Convolutional neural networks II

September 30th, 2019

Yong Jae Lee
UC Davis

Many slides from Rob Fergus, Svetlana Lazebnik, Jia-Bin Huang, Derek Hoiem, Adriana Kovashka, Andrej Karpathy

Announcements

- Sign-up for paper presentations

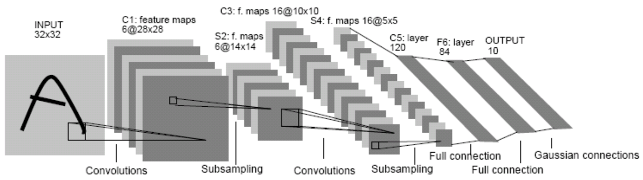
Standard classifiers

Nearest neighbor

10^6 examples

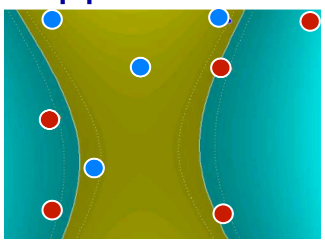
Shakhnarovich, Viola, Darrell 2003
Berg, Berg, Malik 2005...

Neural networks



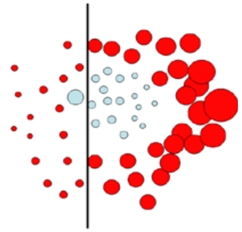
LeCun, Bottou, Bengio, Haffner 1998
Rowley, Baluja, Kanade 1998
…

Support Vector Machines



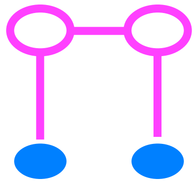
Guyon, Vapnik
Heisele, Serre, Poggio, 2001,…

Boosting



Viola, Jones 2001, Torralba et al. 2004, Opelt et al. 2006,…

Conditional Random Fields



McCallum, Freitag, Pereira 2000; Kumar, Hebert 2003
…

Slide adapted from Antonio Torralba

Deep neural networks

- Lots of hidden layers
- Depth = power (usually)

