Topics:
• Linear Classification, Loss functions
• Gradient Descent

CS 4803-DL / 7643-A
ZSOLT KIRA
• **Assignment 1 out today!**
  • Start early, start early, start early!

• **Piazza:** Enroll now! [https://piazza.com/class/kjsselshfiz18c](https://piazza.com/class/kjsselshfiz18c) (Code: DL2021)
  • **NOTE:** There is an OMSCS section with a DIFFERENT piazza. Make sure you are in the right one

• **Office hours** start this week
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

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

Procedure:
Calculate score per class for example
Return label of maximum score (argmax)
- 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

Components of a Parametric Model

- Data: Image
- Features: Histogram
- 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
What is the simplest function you can think of?

Our model is:

\[ f(x, w) = w \cdot x + b \]

(Note if \( w \) and \( x \) are column vectors we often show this as \( w^T x \))

Image adapted from:
https://en.wikipedia.org/wiki/Linear_equation#/media/File:Linear_Function_Graph.svg
Linear Classification and Regression

Simple linear classifier:

- Calculate score: 
  $$f(x, w) = w \cdot x + b$$
- Binary classification rule ($w$ is a vector): 
  $$y = \begin{cases} 
  1 & \text{if } f(x, w) \geq 0 \\
  0 & \text{otherwise} 
  \end{cases}$$
- For multi-class classifier take class with highest (max) score 
  $$f(x, W) = Wx + b$$
**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 $m = n \times n$
Model

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

Classifier for class 1
\[ \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \]

Classifier for class 2
\[ \begin{bmatrix} w_{21} & w_{22} & \cdots & w_{2m} \end{bmatrix} \]

Classifier for class 3
\[ \begin{bmatrix} w_{31} & w_{32} & \cdots & w_{3m} \end{bmatrix} \]

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

Input image: 56 231 24 2

Stretch pixels into column:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>-0.5</td>
<td>0.1</td>
<td>2.0</td>
</tr>
<tr>
<td>1.5</td>
<td>1.3</td>
<td>2.1</td>
<td>0.0</td>
</tr>
<tr>
<td>0</td>
<td>0.25</td>
<td>0.2</td>
<td>-0.3</td>
</tr>
</tbody>
</table>

\[ W \]

\[ b \]

\[
\begin{align*}
56 & \quad 231 & \quad 24 & \quad 2 \\
1.1 & \quad 3.2 & \quad -1.2 & \\
\end{align*}
\]

\[
\begin{align*}
-96.8 & \quad 437.9 & \quad 61.95 \\
\end{align*}
\]

Cat score: -96.8
Dog score: 437.9
Ship score: 61.95

Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n
We can convert the weight vector back into the shape of the image and visualize.
Interpreting a Linear Classifier

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

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

Geometric Viewpoint

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: $1 \leq L2$ norm $\leq 2$
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
Linear Classifier: Three Viewpoints

**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

Class Scores
- Car
- Coffee Cup
- Bird
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

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

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

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 \(y_i\) 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>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>

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:

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

\[
= \max(0, 1.3 - 4.9 + 1) + \max(0, 2.0 - 4.9 + 1)
\]

\[
= \max(0, -2.6) + \max(0, -1.9)
\]

\[
= 0 + 0
\]

\[
= 0
\]

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

<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>

**Losses:** 0.0

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 \(f(x, W) = Wx\) are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Losses:</td>
<td>2.9</td>
<td>0</td>
<td>12.9</td>
</tr>
<tr>
<td></td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td></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:

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

### 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

Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n
Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x,W) = Wx$ are:

<table>
<thead>
<tr>
<th>Class</th>
<th>Cat</th>
<th>Car</th>
<th>Frog</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

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]

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?

\[
\begin{array}{ccc}
\text{cat} & 3.2 & 1.3 & 2.2 \\
\text{car} & 5.1 & 4.9 & 2.5 \\
\text{frog} & -1.7 & 2.0 & -3.1 \\
\end{array}
\]

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
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:**

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

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

No difference (add constant $1$)

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 \( \mathbf{W} \) the scores \( f(x, \mathbf{W}) = \mathbf{W}x \) 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></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
\[ f(x, W) = Wx \]

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

E.g. Suppose that we found a \( W \) such that \( L = 0 \).

