Deep Learning for Everyone

Bruno Gonçalves

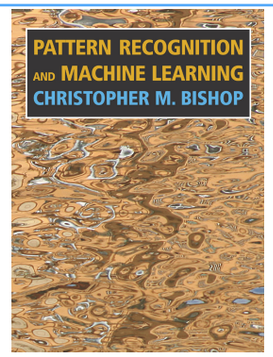
www.data4sci.com/newsletter

graphs4sci.substack.com

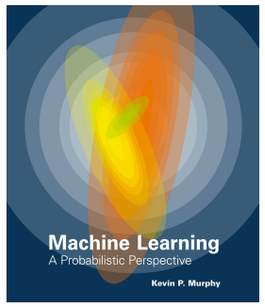
https://github.com/DataForScience/DeepLearning

References

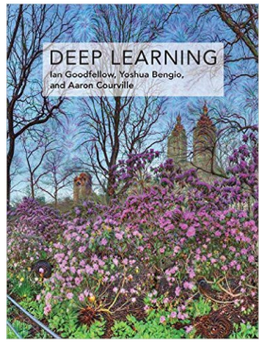
https://github.com/DataForScience/DeepLearning



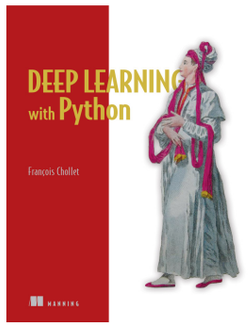
https://amzn.to/2AcDHmX



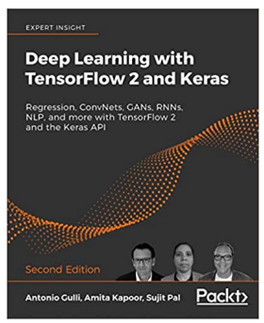
https://amzn.to/3gZZPBu



https://amzn.to/2BGr0RL

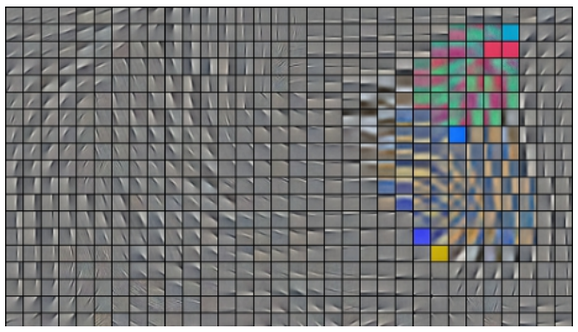


https://amzn.to/30fTJqB



https://amzn.to/30fTMCN

https://amzn.to/2AavBuT



Neural Networks for Machine Learning

Geoff Hinton

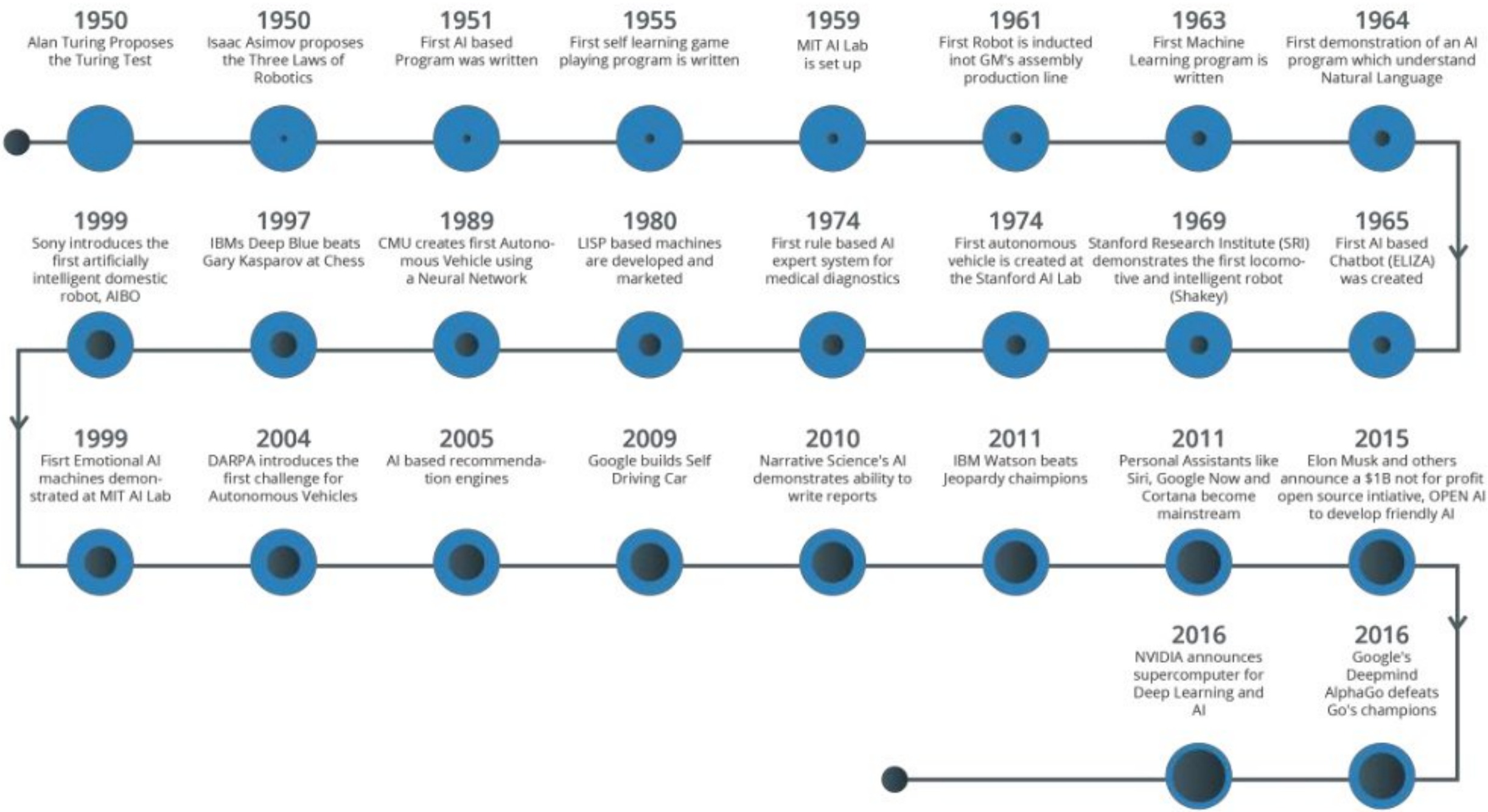
Machine Learning

Andrew Ng

Requirements

https://github.com/DataForScience/DeepLearning

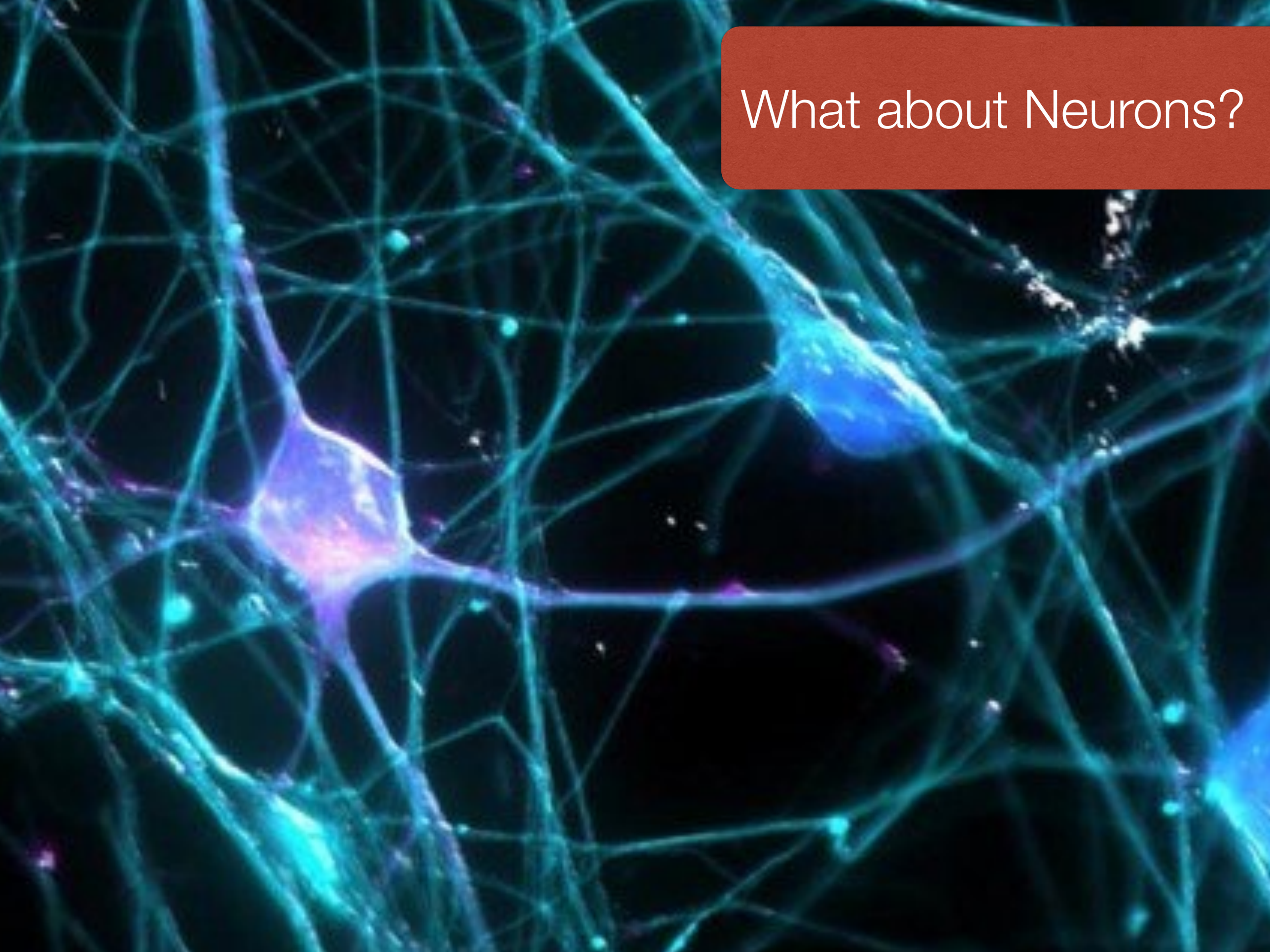
AI Key Milestone Events



No. of transistors per sq. inch

Tracxn

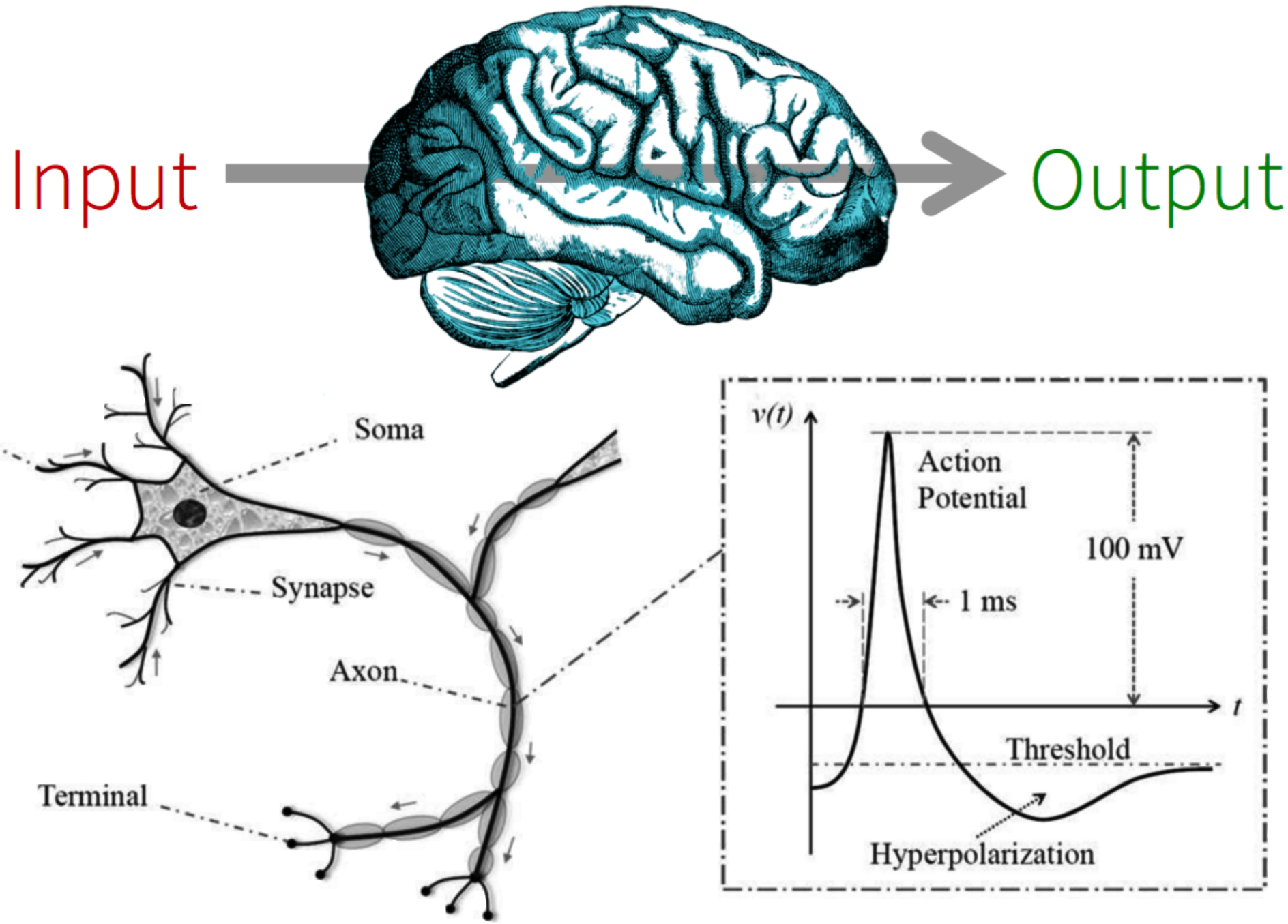
Machine Learning



What about Neurons?

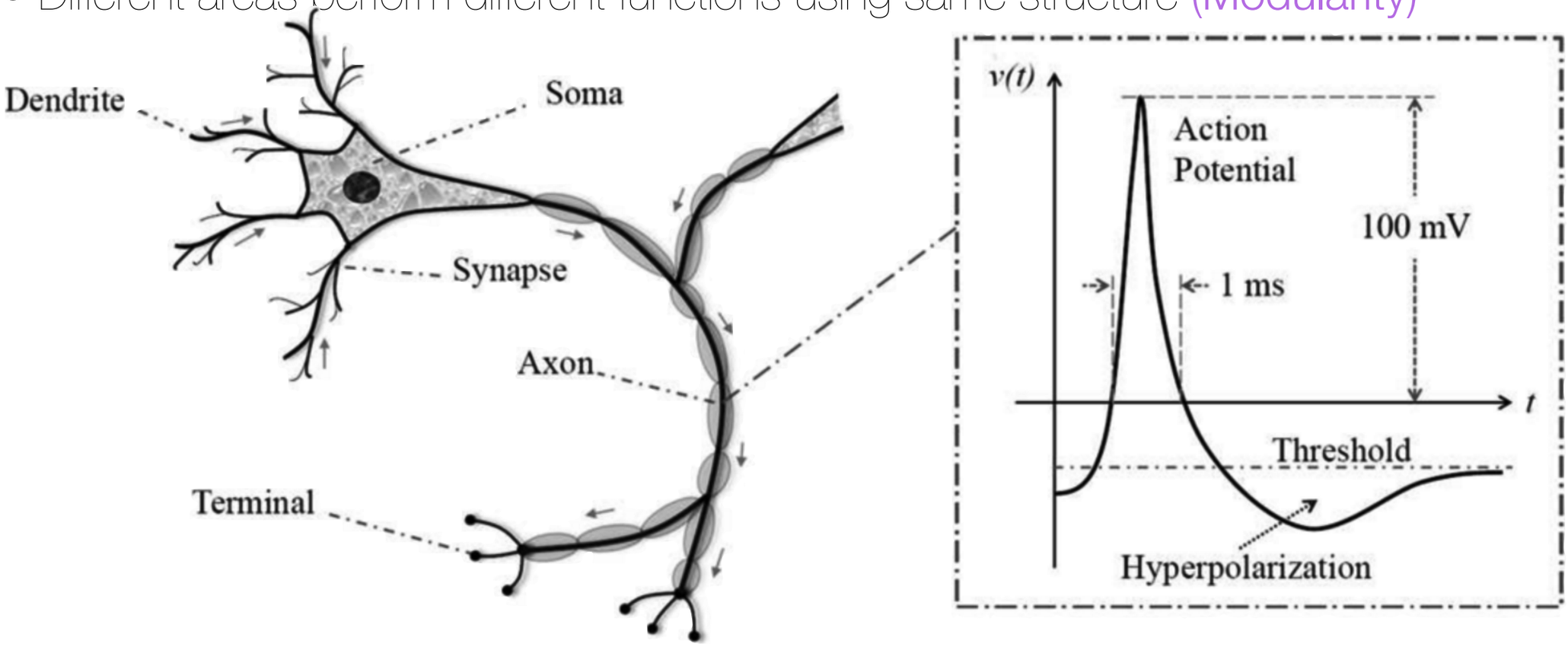
How the Brain "Works" (Cartoon version)

How the Brain "Works" (Cartoon version)