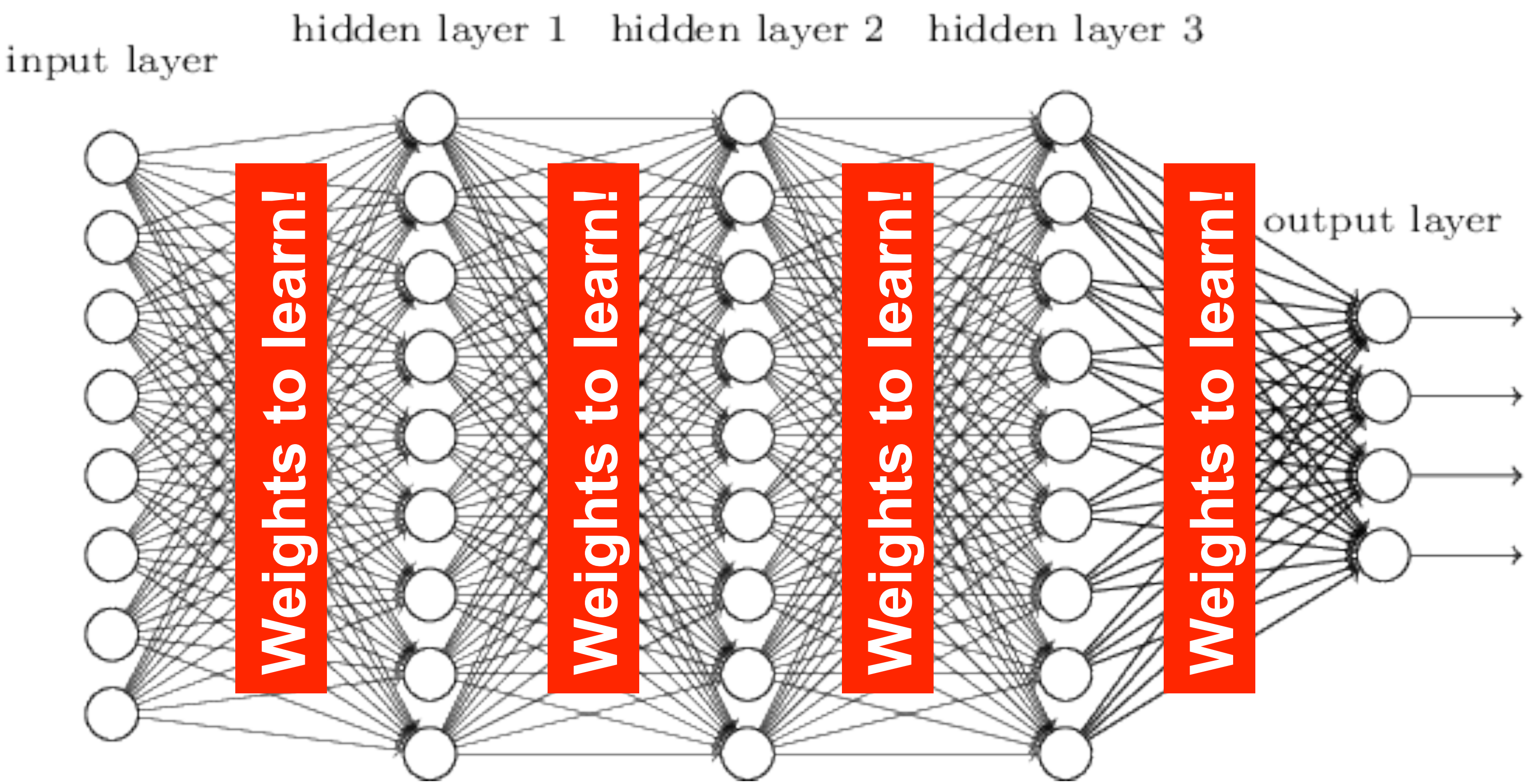


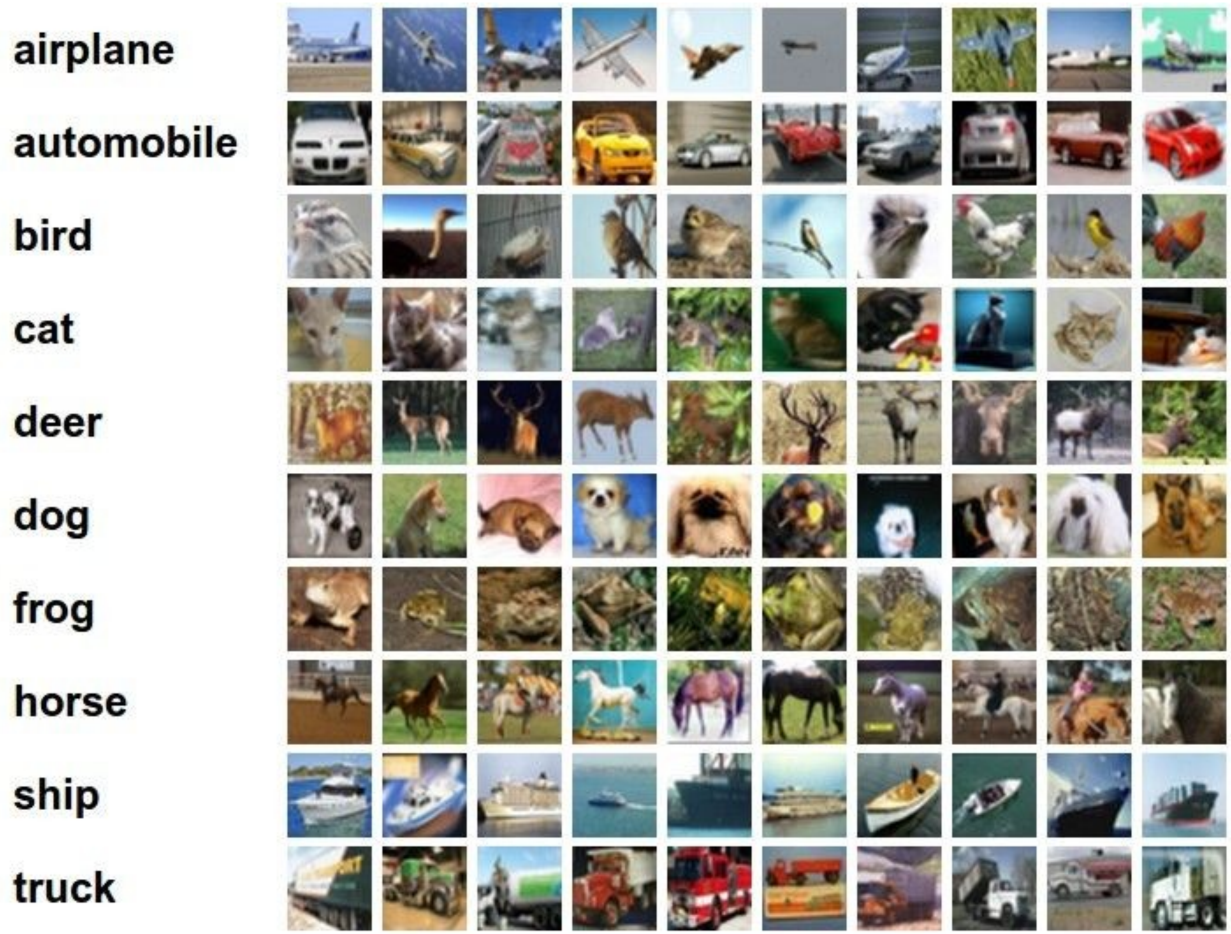
Figure from http://neuralnetworksanddeeplearning.com/chap5.html

