15      def _grad(self, X, y):
16          # number of training examples
17          n = X.shape[0]
18
19          # get scores for each class and example
20          # 2D matrix
21          scores = self._predict_raw(X)
22
23          # transform scores to probabilities
24          # softmax
25          exp_scores = np.exp(scores)
26          probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
27
28          # error
29          probs[range(n),y] -= 1
30
31          # gradient
32          gradient = np.dot(X.T, probs) / n
33
34          return gradient
35
36      # full gradient descent, i.e., not stochasti
37      def _gd(self, X, y, max_iter, alpha=0.05):
38          for i in range(max_iter):
39              self.history["coef"].append(self._theta[1:])
40              self.history["intercept"].append(self._theta[0])
41
42              metrics = self.score(X, y)
43              self.history["cost"].append(metrics["cost"])
44              self.history["acc"].append(metrics["acc"])
45
46              # calculate gradient
47              grad = self._grad(X, y)
48              self.history["grad"].append(grad)
49
50              # do gradient step
51              self._theta -= alpha * grad
52
```

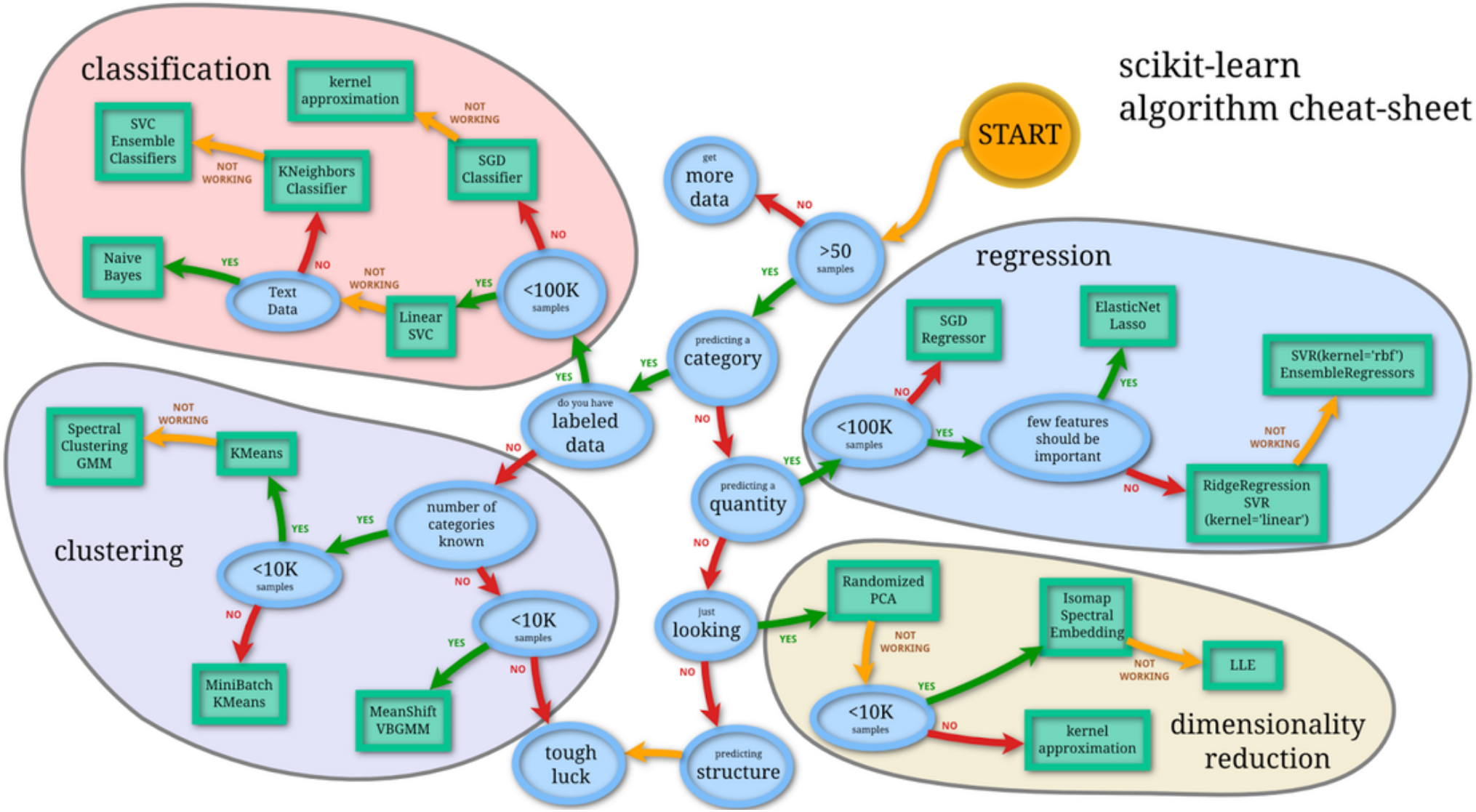
np.dot(X, self.theta)
WX, Perpendicular distance

Compute Probs

Compute Gradient

+ linalg.norm(self.coef_, 1) #L2 regularization

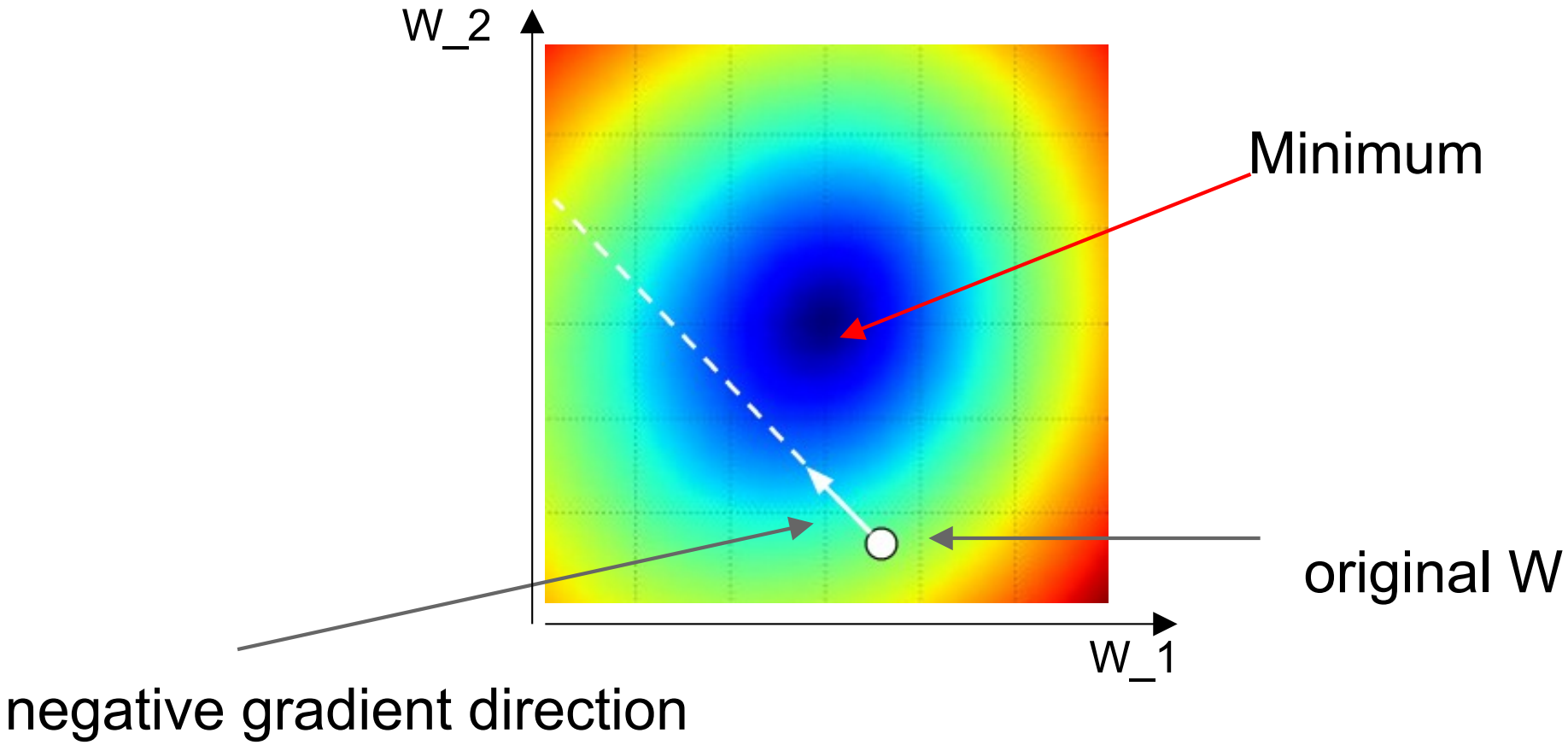
```
93      def _predict_raw(self, X):
94          # check whether X has appended bias feature or not
95          if X.shape[1] == len(self._theta):
96              scores = np.dot(X, self._theta)
97          else:
98              scores = np.dot(X, self.coef_) + self.intercept_
99          return scores
```


scikit-learn
