

Loss Functions, Regularization and Optimization

- Loss Functions
- Regularization
- Optimization



source from <http://cs231n.stanford.edu>

Recall from last time: Linear Classifier



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

TODO:

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)

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

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

A **loss function** tells how good our current classifier is

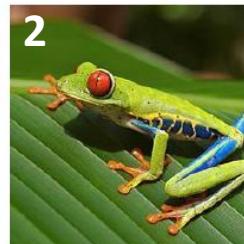
Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^N$$

Where x_i is image and
 y_i is (integer) label
0, 1, 2

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

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j \\ s_j - s_{y_i} & \text{otherwise} \end{cases}$$

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

A **loss function** tells how good our current classifier is

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^N$$

Where x_i is image and
 y_i is (integer) label

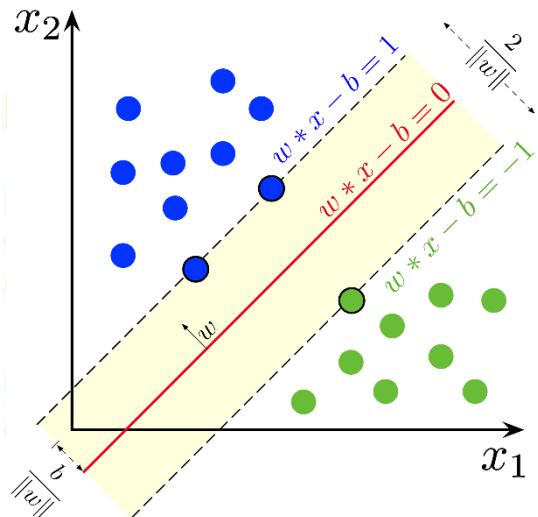
Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

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



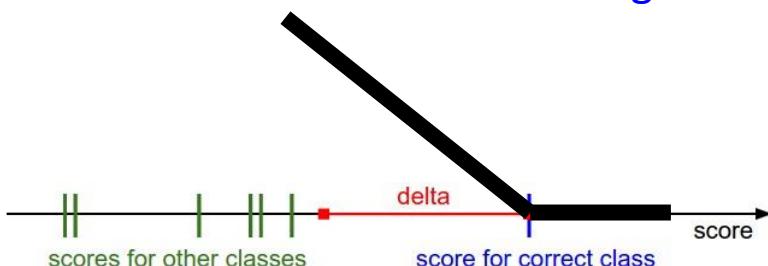
cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1



the SVM loss has the form:

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

hinge loss



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
Losses:	2.9		

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\begin{aligned} L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\ &= \max(0, 5.1 - 3.2 + 1) \\ &\quad + \max(0, -1.7 - 3.2 + 1) \\ &= \max(0, 2.9) + \max(0, -3.9) \\ &= 2.9 + 0 \\ &= 2.9 \end{aligned}$$

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
Losses:	2.9	0	

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\begin{aligned} L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\ &= \max(0, 1.3 - 4.9 + 1) \\ &\quad + \max(0, 2.0 - 4.9 + 1) \\ &= \max(0, -2.6) + \max(0, -1.9) \\ &= 0 + 0 \\ &= 0 \end{aligned}$$

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
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

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

$$\begin{aligned} &= \max(0, 2.2 - (-3.1) + 1) \\ &\quad + \max(0, 2.5 - (-3.1) + 1) \\ &= \max(0, 6.3) + \max(0, 6.6) \\ &= 6.3 + 6.6 \\ &= 12.9 \end{aligned}$$

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
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

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

Loss over full dataset is average:

$$L = \frac{1}{N} \sum_{i=1}^N L_i$$

$$\begin{aligned} L &= (2.9 + 0 + 12.9)/3 \\ &= \mathbf{5.27} \end{aligned}$$

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
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

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

Q: What happens to loss if car scores change a bit?

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
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

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

Q2: what is the min/max possible loss?

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
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

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

Q3: At initialization W is small so all $s \approx 0$. What is the loss?

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
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

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

Q4: What if the sum was over all classes? (including $j = y_i$)

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
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

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

Q5: What if we used mean instead of sum?

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
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

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

Q6: What if we used

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)^2$$

Multiclass SVM Loss: Example code

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

```
def L_i_vectorized(x, y, W):
    scores = W.dot(x)
    margins = np.maximum(0, scores - scores[y] + 1)
    margins[y] = 0
    loss_i = np.sum(margins)
    return loss_i
```

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



cat	3.2
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)



Want to interpret raw classifier scores as **probabilities**

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

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

Softmax
Function

cat	3.2
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)



Want to interpret raw classifier scores as **probabilities**

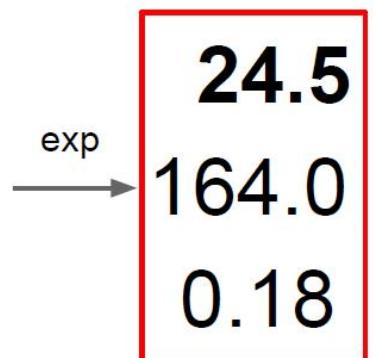
$$s = f(x_i; W)$$

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

Softmax Function

Probabilities
must be ≥ 0

cat	3.2
car	5.1
frog	-1.7



unnormalized
probabilities

Softmax Classifier (Multinomial Logistic Regression)



Want to interpret raw classifier scores as **probabilities**

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

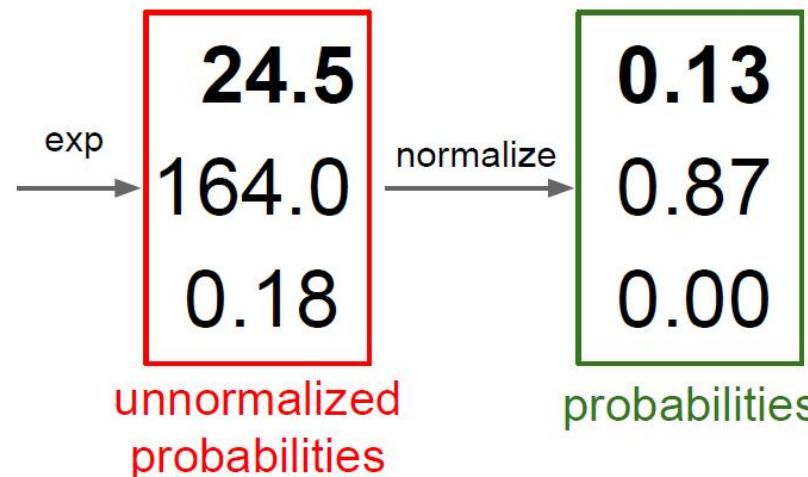
$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

Probabilities
must be ≥ 0

Probabilities
must sum to 1

cat	3.2
car	5.1
frog	-1.7



Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



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

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

Softmax Function

Probabilities
must be ≥ 0

Probabilities
must sum to 1

cat
car
frog

3.2
5.1
-1.7

exp

24.5
164.0
0.18

normalize

0.13
0.87
0.00

probabilities

Unnormalized
log-probabilities / logits

unnormalized
probabilities

Softmax Classifier (Multinomial Logistic Regression)



