Recall from last time: Linear Classifier

Image

Array of **32x32x3** numbers
(3072 numbers total)

$\longrightarrow$ f(**x**,**W**) $\longrightarrow$ **10** numbers giving class scores

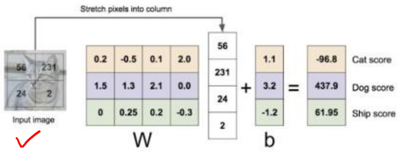
$\uparrow$

W
parameters
or weights

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

Algebraic Viewpoint

$f(x,W) = Wx$

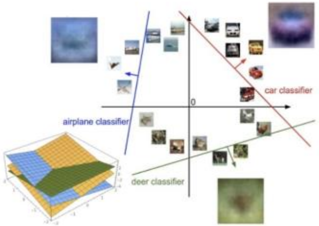


Visual Viewpoint

One template
per class

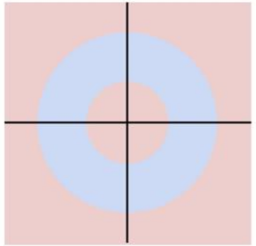
Geometric Viewpoint

Hyperplanes
cutting up space



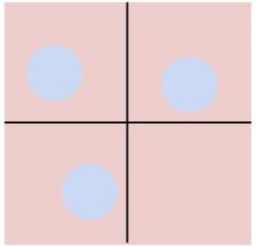
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



Class 1:
Three modes

Class 2:
Everything else



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

TO DO:

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)**

$$\min_{W} \mathrm{Loss}(y, \hat{y})$$

where y is the true label and $\hat{y}$ is the predicted label

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

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

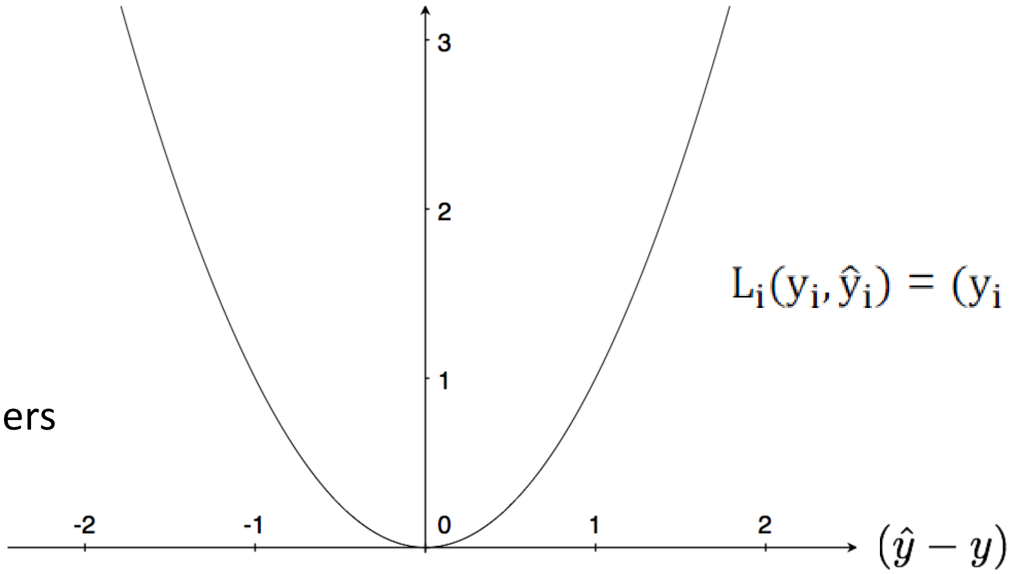
$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

How do we choose L_i ?

YOU get to chose the loss function!

(some are better than others depending on what you want to do)