How do we train them?

- The goal is to iteratively find a set of weights that allow the activations/outputs to match the desired output
- For this, we will *minimize a **loss function***
- The loss function quantifies the agreement between the predicted scores and GT labels

- First, let's simplify and assume we have a single layer of weights in the network

Classification goal



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

Andrej Karpathy

Classification scores

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

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

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

10 numbers, indicating class scores

Andrej Karpathy

Linear classifier

$$\boxed{f(x, W)} = \boxed{W}\boxed{x} \quad \boxed{+b}$$

3072x1

10x1

10x1

10x3072

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

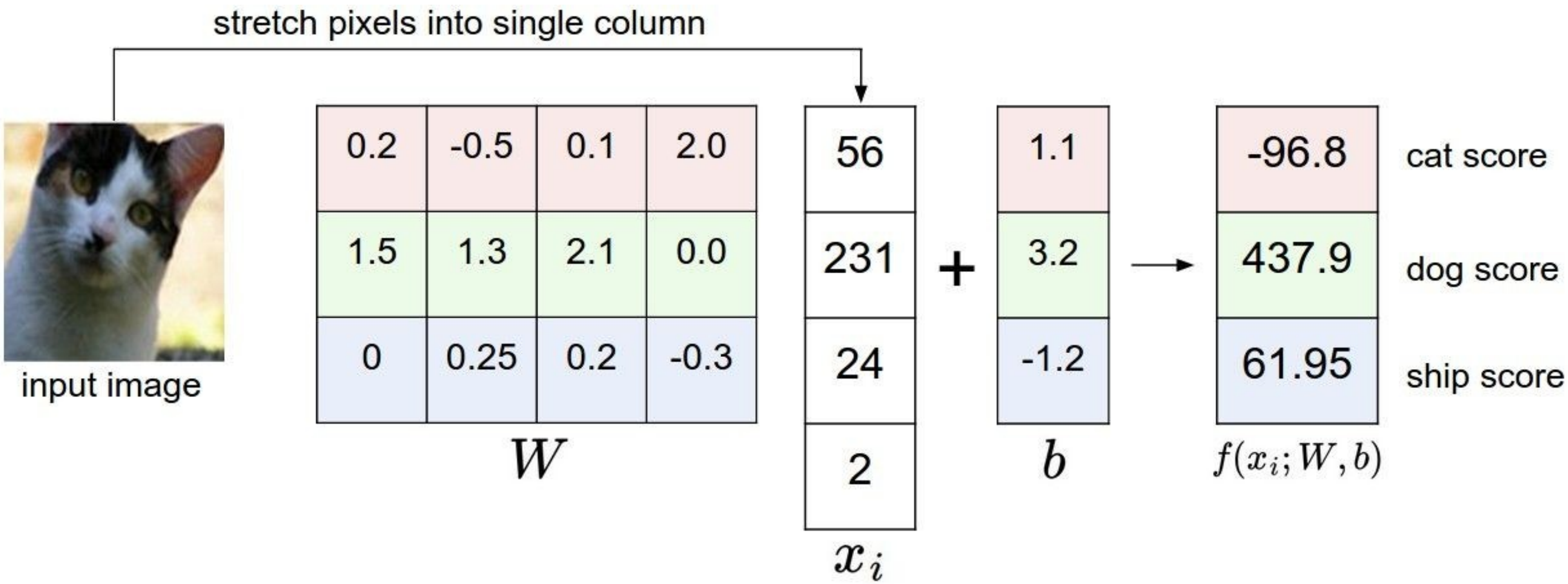
10 numbers, indicating class scores

parameters, or “weights”

