Intro to Deep Learning

DataHacks 2020
Presenter: Victor

What is Deep Learning?

What is Deep Learning?

ARTIFICIAL INTELLIGENCE

Any technique that enables computers to mimic human behavior

MACHINE LEARNING

Ability to learn without explicitly being programmed

DEEP LEARNING

Extract patterns from data using neural networks

MIT 6.S191

What is Deep Learning?

Just like any other ML model, we are given input X and want to predict output Y

1. Email Spam Detection
   a. X = email text body
   b. Y = {Spam, Not Spam}
2. Stock Price Prediction
   a. X = series of stock prices from the last week
   b. Y = stock price for tomorrow
3. Cat/Dog Image Classification
   a. X = image (series of pixel values)
   b. Y = {Cat, Dog}

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

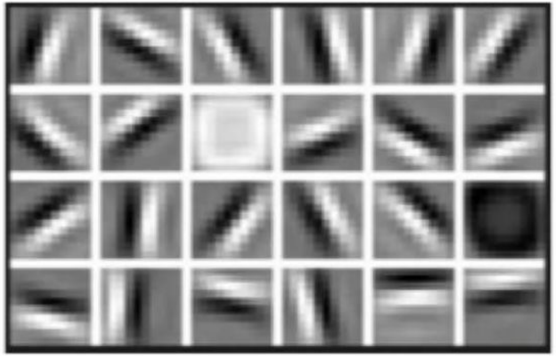
However, the difficulty in a lot of ML problems is feature selection/extraction!

What is Deep Learning?

Hand engineered features are time consuming, brittle, and not scalable in practice

Can we learn the **underlying features** directly from data?

Low Level Features



Lines & Edges

Mid Level Features

Eyes & Nose & Ears

High Level Features

Facial Structure

What Makes Deep Learning Special?

Machine Learning	**Deep Learning**
Works on small datasets	Works on large datasets
Works on low-end computers	Dependent on high-end computer
Takes less time to train	Takes more time to train
Takes more time to test	Takes less time to test

What Makes Deep Learning Special?

1952 Stochastic Gradient Descent

1958 Perceptron
- Learnable Weights

⋮

1986 Backpropagation
- Multi-Layer Perceptron

1995 Deep Convolutional NN
- Digit Recognition

⋮

Neural Networks date back decades, so why the resurgence?

1. Big Data
- Larger Datasets
- Easier Collection & Storage

IMAGENET

WIKIPEDIA
The Free Encyclopedia

2. Hardware
- Graphics Processing Units (GPUs)
- Massively Parallelizable

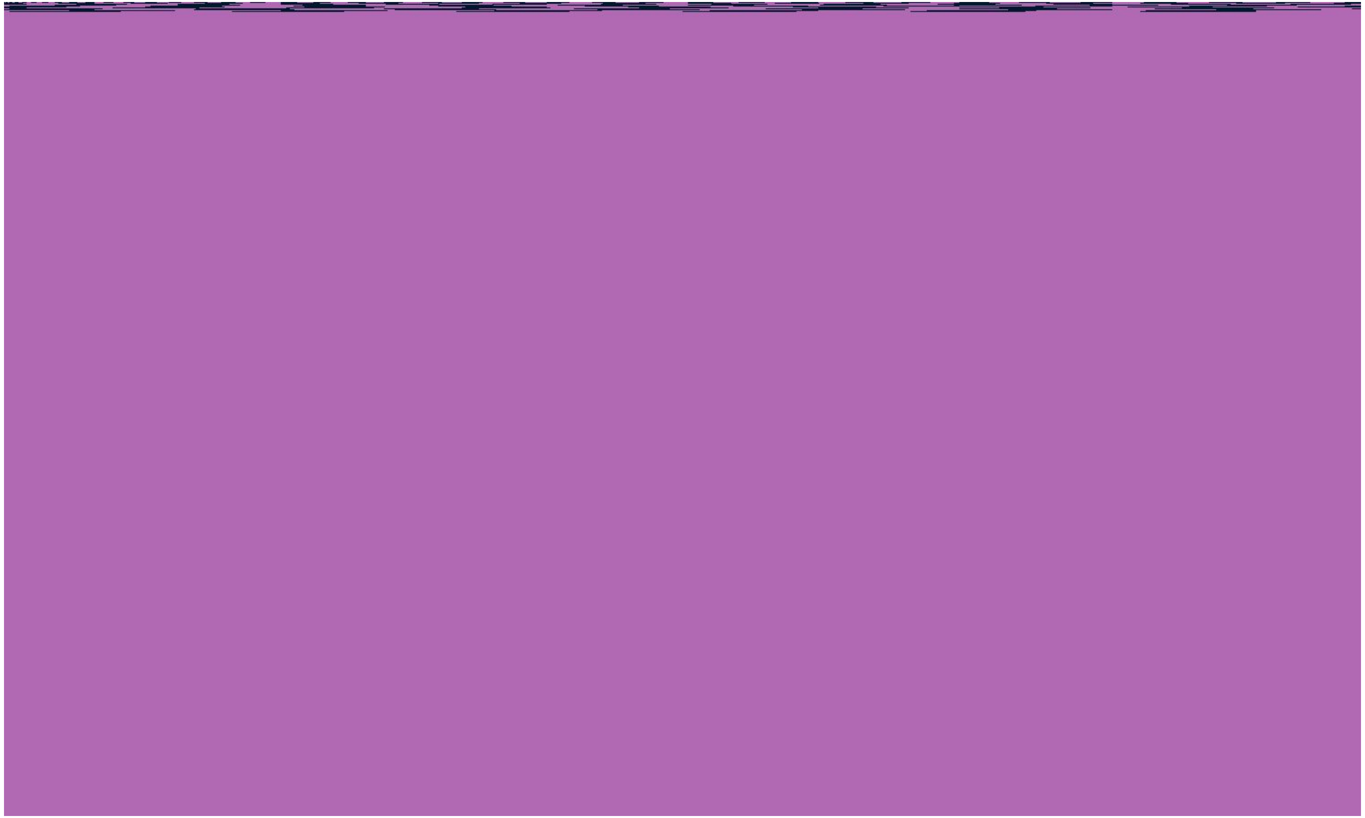
3. Software
- Improved Techniques
- New Models
- Toolboxes