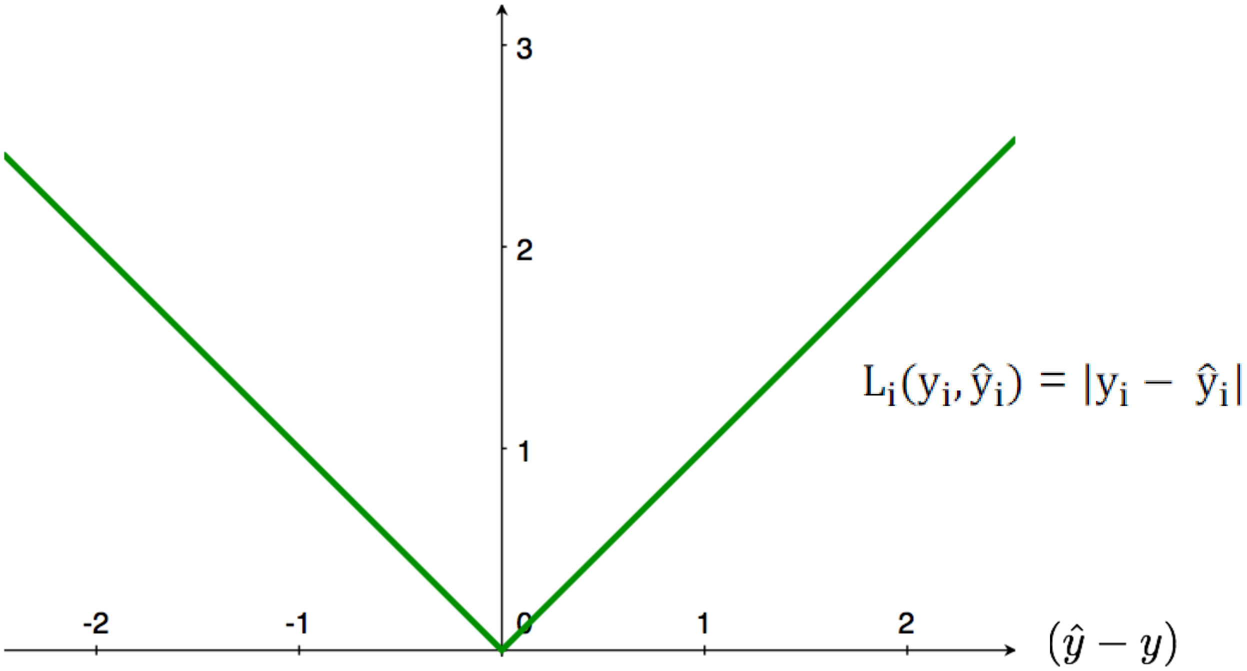
Squared Error (L2)



$$L_i(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$$

Not robust to outliers

L1 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**

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:

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

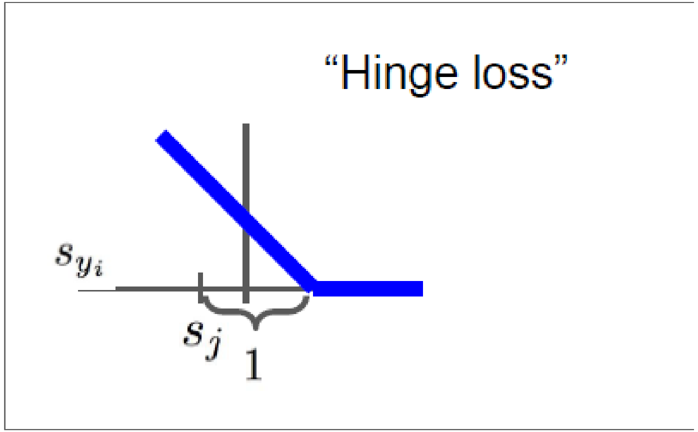
$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

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

Multiclass SVM loss:



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



$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

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

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**

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:

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

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**
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:

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

= max(0, 5.1 - 3.2 + 1)
+max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

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:

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

= max(0, 1.3 - 4.9 + 1)
+max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

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:

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

= max(0, 2.2 - (-3.1) + 1)
+max(0, 2.5 - (-3.1) + 1)
= max(0, 6.3) + max(0, 6.6)
= 6.3 + 6.6
= 12.9

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:

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

Loss over full dataset is average:

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

L = (2.9 + 0 + 12.9)/3
= **5.27**

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:

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

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:

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

Q: 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:

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

Q: 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:

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

Q: 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:

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

Q: What if we used

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

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

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

E.g. Suppose that we found a W such that L = 0.
Is this W unique?

No! 2W also has L = 0!

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	

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

Before:

= max(0, 1.3 - 4.9 + 1)
+max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

With W twice as large:

= max(0, 2.6 - 9.8 + 1)
+max(0, 4.0 - 9.8 + 1)
= max(0, -6.2) + max(0, -4.8)
= 0 + 0
= 0

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

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

E.g. Suppose that we found a W such that L = 0.
Is this W unique?

No! 2W also has L = 0!

How do we choose between W and 2W?

