11.Review of Machine Learning Methods and Applications

M.A.Z. Chowdhury and M.A. Oehlschlaeger

Department of Mechanical, Aerospace and Nuclear Engineering
Rensselaer Polytechnic Institute, Troy, New York

chowdm@rpi.edu and oehlsm@rpi.edu

"A review of the first ten lectures for the mid-term exam."

MANE 4962 and 6962

Outline: Five methods & three tasks

Machine learning methods:

- kNN
- k-means
- Linear regression
- Logistic regression
- Neural Networks

Machine learning tasks:

- Classification
- Regression
- Clustering

Inputs, outputs, and mathematics of ML

Inputs

- The input is typically an n-dimensional vector which quantify what we know about the problem. The components of the vector are termed features. A set of features are a representation of the data.

Outputs

- The output is a dependent variable, commonly known as a target.
- Targets are discrete variables, if we want to identify/categorize/classify something. (classification task)
- Targets are continuous variables, if we want to predict/regress/estimate something. (regression task)

Linear algebra

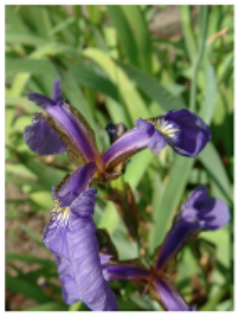
- Vector and matrix algebra

Optimization

- Gradient Descent and its variants

k-Nearest Neighbors (kNN)

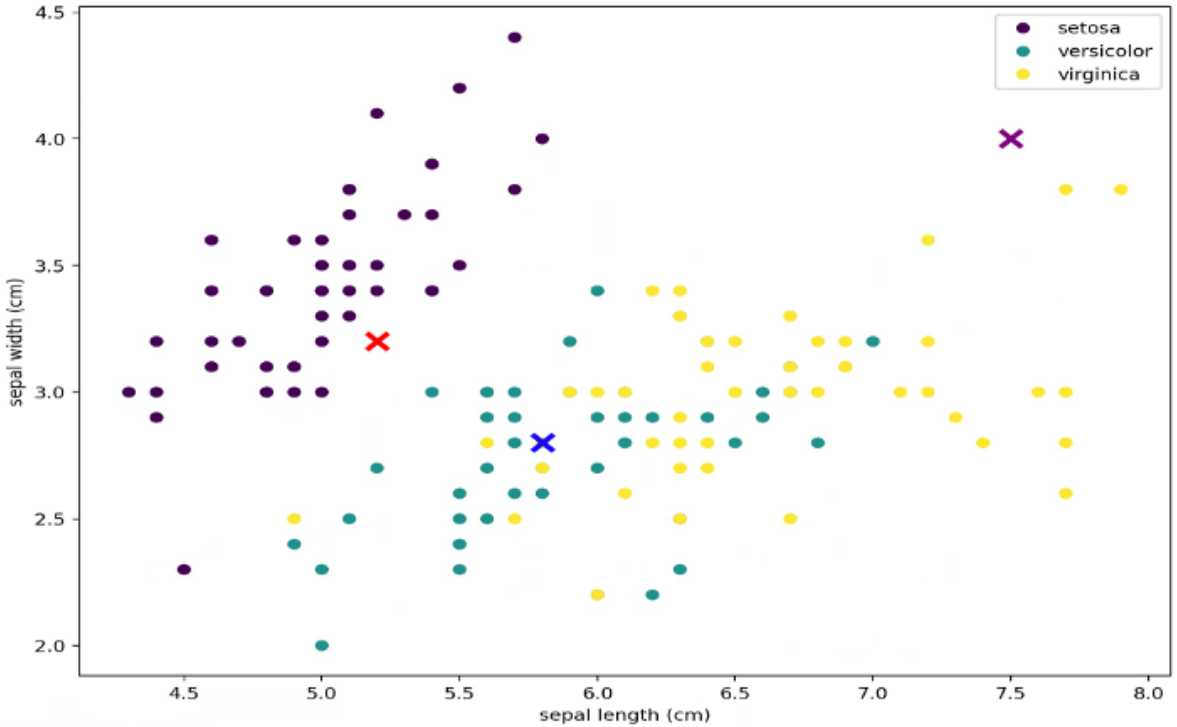
1. Find all the nearest neighbors using a distance metric, $d(\underline{x}, \underline{x}^{'}) = ||\underline{x} - \underline{x}^{'}||_p$
2. For classification, return the average/majority class as the class label.
3. For regression, return local interpolation of the targets of the nearest neighbors.