Want to interpret raw classifier scores as **probabilities**

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

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

Softmax Function

cat
car
frog

3.2
5.1
-1.7

Unnormalized
log-probabilities / logits

exp

24.5
164.0
0.18

unnormalized
probabilities

normalize

0.13
0.87
0.00

probabilities

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

$$\rightarrow L_i = -\log(0.13) \\ = 0.89$$

Softmax Classifier (Multinomial Logistic Regression)



Want to interpret raw classifier scores as **probabilities**

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

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

Softmax Function

Probabilities
must be ≥ 0

Probabilities
must sum to 1

cat
car
frog

3.2
5.1
-1.7

exp

24.5
164.0
0.18

normalize

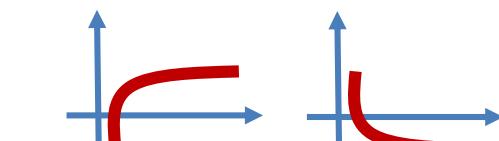
0.13
0.87
0.00

probabilities

Unnormalized
log-probabilities / logits

unnormalized
probabilities

Maximum Likelihood Estimation
Choose weights to maximize the likelihood of the observed data



Softmax Classifier (Multinomial Logistic Regression)



Want to interpret raw classifier scores as **probabilities**

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

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

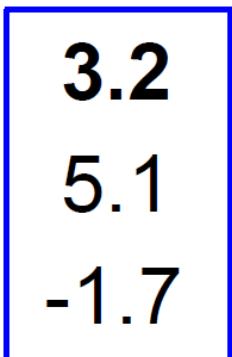
Softmax Function

Probabilities
must be ≥ 0

Probabilities
must sum to 1

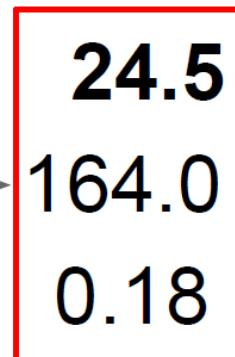
$$L_i = -\log P(Y = y_i|X = x_i)$$

cat
car
frog



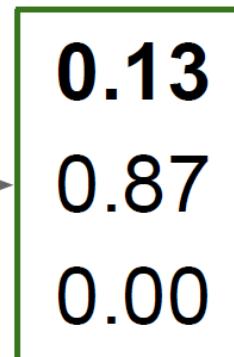
Unnormalized
log-probabilities / logits

exp



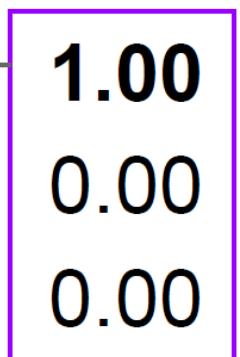
unnormalized
probabilities

normalize



probabilities

→ compare ←



Correct
probs

Softmax Classifier (Multinomial Logistic Regression)



Want to interpret raw classifier scores as **probabilities**

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

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

Softmax Function

cat
car
frog

3.2
5.1
-1.7

Unnormalized
log-probabilities / logits

exp

24.5
164.0
0.18

unnormalized
probabilities

normalize

0.13
0.87
0.00

probabilities

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

compare

1.00
0.00
0.00

Kullback–Leibler divergence

$$D_{KL}(P||Q) = \sum_y P(y) \log \frac{P(y)}{Q(y)}$$

Correct
probs

Softmax Classifier (Multinomial Logistic Regression)



Want to interpret raw classifier scores as **probabilities**

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

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

Softmax Function

cat
car
frog

3.2
5.1
-1.7

Unnormalized
log-probabilities / logits

exp

24.5
164.0
0.18

unnormalized
probabilities

normalize

0.13
0.87
0.00

probabilities

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

compare

1.00
0.00
0.00

Kullback–Leibler divergence

$$D_{KL}(P\|Q) = \sum_y P(y) \log \frac{P(y)}{Q(y)}$$

Correct
probs

$$D_{KL} = P \log P - P \log Q$$

Softmax Classifier (Multinomial Logistic Regression)



Want to interpret raw classifier scores as **probabilities**

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

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

Softmax Function

Probabilities
must be ≥ 0

Probabilities
must sum to 1

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

cat
car
frog

3.2
5.1
-1.7

exp

24.5
164.0
0.18

normalize

0.13
0.87
0.00

probabilities

$$-H(p)$$

$$H(P, Q)$$

Unnormalized
log-probabilities / logits

unnormalized
probabilities

1.00
0.00
0.00

Correct
probs

$$D_{KL} = P \log P - P \log Q$$

Softmax Classifier (Multinomial Logistic Regression)



Want to interpret raw classifier scores as **probabilities**

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

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

Softmax Function

cat
car
frog

3.2
5.1
-1.7

Unnormalized
log-probabilities / logits

exp

24.5
164.0
0.18

unnormalized
probabilities

normalize

0.13
0.87
0.00

probabilities

$-H(p)$

$H(P, Q)$

$D_{KL} = P \log P - P \log Q$

1.00
0.00
0.00

Correct
probs

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

Probabilities
must sum to 1

compare

Cross Entropy

$$H(P, Q) = H(p) + D_{KL}(P||Q)$$

Softmax Classifier (Multinomial Logistic Regression)



Want to interpret raw classifier scores as **probabilities**

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

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

Softmax Function

Maximize probability of correct class

Putting it all together:

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

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

cat	3.2
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)



Want to interpret raw classifier scores as **probabilities**

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

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

Softmax Function

Maximize probability of correct class

3.2

cat

5.1

car

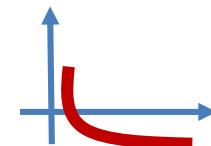
-1.7

frog

Putting it all together:

$$L_i = -\log P(Y = y_i|X = x_i)$$
$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

Q: What is the min/max possible loss L_i ?



Softmax Classifier (Multinomial Logistic Regression)



Want to interpret raw classifier scores as **probabilities**

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

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

Softmax Function

Maximize probability of correct class

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

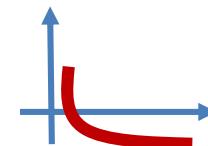
Putting it all together:

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

cat	3.2
car	5.1
frog	-1.7

Q: What is the min/max possible loss L_i ?

A: min 0, max infinity



Softmax Classifier (Multinomial Logistic Regression)



Want to interpret raw classifier scores as **probabilities**

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

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

Softmax Function

Maximize probability of correct class

Putting it all together:

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

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

cat	3.2
car	5.1
frog	-1.7

Q2: At initialization all s will be approximately equal; what is the loss?