algorithm cheat-sheet
START
classification
kernel approximation
NOT WORKING
SVC Ensemble Classifiers
NOT WORKING
KNeighbors Classifier
SGD Classifier
NO
Naive Bayes
YES
NO
Text Data
NOT WORKING
Linear SVC
YES
<100K samples
YES
get more data
NO
>50 samples
YES
regression
SGD Regressor
ElasticNet Lasso
SVR(kernel='rbf') EnsembleRegressors
NO
<100K samples
YES
few features should be important
YES
NOT WORKING
NO
RidgeRegression SVR (kernel='linear')
predicting a category
YES
NO
do you have labeled data
YES
NO
clustering
Spectral Clustering GMM
NOT WORKING
KMeans
YES
number of categories known
YES
NO
<10K samples
NO
MiniBatch KMeans
<10K samples
YES
NO
MeanShift VBGMM
predicting a quantity
YES
NO
just looking
YES
NO
Randomized PCA
NOT WORKING
Isomap Spectral Embedding
NOT WORKING
LLE
YES
<10K samples
NO
kernel approximation
dimensionality reduction
tough luck
predicting structure

Gradient Descent

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Negative gradient direction



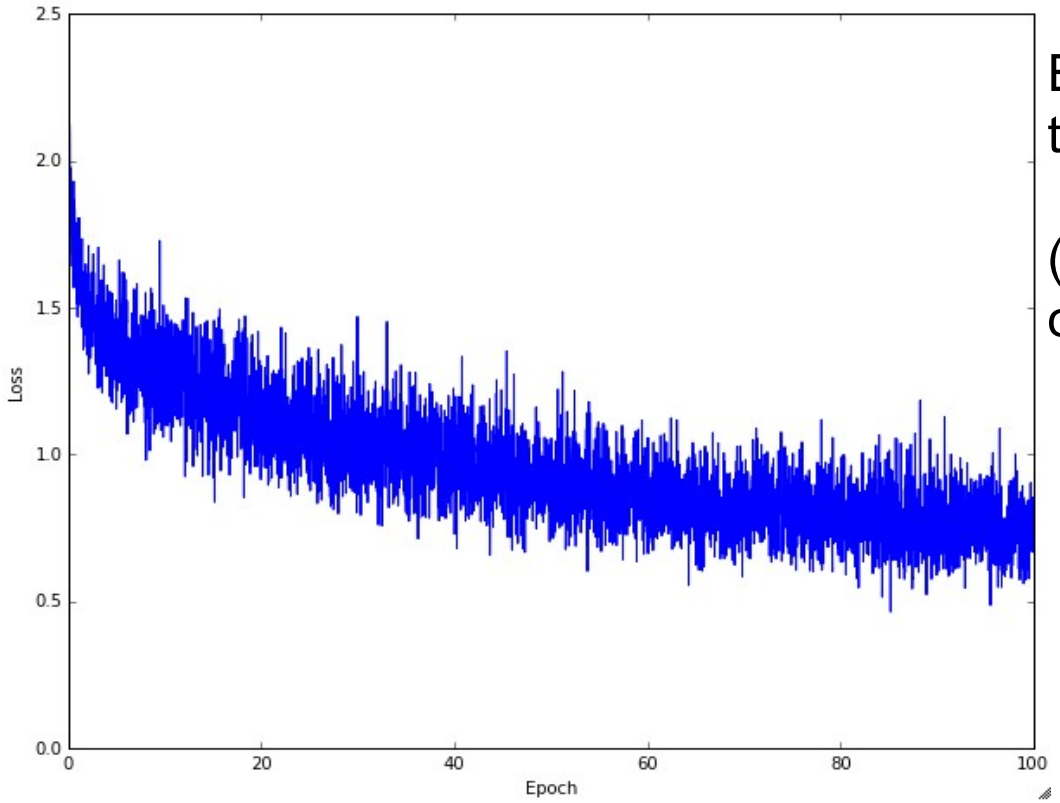
Mini-batch Gradient Descent

- **only use a small portion of the training set to compute the gradient.**

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

- **Common mini-batch sizes are 32/64/128 examples**
  - e.g. Krizhevsky ILSVRC ConvNet used 256 examples

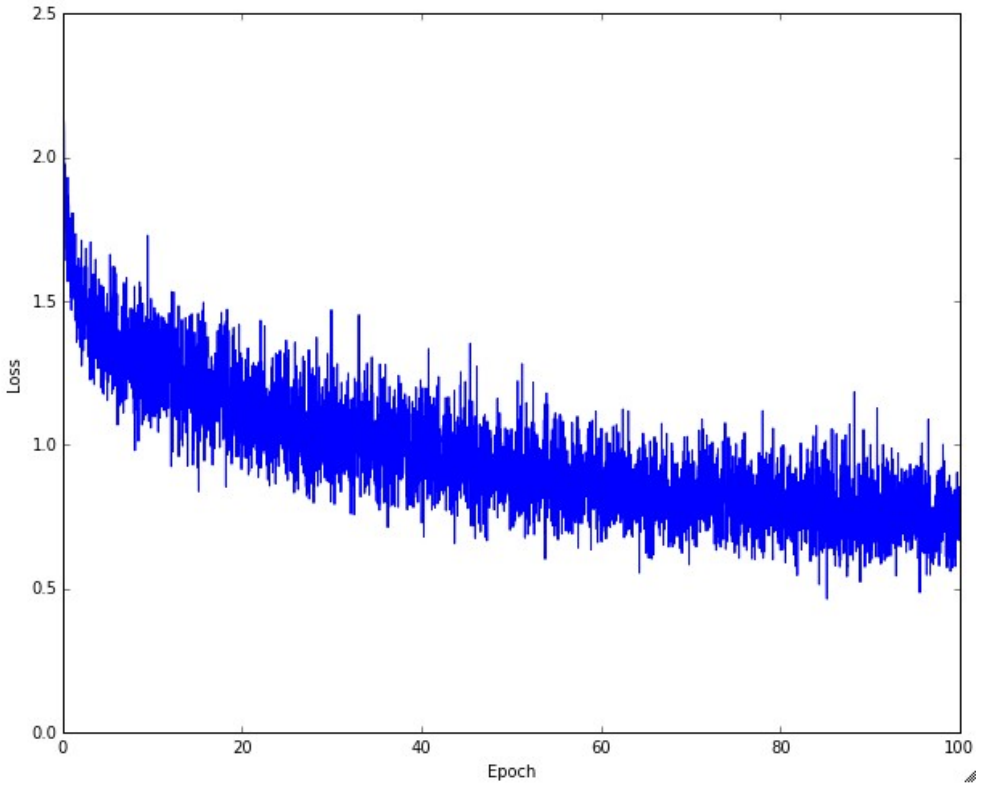
Mini-batch Gradient Descent

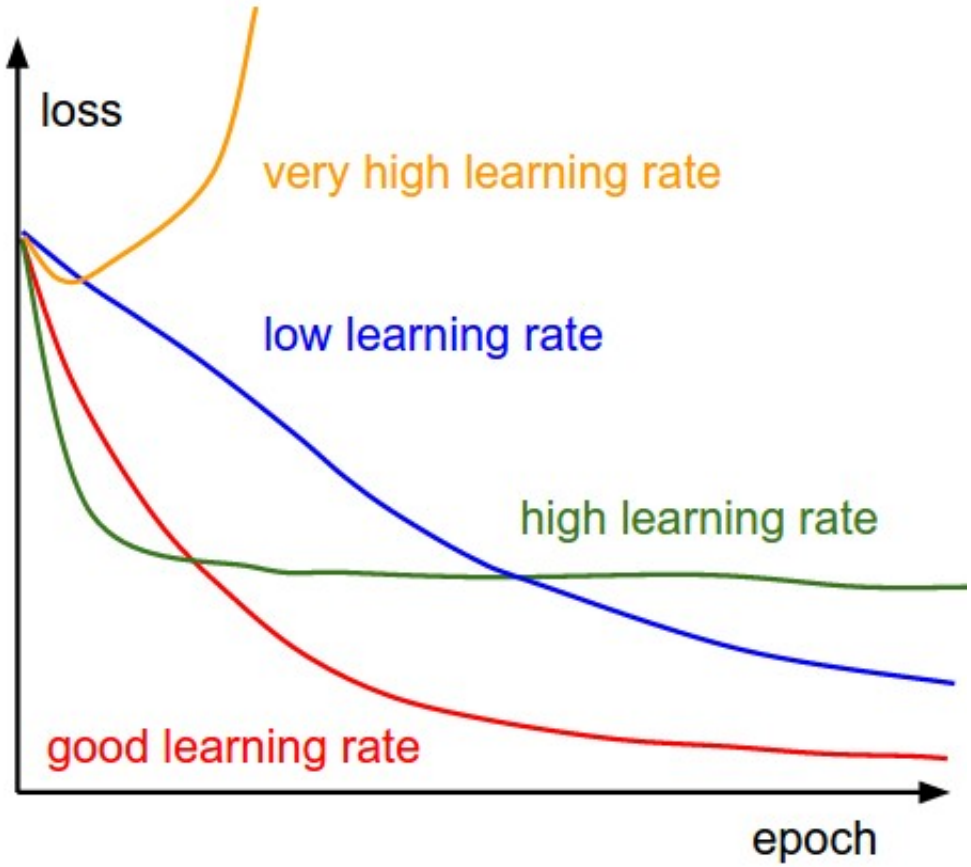


Example of optimization progress while training a neural network.

(Loss over mini-batches goes down over time.)

The effects of step size (or “learning rate”)





Mini-batch Gradient Descent

- **only use a small portion of the training set to compute the gradient.**

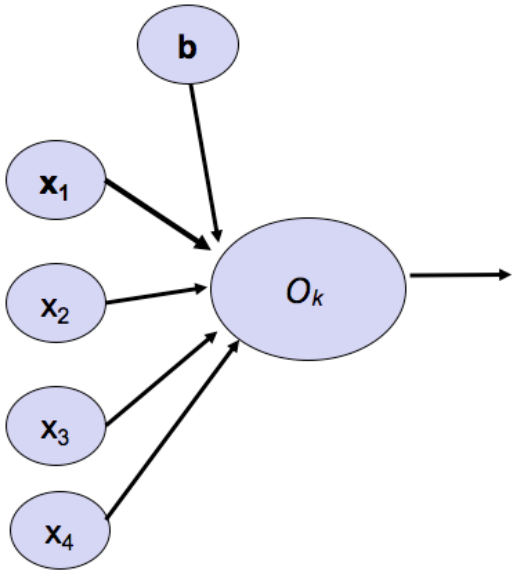
```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