Iris setosa

Iris versicolor

Iris virginica

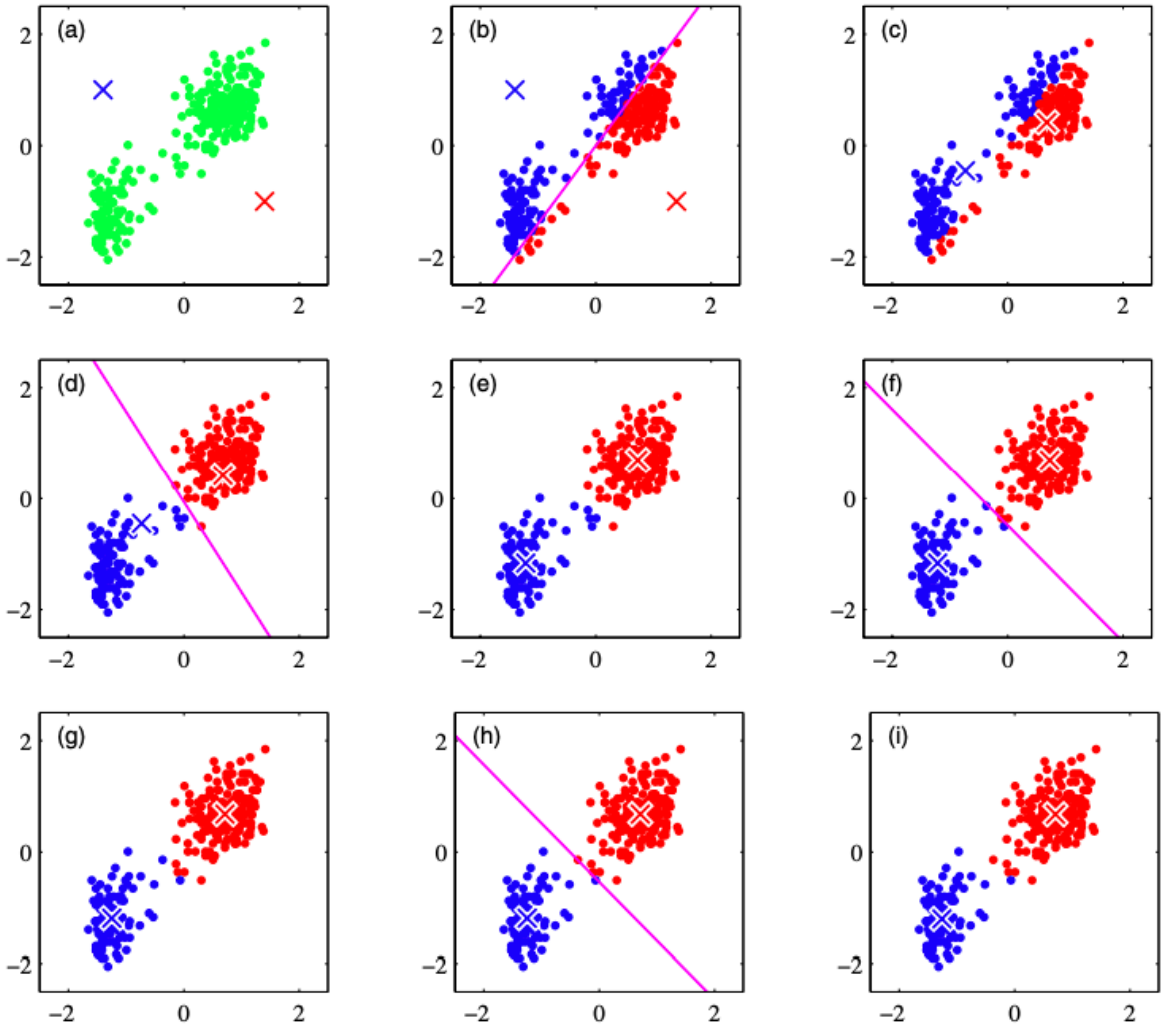


A simple implementation of K-NN

```
class My_KNNClassifier:
    def __init__(self, k=3):
        self.k = k
    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train
    def predict(self, X_test):
        predictions = []
        for i in range(X_test.shape[0]):
            predictions.append(self._knn_classifier(X_test[i]))
        return predictions
    def _knn_classifier(self, X_test):
        distances, targets = [], []
        for i in range(self.X_train.shape[0]):
            distance = np.linalg.norm(self.X_train[i]-X_test)
            distances.append([distance, i])
        distances = sorted(distances)
        for i in range(self.k):
            index = distances[i][1]
            targets.append(self.y_train[index])
        return max(targets, key=targets.count)
model = My_KNNClassifier()
model.fit(X_train, y_train)
preds = model.predict(X_test)
print(accuracy_score(y_test, preds))
```

Check notebook for Lecture 4.

k-means clustering



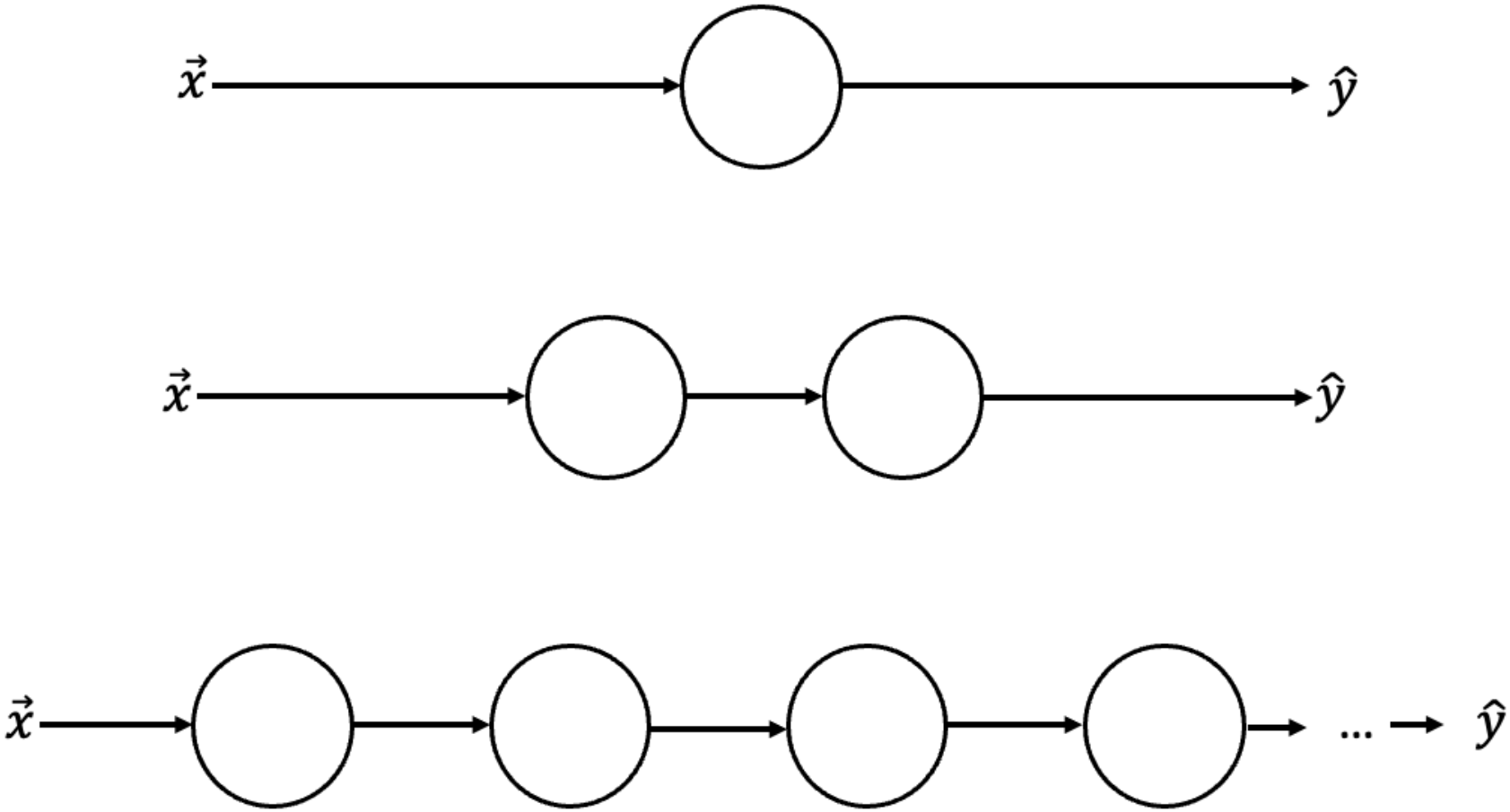
Clustering is not the same as classification so do not use accuracy to evaluate clustering. Use silhouette scores.

A simple implementation of k-means

```python
class KMeans:
    def __init__(self, k):
        self.k = k
        self.cluster_labels = None
    def fit(self, X):
        self.centroids = X[np.random.choice(X.shape[0], self.k, replace=False), :]
        self.cluster_labels = np.arange(self.k)
        while True:
            distances = np.array([np.linalg.norm(X - centroid, axis=1) for centroid in self.centroids])
            self.clusters = np.argmin(distances, axis=0)
            new_centroids = np.array([X[self.clusters == i, :].mean(axis=0) for i in range(self.k)])
            # check convergence
            if np.array_equal(new_centroids, self.centroids):
                break
            else:
                self.centroids = new_centroids
    def predict(self, X):
        distances = np.array([np.linalg.norm(X - centroid, axis=1) for centroid in self.centroids])
        return self.cluster_labels[np.argmin(distances, axis=0)]
```

Check notebook for Lecture 5.

Linear Models and NN



Linear Hypothesis

Consider, m data points in the table,

predict y from a single input feature x

input	target	prediction
x_1	y_1	$\hat{y}_1$
x_2	y_2	$\hat{y}_2$
$\vdots$	$\vdots$	$\vdots$
x_m	y_m	$\hat{y}_m$

- we want to approximate target function $y = f(\underline{x}) = f(x_1, x_2, \ldots, x_n)$
- $\hat{y} = h(\underline{x}; \underline{w}) = w_0 + w_1 x_1$ [for $n = 1$, single feature, we have one input variable]
- x_1 is the input variable/feature not the data point in the equation.
- w's are called model parameters (weights), n is the number of feature
- Model parameters and hyperparameters are not same.
- Model parameters depend on training data, hyperparameters are chosen, or set, or optimized.

