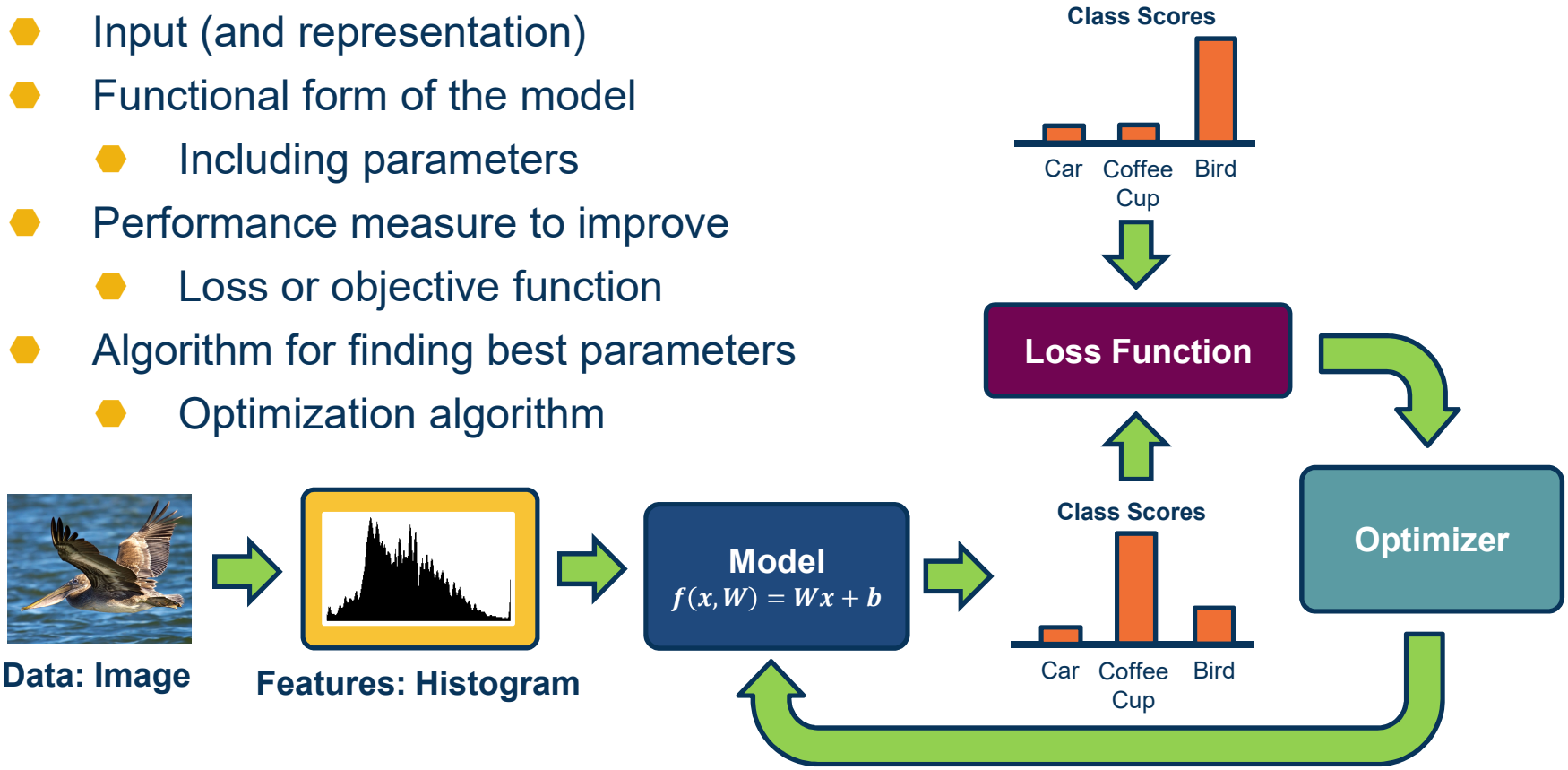
Topics:

- Gradient Descent
- Neural Networks

CS 4644-DL / 7643-A
ZSOLT KIRA

- **Assignment 1 out!**
  - **Due date extended to Feb 5th (7th with grace period)**
  - Start now, start now, start now!
  - Start now, start now, start now!
  - Start now, start now, start now!
- **Piazza**
  - Be active!!!
- **Office hours**
  - Lots of special topics (e.g. PS0, Assignment 1, research paper discussion, etc.)
- Note: Course starting to get math heavy!

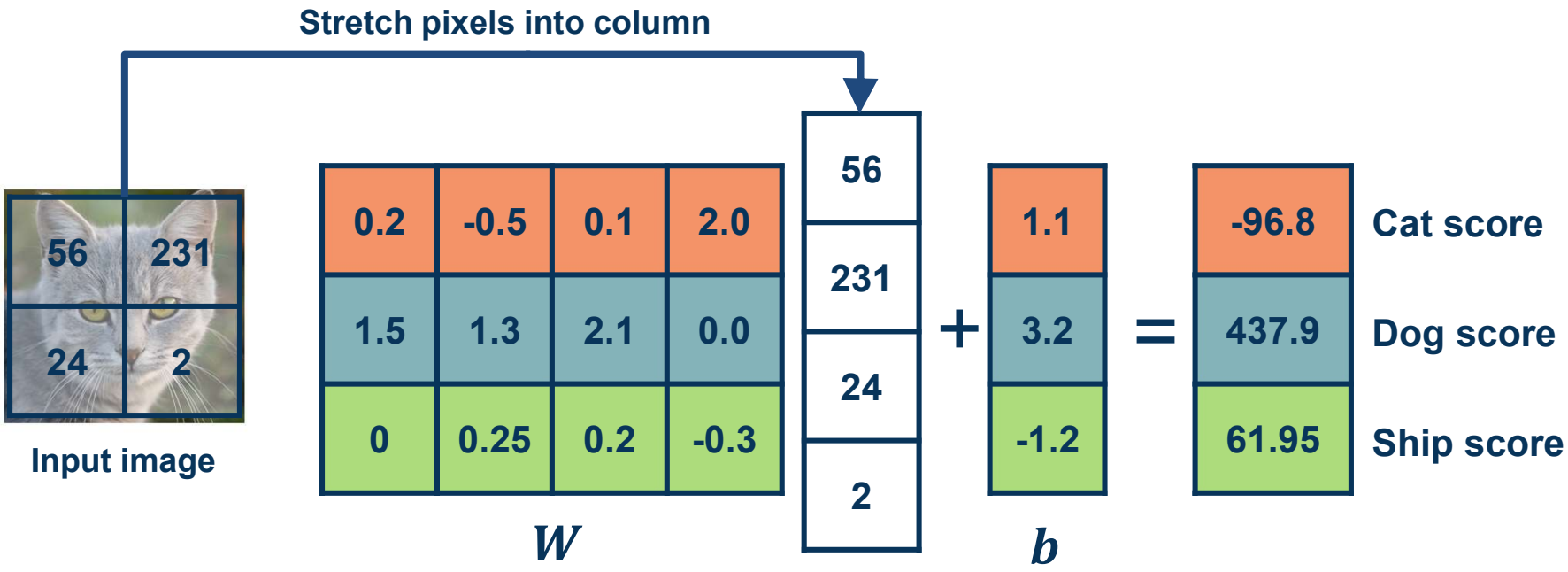
- Input (and representation)
- Functional form of the model
  - Including parameters
- Performance measure to improve
  - Loss or objective function
- Algorithm for finding best parameters
  - Optimization algorithm



