INTRO

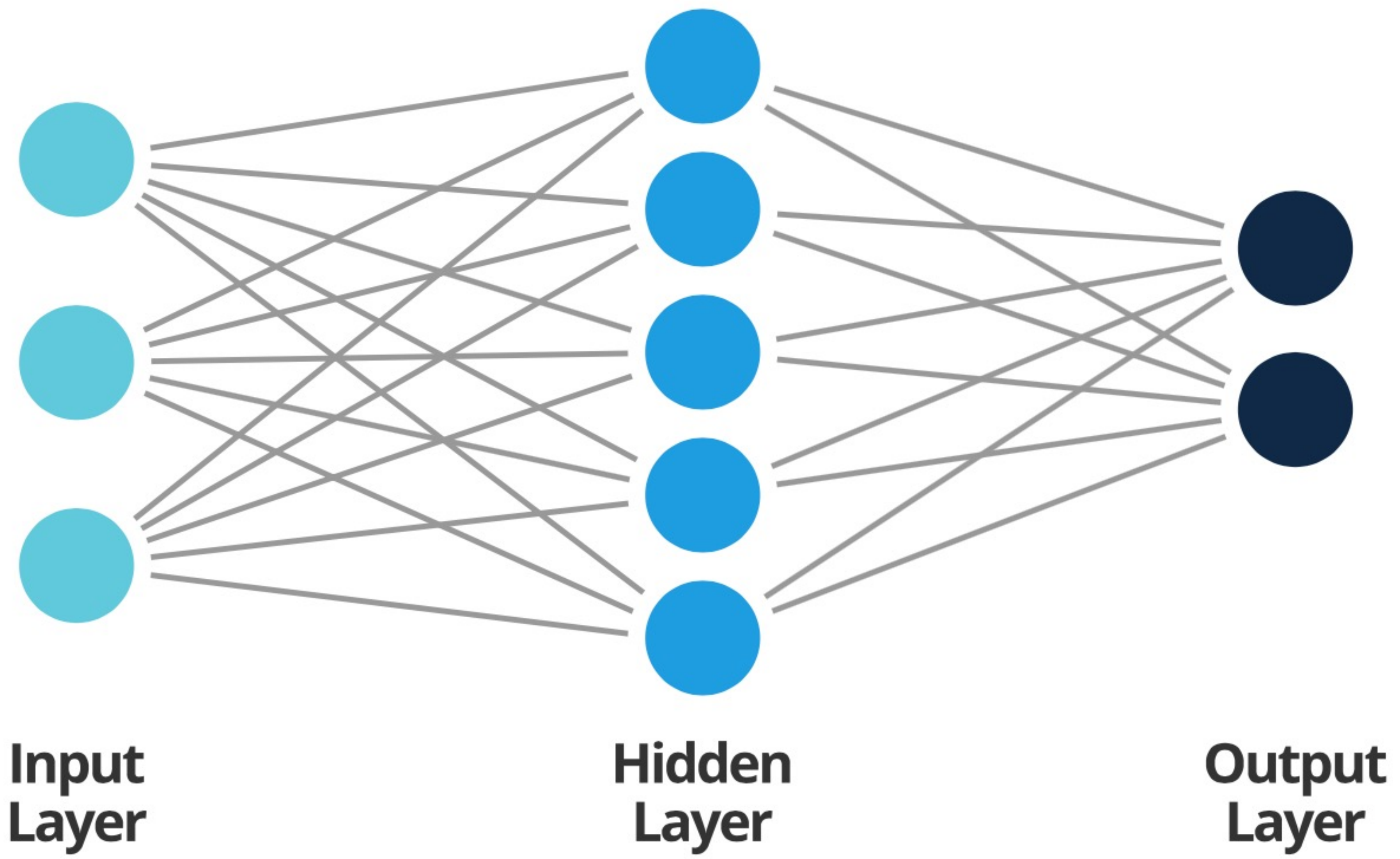
INTRO

- Welcome to BIOF 510! This course will be a continuation of BIOF 509
- My info: anibaljt@nih.gov
- TA info: parkyc@nih.gov

THE COURSE

- We will cover many of the same algorithms as 509 (and some new ones) in more advanced contexts
- Course will consist of two projects, a literature review, and participation (in slack)
- Live lectures on Monday, these will be recorded and posted on canvas
- Christina will post supplementary resources midweek

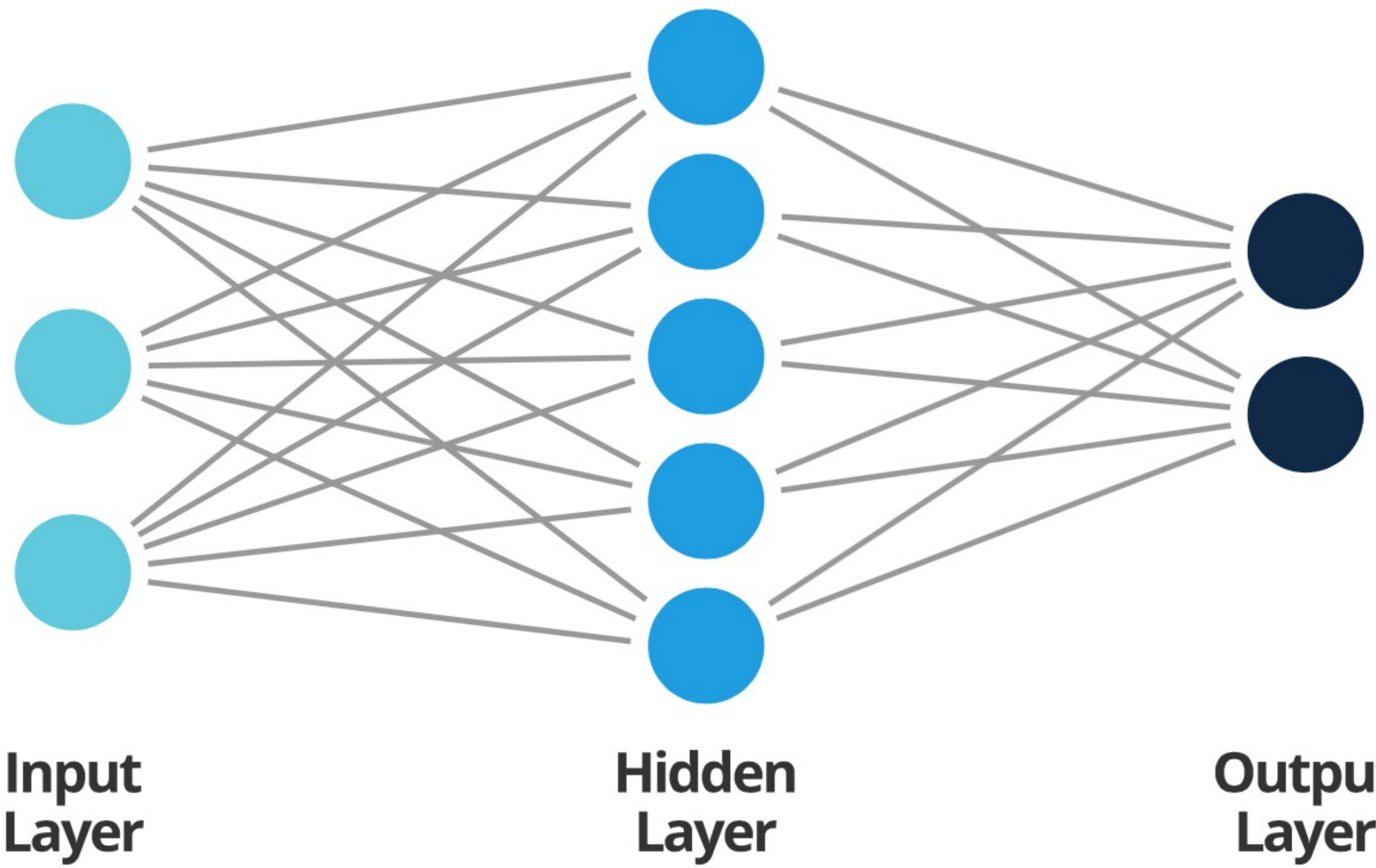
REVIEW OF DEEP LEARNING


Input Layer
Hidden Layer
Output Layer

INPUT LAYER?

What is the input layer?

