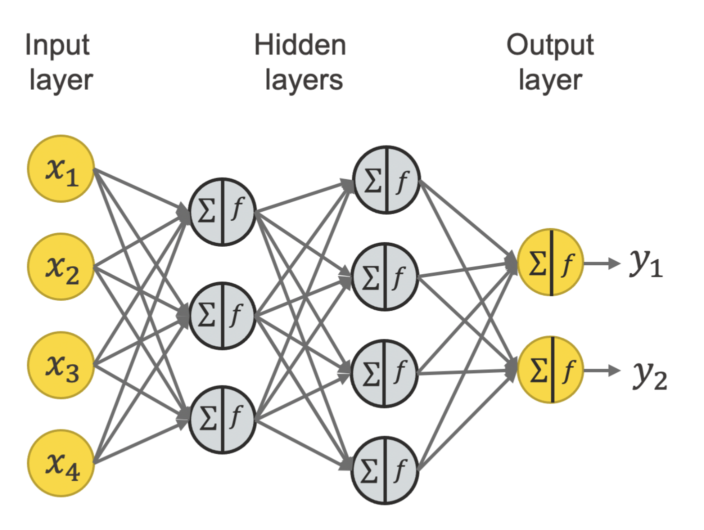


Feedforward Neural Networks

Based on material by Stuart Russel,
Peter Norvig. Slides mainly based on sklearn documentation

Some slides courtesy of Ava Soleimany (MIT) and
Fei-Fei Li, Justin Johnson and Serena Yeung (Standford)

Last modified on 2022/05/07 by f.maire@qut.edu.au

Advanced Reference for Deep Learning

Postgraduate textbook: **Deep Learning**

by Ian Goodfellow, Yoshua Bengio and Aaron Courville

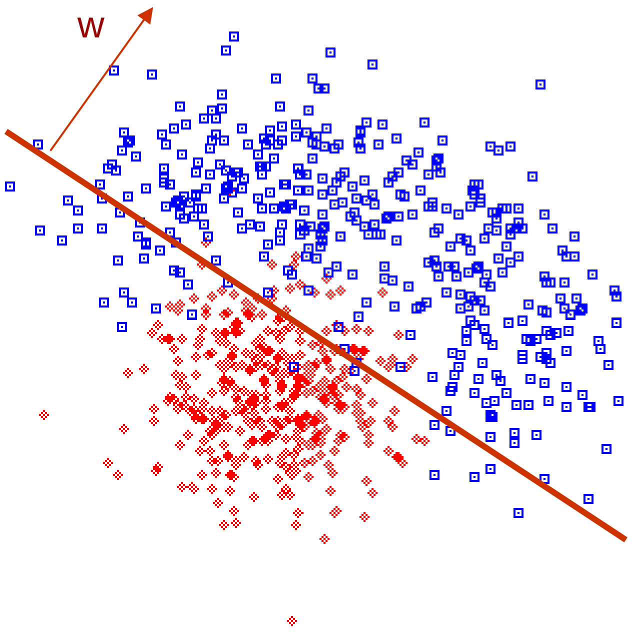
http://www.deeplearningbook.org/

This book provides a sound mathematical treatment of deep learning

Outline

- Revisiting single layer linear neural networks
- The need for multi-layer neural networks
- Representing union of polyhedral regions of the input space
- Core idea for computing the parameters of a feedforward neural network
- Two Python libraries for feedforward networks
  - Sklearn and TensorFlow/Keras