Regularization

λ = regularization strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

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

Simple examples

<u>L2 regularization:</u> $R(W) = \sum_k \sum_l W_{k,l}^2$

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

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

More complex:

Dropout

Batch normalization

Stochastic depth, fractional pooling, etc

Regularization

λ = regularization strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

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

Why regularize?
- Express preferences over weights
- Make the model *simple* so it works on test data

Regularization: Expressing Preferences

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

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

$$w_2 = \boxed{[0.25, 0.25, 0.25, 0.25]}$$

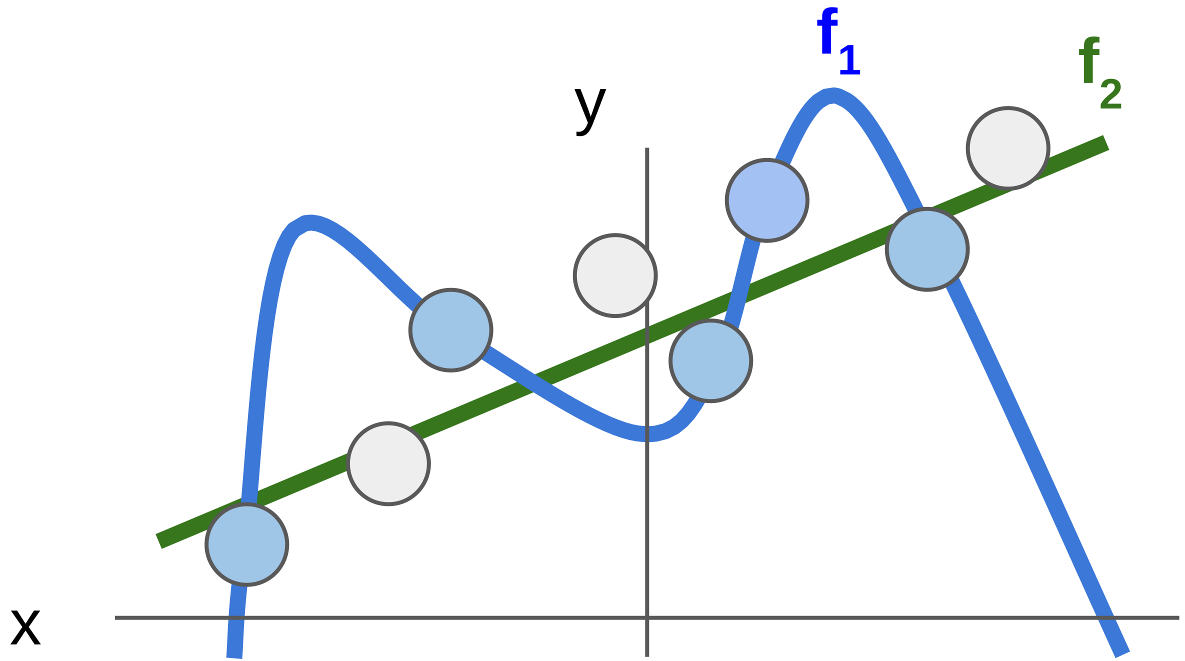
$$w_1^T x = w_2^T x = 1$$

L2 Regularization

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

L2 regularization likes to "spread out" the weights

Regularization: Prefer Simpler Models



Regularization pushes against fitting the data *too* well so we don't fit noise in the 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

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

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

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

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

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

	Unnormalized log-probabilities / logits	exp → unnormalized probabilities	normalize → probabilities	compare	Correct probs
cat	**3.2**	**24.5**	**0.13**	→ compare ←	**1.00**
car	5.1	164.0	0.87		0.00
frog	-1.7	0.18	0.00		0.00

Kullback–Leibler divergence

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

Unnormalized log-probabilities / logits

unnormalized probabilities

probabilities

Correct probs

Softmax Classifier (Multinomial Logistic Regression)

cat	**3.2**
car	5.1
frog	-1.7

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

$$\boxed{s = f(x_i; W)}$$

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

Softmax Function

Maximize probability of correct class

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

Putting it all together:

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

Q: What is the min/max possible loss L_i?
A: min 0, max infinity

Softmax Classifier (Multinomial Logistic Regression)

cat	**3.2**
car	5.1
frog	-1.7

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

$$\boxed{s = f(x_i; W)}$$

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

Softmax Function

Maximize probability of correct class

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

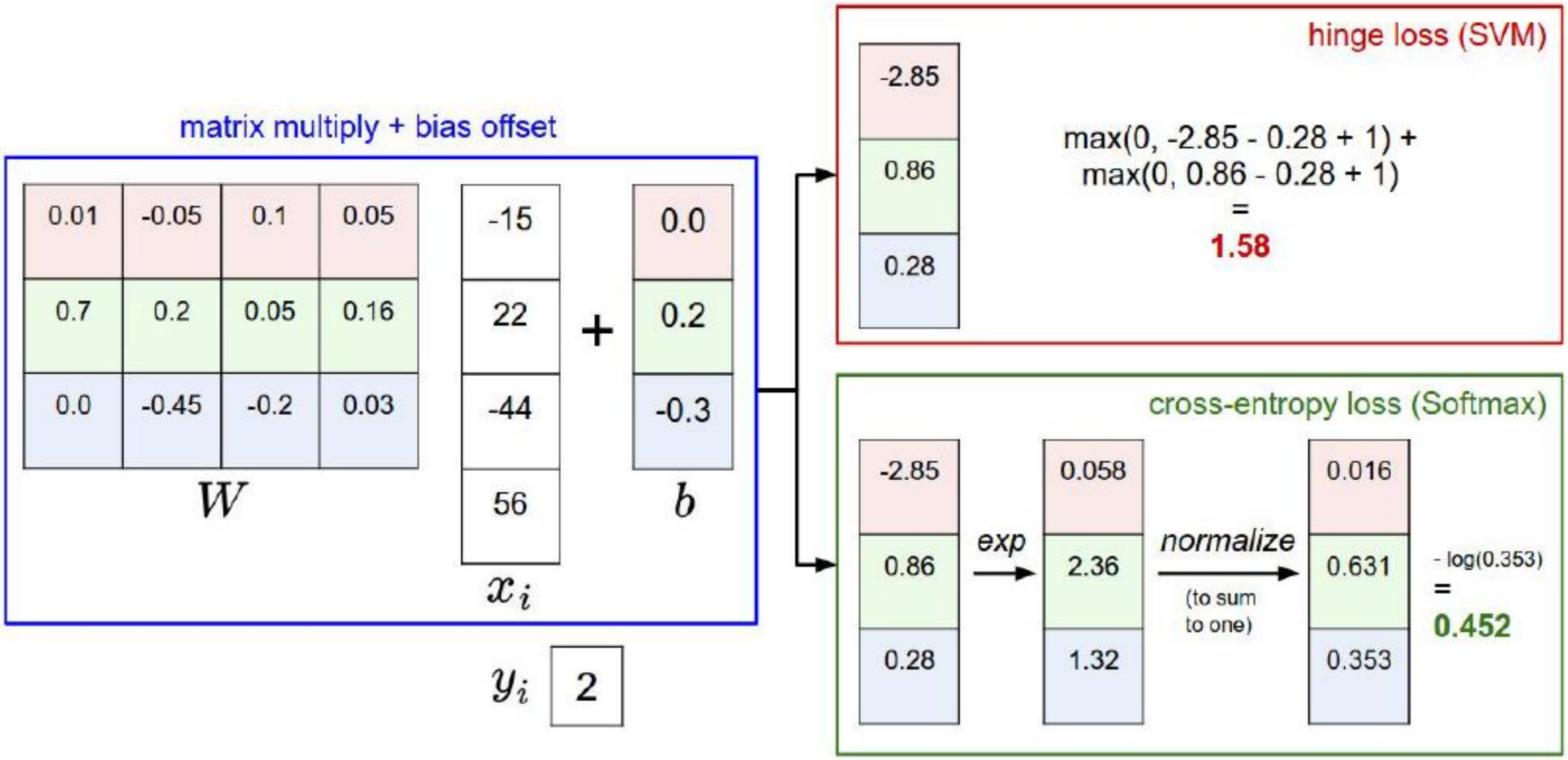
Putting it all together:

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

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

A: log(C), eg log(10) ≈ 2.3

Softmax vs. SVM



Recap

How do we find the best W?