Components of a Parametric Model

Georgia Tech

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

Softmax Classifier (Multinomial Logistic Regression)

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

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

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

Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

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

	Unnormalized log-probabilities / logits	exp	Unnormalized probabilities	normalize	Probabilities	
cat	**3.2**		**24.5**		**0.13**	→ $L_i = -\log(0.13)$
car	5.1		164.0		0.87	
frog	-1.7		0.18		0.00	

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

Cross-Entropy Loss Example

Georgia Tech

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

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**

Data: Image

Features: Histogram

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

Class Scores

Loss Function

Class Scores

Optimizer

Components of a Parametric Model

Georgia Tech

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

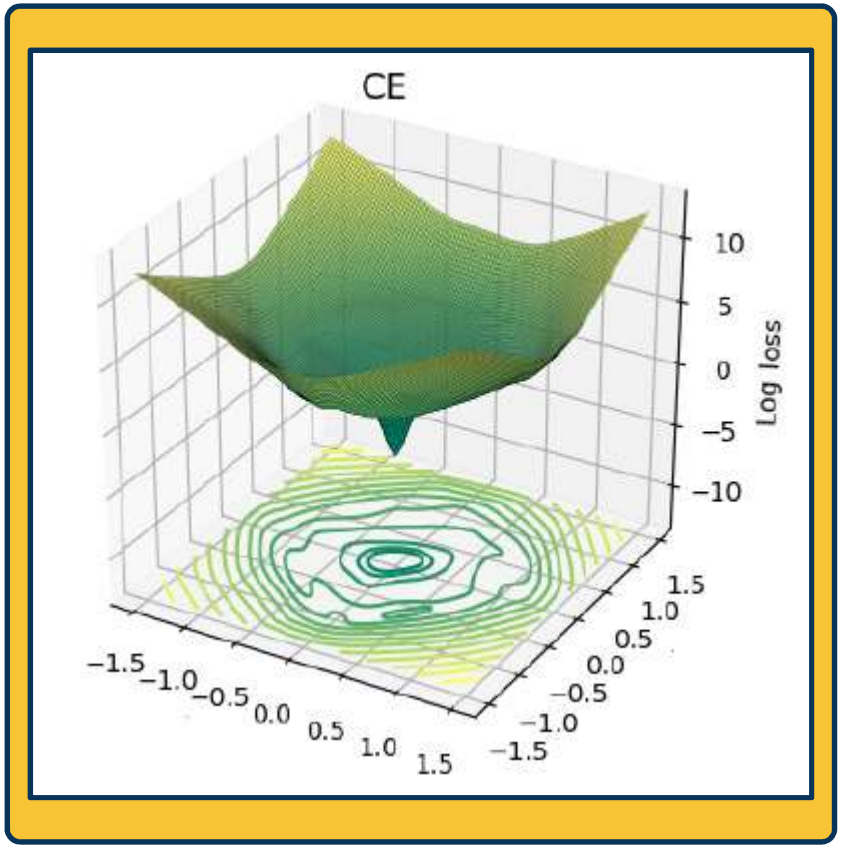
$$\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} & b1 \\ w_{21} & w_{22} & \cdots & w_{2m} & b2 \\ w_{21} & w_{22} & \cdots & w_{3m} & b3 \end{bmatrix}$$