Revisiting Linear Classifiers



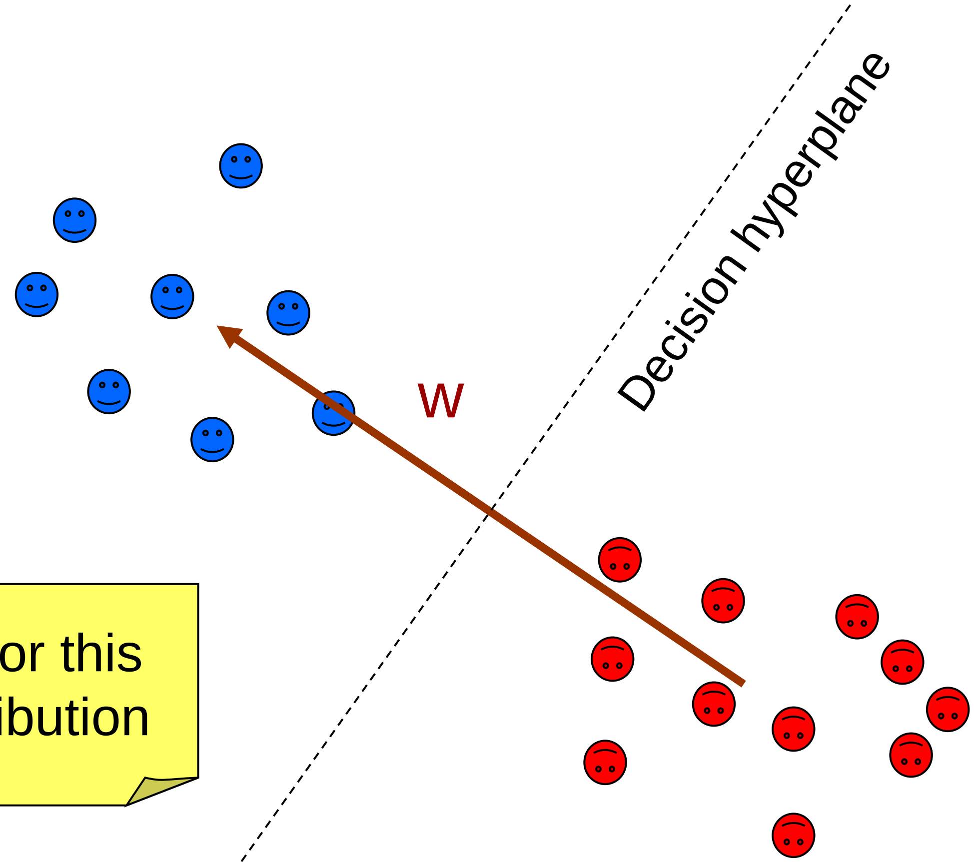
Given two classes positive and negative,
we want to find an hyperplane that minimizes
the misclassification error

