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

- **Piazza**: Enroll now! 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

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

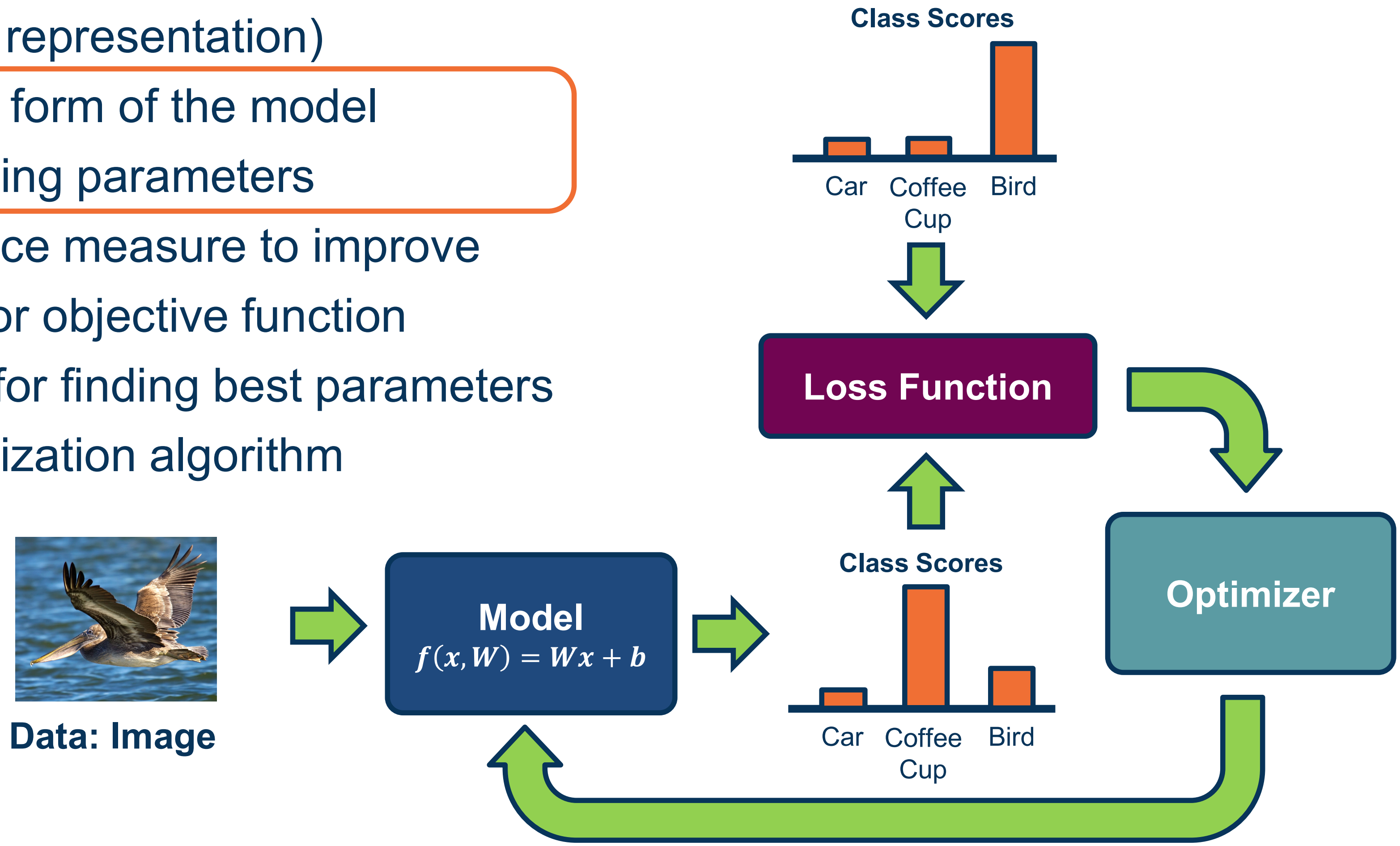
$$f(x, W) = Wx + b$$

Procedure:

Calculate score per class for example

Return label of maximum score (argmax)

- 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

$$f(x, w) = y$$

Classifier

Input (vector)

Weights

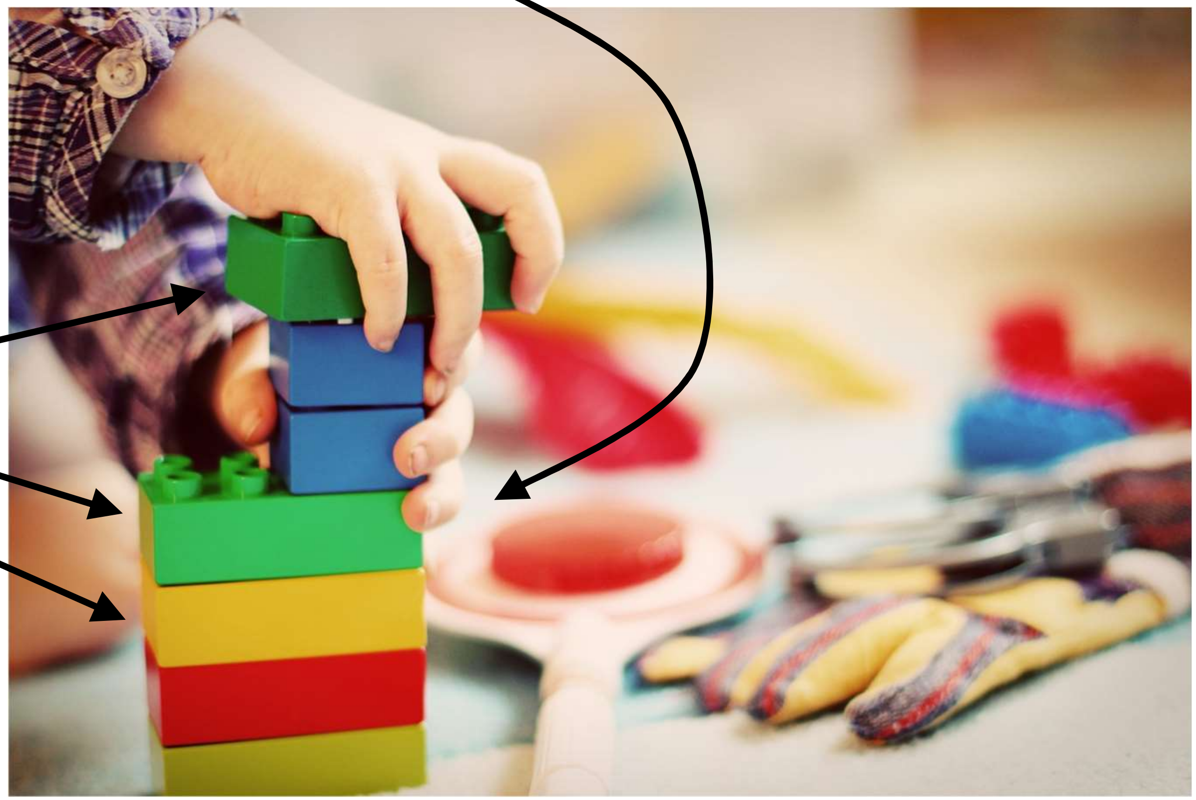
Output (scalar or vector)