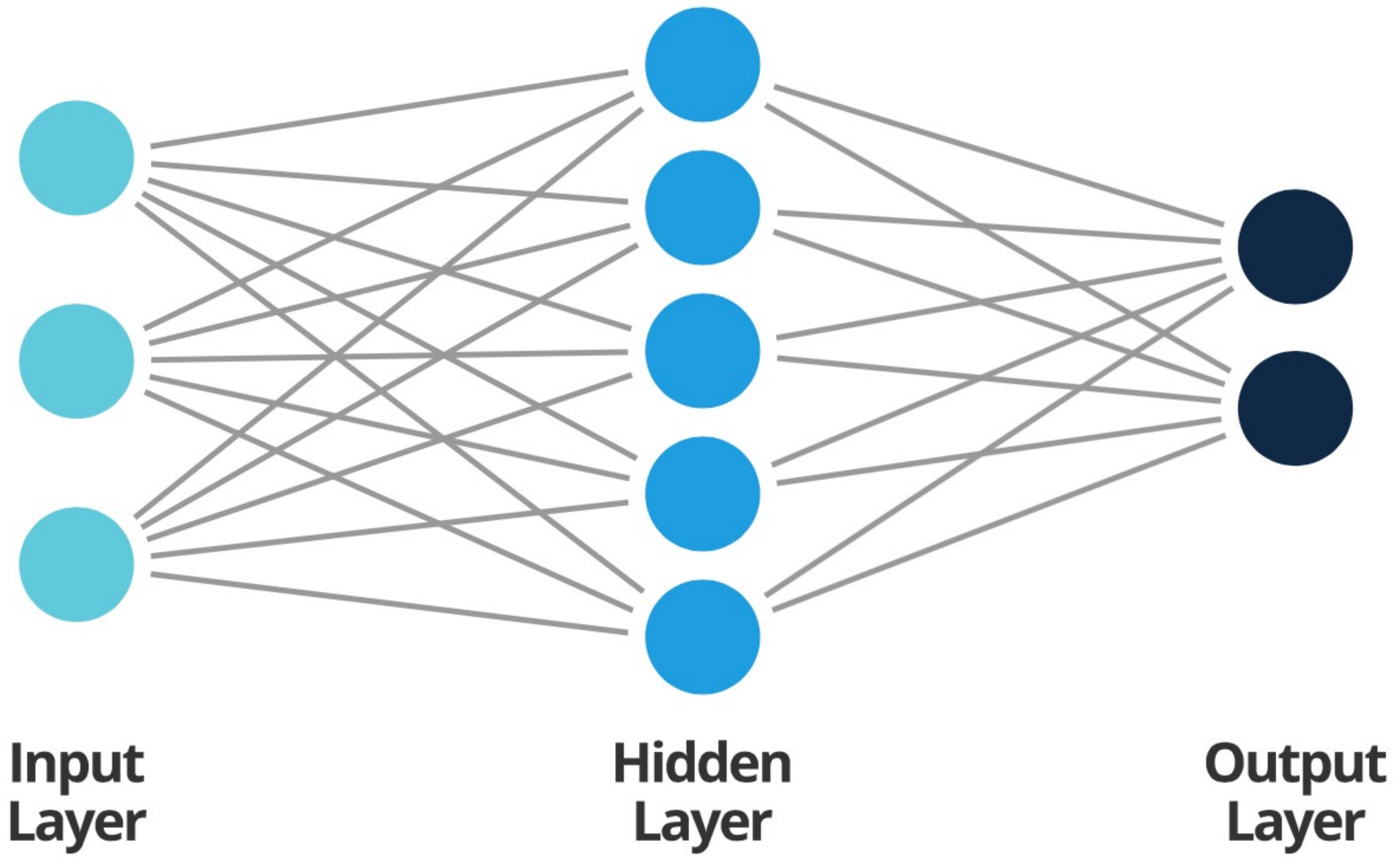
Whatever you are wanting to analyze!