sign of $w*x+b$

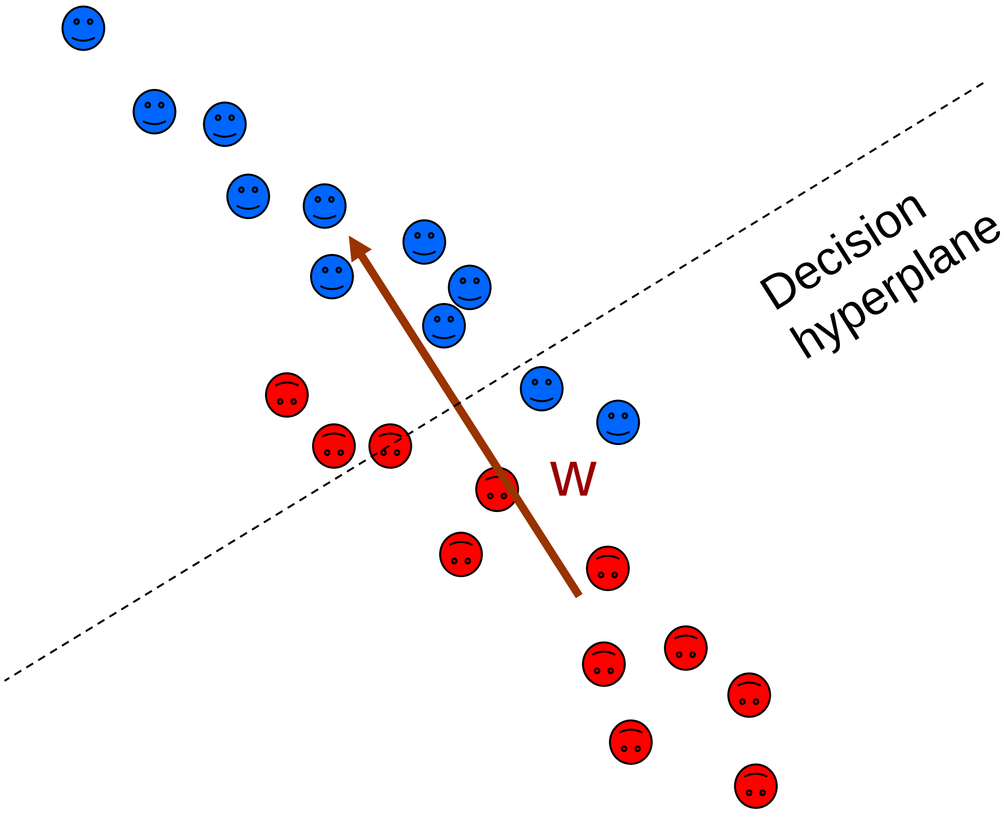
where

x is an input vector
w is the normal vector to the hyperplane
b is a scalar

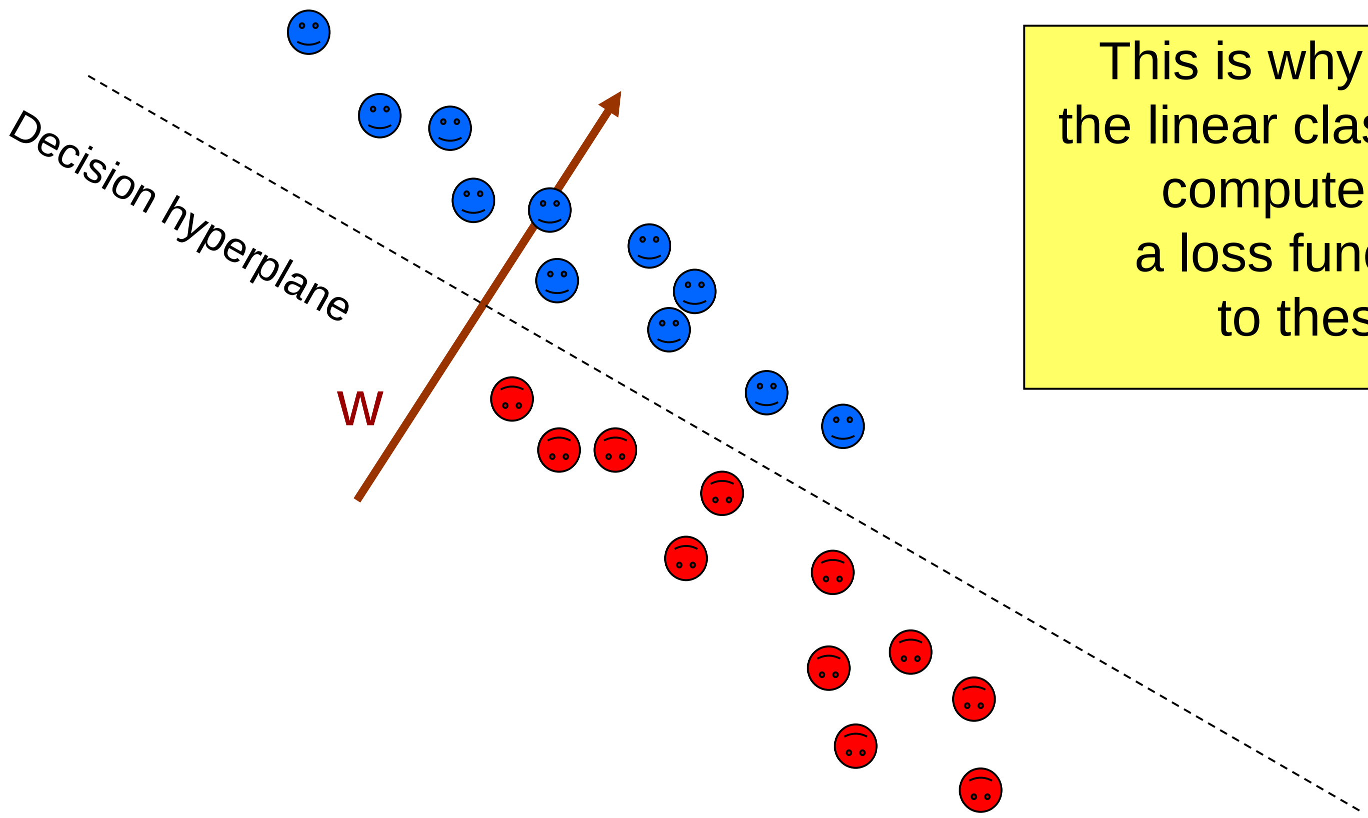
Simple idea to compute *w* ;
compute the centroids *mPlus* and *mMinus* of the two classes, and use *w* = *mPlus* - *mMinus*



