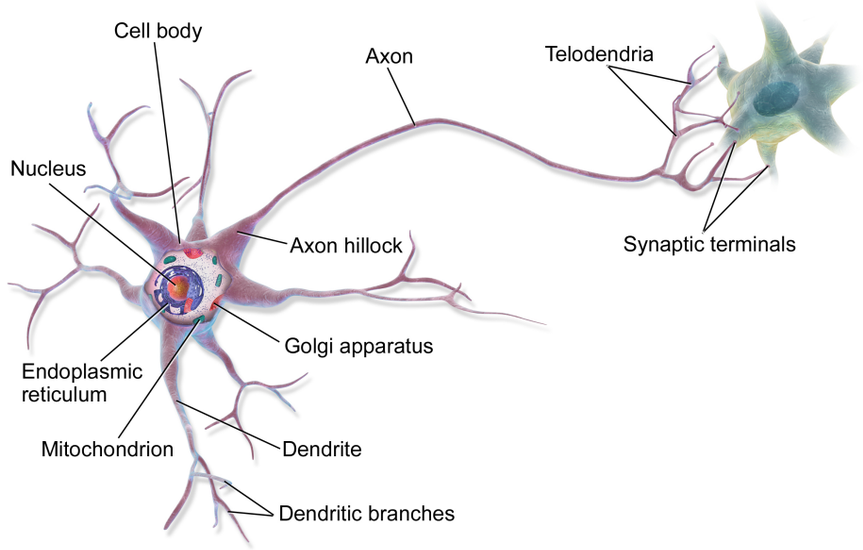
Preview:
Neural Networking

Thomas Schwarz, SJ

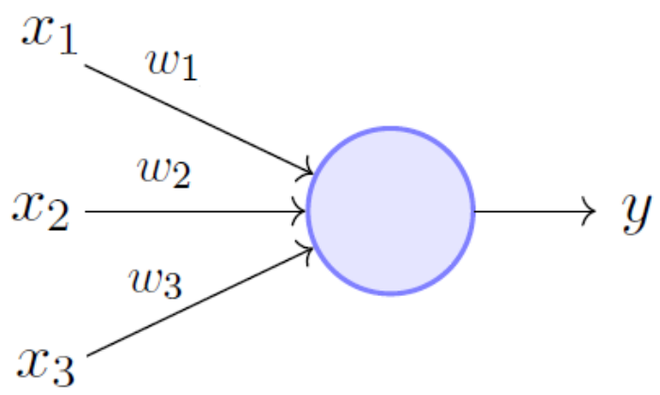
Perceptrons

- Neural networks are biology inspired
  - Instead of using Good Old AI (GOAI):
    - Try to build an artificial brain
- Brains are made up of dendrites / neurons
  - Have inputs that are activated by electricity
  - Have outputs that activate electricity



Perceptrons

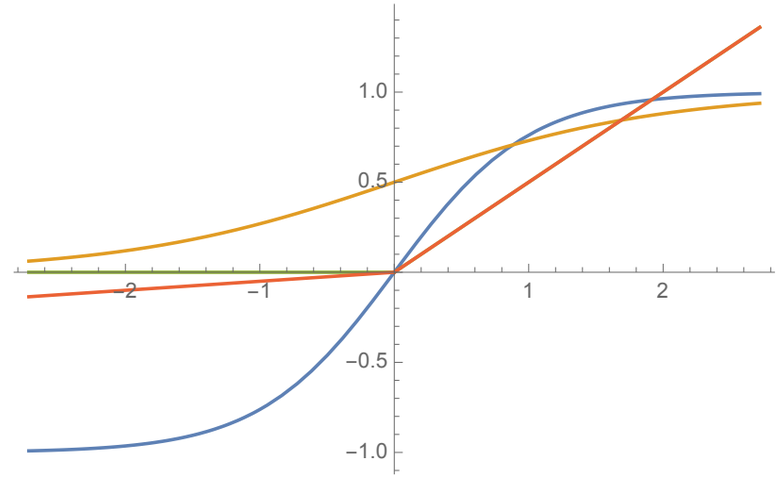
- Make a very simple model:
  - A perceptron is a unit that takes a number of inputs
  - Takes the weighted sum of the inputs
  - Subjects it to an activation function
- Outputs the result $y = f(\sum_{i=1}^{n} w_i x_i)$