- **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

Model: Discriminative Parameterized Function

Neural Network

Linear
classifiers



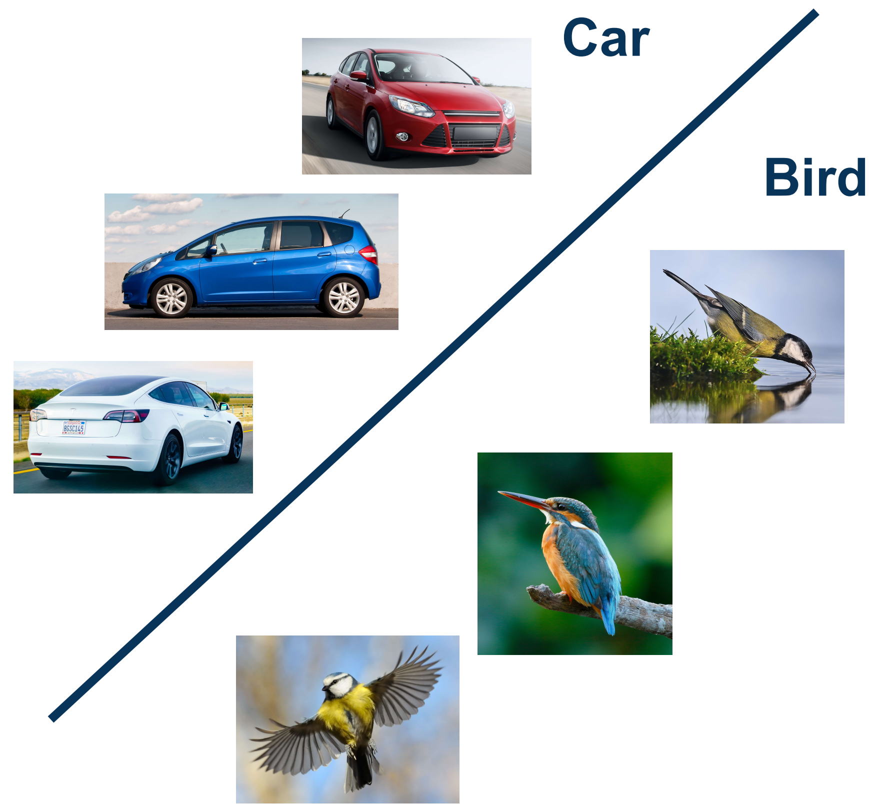
This image is CC0 1.0 public domain

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

Deep Learning as Legos

Georgia Tech

- **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



Data: Image

Model

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

Class Scores

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

Flatten

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

To simplify notation we will refer to inputs as $x_1 \cdots x_m$ where $m = n \times n$

Input Dimensionality

Georgia Tech

Model

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

Classifier for class 1
Classifier for class 2
Classifier for class 3

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

$$\boldsymbol{W} \qquad \boldsymbol{x} \qquad \boldsymbol{b}$$

(Note that in practice, implementations can use $\mathrm{x}\mathrm{W}$ instead, assuming a different shape for W. That is just a different convention and is equivalent.)

Weights

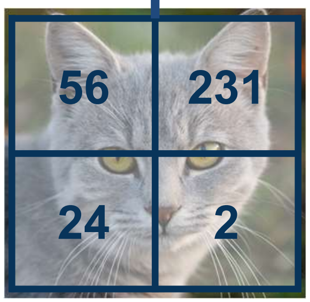
- We can move the bias term into the weight matrix, and a "1" at the end of the input
- Results in **one matrix-vector multiplication**!

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

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

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

Stretch pixels into column



Input image

$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W} \begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{b} = \begin{array}{c} \begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix} \end{array} \begin{array}{l} \text{Cat score} \\ \text{Dog score} \\ \text{Ship score} \end{array}$$

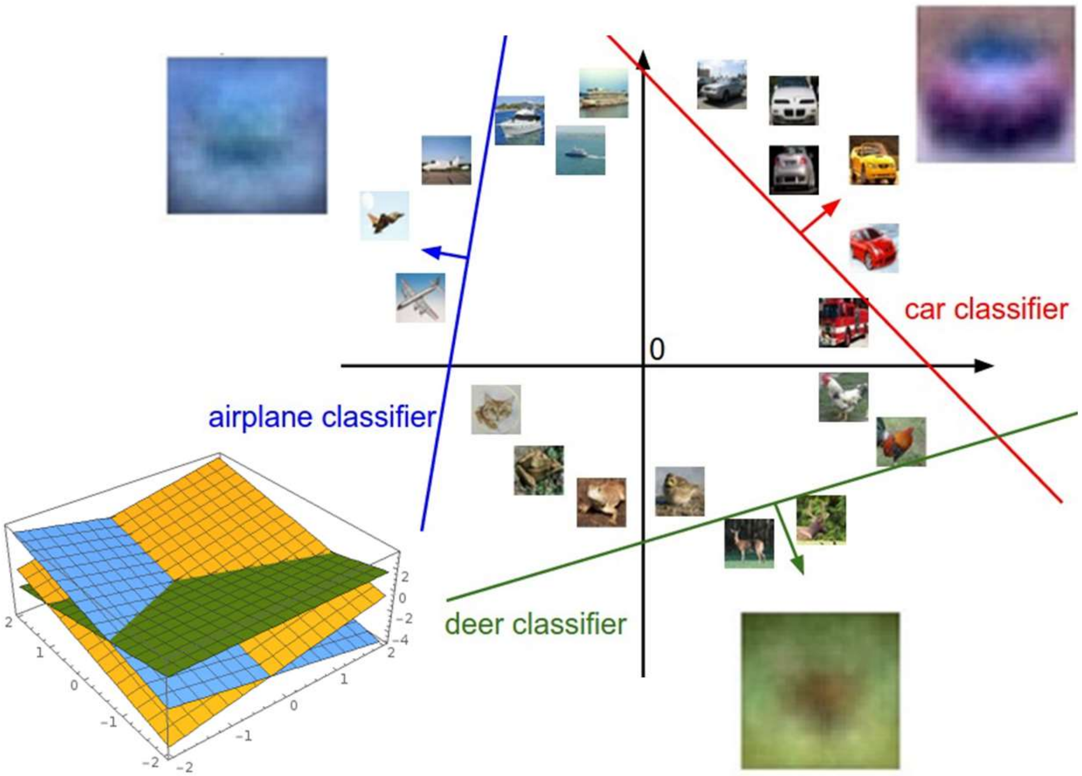
Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Visual Viewpoint