Works well for this type of distribution



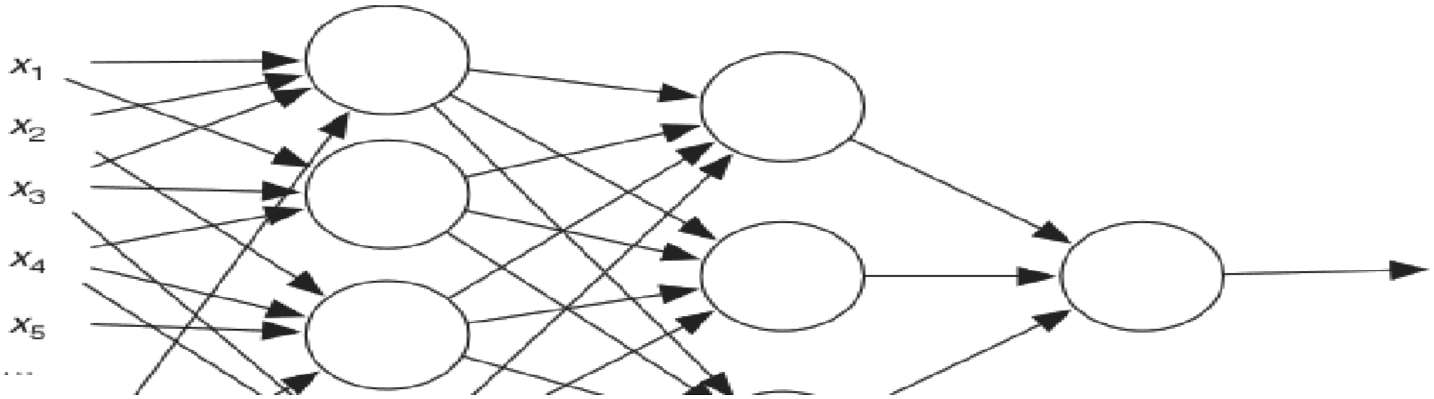
Does not work so well for this type of distribution



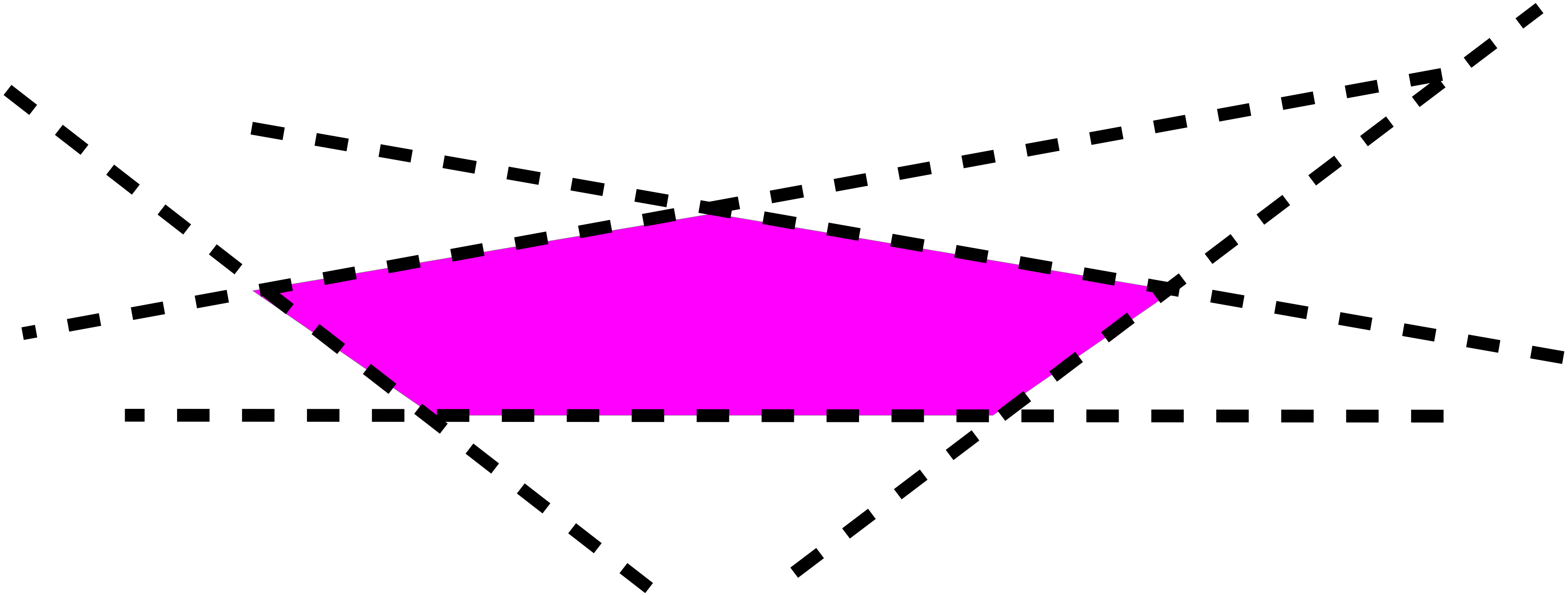
This is why the parameters of the linear classifier are in practice computed by minimizing a loss function with respect to these parameters