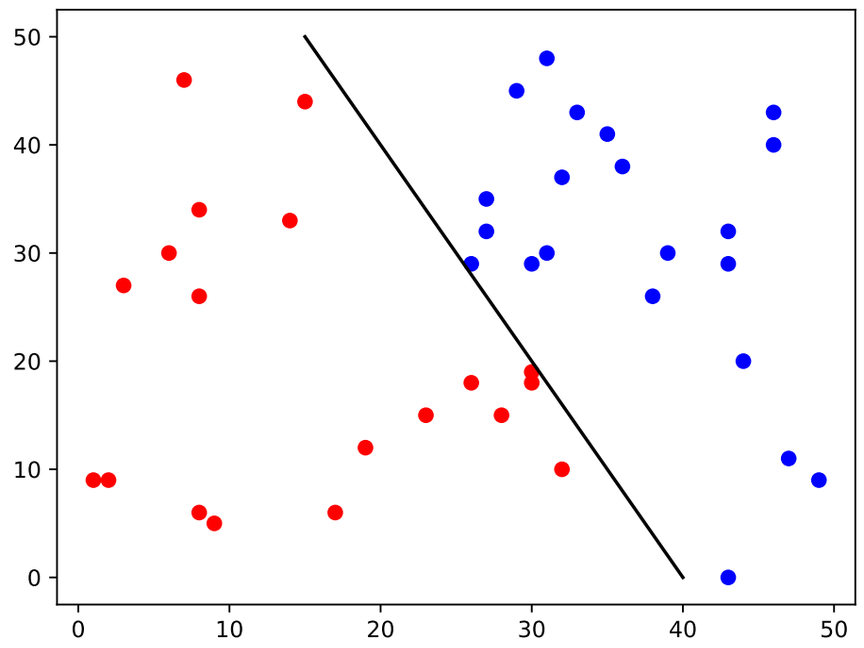
Perceptron Model (Minsky-Papert in 1969)

Perceptrons

- Activation functions are various: sigmoid, tanh, step functions, …



- A single perceptron can be made to classify data points along a hyper-plane

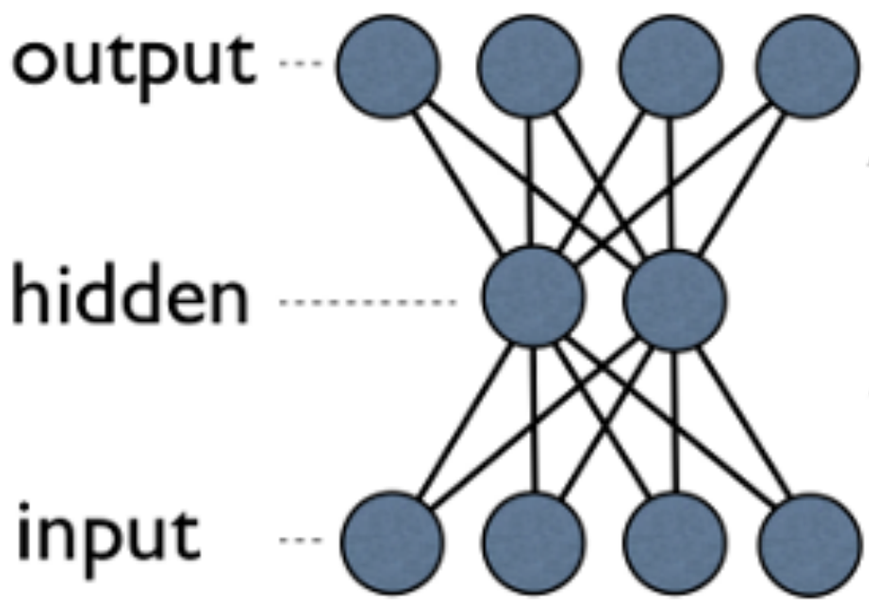


Perceptrons

- Perceptrons:
  - 1943: McCulloch and Pitts model of a neuron, very much a perceptron with step-wise activation and equal weight
  - 1949: Hebb: Weights are different in nature
  - 1958: Rosenblatt: Perceptrons work as a generic tool
  - 1969: Minsky and Seymore: Perceptrons are really too limited

