

Topics:

- Linear Classification, Loss functions
- Gradient Descent

CS 4644-DL / 7643-A
ZSOLT KIRA

- **Assignment 1 out!**

- Due **Jan 31st 11:59pm** (grace period Feb 2th).
- Submit something by Jan 31st!
- Start early, start early, start early!

- **Piazza:** Please make sure to actively check and participate!

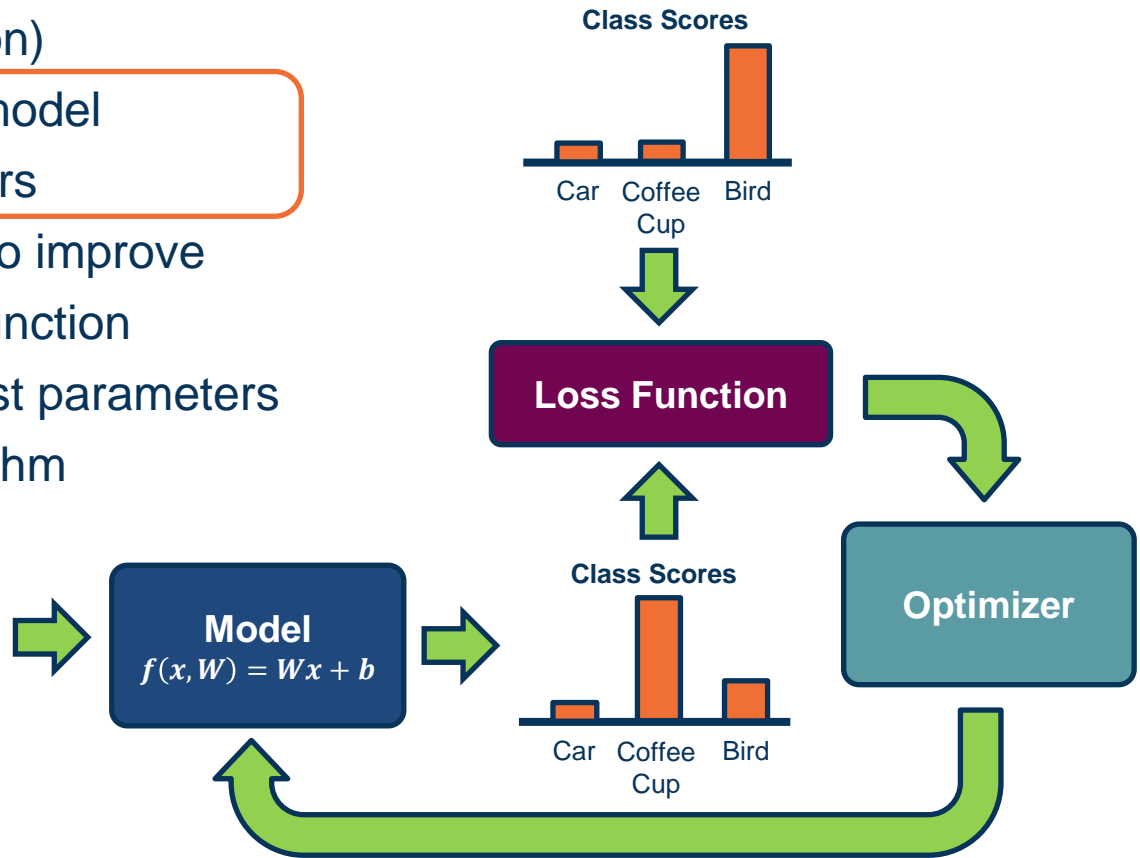
- **Office hours** schedule on webpage:

https://faculty.cc.gatech.edu/~zk15/teaching/AY2025_cs764_3_spring/index.html

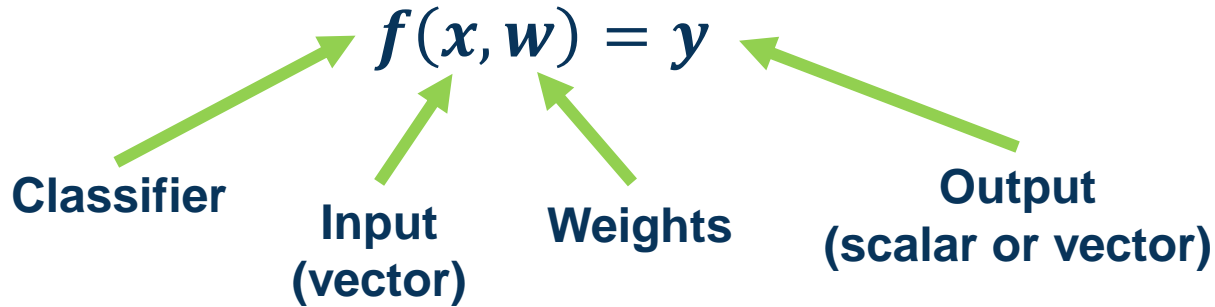
- Input (and representation)
- Functional form of the model
 - Including parameters
- Performance measure to improve
 - Loss or objective function
- Algorithm for finding best parameters
 - Optimization algorithm