more fancy update formulas include adding momentum, Adagrad, RMSProp, Adam, …

Outline

- Introduction
- Binomial logistic regression
- Multinomial Logistic regression
- Linear Classifier via separating hyperplane
- Multinomial Logistic regression Classifier
- Linear classifier demo
- Learning Softmax Classifiers via optimization
  - Implementation (SoftMax Classifier)
- Regressions in matrix terms and in graph terms
- Summary

Activation matrices and model matrices for a single target variable



4 1 1 Prediction

150 X 4 W = 150 P

W = W -α ∇_W = X E

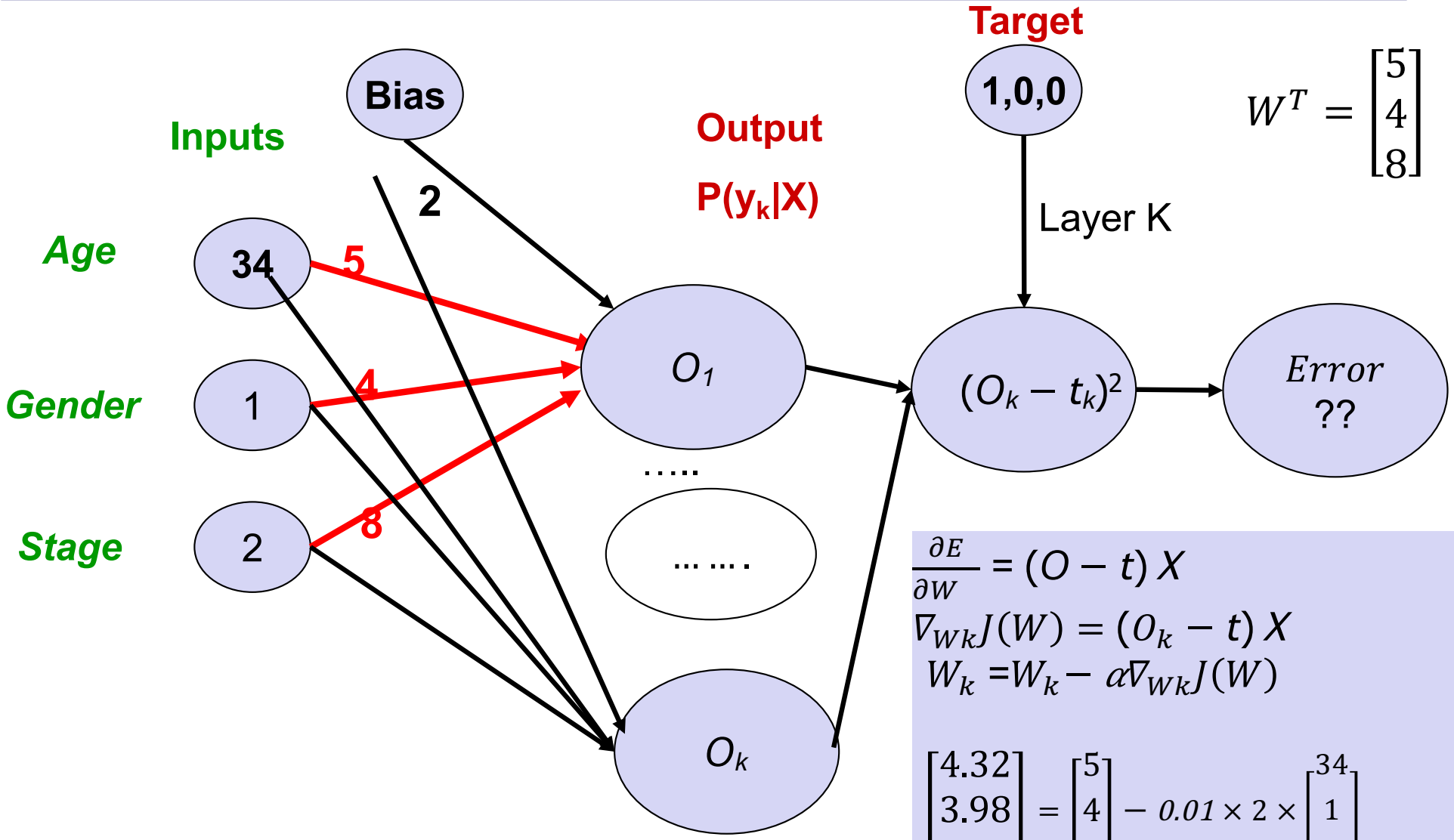
Weighted Sum

Model

Data

$$\begin{bmatrix} 4.32 \\ 3.98 \\ 7.92 \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \\ 8 \end{bmatrix} - 0.01 \times 2 \times \begin{bmatrix} 34 \\ 1 \\ 4 \end{bmatrix}$$