- 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

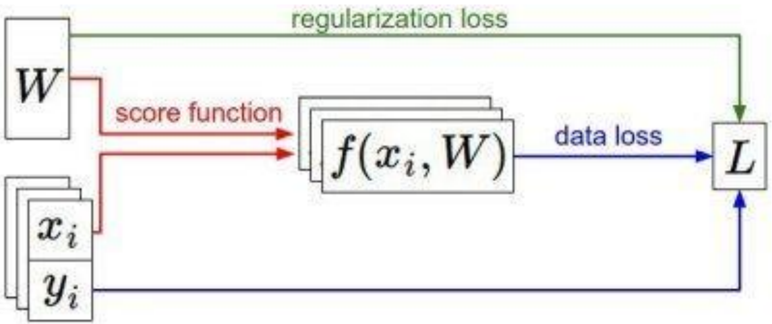
$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

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 loss



How to find best weights w^*?

$$
\begin{aligned}
w^* &= \arg\min_w \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i, w), y_i) + R(w) \\
&= \arg\min_w g(w)
\end{aligned}
$$

How to find best weights?

$$w^* = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i, w), y_i) + R(w)$$
$$= \arg\min_w g(w)$$

How to find best weights?

$$w^* = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i, w), y_i) + R(w)$$
$$= \arg\min_w g(w)$$

Idea #1: Closed Form Solution

Works for some models (linear regression) but in general very hard (even for one-layer net)

How to find best weights?

$$w^* = \arg\min_w \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i, w), y_i) + R(w)$$

$$= \arg\min_w g(w)$$

Idea #1: Closed Form Solution

Works for some models (linear regression) but in general very hard (even for one-layer net)

Idea #2: Random Search

Try a bunch of random values for w, keep the best one

Extremely inefficient in high dimensions

How to find best weights?

$$w^* = \arg\min_w \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i, w), y_i) + R(w)$$
$$= \arg\min_w g(w)$$

Idea #1: Closed Form Solution
Works for some models (linear regression) but in general very hard (even for one-layer net)

Idea #2: Random Search
Try a bunch of random values for w, keep the best one
Extremely inefficient in high dimensions

Idea #3: Go downhill
Start somewhere random, take tiny steps downhill and repeat
Good idea!

Which way is downhill?

Recall from single-variable calculus:

$$g'(w) = \lim_{h \to 0} \frac{g(w+h) - g(w)}{h}$$

If w is a scalar the **derivative** tells us how much g will change if w changes a little bit

Which way is downhill?

Recall from single-variable calculus:

$$g'(w) = \lim_{h \to 0} \frac{g(w+h) - g(w)}{h}$$

If w is a scalar the **derivative** tells us how much g will change if w changes a little bit

And multivariable calculus:

$$\frac{\partial g}{\partial w_i}(w) = \lim_{h \to 0} \frac{g(w + h\hat{e}_i) - g(w)}{h}$$

If w is a vector then we can take **partial derivatives** with respect to each component

$$\nabla g(w) = \left(\frac{\partial g}{\partial w_1}(w), \ldots, \frac{\partial g}{\partial w_K}(w) \right)$$

The **gradient** is the vector of all partial derivatives; it has the same shape as w

Which way is downhill?

Recall from single-variable calculus:

$$g'(w) = \lim_{h \to 0} \frac{g(w+h) - g(w)}{h}$$

If w is a scalar the **derivative** tells us how much g will change if w changes a little bit

And multivariable calculus:

$$\frac{\partial g}{\partial w_i}(w) = \lim_{h \to 0} \frac{g(w + h\hat{e}_i) - g(w)}{h}$$

If w is a vector then we can take **partial derivatives** with respect to each component

$$\nabla g(w) = \left(\frac{\partial g}{\partial w_1}(w), \ldots, \frac{\partial g}{\partial w_K}(w) \right)$$

The **gradient** is the vector of all partial derivatives; it has the same shape as w

Useful fact: The gradient of g at w gives the **direction of steepest increase** thus we need **–ve gradient**

In summary:

- Numerical 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.**

Gradient descent

The simple algorithm behind all deep learning

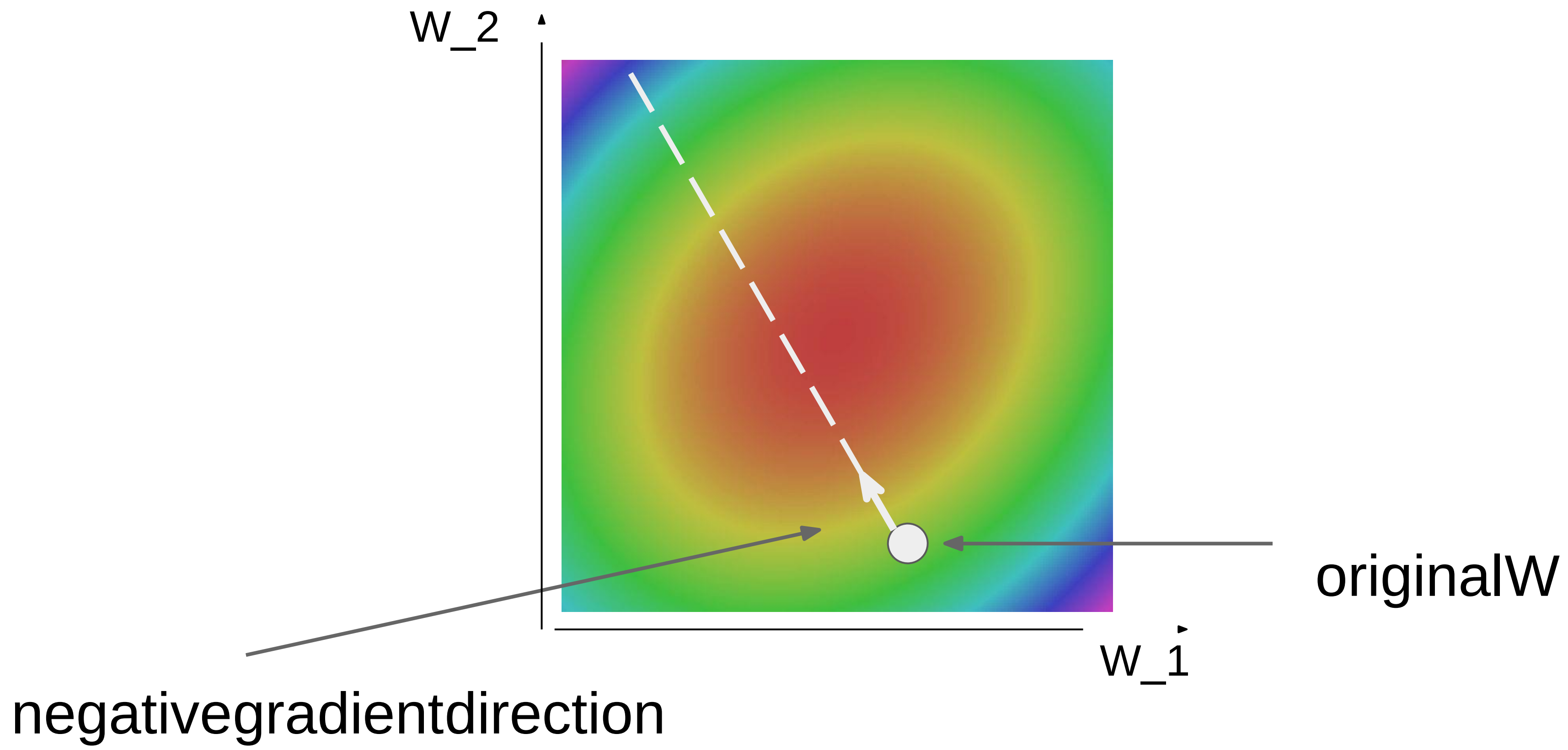
Initialize w randomly
While true:
  Compute gradient $\nabla g(w)$ at current point
  Move downhill a little bit. $w = w - \alpha \nabla g(w)$

Gradient descent

The simple algorithm behind all deep learning

Initialize w randomly
While true:
    Compute gradient $\nabla g(w)$ at current point
    Move downhill a little bit: $w = w - \alpha \nabla g(w)$

Learning rate: How big each step should be



W_2
W_1
originalW
negativegradientdirection

Gradient descent

The simple algorithm behind all deep learning

Full sum expensive when N is large!

In practice we **estimate** the gradient using a small **minibatch** of data (Stochastic gradient descent)

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

Initialize w randomly
While true:
    Compute gradient $\nabla g(w)$ at current point
    Move downhill a little bit: $w = w - \alpha \nabla g(w)$

Learning rate: How big each step should be

Gradient descent

The simple algorithm behind all deep learning

Full sum expensive when N is large!

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

In practice we **estimate** the gradient using a small **minibatch** of data (Stochastic gradient descent)

Initialize w randomly
While true:
  Compute gradient $\nabla g(w)$ at current point
  Move downhill a little bit: $w = w - \alpha \nabla g(w)$

The **update rule** is the formula for updating the weights at each iteration; in practice people use fancier rules (momentum, RMSProp, Adam, etc)

Learning rate: How big each step should be