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

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

Interpreting a Linear Classifier



Plot created using Wolfram Cloud

Geometric Viewpoint

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

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

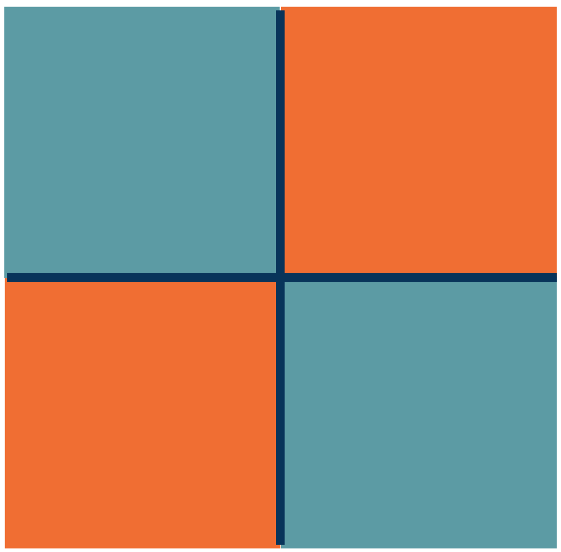
Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Interpreting a Linear Classifier

Hard Cases for a Linear Classifier

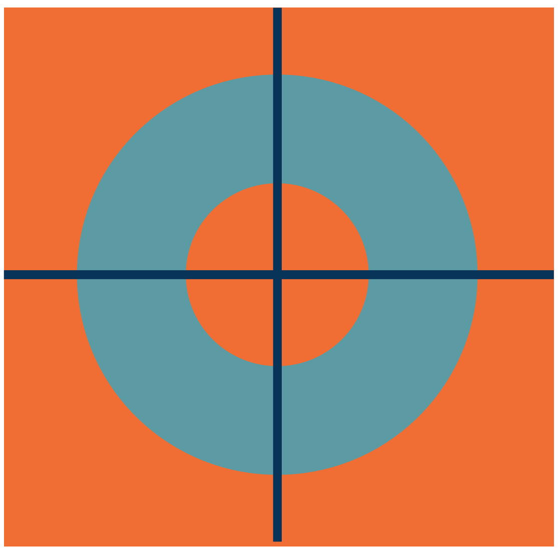
Class 1:
number of pixels > 0 odd

Class 2:
number of pixels > 0 even



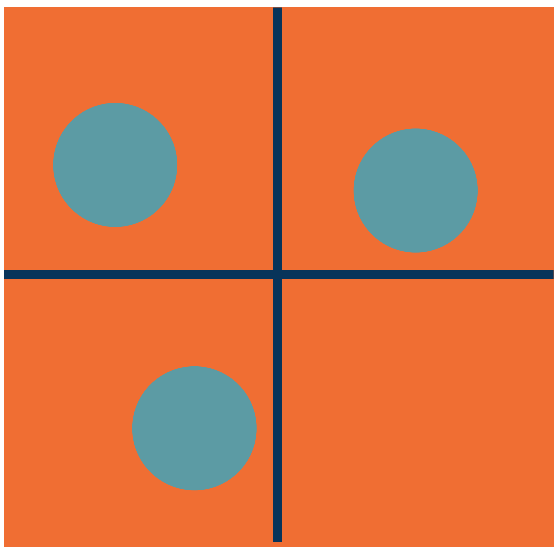
Class 1:
1 < = L2 norm < = 2

Class 2:
Everything else



Class 1:
Three modes

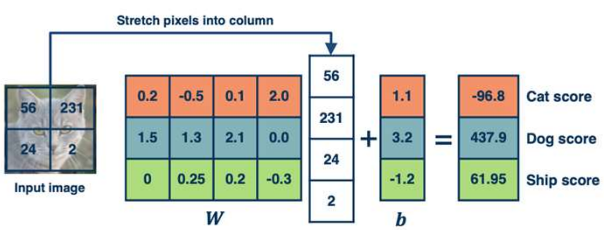
Class 2:
Everything else



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

Algebraic Viewpoint

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

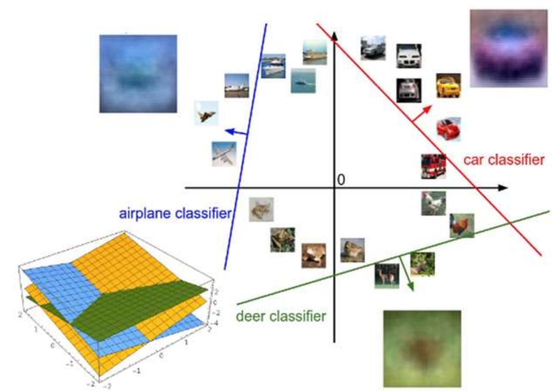


Visual Viewpoint

One template per class

Geometric Viewpoint

Hyperplanes cutting up space

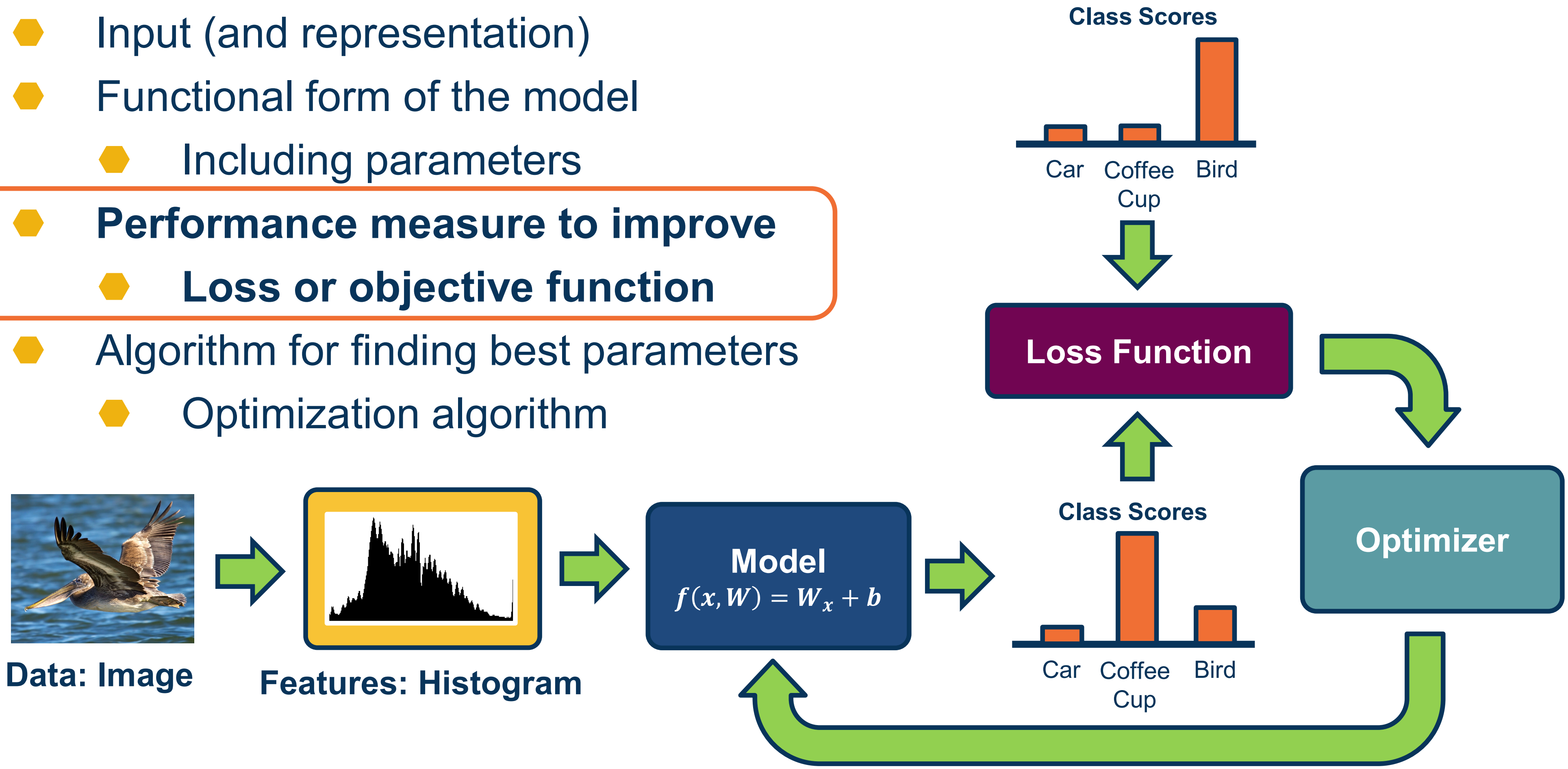


Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Performance Measure for a Classifier

Georgia Tech

- 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



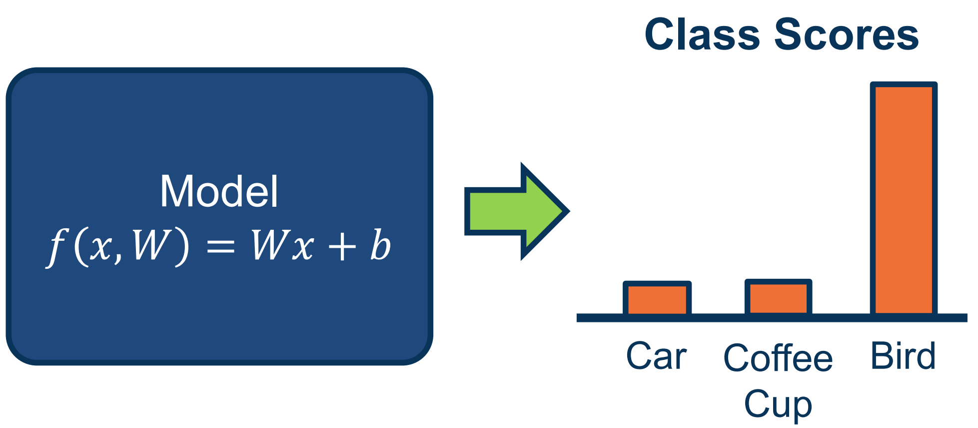
Components of a Parametric Model

Classification using Scores

- The output of a classifier can be considered a **score**
- For binary classifier, use rule:

$$\boldsymbol{y} = \begin{cases} \mathbf{1} & \textbf{if } \boldsymbol{f(x, w) >= 0} \\ \mathbf{0} & \textbf{otherwise} \end{cases}$$

  - 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



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

$$s = f(x, W)$$ **Scores**

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

Converting Scores to Probabilities

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:

$$L = \frac{1}{N} \sum L_1(f(x_i, W), y_i)$$

Performance Measure

Multiclass SVM loss:

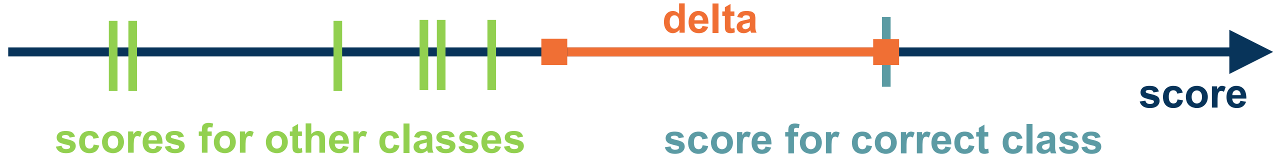
Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label,

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

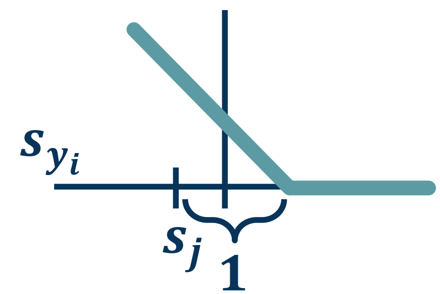
the SVM loss has the form:

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

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



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:

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

= 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**		

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

SVM Loss Example

Georgia Tech

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:	**0.0**	

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

SVM Loss Example

Georgia Tech

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 $\boldsymbol{W}$ the scores $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{W})=\boldsymbol{W}\boldsymbol{x}$ 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

SVM Loss Example

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 $\boldsymbol{W}$ the scores $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{W})=\boldsymbol{W}\boldsymbol{x}$ 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 $\boldsymbol{W}$ the scores $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{W})=\boldsymbol{W}\boldsymbol{x}$ are:

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



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

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

C-1

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

SVM Loss Example

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 $\boldsymbol{W}$ the scores $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{W})=\boldsymbol{W}\boldsymbol{x}$ 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 $\boldsymbol{W}$ the scores $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{W}) = \boldsymbol{W}\boldsymbol{x}$ 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

SVM Loss Example

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:



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

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

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

$$s = f(x, W)$$ **Scores**

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

Performance Measure for Probabilities

- 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)