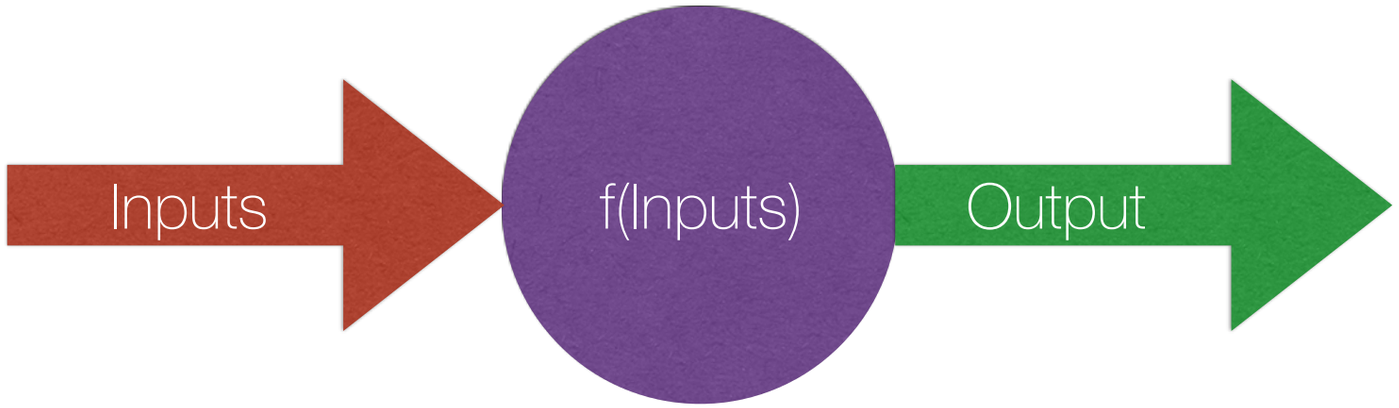


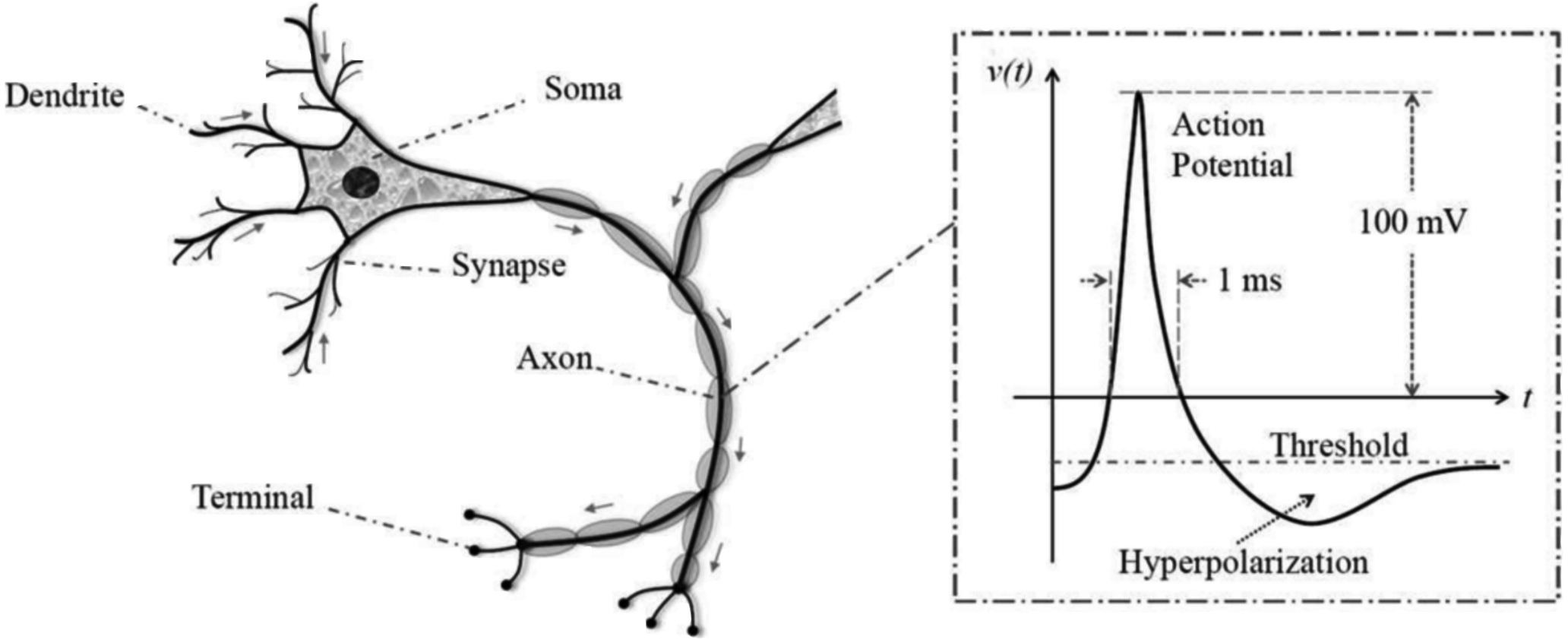
How the Brain "Works" (Cartoon version)

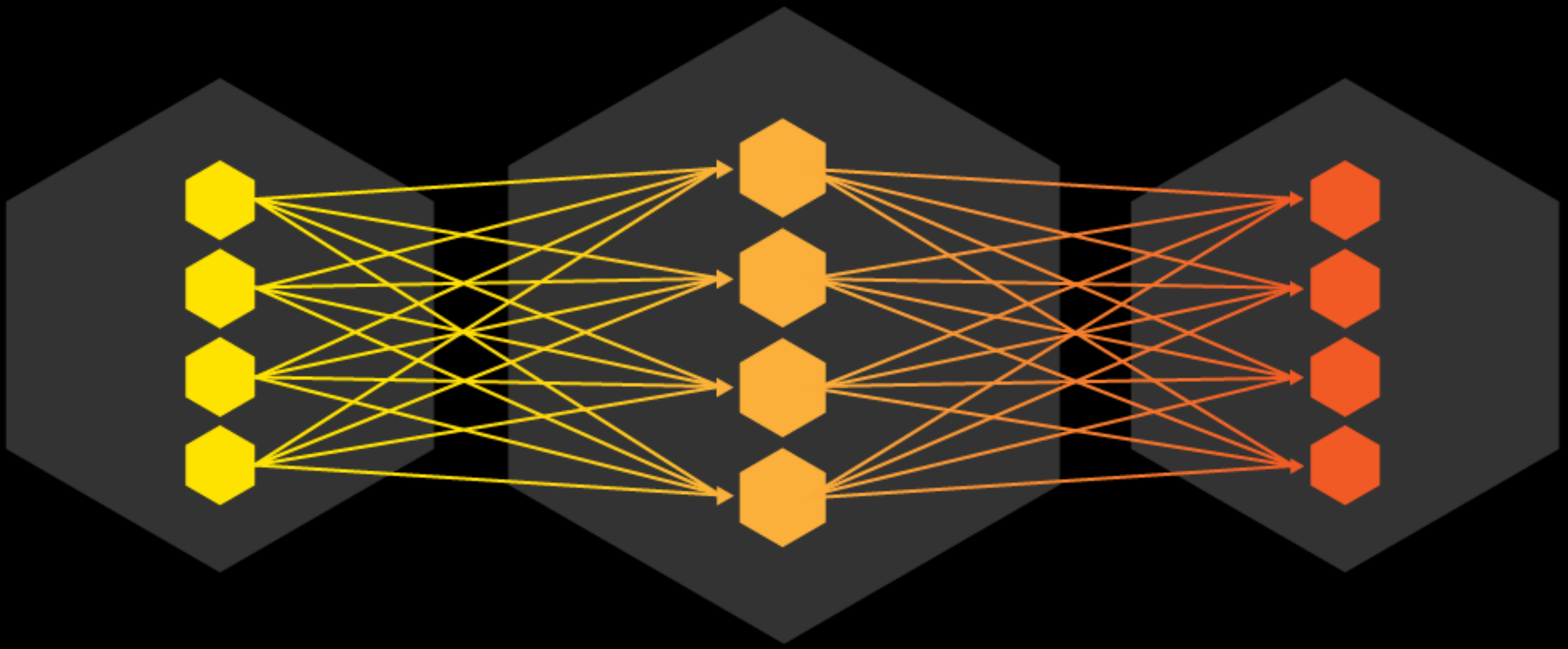
- Each neuron receives input from other neurons
- 10^{11} neurons, each with with 10^{4} weights
- Weights can be positive or negative
- Weights adapt during the learning process
- "neurons that fire together wire together" (Hebb)
- Different areas perform different functions using same structure (Modularity)



How the Brain "Works" (Cartoon version)







INPUT TERMS

FEATURES
PREDICTIONS
ATTRIBUTES
PREDICTABLE VARIABLES

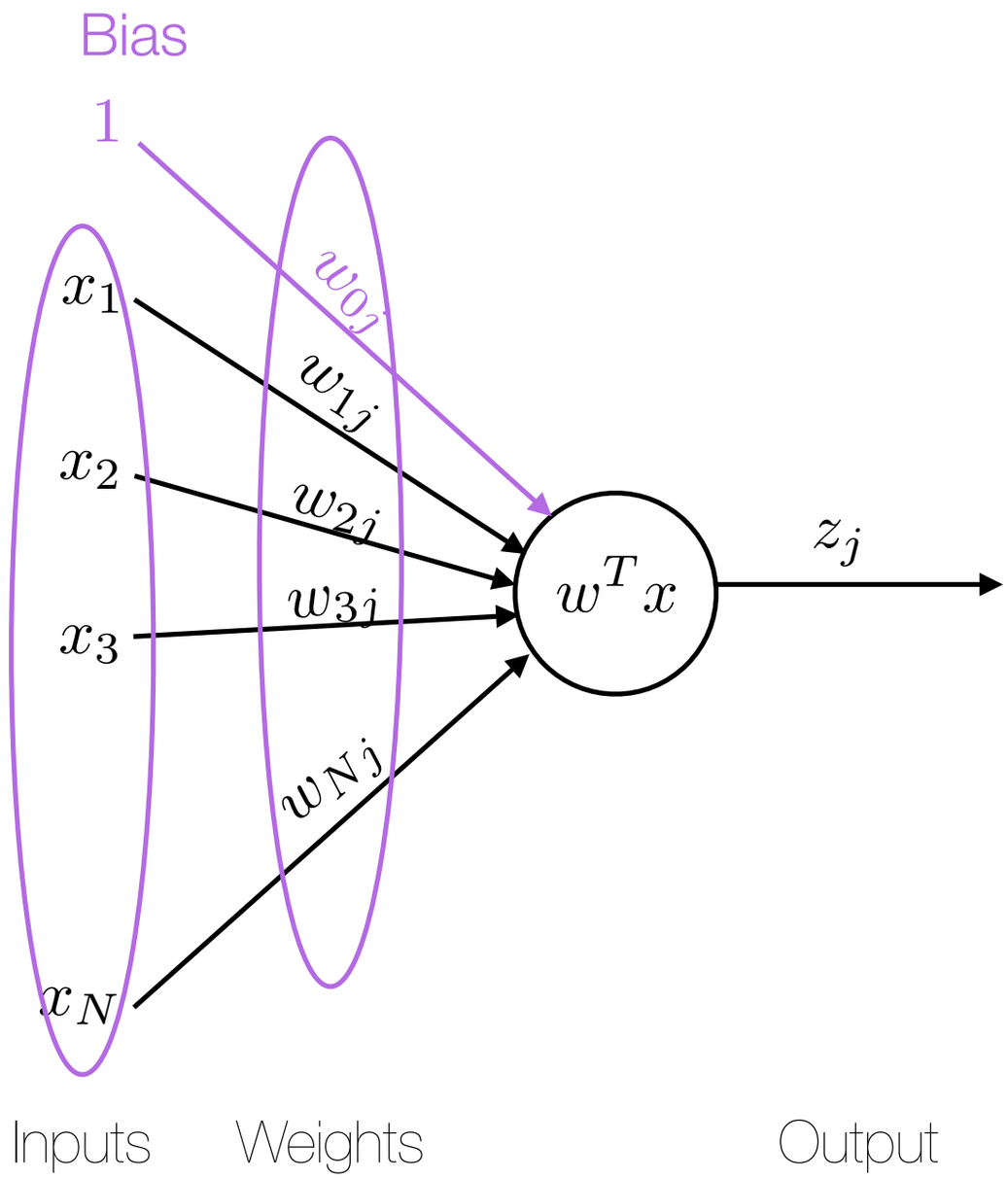
MACHINE

ALGORITHMS
TECHNIQUES
MODELS

OUTPUT TERMS

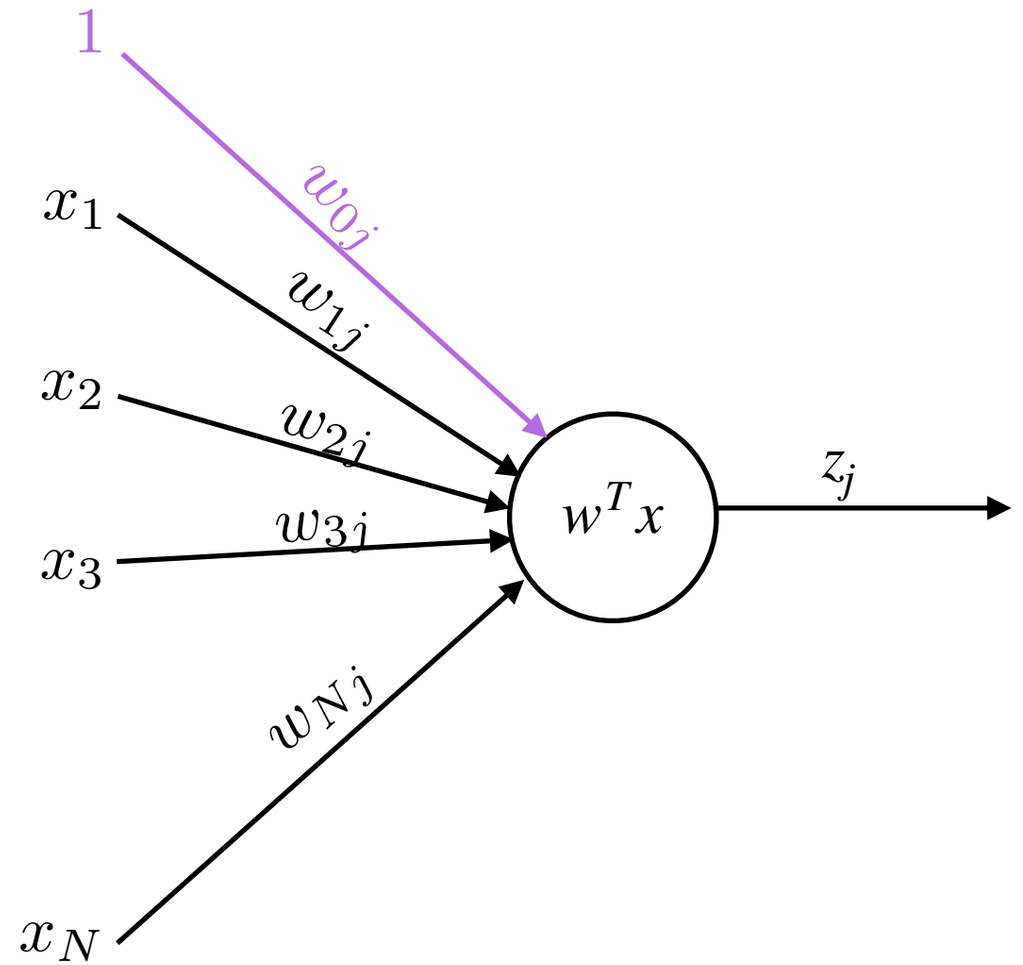
CLASSES
RESPONSES
TARGETS
DEPENDANT VARIABLES

Perceptron


Bias
1
w_{0j}
x_1
w_{1j}
x_2
w_{2j}
x_3
w_{3j}
w_{Nj}
x_N
$w^T x$
z_j
Inputs
Weights
Output

Perceptron - Forward Propagation

- The output of a perceptron is determined by a sequence of steps:
    - obtain the inputs
    - multiply the inputs by the respective weights

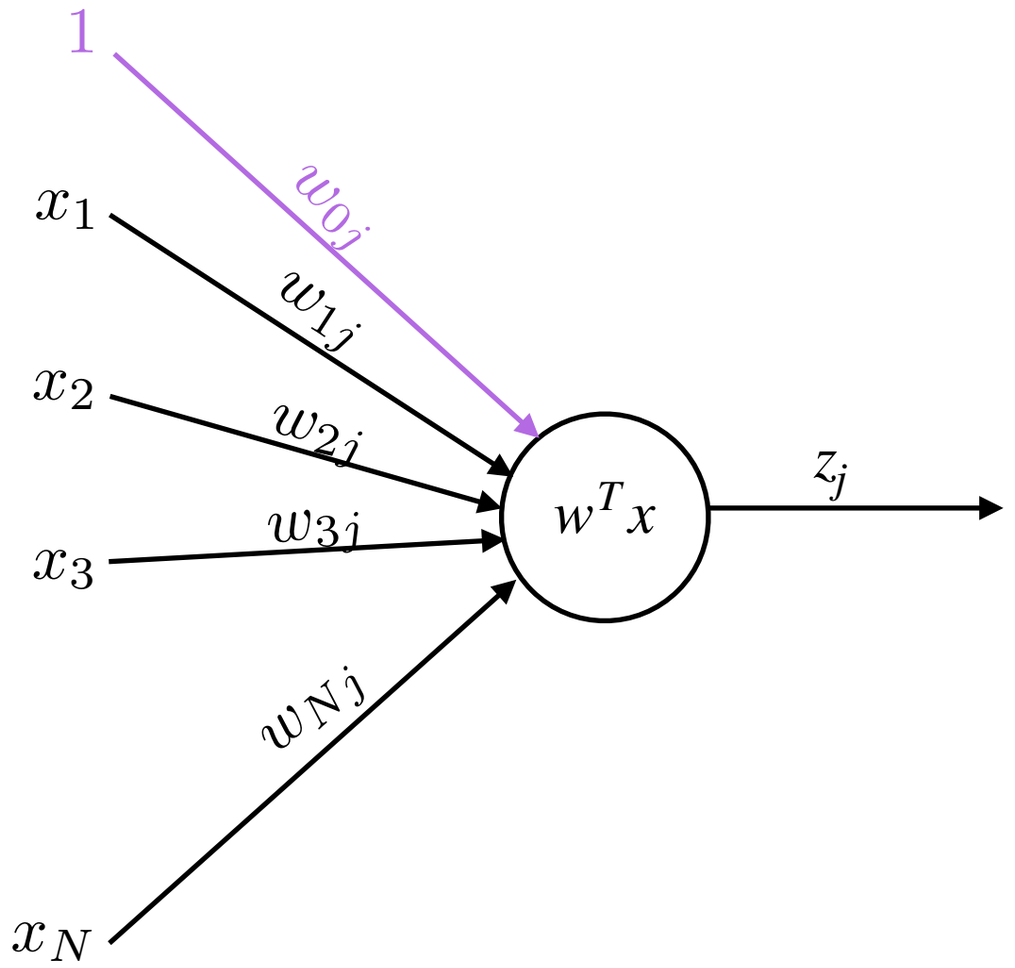


Perceptron - Forward Propagation

```
def forward(Theta, X, active):
    N = X.shape[0]

    # Add the bias column
    X_ = np.concatenate((np.ones((N, 1)), X), 1)

    # Multiply by the weights
    z = np.dot(X_, Theta.T)

    return a
```

Perceptron - Training

- Training Procedure:
  - If correct, do nothing
  - If output incorrectly outputs 0, add input to weight vector
  - if output incorrectly outputs 1, subtract input to weight vector
- Guaranteed to converge, if a correct set of weights exists
- Given enough features, perceptrons can learn almost anything
- Specific Features used limit what is possible to learn