Cost function

How to train a linear model?

$\rightarrow$ estimate mistakes and correct them via error measure.

It measures the total amount of incorrect predictions across the data points.

- squared error (loss), $\mathrm{L} = (y - \hat{y})^2$
- mean squared error, $mse = \frac{1}{m}\sum_i (y_i - \hat{y}_i)^2$
- cost function, $J = \frac{1}{2m}\sum_i (y_i - \hat{y}_i)^2$

Gradient Descent to Minimize J

$$J = \frac{1}{2m}\sum_i (y_i - w_0 - w_1 x_i)^2$$

$$\implies \frac{\partial J}{\partial w_0} = \frac{1}{2m}\sum_i \frac{\partial}{\partial w_0}(y_i - w_0 - w_1 x_i)^2$$

$$\implies \frac{\partial J}{\partial w_0} = \frac{1}{2m}\sum_i \{2(y_i - w_0 - w_1 x_i)\dot{(}0 - 1 - 0)\}$$

$$\implies \frac{\partial J}{\partial w_0} = -\frac{1}{m}\sum_i \{y_i - (w_0 + w_1 x_i)\}$$

$$\implies \frac{\partial J}{\partial w_0} = -\frac{1}{m}\sum_i (y_i - \hat{y}_i)$$

$$\therefore \frac{\partial J}{\partial w_0} = -\frac{1}{m}\sum_i (y_i - \hat{y}_i) = -\text{ average error}$$

Gradient Descent to Minimize J

$$J = \frac{1}{2m}\sum_i (y_i - w_0 - w_1 x_i)^2$$

$$\implies \frac{\partial J}{\partial w_1} = \frac{1}{2m}\sum_i \frac{\partial}{\partial w_1}(y_i - w_0 - w_1 x_i)^2$$

$$\implies \frac{\partial J}{\partial w_1} = \frac{1}{2m}\sum_i \{2(y_i - w_0 - w_1 x_i)\dot{(}0 - 0 - x_i)\}$$

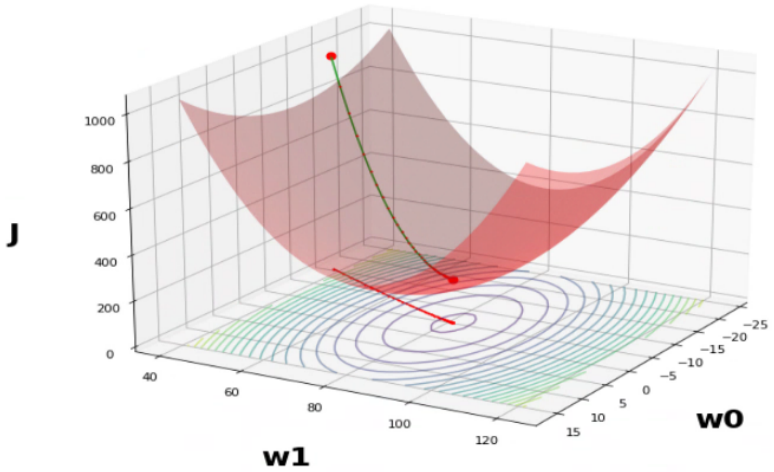
$$\implies \frac{\partial J}{\partial w_1} = -\frac{1}{m}\sum_i \{y_i - (w_0 + w_1 x_i)\}x_i$$

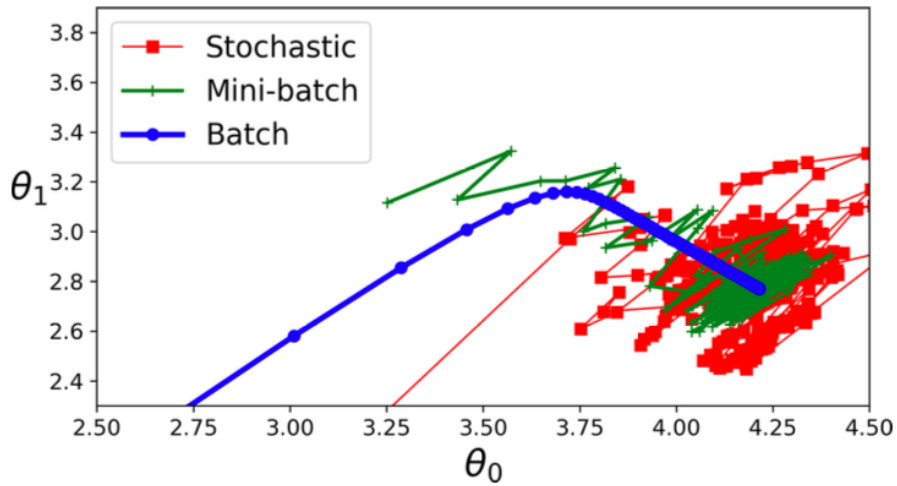
$$\implies \frac{\partial J}{\partial w_1} = -\frac{1}{m}\sum_i (y_i - \hat{y}_i)x_i$$

$$\therefore \frac{\partial J}{\partial w_1} = -\frac{1}{m}\sum_i (y_i - \hat{y}_i)x_i = -\text{ average error } \times \text{input}$$

Comparison of Gradient Descent Methods

- $w_{j+1} = w_j - \alpha \frac{\partial J}{\partial w_j}$
- Full batch, mini-batch, stochastic (SGDRegressor in Scikit).
- Works very well with large number of features with scaling.
- BGD with good learning schedule vs SGD vs mBGD, which would be better?
- Alternatives: Momentum, NAG, AdaGrad, RMSProp, Adam, AdaMax, Nadam.