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

Ground Truth **Prediction**

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

$$= \underbrace{\sum_y p^*(y) \log(p^*(y))}_{-H(p^*)} - \underbrace{\sum_y p^*(y) \log(\hat{p}(y))}_{H(p^*,\hat{p})}$$

$-H(p^*)$ (negative entropy, term goes away because not a function of model, *W*, parameters we are minimizing over)

$H(p^*, \hat{p})$ (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): $\min_w (-log\, \hat{p}(y_i))$

Performance Measure for Probabilities

Georgia Tech

Softmax Classifier (Multinomial Logistic Regression)

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

Probabilities must be >= 0

Probabilities must sum to 1

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

	Unnormalized log-probabilities / logits	$\xrightarrow{\text{exp}}$	Unnormalized probabilities	$\xrightarrow{\text{normalize}}$	Probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

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

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**

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

Probabilities must be >= 0

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

Softmax Function

Probabilities must sum to 1

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

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

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**

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

Probabilities must be >= 0

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

Softmax Function

Probabilities must sum to 1

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

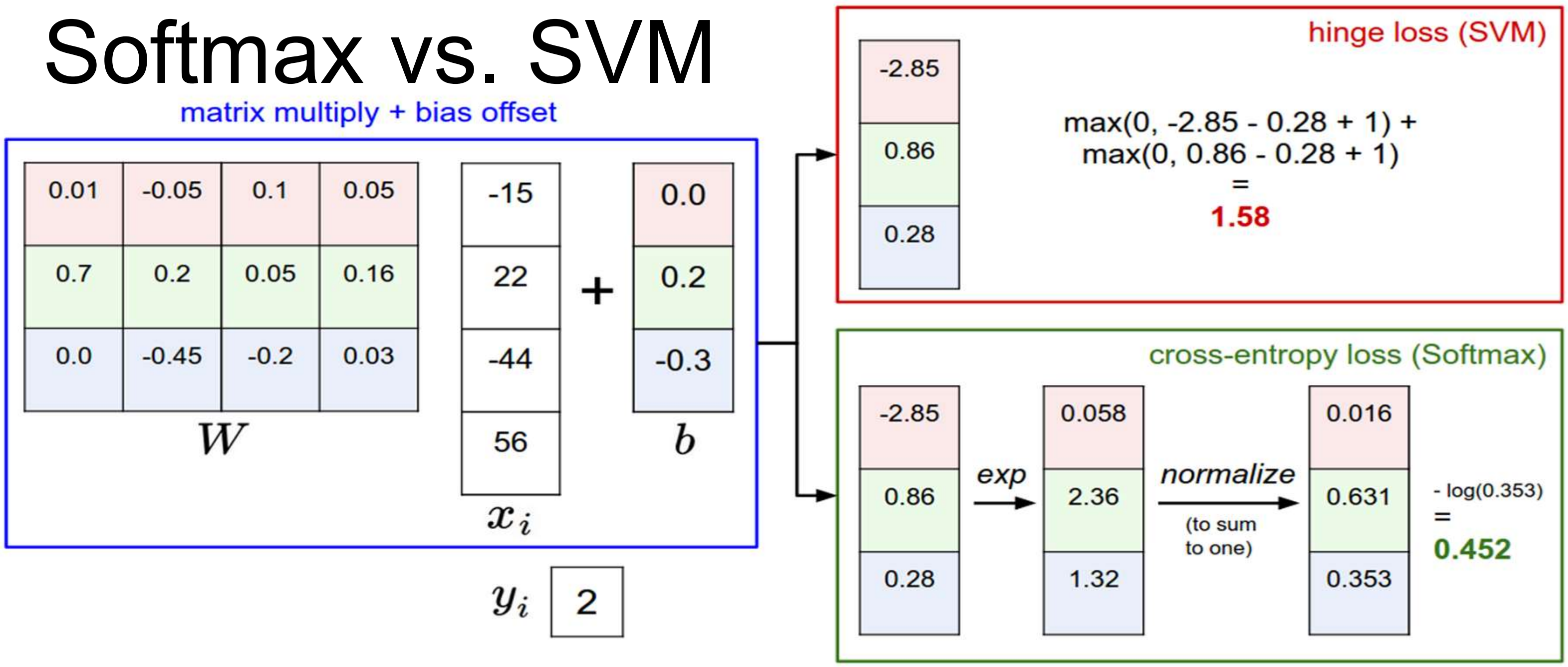
$$L_i = -\log(0.13)$$

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

Log(C), e.g. log(10) ≈ 2

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

Softmax vs. SVM



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

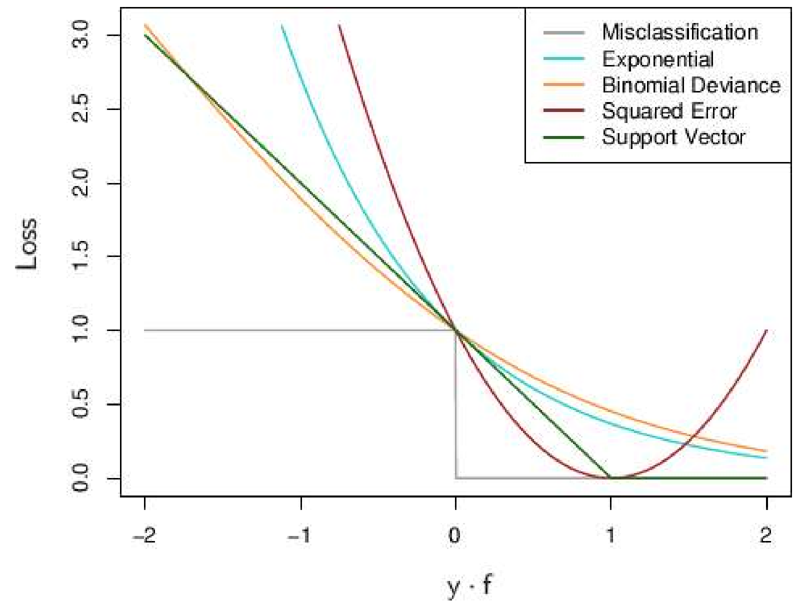
Cross-Entropy Loss Example

If we are performing **regression,** we can directly optimize to match the ground truth value

- **Example:** House price prediction

$$L_i = |y - Wx_i| \qquad \textbf{L1}$$

$$L_i = |y - Wx_i|^2 \qquad \textbf{L2}$$



Source: https://raw.githubusercontent.com/rohan-varma/rohan-blog/gh-pages/images/loss3.jpg