Q: Is this \( W \) unique?

No 2W also has L=0

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

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

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**.
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

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

**Cross-Entropy Loss Example**

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 \( \geq 0 \)

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

Probabilities must sum to 1

\[ \mathcal{L}_i = -\log P(Y = y_i|X = x_i) \]

Q: What is the min/max of possible loss \( \mathcal{L}_i \)?

Infimum is 0, max is unbounded (inf)

\[ \mathcal{L}_i = -\log(0.13) \]

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 \( \geq 0 \)

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

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

\[
\begin{array}{c|c|c|c|c}
W & x_i & b & y_i & 2 \\
0.01 & 0.7 & 0.0 & \text{0.0} & -15 \\
-0.05 & 0.2 & 0.05 & 0.0 & 22 \\
0.1 & -0.45 & 0.16 & 0.2 & -44 \\
0.05 & 0.03 & -0.3 & -0.3 & 56 \\
\end{array}
\]

**hinge loss (SVM)**

\[
\text{max}(0, -2.85 - 0.28 + 1) + \\
\text{max}(0, 0.86 - 0.28 + 1) = 1.58
\]

**cross-entropy loss (Softmax)**

\[
\begin{array}{c|c|c|c|c}
\text{exp} & x_i & 0.058 & 0.016 & \text{normalize} \\
-2.85 & 0.86 & 2.36 & 0.631 & \text{(to sum to one)} \\
0.28 & 0.28 & 1.32 & 0.353 & \text{- log(0.353)} \\
\end{array}
\]

\[
= 0.452
\]

Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n
If we are performing **regression**, we can directly optimize to match the ground truth value.

- **Example**: House price prediction

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

\[
L_i = |y - Wx_i|^2 \quad \text{L2}
\]

- For probabilities

\[
L_i = |y - Wx_i| = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{Logistic}
\]

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

Class Scores

Loss Function

Optimizer
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: \( f(x, \mathbf{W}) = \mathbf{W}x \)

- Choose loss function: \( L_i = |y - \mathbf{W}x_i|^2 \)

- 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

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
<table>
<thead>
<tr>
<th>current $W$:</th>
<th>gradient $dW$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34,</td>
<td>[?,</td>
</tr>
<tr>
<td>-1.11,</td>
<td>?,</td>
</tr>
<tr>
<td>0.78,</td>
<td>?,</td>
</tr>
<tr>
<td>0.12,</td>
<td>?,</td>
</tr>
<tr>
<td>0.55,</td>
<td>?,</td>
</tr>
<tr>
<td>2.81,</td>
<td>?,</td>
</tr>
<tr>
<td>-3.1,</td>
<td>?,</td>
</tr>
<tr>
<td>-1.5,</td>
<td>?,</td>
</tr>
<tr>
<td>0.33,...]</td>
<td>?,</td>
</tr>
<tr>
<td><strong>loss 1.25347</strong></td>
<td><strong>[?,...,]</strong></td>
</tr>
</tbody>
</table>

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,</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
current $W$:  

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

loss $1.25347$

$W + h$ (first dim):  

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

loss $1.25322$

gradient $dW$:  

\[
\begin{bmatrix}
-2.5, \\
?, \\
?, \\
(1.25322 - 1.25347)/0.0001 = -2.5 \\
?, \\
?, \ldots
\end{bmatrix}
\]
<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>
<tr>
<td><strong>loss 1.25347</strong></td>
<td><strong>loss 1.25353</strong></td>
<td></td>
</tr>
</tbody>
</table>

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

$W + h$ (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 

$(1.25353 - 1.25347)/0.0001 = 0.6$ 

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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
<table>
<thead>
<tr>
<th>current $W$:</th>
<th>$W + h$ (third dim):</th>
<th>gradient $dW$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34,</td>
<td>[0.34,</td>
<td>[-2.5,</td>
</tr>
<tr>
<td>-1.11,</td>
<td>-1.11,</td>
<td>0.6,</td>
</tr>
<tr>
<td>0.78,</td>
<td>0.78 + 0.0001,</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 1.25347                |
current $W$:  

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

loss 1.25347

$W + h$ (third dim):  

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

loss 1.25347

gradient $dW$:  

\[
\begin{bmatrix}
-2.5, \\
0.6, \\
0, \\
?, \\
?, \\
?, \\
?, \\
?, \\
?, \\
\end{bmatrix}
\]

\[
\frac{(1.25347 - 1.25347)}{0.0001} = 0
\]
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.