TensorFlow

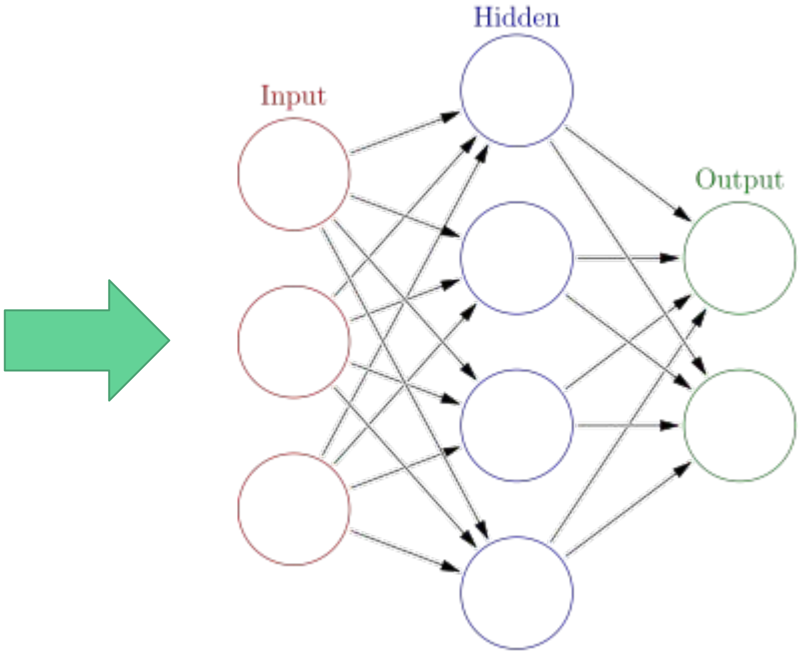
MIT 6.S191

How Does Deep Learning Work?

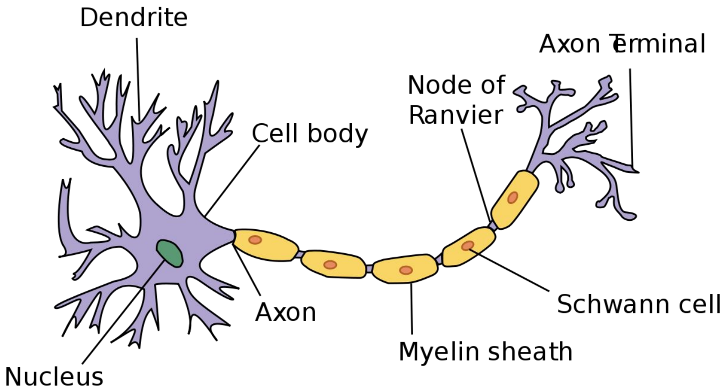
How Does Deep Learning Work?



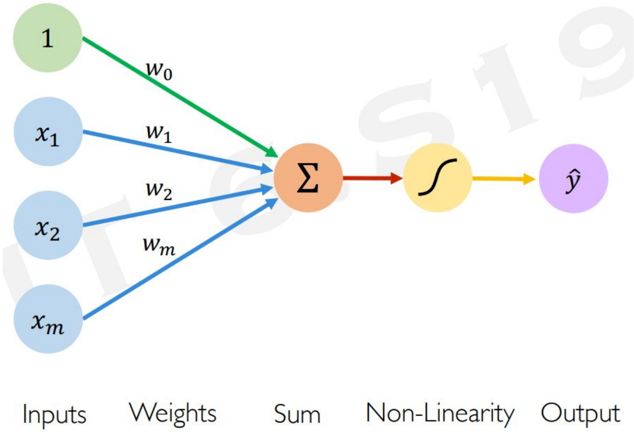
What is a Neural Network?



