Module 1: Introduction to Neural Networks

Georgia Tech

Supervised Learning

- Train Input: $\{X, Y\}$
- Learning output: $f : X \rightarrow Y$, e.g. $P(y|x)$

Unsupervised Learning

- Input: $\{X\}$
- Learning output: $P(x)$
- Example: Clustering, density estimation, etc.

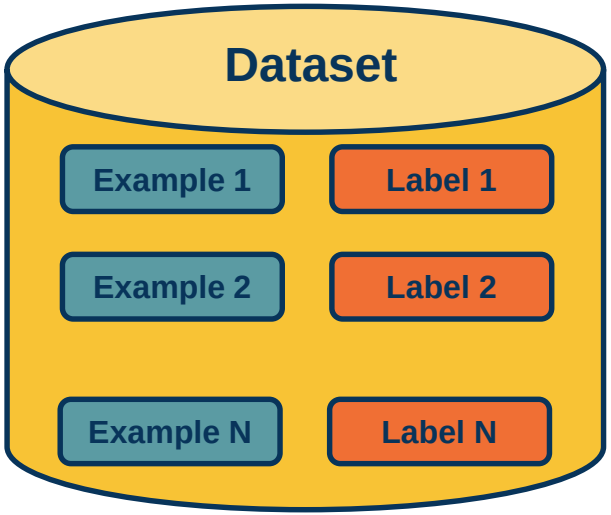
Reinforcement Learning

- Supervision in form of **reward**
- No supervision on what action to take

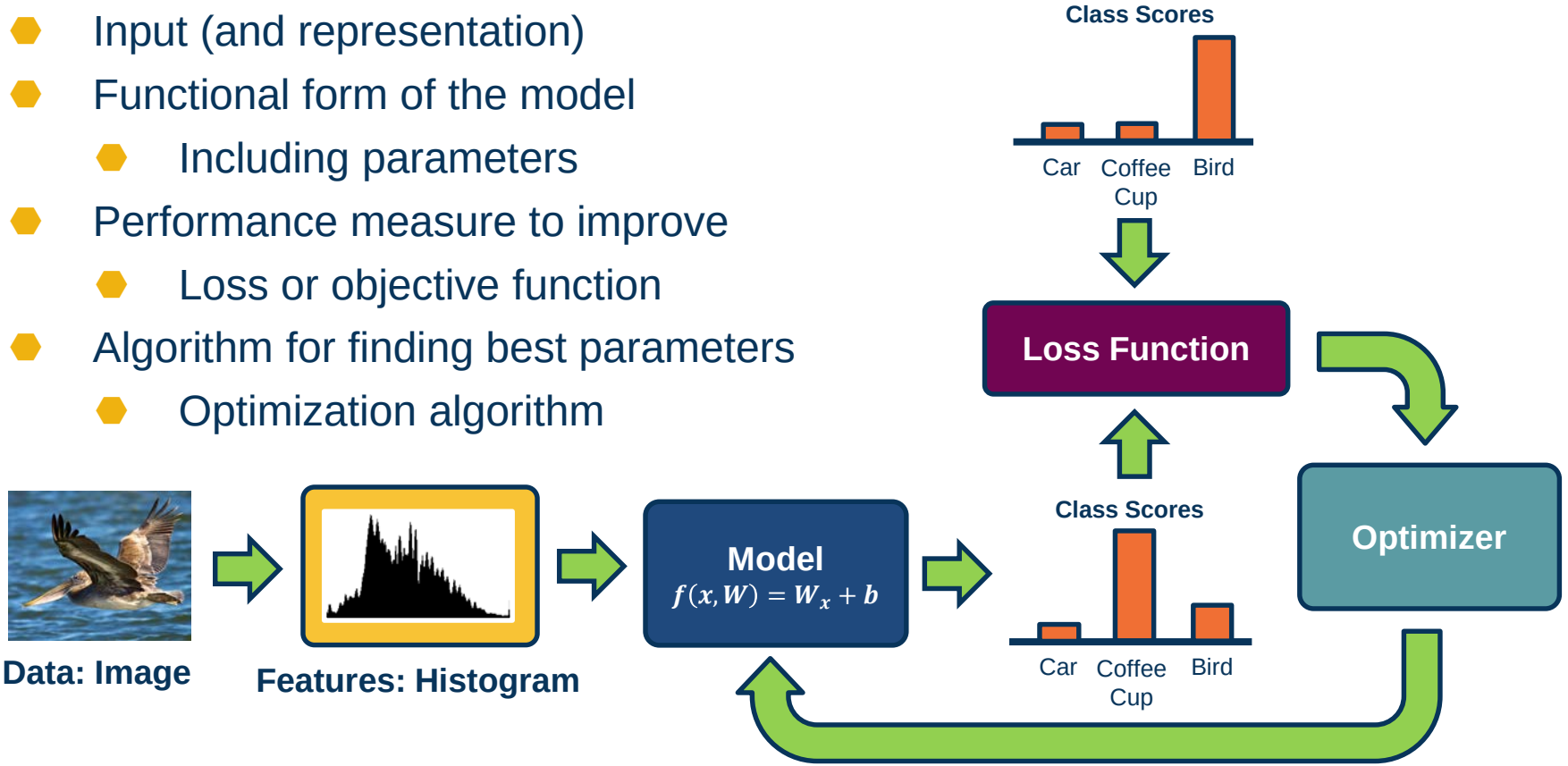
