Topics:
• Linear Classification, Loss functions
• Gradient Descent

CS 4644-DL / 7643-A
ZSOLT KIRA
• **Assignment 1 out last week!**
  • Start early, start early, start early!
  • HW1 Tutorial: [https://piazza.com/class/ky0k0ha5vgy1mk?cid=41](https://piazza.com/class/ky0k0ha5vgy1mk?cid=41)

• **Piazza:** Enroll now! [https://piazza.com/gatech/spring2022/cs46447643a](https://piazza.com/gatech/spring2022/cs46447643a) (Code: DLSPR2022)
  • **NOTE:** There is an OMSCS section with a Ed. Make sure you are in the right one

• **Office hours** schedule:
  [https://piazza.com/class/ky0k0ha5vgy1mk?cid=40](https://piazza.com/class/ky0k0ha5vgy1mk?cid=40)
Parametric Model

Explicitly model the function $f : X \rightarrow Y$ in the form of a parametrized function
$f(x, W) = y$, examples:

- Logistic regression/classification
- Neural networks

Capacity (size of hypothesis class) does not grow with size of training data!

Learning is search

Parametric – Linear Classifier

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

Procedure:

Calculate score per class for example
Return label of maximum score (argmax)
Components of a Parametric Model

- 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

Model \( f(x, W) = Wx + b \)

Class Scores

Car  Coffee  Cup  Bird

Loss Function

Optimizer
**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
Deep Learning as Legos

Neural Network

Linear classifiers

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

This image is CC0 1.0 public domain
**Idea:** Separate classes via high-dimensional linear separators (hyper-planes)

- One of the simplest parametric models, **but** surprisingly effective
- Very commonly used!
- Let’s look more closely at each element
To simplify notation we will refer to inputs as $x_1 \cdots x_m$ where $m = n \times n$. 

$$x = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{bmatrix}$$

$$x = \begin{bmatrix} x_{11} \\ x_{12} \\ \vdots \\ x_{21} \\ x_{22} \\ \vdots \\ x_{n1} \\ \vdots \\ x_{nn} \end{bmatrix}$$
Weights

Model
\[
f(x, W) = Wx + b
\]

Classifier for class 1
\[
\begin{bmatrix}
w_{11} & w_{12} & \cdots & w_{1m} \\
w_{21} & w_{22} & \cdots & w_{2m} \\
w_{31} & w_{32} & \cdots & w_{3m}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_m
\end{bmatrix}
+ \begin{bmatrix}
b_1 \\
b_2 \\
b_3
\end{bmatrix}
\]

(Note that in practice, implementations can use \(xW\) instead, assuming a different shape for \(W\). That is just a different convention and is equivalent.)
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!

Model
\[ f(x, W) = Wx + b \]

\[
\begin{bmatrix}
  w_{11} & w_{12} & \cdots & w_{1m} & b_1 \\
  w_{21} & w_{22} & \cdots & w_{2m} & b_2 \\
  w_{31} & w_{32} & \cdots & w_{3m} & b_3 \\
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_m \\
  1
\end{bmatrix}
\]
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Input image

Stretch pixels into column

$W$

$b$

Cat score
Dog score
Ship score

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

We can convert the weight vector back into the shape of the image and visualize
Geometric Viewpoint

\[ f(x, W) = Wx + b \]

Array of \(32 \times 32 \times 3\) numbers (3072 numbers total)

Plot created using Wolfram Cloud

Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n
Hard Cases for a Linear Classifier

Class 1: number of pixels > 0 odd
Class 2: number of pixels > 0 even

Class 1: 1 ≤ L2 norm ≤ 2
Class 2: Everything else

Class 1: Three modes
Class 2: Everything else

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
Components of a Parametric Model

- **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 → Model $f(x,W) = Wx + b$ → Loss Function → Optimizer
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**

\[ f(x, W) = Wx + b \]
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

\[
\begin{align*}
    s &= f(x, W) & \text{Scores} \\
    P(Y = k | X = x) &= \frac{e^{sk}}{\sum_{j} e^{sj}} & \text{Softmax Function}
\end{align*}
\]
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:

\[
L = \frac{1}{N} \sum L_1(f(x_i, W), y_i)
\]
**Multiclass SVM loss:**

Given an example \((x_i, y_i)\) where \(x_i\) is the image and \(y_i\) is the (integer) label, and using the shorthand for the scores vector: \(s = f(x_i, W)\)

the SVM loss has the form:

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

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

*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:

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

= \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:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
<tr>
<td>Losses:</td>
<td>2.9</td>
<td>4.9</td>
<td>2.0</td>
</tr>
</tbody>
</table>

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:

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

Suppose: 3 training examples, 3 classes.