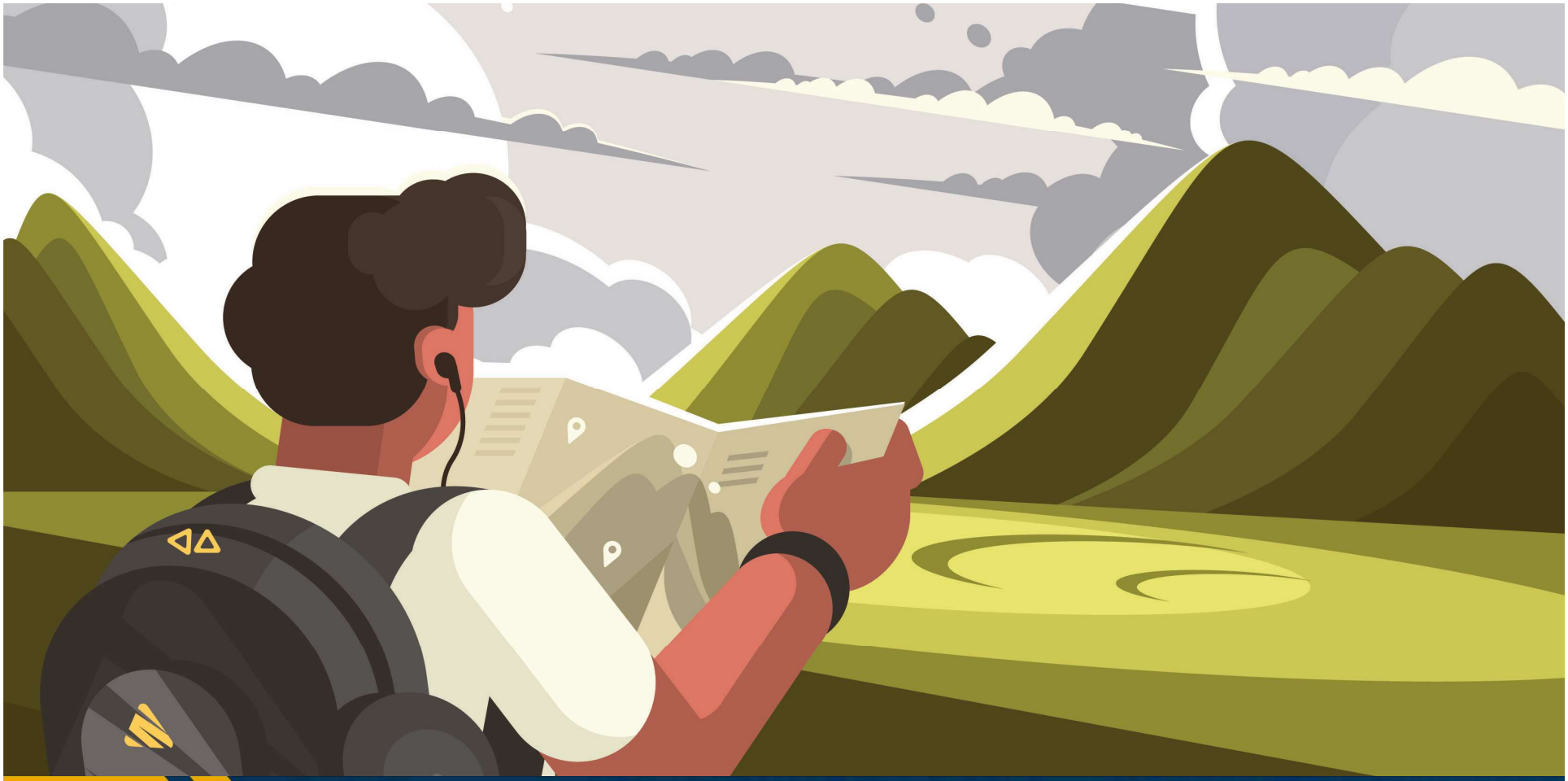
Loss

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

- 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

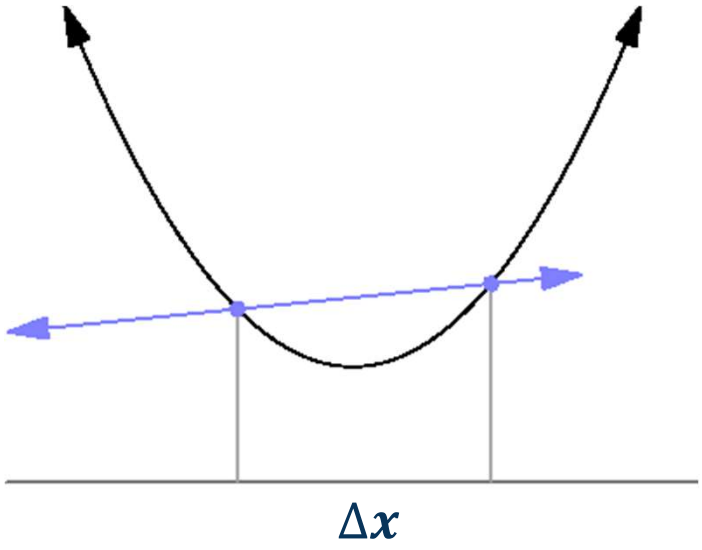
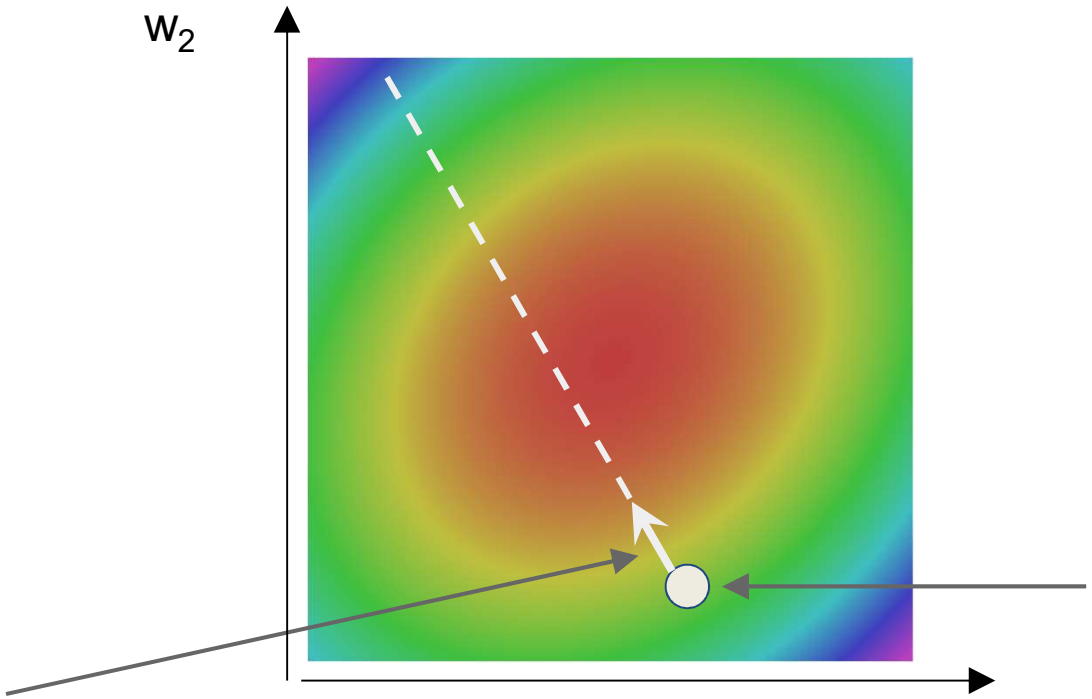


Image and equation from: https://en.wikipedia.org/wiki/Derivative#/media/File:Tangent_animation.gif