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

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

**Derivation of Update Rule**
For some functions, we can analytically derive the partial derivative

**Example:**

Function: 
\[ f(w, x_i) = w^T x_i \]

Loss: 
\[ (y_i - w^T x_i)^2 \]

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

**Update Rule**
\[ w_j \leftarrow w_j + 2\eta \sum_{k=1}^{N} \delta_k x_{kj} \]

**Derivation of Update Rule**

\[
L = \sum_{k=1}^{N} (y_k - w^T x_k)^2
\]

\[
\frac{\partial L}{\partial w_j} = \sum_{k=1}^{N} \frac{\partial}{\partial w_j} (y_k - w^T x_k)^2
\]

Gradient descent tells us we should update \( w \) as follows to minimize \( L \):

\[
w_j \leftarrow w_j - \eta \frac{\partial L}{\partial w_j}
\]

So what's \( \frac{\partial L}{\partial w_j} \)?

\[
\frac{\partial L}{\partial w_j} = -2 \sum_{k=1}^{N} \delta_k \frac{\partial}{\partial w_j} w^T x_k
\]

...where...

\[
\delta_k = y_k - w^T x_k
\]

\[
\frac{\partial}{\partial w_j} w^T x_k = \sum_{l=1}^{m} w_l x_{kl}
\]

\[
= -2 \sum_{k=1}^{N} \delta_k x_{kj}
\]
If we add a non-linearity (sigmoid), derivation is more complex

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

First, one can derive that: \( \sigma'(x) = \sigma(x)(1 - \sigma(x)) \)

\[ f(x) = \sigma \left( \sum_k w_k x_k \right) \]

\[ L = \sum_l \left( y_l - \sigma \left( \sum_k w_k x_{lk} \right) \right)^2 \]

\[ \frac{\partial L}{\partial w_j} = \sum_l 2 \left( y_l - \sigma \left( \sum_k w_k x_{lk} \right) \right) \left( - \frac{\partial}{\partial w_j} \sigma \left( \sum_k w_k x_{lk} \right) \right) \]

\[ = \sum_l -2 \left( y_l - \sigma \left( \sum_k w_k x_{lk} \right) \right) \sigma' \left( \sum_k w_k x_{lk} \right) \frac{\partial}{\partial w_j} \sum_k w_k x_{lk} \]

\[ = \sum_l -2 \delta_l \sigma(d_i) (1 - \sigma(d_i)) x_{ij} \]

where \( \delta_l = y_l - f(x_i) \quad d_i = \sum w_k x_{ik} \)

The sigmoid perception update rule:

\[ w_j \leftarrow w_j + 2\eta \sum_{k=1}^{N} \delta_i \sigma_i (1 - \sigma_i) x_{ij} \]

where \( \sigma_i = \sigma \left( \sum_{j=1}^{m} w_j x_{ij} \right) \quad \delta_i = y_i - \sigma_i \)
Given a library of simple functions

\[ \sin(x), \log(x), \cos(x), x^3, \exp(x) \]

Compose into a complicate function

\[ -\log \left( \frac{1}{1 + e^{-w \cdot x}} \right) \]

\[ w \cdot x \rightarrow u \rightarrow \frac{1}{1 + e^{-u}} \rightarrow p \rightarrow -\log(p) \rightarrow L \]