Softmax Classifier (Multinomial Logistic Regression)



Want to interpret raw classifier scores as **probabilities**

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

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

Softmax Function

Maximize probability of correct class

Putting it all together:

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

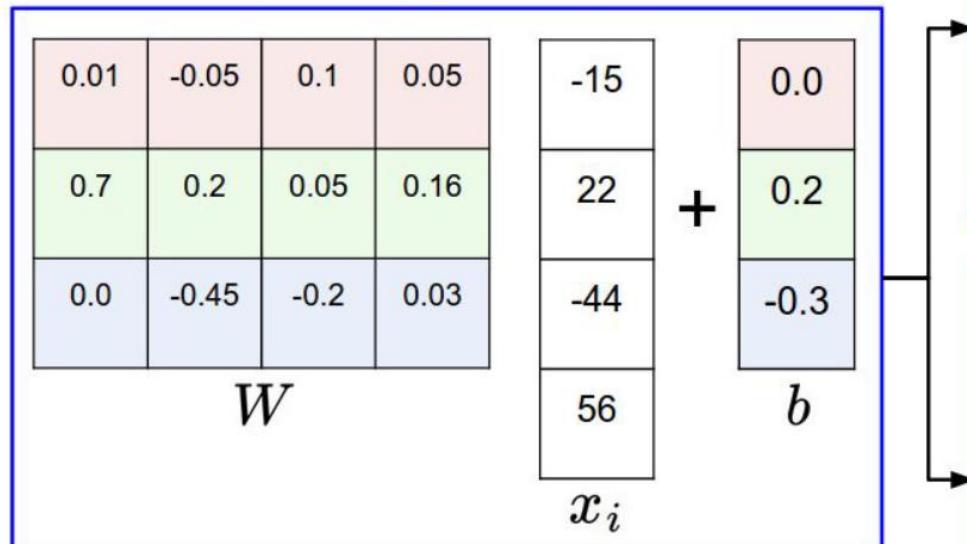
$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

cat	3.2
car	5.1
frog	-1.7

Q2: At initialization all s will be approximately equal; what is the loss?
A: $\log(C)$, eg $\log(10) \approx 2.3$

Softmax vs. SVM

matrix multiply + bias offset



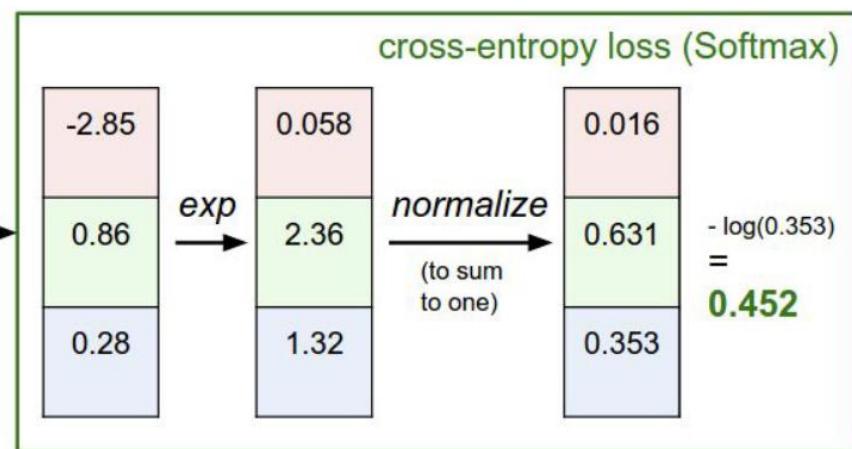
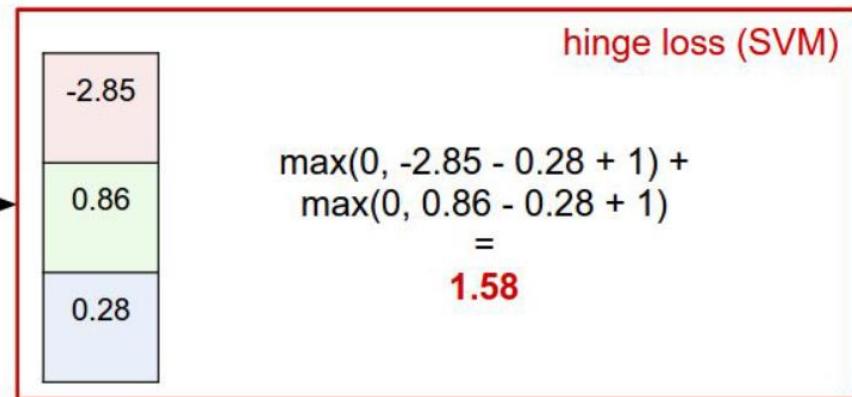
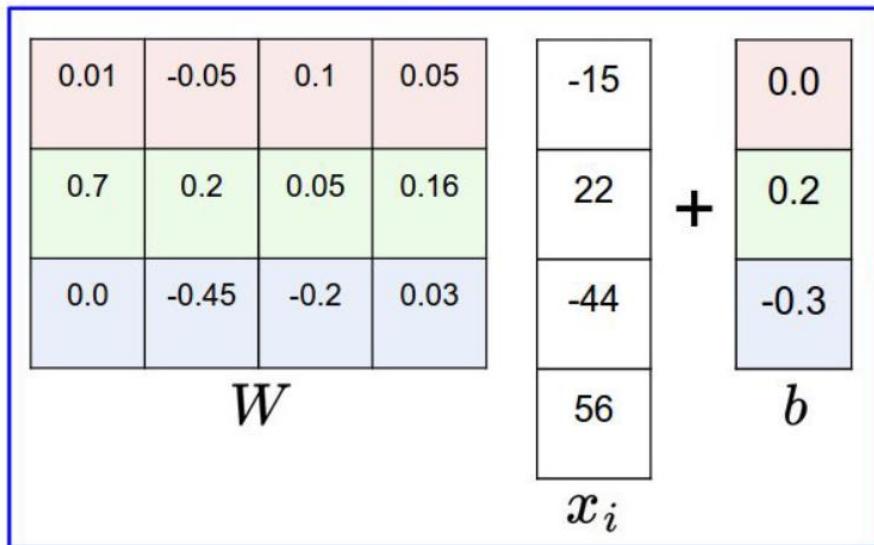
hinge loss (SVM)

$$\begin{aligned} & \max(0, -2.85 - 0.28 + 1) + \\ & \max(0, 0.86 - 0.28 + 1) \\ & = \\ & \mathbf{1.58} \end{aligned}$$

y_i 2

Softmax vs. SVM

matrix multiply + bias offset



Cross-Entropy vs SVM Loss

$$L_i = -\log \left(\frac{\exp(s_{y_i})}{\sum_j \exp(s_j)} \right)$$

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

assume scores:
[10, -2, 3]

[10, 9, 9]

[10, -100, -100]

and $y_i = 0$

Q: What is cross-entropy loss?
What is SVM loss?

Cross-Entropy vs SVM Loss

$$L_i = -\log \left(\frac{\exp(s_{y_i})}{\sum_j \exp(s_j)} \right)$$

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

assume scores:

[10, -2, 3]

[10, 9, 9]

[10, -100, -100]

and $y_i = 0$

Q: What is cross-entropy loss?
What is SVM loss?

A: Cross-entropy loss > 0
SVM loss = 0

Cross-Entropy vs SVM Loss

$$L_i = -\log \left(\frac{\exp(s_{y_i})}{\sum_j \exp(s_j)} \right)$$

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

assume scores:

[10, -2, 3]

[10, 9, 9]

[10, -100, -100]

and $y_i = 0$

Q: What happens to each loss if I slightly change the scores of the last datapoint?

Cross-Entropy vs SVM Loss

$$L_i = -\log \left(\frac{\exp(s_{y_i})}{\sum_j \exp(s_j)} \right)$$

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

assume scores:

[10, -2, 3]

[10, 9, 9]

[10, -100, -100]

and

$$y_i = 0$$

Q: What happens to each loss if I slightly change the scores of the last datapoint?