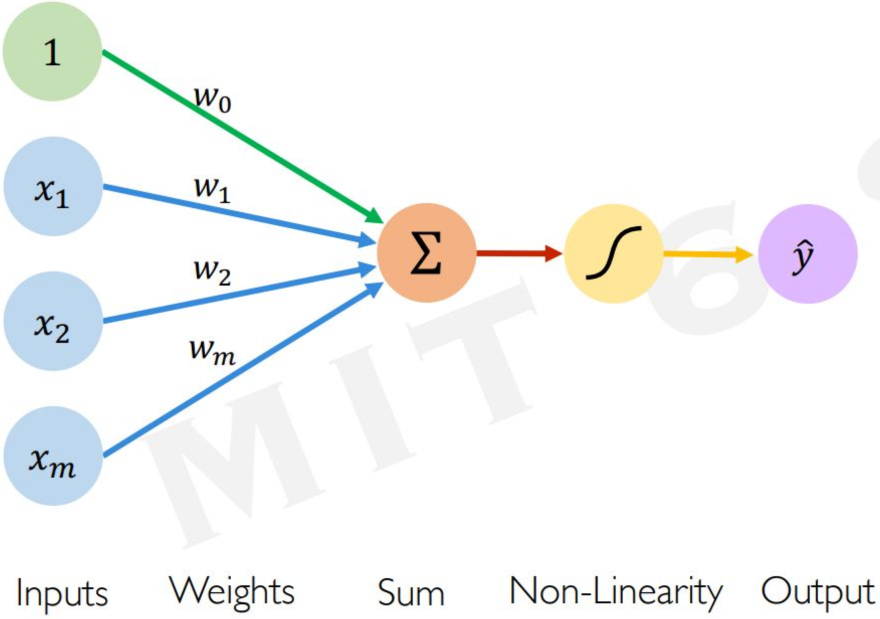
What is a Perceptron?



$$\hat{y} = g(\, w_0 + \boldsymbol{X}^T \boldsymbol{W} \,)$$



The Perceptron



Output

Linear combination of inputs

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i \right)$$

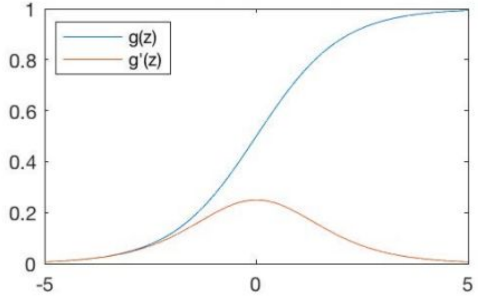
Non-linear activation function

Bias

MIT 6.S191

Common Activation Functions

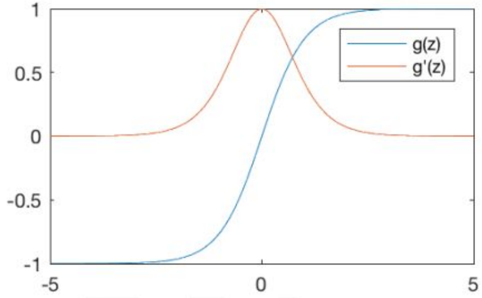
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

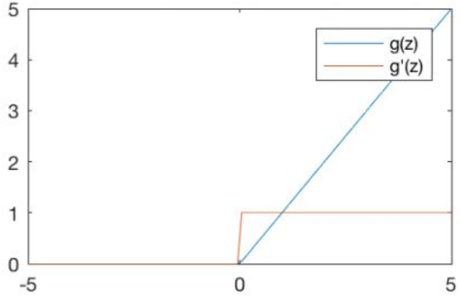
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)

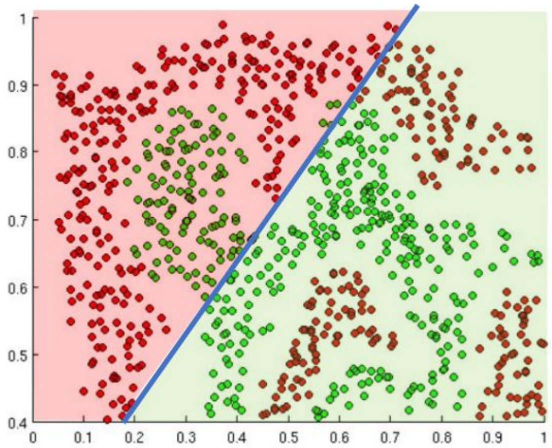


$$g(z) = \max(0, z)$$

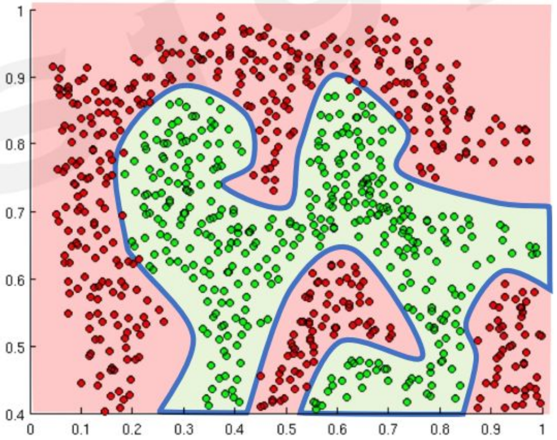
$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

Importance of Activation Functions

The purpose of activation functions is to ***introduce non-linearities*** *into the network*