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

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

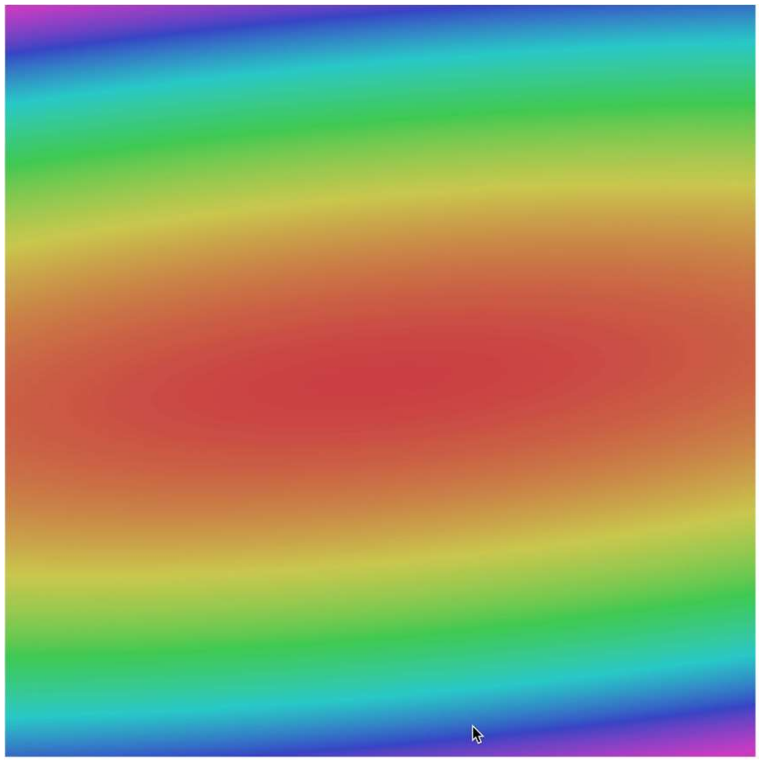
http://demonstrations.wolfram.com/VisualizingTheGradientVector/

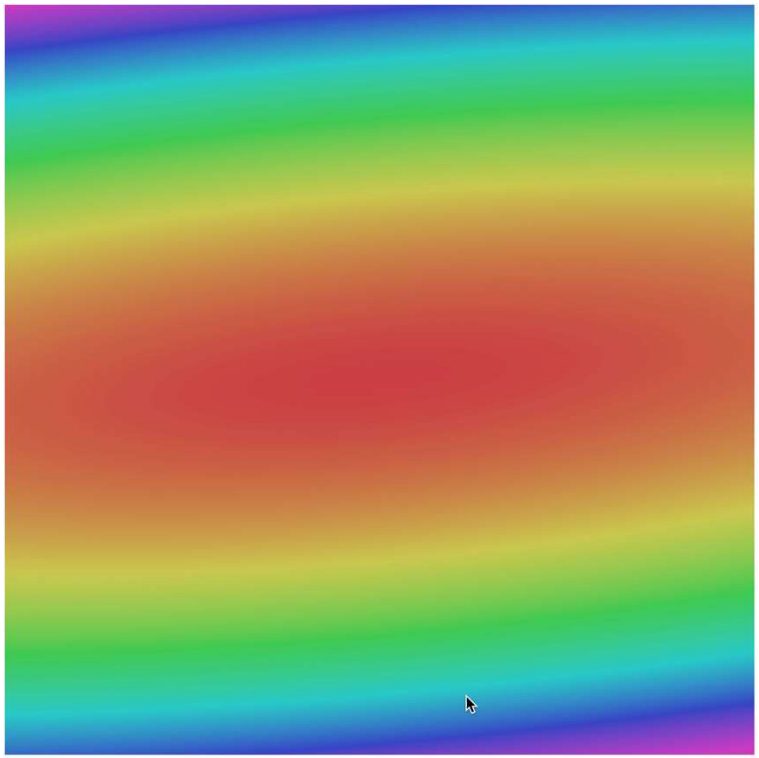


original W

negative gradient direction

Gradient Descent





Gradient Descent

w_1

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

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

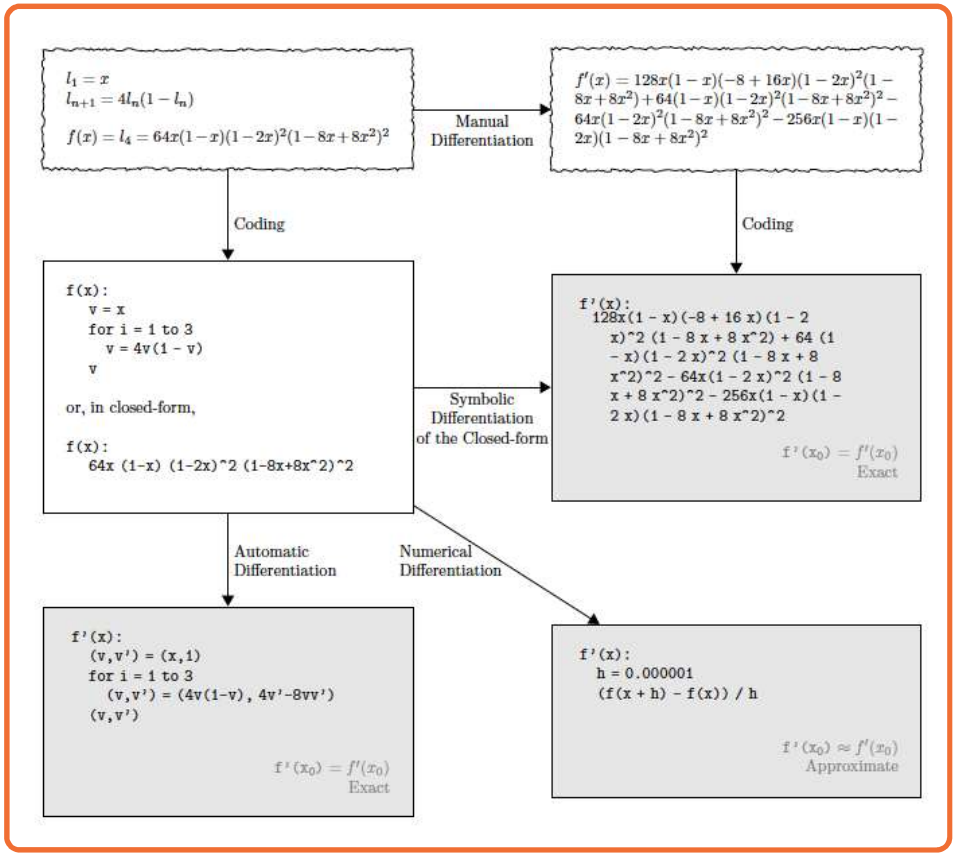
Gradient descent is guaranteed to converge under some conditions

- 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



Georgia Tech

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, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,…]	[0.34 + **0.0001**, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,…]	[?, ?, ?, ?, ?, ?, ?, ?, ?,…]
loss 1.25347	**loss 1.25322**	