Logistic Regression

Consider, m data points.
Predict y from input variables or features x_1, x_2

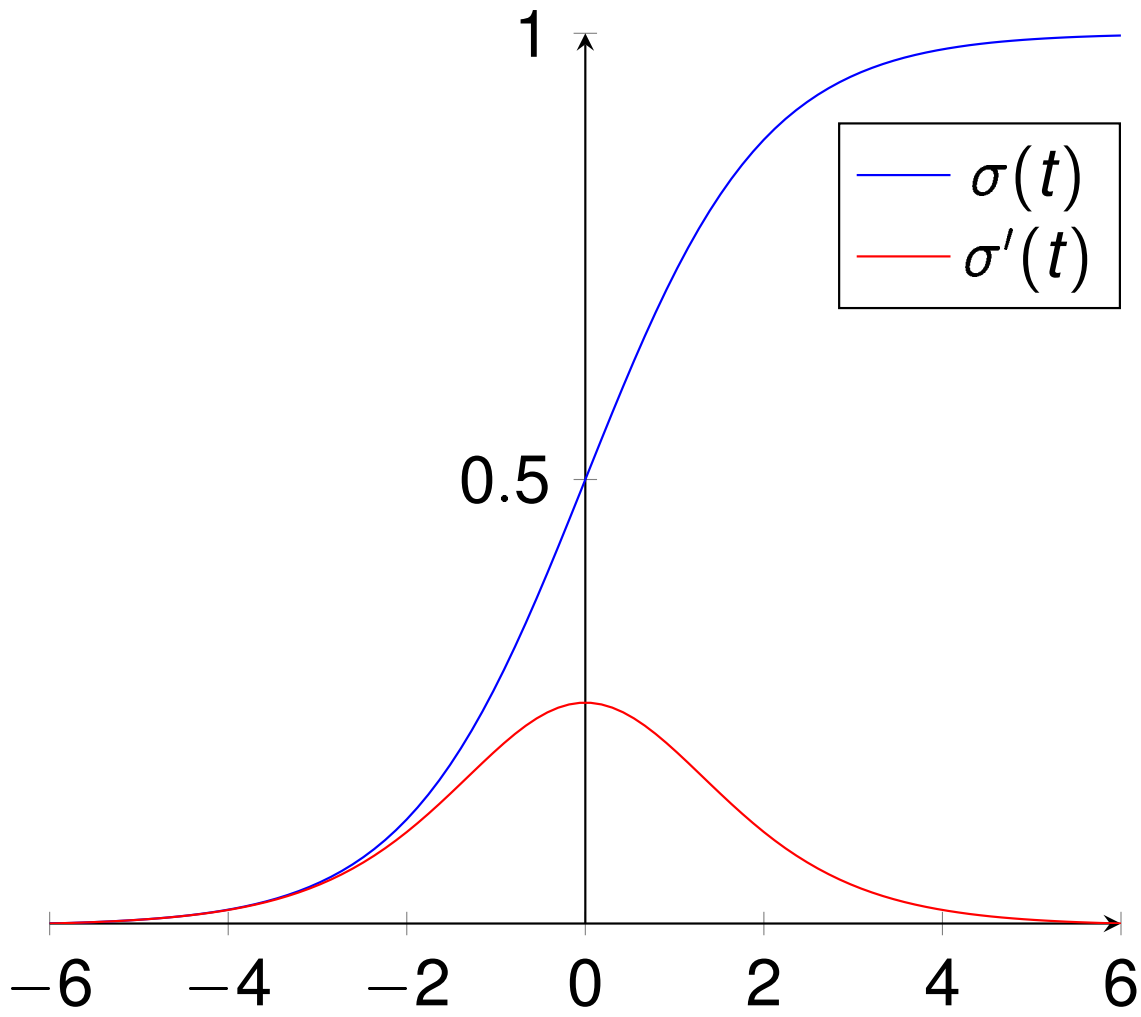
example	input $\underline{x} = (x_1, x_2)$	output y	prediction $\hat{y}$
1	$x_1^{(1)}, x_2^{(1)}$	y_1 = 0 or 1	$\hat{y_1} = [0, 1]$
2	$x_1^{(2)}, x_2^{(2)}$	y_2 = 0 or 1	$\hat{y_2} = [0, 1]$
⋮	⋮	⋮	⋮
m	$x_1^{(m)}, x_2^{(m)}$	y_m = 0 or 1	$\hat{y_m} = [0, 1]$

Logistic Regression

- $\hat{p} = h(\underline{x}; \underline{w}) = \sigma(w_0 + w_1 x_1 + w_2 x_2)$
- $z = w_0 + w_1 x_1 + w_2 x_2$
- $\sigma(t) = \frac{1}{1+e^{-t}}$
- $\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5 \\ 1 & \text{if } \hat{p} \geq 0.5 \end{cases}$
- It handles extreme feature values quite well.

Cost function:, $J = -\frac{1}{m} \sum_i [y_i ln\hat{y}_i + (1 - y_i) ln(1 - \hat{y}_i)]$

Sigmoid function



$\sigma'(t) = \sigma(t)[1 - \sigma(t)]$ [Check it]

Cost function

How to train a logistic model for classification?

$\rightarrow$ "Squish" outputs between 0 and 1

$\rightarrow$ Binary classification

Least squares, absolute, and/or squared error are not good cost functions for classification because they do not penalize the model enough.

- cost function, $J = -\frac{1}{m}\sum_i [y_i ln\hat{y}_i + (1 - y_i) ln(1 - \hat{y}_i)]$
- If $y_i = \hat{y}_i$, for all data points then $J = 0$.
- If $y_i \neq \hat{y}_i$, for some data points then $J \rightarrow \infty$
- J should be positive, differentiable, and continuous.

Outputs of this cost function

Let's consider a single data point,

- $J = -[y ln \hat{y} + (1 - y) ln(1 - \hat{y})]$

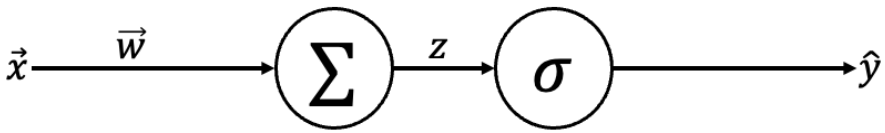
Correct classification

- $J \approx 0$, when $y = 0$ and prediction is $\hat{y} \approx 0$
- $J \approx 0$, when $y = 1$ and prediction is $\hat{y} \approx 1$

Incorrect classification

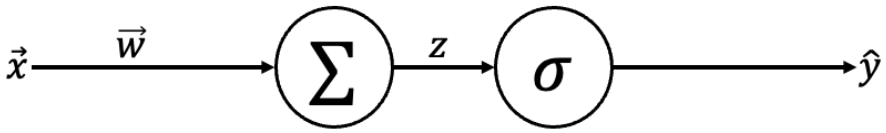
- $J \to \infty$, when $y = 0$ and prediction is $\hat{y} \approx 1$
- $J \to \infty$, when $y = 1$ and prediction is $\hat{y} \approx 0$

Derivative of log-loss cost function



$$\frac{\partial J}{\partial w} = ?$$

$$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w}$$



Consider, the binary cross-entropy loss for a single data point,

$$J = -[y ln\hat{y} + (1-y) ln(1-\hat{y})]$$

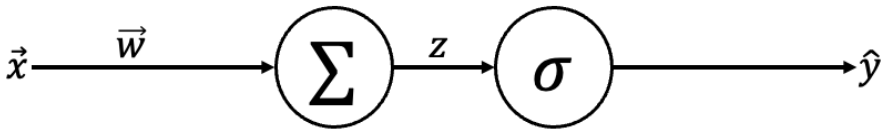
$$\implies \frac{\partial J}{\partial \hat{y}} = -[\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}]$$

$$\frac{\partial \hat{y}}{\partial z} = \frac{\partial \sigma(z)}{\partial z}$$

$$\implies \frac{\partial \hat{y}}{\partial z} = \sigma^{'}(z)$$

$$\implies \frac{\partial \hat{y}}{\partial z} = \sigma(z)[(1-\sigma(z))]$$

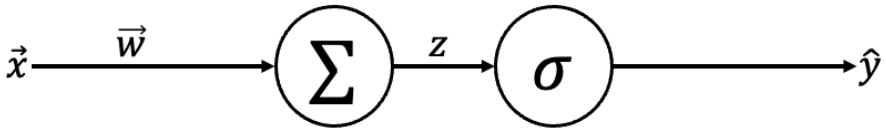
$$\implies \frac{\partial \hat{y}}{\partial z} = \hat{y}(1-\hat{y})$$



$$\frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} = -[\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}]\hat{y}(1-\hat{y})$$

$$\frac{\partial J}{\partial z} = -(\frac{y-y\hat{y}-\hat{y}+y\hat{y}}{\hat{y}(1-\hat{y})})\hat{y}(1-\hat{y})$$

$$\frac{\partial J}{\partial z} = -(y-\hat{y}) = -\textbf{error}$$



$$\frac{\partial z}{\partial w_1} = \frac{\partial}{\partial w_1}(w_0 + w_1 x_1) = x_1 = \textbf{input} \quad \text{[single feature]}$$