Multi-class Classifier



$$\frac{\partial E}{\partial W} = (O - t)\,X$$

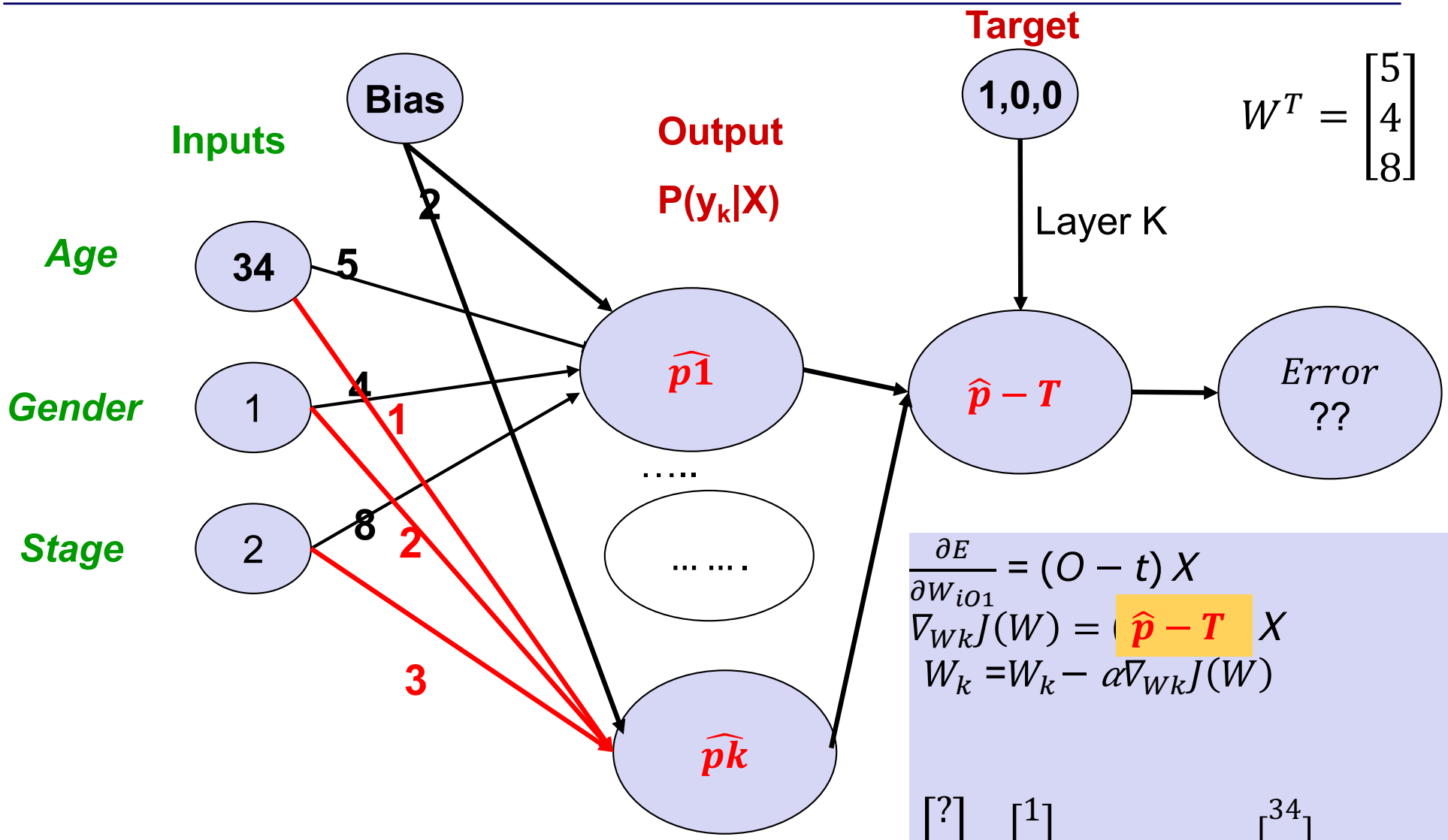
$$\nabla_{Wk} J(W) = (O_k - t)\,X$$

$$W_k = W_k - \alpha \nabla_{Wk} J(W)$$

$$\begin{bmatrix} 4.32 \\ 3.98 \\ 7.92 \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \\ 8 \end{bmatrix} - 0.01 \times 2 \times \begin{bmatrix} 34 \\ 1 \\ 4 \end{bmatrix}$$

$$\nabla_{Wk} J(W) = (O_k - t)\,X$$

Quiz-like questions: Multi-class classifier



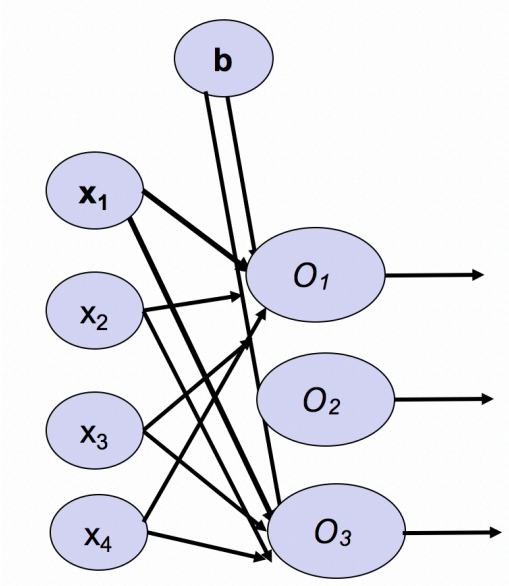
$$\frac{\partial E}{\partial W_{iO1}} = (O - t)\,X$$

$$\nabla_{Wk} J(W) = (\hat{p} - T)\,X$$

$$W_k = W_k - \alpha \nabla_{Wk} J(W)$$

$$\begin{bmatrix} ? \\ ? \\ ? \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} - 0.01 \times ?? \times \begin{bmatrix} 34 \\ 1 \\ 4 \end{bmatrix}$$