A: Cross-entropy loss will change;
SVM loss will stay the same

Cross-Entropy vs SVM Loss

$$L_i = -\log \left(\frac{\exp(s_{y_i})}{\sum_j \exp(s_j)} \right)$$

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

assume scores:

[10, -2, 3]

[10, 9, 9]

[10, -100, -100]

and $y_i = 0$

Q: What happens to each loss if I double the score of the correct class from 10 to 20?

Cross-Entropy vs SVM Loss

$$L_i = -\log \left(\frac{\exp(s_{y_i})}{\sum_j \exp(s_j)} \right)$$

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

assume scores:

[10, -2, 3]

[10, 9, 9]

[10, -100, -100]

and

$$y_i = 0$$

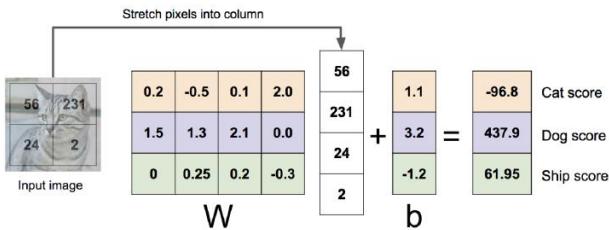
Q: What happens to each loss if I double the score of the correct class from 10 to 20?

A: Cross-entropy loss will decrease,
SVM loss still 0

Recap: Three ways to think about linear classifiers

Algebraic Viewpoint

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



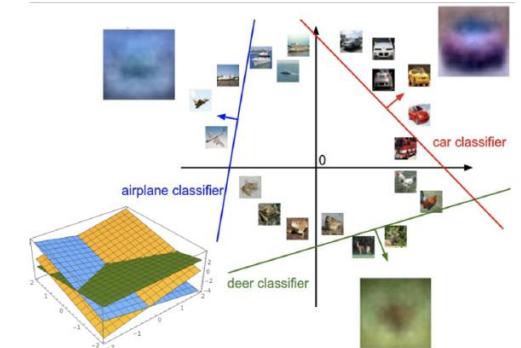
Visual Viewpoint

One template per class



Geometric Viewpoint

Hyperplanes cutting up space



Recap: Loss Functions quantify preferences

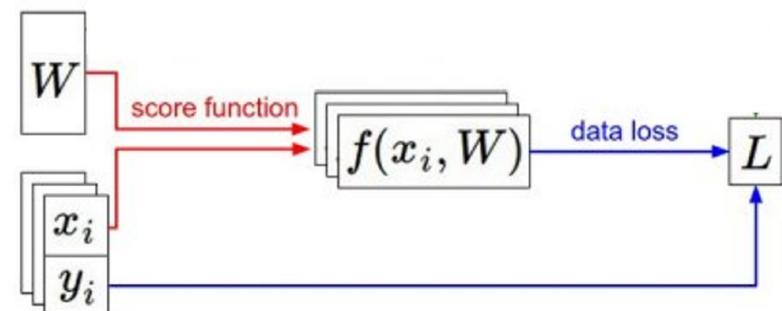
- We have some dataset of (x, y)
- We have a **score function**:
- We have a **loss function**:

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

Linear classifier

Softmax: $L_i = -\log \left(\frac{\exp(s_{y_i})}{\sum_j \exp(s_j)} \right)$

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



Recap: Loss Functions quantify preferences

- We have some dataset of (x, y)
- We have a **score function**:
- We have a **loss function**:

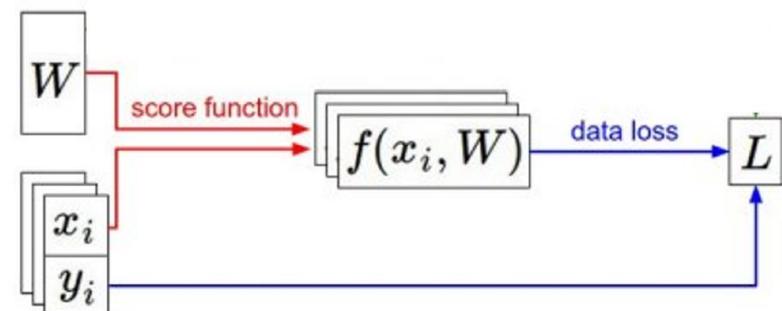
Q: How do we find the best W, b ?

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

Linear classifier

Softmax: $L_i = -\log \left(\frac{\exp(s_{y_i})}{\sum_j \exp(s_j)} \right)$

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



Regularization

Last Time: Loss Functions quantify preferences

- We have some dataset of (x, y)
- We have a **score function**:
- We have a **loss function**:

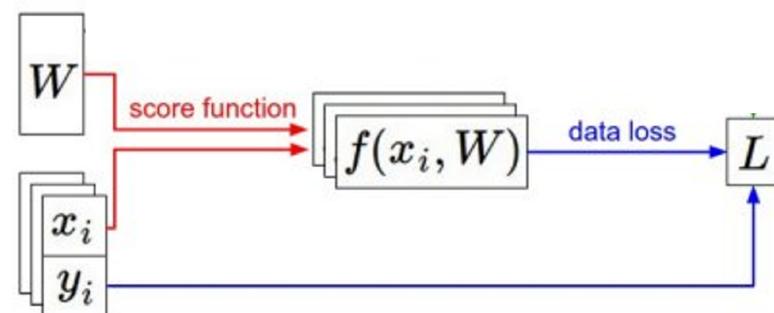
Problem: Loss functions encourage good performance on training data but we really care about test data

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

Linear classifier

Softmax: $L_i = -\log \left(\frac{\exp(s_{y_i})}{\sum_j \exp(s_j)} \right)$

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



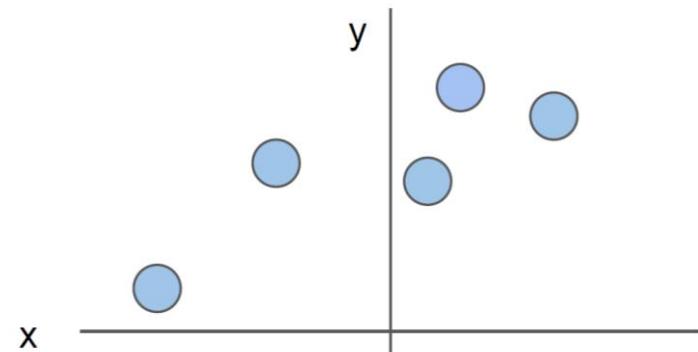
Overfitting

A model is **overfit** when it performs too well on the training data, and has poor performance for unseen data

Overfitting

A model is **overfit** when it performs too well on the training data, and has poor performance for unseen data