Data: Image



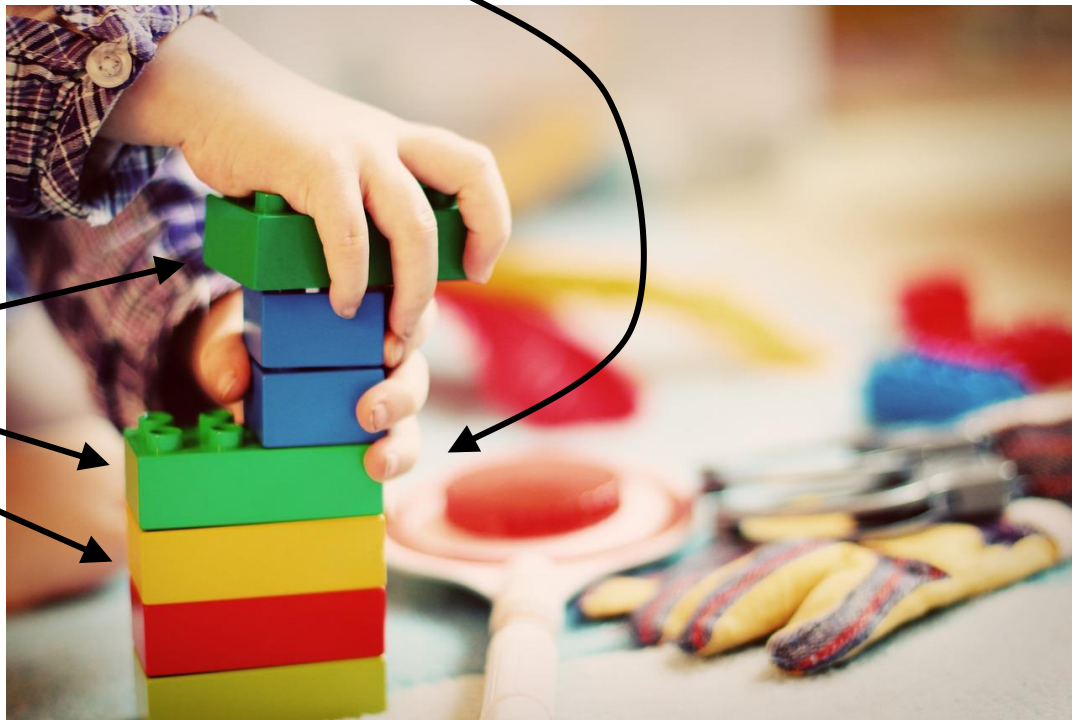
Components of a Parametric Model



- **Input:** Continuous number or vector
- **Output:** A continuous number
 - For classification typically a **score**
 - For regression what we want to regress to (house prices, crime rate, etc.)
- **w is a vector and weights** to optimize to fit target function

Neural Network

Linear
classifiers



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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Deep Learning as Legos

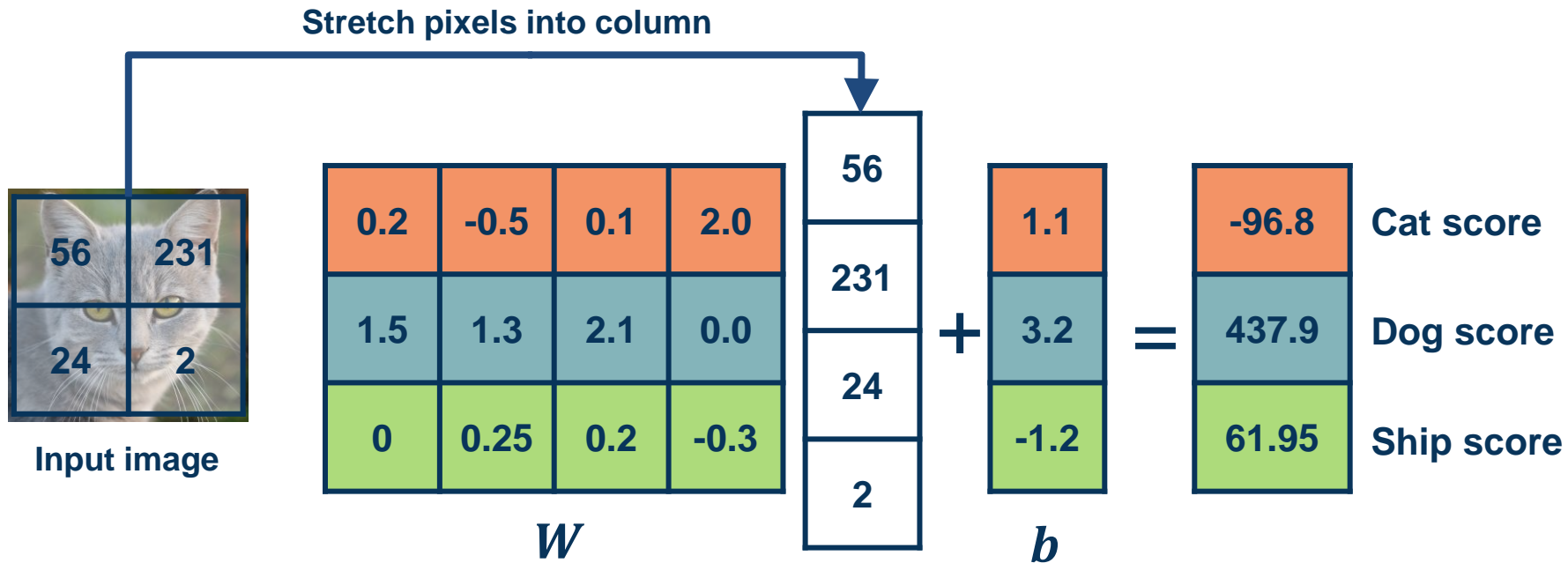
- We can move the bias term into the weight matrix, and a “1” at the end of the input
- Results in **one matrix-vector multiplication!**