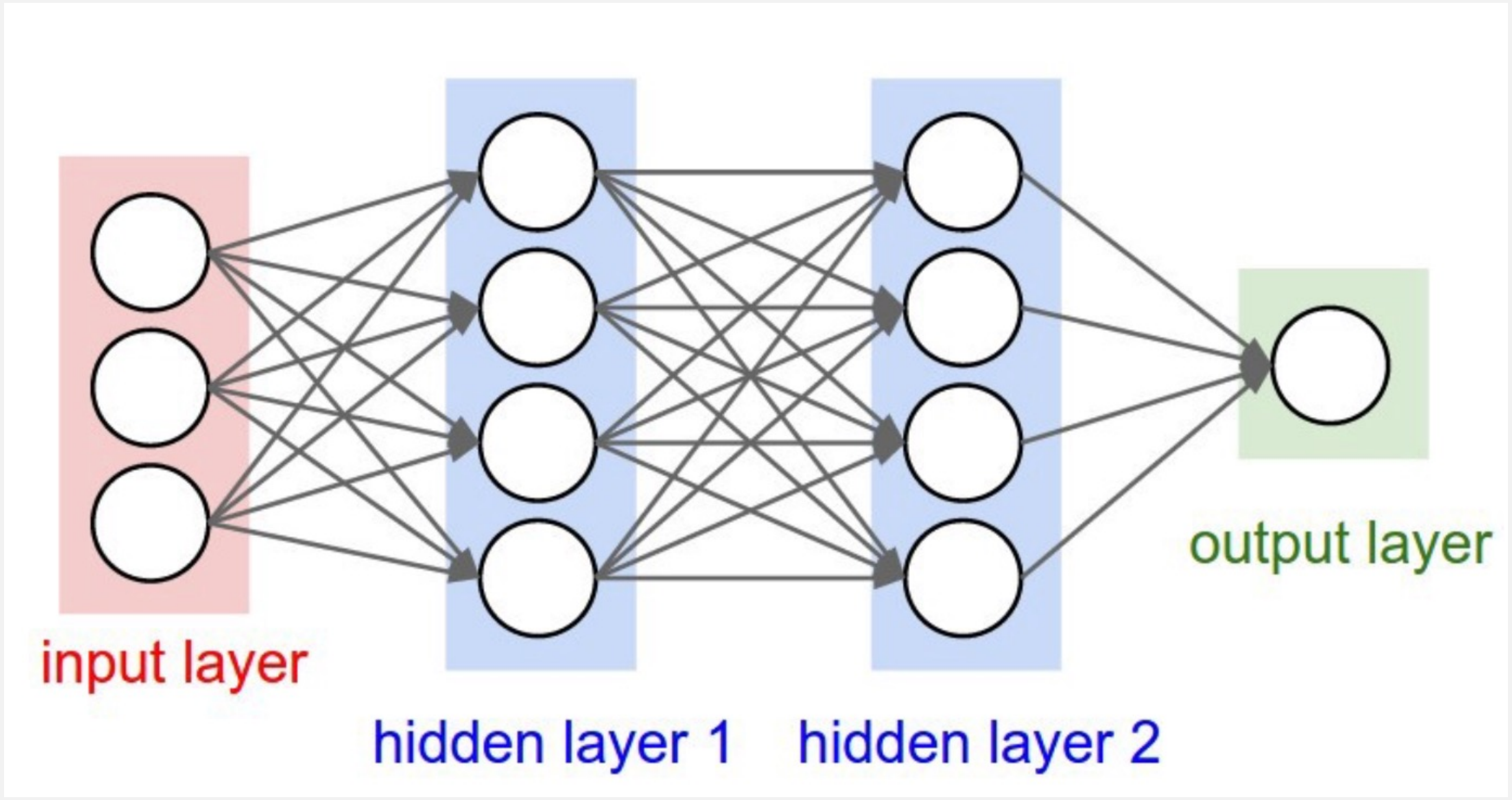

Input Layer
Hidden Layer
Output Layer

HIDDEN LAYERS

The hidden layers are the weighted combinations of features

Weights are initialized randomly and optimized


Input Layer
Hidden Layer
Output Layer

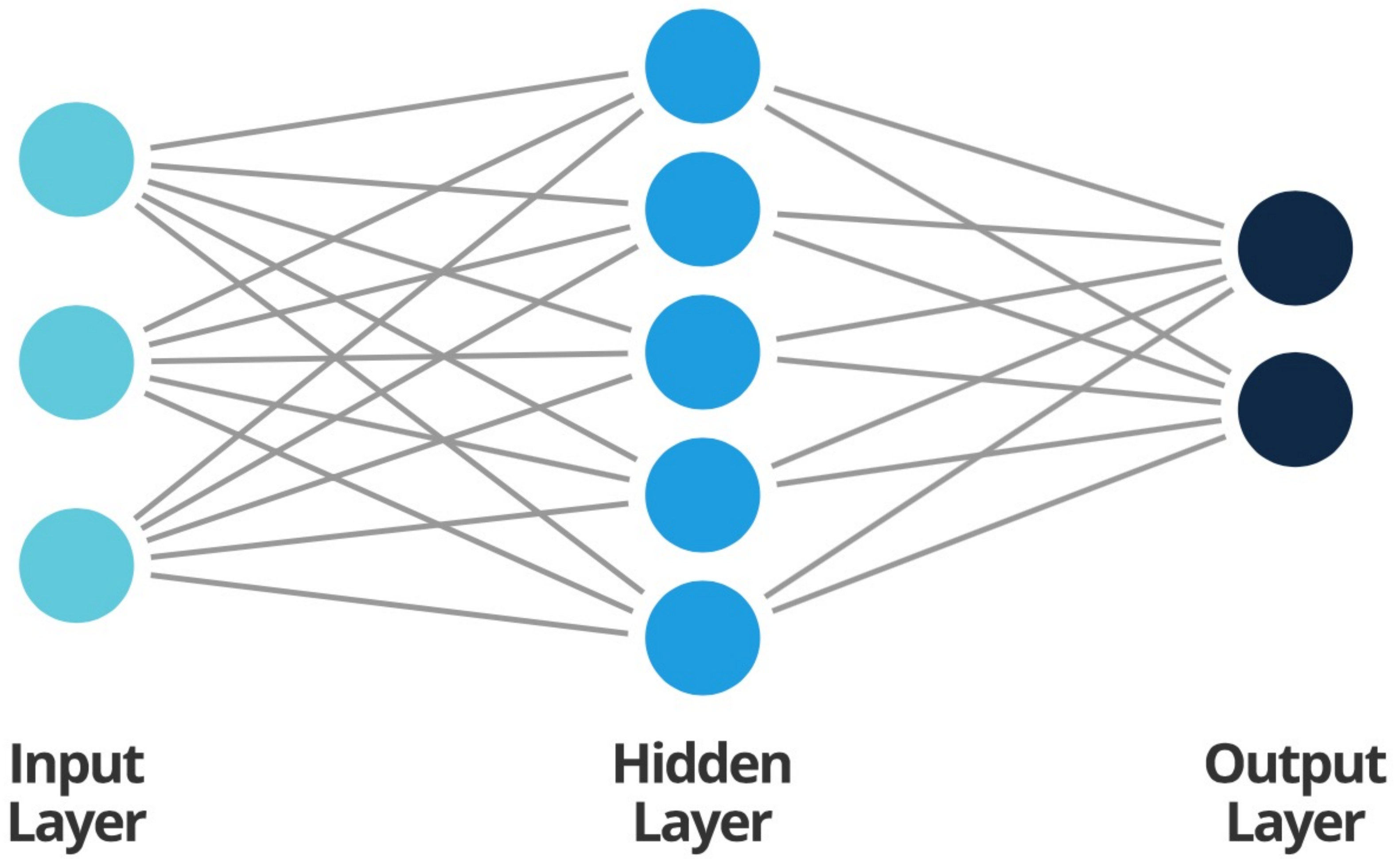

input layer
hidden layer 1
hidden layer 2
output layer

OUTPUT LAYER

In most cases, there one node per label you can predict

Features strongly correlated to a label will have high weights for that corresponding neuron

Label corresponding to the neuron with the highest value is assigned to the input data point