Neural Networks

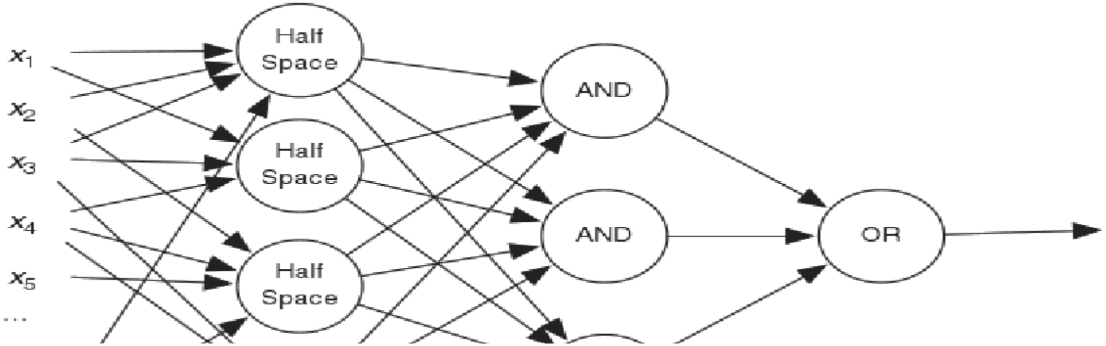
- Not all classes are linearly separable !!
- Multilayer neural networks can overcome this problem



Polyhedron by intersection half-spaces



Network structure to approximate generic partitions

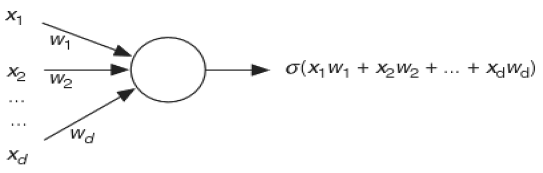


What type of region can we represent with this network?

Answer: unions of polyhedra

Sigmoid neuron

- Output of the sigmoid neuron



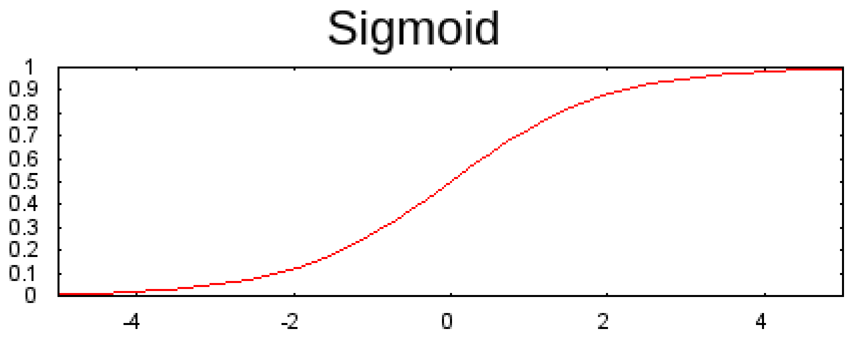
$$a = \sigma(x_1 w_1 + \cdots + x_d w_d)$$

- Sigmoid function

$$\sigma(y) = \frac{1}{1 + e^{-y}}$$

- Derivative

$$\sigma'(y) = \frac{d\sigma}{dy} = \sigma(y)(1 - \sigma(y))$$



The sigmoid is a smoothed version of the step function.
It converts the weighted sum to a value between 0 and 1

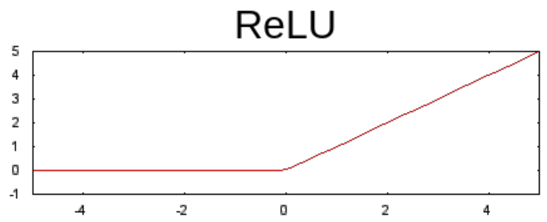
Adding layers for more representation capacity

- Stacking nonlinearities on nonlinearities lets us model very complicated relationships between the inputs and the predicted outputs.
- In brief, each layer is effectively learning a more complex, higher-level function over the raw inputs.