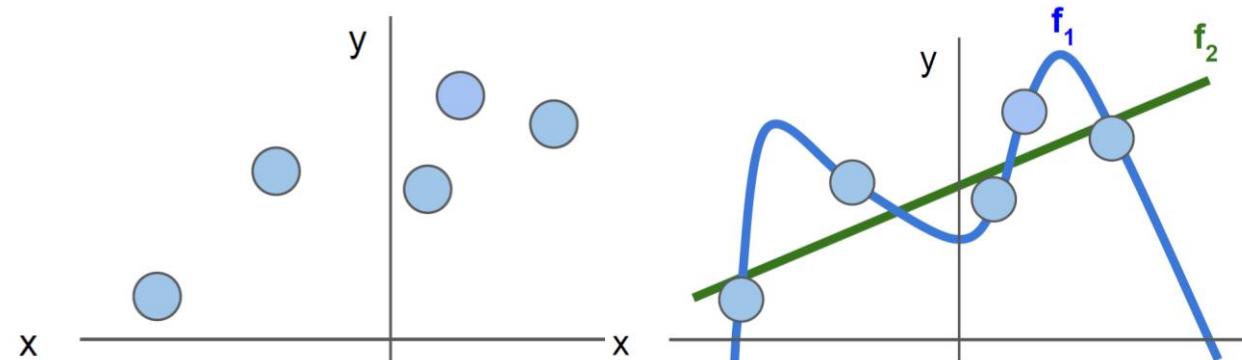
Example



Overfitting

A model is **overfit** when it performs too well on the training data, and has poor performance for unseen data

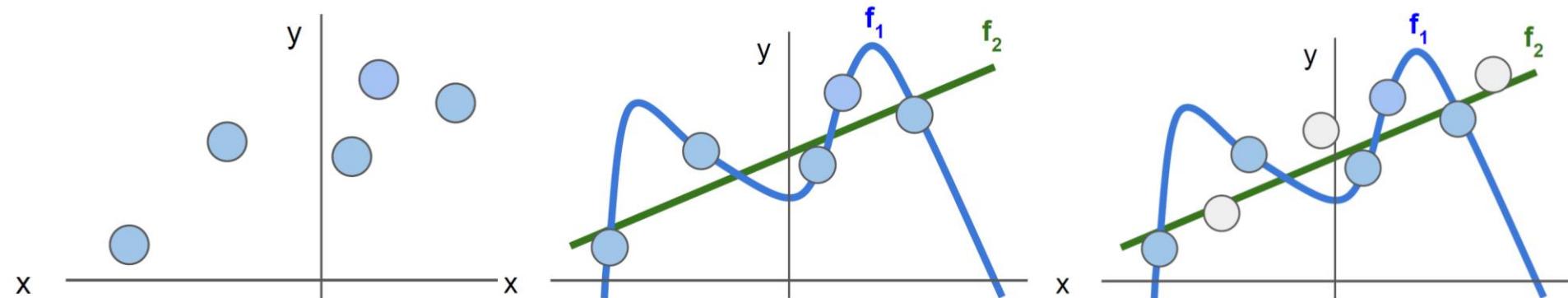
Example



Overfitting

A model is **overfit** when it performs too well on the training data, and has poor performance for unseen data

Example



Overfitting

A model is **overfit** when it performs too well on the training data, and has poor performance for unseen data

Example: Linear classifier with 1D inputs, 2 classes, softmax loss

$$s_i = w_i x + b_i$$

$$p_i = \frac{\exp(s_i)}{\exp(s_1) + \exp(s_2)}$$

$$L = -\log(p_y)$$

Overfitting

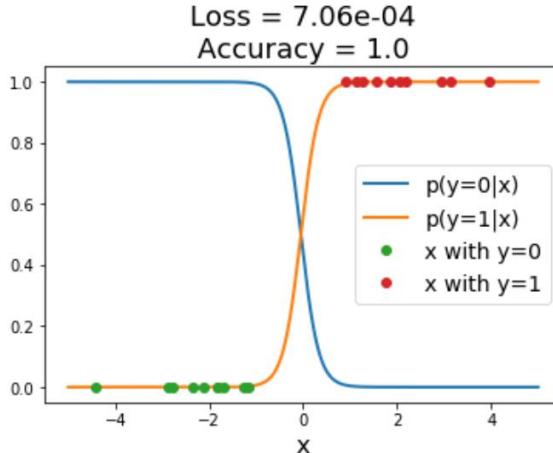
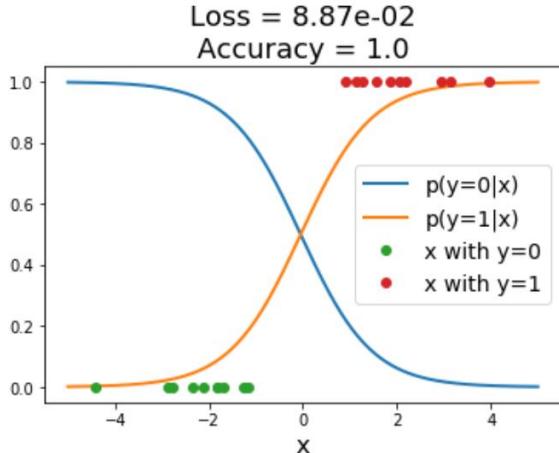
A model is **overfit** when it performs too well on the training data, and has poor performance for unseen data

Example: Linear classifier with 1D inputs, 2 classes, softmax loss

$$s_i = w_i x + b_i$$

$$p_i = \frac{\exp(s_i)}{\exp(s_1) + \exp(s_2)}$$

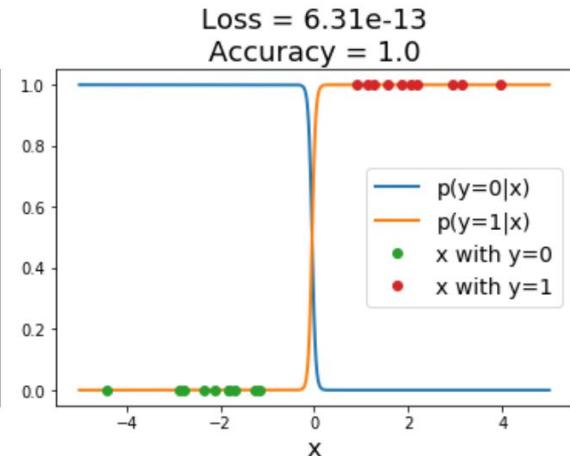
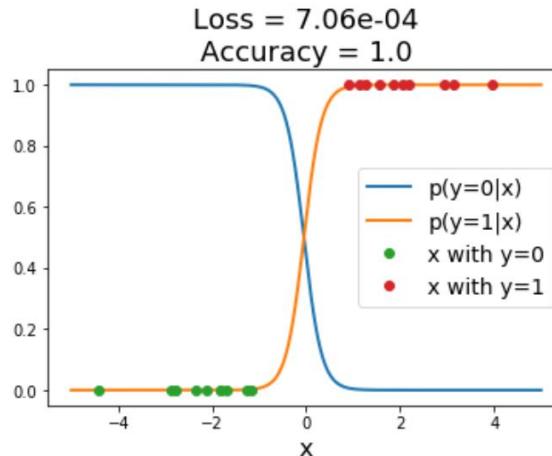
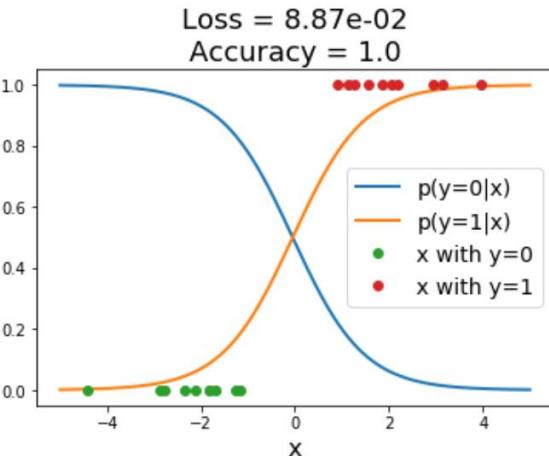
$$L = -\log(p_y)$$



Both models have perfect accuracy on train data!