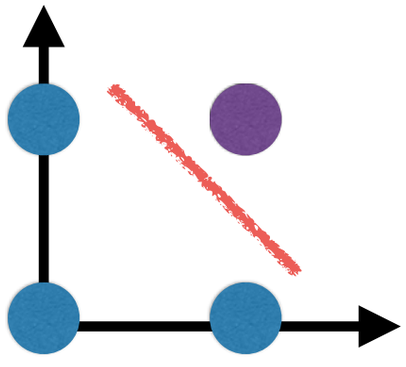


Linear Boundaries

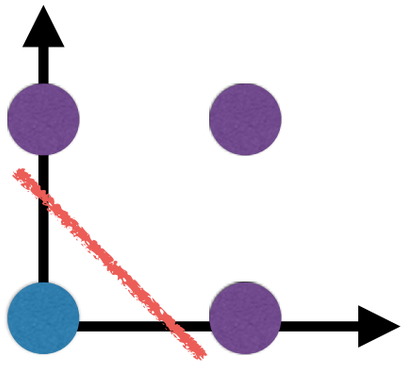
1

0

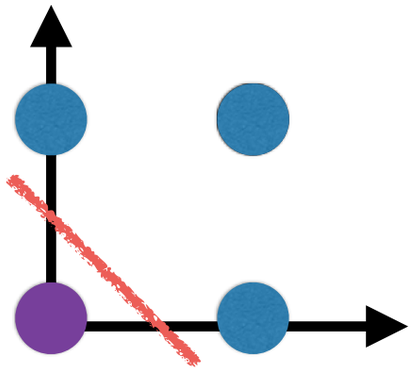
AND



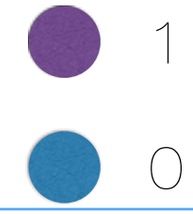
OR



NOR

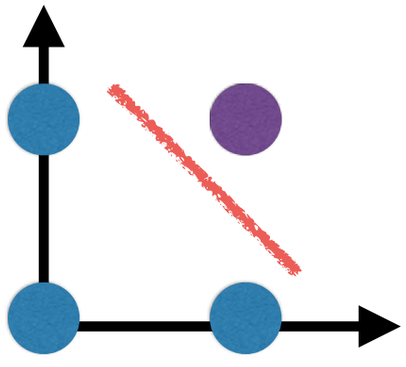


Linear Boundaries



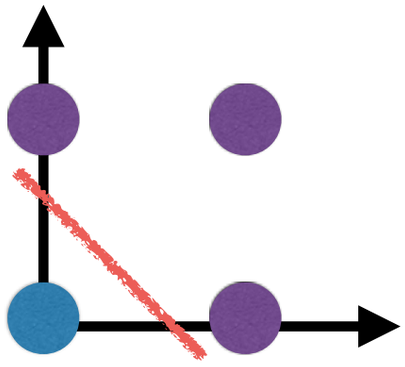
- Perceptrons rely on hyperplanes to separate the data points. Unfortunately, this is not always possible:

AND



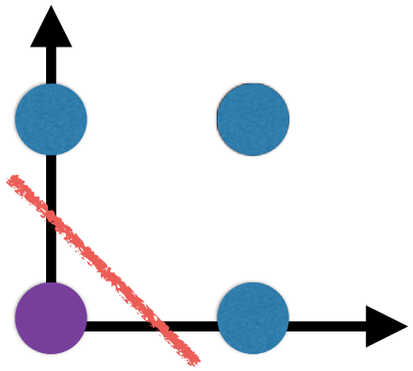
Possible

OR



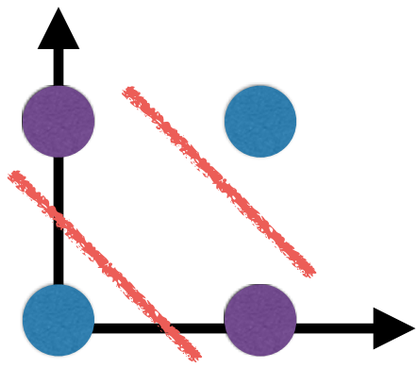
Possible

NOR



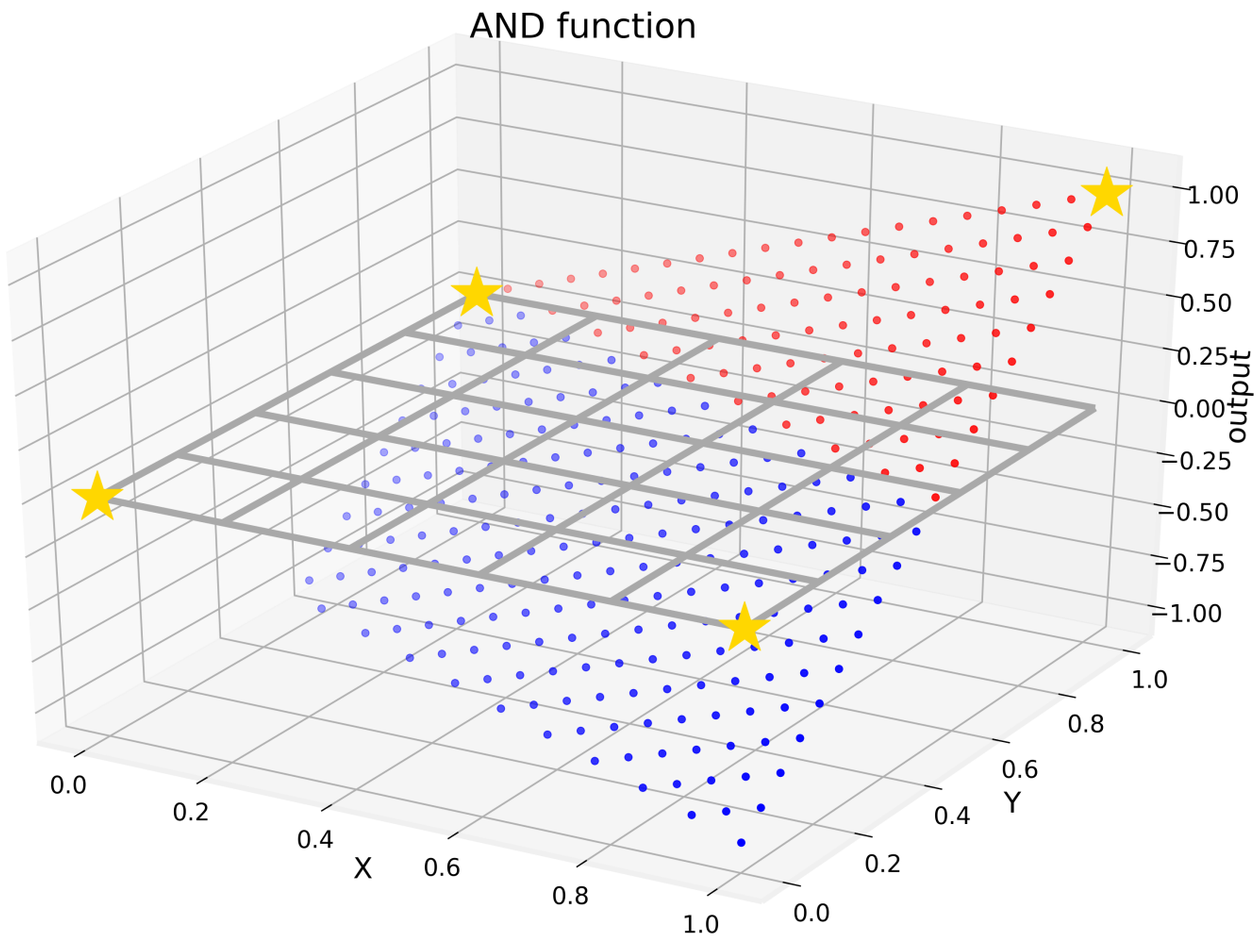
Possible

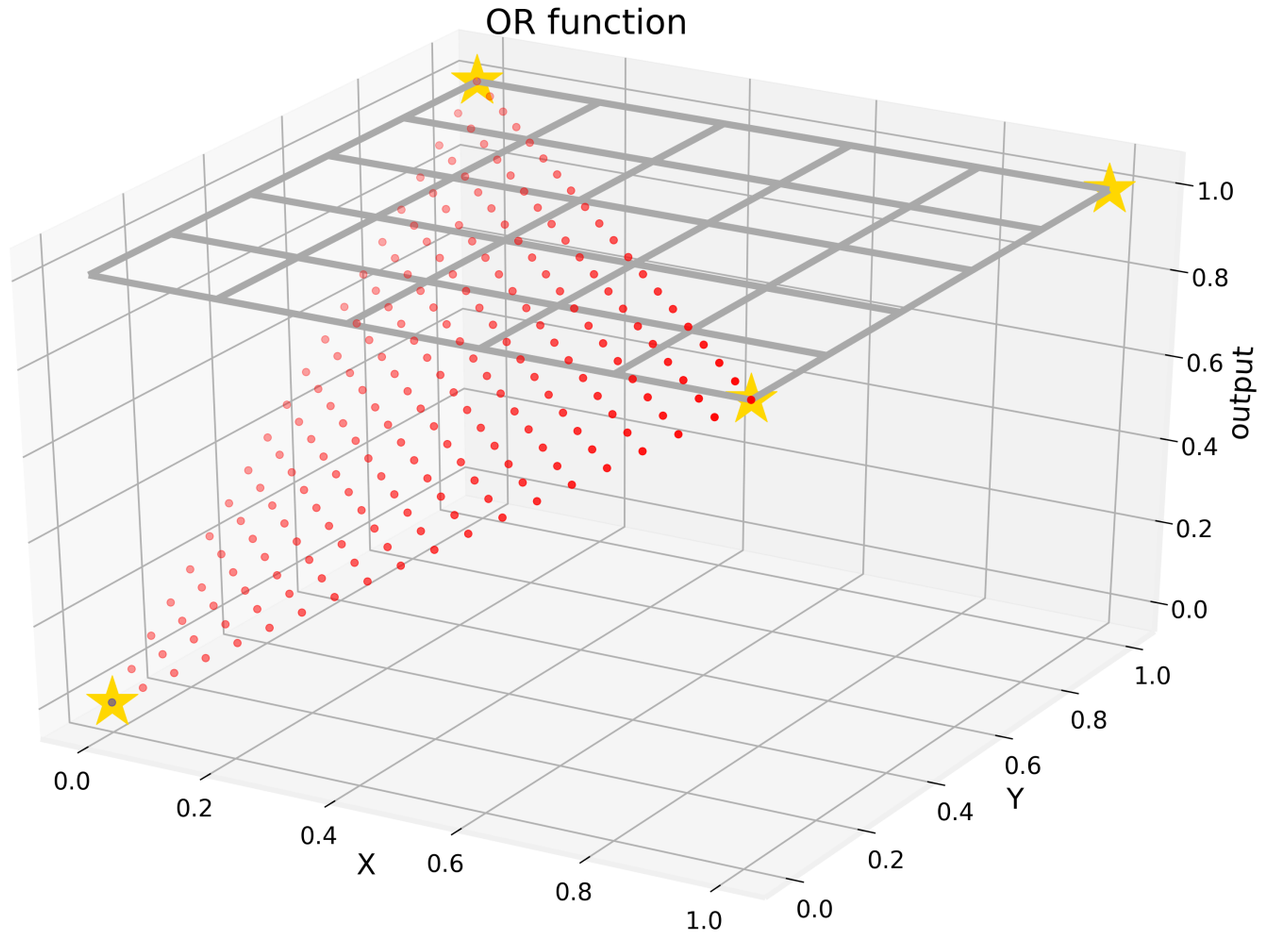
XOR



Impossible

Linear Boundaries

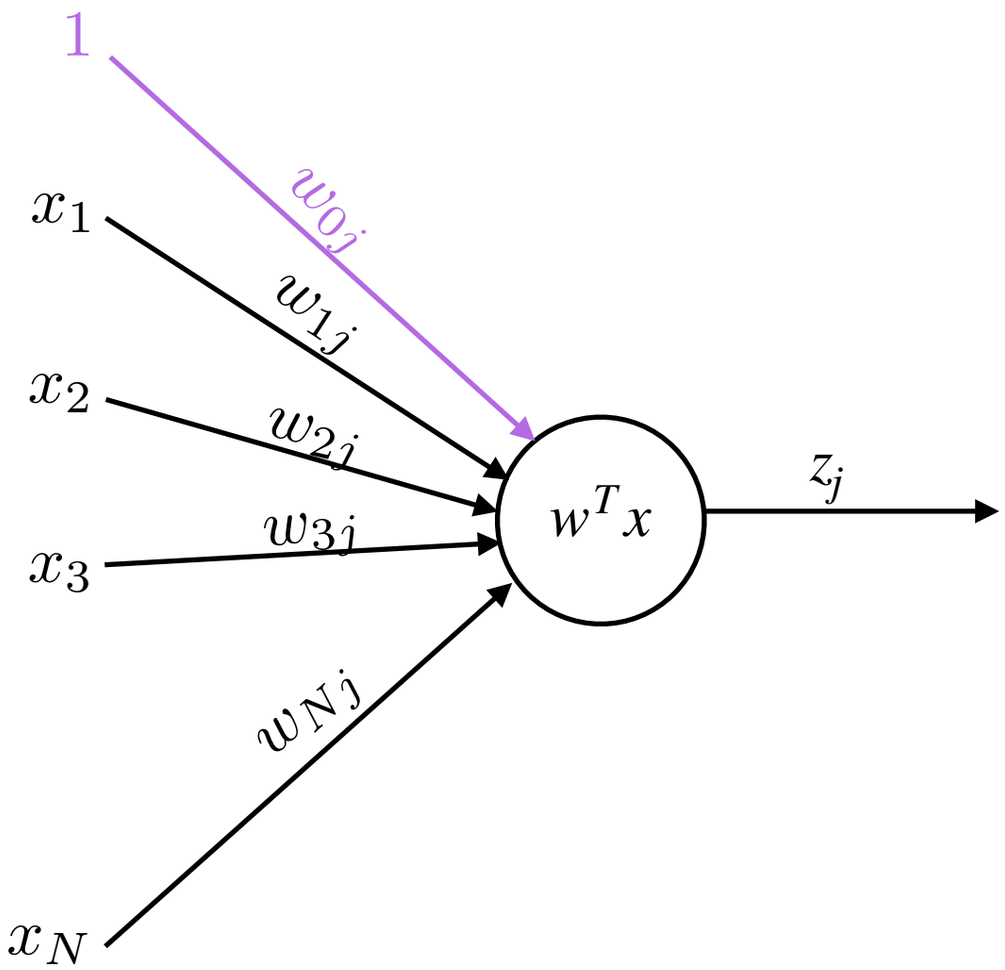




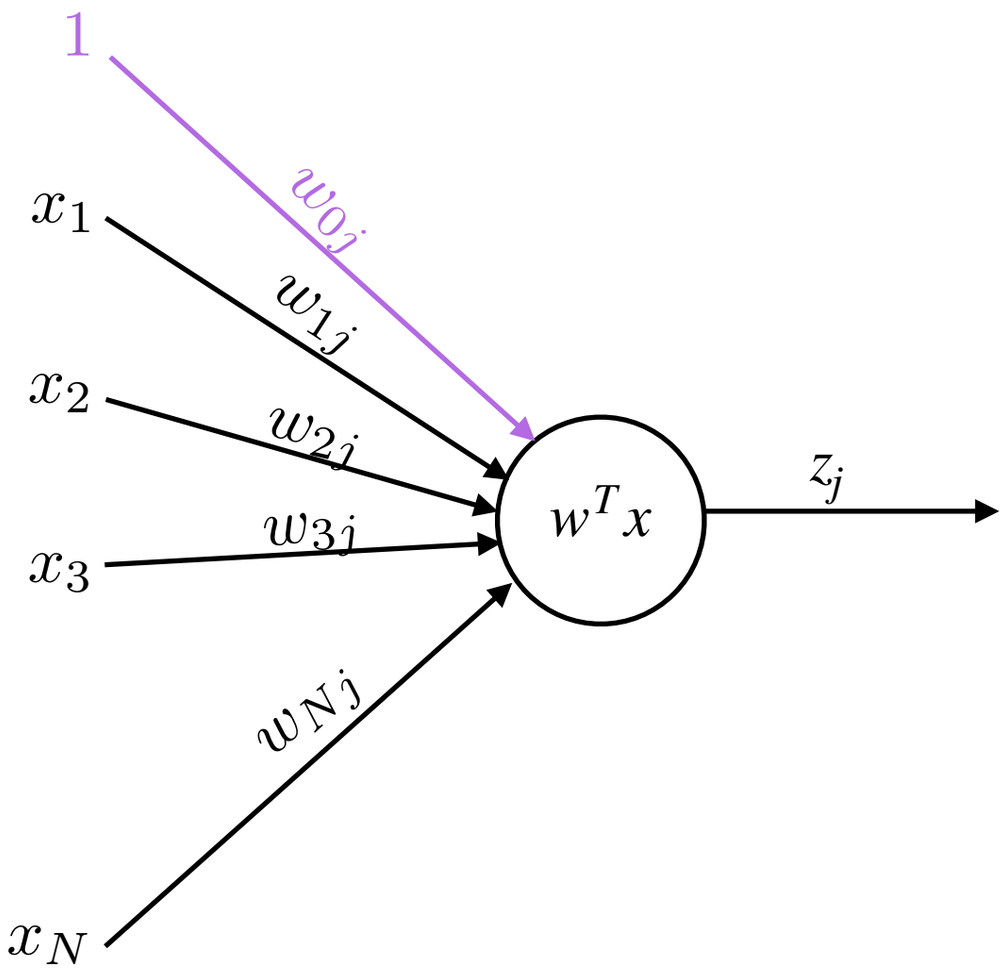
Code - Perceptron / Forward Propagation

https://github.com/DataForScience/DeepLearning

Perceptron



Perceptron



- This is just a graphical representation of:

$$z = w^T x$$

which is just linear regression!

Linear Regression

$$\vec{x_i} = (x_0, x_1, \cdots, x_n)^T$$

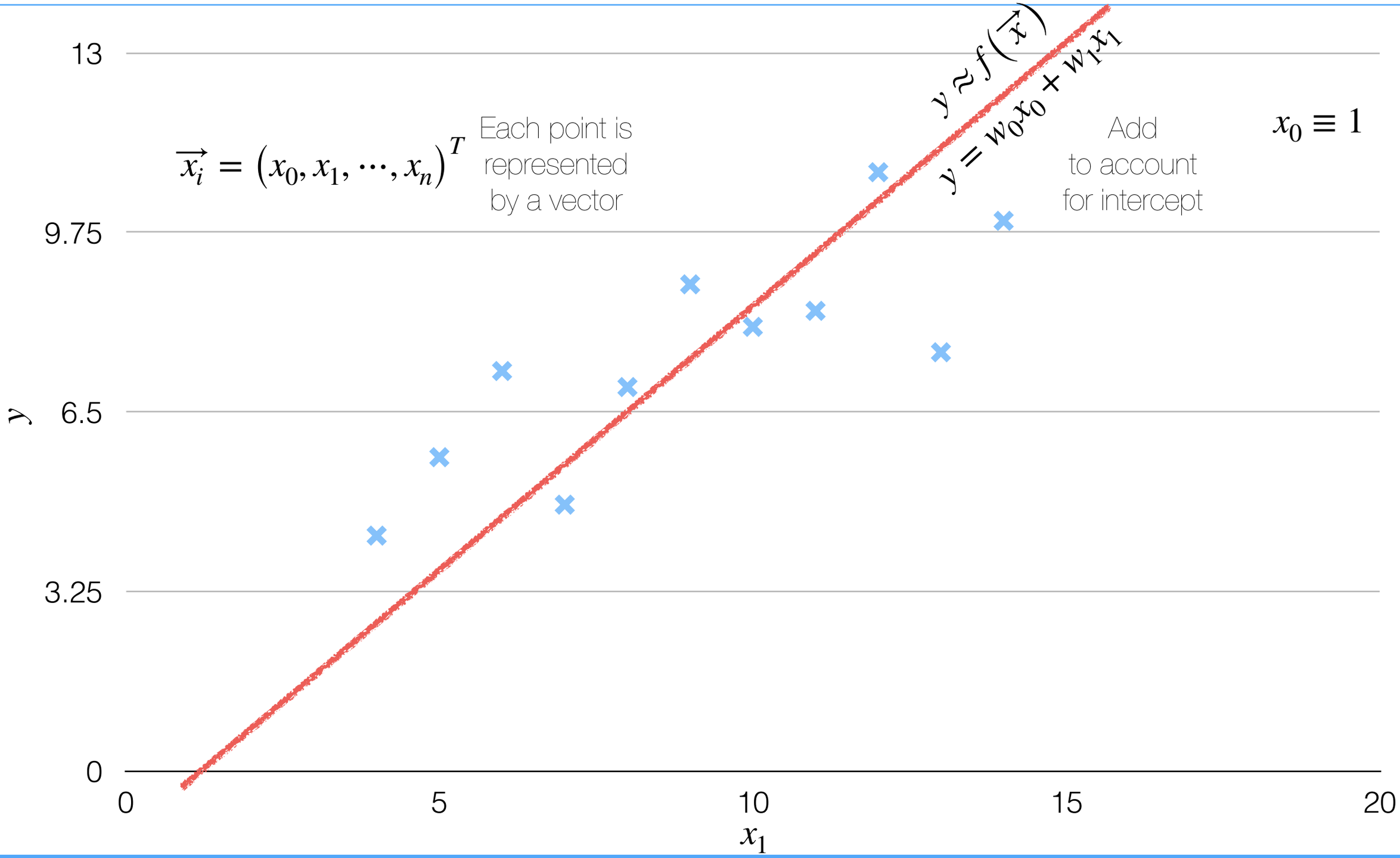
Each point is represented by a vector

$$y \approx f(\vec{x})$$

$$y = w_0 x_0 + w_1 x_1$$

Add to account for intercept

$$x_0 \equiv 1$$



Optimization Problem

- (Machine) Learning can be thought of as an optimization problem.