Dataset

$$X = \{x_1, x_2, \ldots, x_N\} \; where \; x \in \mathbb{R}^d$$ **Examples**

$$Y = \{y_1, y_2, \ldots, y_N\} \; where \; y \in \mathbb{R}^c$$ **Labels**

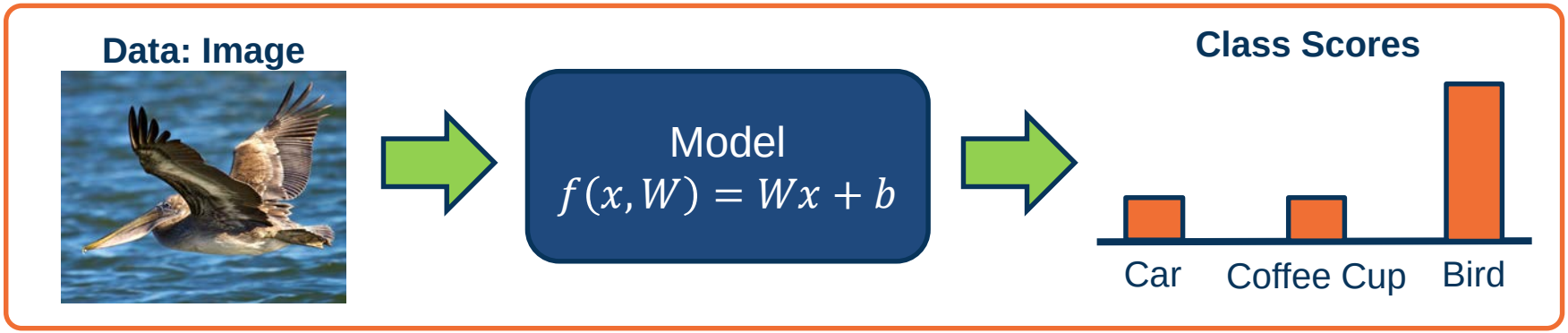


- Input (and representation)
- Functional form of the model
  - Including parameters
- Performance measure to improve
  - Loss or objective function
- Algorithm for finding best parameters
  - Optimization algorithm