Overfitting

A model is **overfit** when it performs too well on the training data, and has poor performance for unseen data



Both models have perfect accuracy on train data!

Example: Linear classifier with 1D inputs, 2 classes, softmax loss

$$s_i = w_i x + b_i$$

$$p_i = \frac{\exp(s_i)}{\exp(s_1) + \exp(s_2)}$$

$$L = -\log(p_y)$$

Low loss, but unnatural “cliff” between training points

Regularization: Beyond Training Error

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)$$

Data loss: Model predictions
should match training data

Regularization: Beyond Training Error

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}} + \underbrace{\lambda R(W)}_{\text{Regularization: Prevent the model from doing too well on training data}}$$

λ is a hyperparameter giving regularization strength

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Regularization: Beyond Training Error

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i) + \lambda R(W)$$

Data loss: Model predictions should match training data

λ is a hyperparameter giving regularization strength

Regularization: Prevent the model from doing *too well* on training data

Simple examples

L2 regularization: $R(W) = \sum_{k,l} W_{k,l}^2$

L1 regularization: $R(W) = \sum_{k,l} |W_{k,l}|$

Regularization: Beyond Training Error

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i) + \lambda R(W)$$

Data loss: Model predictions should match training data

λ is a hyperparameter giving regularization strength

Regularization: Prevent the model from doing *too well* on training data

Simple examples

L2 regularization: $R(W) = \sum_{k,l} W_{k,l}^2$

L1 regularization: $R(W) = \sum_{k,l} |W_{k,l}|$

More complex:

Dropout

Batch normalization

Cutout, Mixup, Stochastic depth, etc...

Regularization: Prefer Simpler Models

Example: Linear classifier with 1D inputs, 2 classes, softmax loss

$$s_i = w_i x + b_i \quad p_i = \frac{\exp(s_i)}{\exp(s_1) + \exp(s_2)}$$

$$L = -\log(p_y) + \lambda \sum_i w_i^2$$

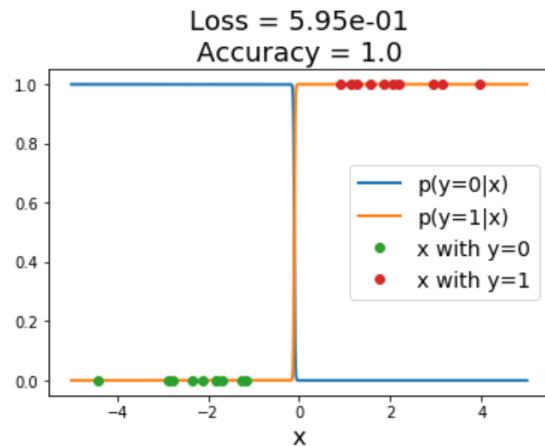
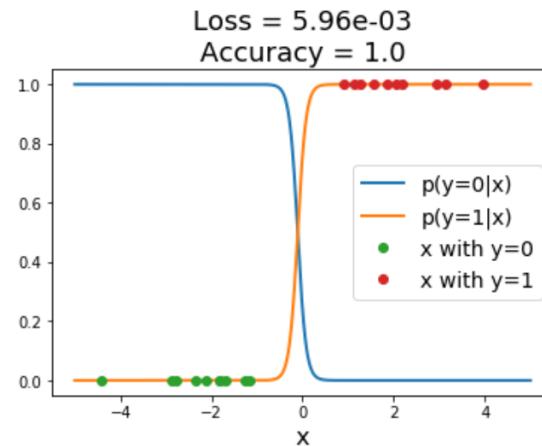
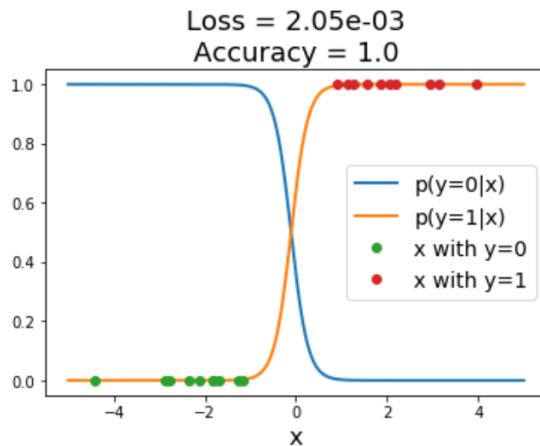
Regularization: Prefer Simpler Models

Example: Linear classifier with 1D inputs, 2 classes, softmax loss

$$s_i = w_i x + b_i \quad p_i = \frac{\exp(s_i)}{\exp(s_1) + \exp(s_2)}$$

$$L = -\log(p_y) + \lambda \sum_i w_i^2$$

Regularization term causes loss to increase for model with sharp cliff



Regularization: Expressing Preferences

L2 Regularization

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$R(W) = \sum_{k,l} W_{k,l}^2$$

$$w_1^T x = w_2^T x = 1$$

Same predictions, so data loss will always be the same

Regularization: Expressing Preferences

L2 Regularization

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$R(W) = \sum_{k,l} W_{k,l}^2$$

L2 regularization prefers weights to be “spread out”

$$w_1^T x = w_2^T x = 1$$

Same predictions, so data loss will always be the same

Finding a good W

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i) + \lambda R(W)$$

Loss function consists of **data loss** to fit the training data and **regularization** to prevent overfitting

Optimization

$$w^* = \arg \min_w L(w)$$

Optimization



[This image](#) is CC0 1.0 public domain



[Walking man image is CC0 1.0 public domain](#)

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)
```

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

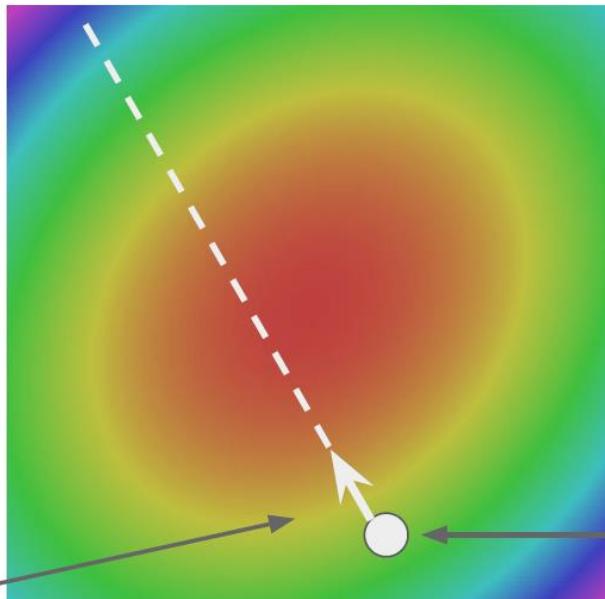
```
# Assume X_test is [3073 x 10000], Y_test [10000 x 1]
scores = Wbest.dot(Xte_cols) # 10 x 10000, the class scores for all test examples
# find the index with max score in each column (the predicted class)
Yte_predict = np.argmax(scores, axis = 0)
# and calculate accuracy (fraction of predictions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