$$\text{Model}$$
$$f(x, W) = Wx + b$$

$$\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} & b_1 \\ w_{21} & w_{22} & \cdots & w_{2m} & b_2 \\ w_{31} & w_{32} & \cdots & w_{3m} & b_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \\ 1 \end{bmatrix}$$

W x

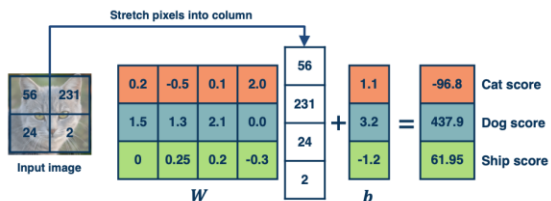
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Algebraic Viewpoint

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



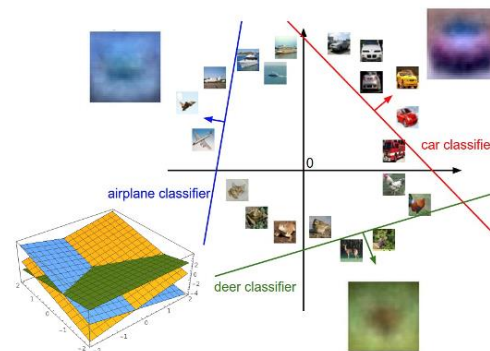
Visual Viewpoint

One template per class



Geometric Viewpoint

Hyperplanes cutting up space



Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Performance Measure for a Classifier

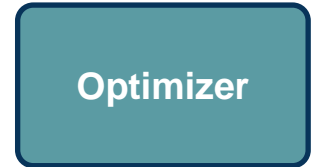
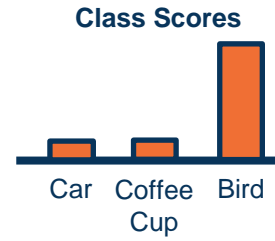
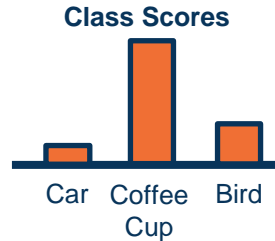
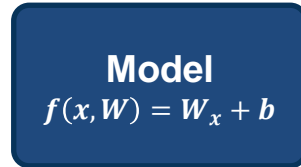
- Input (and representation)
- Functional form of the model
 - Including parameters
- **Performance measure to improve**
 - **Loss or objective function**
- Algorithm for finding best parameters
 - Optimization algorithm



Data: Image



Features: Histogram

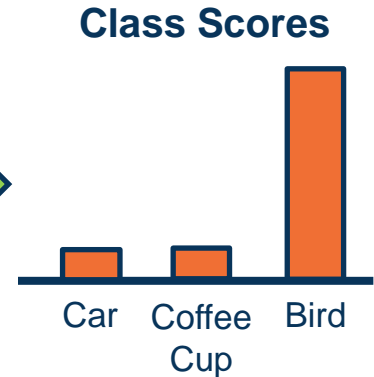


Components of a Parametric Model

- The output of a classifier can be considered a **score**
- For binary classifier, use rule:
$$y = \begin{cases} 1 & \text{if } f(x, w) \geq 0 \\ 0 & \text{otherwise} \end{cases}$$
- Can be used for many classes by considering one class versus all the rest (one versus all)
- For multi-class classifier can take the maximum

Model

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



We need a performance measure to **optimize**

- ◆ Penalizes model for being wrong
- ◆ Allows us to modify the model to reduce this penalty
- ◆ Known as an **objective** or **loss** function

In machine learning we use **empirical risk minimization**

- ◆ Reduce the loss over the **training** dataset
- ◆ We **average** the loss over the training data

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:

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

Multiclass SVM loss:

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

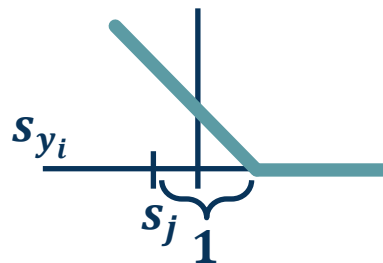


and using the shorthand for the
scores vector: $s = f(x_i, W)$

the SVM loss has the form:

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

Example: “Hinge Loss”



Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

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:

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

$$\begin{aligned} &= \max(0, 5.1 - 3.2 + 1) \\ &\quad + \max(0, -1.7 - 3.2 + 1) \\ &= \max(0, 2.9) + \max(0, -3.9) \\ &= 2.9 + 0 \\ &= 2.9 \end{aligned}$$

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

Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

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:

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

$$\begin{aligned} &= \max(0, 1.3 - 4.9 + 1) \\ &\quad + \max(0, 2.0 - 4.9 + 1) \\ &= \max(0, -2.6) + \max(0, -1.9) \\ &= 0 + 0 \\ &= 0 \end{aligned}$$

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:

0.0

Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Multiclass SVM loss:

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

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

No change for small values

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 slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Multiclass SVM loss:

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

Q: What is min/max of loss value?