We can check the above for more than one features and write generally as

$$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w} = -(y - \hat{y})x = \textbf{error} \times \textbf{input}$$

Considering m data points we obtain,

$$\frac{\partial J}{\partial w_j} = -\frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i) x_j$$

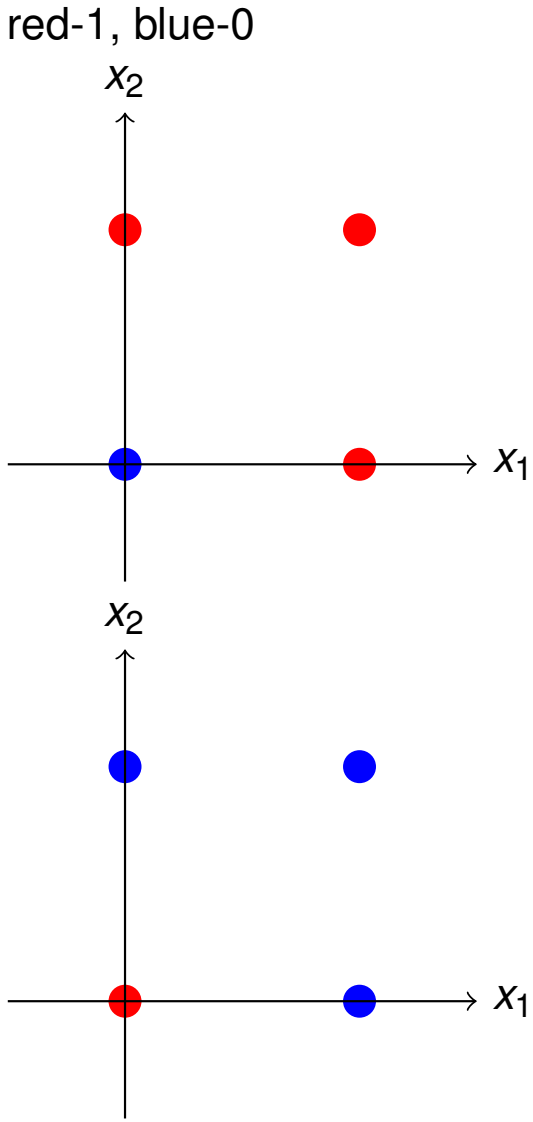
OR Gate and NOR Gate

input $\underline{x} = (x_1, x_2)$	output y	prediction
(0,0)	0	< 0.5
(0,1)	1	≥ 0.5
(1,0)	1	≥ 0.5
(1,1)	1	≥ 0.5

OR Gate: $w_0 = -1$ $w_1 = 2$ $w_2 = 2$

input $\underline{x} = (x_1, x_2)$	output y	prediction
(0,0)	1	≥ 0.5
(0,1)	0	< 0.5
(1,0)	0	< 0.5
(1,1)	0	< 0.5

NOR Gate: $w_0 = 1$ $w_1 = -2$ $w_2 = -2$
Note: w's are not guaranteed to be unique



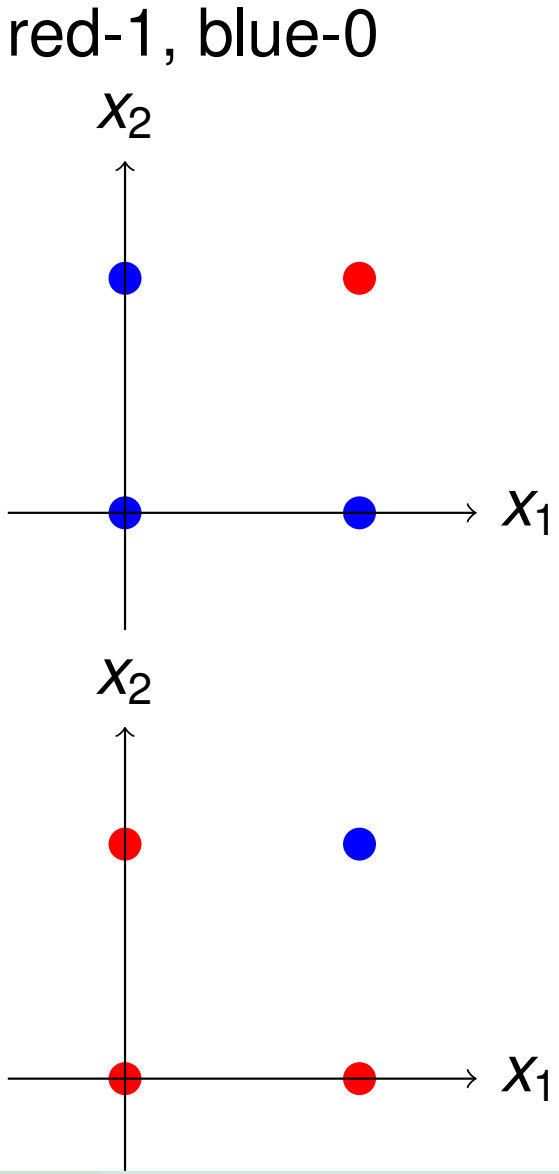
AND Gate and NAND Gate

input $\underline{x} = (x_1, x_2)$	output y	prediction
(0,0)	0	< 0.5
(0,1)	0	< 0.5
(1,0)	0	< 0.5
(1,1)	1	≥ 0.5

AND Gate: $w_0 = -3$ $w_1 = 2$ $w_2 = 2$

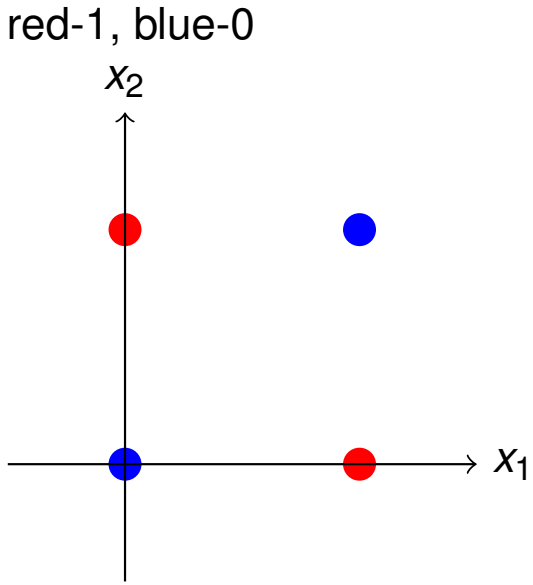
input $\underline{x} = (x_1, x_2)$	output y	prediction
(0,0)	1	≥ 0.5
(0,1)	1	< 0.5
(1,0)	1	< 0.5
(1,1)	0	< 0.5

NAND Gate: $w_0 = 3$ $w_1 = -2$ $w_2 = -2$
Note: w's are not guaranteed to be unique