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,…]

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

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

?,
?,…]

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

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,…]

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

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

?,…]

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,…]

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

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

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

Numerical vs Analytic Gradients

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

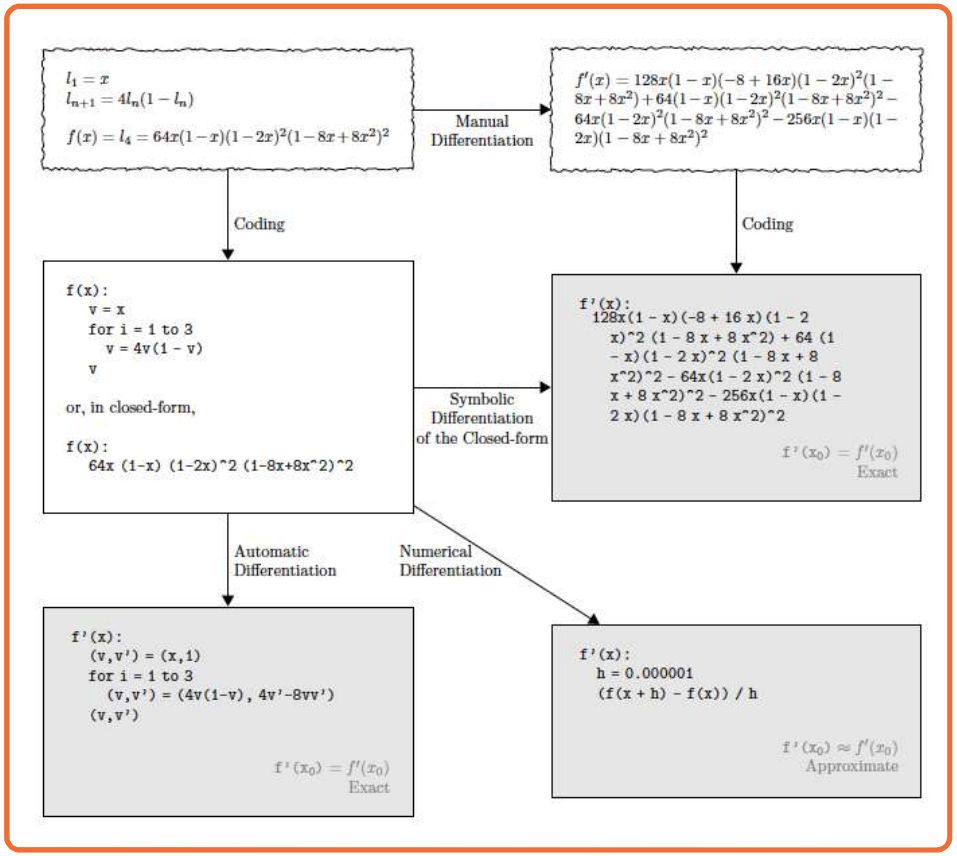
Numerical gradient: slow :(, approximate :(, easy to write :)
Analytic gradient: fast :), exact :), error-prone :(

In practice: Derive analytic gradient, check your implementation with numerical gradient.
This is called a **gradient check.**

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



Georgia Tech

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

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

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 $\mathbf{x_i}$ are column vectors, so same as $w \cdot x_i$)

Dataset: N examples (indexed by k)

Update Rule

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

Derivation of Update Rule

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

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 $\frac{\partial L}{\partial w_j}$?

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

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

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

…where… $\delta_k = y_k - w^T x_k$

$$= -2 \sum_{k=1}^{N} \delta_k \frac{\partial}{\partial w_j} \sum_{i=1}^{m} w_i x_{ki}$$

$$= -2 \sum_{k=1}^{N} \delta_k x_{kj}$$

If we add a **non-linearity (sigmoid),** derivation is more complex

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

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

$$\mathbf{f(x)} = \sigma\left(\sum_k w_k x_k\right)$$

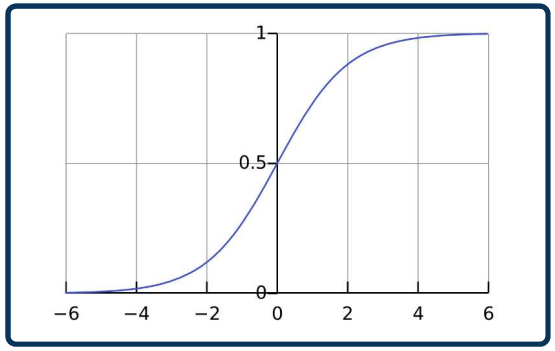
$$\mathrm{L} = \sum_i \left(y_i - \sigma\left(\sum_k w_k x_{ik}\right)\right)^2$$

$$\frac{\partial L}{\partial w_j} = \sum_i 2\left(y_i - \sigma\left(\sum_k w_k x_{ik}\right)\right)\left(-\frac{\partial}{\partial w_j}\sigma\left(\sum_k w_k x_{ik}\right)\right)$$

$$= \sum_i -2\left(y_i - \sigma\left(\sum_k w_k x_{ik}\right)\right)\sigma'\left(\sum_k w_k x_{ik}\right)\frac{\partial}{\partial w_j}\sum_k w_k x_{ik}$$

$$= \sum_i -2\delta_i \sigma(\mathbf{d}_i)(1-\sigma(\mathbf{d}_i))x_{ij}$$

where $\delta_i = y_i - \mathbf{f}(x_i)$ $\quad d_i = \sum w_k x_{ik}$



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 $\quad \sigma_i = \sigma\left(\sum_{j=1}^{m} w_j x_{ij}\right)$