ReLU activation function

- The rectified linear unit activation function (or ReLU, for short) often works a little better than a smooth function like the sigmoid, while also being significantly easier to compute.

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

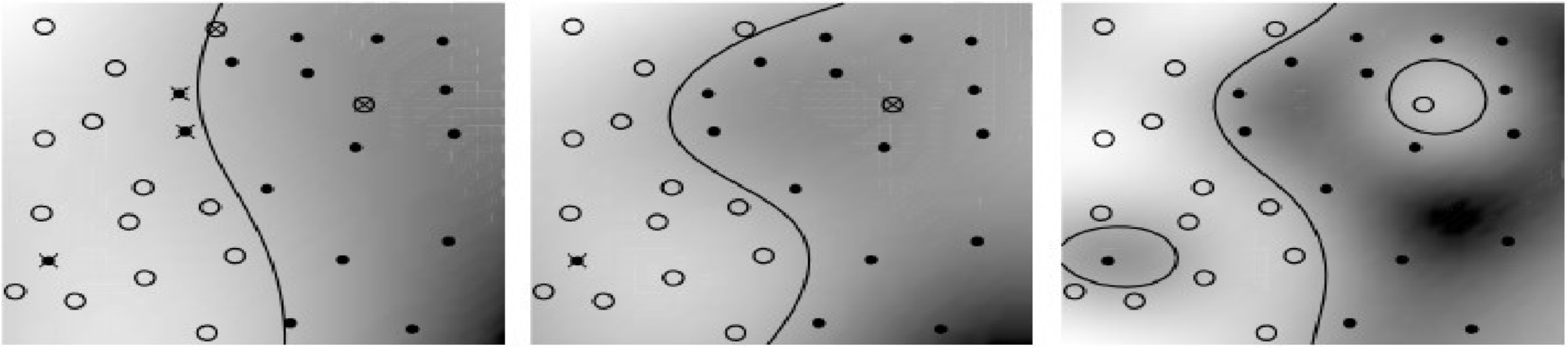


- The superiority of ReLU is based on empirical findings, probably driven by ReLU having a more useful range of responsiveness. A sigmoid's responsiveness falls off relatively quickly on both sides.

Gradients can easily vanish with a sigmoid function

There exist other activation functions

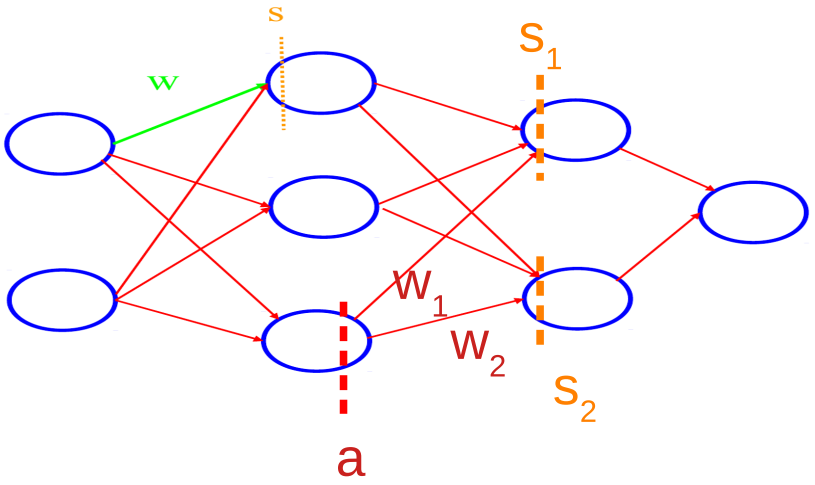
Landscape defined by NN's



Backpropagation training algorithm

- A technique to minimize loss by computing the gradients of loss with respect to the model's parameters, conditioned on training data.
- Informally, gradient descent iteratively adjusts parameters, gradually finding the best combination of weights and bias to minimize loss.