XOR Gate

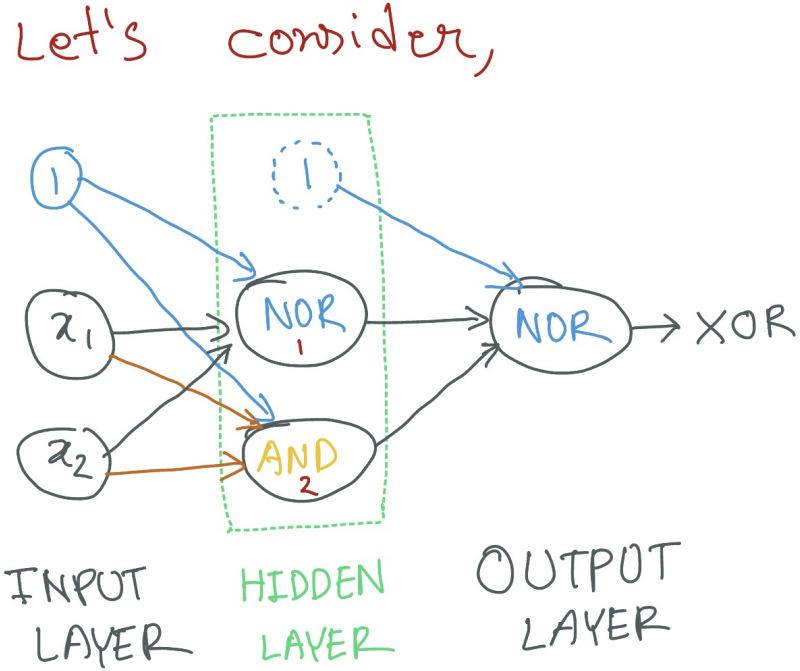
input $\underline{x} = (x_1, x_2)$	output y	prediction
(0,0)	0	×
(0,1)	1	×
(1,0)	1	×
(1,1)	0	×



XOR Gate: No solution from logistic regression!

- ☞ Use non-linear feature transformation. (The kernel trick) SVMs.
- ☞ Add extra layers. (Deep neural nets)

Solving XOR Gate



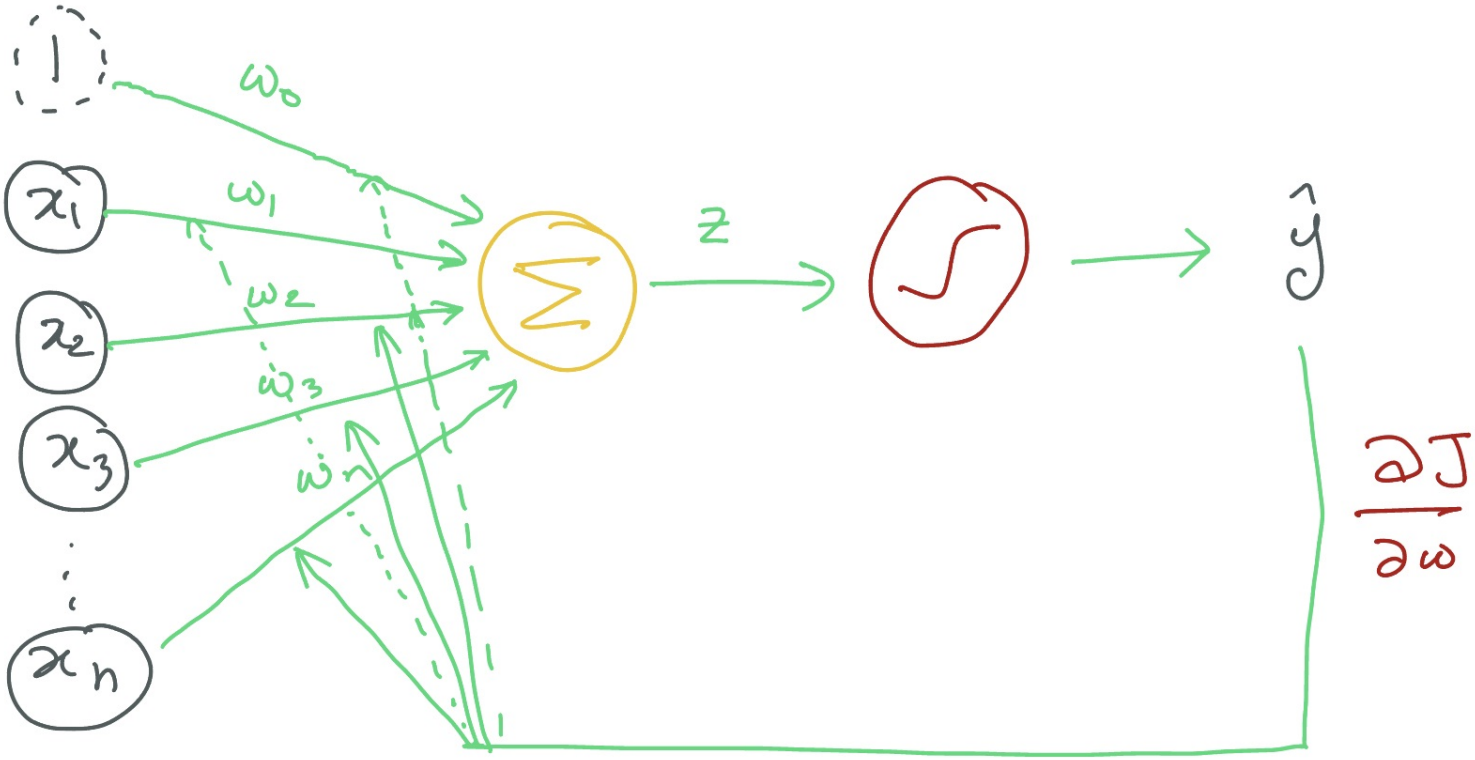
$\underline{x}$ (x_1, x_2)	(h_1, h_2) = (NOR, AND)	NOR(h_1, h_2) = XOR
(0,0)	1, 0	0
(1,0)	0, 0	1
(0,1)	0, 0	1
(1,1)	0, 1	0

First NOR: $w_{01}^{(1)} = 1\ w_{11}^{(1)} = -2\ w_{21}^{(1)} = -2$

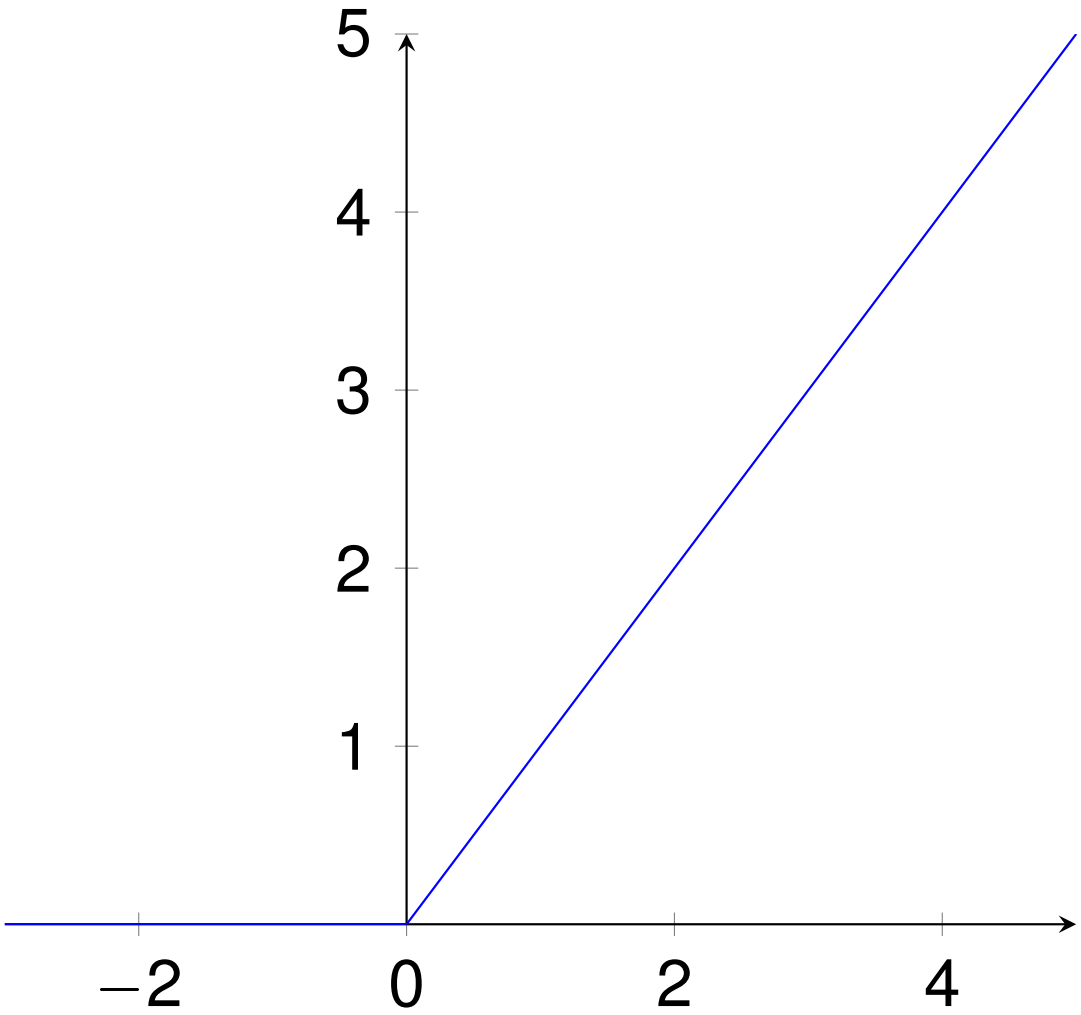
AND Gate: $w_{02}^{(1)} = -3\ w_{12}^{(1)} = 2\ w_{22}^{(1)} = 2$