With some \(W\) the scores \(f(x, W) = Wx\) are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Losses:</td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{cat: } & 3.2 \quad \text{1.3} \quad \text{2.2} \\
\text{car: } & 5.1 \quad \text{4.9} \quad \text{2.5} \\
\text{frog: } & -1.7 \quad \text{2.0} \quad \text{-3.1}
\end{align*}
\]

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, \infty]

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

<table>
<thead>
<tr>
<th>Class</th>
<th>Score for cat</th>
<th>Score for car</th>
<th>Score for frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

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

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:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>score</td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
<tr>
<td></td>
<td>1.3</td>
<td>4.9</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>2.2</td>
<td>2.5</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

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?

<table>
<thead>
<tr>
<th>Class</th>
<th>Score 1</th>
<th>Score 2</th>
<th>Score 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

No difference Scaling by constant

*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:

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

\[
L = \frac{(2.9 + 0 + 12.9)}{3} = 5.27
\]

Suppose: 3 training examples, 3 classes.
With some \(W\) the scores \(f(x_i, W) = Wx\) 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
E.g. Suppose that we found a $W$ such that $L = 0$.

Q: Is this $W$ unique?

No $2W$ also has $L=0$

$f(x, W) = Wx$

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

\[
P(Y = k|X = x) = \frac{e^{s_k}}{\sum_j e^{s_j}}
\]

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

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)

\[
\min_w KL(p^* || \hat{p}) = \sum_y p^*(y) \log \frac{p^*(y)}{\hat{p}(y)}
\]

\[
= \sum_y p^*(y) \log(p^*(y)) - \sum_y p^*(y) \log(\hat{p}(y))
\]

\[
-H(p^*) \quad \text{(negative entropy, term goes away because not a function of model, } W, \text{ parameters we are minimizing over)}
\]

\[
H(p^*, \hat{p}) \quad \text{(Cross-Entropy)}
\]

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 $\geq 0$

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

Probabilities must sum to 1

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

Unnormalized log-probabilities / logits

Unnormalized probabilities

Probabilities

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 $\geq 0$

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

Probabilities must sum to 1

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

Infimum is 0, max is unbounded (inf)

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

Infimum is 0, max is unbounded (inf)
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

\[ s = f(x_i; W) \]

Probabilities must be \( \geq 0 \)

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

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
Softmax vs. SVM

matrix multiply + bias offset

Softmax vs. SVM

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

Cross-Entropy Loss Example
If we are performing **regression**, we can directly optimize to match the ground truth value

**Example:** House price prediction

\[
L_i = |y - WX_i| \quad \text{L1}
\]

\[
L_i = (y - WX_i)^2 \quad \text{L2}
\]
Often, we add a **regularization term** to the loss function

L1 Regularization

\[ L_i = |y - W x_i|^2 + |W| \]

**Example regularizations:**

- L1/L2 on weights (encourage small values)
Gradient Descent
Components of a Parametric Model

- 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 → Model: \( f(x, W) = Wx + b \) → Loss Function → Optimizer

Class Scores:
- Car
- Coffee Cup
- Bird
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
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)**:

\[
f'(a) = \lim_{{h \to 0}} \frac{f(a + h) - f(a)}{h}\]

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

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

- Choose a model: $f(x, W) = Wx$
- Choose loss function: $L_i = |y - Wx_i|^2$
- Calculate partial derivative for each parameter: $\frac{\partial L}{\partial w_i}$
- Update the parameters: $w_i = w_i - \frac{\partial L}{\partial w_i}$
- Add learning rate to prevent too big of a step: $w_i = w_i - \alpha \frac{\partial L}{\partial w_i}$
- Repeat (from Step 3)
Gradient Descent

original $W$

negative gradient direction

$w_1$

$w_2$

http://demonstrations.wolfram.com/VisualizingTheGradientVector/
Gradient Descent
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 $W$:

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

loss 1.25347

gradient $dW$:

<table>
<thead>
<tr>
<th>current W:</th>
<th>W + h (first dim):</th>
<th>gradient dW:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34,</td>
<td>[0.34 + 0.0001,</td>
<td>[?,</td>
</tr>
<tr>
<td>-1.11,</td>
<td>-1.11,</td>
<td>?,</td>
</tr>
<tr>
<td>0.78,</td>
<td>0.78,</td>
<td>?,</td>
</tr>
<tr>
<td>0.12,</td>
<td>0.12,</td>
<td>?,</td>
</tr>
<tr>
<td>0.55,</td>
<td>0.55,</td>
<td>?,</td>
</tr>
<tr>
<td>2.81,</td>
<td>2.81,</td>
<td>?,</td>
</tr>
<tr>
<td>-3.1,</td>
<td>-3.1,</td>
<td>?,</td>
</tr>
<tr>
<td>-1.5,</td>
<td>-1.5,</td>
<td>?,</td>
</tr>
<tr>
<td>0.33,...]</td>
<td>0.33,...]</td>
<td>?,...]</td>
</tr>
</tbody>
</table>

loss 1.25347  loss 1.25322

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
<table>
<thead>
<tr>
<th>current W:</th>
<th>W + h (first dim):</th>
<th>gradient dW:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33, ...]</td>
<td>[0.34 + 0.0001, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33, ...]</td>
<td>[-2.5, ?, ?, ?, (1.25322 - 1.25347)/0.0001 = -2.5]</td>
</tr>
<tr>
<td>loss 1.25347</td>
<td>loss 1.25322</td>
<td></td>
</tr>
</tbody>
</table>

\[
\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}
\]
<table>
<thead>
<tr>
<th>current $W$:</th>
<th>$W + h$ (second dim):</th>
<th>gradient $dW$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[0.34, -1.11 + 0.0001, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[-2.5, ?, ?, ?, ?, ?, ?, ?,...]</td>
</tr>
</tbody>
</table>

loss 1.25347  
loss 1.25353  

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

\[
\begin{array}{c}
0.34, \\
-1.11, \\
0.78, \\
0.12, \\
0.55, \\
2.81, \\
-3.1, \\
-1.5, \\
0.33, \ldots
\end{array}
\]

loss 1.25347

$W + h$ (second dim):

\[
\begin{array}{c}
0.34, \\
-1.11 + 0.0001, \\
0.78, \\
0.12, \\
0.55, \\
2.81, \\
-3.1, \\
-1.5, \\
0.33, \ldots
\end{array}
\]

loss 1.25353

gradient $dW$:

\[
\begin{array}{c}
-2.5, \\
0.6, \\
?, \\
?, \\
?, \\
?, \\
?, \ldots
\end{array}
\]

\[
\frac{(1.25353 - 1.25347)}{0.0001} = 0.6
\]
<table>
<thead>
<tr>
<th>current W:</th>
<th>W + h (third dim):</th>
<th>gradient dW:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[0.34, -1.11, 0.78 + 0.0001, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[-2.5, 0.6, ?, ?, ?, ?, ?,...]</td>
</tr>
</tbody>
</table>

loss 1.25347  
loss 1.25347

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

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

**loss 1.25347**

### \(W + h\) (third dim):

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

**loss 1.25347**

### gradient \(dW:\)

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

\[
\frac{(1.25347 - 1.25347) \times 0.0001}{0.0001} = 0
\]
Numerical vs Analytic Gradients

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

**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
For some functions, we can analytically derive the partial derivative.

**Example:**

<table>
<thead>
<tr>
<th>Function</th>
<th>Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(w, x_i) = w^T x_i$</td>
<td>$(y_i - w^T x_i)^2$</td>
</tr>
</tbody>
</table>

(Assume $w$ and $x_i$ are column vectors, so same as $w \cdot x_i$)

**Update Rule**

$$w_j \leftarrow w_j + 2\eta \sum_{k=1}^{N} \delta_k x_{kj}$$