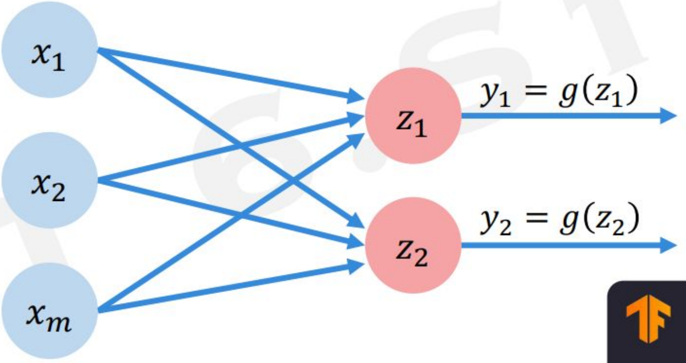
Linear activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

Multi-Output Perceptron

Because all inputs are densely connected to all outputs, these layers are called **Dense** layers

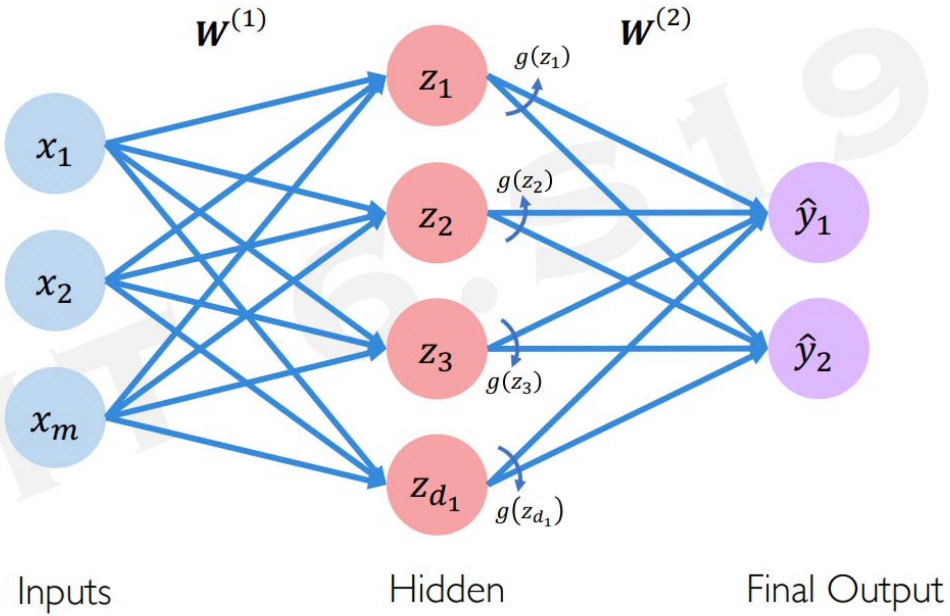



```python
import tensorflow as tf

layer = tf.keras.layers.Dense(
    units=2)
```

$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

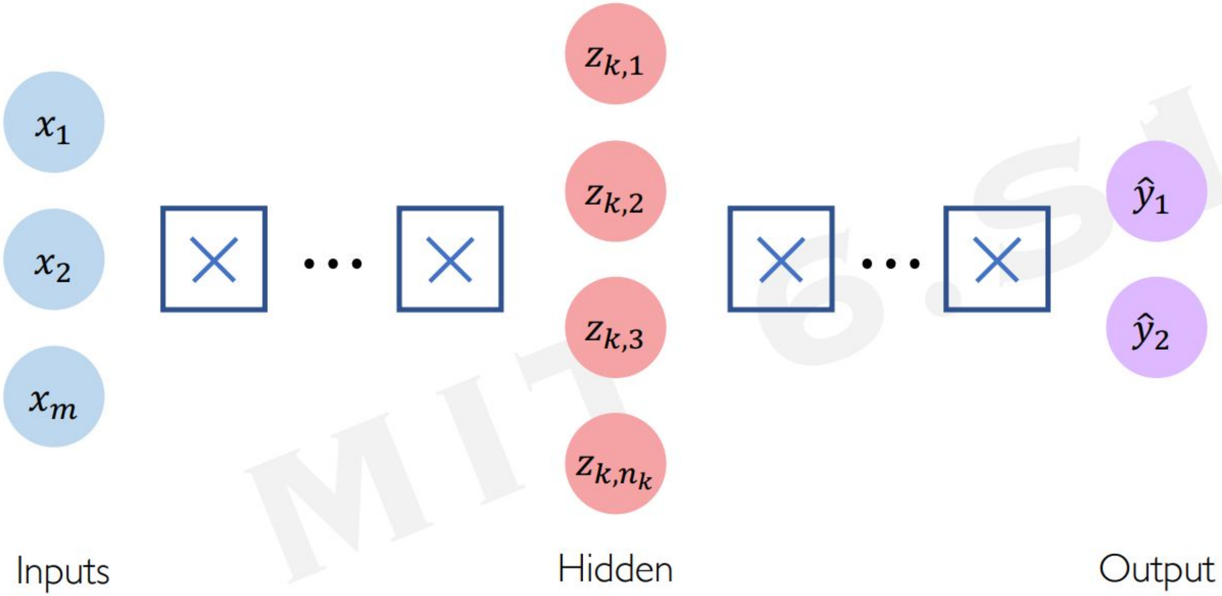
MIT 6.S191

Single Layer Neural Network



$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j\, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} g(z_j)\, w_{j,i}^{(2)}\right)$$

Deep Neural Networks




```python
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Dense(n₁),
    tf.keras.layers.Dense(n₂),
    ⋮
    tf.keras.layers.Dense(2)
])
```