Input Layer
Hidden Layer
Output Layer

NEURAL NETWORK COMPONENTS

Now, we will cover what make neural networks so powerful

I. Activation Function

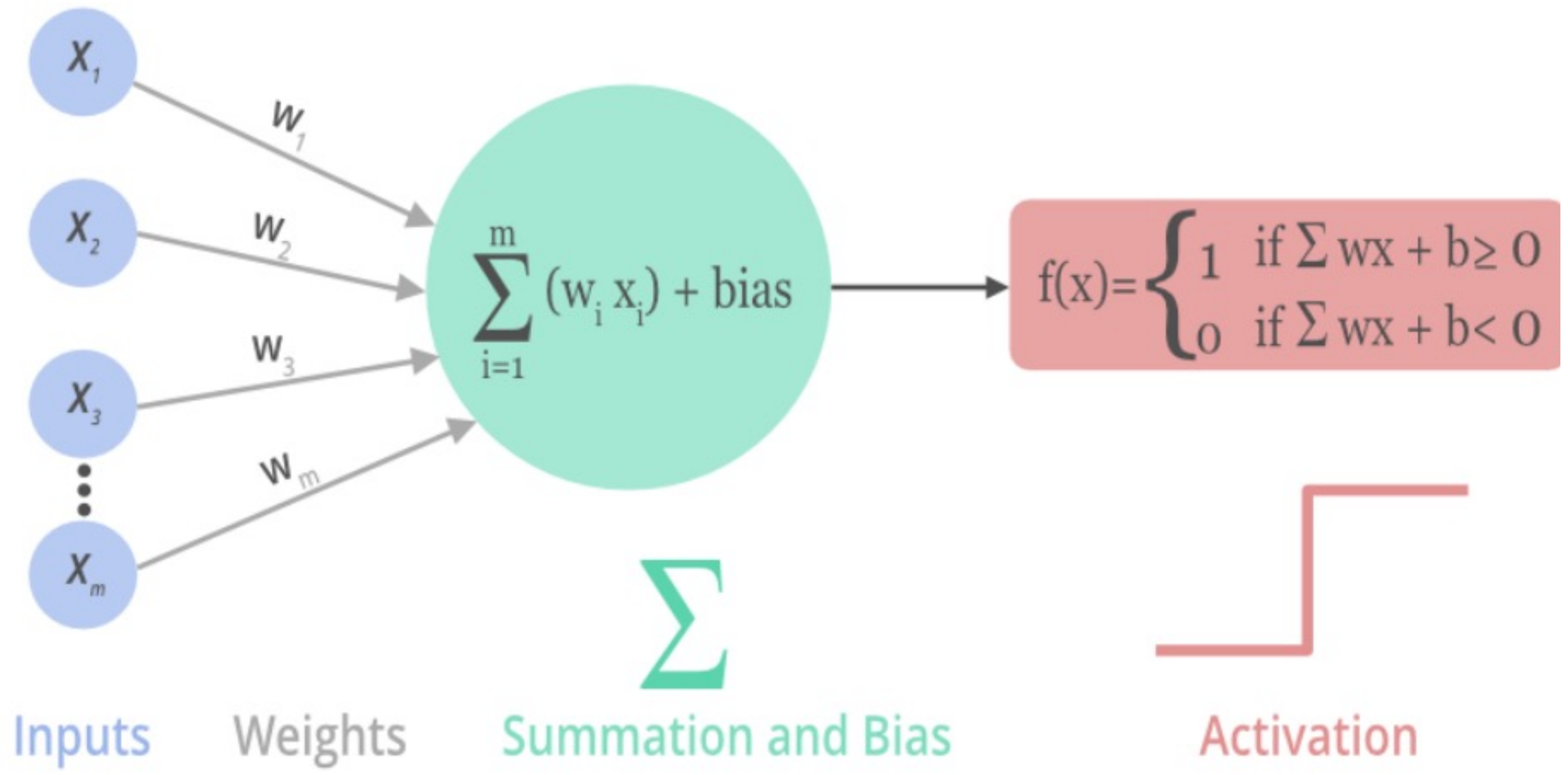
II. Loss Function

III. Optimization Function

ACTIVATION FUNCTIONS

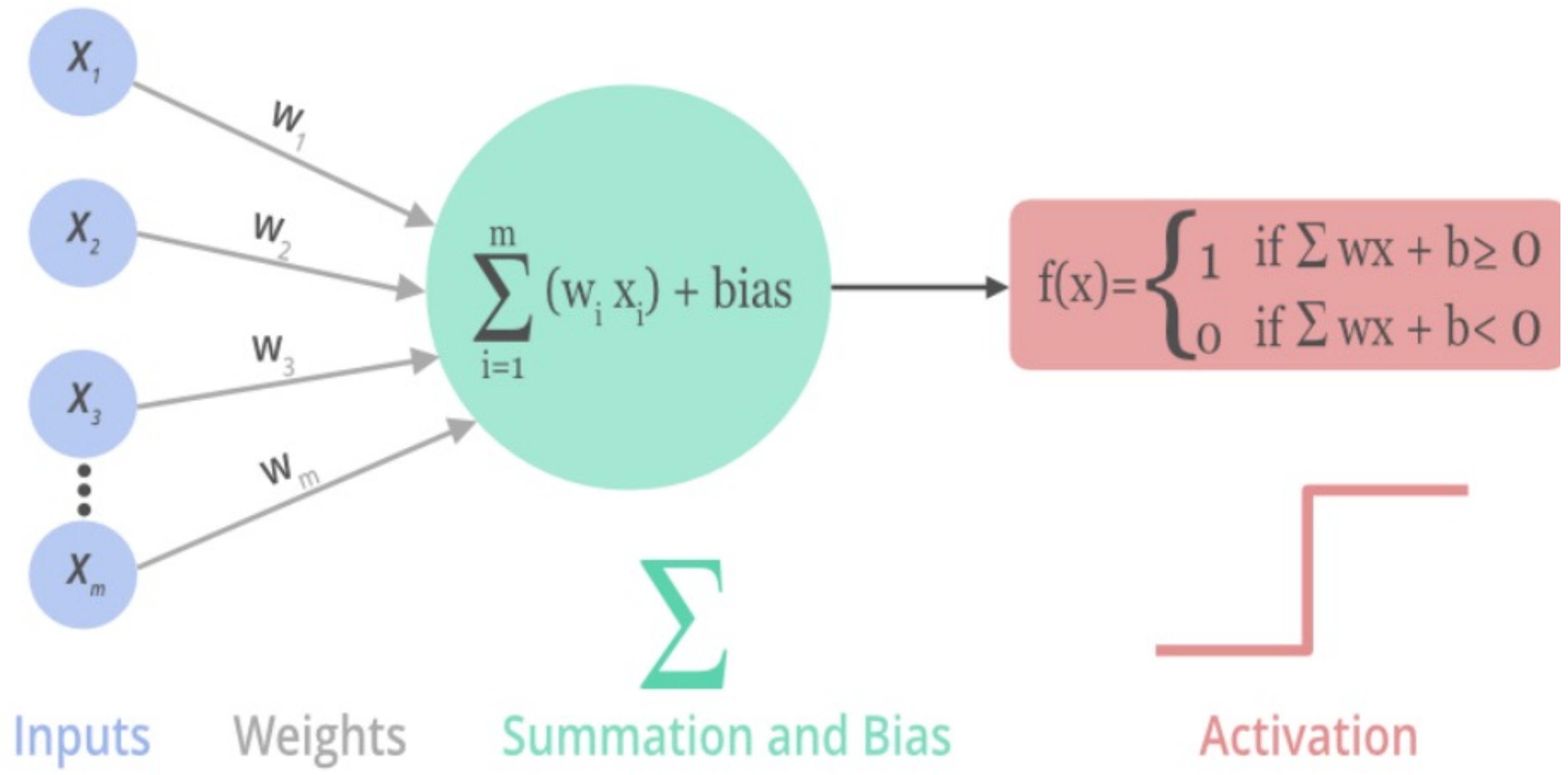
Sometimes data cannot be separated using linear methods!

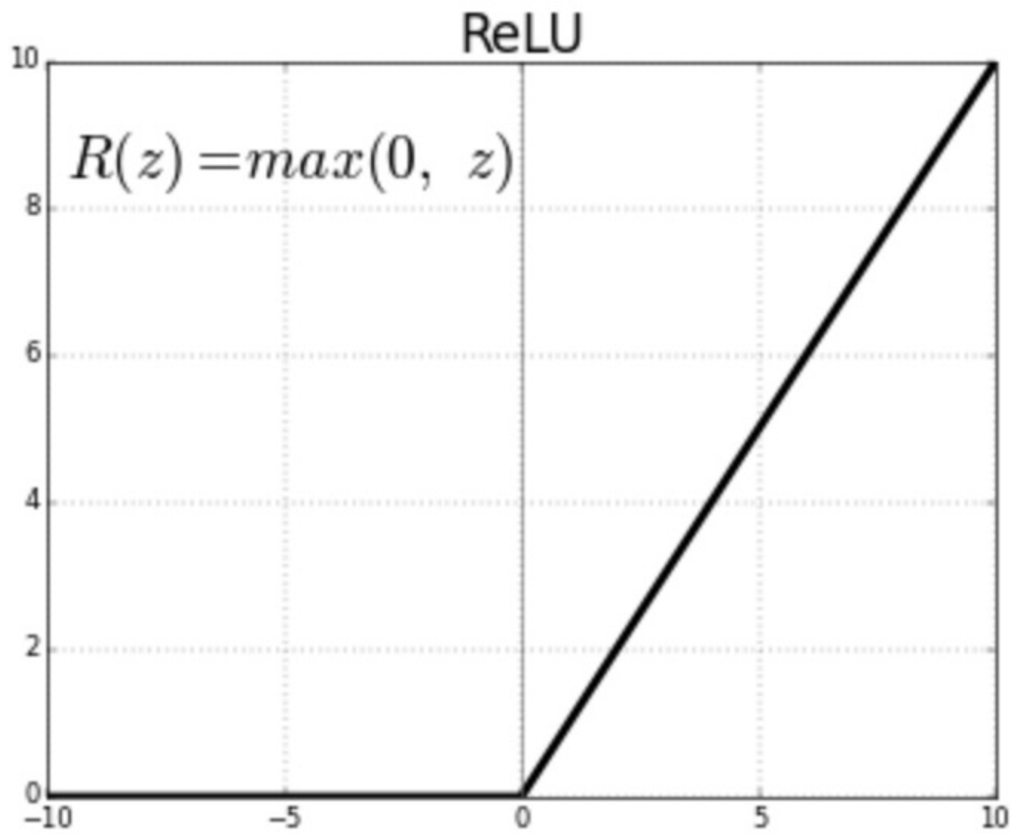
Activation functions transform the value of each neuron into a non-linear space


X_1
X_2
X_3
X_m
w_1
w_2
w_3
w_m
$\sum_{i=1}^{m}(w_i x_i) + \text{bias}$
$f(x)=\begin{cases}1 & \text{if } \sum wx + b \geq 0\\ 0 & \text{if } \sum wx + b < 0\end{cases}$
$\sum$
Inputs
Weights
Summation and Bias
Activation

ACTIVATION FUNCTIONS

- The rectified linear unit function (ReLU) is the most widely used activation function
- ReLU does not affect values which are greater than zero, sets all negative values to zero
- This prevent negative neurons (mostly negative weights) from being inputs into future neurons
- These "bad" features would hinder learning + classification by canceling out good features


X_1
X_2
X_3
X_m
w_1
w_2
w_3
w_m
$\sum_{i=1}^{m}(w_i x_i) + \text{bias}$
$f(x)=\begin{cases}1 & \text{if } \sum wx + b \geq 0 \\ 0 & \text{if } \sum wx + b < 0\end{cases}$
Σ
Inputs
Weights
Summation and Bias
Activation



ReLU Activation Function

LOSS FUNCTIONS

We know that neural networks update the weights based on prediction accuracy

How does the neural network assess prediction accuracy? With a loss function

Labeled data has absolute ground truth (so the probability value = one)

Output layer of the network calculates the probability that a data point has a given label

Most basic way to calculate error is through subtraction

Data point is in class 2, NN thinks data has a 73% probability of being in class 2: 1-.73 = 0.27

LOSS FUNCTIONS

Loss functions are cleverer than just subtraction

We want wrong, confident predictions to have a large loss (weight adjustments will be bigger)

