# Review of Machine Learning

Kate Farrahi

ECS Southampton

January 24, 2019

# Three Types of Learning

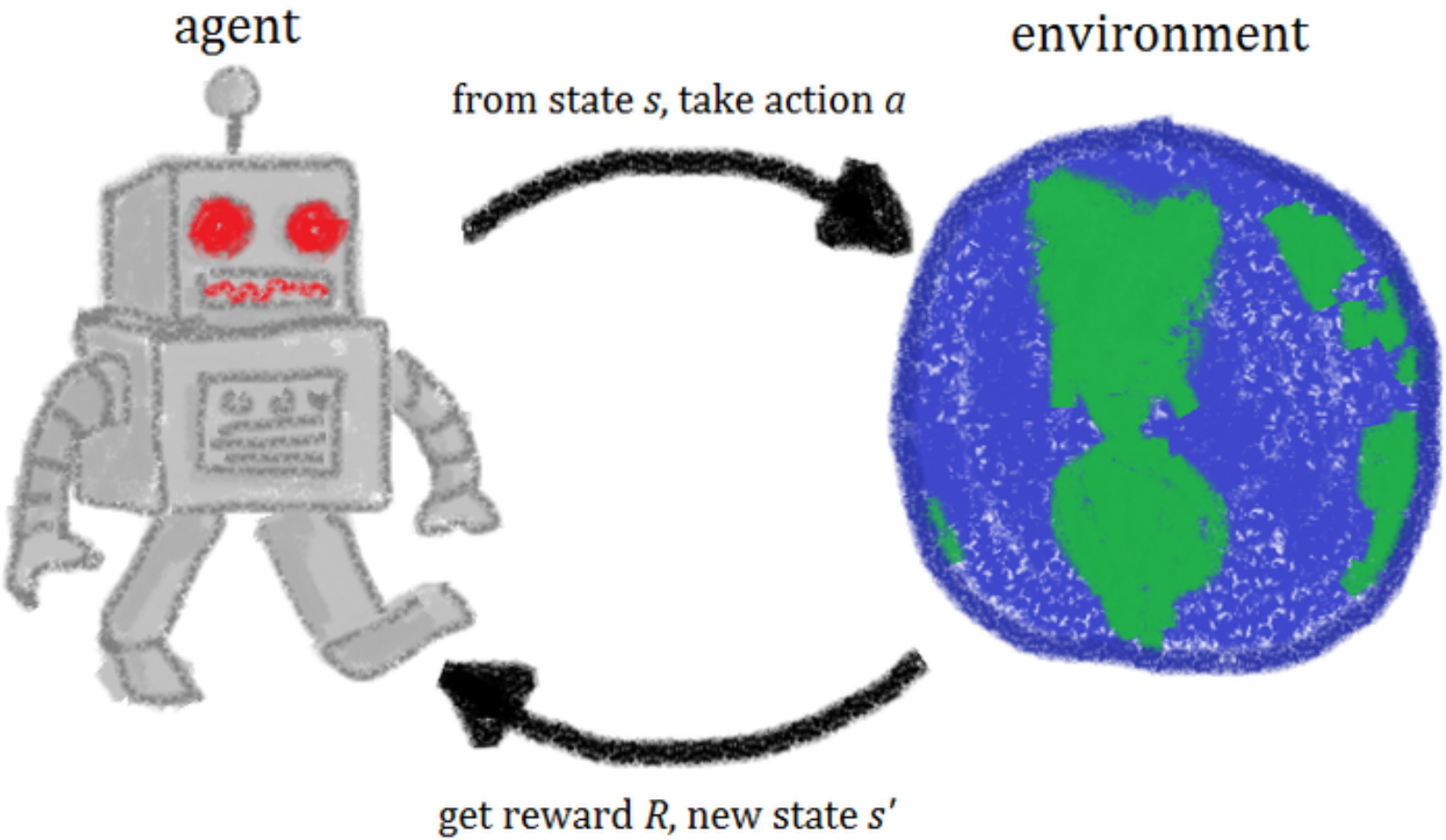
- Supervised Learning - learn to predict an output when given an input vector
- Reinforcement Learning - learn to select an action to maximize the expected sum of future rewards (payoff)
- Unsupervised Learning - discover a good internal representation of the input

# Supervised Learning

1

[1] Newell, Alejandro, Kaiyu Yang, and Jia Deng. "Stacked hourglass networks for human pose estimation." European Conference on Computer Vision. Springer, Cham, 2016.

# Reinforcement Learning




[2]

[2] Reference: Wikipedia
https://simple.wikipedia.org/wiki/Reinforcement_learning

# Unsupervised Learning

[3]

[3] https://skymind.ai/wiki/generative-adversarial-network-gan and https://github.com/robbiebarrat/art-DCGAN

# Two Types of Supervised Learning

- Classification: The program is asked to specify which of $k$ categories some input belongs to. The learning algorithm produces a function $f : \mathbb{R}^n \to \{1, ..., k\}$.
- Regression: The program is asked predict a numerical value given some input. The learning algorithm outputs a function $f : \mathbb{R}^n \to \mathbb{R}$.