$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

MIT 6.S191

Training Neural Networks

Loss Optimization

We want to find the network weights that ***achieve the lowest loss***

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

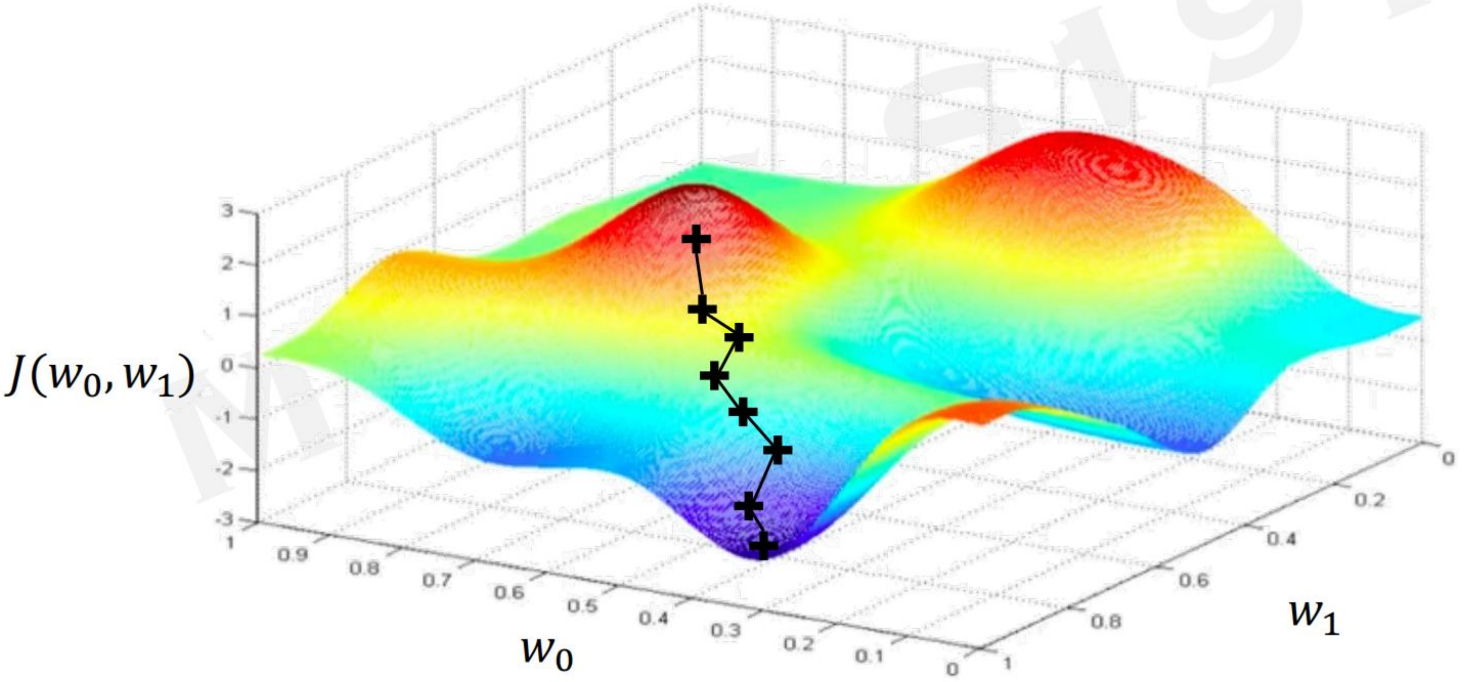
$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W})$$

Remember:

$$\boldsymbol{W} = \left\{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \cdots\right\}$$

MIT 6.S191

Gradient Descent



Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\boxed{\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

```python
import tensorflow as tf

weights = tf.Variable([tf.random.normal()])

while True:    # loop forever
    with tf.GradientTape() as g:
        loss = compute_loss(weights)
        gradient = g.gradient(loss, weights)

    weights = weights - lr * gradient
```