$$\delta_i = y_i - \sigma_i$$

Adding a Non-Linear Function

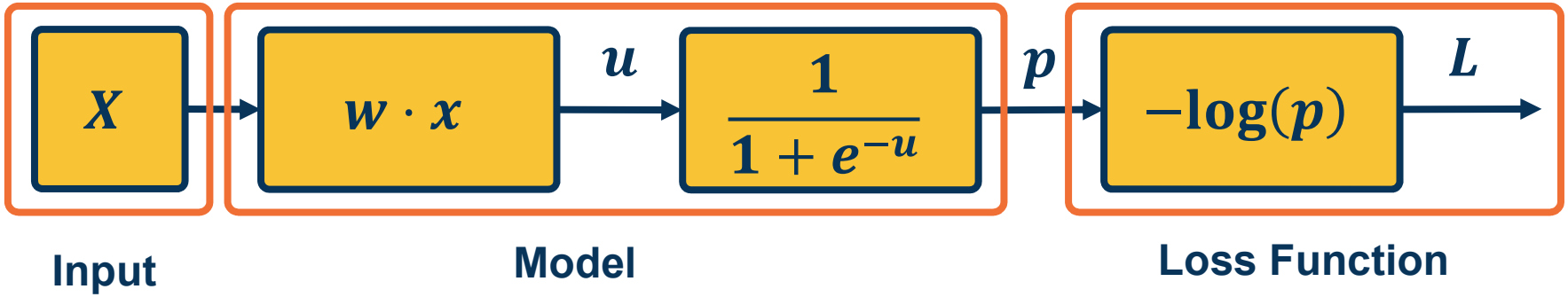
Georgia Tech

Neural Network View of a Linear Classifier

A **linear classifier** can be broken down into:

- Input
- A function of the input
- A loss function

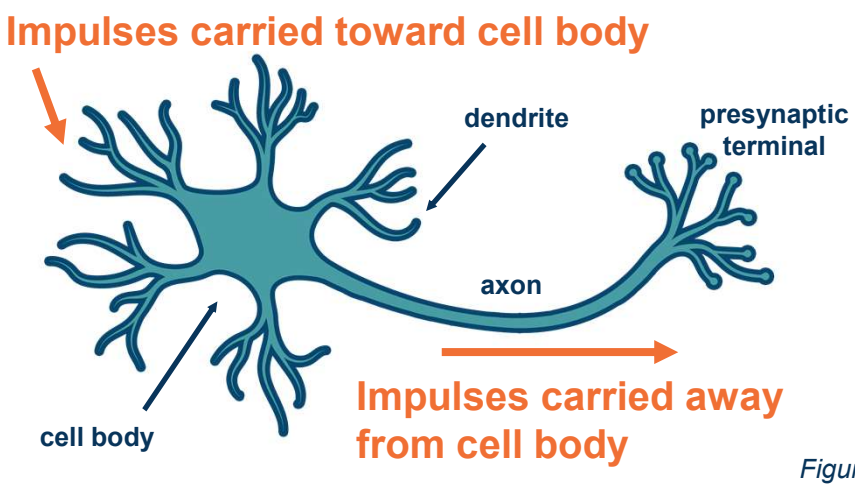
It's all just one function that can be **decomposed** into building blocks

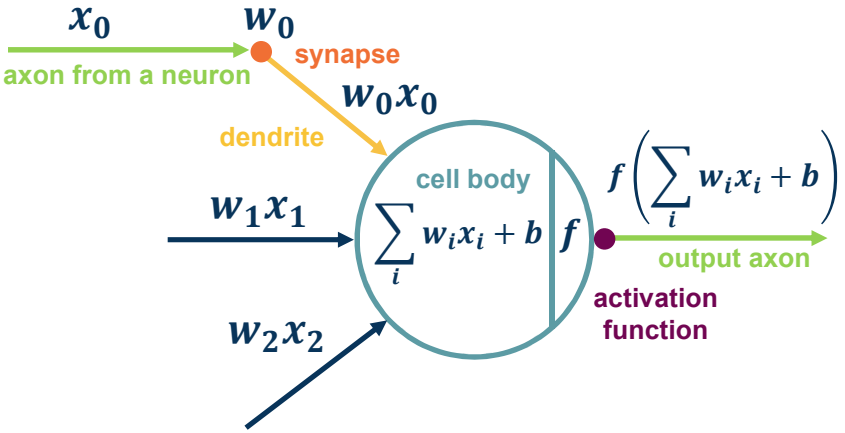


What Does a Linear Classifier Consist of?

A simple **neural network** has similar structure as our linear classifier:

- A neuron takes input (firings) from other neurons **(-> input to linear classifier)**
- The inputs are summed in a weighted manner **(-> weighted sum)**
  - Learning is through a modification of the weights
- If it receives enough input, it “fires” (threshold or if weighted sum plus bias is high enough)



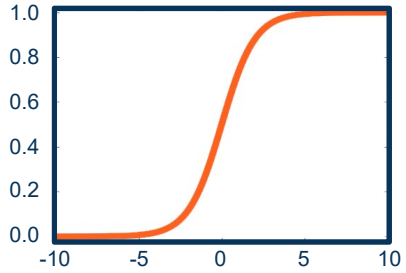


Figures adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Origins of the Term Neural Network

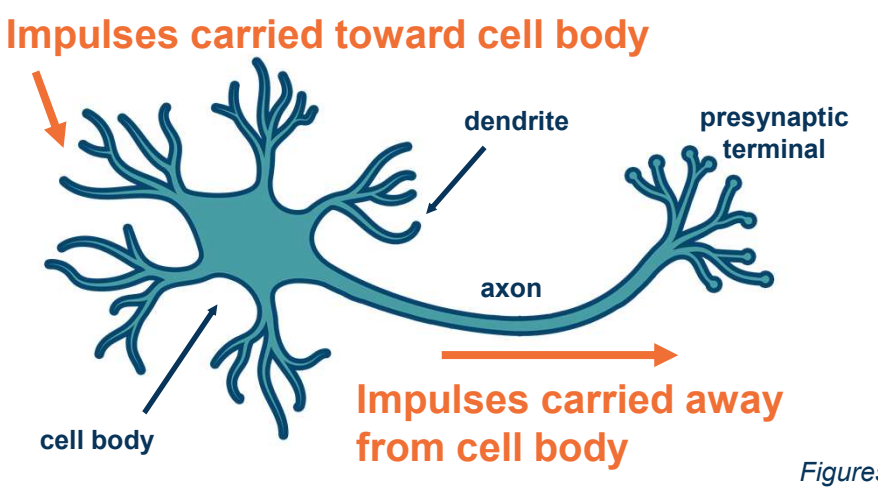
Georgia Tech

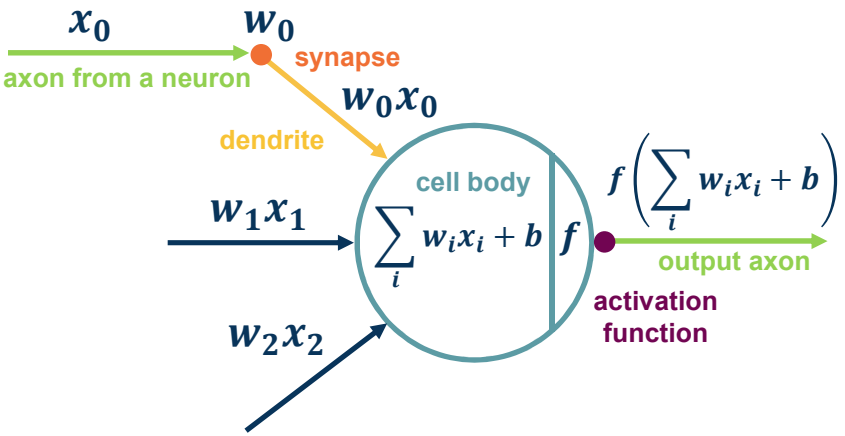
As we did before, the output of a neuron can be modulated by a non-linear function (e.g. sigmoid)