Regression Example

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

L1 Regularization

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

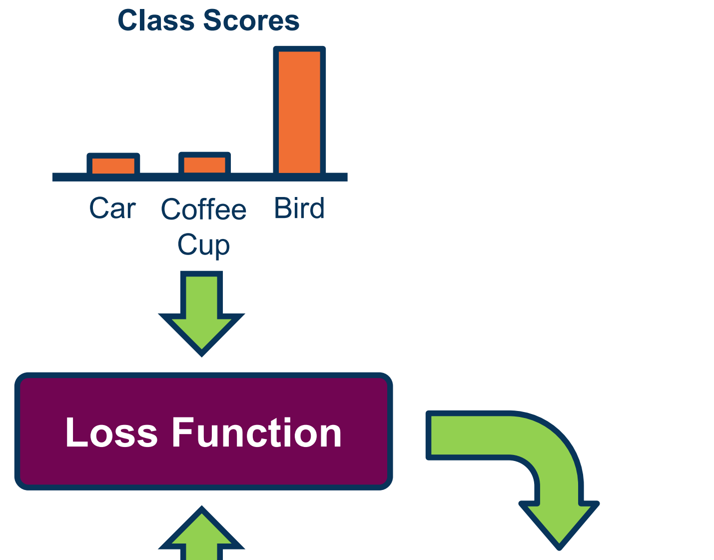
Example regularizations:

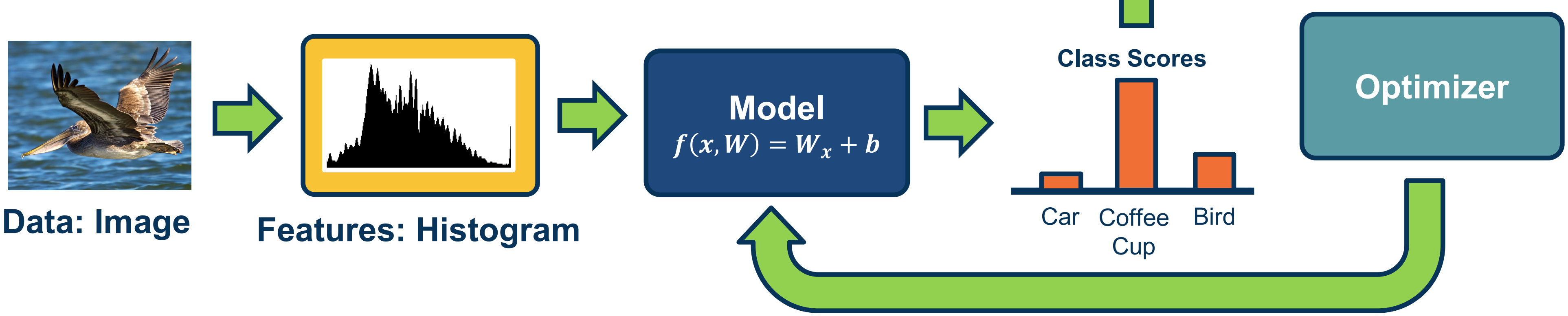
- L1/L2 on weights (encourage small values)

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**





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

$$\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} & b1 \\ w_{21} & w_{22} & \cdots & w_{2m} & b2 \\ w_{21} & w_{22} & \cdots & w_{3m} & b3 \end{bmatrix}$$

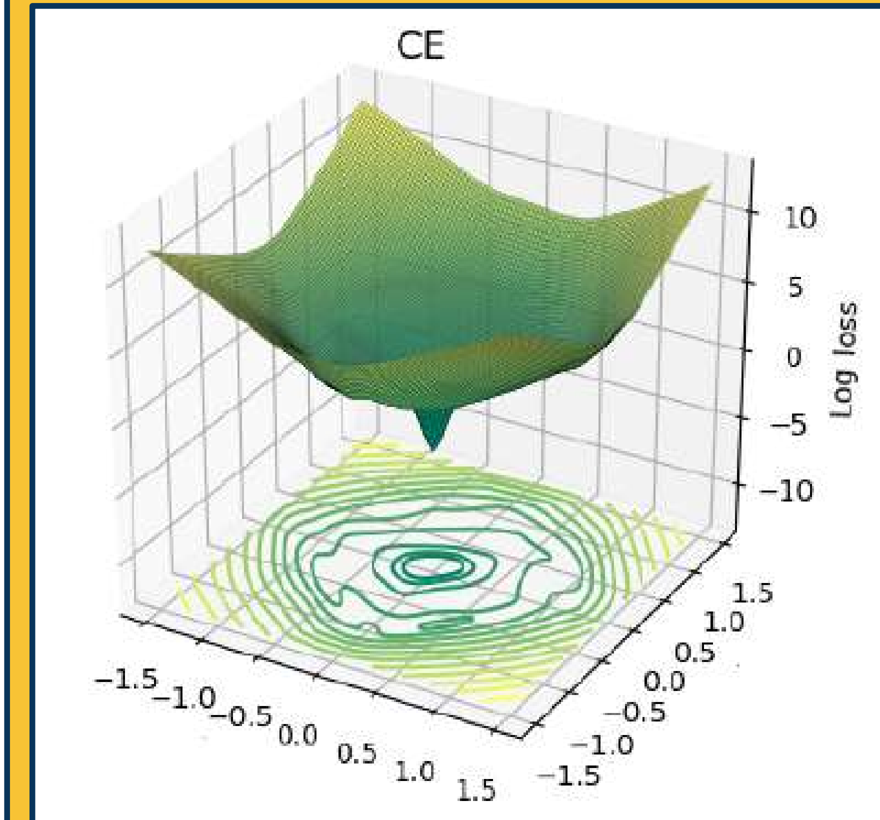
Loss

Optimization

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!