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

Strategy #2: Follow the slope



w_2

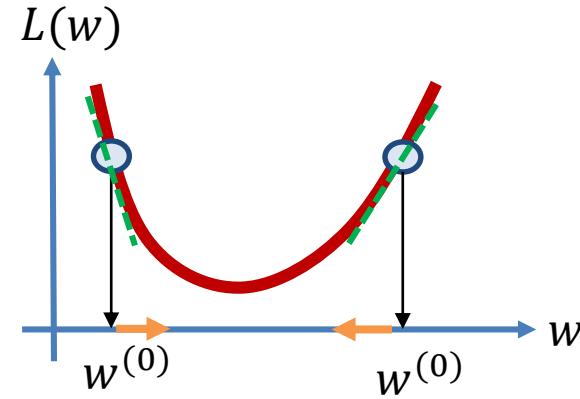


original W

negative gradient direction

Gradient Descent

$$w^{(t+1)} = w^{(t)} - \alpha \nabla_w L(x; w)$$



Gradient Descent

$$w^{(t+1)} = w^{(t)} - \alpha \nabla_w L(x; w)$$

```
# Vanilla Gradient Descent

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

Idea #2: Follow the slope

In 1-dimension, the **derivative** of a function gives the slope:

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

Idea #2: Follow the slope

In 1-dimension, the **derivative** of a function gives the slope:

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

In multiple dimensions, the **gradient** is the vector
of (partial derivatives) along each dimension

The slope in any direction is the **dot product** of the direction with the gradient
The direction of steepest descent is the **negative 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:

[?,
?,
?,
?,
?,
?,
?,
?,
?,
?,...]

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:

[?,
?,
?,
?,
?,
?,
?,
?,
?,...]

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,
?,
?,

$$\frac{(1.25322 - 1.25347)}{0.0001} = -2.5$$

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

?,
?,...]

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 (second dim):

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

loss 1.25353

gradient dW:

[-2.5,
?,
?,
?,
?,
?,
?,
?,
?,
?,...]

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 (second dim):

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

loss 1.25353

gradient dW:

[-2.5,
0.6,
?,
?,
?,
?,
?,
?,
?,
?]

$$\frac{(1.25353 - 1.25347)}{0.0001} = 0.6$$

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

?,...]

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 (third dim):

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

loss 1.25347

gradient dW:

[-2.5,
0.6,
?,
?,
?,
?,
?,
?,
?,
?,...]

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 (third dim):

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

loss 1.25347

gradient dW:

[-2.5,
0.6,
0,
?,
?

$$\frac{(1.25347 - 1.25347)}{0.0001} = 0$$

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

?, ...]

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 (third dim):

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

loss 1.25347

gradient dW:

[-2.5,
0.6,
0,
?,
?

?,...]

Numeric Gradient

- Slow! Need to loop over all dimensions
- Approximate

This is silly. The loss is just a function of W :

$$L = \frac{1}{N} \sum_{i=1}^N L_i + \sum_k W_k^2$$

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

$$s = f(x; W) = Wx$$

want $\nabla_W L$

This is silly. The loss is just a function of W:

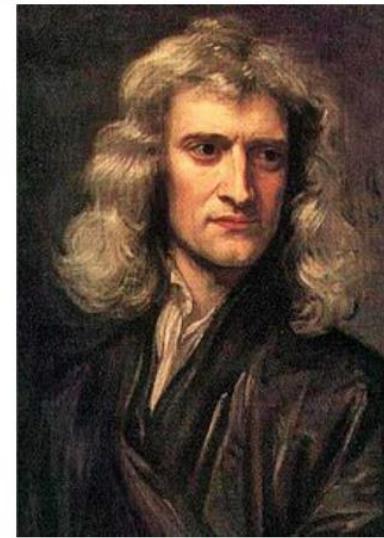
$$L = \frac{1}{N} \sum_{i=1}^N L_i + \sum_k W_k^2$$

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

$$s = f(x; W) = Wx$$

want $\nabla_W L$

Use calculus to compute an analytic gradient



[This image](#) is in the public domain



[This image](#) is in the public domain

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:

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

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



Computing Gradients

- **Numeric gradient**: approximate, slow, easy to write
- **Analytic gradient**: exact, fast, error-prone

In practice: Always use analytic gradient, but check implementation with numerical gradient. This is called a **gradient check**.

```
def grad_check_sparse(f, x, analytic_grad, num_checks=10, h=1e-7):
    """
    sample a few random elements and only return numerical
    in this dimensions.
    """

```

Computing Gradients

- **Numeric gradient:** approximate, slow, easy to write
- **Analytic gradient:** exact, fast, error-prone

```
torch.autograd.gradcheck(func, inputs, eps=1e-06, atol=1e-05, rtol=0.001,  
raise_exception=True, check_sparse_nnz=False, nondet_tol=0.0)
```

[SOURCE] ↗

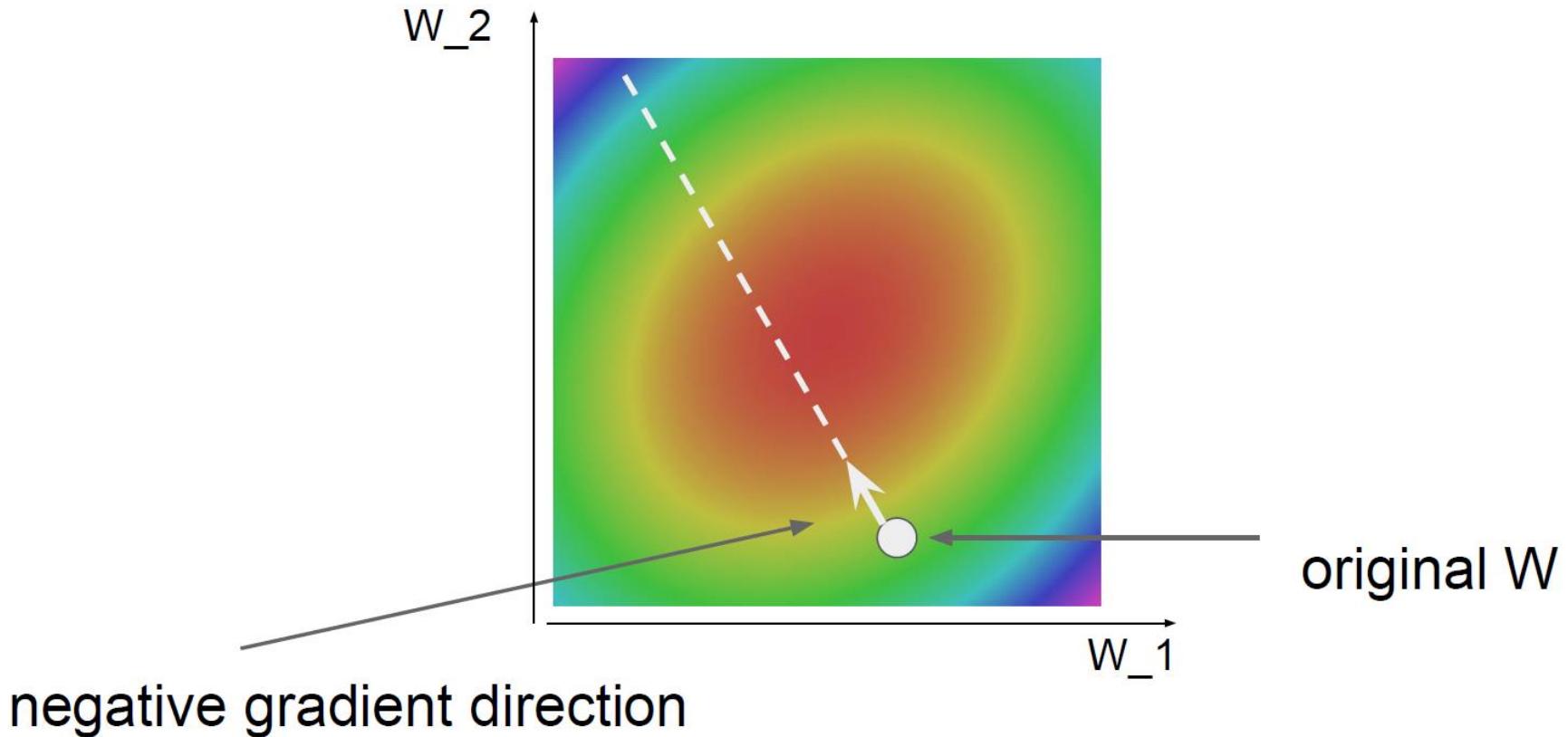
Check gradients computed via small finite differences against analytical gradients w.r.t. tensors in `inputs` that are of floating point type and with `requires_grad=True`.

The check between numerical and analytical gradients uses `allclose()`.

Gradient Descent