Andrej Karpathy

Linear classifier

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



Andrej Karpathy

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)**

Linear classifier

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**

Adapted from Andrej Karpathy

Linear classifier: 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**

Hinge 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 loss has the form:

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

Want: $s_{y_i} >= s_j + 1$
i.e. $s_j - s_{y_i} + 1 <= 0$

If true, loss is 0
If false, loss is magnitude of violation

Adapted from Andrej Karpathy

Linear classifier: 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**
Loss:	2.9		

Hinge 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 loss has the form:

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

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

Adapted from Andrej Karpathy

Linear classifier: 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**
Loss:	2.9	0	

Hinge 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 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

Adapted from Andrej Karpathy

Linear classifier: 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**
Loss:	2.9	0	12.9

Hinge 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 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, 5.3 + 1)
+ max(0, 5.6 + 1)
= 6.3 + 6.6
= 12.9

Adapted from Andrej Karpathy

Linear classifier: 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**
Loss:	2.9	0	12.9

Hinge 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 loss has the form:

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

and the full training loss is the mean over all examples in the training data:

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

L = (2.9 + 0 + 12.9)/3
= 15.8 / 3 = **5.3**

Linear classifier: Hinge loss

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

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

Adapted from Andrej Karpathy

Linear classifier: Hinge loss

λ = regularization strength
(hyperparameter)

Weight Regularization

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

In common use:

L2 regularization

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

L1 regularization

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

Dropout

Adapted from Andrej Karpathy

Another loss: Softmax (cross-entropy)

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:

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

cat	**3.2**
car	5.1
frog	-1.7

Another loss: Softmax (cross-entropy)