Sigmoid Activation Function

$$\frac{1}{1+e^{-x}}$$





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

Adding Non-Linearities

Georgia Tech

We can have **multiple** neurons connected to the same input

Corresponds to a multi-class classifier

- Each output node outputs the score for a class

$$f(x, W) = \sigma(Wx + b) \quad \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} & b1 \\ w_{21} & w_{22} & \cdots & w_{2m} & b2 \\ w_{21} & w_{22} & \cdots & w_{3m} & b3 \end{bmatrix}$$

- Often called fully connected layers
  - Also called a linear *projection layer*

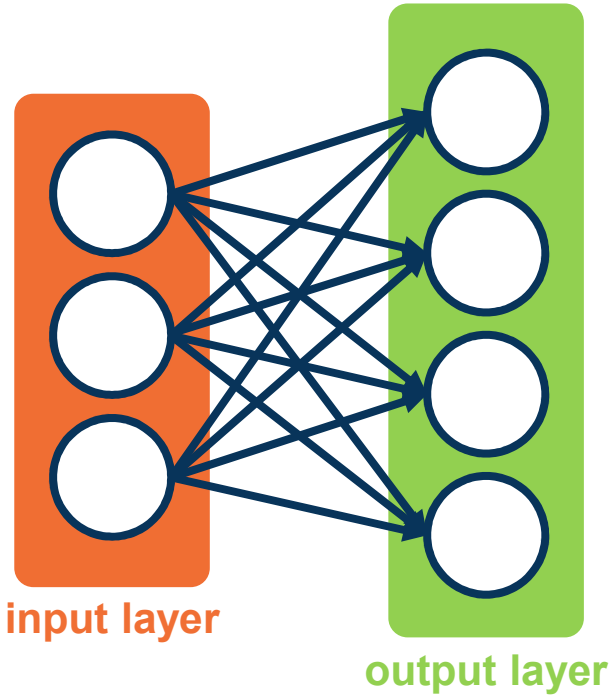


Figure adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

- Each input/output is a **neuron (node)**
- A linear classifier (+ optional non-linearity) is called a **fully connected** layer
- Connections are represented as **edges**
- Output of a particular neuron is referred to as **activation**
- This will be expanded as we view computation in a neural network as a **graph**

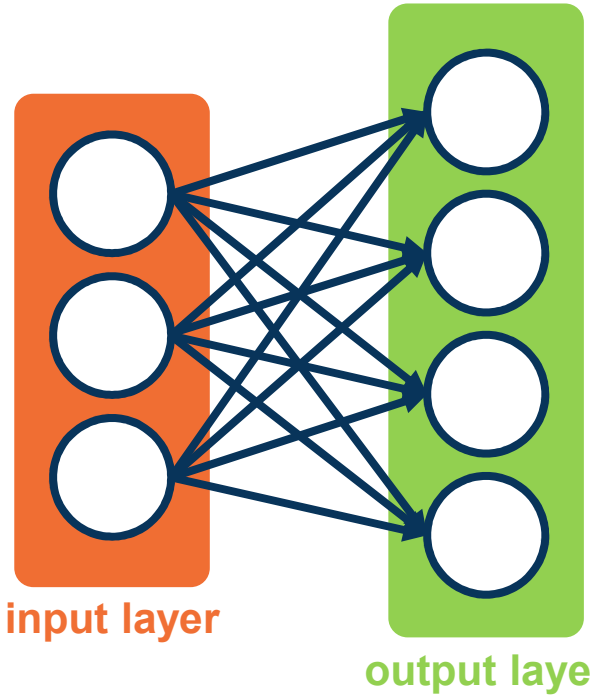


Figure adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Neural Network Terminology

We can **stack** multiple layers together

- Input to second layer is output of first layer

Called a **2-layered neural network** (input is not counted)

Because the middle layer is neither input or output, and we don't know what their values represent, we call them **hidden** layers

- We will see that they end up learning effective features

This **increases** the representational power of the function!

- Two layered networks can represent any continuous function

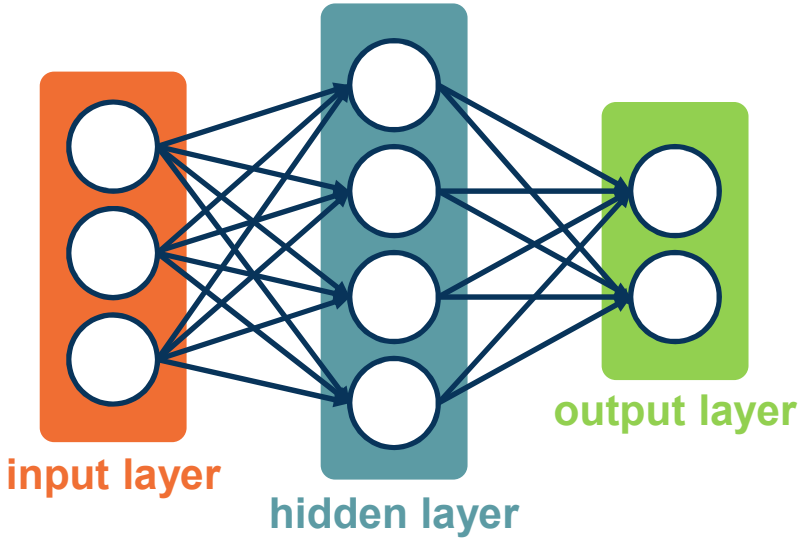
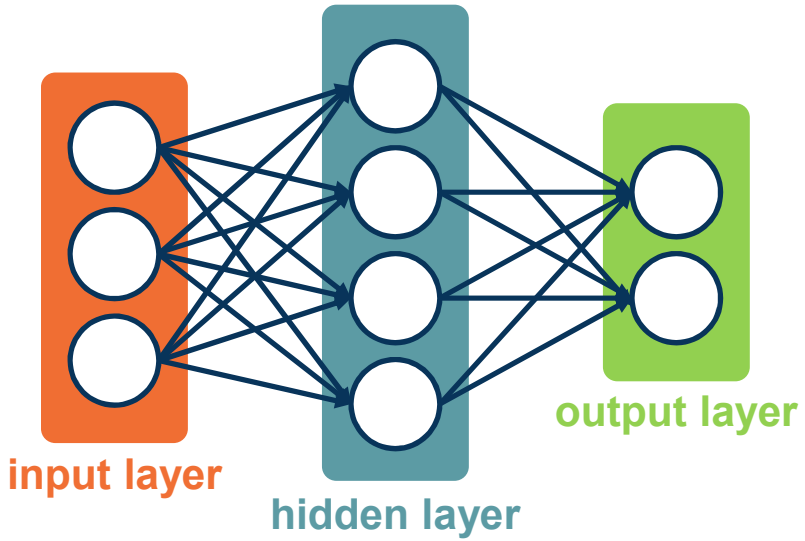