Components of a Parametric Model

Georgia Tech

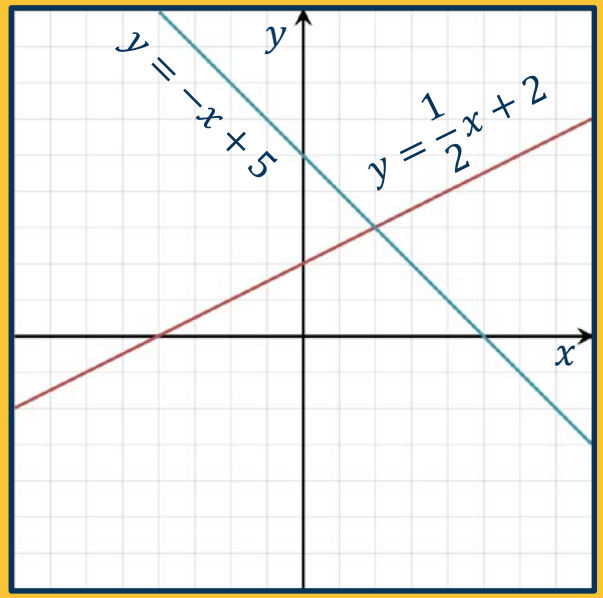


Input $\{X, Y\}$ where:

- X is an image
- Y is a **ground truth label** annotated by an expert (human)
- $f(x, W) = W_x + b$ is our model, chosen to be a linear function in this case
- W and b are the parameters (**weights**) of our model that must be learned

What is the **simplest function** you can think of?

$$y = mx + b$$



Our model is:

$$f(x, w) = wx + b$$

Classifier Result

Weights

Input

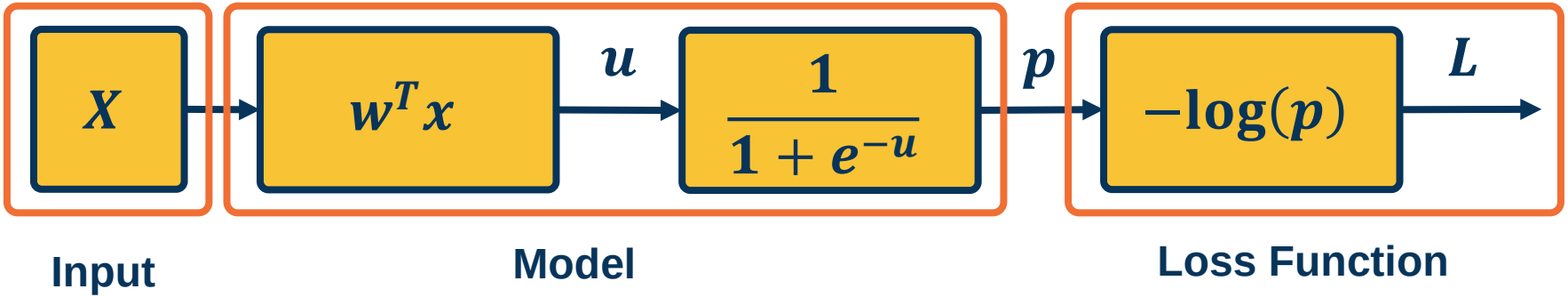
Bias (scalar)