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

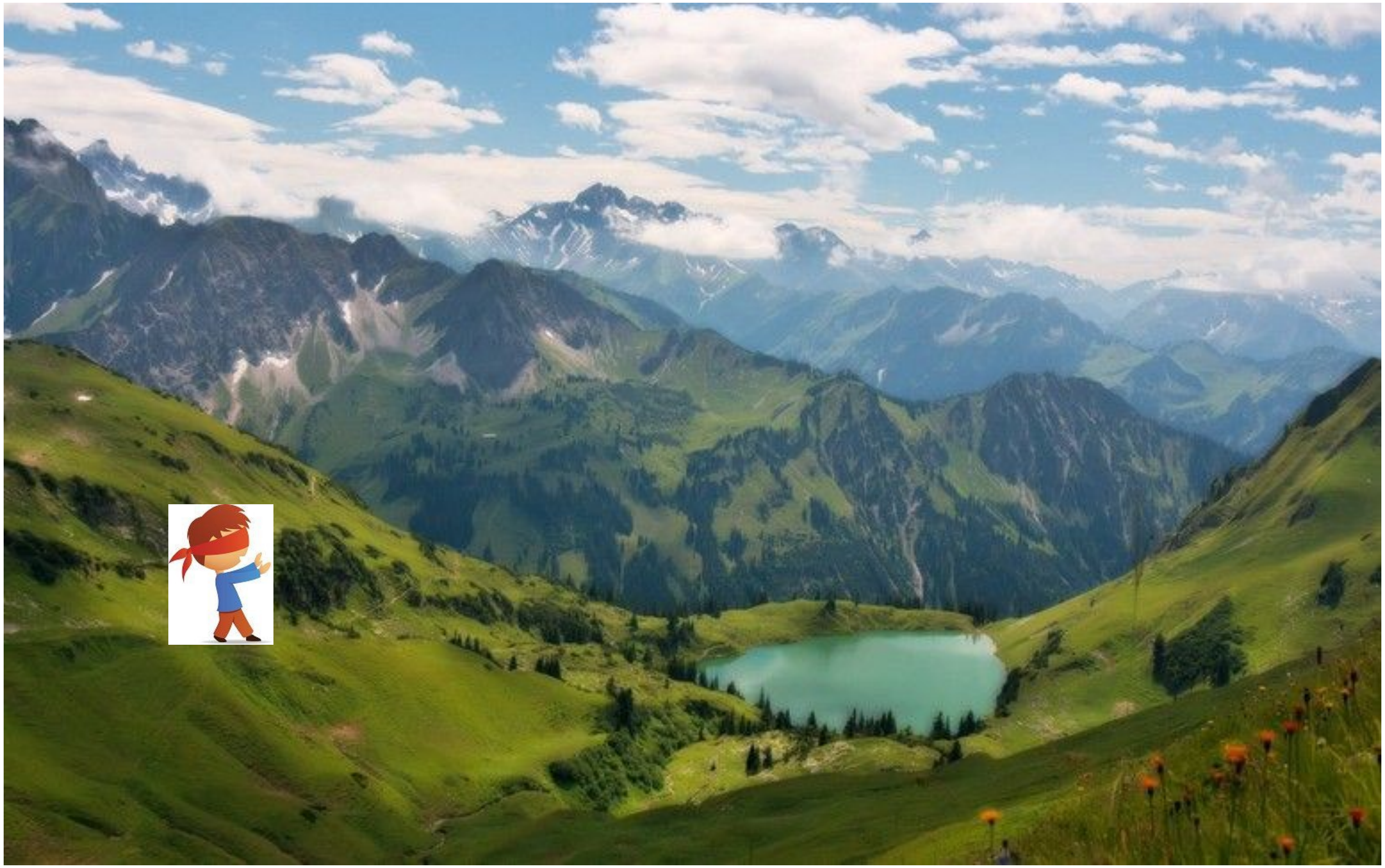
unnormalized probabilities

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

unnormalized log probabilities

probabilities

How to minimize the loss function?



Andrej Karpathy

How to minimize the loss function?

In 1-dimension, 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).

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:	**W + h** (first dim)**:**	**gradient dW:**
[0.34,	[0.34 + **0.0001**,	[**-2.5**,
-1.11,	-1.11,	?,
0.78,	0.78,	?,
0.12,	0.12,	(1.25322 - 1.25347)/0.0001 = -2.5
0.55,	0.55,	$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$
2.81,	2.81,	?,
-3.1,	-3.1,	?,…]
-1.5,	-1.5,	
0.33,…]	0.33,…]	
loss 1.25347	**loss 1.25322**	

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

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

(1.25353 - 1.25347)/0.0001
= 0.6

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

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

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!

$\nabla_W L$ = ...

current W:

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

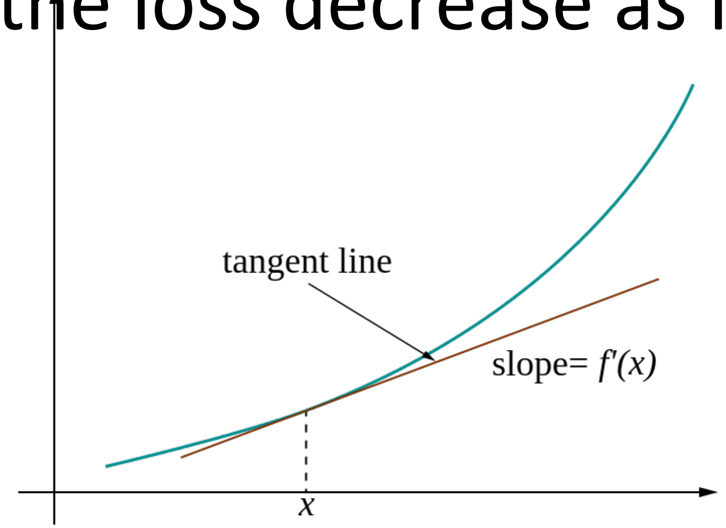
dW = ...
(some function
data and W)

gradient dW:

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

Loss gradients

- Denoted as (diff notations): $$\frac{\partial E}{\partial w_{ji}^{(1)}} \qquad \nabla_W L$$
- i.e. how the loss changes as a function of the weights
- We want to change the weights in such a way that makes the loss decrease as fast as possible