[0,inf]

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 slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Multiclass SVM loss:

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

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

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



C-1

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

Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Multiclass SVM loss:

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

Q: What if the sum was over all classes?

(including $j = y_i$)

No difference
(add constant 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

Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Multiclass SVM loss:

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

Q: What if we used mean instead of sum?

No difference
Scaling by constant

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 slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

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:

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

$$L = (2.9 + 0 + 12.9)/3 \\ = 5.27$$

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

Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Several issues with scores:

- Not very interpretable (no bounded value)

We often want **probabilities**

- More interpretable
- Can relate to probabilistic view of machine learning

We use the **softmax** function to convert scores to probabilities

$$s = f(x, W) \quad \text{Scores}$$

$$P(Y = k | X = x) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{Softmax Function}$$

- If we use the softmax function to convert scores to probabilities, the right loss function to use is **cross-entropy**
- Can be derived by looking at the distance between two probability distributions (output of model and ground truth)
- Can also be derived from a maximum likelihood estimation perspective

$$s = f(x, W) \quad \text{Scores}$$

$$P(Y = k|X = x) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{Softmax Function}$$

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

Maximize log-prob of correct class =
Maximize the log likelihood
= Minimize the negative log likelihood

- ◆ If we use the softmax function to convert scores to probabilities, the right loss function to use is **cross-entropy**
- ◆ Goal: Minimize KL-divergence (distance measure b/w probability distributions)

$$p^* = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \hat{p} = \begin{bmatrix} P(Y = 1|x, w) \\ P(Y = 2|x, w) \\ P(Y = 3|x, w) \\ P(Y = 4|x, w) \\ P(Y = 5|x, w) \\ P(Y = 6|x, w) \\ P(Y = 7|x, w) \\ P(Y = 8|x, w) \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.01 \\ 0.01 \\ 0.01 \\ 0.01 \\ 0.01 \\ 0.15 \\ 0.3 \end{bmatrix}$$

Ground Truth

Prediction

$$\begin{aligned} \min_w KL(p^* || \hat{p}) &= \sum_y p^*(y) \log \frac{p^*(y)}{\hat{p}(y)} \\ &= \sum_y p^*(y) \log(p^*(y)) - \sum_y p^*(y) \log(\hat{p}(y)) \\ &\quad \underbrace{-H(p^*)}_{\text{(negative entropy, term goes away because not a function of model, } W, \text{ parameters we are minimizing over)}} \quad \underbrace{H(p^*, \hat{p})}_{\text{(Cross-Entropy)}} \end{aligned}$$

Since p^* is one-hot (0 for non-ground truth classes), all we need to minimize is (where i is ground truth class): $\min_w (-\log \hat{p}(y_i))$

Softmax Classifier (Multinomial Logistic Regression)



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

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

Probabilities
must be ≥ 0

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

Softmax
Function

Probabilities
must sum to 1

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

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



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

Q: How is it
possible that non-
GT probabilities
aren't in loss?

Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Cross-Entropy Loss Example

Softmax Classifier (Multinomial Logistic Regression)



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

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

Probabilities
must be ≥ 0

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

Probabilities
must sum to 1

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

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

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

Infimum is 0, max is unbounded (inf)

Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Softmax Classifier (Multinomial Logistic Regression)



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

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

Probabilities
must be ≥ 0

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

Softmax
Function

Probabilities
must sum to 1

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

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

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

Log(C), e.g. $\log(10) \approx 2$

Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Often, we add a **regularization term** to the loss function

L1 Regularization

$$L_i = |y - Wx_i|^2 + |W|$$

Example regularizations:

- ◆ L1/L2 on weights (encourage small values)

Gradient Descent

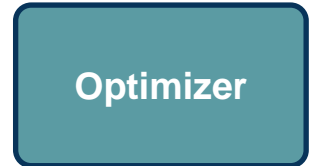
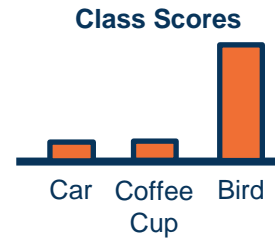
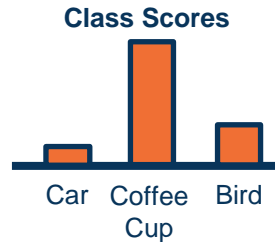
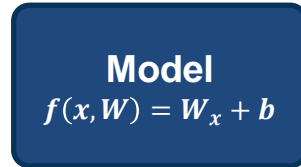
- Input (and representation)
- Functional form of the model
 - Including parameters
- Performance measure to improve
 - Loss or objective function
- **Algorithm for finding best parameters**
 - **Optimization algorithm**