Last NOR : $w_{01}^{(2)} = 1\ w_{11}^{(2)} = -2\ w_{21}^{(2)} = -2$

Moving beyond gates (More than 2 features)

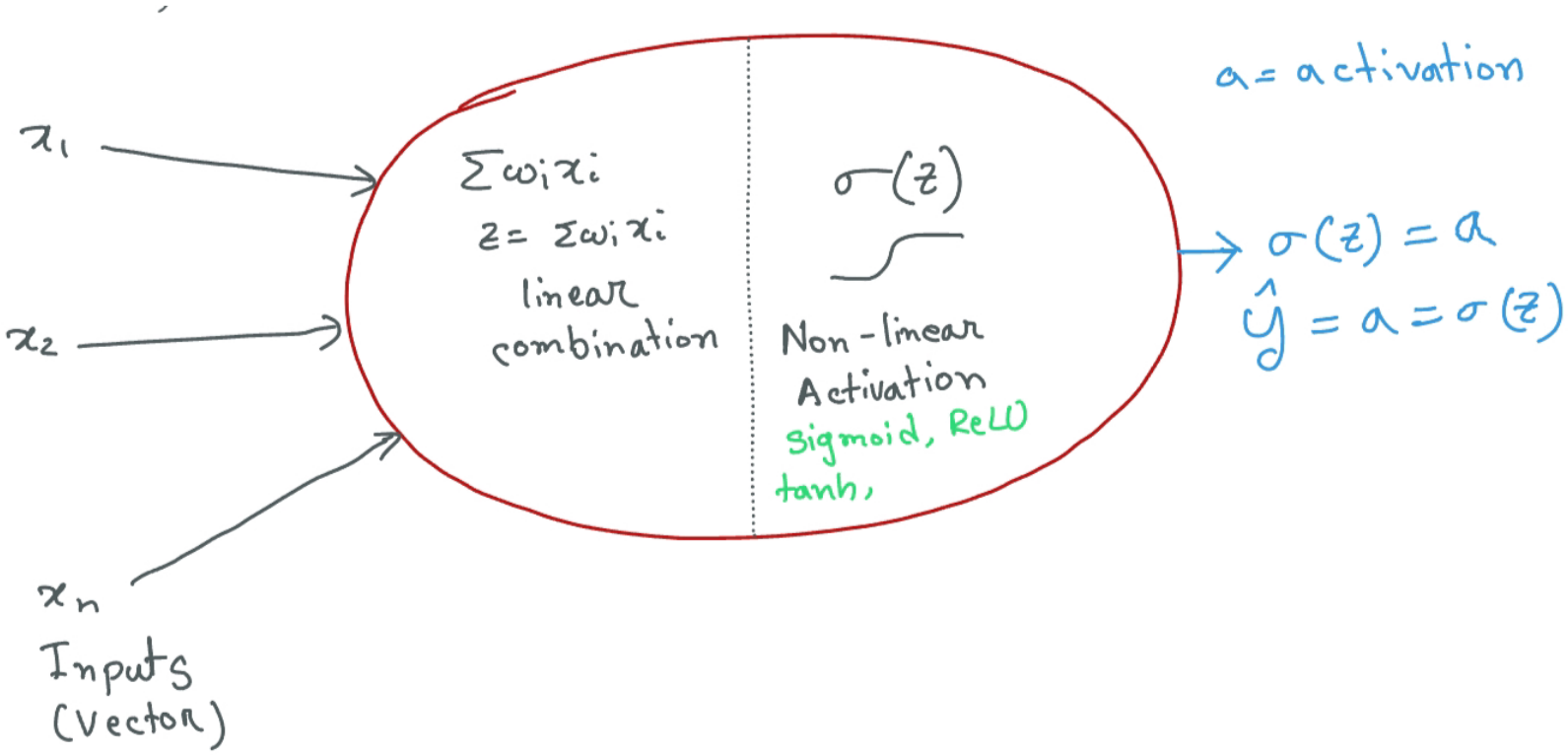


Rectified Linear Unit

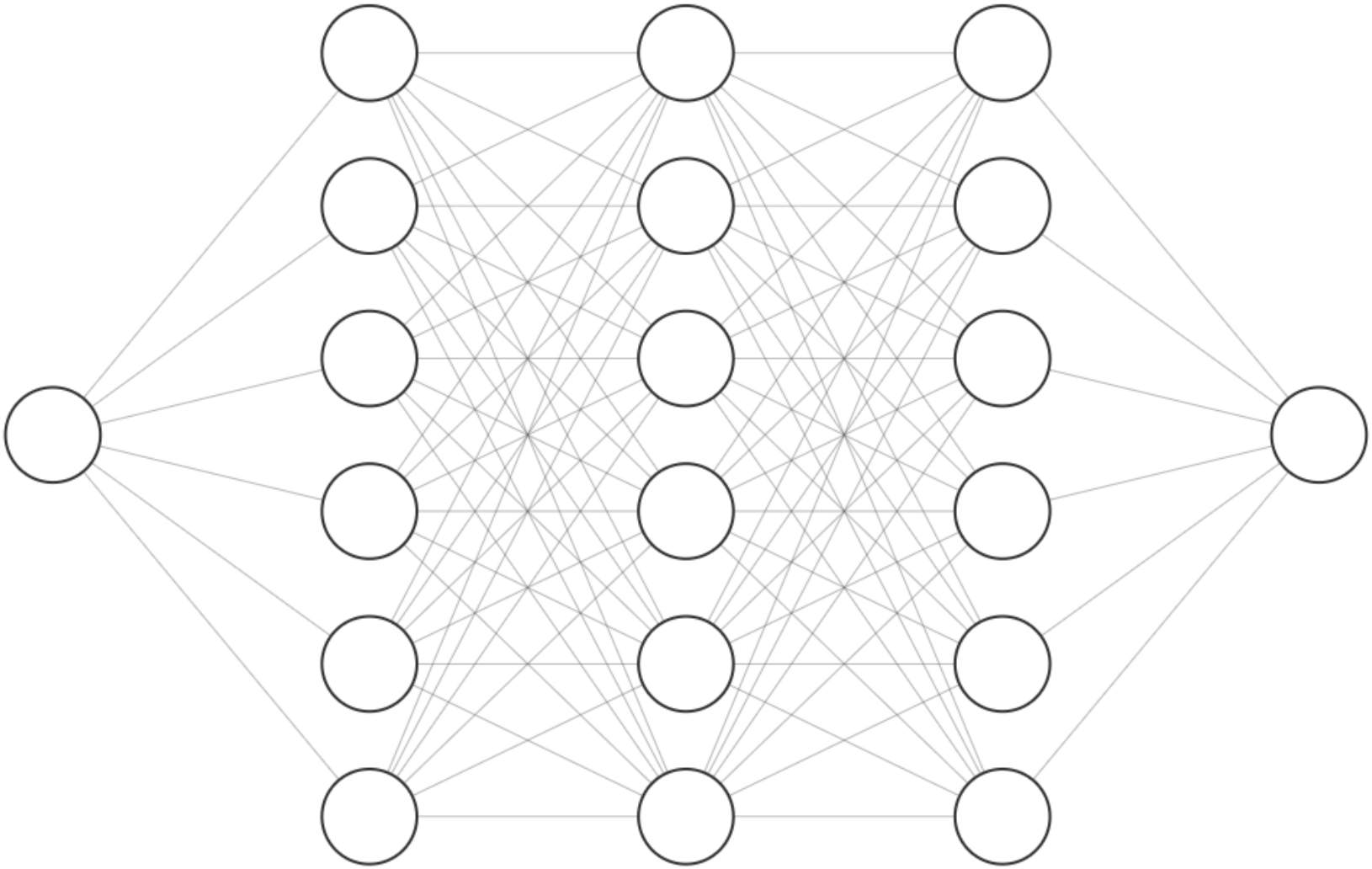


$ReLU(t) = max(0, t)$

Artificial Neuron



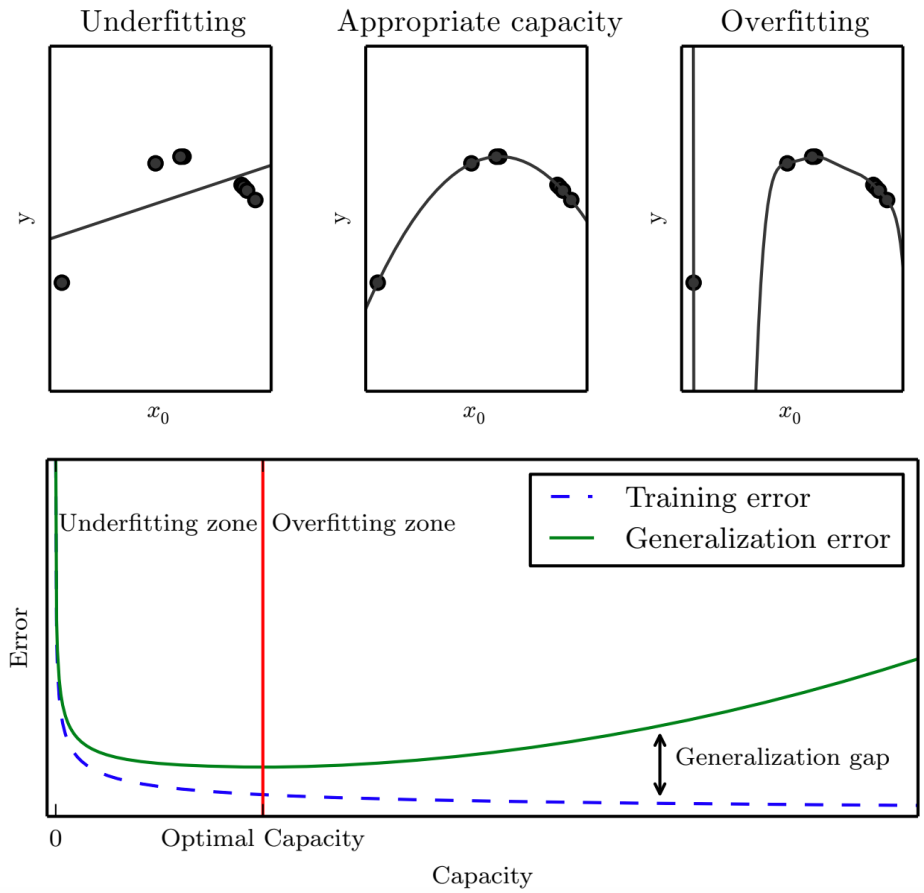
A fully connected deep ANN



A fully connected deep ANN

```
model = Sequential()
model.add(Dense(6, input_dim=1,
        activation='relu'))
model.add(Dense(6,activation='relu'))
model.add(Dense(6,activation='relu'))
model.add(Dense(1))
opt = optimizers.Adam(learning_rate=0.001)
mse = tf.keras.losses.MeanSquaredError(
    reduction=tf.keras.losses.Reduction.SUM)
model.compile(loss=mse, optimizer=opt)
model.fit(X,y,epochs=2000,batch_size=10, verbose=0)
predictions = model.predict(X)
```

Analyzing the learning process



Overfitting, underfitting, capacity, bias, and variance

Regularization

- To regularize a model, means to constrain it and not make it too complex.
- **Ridge:** Force model to choose smaller weights. $\lambda \sum |w_i|^2$ term is added to cost function. λ is the strength of regularization, a hyperparameter.
- **Lasso:** Make weights of less important features zero. $\lambda \sum |w_i|$.
- **ElasticNet:** $r\lambda \sum |w_i| + \frac{1-r}{2}\lambda \sum |w_i|^2$ term is added to cost function, r is the mix ratio.
- **Early stopping**: Stop training as soon as the validation error reaches a minimum.