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial s} \times \frac{\partial s}{\partial w}$$



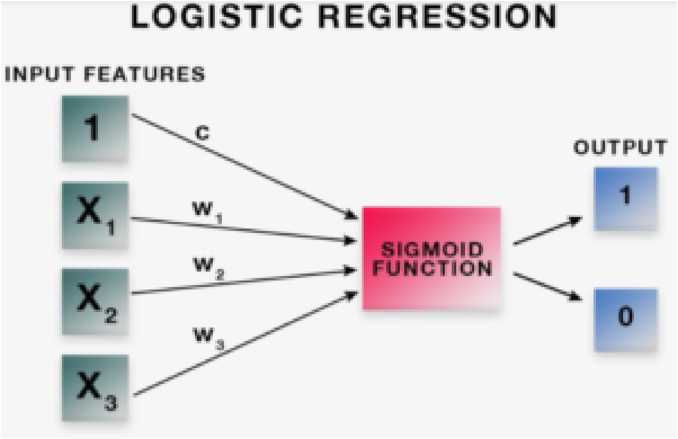
Backprop is a clever application of the chain rule of calculus

$$\frac{\partial E}{\partial a} = w_1 \times \frac{\partial E}{\partial s_1} + w_2 \times \frac{\partial E}{\partial s_2}$$

Feedforward neural networks in sklearn

sklearn.linear_model.**LogisticRegression**

- For binary classification use *logsig*,
  for multi-class problem use *softmax*

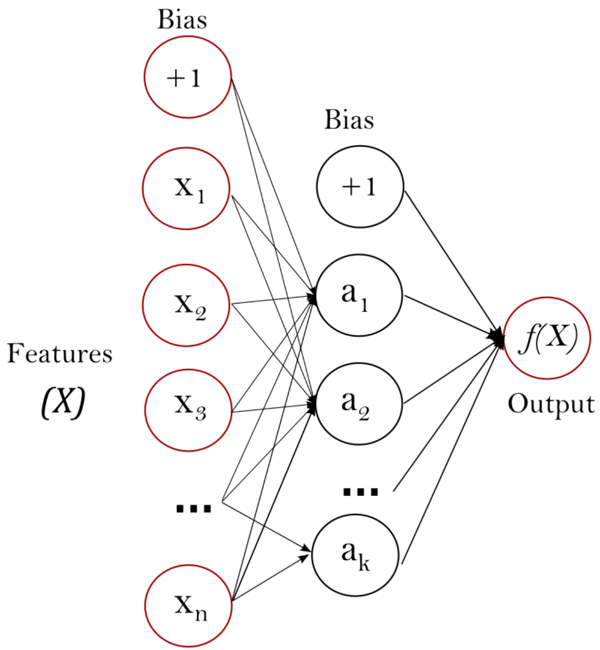


$$
\begin{aligned}
&\implies p(X) = \frac{e^{(\beta_o+\beta_1 x)}}{e^{(\beta_o+\beta_1 x)} + 1}\\
&\implies p(e^{(\beta_o+\beta_1 x)} + 1) = e^{(\beta_o+\beta_1 x)}\\
&\implies p.e^{(\beta_o+\beta_1 x)} + p = e^{(\beta_o+\beta_1 x)}\\
&\implies p = e^{(\beta_o+\beta_1 x)} - p.e^{(\beta_o+\beta_1 x)}\\
&\implies p = e^{(\beta_o+\beta_1 x)}(1-p)\\
&\implies \frac{p}{1-p} = e^{(\beta_o+\beta_1 x)}\\
&\implies \ln(\tfrac{p}{1-p}) = \beta_0 + \beta_1 x
\end{aligned}
$$

Log of the ratio of the probs of
positive and negative classes
is modeled as a linear function

sklearn.neural_network.**MLPClassifier**

- **Multi-layer Perceptron** (MLP) is a supervised learning algorithm that learns a function by training on a dataset
- Given a set of features and a target, a MLP can learn a non-linear function approximator for either classification or regression.
- It is different from **logistic regression**, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers.



Feedforward neural networks in Keras

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Check

- https://keras.io/guides/
- https://www.tensorflow.org/tutorials/keras/classification