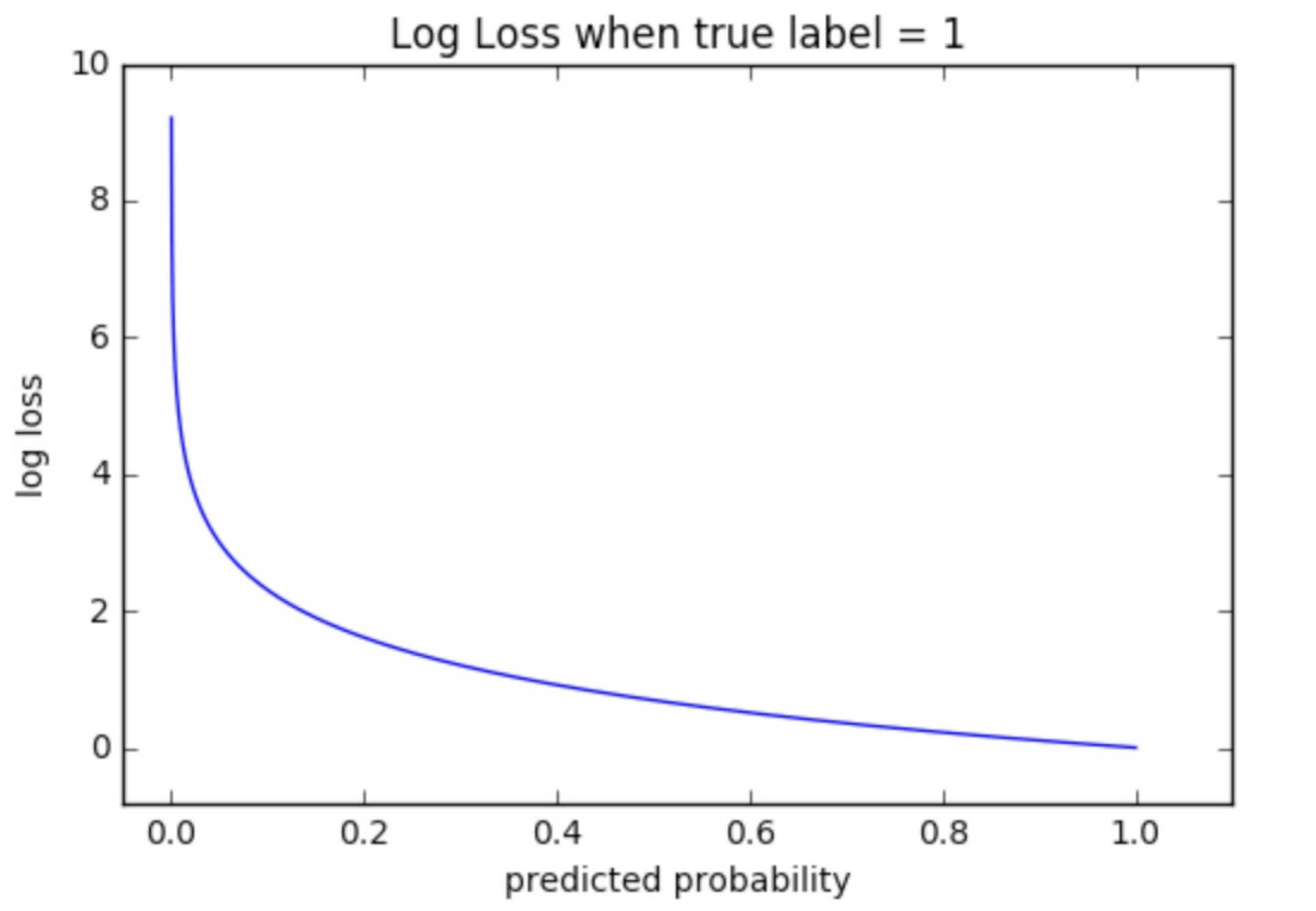
Cross-entropy loss is a very common choice for machine learning tasks

Can accommodate binary and multiclass data

LOSS FUNCTIONS

- Cross entropy loss uses the log of the probability to severely penalize confident mistakes
- In multiclass cases, adds up the loss from across all the classes (see figure)
- In multiclass case, several big mistakes are worse than a bunch of small ones

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$


Log Loss when true label = 1
log loss
predicted probability
10
8
6
4
2
0
0.0
0.2
0.4
0.6
0.8
1.0

OPTIMIZATION FUNCTIONS

- The error is derived from the loss function, and we want to move down the slope of the error function to find the minima
- Partial derivatives are used to determine the slope of the error function for each feature/weight pair
- Once we have the individual slopes, increase/decrease the weights so the neurons will contribute less to the error
- Excellent example (with actual numbers) here

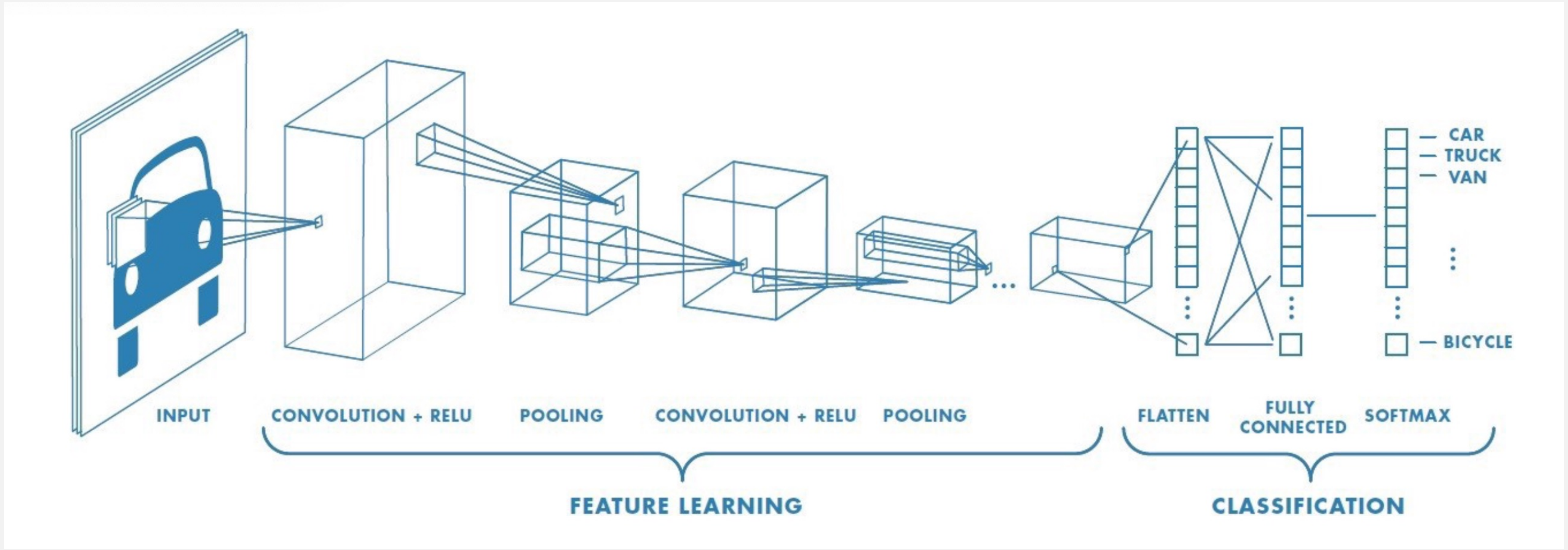
CONVOLUTIONAL NEURAL NETWORKS

CONVOLUTIONAL NEURAL NETWORK (CNN)

- CNNs combine the convolution operation with the traditional ANN structure
- The convolution operation captures local details about an image/matrix
- Used in image analysis to identify important features that can be used for classification

CONVOLUTIONAL NEURAL NETWORK (CNN)

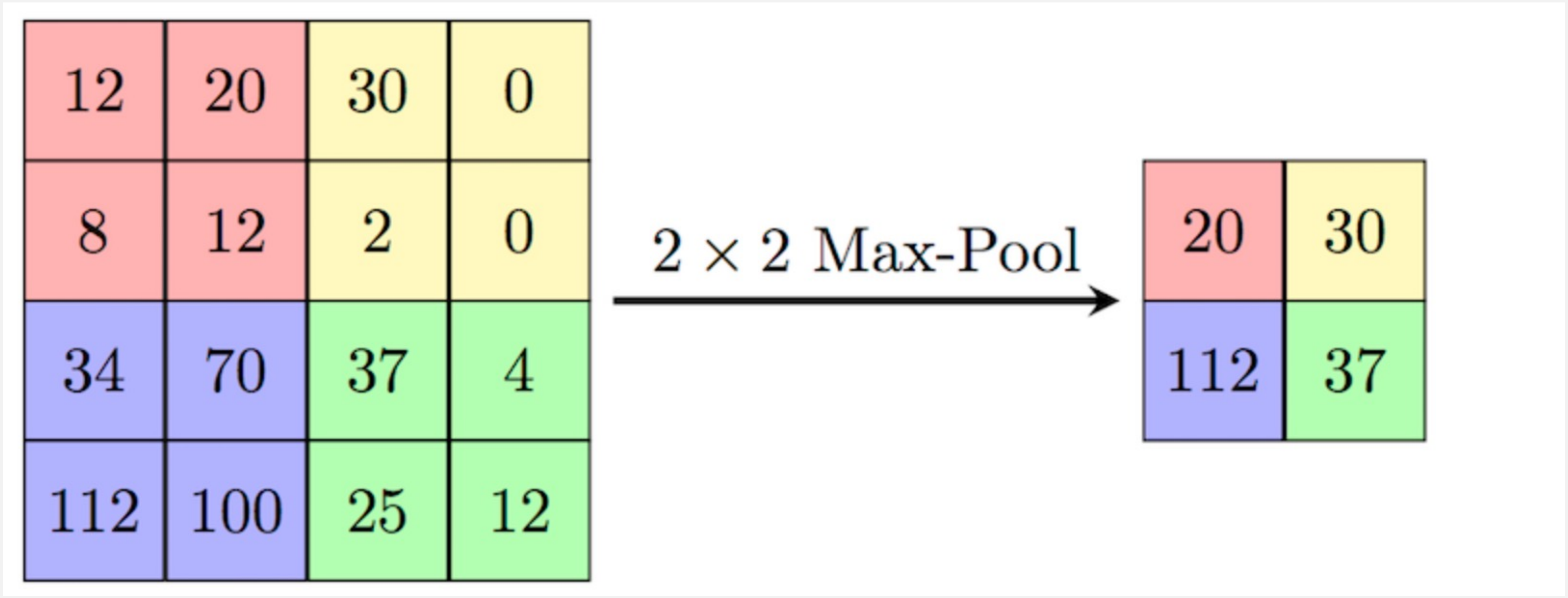
- Many different convolution kernels are passed over the regions of the data
- This captures different perspectives of each region that could represent key features
- These features are fed into an MLP to optimally weighted feature combos