Neural Networks

- Create network of perceptrons = neural networks
  - 1989: Neural networks shown to be “universal approximators”
  - 1986: Hinton: Backpropagation learning algorithm



Neural Networks

- 1991: Hochreiter: Deep neural networks (with many different hidden layers) are difficult to train with back-propagation
  - Vanishing or exploding gradients
- 2000 - 2020: Lots of training data, Use of GPU (70 times faster), New types of networks, Better learning algorithms
  -

Working with Neural Networks

- Several Python based packages for deep-learning
  - Keras
  - PyTorch
- Run best on GPU

Simple Example

- Pima Indian Data Set
  - Preparing the data:
    - Need to load data into a tensor
    - Need to separate training from testing

Simple Example

- Imports first

```
from keras.models import Sequential, load_model
from keras.layers import Dense
import numpy as np
import os

np.random.seed(8)
```

Simple Example

- Need to load data:

```
Pregnancies,Glucose,BloodPressure,SkinThickness,Insulin,BMI,DiabetesPedigree
Function,Age,Outcome
6,148,72,35,0,33.6,0.627,50,1
1,85,66,29,0,26.6,0.351,31,0
8,183,64,0,0,23.3,0.672,32,1
1,89,66,23,94,28.1,0.167,21,0
```

- Use Numpy's loadtxt

```
dataset = np.loadtxt("diabetes.csv", delimiter = ',',
skiprows=1)
```

Simple Example

- Separate output from input parameters
  - Output is the last column
    - This is where slicing comes in

```
X = dataset[:,:-1]
Y = dataset[:,-1]
```

Simple Example

```
>>> X
array([[  6.   , 148.   ,  72.   , ...,  33.6  ,   0.627,  50.   ],
       [  1.   ,  85.   ,  66.   , ...,  26.6  ,   0.351,  31.   ],
       [  8.   , 183.   ,  64.   , ...,  23.3  ,   0.672,  32.   ],
       ...,
       [  5.   , 121.   ,  72.   , ...,  26.2  ,   0.245,  30.   ],
       [  1.   , 126.   ,  60.   , ...,  30.1  ,   0.349,  47.   ],
       [  1.   ,  93.   ,  70.   , ...,  30.4  ,   0.315,  23.   ]])
>>> Y
array([1., 0., 1., 0., 1., 0., 1., 0., 1., 1., 0., 1., 0., 1., 1., 1., 1.,
       1., 0., 1., 0., 0., 1., 1., 1., 1., 1., 0., 0., 0., 0., 1., 0., 0.,
       0., 0., 0., 1., 1., 1., 0., 0., 0., 1., 0., 1., 0., 0., 1., 0., 0.,
       0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 1., 0., 1., 0.,
```

Simple Example

- Now separate into training and testing data, using slices
  - There are 768 records, we use the last 68 as test:

```
Xtrain = X[:700,:]
Ytrain = Y[:700]
Xtest = X[700:,:]
Ytest = Y[700:]
```

Simple Example

- Now use Keras magic:
  - Define a neural network with
    - an initial layer of 8 neurons,
    - a first layer with 16 neurons
    - a second layer with 8 neurons
    - an output layer with 1 neuron

Simple Example

- Sequential models are easiest

```
model=Sequential()
model.add(Dense(16, input_dim=8, activation = 'relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

- Now train

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

model.fit(Xtrain, Ytrain, epochs = 150, batch_size=10,
verbose=0)
```

Simple Example

- Get accuracy: (Usually around 70% on test data)

```
score=model.evaluate(Xtest, Ytest, verbose=0)
print(f'{model.metrics_names[1]} {score[1]}')
```