Georgia Tech

Derivatives

- 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

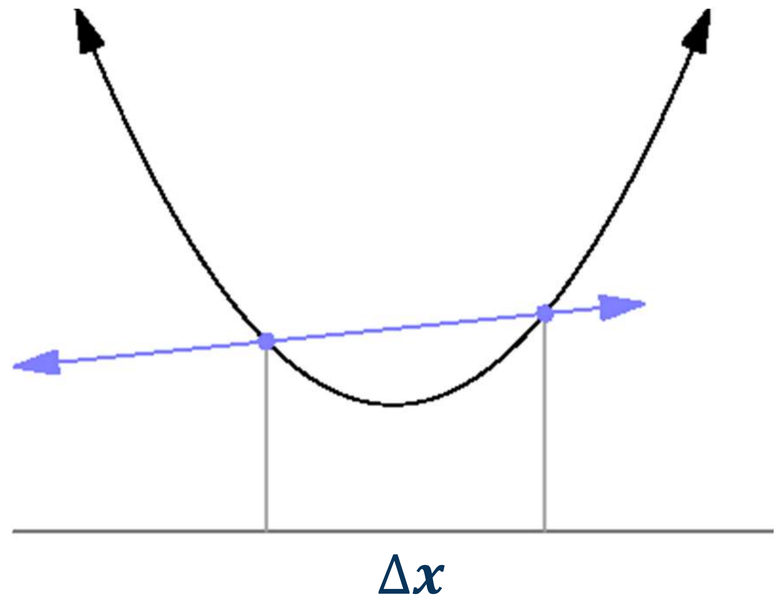
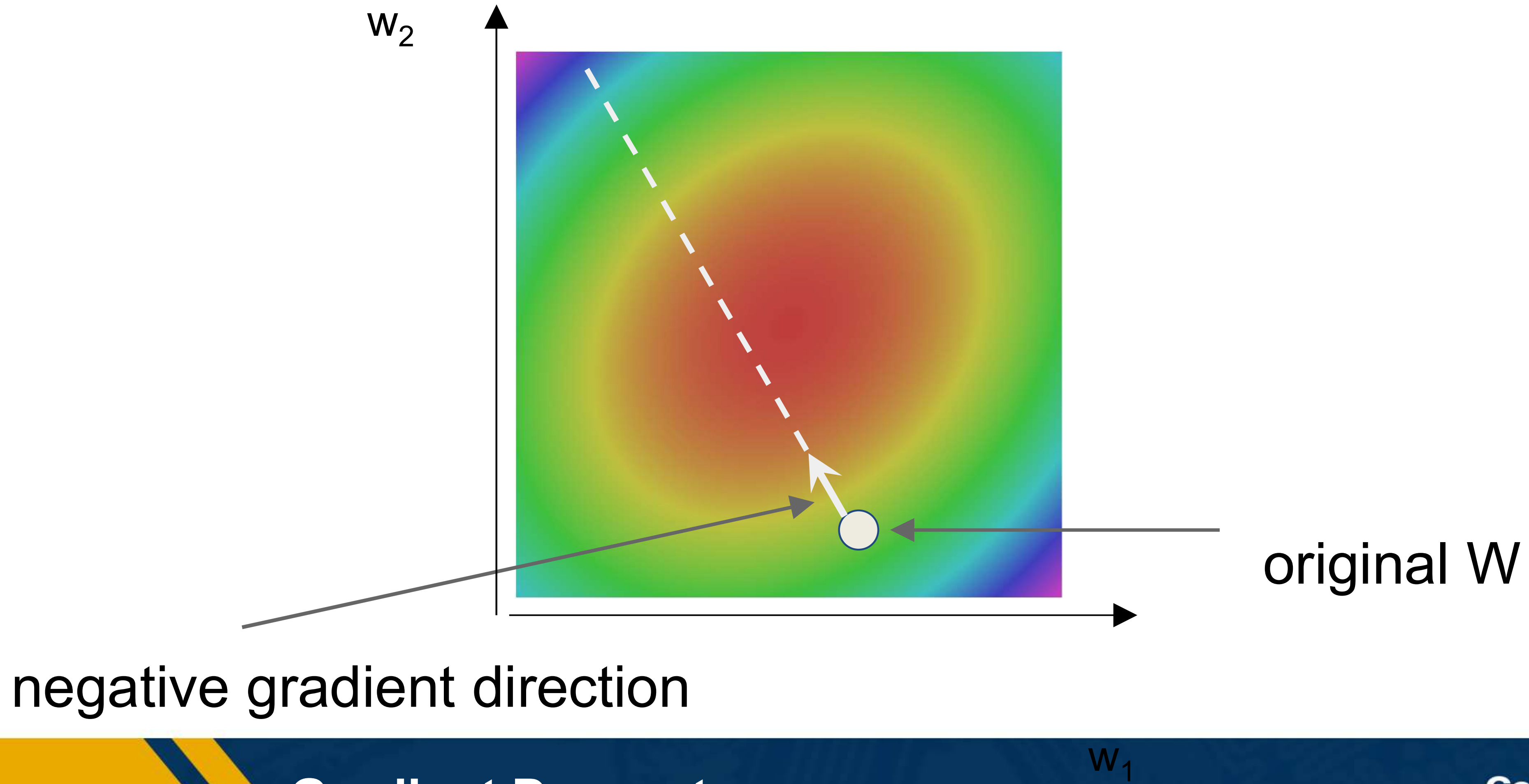


Image and equation from: https://en.wikipedia.org/wiki/Derivative#/media/File:Tangent_animation.gif

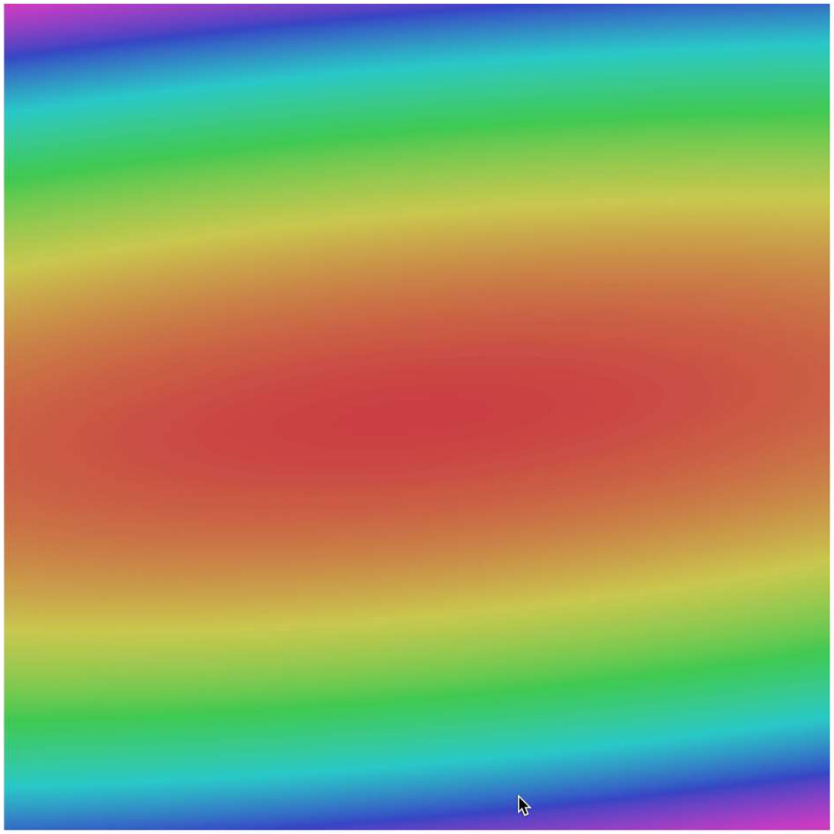
This idea can be turned into an **algorithm (gradient descent)**