CAR
TRUCK
VAN
BICYCLE
INPUT
CONVOLUTION + RELU
POOLING
CONVOLUTION + RELU
POOLING
FLATTEN
FULLY CONNECTED
SOFTMAX
FEATURE LEARNING
CLASSIFICATION

MAX POOLING

Selects the highest kernel output value in a localized region of the data

This value is used to represent whatever structure/object/entity is in that local region


12
20
30
0
8
12
2
0
34
70
37
4
112
100
25
12
2 × 2 Max-Pool
20
30
112
37

AUTOENCODER

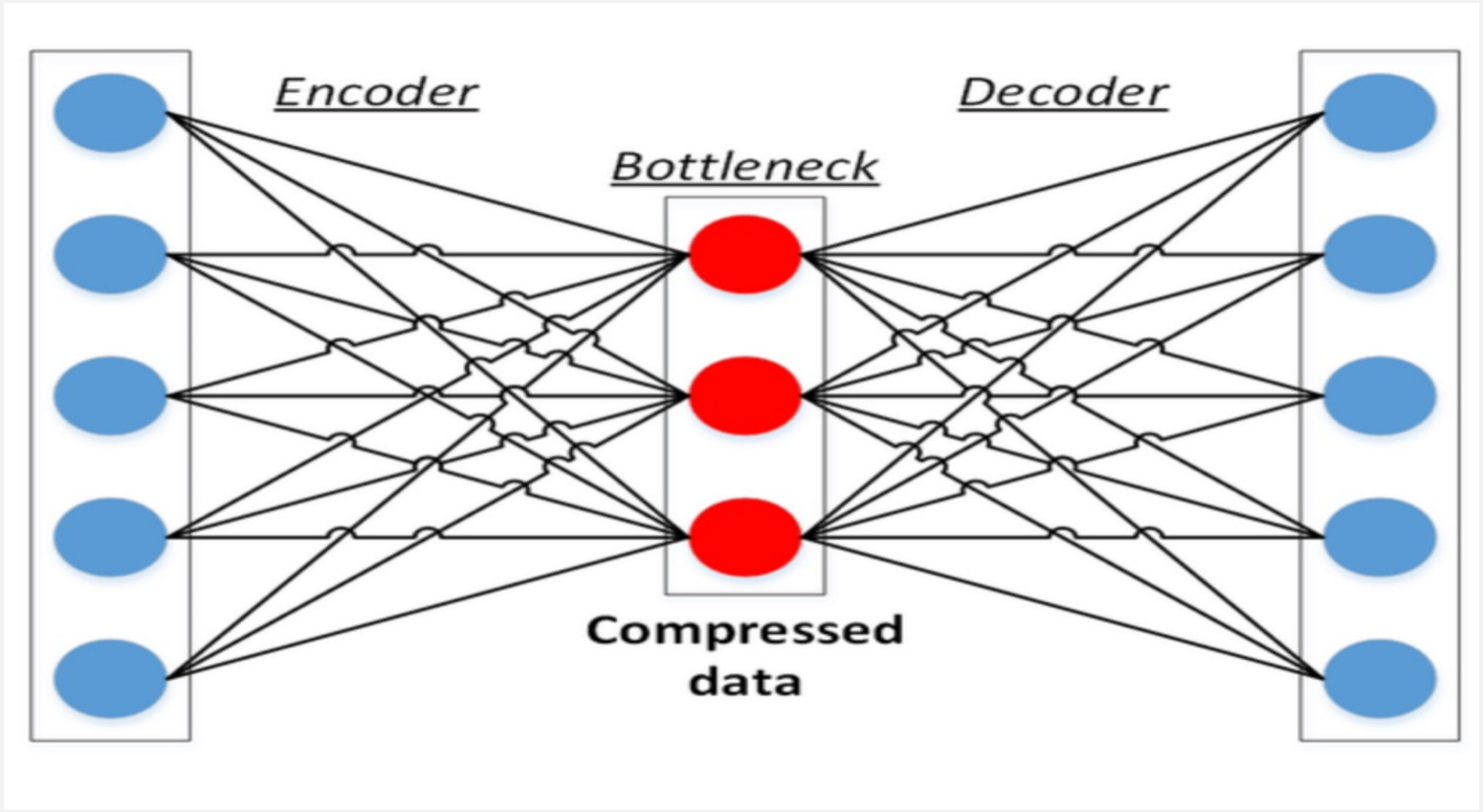
THE AUTOENCODER

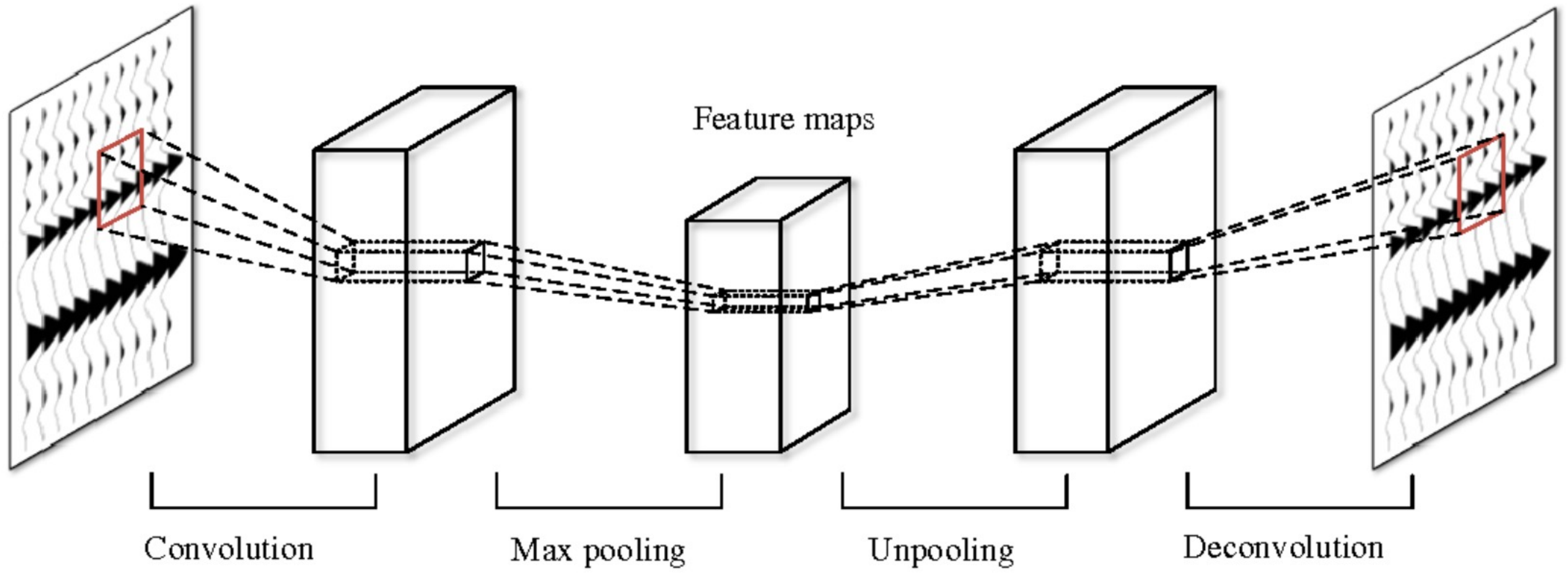
A common type of "unsupervised" deep learning is dimensionality reduction via the autoencoder