- Scikit implements regularization for SGDRegressor, logistic regression and many other methods. Read the documentation carefully.

Testing and Validating

- Some data is kept away so the model can generalize on data it has never seen.
- multiple sets - commonly training data, testing data and validation data.
- Other splits performed for finding hyperparamters or stricter evaluation
- kfoldCV, RFECV, LOOCV etc.
- Hyperparameters are tuned with CV or eshtablished from heuristics and experience
- Changing a hyperparamter will yield a new model/hypothesis.

- **No Free Lunch Theorem:** If you make absolutely no assumption about the data, then there is no reason to prefer one model over any other. (Wolpert and Macready, 1996)
- No model is a priori guaranteed to work better.
- **Universal approximation theorem:** One hidden layer is enough to represent (not learn) an approximation of any function to an arbitrary degree of accuracy. (Goodfellow et al.)

Metrics: Evaluating ML models

For classification

- Confusion matrix
- Classification accuracy (CA)
- Precision (P), Recall (R), F1 score
- Precision-recall curve
- Receiver operating characteristic (ROC) curve

For regression

- Coefficient of determination
- MAE
- MSE
- Plots of actual vs predicted values, or residuals vs predicted values are also useful. The coefficient of determination is useful for linear regression.

For clustering

- Elbow plot
- Silhouette scores