```
# Vanilla Gradient Descent

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



Batch Gradient Descent

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W) + \lambda R(W)$$

Full sum expensive
when N is large!

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Stochastic Gradient Descent (SGD)

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W) + \lambda R(W)$$

Full sum expensive
when N is large!

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Approximate sum
using a **minibatch** of
examples
32 / 64 / 128 common

```
# 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
```

Stochastic Gradient Descent (SGD)

$$\begin{aligned} L(W) &= \mathbb{E}_{(x,y) \sim p_{data}} [L(x, y, W)] + \lambda R(W) \\ &\approx \frac{1}{N} \sum_{i=1}^N L(x_i, y_i, W) + \lambda R(W) \end{aligned}$$

Think of loss as an expectation over the full **data distribution** p_{data}

Approximate expectation via sampling

Stochastic Gradient Descent (SGD)

$$\begin{aligned} L(W) &= \mathbb{E}_{(x,y) \sim p_{data}} [L(x, y, W)] + \lambda R(W) \\ &\approx \frac{1}{N} \sum_{i=1}^N L(x_i, y_i, W) + \lambda R(W) \end{aligned}$$

Think of loss as an expectation over the full **data distribution** p_{data}

Approximate expectation via sampling

$$\begin{aligned} \nabla_W L(W) &= \nabla_W \mathbb{E}_{(x,y) \sim p_{data}} [L(x, y, W)] + \lambda \nabla_W R(W) \\ &\approx \sum_{i=1}^N \nabla_w L_W(x_i, y_i, W) + \nabla_w R(W) \end{aligned}$$

Aside: Image Features

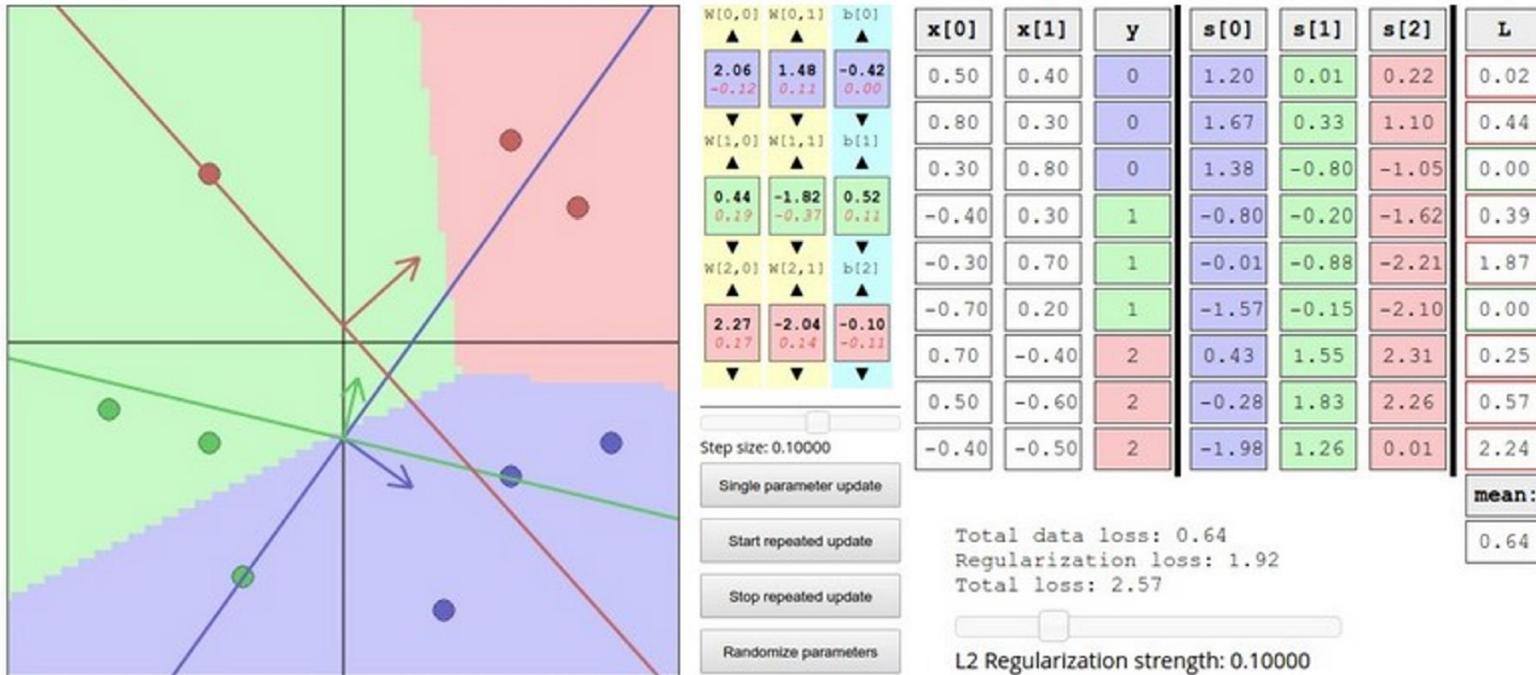


Class scores

$$f(x) = Wx$$



Interactive Web Demo



<http://vision.stanford.edu/teaching/cs231n-demos/linear-classify/>

Practical Issue: Numeric stability

may be very large

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

dividing large numbers can be numerically unstable

A common choice is to set $\log C = -\max_j f_j$

```
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
```

Reading

<https://cs231n.github.io/linear-classify/>

Next time:

Introduction to neural networks

Backpropagation