MIT 6.S191

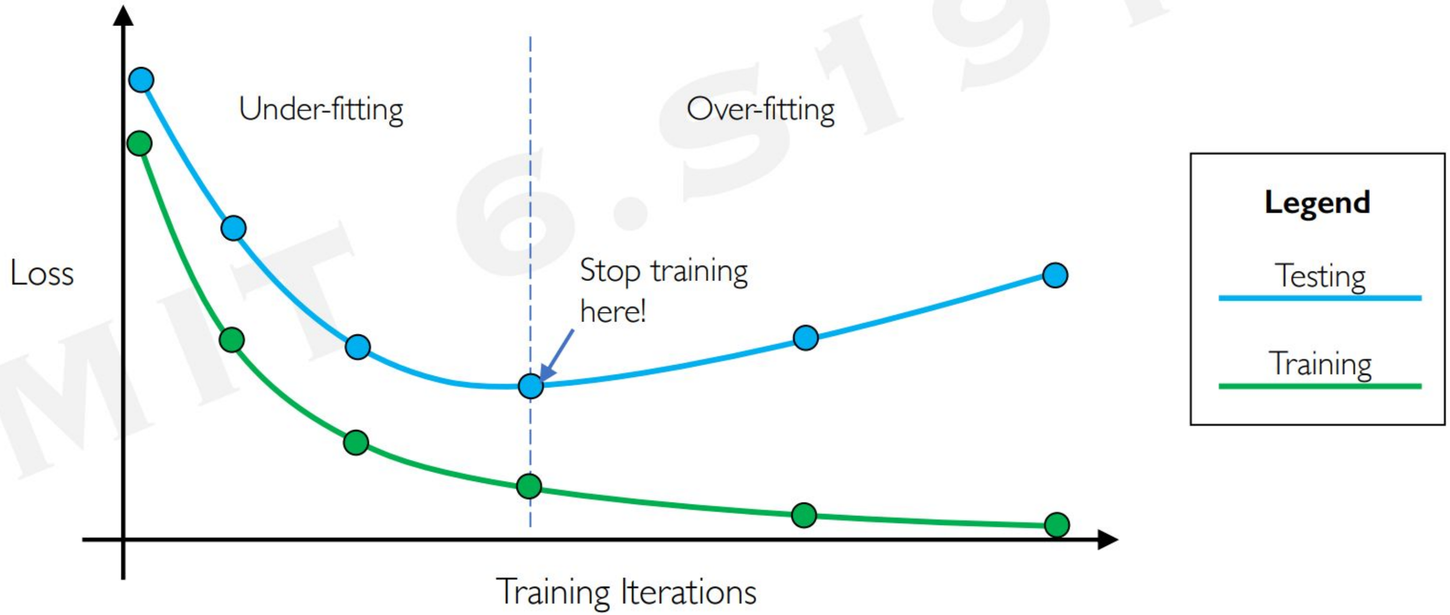
Putting It All Together

```python
import tensorflow as tf

model = tf.keras.Sequential([...])

# pick your favorite optimizer
optimizer = tf.keras.optimizer.SGD()

while True: # loop forever

    # forward pass through the network
    prediction = model(x)

    with tf.GradientTape() as tape:
        # compute the loss
        loss = compute_loss(y, prediction)

    # update the weights using the gradient
    grads = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables)))
```