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

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

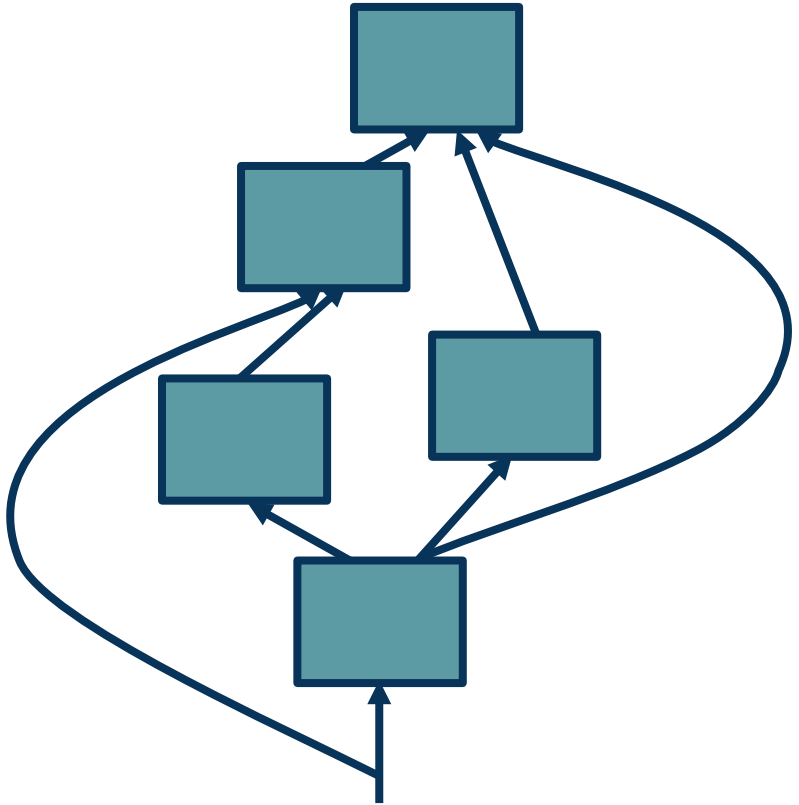


To develop a general algorithm for this, we will view the function as a **computation graph**

Graph can be any **directed acyclic graph (DAG)**

- Modules must be differentiable to support gradient computations for gradient descent

A **training algorithm** will then process this graph, **one module at a time**

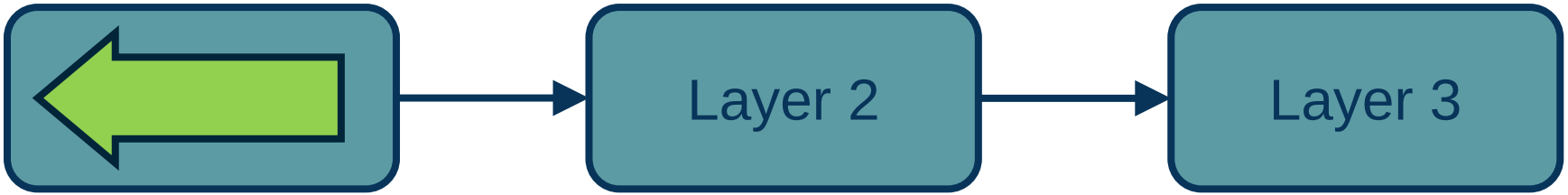


Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

Step 1: Computer Loss on Mini-Batch: **Forward Pass**

Step 2: Compute Gradients wrt parameters: **Backward Pass**

Step 3: Use **gradient** to update **all parameters** at the end



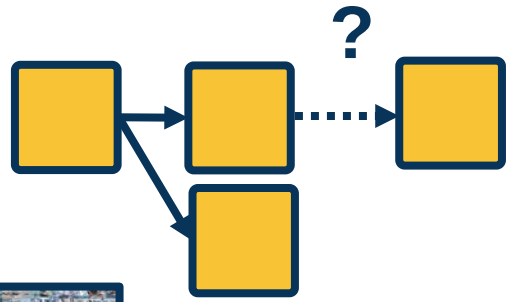
$$w_i = w_i - \alpha \frac{\partial L}{\partial w_i}$$

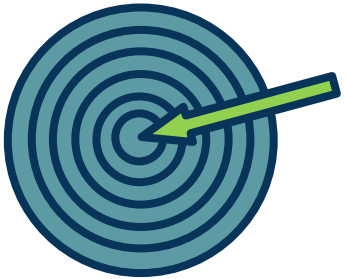
Backpropagation is the application of gradient descent to a computation graph via the chain rule!

Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

There are still many design decisions that must be made:

- **Architecture**
- **Data Considerations**
- **Training and Optimization**
- **Machine Learning Considerations**





Local Minima