# How Supervised Learning Typically Works

- Start by choosing a model-class: $\hat{y} = f(\boldsymbol{x}; \boldsymbol{W})$ where the model-class $f$ is a way of using some numerical parameters, $\boldsymbol{W}$, to map each input vector $\boldsymbol{x}$ to a predicted output $\hat{y}$.
- Learning means adjusting the parameters to reduce the discrepancy between the target output $y$ on each training case and the output $\hat{y}$, predicted by the model.

# Generative vs. Discriminative Models

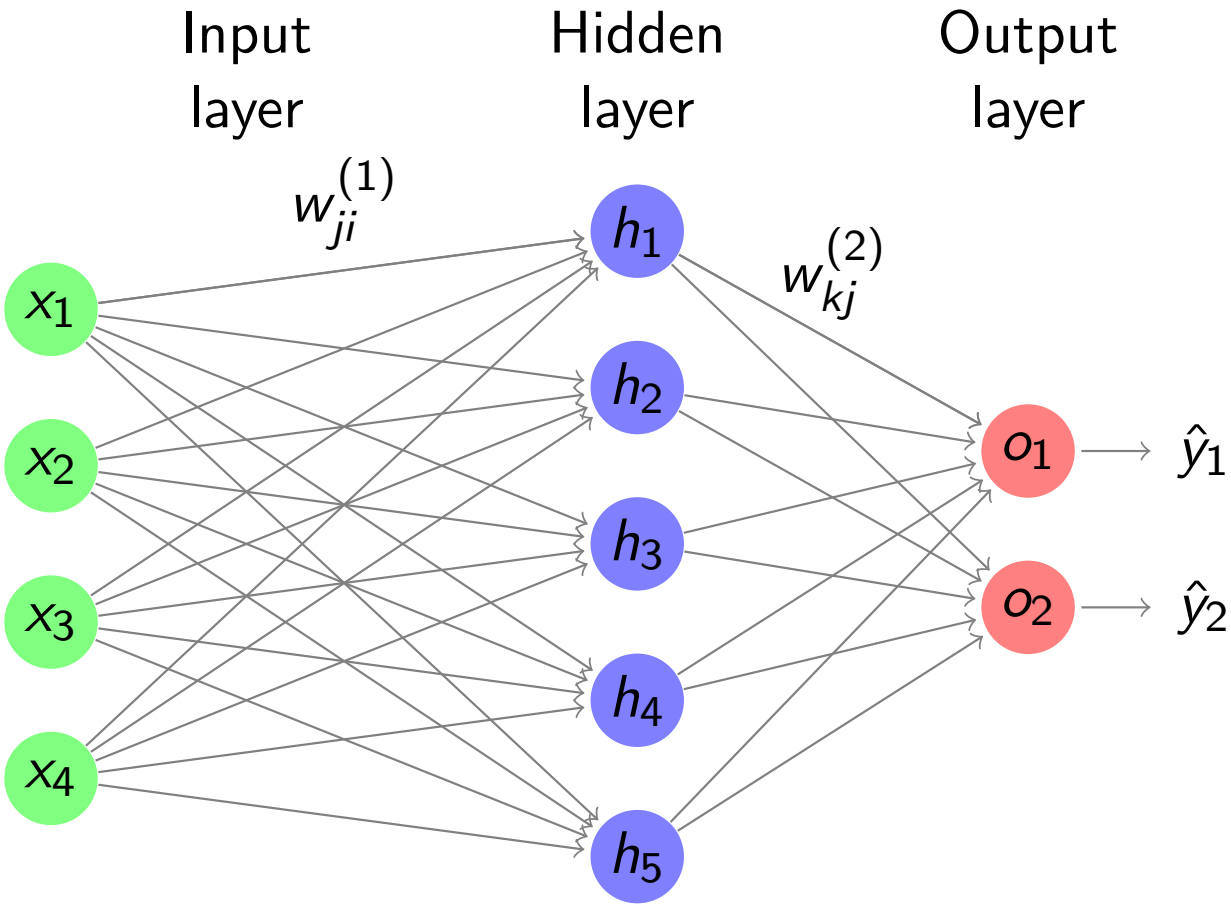
Generative vs. discriminative approaches to classification use different statistical modelling.