Data: Image



Features: Histogram



Components of a Parametric Model

Given a model and loss function, finding the best set of weights is a **search problem**

- Find the best combination of weights that minimizes our loss function

Several classes of methods:

- Random search
- Genetic algorithms (population-based search)
- Gradient-based optimization

In deep learning, **gradient-based methods are dominant** although not the only approach possible

$$\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} & b_1 \\ w_{21} & w_{22} & \cdots & w_{2m} & b_2 \\ w_{31} & w_{32} & \cdots & w_{3m} & b_3 \end{bmatrix}$$

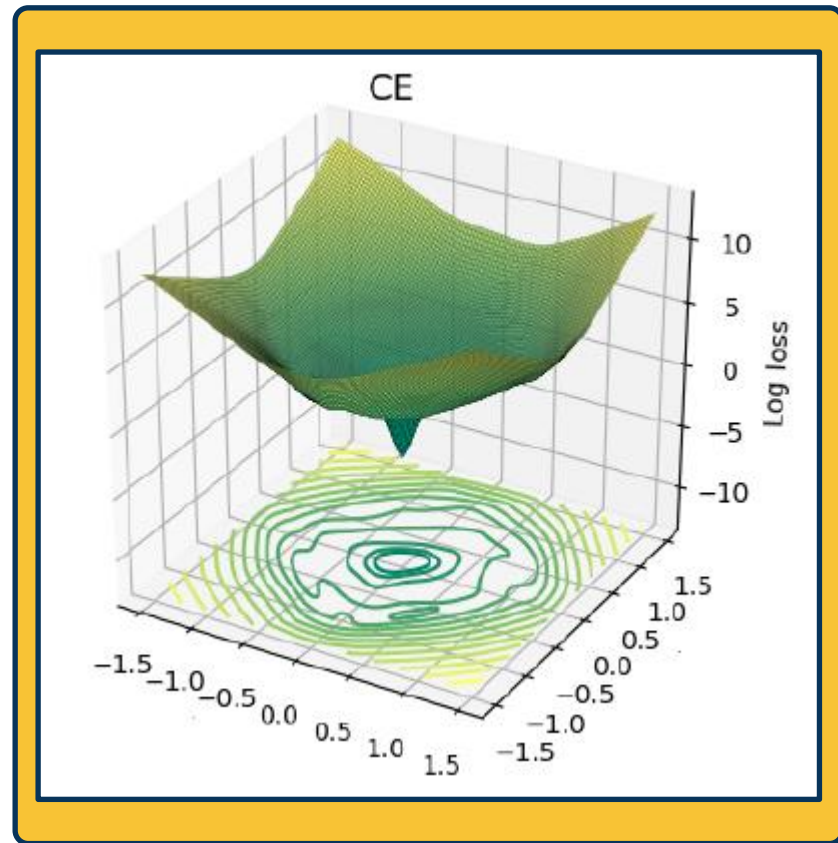


Loss

As weights change, the loss changes as well

- ◆ This is often somewhat-smooth locally, so small changes in weights produce small changes in the loss

We can therefore think about **iterative algorithms** that take **current values of weights** and **modify them a bit**



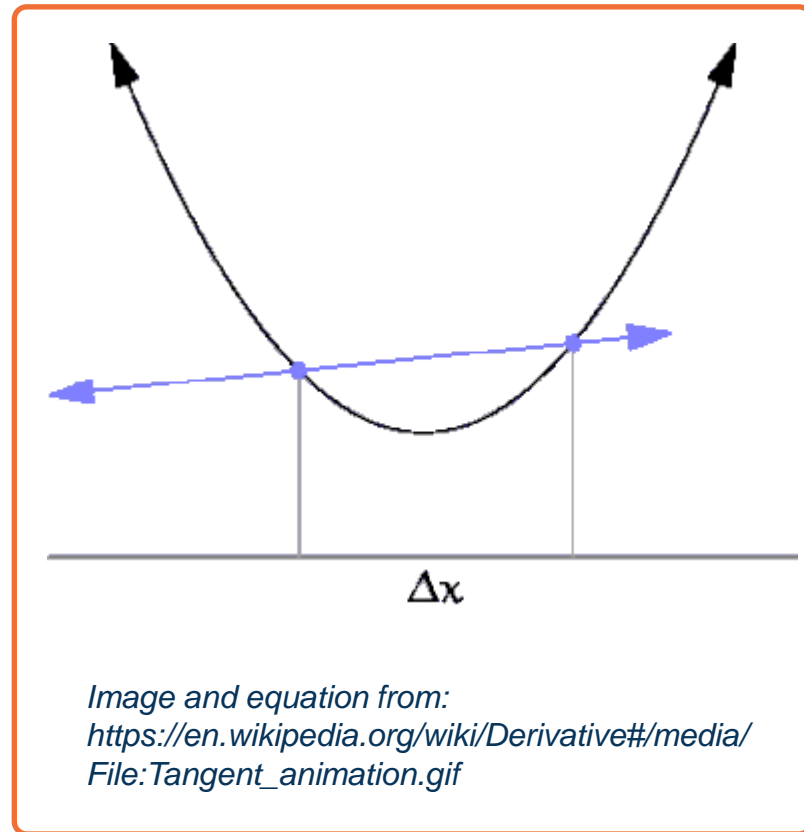


Strategy: Follow the Slope!