Figure adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Connecting Many Layers

The same two-layered neural network **corresponds to adding another weight matrix**

- We will prefer the linear algebra view, but use some terminology from neural networks (& biology)



$$x \quad W_1 \quad W_2$$

$$=$$

$$f(x, W_1, W_2) = \sigma(W_2\sigma(W_1 x))$$

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

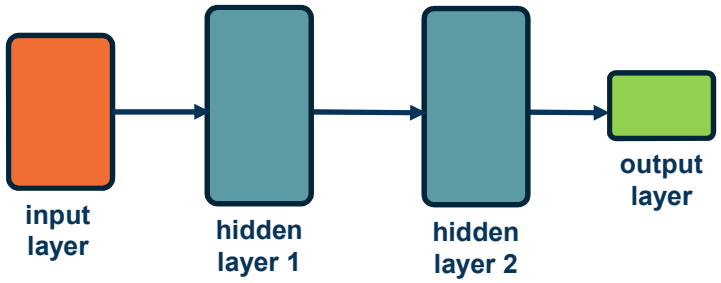
The Linear Algebra View

Large (deep) networks can be built by adding more and more layers

Three-layered neural networks can represent **any function**

- The number of nodes could grow unreasonably (exponential or worse) with respect to the complexity of the function

We will show them **without edges**:



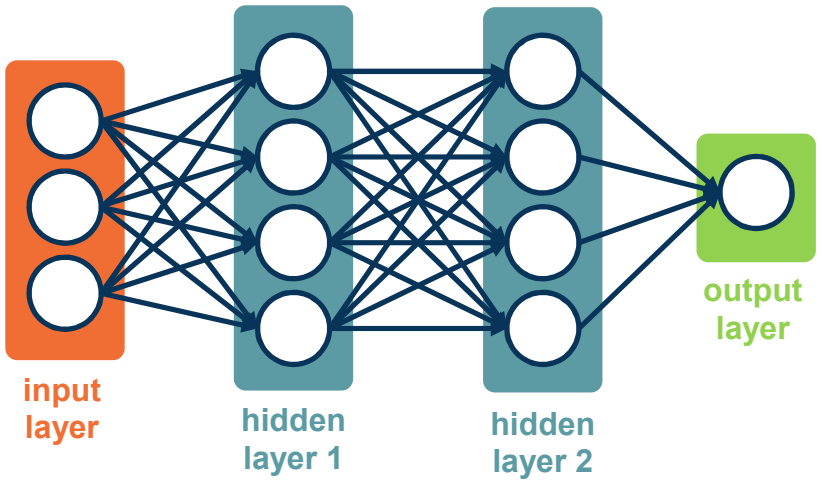


Figure adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Adding More Layers!

Computation Graphs

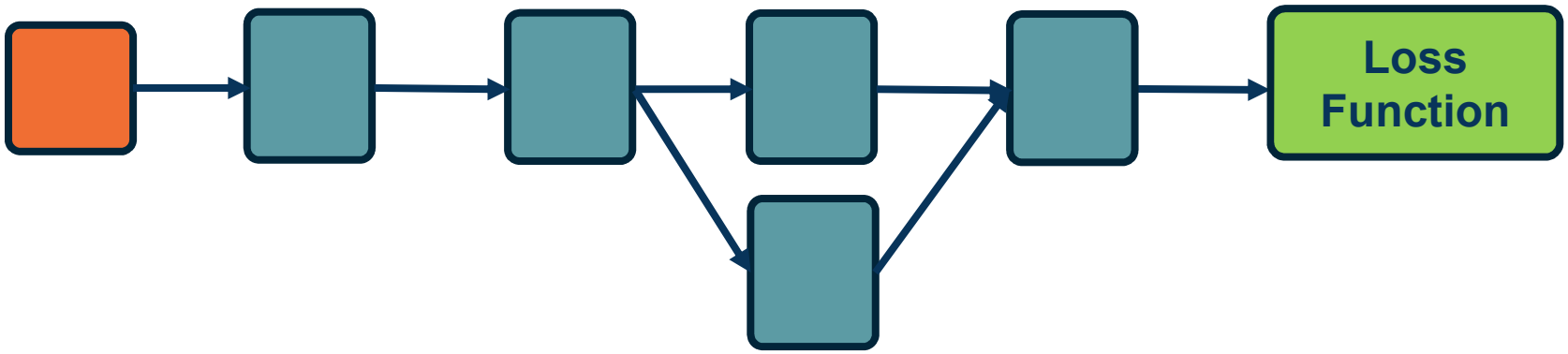
Functions can be made **arbitrarily complex** (subject to memory and computational limits), e.g.:

$$f(x, W) = \sigma(W_5\sigma(W_4\sigma(W_3\sigma(W_2\sigma(W_1 x))))$$

We can use **any type of differentiable function (layer)** we want!

- At the end, **add the loss function**

Composition can have **some structure**



Adding Even More Layers

- 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
  - Analytical

- Key idea: Can we do this across an arbitrary composition of functions (computation graph)?

Summary