- Optimization Problems have 3 distinct pieces:

    - The constraints

    - The function to optimize

    - The optimization algorithm.

Optimization Problem

- (Machine) Learning can be thought of as an optimization problem.

- Optimization Problems have 3 distinct pieces:

  - The constraints — Problem Representation
  - The function to optimize — Prediction Error
  - The optimization algorithm. — Gradient Descent

Linear Regression

- We are assuming that our functional dependence is of the form:

$$f\left(\overrightarrow{x}\right) = w_0 + w_1 x_1 + \cdots + w_n x_n \equiv X\overrightarrow{w}$$

- In other words, at each step, our hypothesis is:

$$h_w(X) = X\overrightarrow{w} \equiv \hat{y}$$

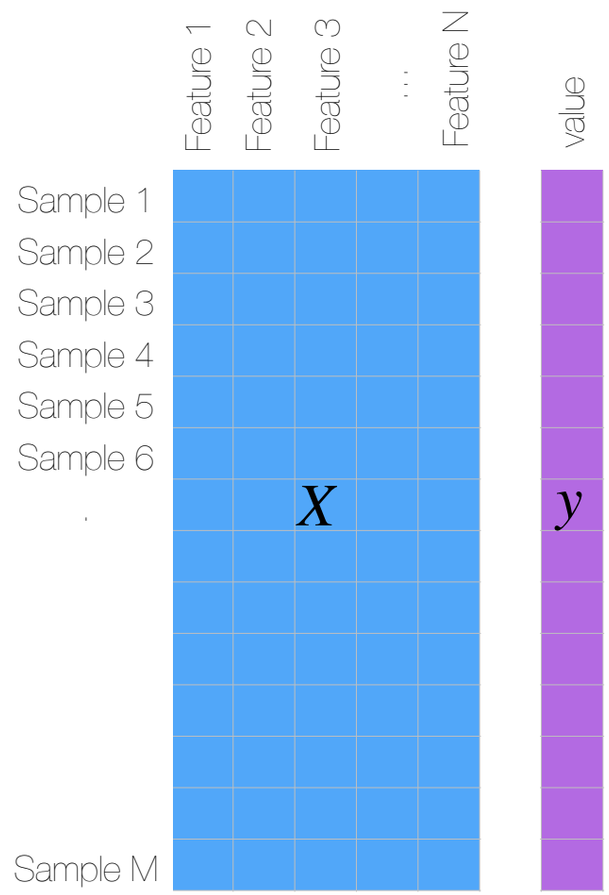
and it imposes a Constraint on the solutions that can be found.

- We quantify our far our hypothesis is from the correct value using an Error Function:

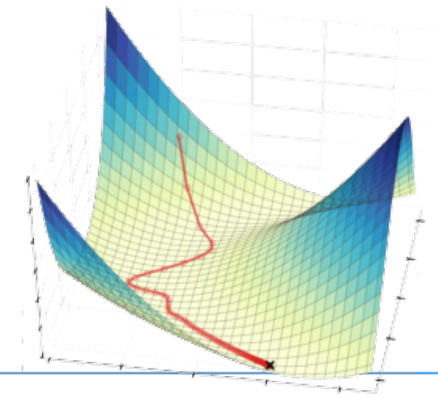
$$J_w\left(X, \overrightarrow{y}\right) = \frac{1}{2m}\sum_i \left[h_w\left(x^{(i)}\right) - y^{(i)}\right]^2$$

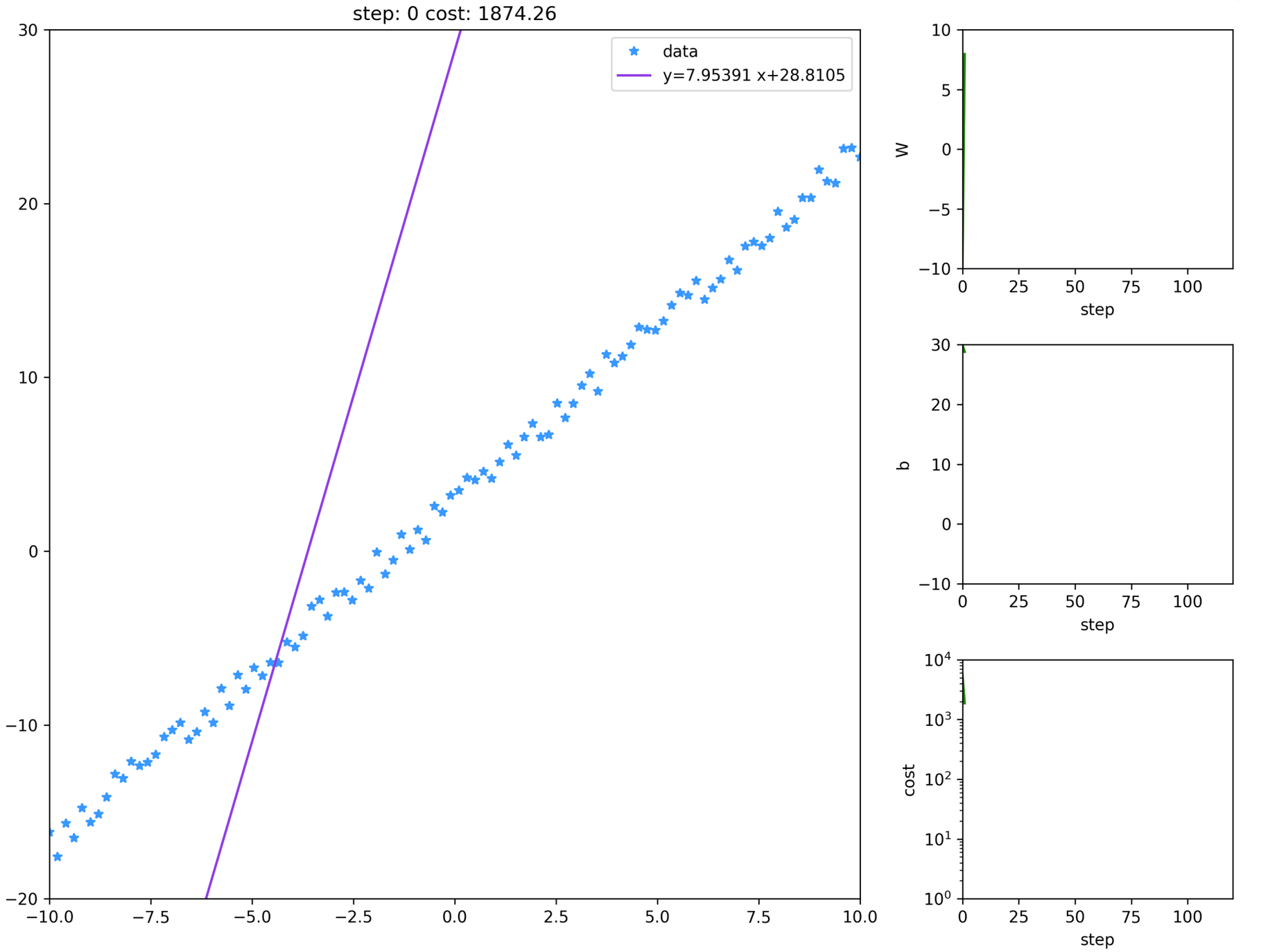
or, vectorially:

$$J_w\left(X, \overrightarrow{y}\right) = \frac{1}{2m}\left[X\overrightarrow{w} - \overrightarrow{y}\right]^2$$



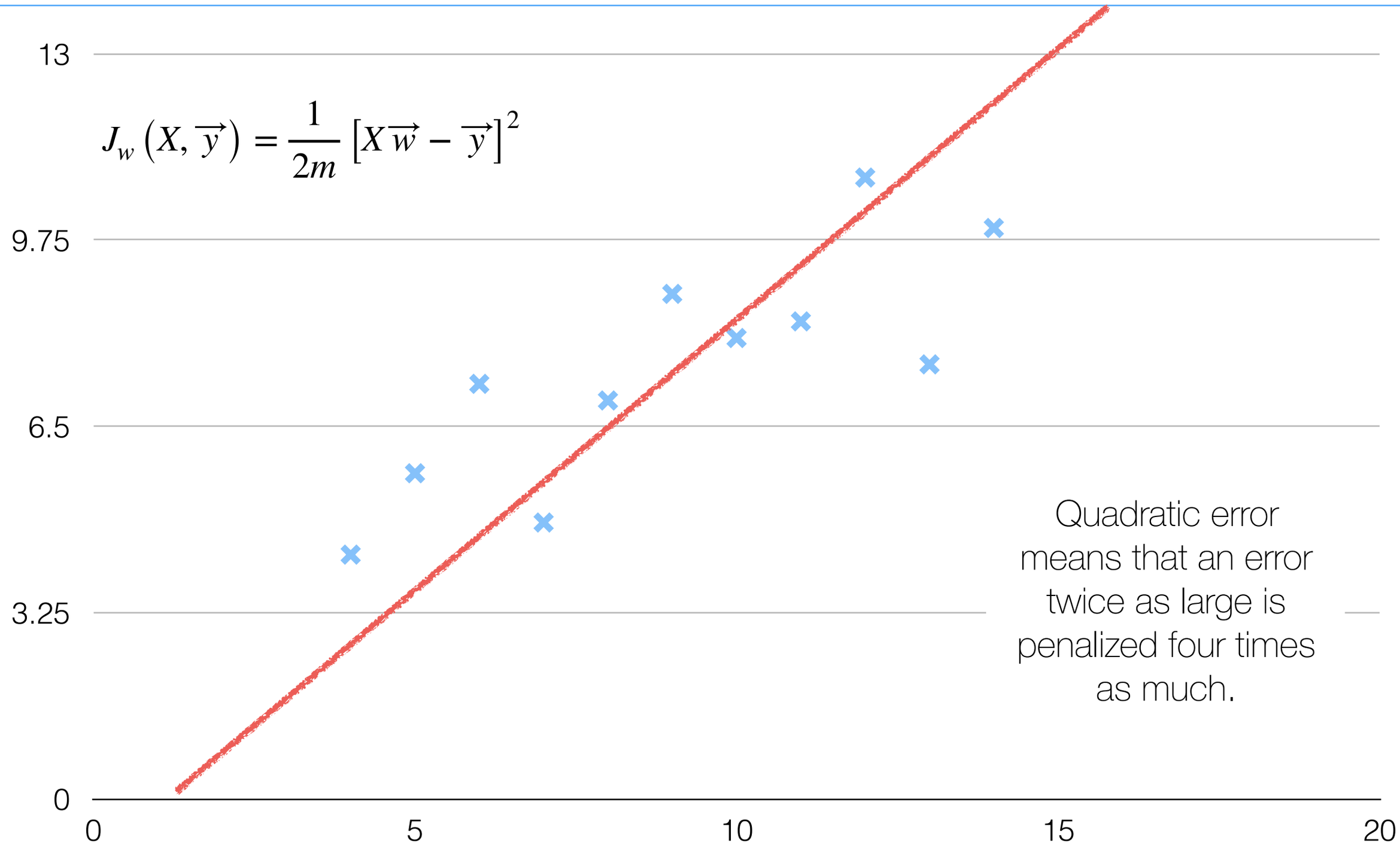
Linear Regression





Geometric Interpretation

$$J_w\left(X, \vec{y}\right) = \frac{1}{2m}\left[X\vec{w} - \vec{y}\right]^2$$

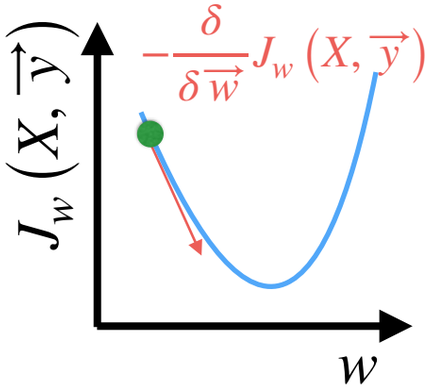


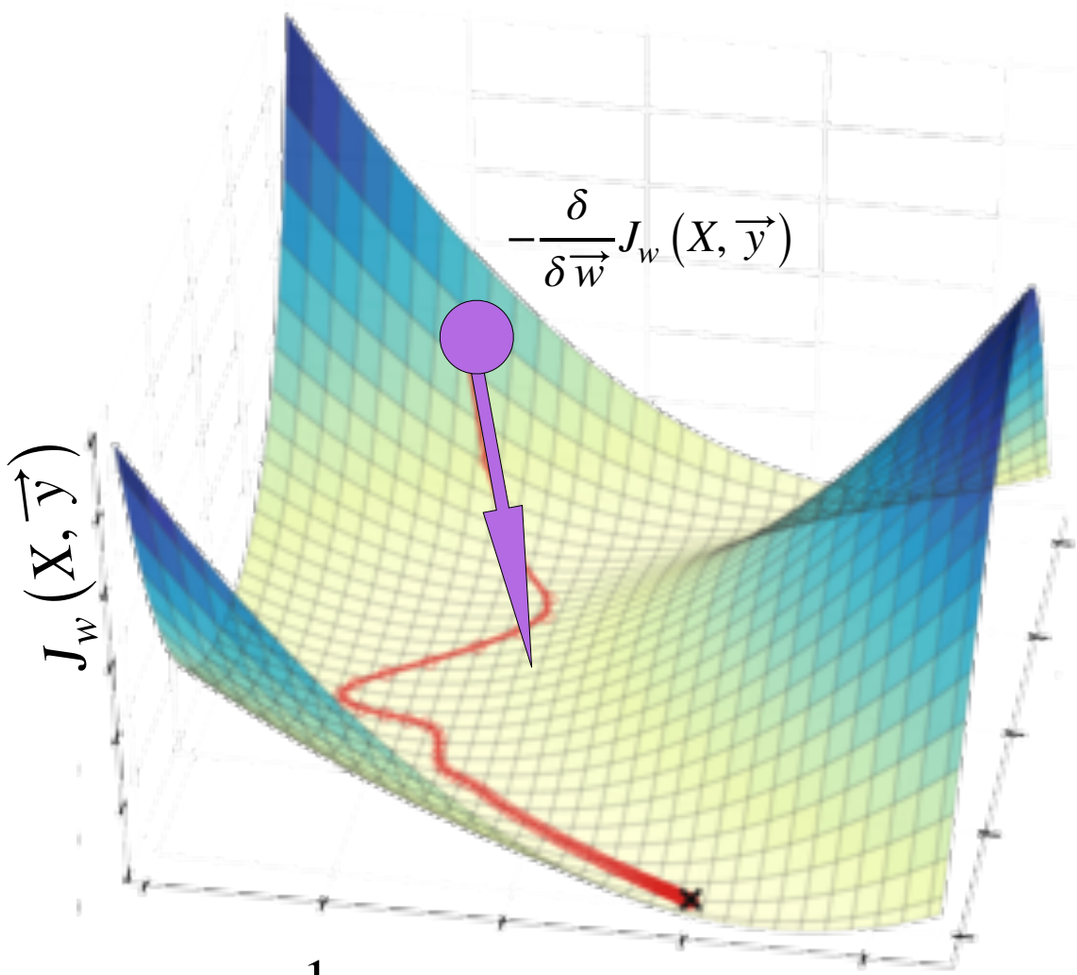
Quadratic error means that an error twice as large is penalized four times as much.