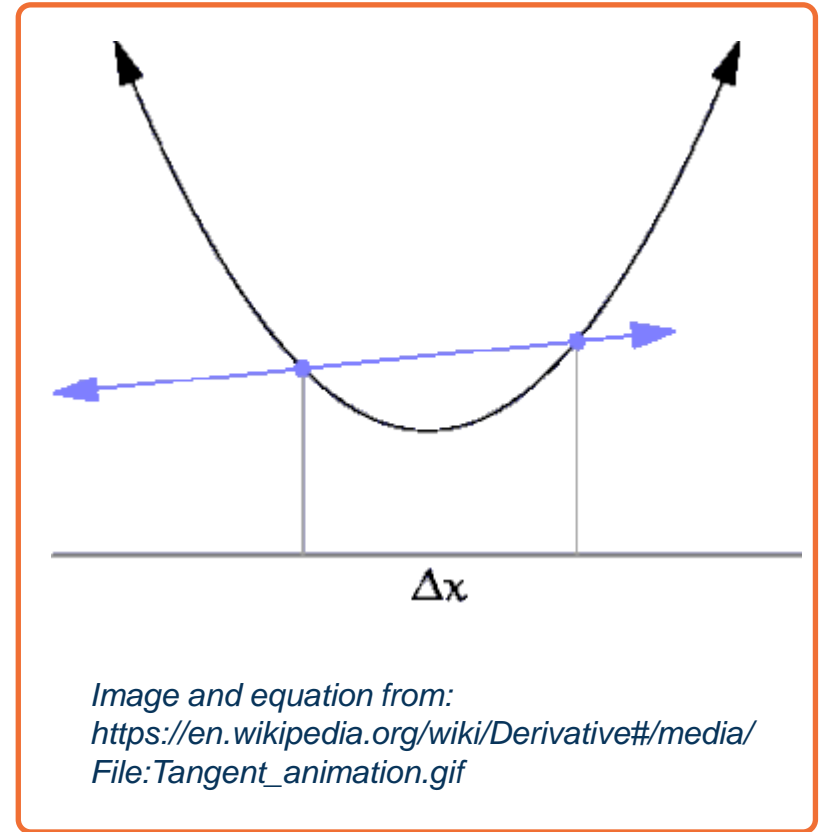
- We can find the steepest descent direction by computing the **derivative (gradient)**:

$$f'(a) = \lim_{h \rightarrow 0} \frac{f(a + h) - f(a)}{h}$$

- Steepest descent direction is the **negative gradient**
- **Intuitively:** Measures how the function changes as the argument a changes by a small step size
 - As step size goes to zero
- **In Machine Learning:** Want to know how the **loss function** changes **as weights** are varied
 - Can consider each parameter separately by taking **partial derivative** of loss function with respect to that parameter