Gradient descent

- We'll update weights iteratively
- Move in direction opposite to gradient:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

Time

Learning rate

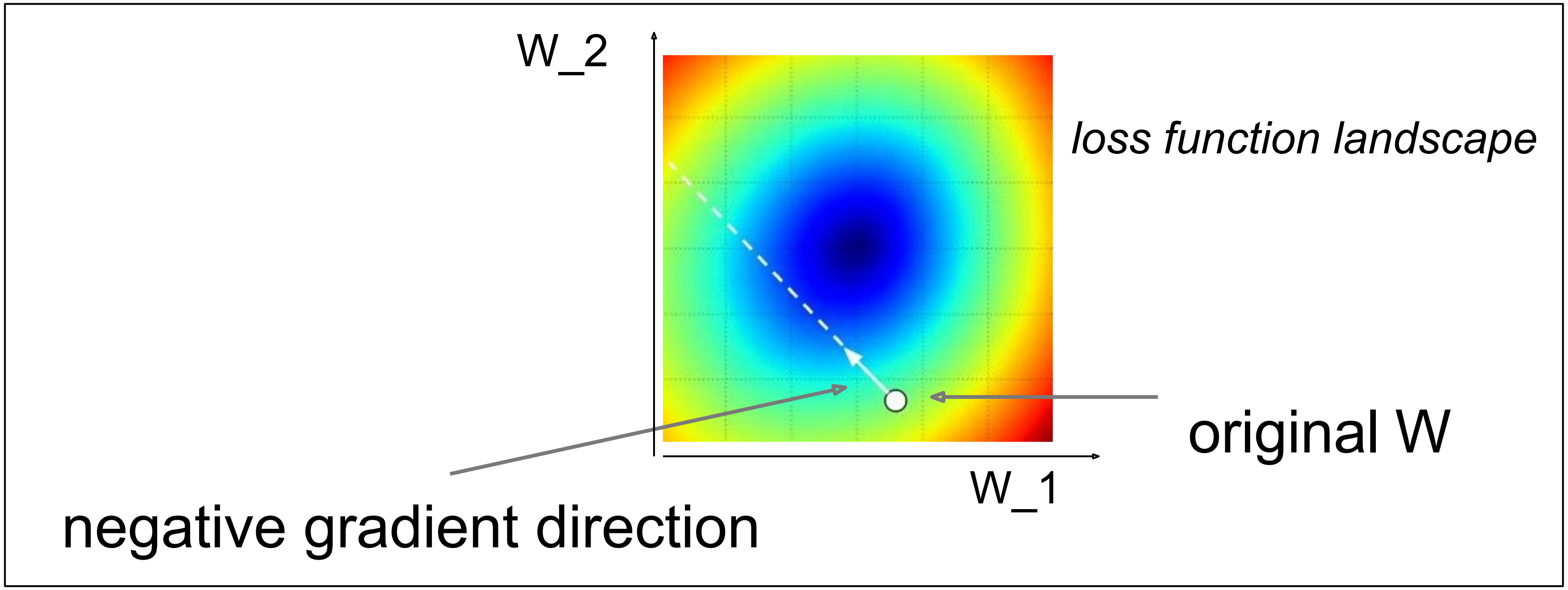


Figure from Andrej Karpathy

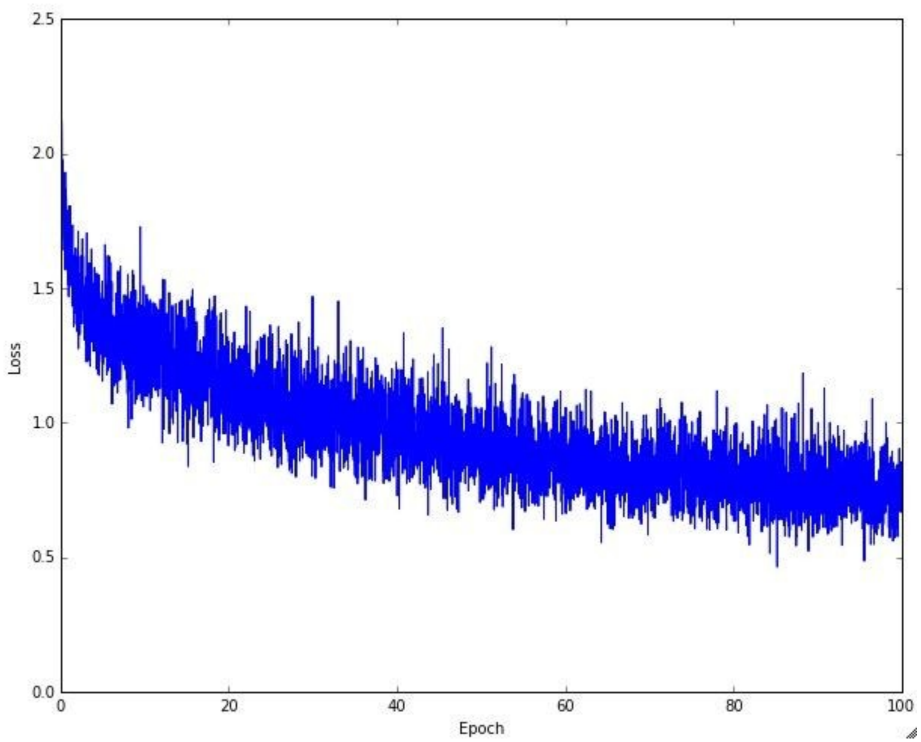
Gradient descent

- Iteratively *subtract* the gradient with respect to the model parameters (w)
- i.e. we’re moving in a direction opposite to the gradient of the loss
- i.e. we’re moving towards *smaller* loss