Gradient Descent

- Goal: Find the minimum of $J_w\left(X, \vec{y}\right)$ by varying the components of $\vec{w}$

- Intuition: Follow the slope of the error function until convergence





- Algorithm:

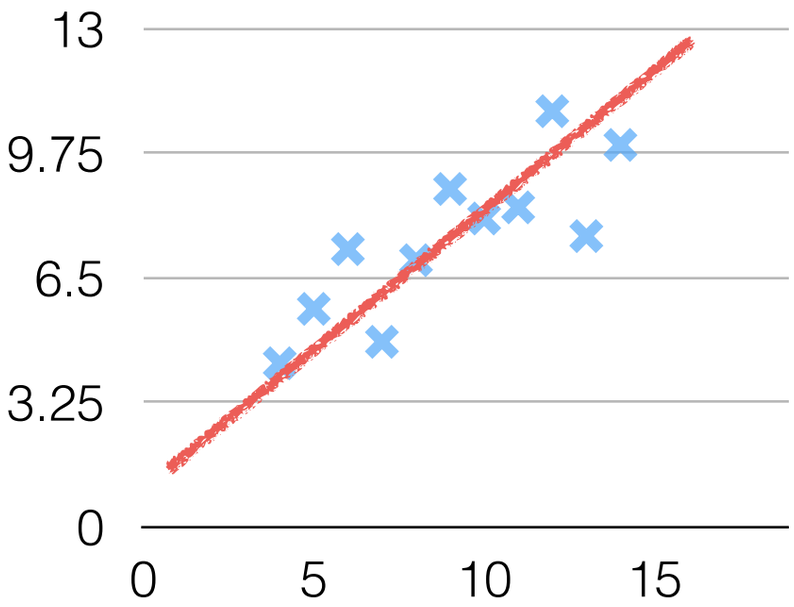
  - Guess $\vec{w}^{(0)}$ (initial values of the parameters)
  - Update until "convergence":

step size

$$w_j = w_j - \alpha \frac{\delta}{\delta w_j} J_w\left(X, \vec{y}\right) \qquad \frac{\delta}{\delta w_j} J_w\left(X, \vec{y}\right) = \frac{1}{m} X^T \cdot \left(h_w(X) - \vec{y}\right)$$

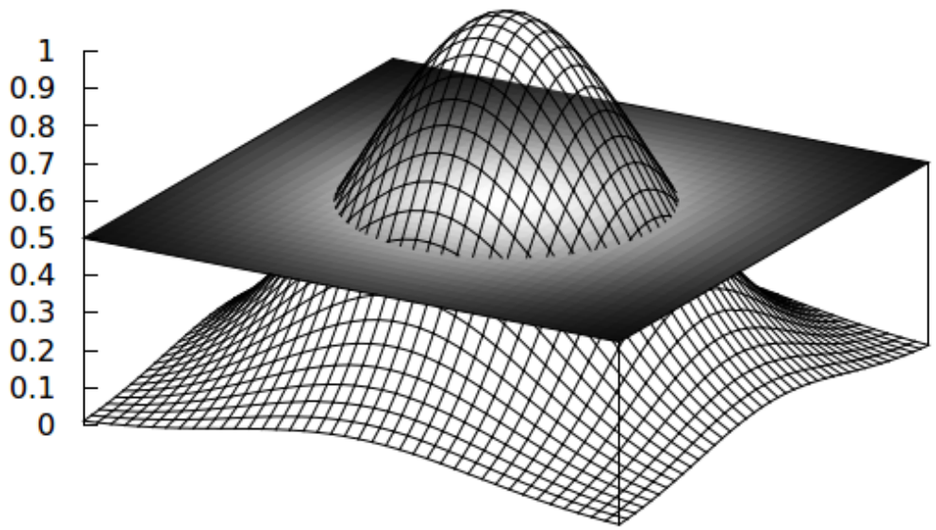
Geometric Interpretation

2D



$$y = w_0 + w_1 x_1$$

3D



$$y = w_0 + w_1 x_1 + w_2 x_2$$

nD

$$y = X\vec{w}$$

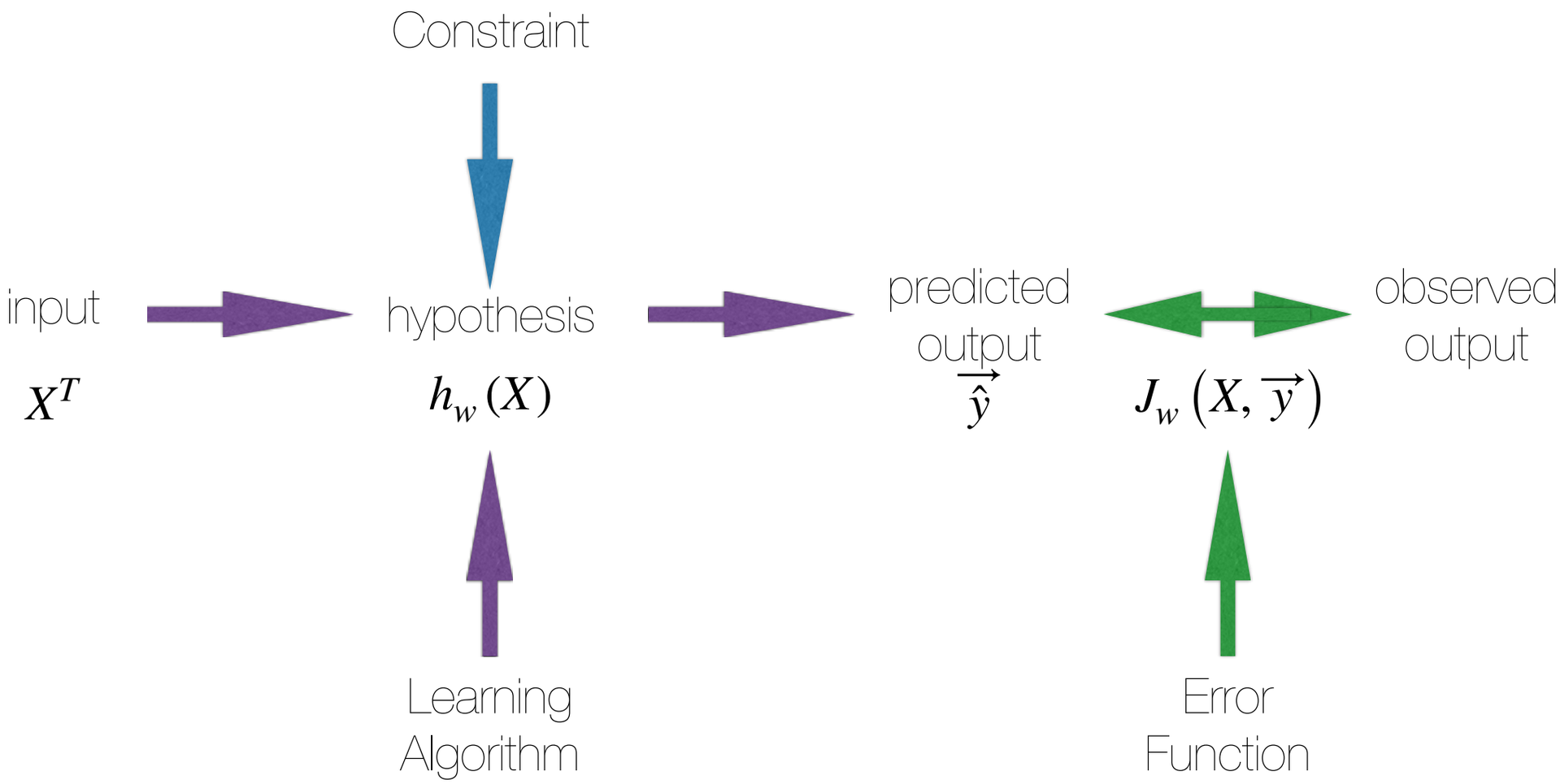
Finds the hyperplane that splits the points in two such that the errors on each side balance out

Add to account for intercept $x_0 \equiv 1$

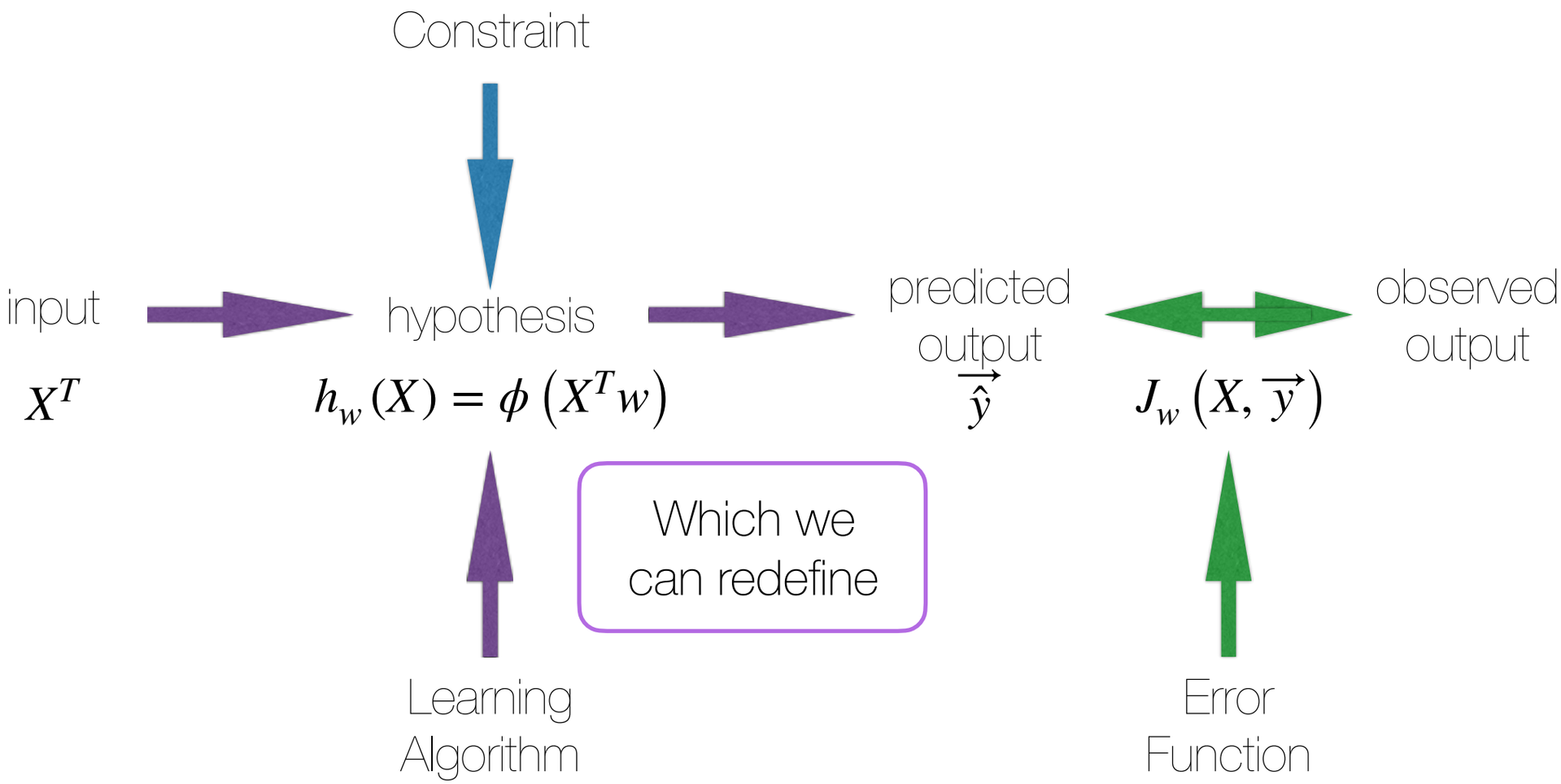
Code - Linear Regression

https://github.com/DataForScience/DeepLearning

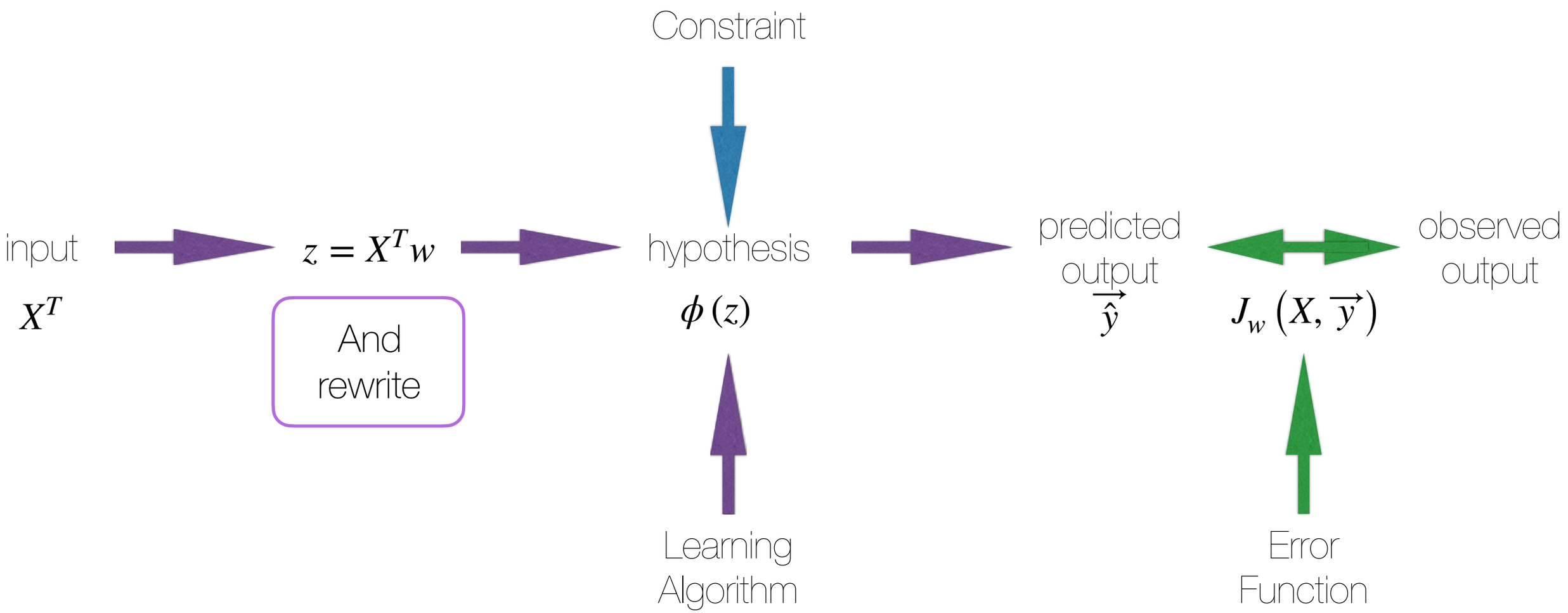
Learning Procedure



Learning Procedure



Learning Procedure



Logistic Regression (Classification)

- Not actually regression, but rather Classification

- Predict the probability of instance belonging to the given class:

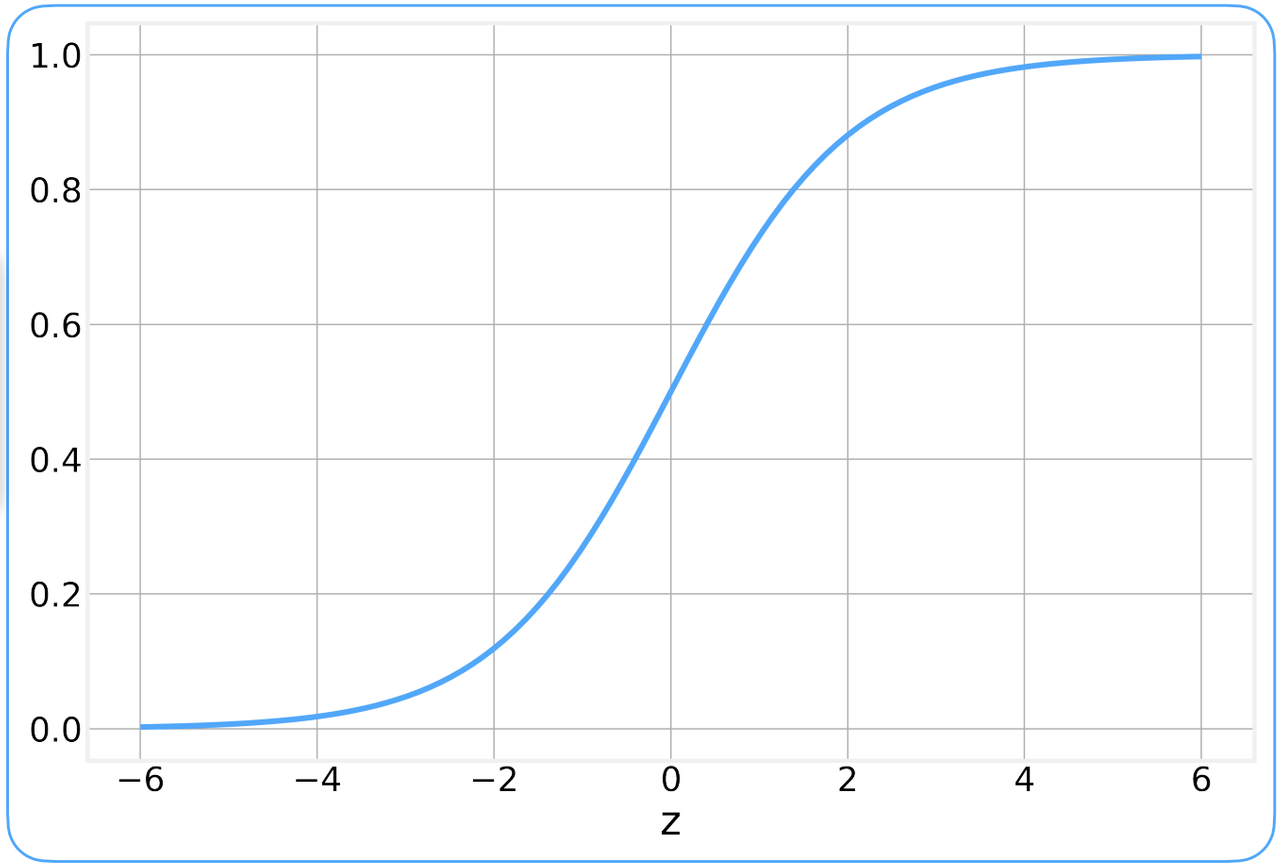
$$h_w(X) \in [0,1]$$

- Use sigmoid/logistic function to map weighted inputs to $[0,1]$

$$h_w(X) = \phi\left(X\vec{w}\right)$$

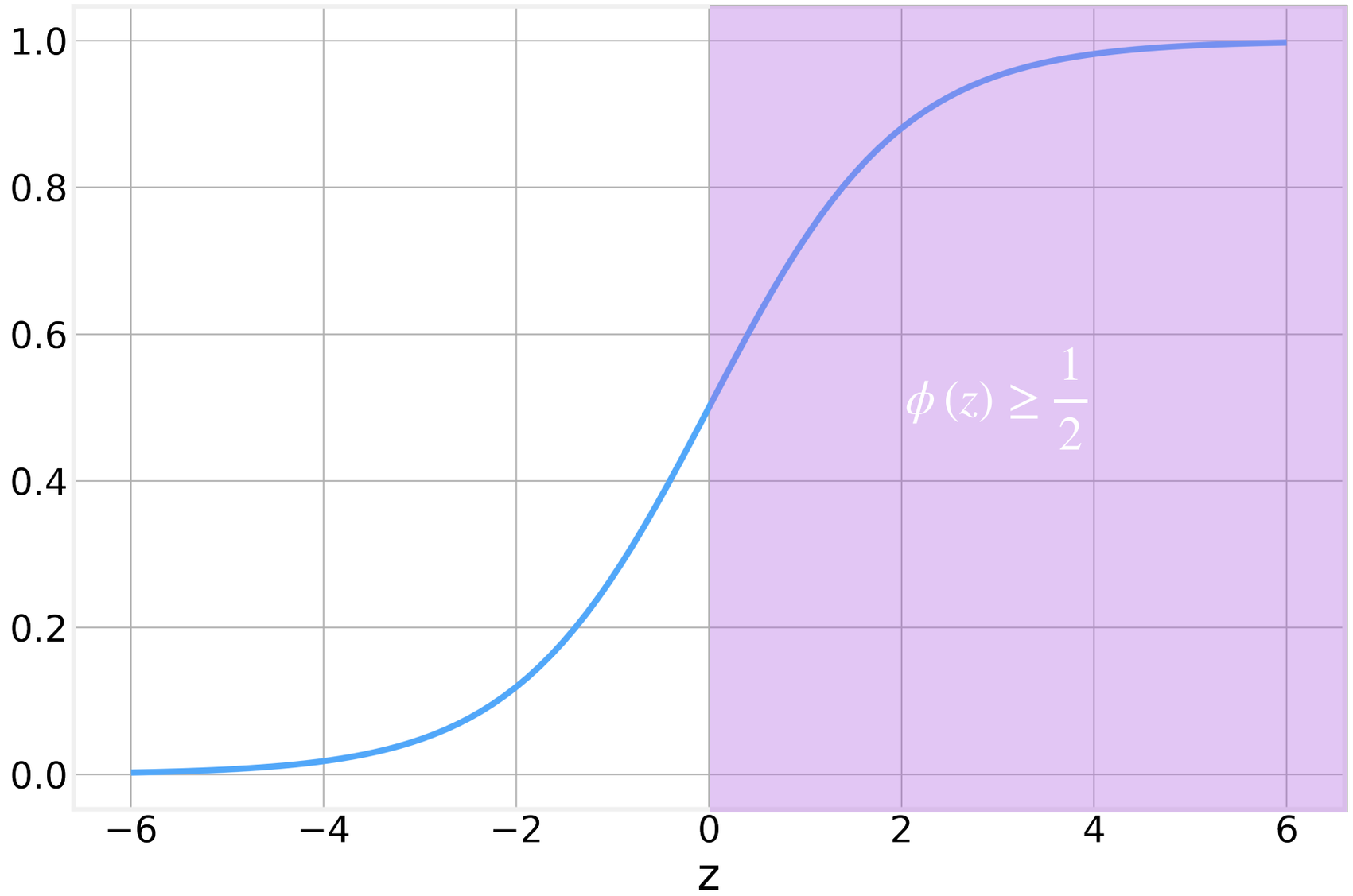
1 - part of the class
0 - otherwise

z encapsulates all the parameters and input values



Geometric Interpretation

maximize the value of z for members of the class



Logistic Regression

- Error Function - Cross Entropy

$$J_w\left(X, \overrightarrow{y}\right) = -\frac{1}{m}\left[y^T \log\left(h_w\left(X\right)\right) + \left(1-y\right)^T \log\left(1-h_w\left(X\right)\right)\right]$$

measures the "distance" between two probability distributions

$$h_w\left(X\right) = \frac{1}{1+e^{-X\overrightarrow{w}}}$$

- Effectively treating the labels as probabilities (an instance with label=1 has Probability 1 of belonging to the class).