Strip the data down to a few features which correspond to combinations of features transformed by an activation function

Optimize the weights of these features such that the output layer is identical to the input layer

The resultant features capture the useful info in the data and can be used for low-dimensional, noiseless analysis (aka clustering)


Encoder
Decoder
Bottleneck
Compressed data

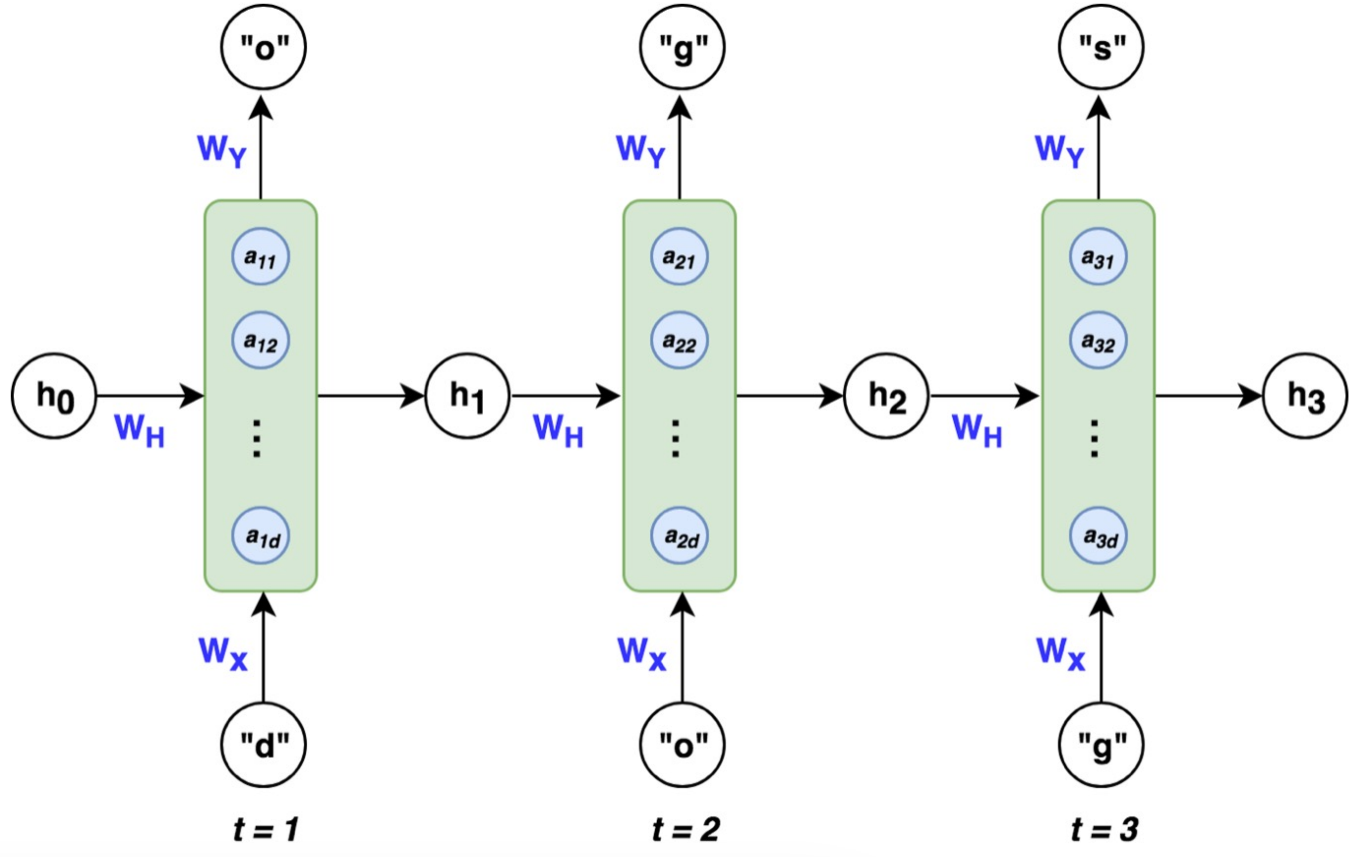

Feature maps
Convolution
Max pooling
Unpooling
Deconvolution

LSTM

RECURRENT NEURAL NETWORKS

- ANNs connected together
- Information from prior networks are used to influence predictions of the future ones
- Simple example of what RNNs can learn: x is important if y occurred in a prior state