Adapted from slides by: Marc'Aurelio Ranzato, Yann LeCun
Linear Algebra View: Vector and Matrix Sizes
\[
\begin{bmatrix}
  w_{11} & w_{12} & \cdots & w_{1m} & b_1 \\
  w_{21} & w_{22} & \cdots & w_{2m} & b_2 \\
  w_{31} & w_{32} & \cdots & w_{3m} & b_3
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_m \\
  1
\end{bmatrix}
\]

\[W \times x\]

Sizes: \([c \times (d + 1)]\) \([((d + 1) \times 1] \]

Where \(c\) is number of classes

\(d\) is dimensionality of input
Conventions:

- Size of derivatives for scalars, vectors, and matrices:
  Assume we have scalar \( s \in \mathbb{R}^1 \), vector \( v \in \mathbb{R}^m \), i.e. \( v = [v_1, v_2, \ldots, v_m]^T \) and matrix \( M \in \mathbb{R}^{k \times \ell} \).

- What is the size of \( \frac{\partial v}{\partial s} \)? \( \mathbb{R}^{m \times 1} \) (column vector of size \( m \)).

- What is the size of \( \frac{\partial s}{\partial v} \)? \( \mathbb{R}^{1 \times m} \) (row vector of size \( m \)).

\[
\begin{bmatrix}
\frac{\partial v_1}{\partial s} \\
\frac{\partial v_2}{\partial s} \\
\vdots \\
\frac{\partial v_m}{\partial s}
\end{bmatrix}
\]

\[
\begin{bmatrix}
\frac{\partial s}{\partial v_1} & \frac{\partial s}{\partial v_1} & \cdots & \frac{\partial s}{\partial v_m}
\end{bmatrix}
\]
Conventions:

- What is the size of $\frac{\partial v^1}{\partial v^2}$? A matrix:

  $\begin{bmatrix}
  \frac{\partial v^1}{\partial v^1} & \cdots & \cdots & \cdots \\
  \vdots & \ddots & \cdots & \cdots \\
  \vdots & \cdots & \ddots & \cdots \\
  \vdots & \cdots & \cdots & \ddots
  \end{bmatrix}$

- This matrix of partial derivatives is called a **Jacobian**

(Note this is slightly different convention than on [Wikipedia](https://en.wikipedia.org/wiki/Jacobian_matrix))
Conventions:

What is the size of $\frac{\partial s}{\partial M}$? A matrix:

$$
\begin{bmatrix}
\frac{\partial s}{\partial m_{[1,1]}} & \cdots & \cdots & \cdots & \cdots \\
\vdots & \ddots & \cdots & \cdots & \cdots \\
\vdots & \cdots & \frac{\partial s}{\partial m_{[i,j]}} & \cdots & \cdots \\
\vdots & \cdots & \cdots & \cdots & \cdots \\
\vdots & \cdots & \cdots & \cdots & \cdots \\
\end{bmatrix}
$$
- What is the size of $\frac{\partial L}{\partial W}$?

- Remember that loss is a **scalar** and $W$ is a matrix:

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

Jacobian is also a matrix:

$$
\begin{bmatrix}
\frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{12}} & \ldots & \frac{\partial L}{\partial w_{1m}} & \frac{\partial L}{\partial b_1} \\
\frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{22}} & \ldots & \frac{\partial L}{\partial w_{2m}} & \frac{\partial L}{\partial b_2} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\frac{\partial L}{\partial w_{31}} & \frac{\partial L}{\partial w_{32}} & \ldots & \frac{\partial L}{\partial w_{3m}} & \frac{\partial L}{\partial b_3}
\end{bmatrix}
$$
Batches of data are **matrices** or **tensors** (multi-dimensional matrices)

**Examples:**
- Each instance is a vector of size $m$, our batch is of size $[B \times m]$
- Each instance is a matrix (e.g. grayscale image) of size $W \times H$, our batch is $[B \times W \times H]$
- Each instance is a multi-channel matrix (e.g. color image with R,B,G channels) of size $C \times W \times H$, our batch is $[B \times C \times W \times H]$

**Jacobians become tensors which is complicated**
- Instead, flatten input to a vector and get a vector of derivatives!
- This can also be done for partial derivatives between two vectors, two matrices, or two tensors