- Gradient - same as Logistic Regression

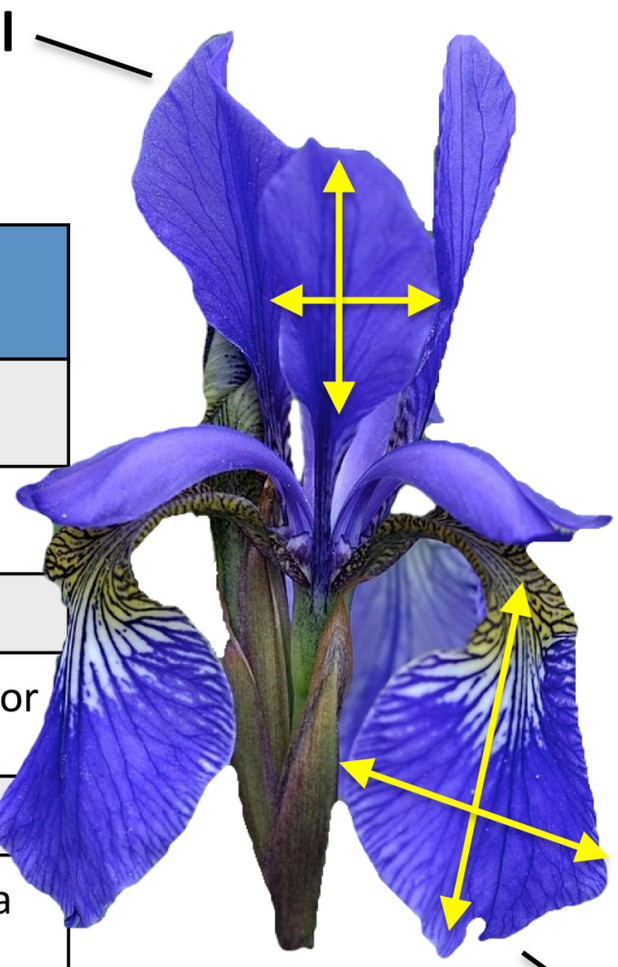
$$w_j = w_j - \alpha \frac{\delta}{\delta w_j} J_w\left(X, \overrightarrow{y}\right)$$

$$\frac{\delta}{\delta w_j} J_w\left(X, \overrightarrow{y}\right) = \frac{1}{m} X^T \cdot \left(h_w\left(X\right) - \overrightarrow{y}\right)$$

Iris dataset

Samples
(instances, observations)

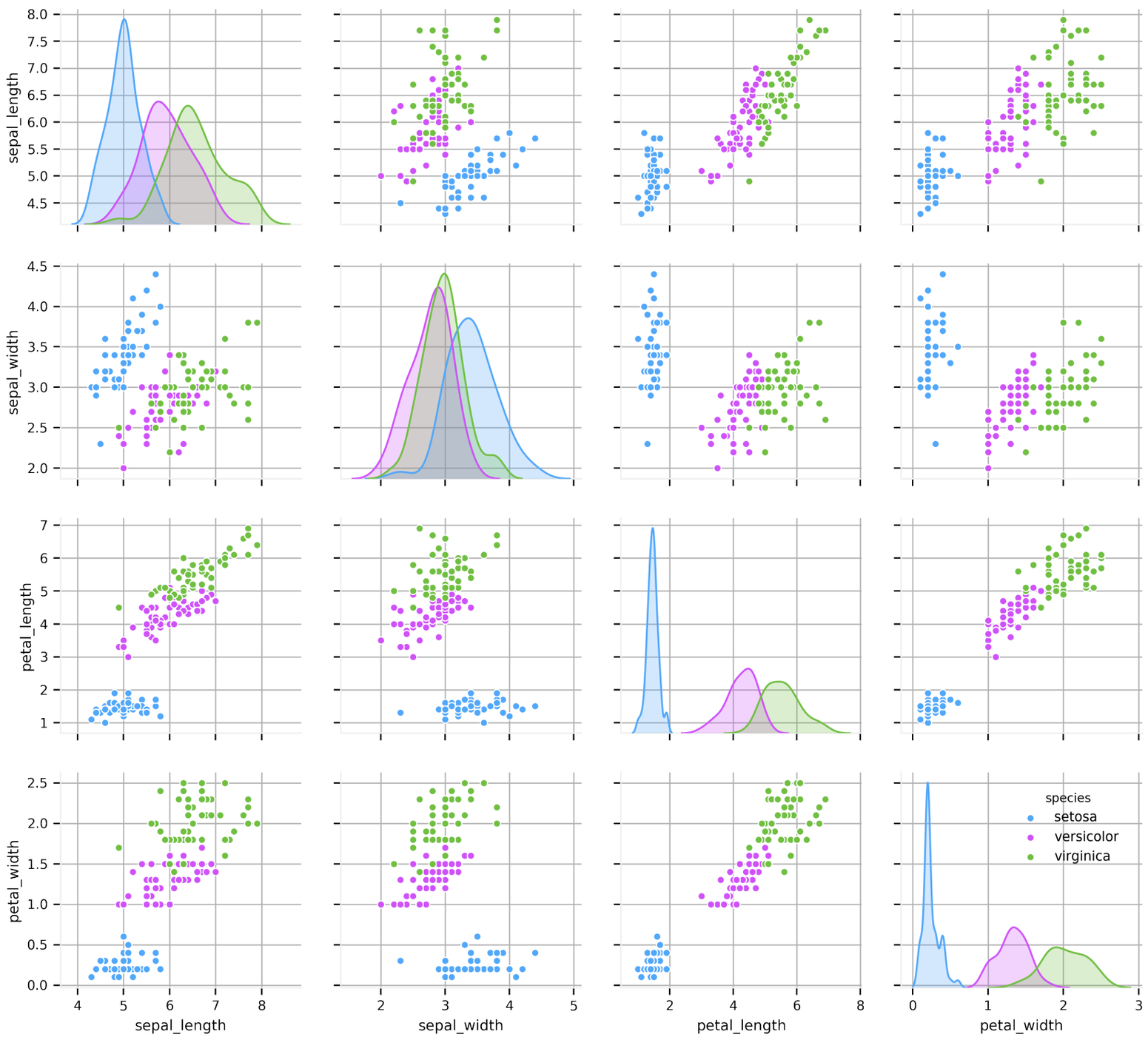
	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica



Features
(attributes, measurements, dimensions)

Class labels
(targets)

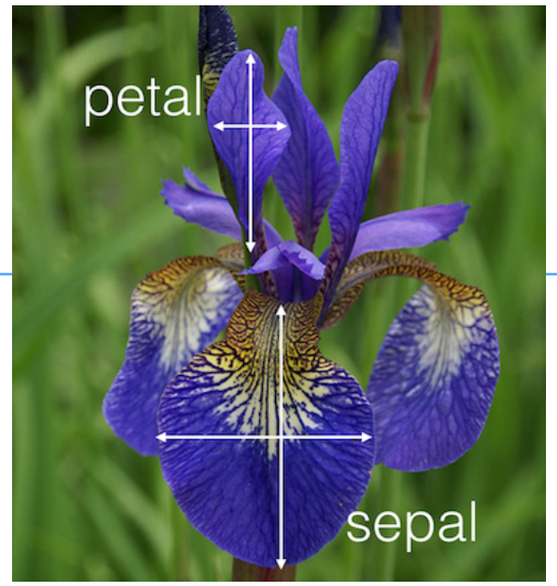
Iris dataset

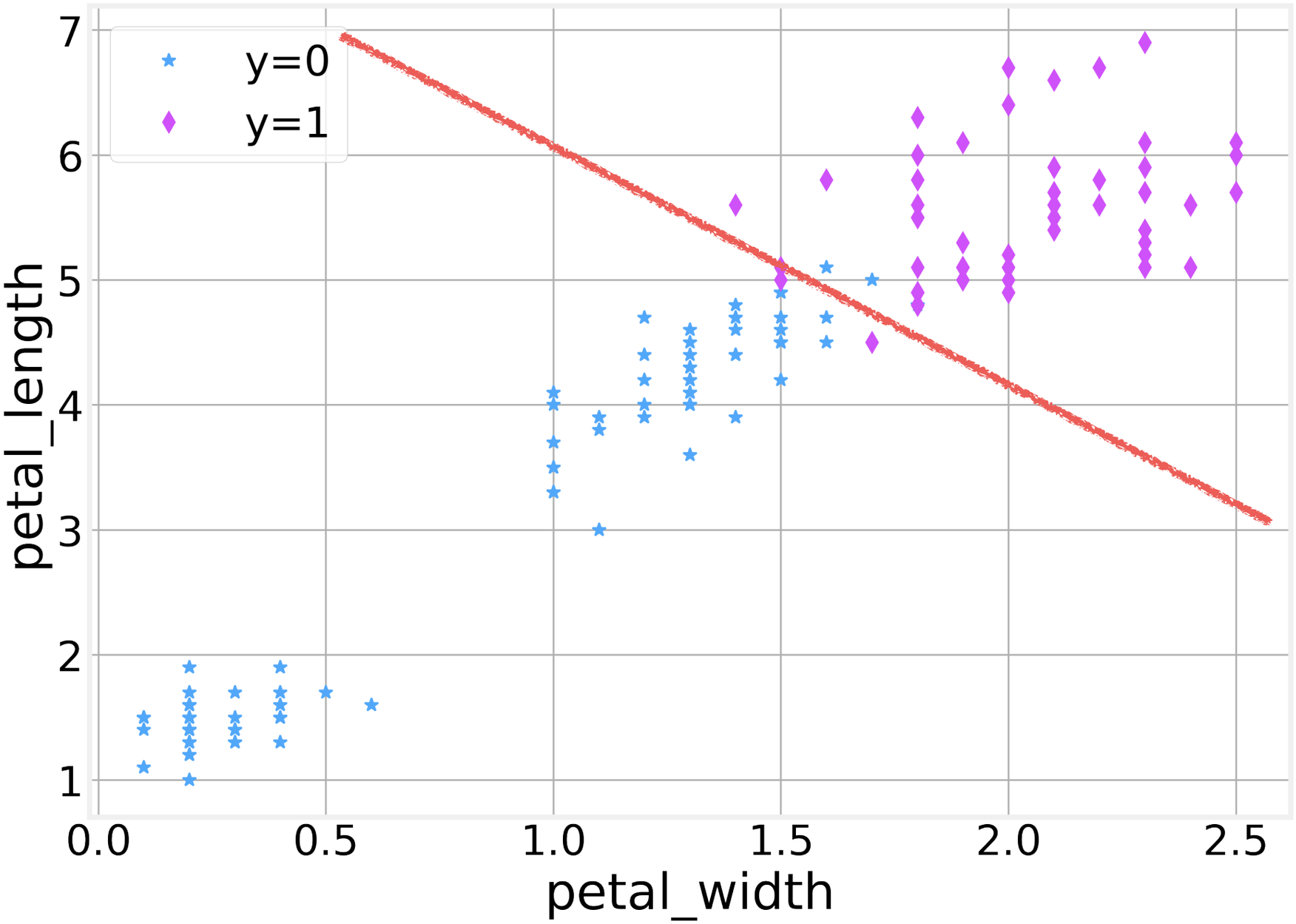


Code - Logistic Regression

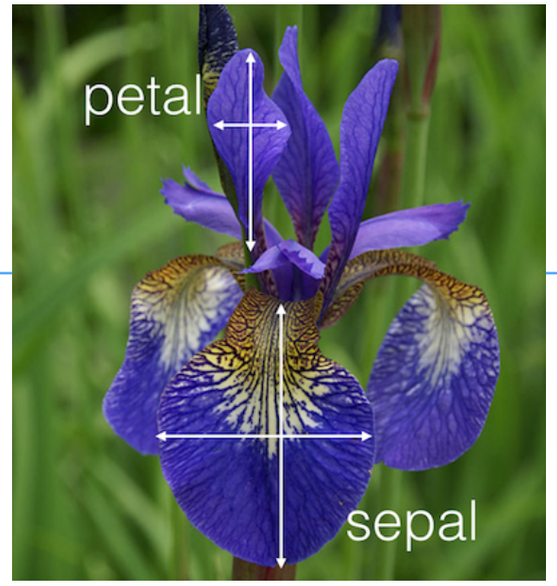
https://github.com/DataForScience/DeepLearning

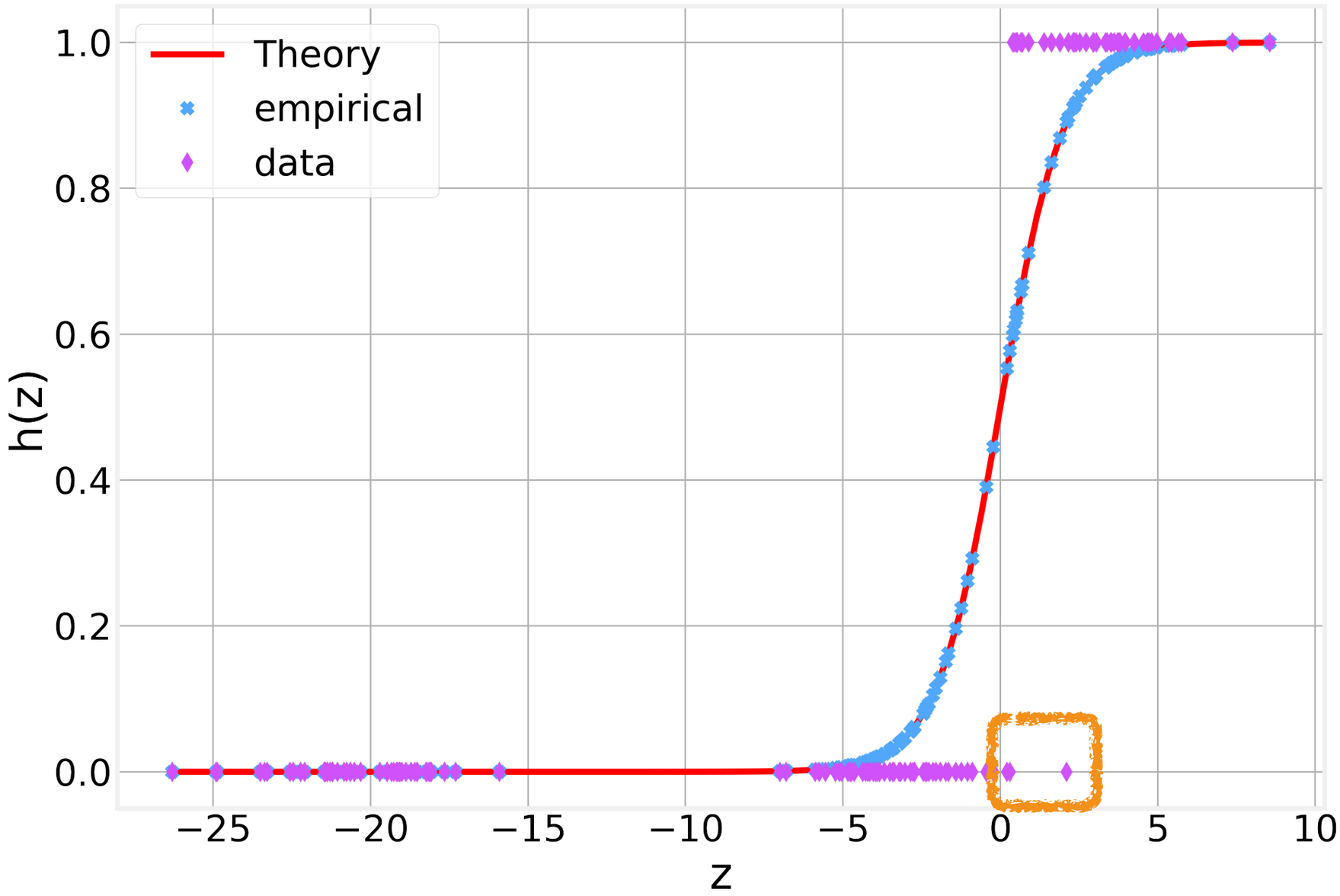
Logistic Regression



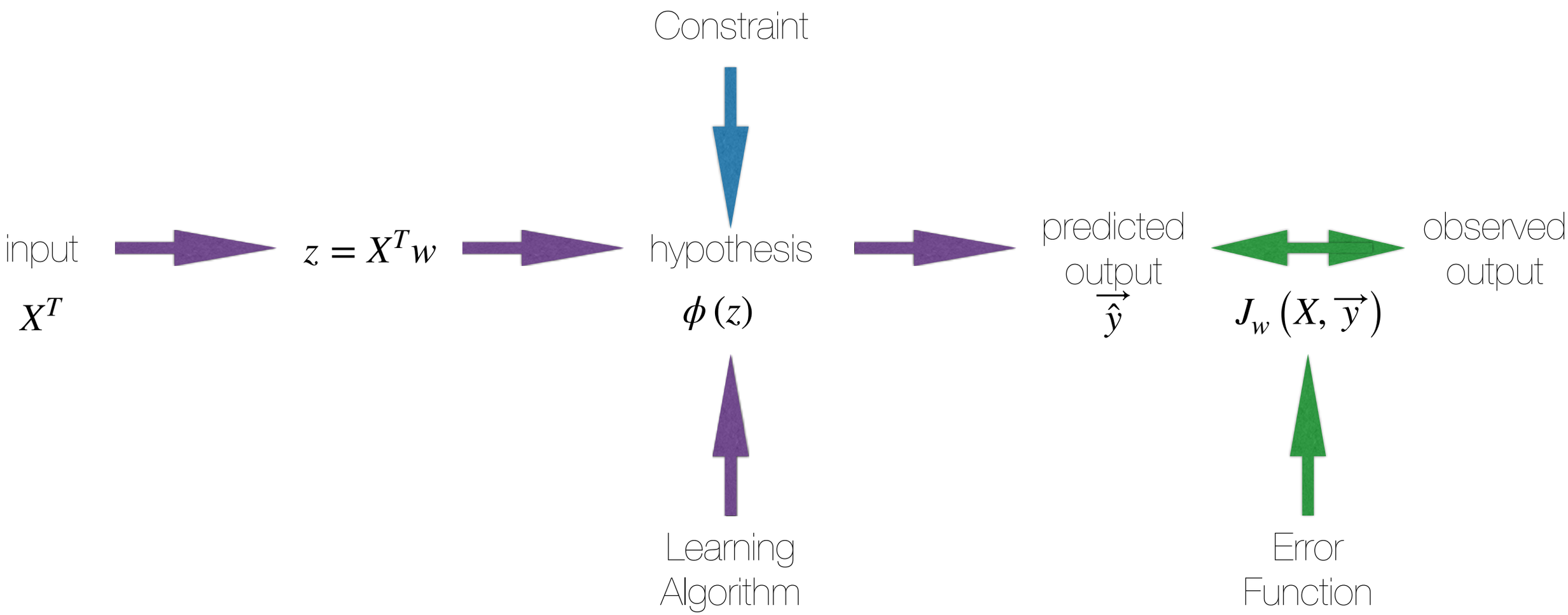


Logistic Regression





Learning Procedure



Comparison

- Linear Regression

$$z = X\vec{w}$$

$$h_w(X) = \phi(Z)$$

$$\phi(Z) = Z$$

$$J_w\left(X, \vec{y}\right) = \frac{1}{2m}\left[h_w(X) - \vec{y}\right]^2$$

$$\frac{\delta}{\delta w_j} J_w\left(X, \vec{y}\right) = \frac{1}{m} X^T \cdot \left(h_w(X) - \vec{y}\right)$$