Simple Example



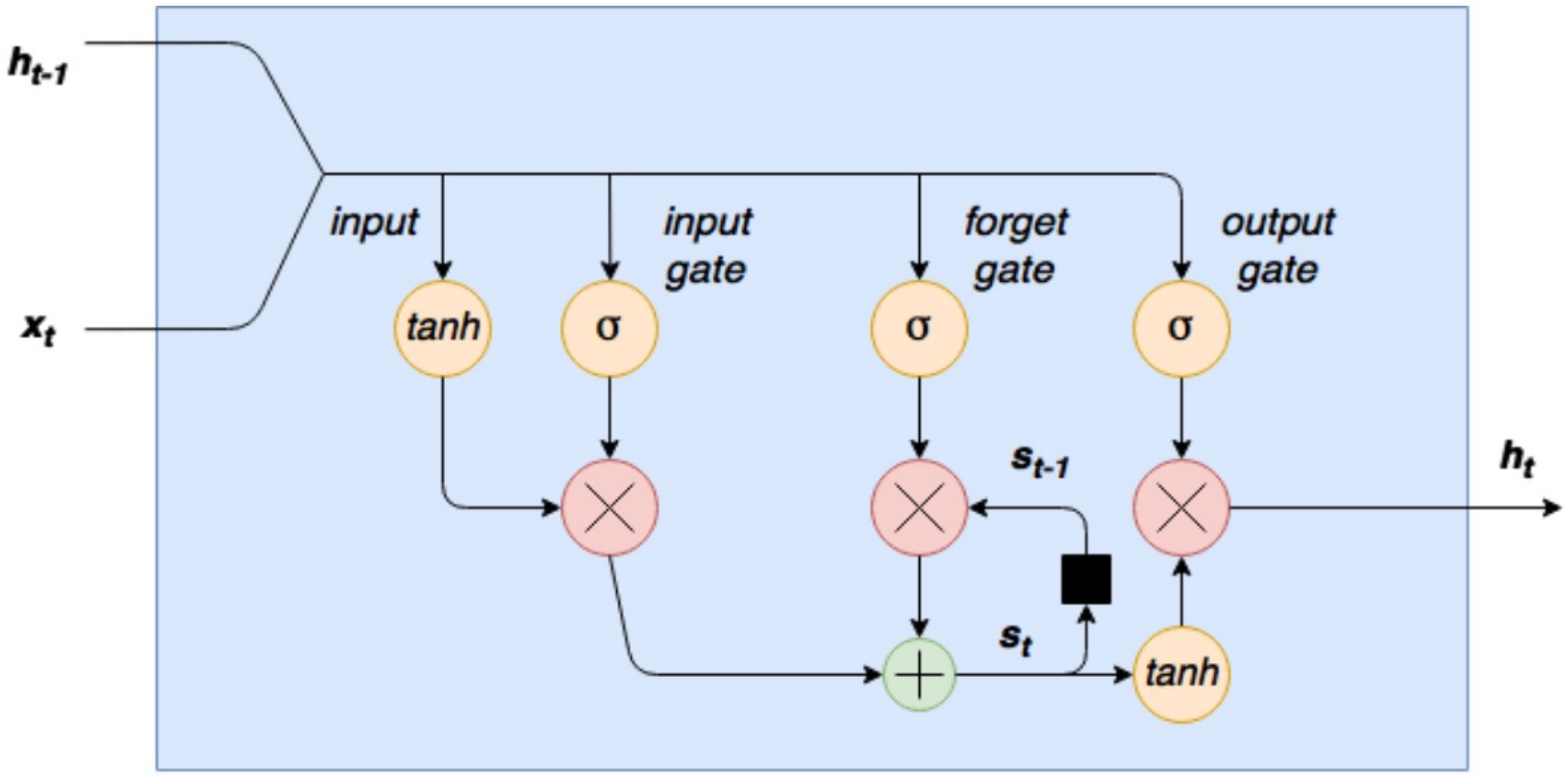
THE EPIC LSTM

LSTMs have 3 main components

I. Long Term Memory (important long-term info)

II. The input from the prior state (short term info, like RNN)

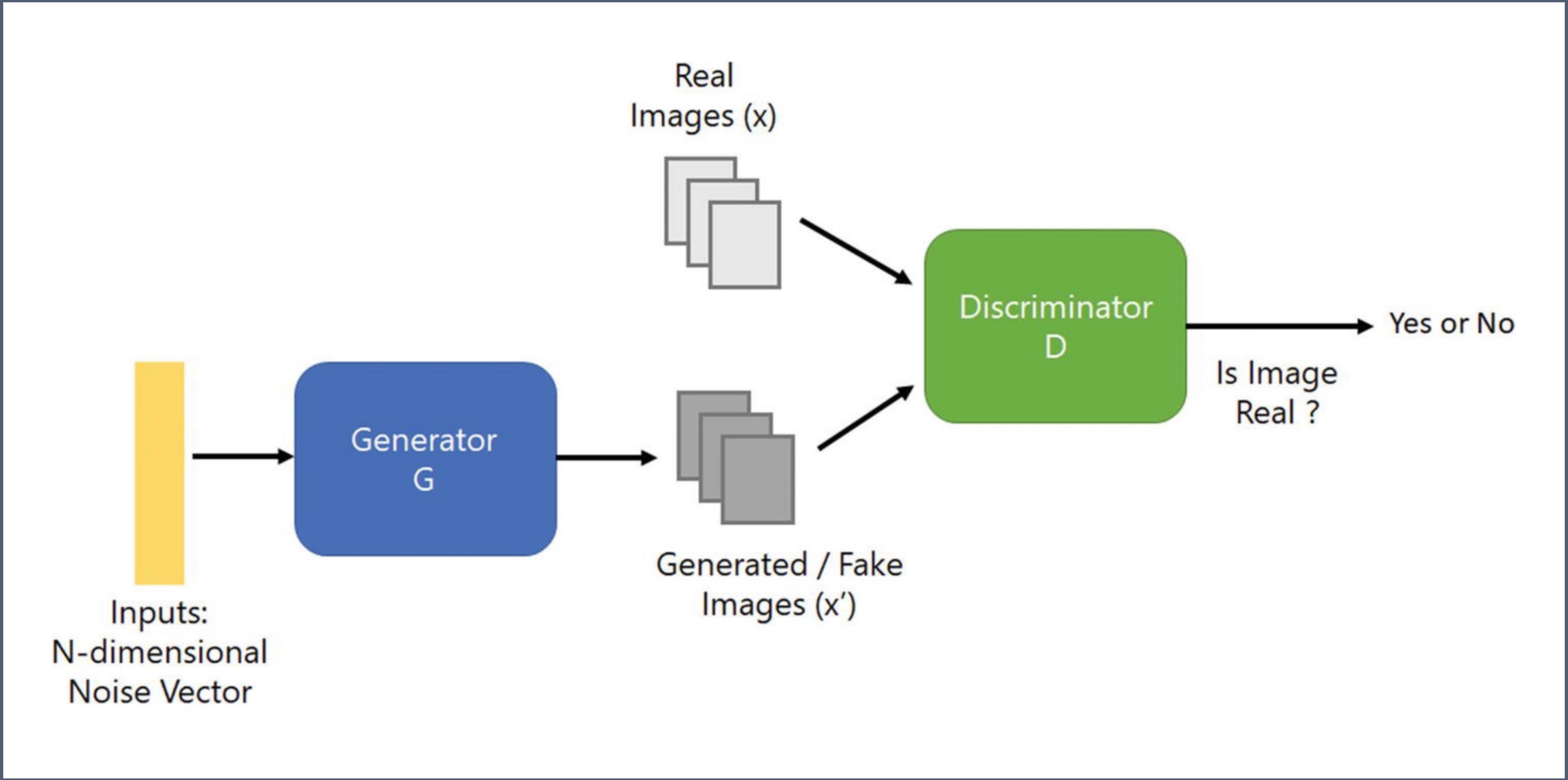
III. Output – determining what moves on


h_{t-1}
x_t
input
input gate
forget gate
output gate
tanh
σ
σ
σ
s_{t-1}
h_t
s_t
tanh

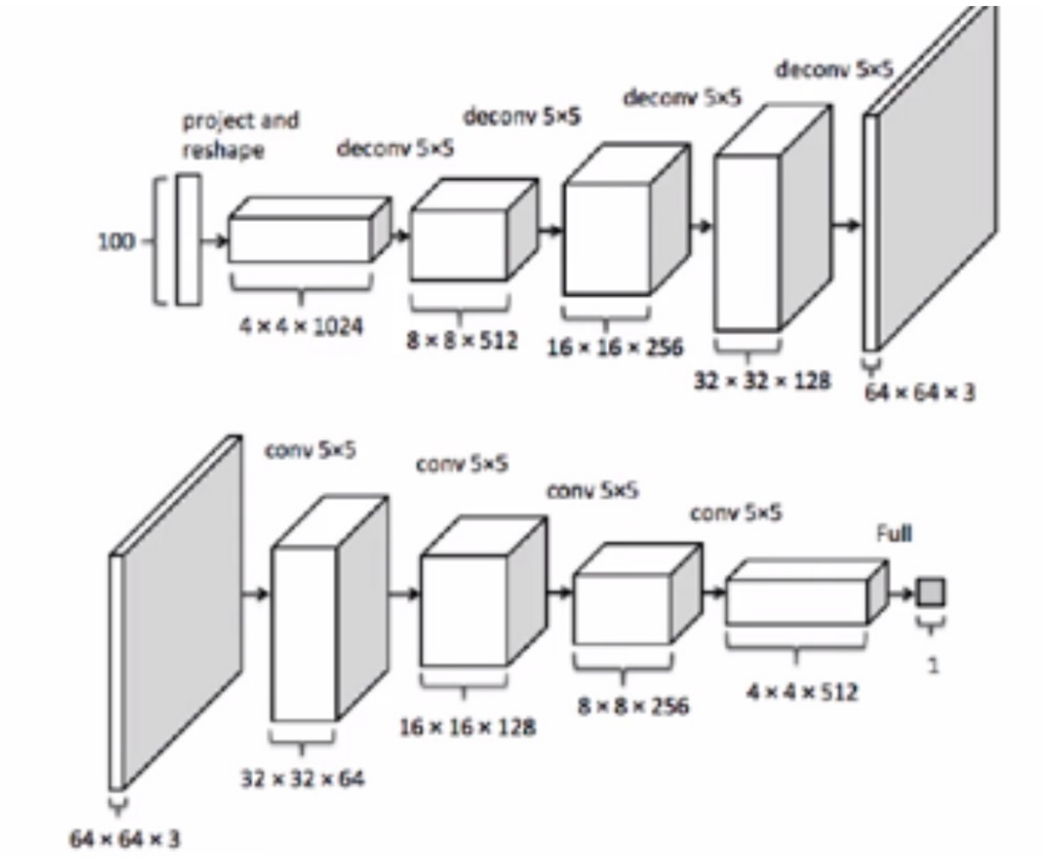
GENERATIVE ADVERSARIAL NETWORKS (GAN)

GENERATIVE ADVERSARIAL NETWORKS

- Deep Learning becomes a game
- Two neural networks compete against each other
- One of them generates fake data, and other tries to distinguish real from fake


Real Images (x)
Discriminator D
Yes or No
Is Image Real ?
Generator G
Generated / Fake Images (x′)
Inputs: N-dimensional Noise Vector

FOR IMAGES


100
project and reshape
deconv 5×5
deconv 5×5
deconv 5×5
deconv 5×5
4 × 4 × 1024
8 × 8 × 512
16 × 16 × 256
32 × 32 × 128
64 × 64 × 3
conv 5×5
conv 5×5
conv 5×5
conv 5×5
Full
64 × 64 × 3
32 × 32 × 64
16 × 16 × 128
8 × 8 × 256
4 × 4 × 512
1

TRAINING A GAN

- Remember, this is a game
- If the discriminator does well, the generator must be losing
- If the generator does well, the discriminator must be losing
- Lower loss for the discriminator = higher loss for the generator
- Higher loss for the discriminator = lower loss for the generator

TRAINING A GAN

- If we train both networks at the same time, it will be like trying to hit a moving target
- We train the generator for a few epochs, and do not change the weights of the discriminator
- We then train the discriminator for a few epochs, do not change the weights of the generator
- Repeat for a certain number of epochs until the discriminator fails half the time

GENERAL GUIDELINES

- Use the same things you would on a regular neural network
- Activation functions
- Optimization functions
- Loss functions
- Batch normalization (prevents GANs from replicating the same sample over and over)
- Dropouts