$$f'(a) = \lim_{h \rightarrow 0} \frac{f(a+h) - f(a)}{h}$$

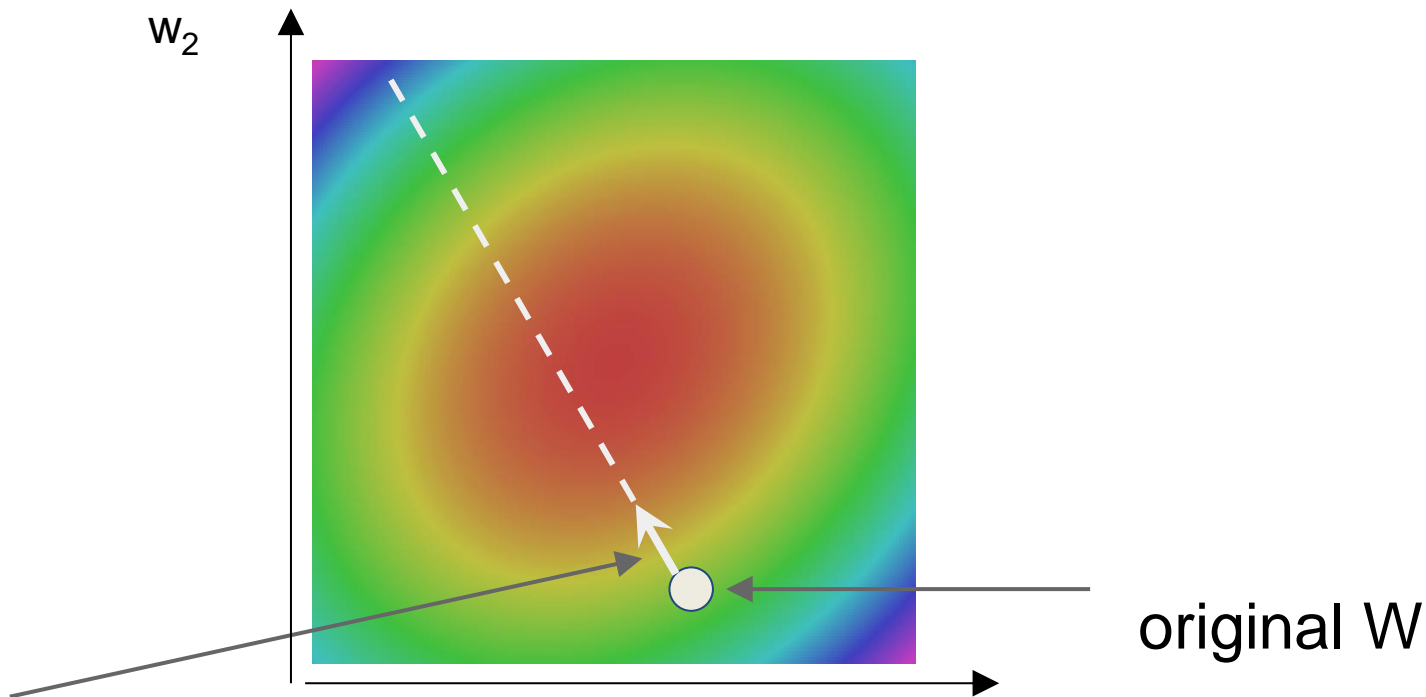


This idea can be turned into an **algorithm (gradient descent)**

1. Choose a model: $f(x, W) = Wx$
2. Choose loss function: $L_i = (y - Wx_i)^2$
3. Calculate partial derivative for each parameter: $\frac{\partial L}{\partial w_i}$
4. Update the parameters: $w_i = w_i - \frac{\partial L}{\partial w_i}$

Instead: Add learning rate to prevent too big of a step: $w_i = w_i - \alpha \frac{\partial L}{\partial w_i}$

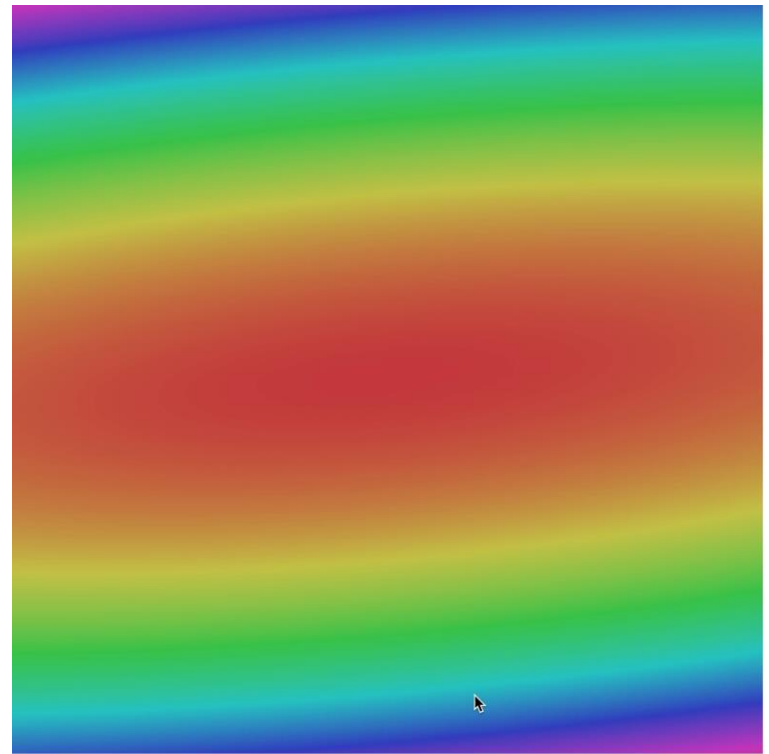
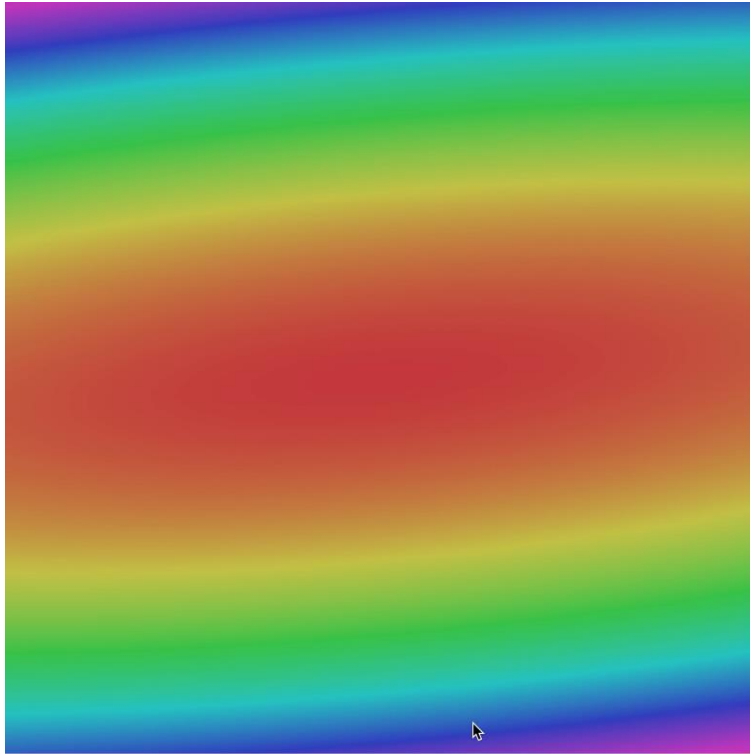
5. Repeat (from Step 3)



negative gradient direction

Gradient Descent

W_1



Gradient Descent

W_1

Often, we only compute the gradients across a small subset of data

◆ Full Batch Gradient Descent $L = \frac{1}{N} \sum L(f(x_i, W), y_i)$

◆ Mini-Batch Gradient Descent $L = \frac{1}{M} \sum L(f(x_i, W), y_i)$

◆ Where M is a *subset* of data

◆ We iterate over mini-batches:

◆ Get mini-batch, compute loss, compute derivatives, and take a set

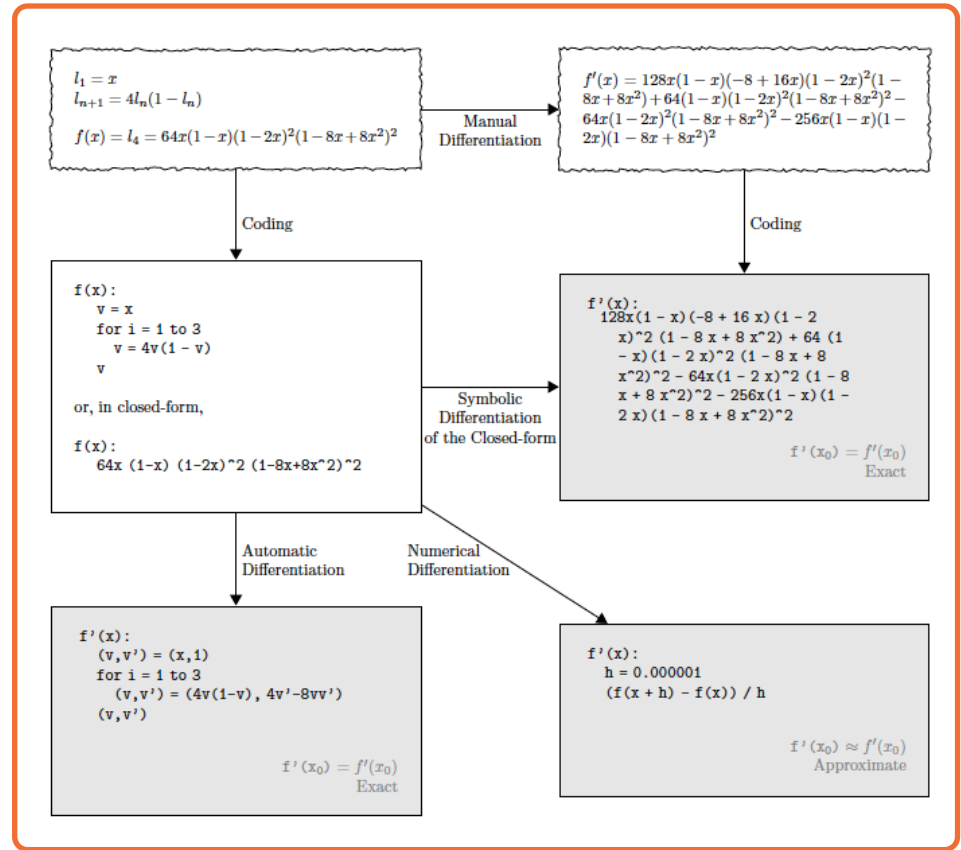
Gradient descent is guaranteed to converge under some conditions

- ◆ For example, learning rate has to be appropriately reduced throughout training
- ◆ It will converge to a *local* minima
 - ◆ Small changes in weights would not decrease the loss
- ◆ It turns out that some of the local minima that it finds in practice (if trained well) are still pretty good!

We know how to compute the model output and loss function

Several ways to compute $\frac{\partial L}{\partial w_i}$

- Manual differentiation
- Symbolic differentiation
- Numerical differentiation
- Automatic differentiation



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,
?,
?,

$$(1.25322 - 1.25347)/0.0001 = -2.5$$

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

?,
?,...]

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,
?,
?,

$$(1.25353 - 1.25347)/0.0001 = 0.6$$

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

?,...]

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):

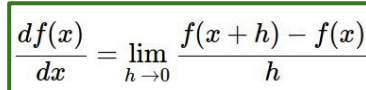
[0.34,
-1.11,
0.78 + **0.0001**,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,...]

loss 1.25347

gradient dW:

[-2.5,
0.6,
0,
?,
?,...]


$$(1.25347 - 1.25347)/0.0001 = 0$$


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

?,...]

Numerical vs Analytic Gradients

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

Numerical gradient: slow :(, approximate :(, easy to write :)

Analytic gradient: fast :), exact :), error-prone :(

In practice: Derive analytic gradient, check your implementation with numerical gradient.

This is called a **gradient check**.

- ◆ Components of parametric classifiers:
 - ◆ Input/Output: Image/Label
 - ◆ Model (function): Linear Classifier + Softmax
 - ◆ Loss function: Cross-Entropy
 - ◆ Optimizer: Gradient Descent

- ◆ Ways to compute gradients
 - ◆ Numerical
 - ◆ Next: Analytical, automatic differentiation