- Logistic Regression

$$z = X\vec{w}$$

$$h_w(X) = \phi(Z)$$

$$\phi(Z) = \frac{1}{1 + e^{-Z}}$$

$$J_w\left(X, \vec{y}\right) = -\frac{1}{m}\left[y^T \log\left(h_w(X)\right) + \left(1 - y\right)^T \log\left(1 - h_w(X)\right)\right]$$

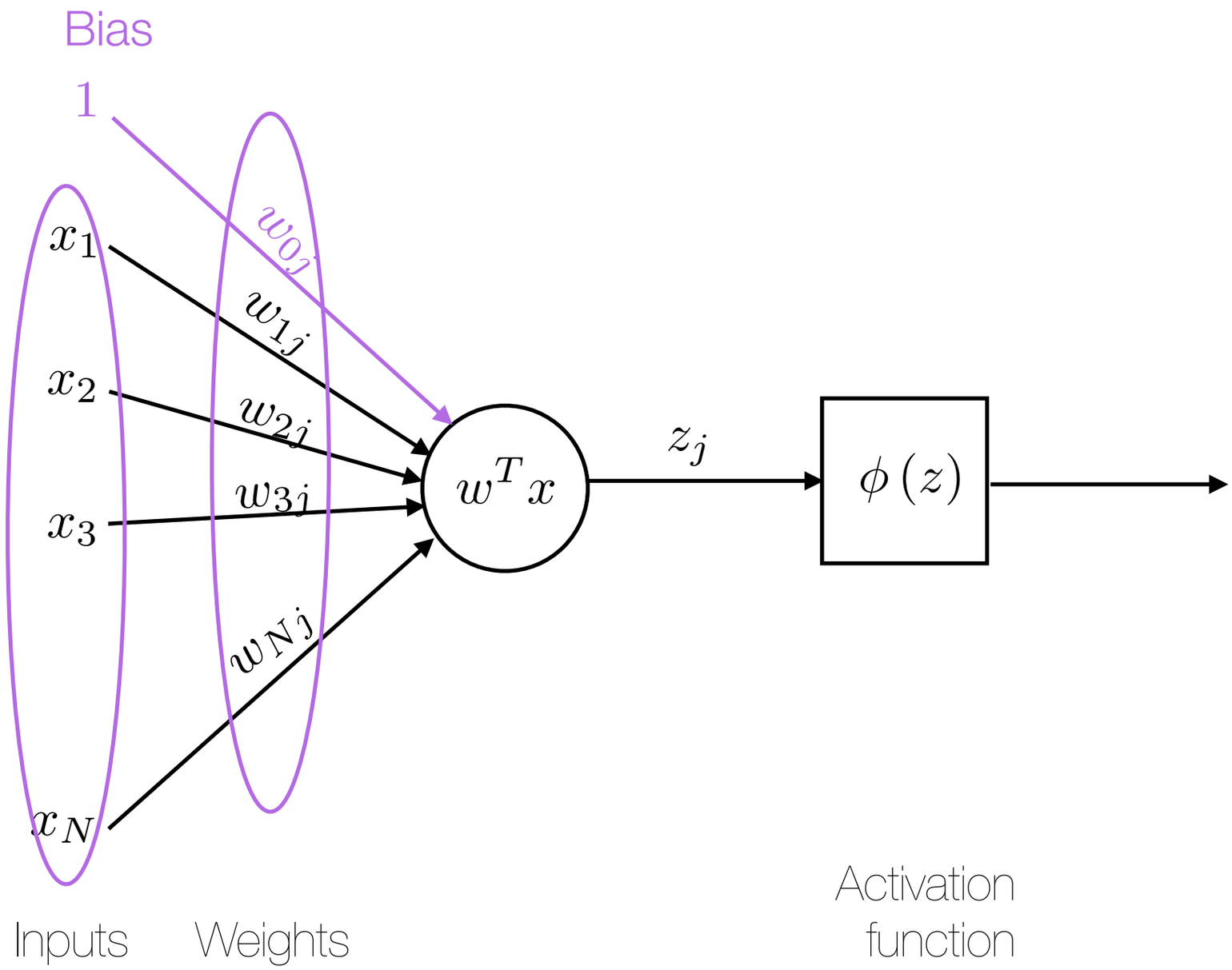
$$\frac{\delta}{\delta w_j} J_w\left(X, \vec{y}\right) = \frac{1}{m} X^T \cdot \left(h_w(X) - \vec{y}\right)$$

Map features to a continuous variable

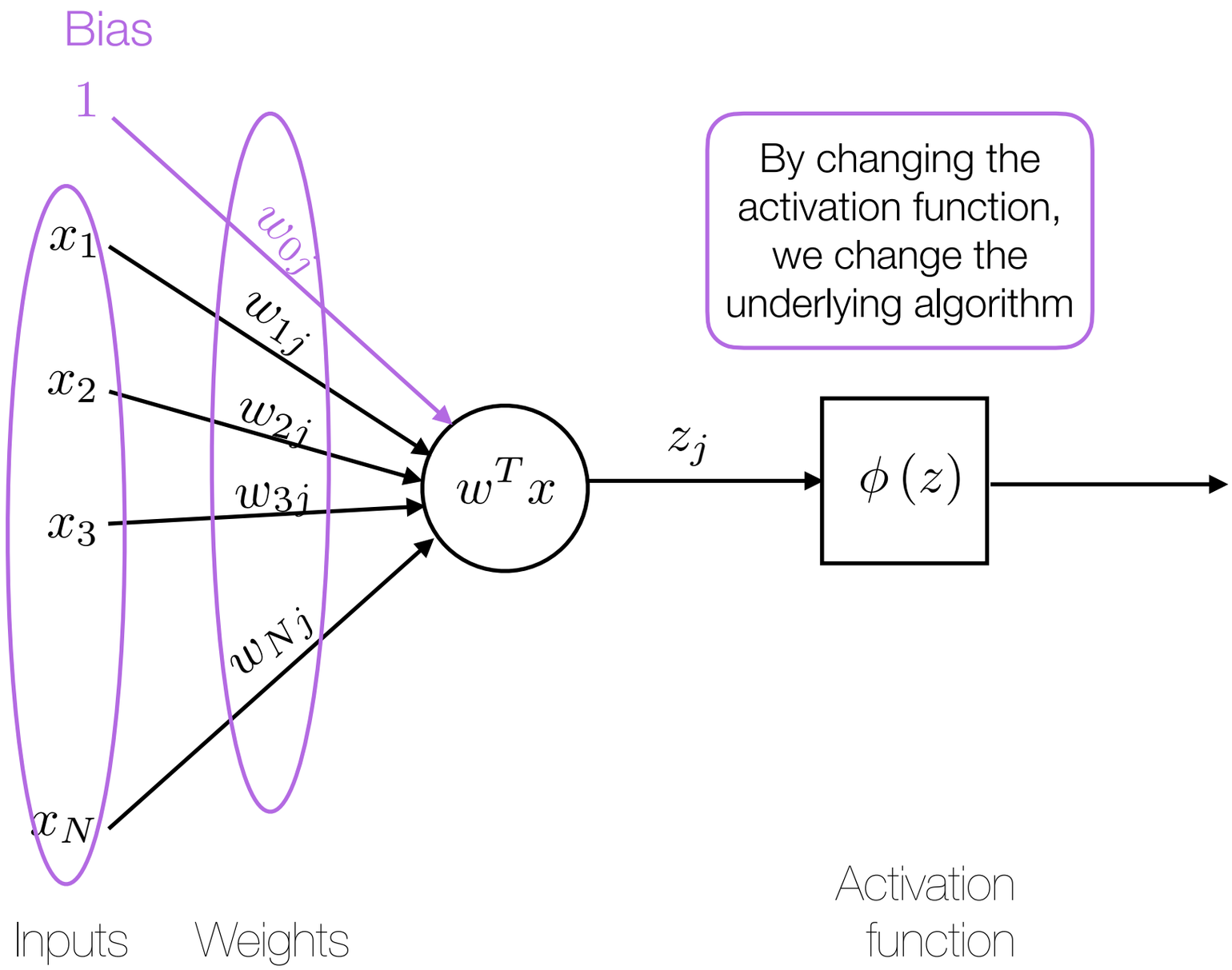
Predict based on continuous variable

Compare prediction with reality

Learning Procedure



Generalized Perceptron



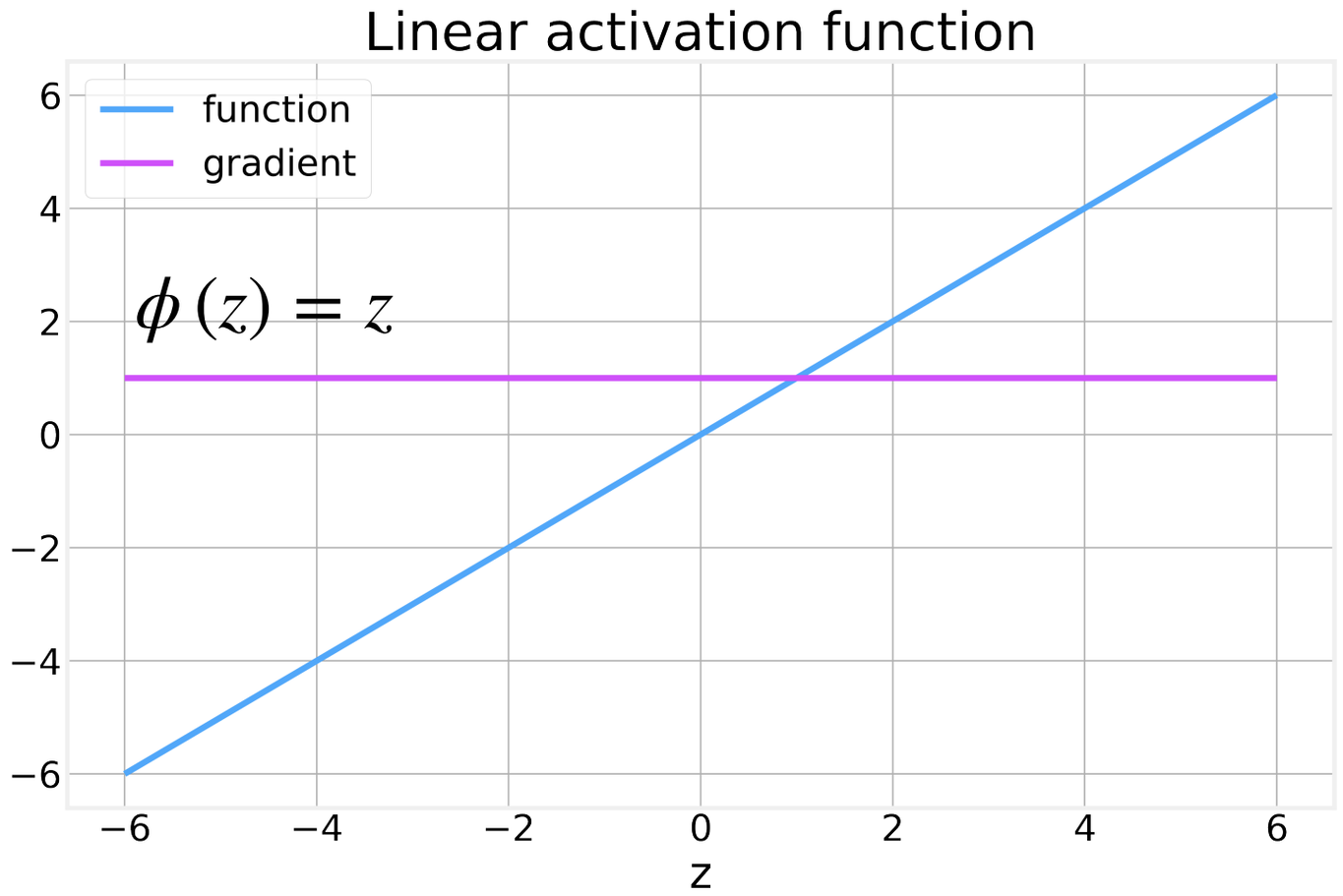
By changing the activation function, we change the underlying algorithm

Activation Function

- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features
- Each layer builds up a more abstract representation of the data

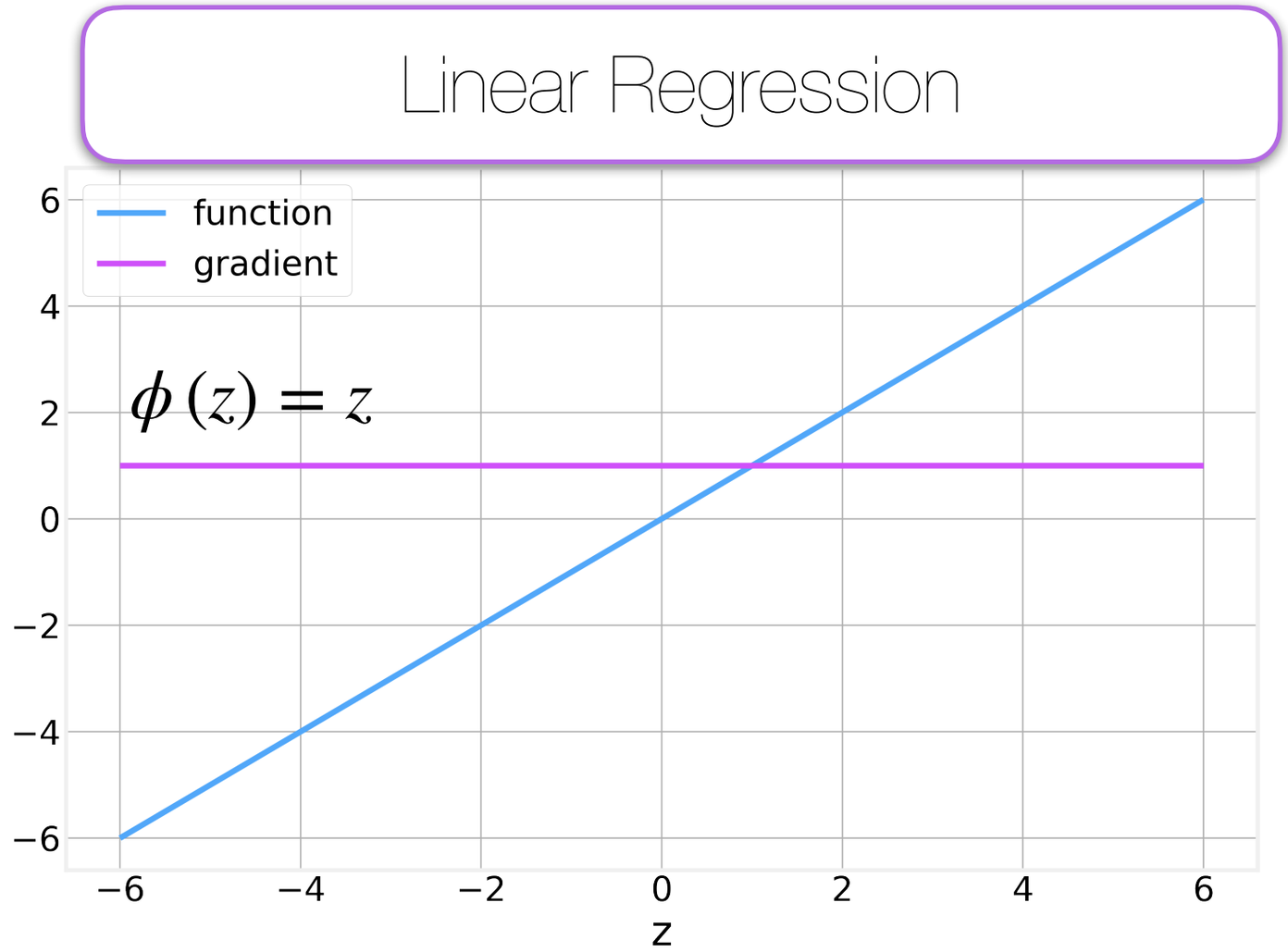
Activation Function - Linear

- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features
- Each layer builds up a more abstract representation of the data
- The simplest