- Generative models model the distribution of individual classes. Given an observable variable $X$ and a target variable $Y$, a generative model is a statistical model that tries to model $P[X|Y]$ and $P[Y]$ in order to model the joint probability distribution $P[X, Y]$.
- Discriminative models learn the boundary between classes. A discriminative models is a model of the conditional probability of the target $Y$, given an observation $X$, $P[Y|X; W]$. [4]

Additional Reading
`http://cs229.stanford.edu/notes/cs229-notes2.pdf`

[4] Reference: `https://en.wikipedia.org/wiki/Generative_model`

# Multilayer Perceptron



# Review: Batch Gradient Descent

**begin initialize** $n_H$, $w$, $Th$, $\eta$, $m \leftarrow 0$, $r \leftarrow 0$
  **do** $r \leftarrow r + 1$ **(increment epoch)**
    $m \leftarrow 0$; $\Delta w \leftarrow 0$
    **do** $m \leftarrow m + 1$
      $x^m \leftarrow$ **selected pattern**
      $\Delta w \leftarrow \Delta w - \eta \frac{\partial J}{\partial w}$
    **until** $m = M$
    $w \leftarrow w + \Delta w$
  **until** $J(w) < Th$
**return** $w$
**end**

# Review: Stochastic Gradient Descent

**begin initialize** $n_H$, $w$, $Th$, $\eta$, $m = 0$, $r = 0$
  **do** $r \leftarrow r + 1$ **(increment epoch)**
    $m \leftarrow 0$
    **do** $m \leftarrow m + 1$
      $x^m \leftarrow$ **selected pattern (randomly)**
      $w \leftarrow w - \eta \frac{\partial J}{\partial w}$
    **until** $m = M$
  **until** $J(w) < Th$
**return** $w$
**end**

# The Cost Function (measure of discrepancy)

Recall from Foundations of Machine Learning:

- Mean Squared Error (MSE) for $M$ data points is given by $MSE = \frac{1}{2*M} \sum_{i=1}^{M} (\hat{y}_i - y_i)^2$
- $\frac{1}{2*M}$ just a constant so can be replaced by $\frac{1}{2}$ or $\frac{1}{M}$
- Other cost functions can be used as well, for example the KL divergence or Hellinger distance [5]
- MSE can be slow to learn, especially if the predictions are very far off the targets [6]
- Cross-Entropy Cost function is generally a better choice of cost function (discussed next)

[5] https://stats.stackexchange.com/questions/154879/a-list-of-cost-functions-used-in-neural-networks-alongside-applications

[6] http://neuralnetworksanddeeplearning.com/chap3.html

# Cross-Entropy Cost Function

- $J = -\frac{1}{M}\sum_x [y ln(\hat{y}) + (1-y) ln(1-\hat{y})]$
  where $M$ is the number of data points in the training set
- The cross-entropy cost function is non-negative, $J > 0$
- $J \approx 0$ when the prediction and targets are equal (i.e. $y = 0$ and $\hat{y} = 0$ or when $y = 1$ and $\hat{y} = 1$)
- $\frac{\partial J}{\partial w_{ij}}$ is proportional to the error in the output $(\hat{y} - y)$ and therefore, the larger the error, the faster the neuron will learn!

# Cross-Entropy Cost Function - What Does it Mean?

- The cross-entropy can be thought of as a **measure of surprise**.
- Given some input $x_i$, we can think of $\hat{y}_i$ as the estimated probability that $x_i$ belongs to class 1, and $1 - \hat{y}_i$ is the estimated probability that it belongs to class 0.
- Note the extreme case of infinite cross-entropy, if your model believes that a class has 0 probability of occurrence, and yet the class appears in the data, the "surprise" of your model will be infinitely great.

# General Cross-Entropy Cost Function for Neural Networks

$J = -\frac{1}{M}\sum_{i=1}^{M}\sum_{k=1}^{K}[y_k^{(i)} ln(\hat{y}_k^{(i)}) + (1 - y_k^{(i)}) ln(1 - \hat{y}_k^{(i)})]$
where $M$ is the number of data points in the training set and $K$ is the number of classes of the classification problem, $\hat{y} \in \mathbb{R}^K$

# Softmax

The softmax is an activation function used at the output layer of a neural network that forces the outputs to sum to 1 so that they can represent a probability distribution across a discrete mutually exclusive alternatives.

$y_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_j}}$, for $j = 1, 2, ..., K$

Note $\frac{\partial y_i}{z_i} = y_i(1 - y_i)$

- The output of a sofmax layer is a set of positive numbers which sum up to 1 and can be thought of as a probability distribution

# Softmax

Exercise: Construct an example showing explicitly that in a network with a sigmoid output layer, the output activations won't always sum to 1.

# A Quick Introduction to Tensors

A tensor, $T$, is a generalization of vectors and matrices, which can be thought of as a multidimensional array.

- A $0D$ tensor is a scalar
- A $1D$ tensor is a vector
- A $2D$ tensor is a matrix
- A $3D$ tensor can be thought of as a vector of identically sized matrices
- A $4D$ tensor can be thought of as a matrix of identically sized matrices or a sequence of $3D$ tensors
- etc.

# A Quick Introduction to Tensors

- A tensor is a container which can house data in $N$ dimensions. Often and erroneously used interchangeably with the matrix (which is specifically a 2-dimensional tensor), tensors are generalizations of matrices to N-dimensional space.
- Tensors are used to encode the data as well as the model parameters
- Data manipulation via tensors enables CPUs and GPUs to run at peak performance

# A Quick Introduction to Tensors

Link to PyTorch Tensor Basics `https://gist.github.com/kfarrahi/f8f51824b23252c81d5aaec461365e73`