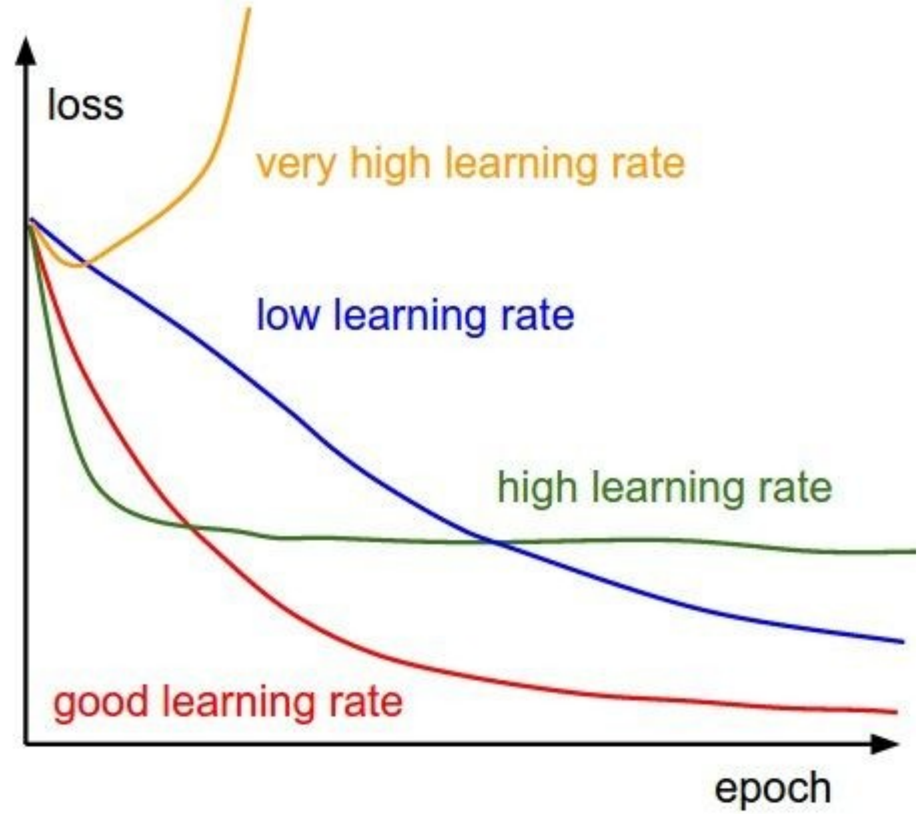
Mini-batch gradient descent

- In classic gradient descent, we compute the gradient from the loss for all training examples (can be slow)
- So, use only use *some* of the data for each gradient update
- We cycle through all the training examples multiple times
- Each time we've cycled through all of them once is called an 'epoch'

Learning rate selection



The effects of step size (or "learning rate")



Andrej Karpathy

Questions?

See you Wednesday!