MIT 6.S1

Prevent Overfitting

- Stop training before we have a chance to overfit

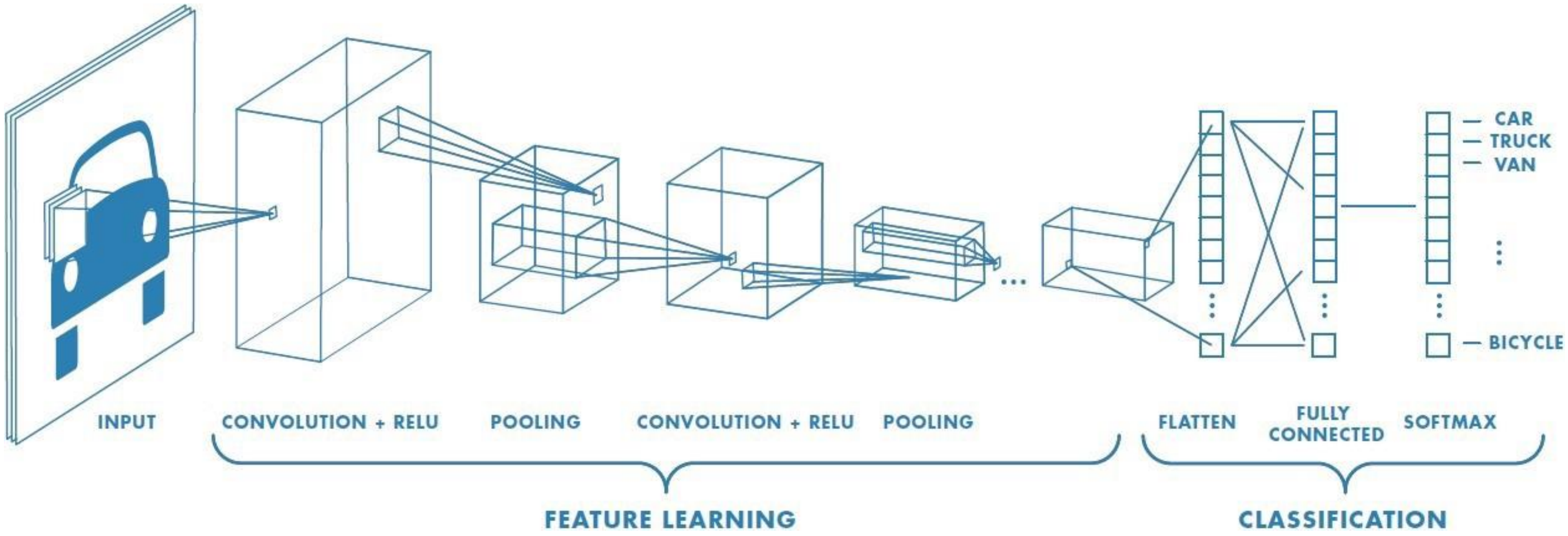


MIT 6.S191

Deep Learning for NLP

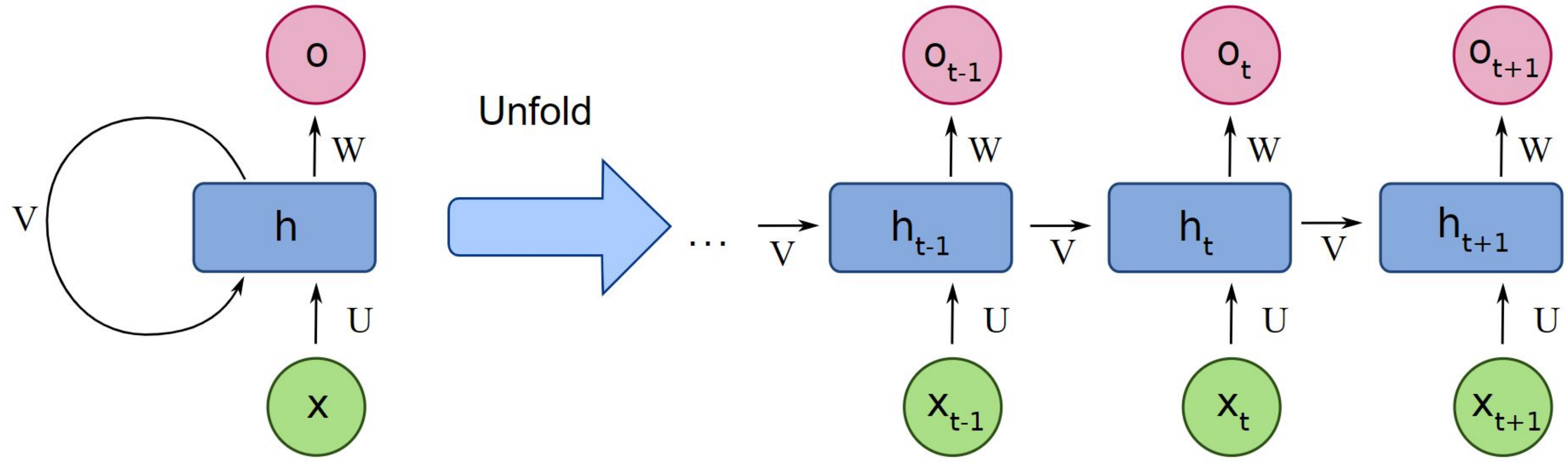
Convolutional Neural Networks (CNNs)

- Mainly used for computer vision tasks
  - image recognition, object detection, etc.



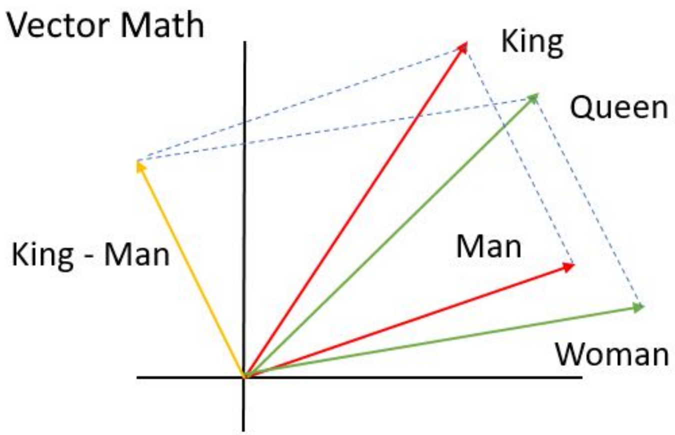
Recurrent Neural Networks (RNNs)

- Mainly used for NLP
  - Language modeling, **word embeddings**, etc.



Representing Text Data

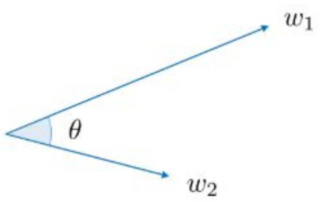
- Idea: Each word is vectorized (converting text into numbers)
- We take each sentence and convert it into a vector of size (number of unique words in all sentences)
- The goal is to get similar sentences closer together in a high dimensional vector space



Cosine similarity — The cosine similarity between words w_1 and w_2 is expressed as follows:

$$\boxed{\text{similarity} = \frac{w_1 \cdot w_2}{||w_1||\ ||w_2||} = \cos(\theta)}$$

Remark: θ is the angle between words w_1 and w_2.