Activation matrices and model matrices: Multiple Targets



4 3 3 Class

150 X 4 W = 150 $\hat{p}$ T

.2 .7 .1 0 1 0
1 0 0

$\nabla_W = X^T(\hat{p} - T)$
$\nabla_W = X^T E$

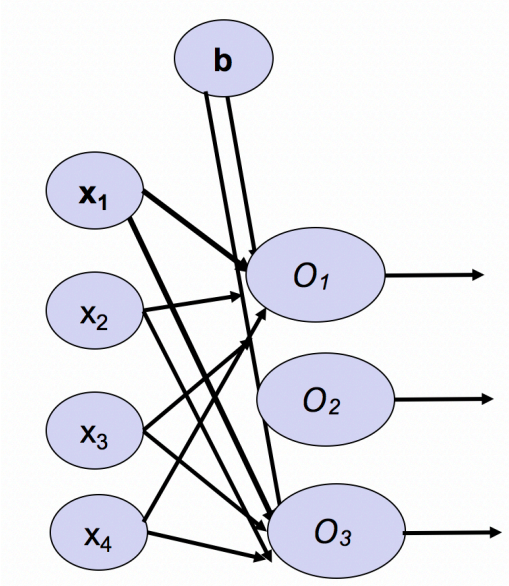
W = W -α ∇_W

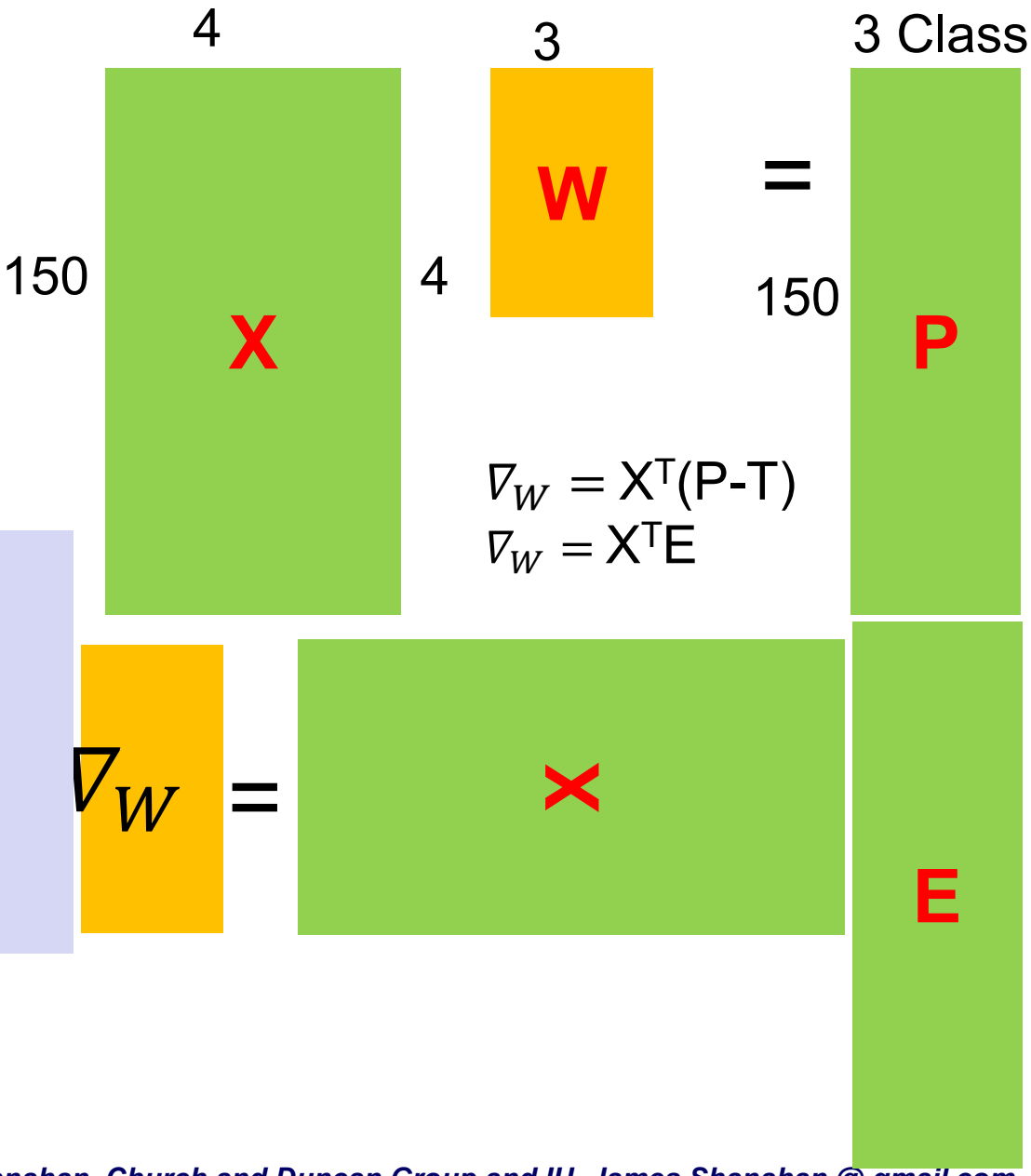
× E

∇_W

Model
Data

Activation matrices and model matrices: Multiple Targets






```
dW2 <- t(X) %*% deltaScores. #input layer
Browse[2]> dim(X)
[1]150  4
Browse[2]> dim(scores)
[1]150  3
Browse[2]> nablaW
[,1]         [,2]         [,3]
[1,] 0.000000000  0.0000000000  0.000000000
[2,] 0.007346364 -0.0017693039 -0.005577060
[3,] 0.014880768 -0.0032287482 -0.011652020
[4,] 0.000000000  0.0000000000  0.000000000
```

Model

Data

Outline

- Introduction
- Binomial logistic regression
- Multinomial Logistic regression
- Linear Classifier via separating hyperplane
- Multinomial Logistic regression Classifier
- Linear classifier demo
- Learning Softmax Classifiers via optimization
  - Implementation (SoftMax Classifier)
- Regressions in matrix terms and in graph terms
- Summary

End of lecture