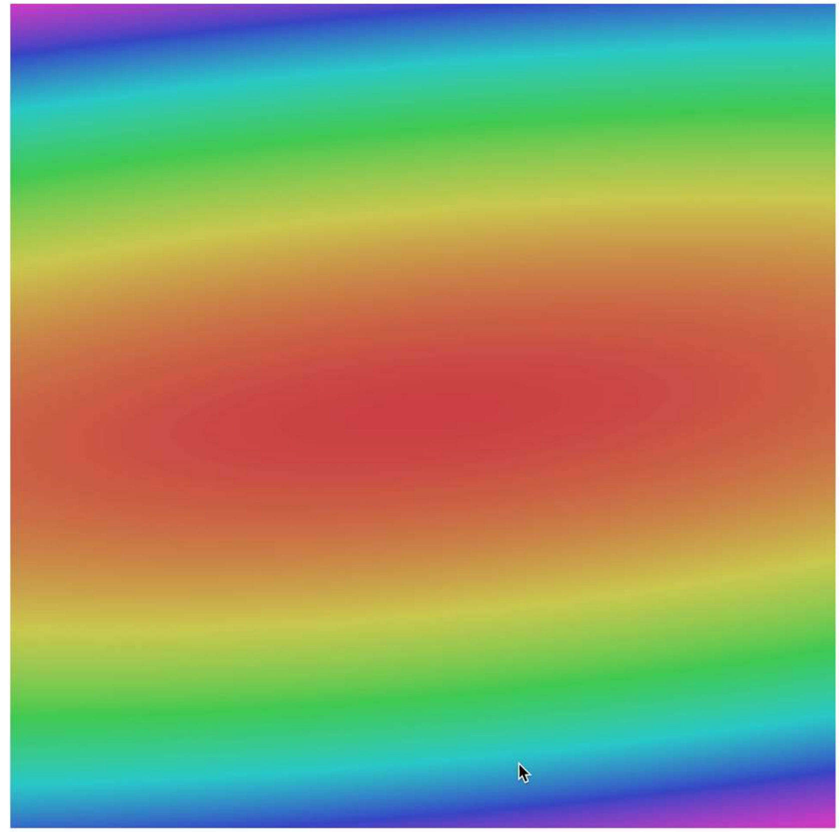
- Choose a model: $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{W}) = \mathrm{Wx}$
- Choose loss function: $\boldsymbol{L_i} = |\boldsymbol{y} - \boldsymbol{W}\boldsymbol{x_i}|^{2}$
- Calculate partial derivative for each parameter: $\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{w_i}}$
- Update the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

http://demonstrations.wolfram.com/VisualizingTheGradientVector/



Gradient Descent





Gradient Descent

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

- Full Batch Gradient Descent $$L = \frac{1}{N} \sum L\left(f(x_i, W), y_i\right)$$
- Mini-Batch Gradient Descent $$L = \frac{1}{M} \sum L\left(f(x_i, W), y_i\right)$$
  - Where M is a *subset* of data
- We iterate over mini-batches:
  - Get mini-batch, compute loss, compute derivatives, and take a set

Mini-Batch Gradient Descent

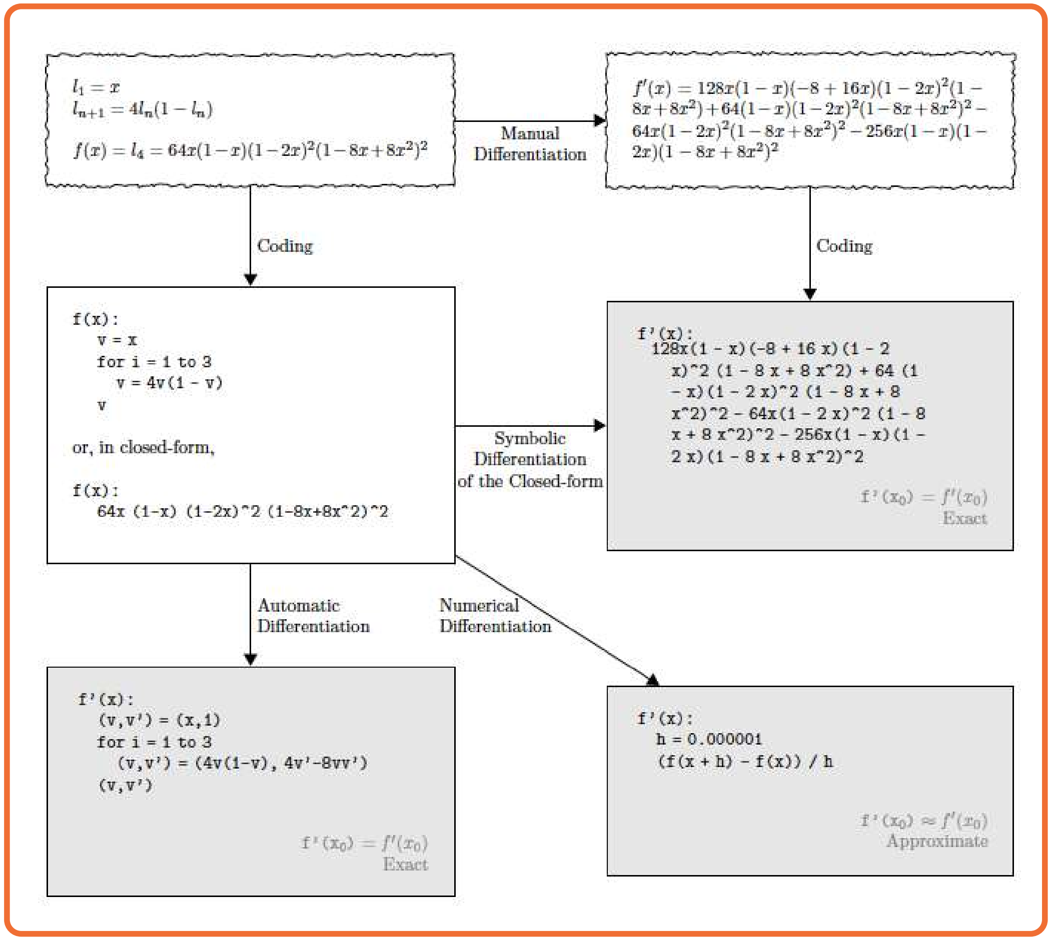
Gradient descent is guaranteed to converge under some conditions

- 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:	**W + h** (first dim)**:**	**gradient dW:**
[0.34,	[0.34 + **0.0001**,	[?,
-1.11,	-1.11,	?,
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 1.25322**	

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,	
0.55,	0.55,	
2.81,	2.81,	
-3.1,	-3.1,	
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25322**	

(1.25322 - 1.25347)/0.0001
= -2.5

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

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

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

?,…]

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**	

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

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

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

Summary

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 $\mathbf{x_i}$ 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