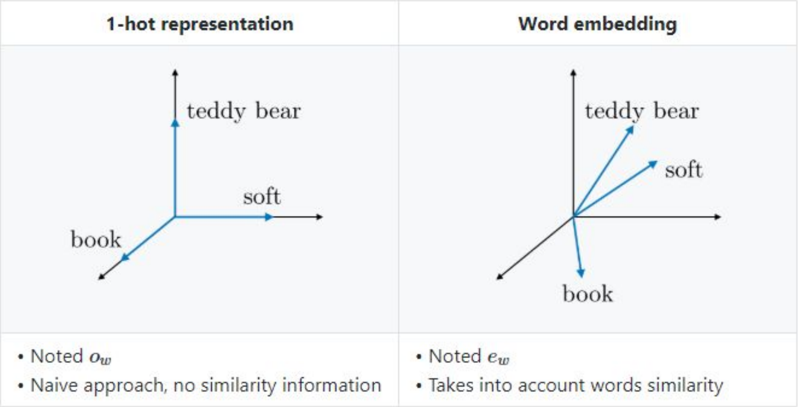


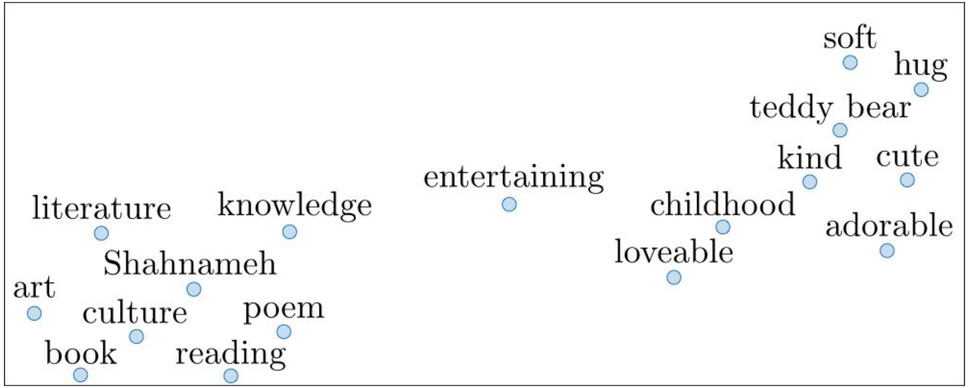
Bag of Words (BOW)

- Example 1:
    - Dictionary = {the, cat, sat, dog, orange, apple, grape}
        - Sentence 1 = “The cat sat” vec1 = [1, 1, 1, 0, 0, 0, 0]
        - Sentence 2 = “The dog sat” vec2 = [1, 0, 1, 1, 0, 0, 0]
        - Sentence 3 = “Orange, apple, grape” vec3 = [0, 0, 0, 0, 1, 1, 1]

- Example 2:

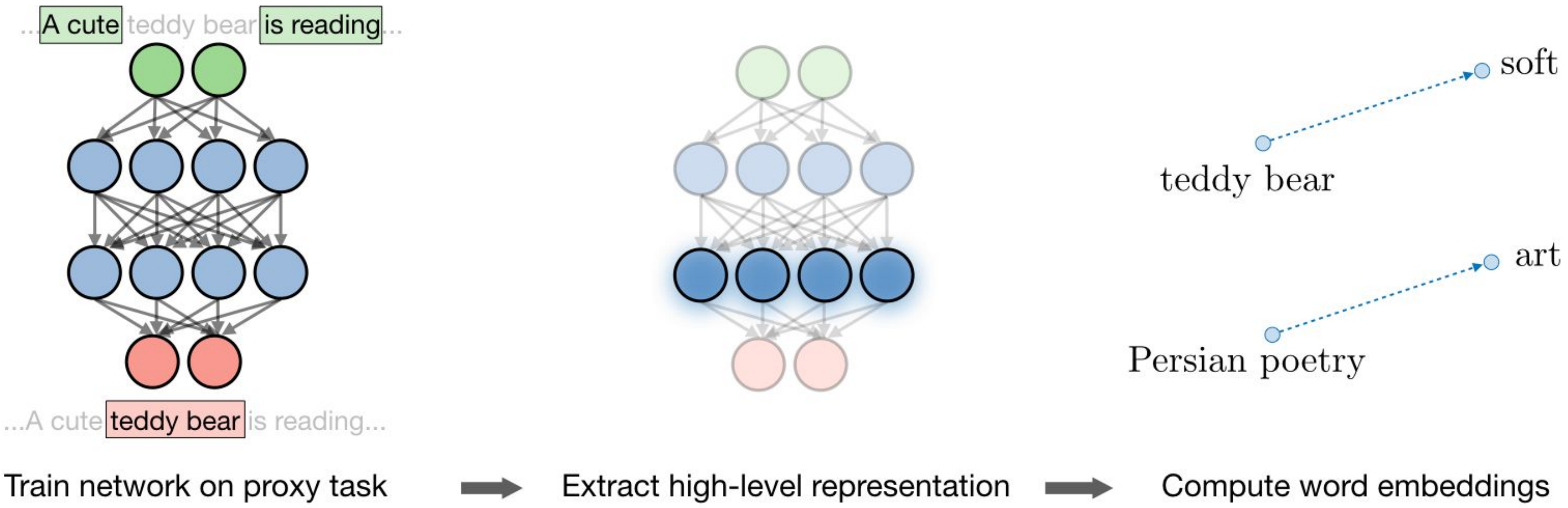
Word Embeddings





Word2Vec

- Word2vec is a framework aimed at learning word embeddings by estimating the likelihood that a given word is surrounded by other words



Deep Learning Sources

- Introduction to Deep Learning
- An Introduction to Deep Learning
- Introduction to Deep Learning
- Neural networks and deep learning
- A Beginner's Guide to Neural Networks and Deep Learning
- CS 230 - Recurrent Neural Networks Cheatsheet
- Recurrent Neural Networks

NLP Resources

- Natural Language Toolkit — NLTK 3.4.5 documentation
- Working With Text Data — scikit-learn 0.22.1 documentation
- 6.2. Feature extraction — scikit-learn 0.22.1 documentation
- gensim: models.word2vec – Word2vec embeddings
- A Beginner's Guide to Word2Vec and Neural Word Embeddings
- Introduction to Word Embedding and Word2Vec
- Word2Vec Tutorial - The Skip-Gram Model
- Word2vec Tutorial