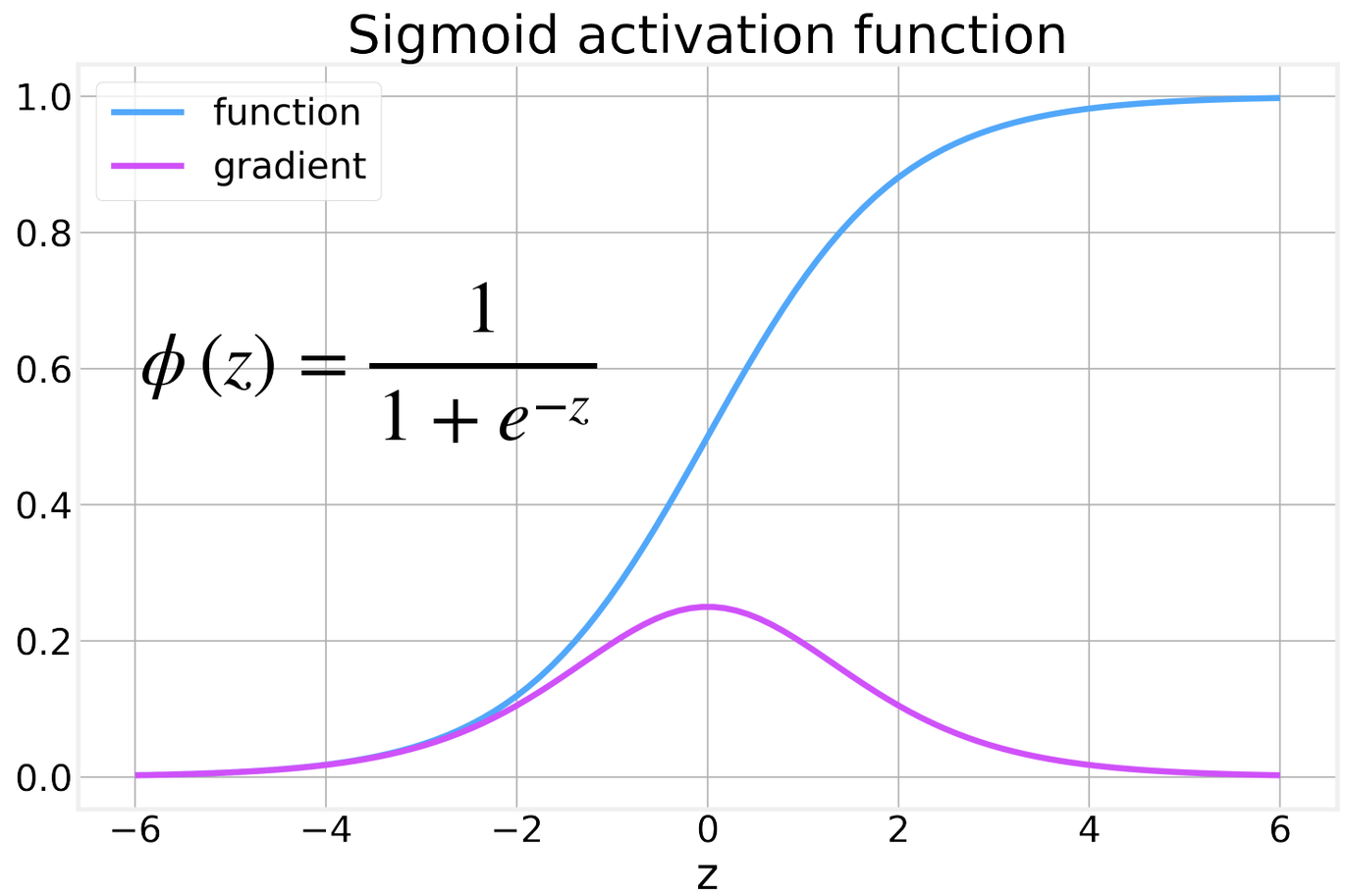
Activation Function - Linear

- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features
- Each layer builds up a more abstract representation of the data
- The simplest



Activation Function - Sigmoid

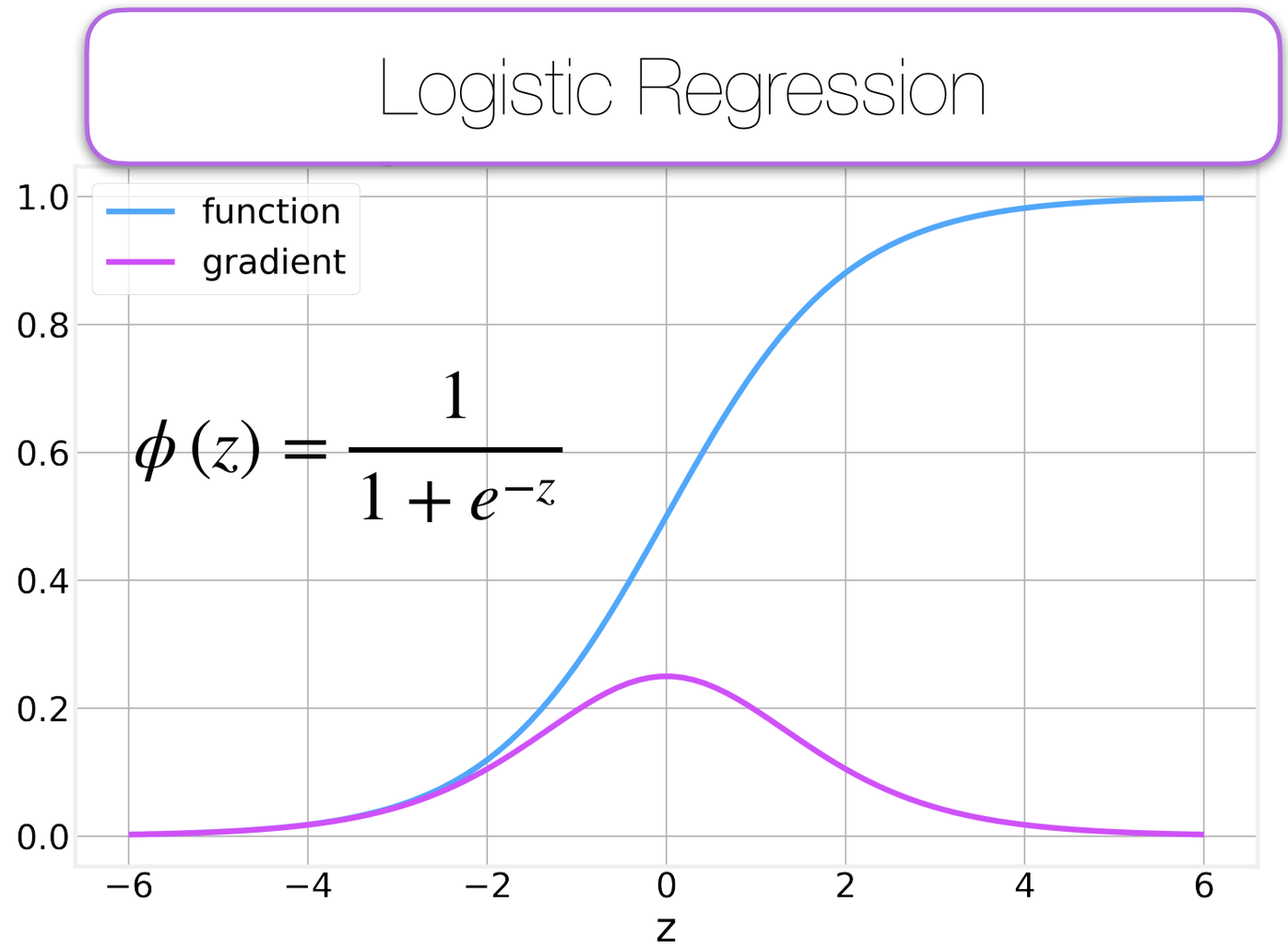
- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features
- Each layer builds up a more abstract representation of the data
- Perhaps the most common



Activation Function - Sigmoid

- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features
- Each layer builds up a more abstract representation of the data
- Perhaps the most common

Logistic Regression



Forward Propagation

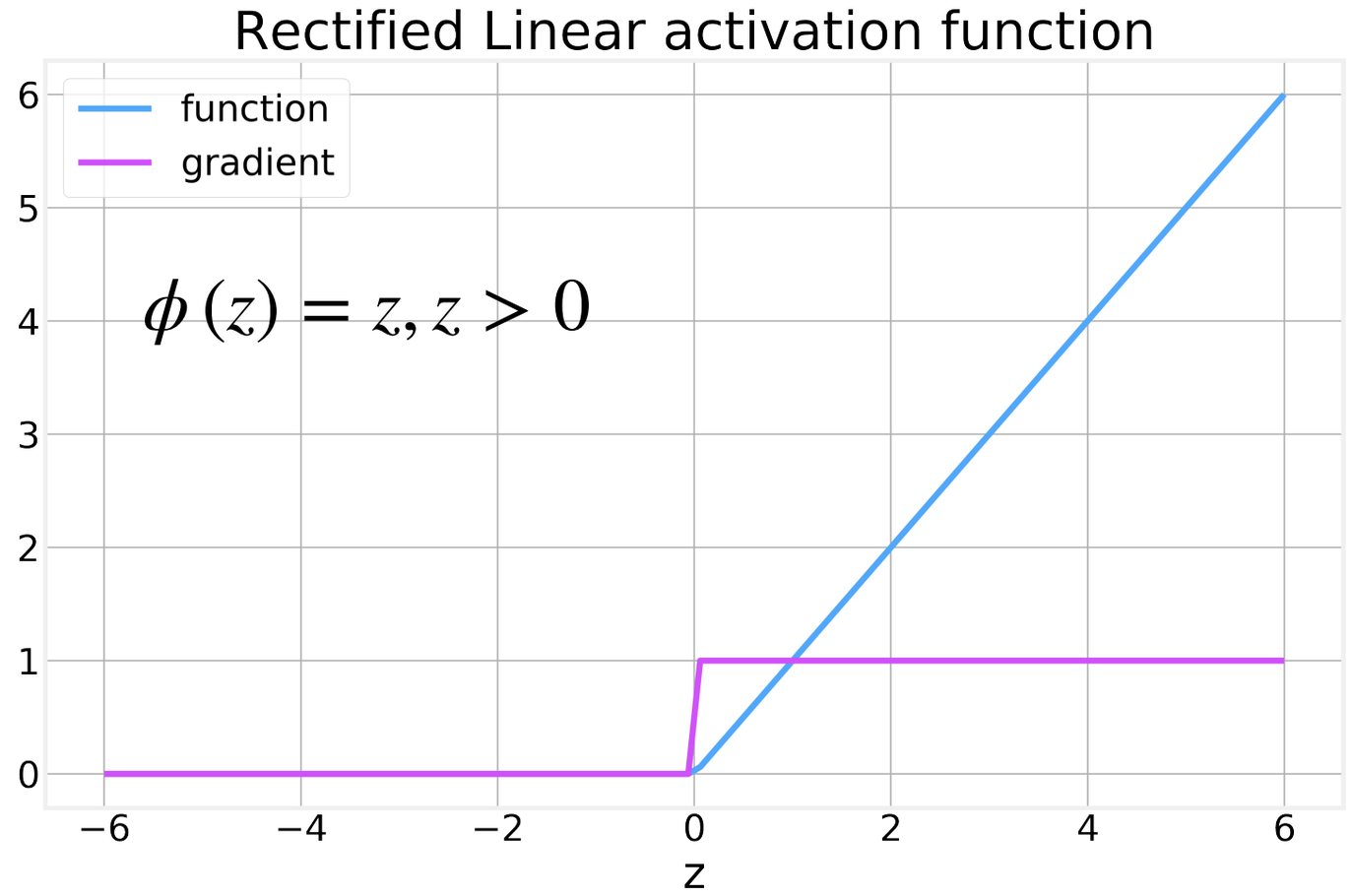
- The output of a perceptron is determined by a sequence of steps:
    - obtain the inputs
    - multiply the inputs by the respective weights
    - calculate output using the activation function

Code - Forward Propagation

https://github.com/DataForScience/DeepLearning

Activation Function - ReLu

- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features
- Each layer builds up a more abstract representation of the data
- Results in faster learning than with sigmoid

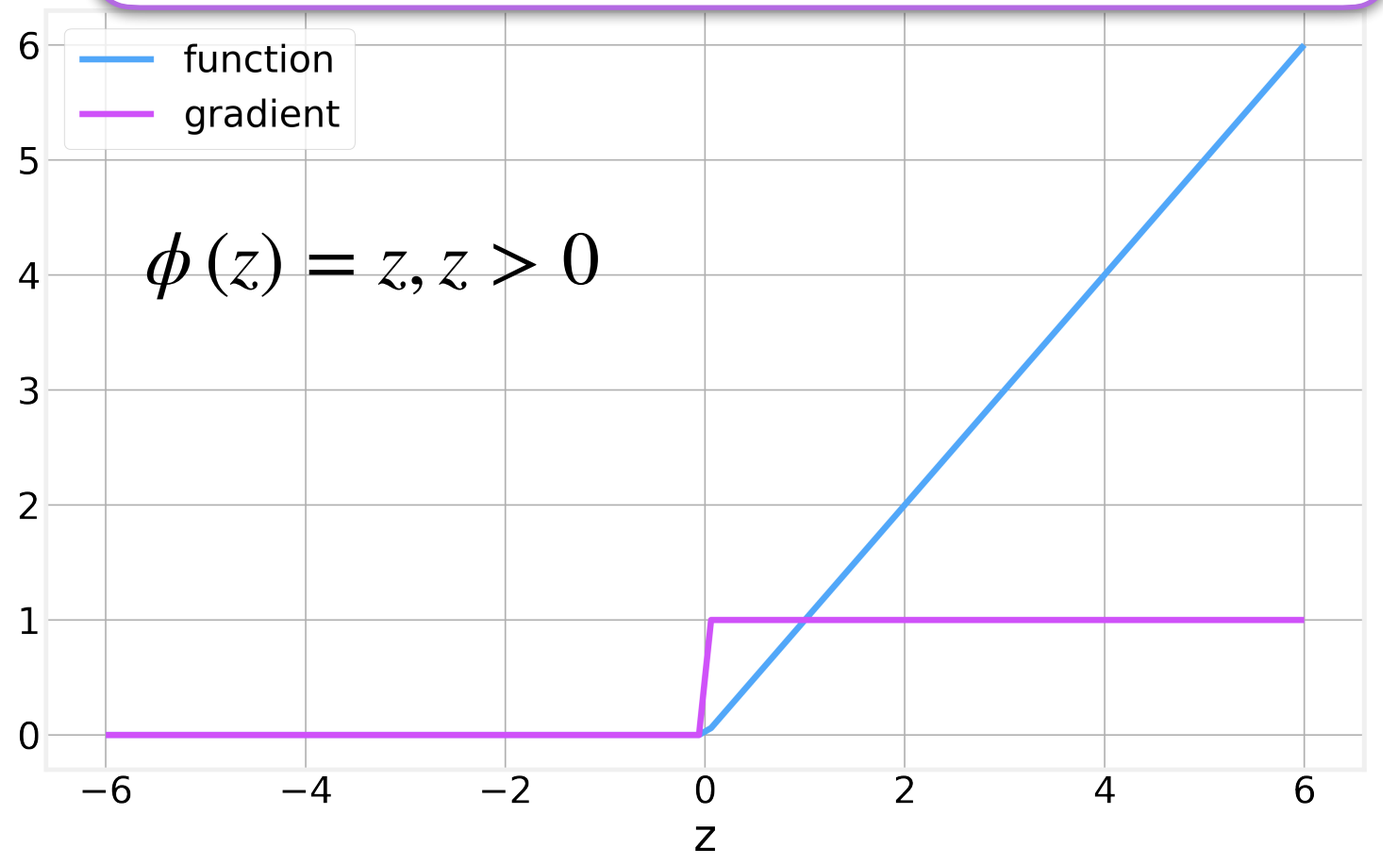


Activation Function - ReLu

- Non-Linear function
- Differentiable
- non-decreasing

- Compute new sets of features
- Each layer builds up a more abstract representation of the data
- Results in faster learning than with sigmoid

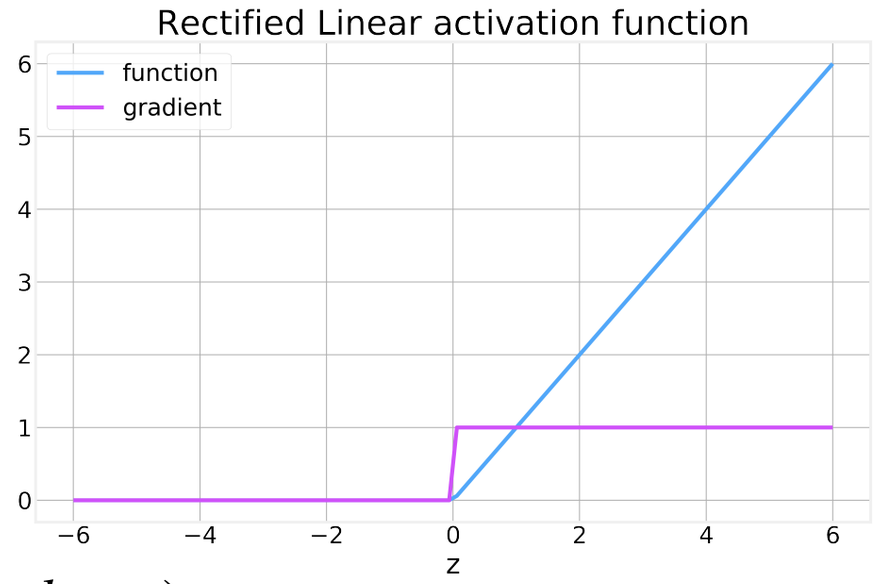
Stepwise Regression



Stepwise Regression

https://en.wikipedia.org/wiki/Multivariate_adaptive_regression_spline

- Multivariate Adaptive Regression Spline (MARS) is the best known example
- Fit curves using a linear combination of: $\hat{f}(x) = \sum_i c_i B_i(x)$
- The basis functions $B_i(x)$ can be:
  - Constant
  - "Hinge" functions of the form: $max(0, x - b)$ and $max(0, b - x)$
  - Products of hinges



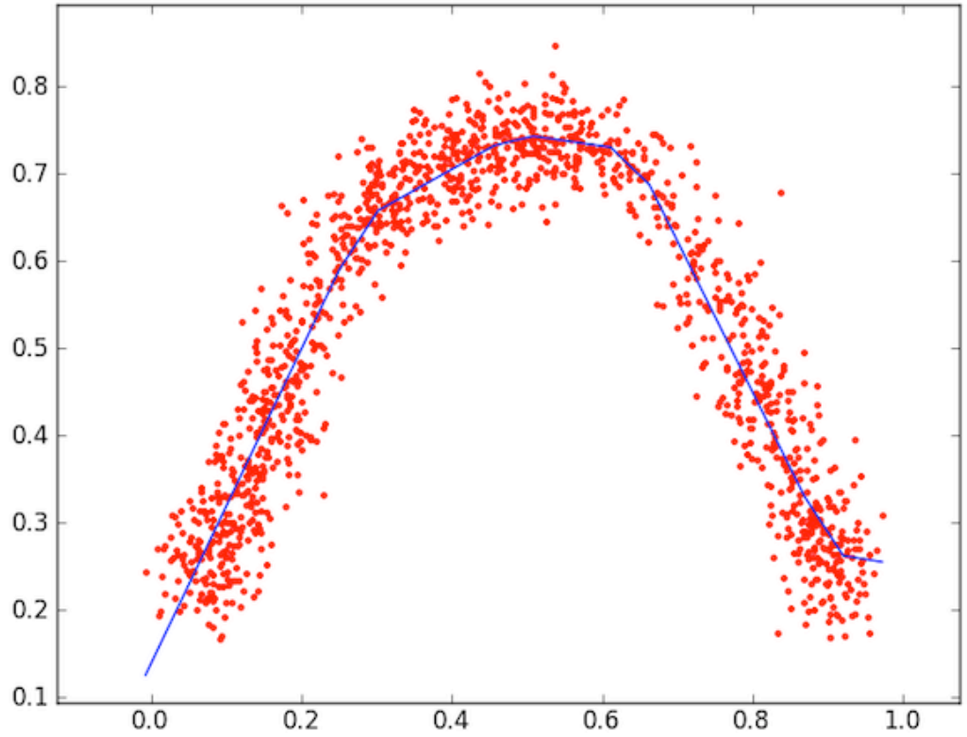
Stepwise Regression

https://en.wikipedia.org/wiki/Multivariate_adaptive_regression_spline

- Multivariate Adaptive Regression Spline (MARS) is the best known example

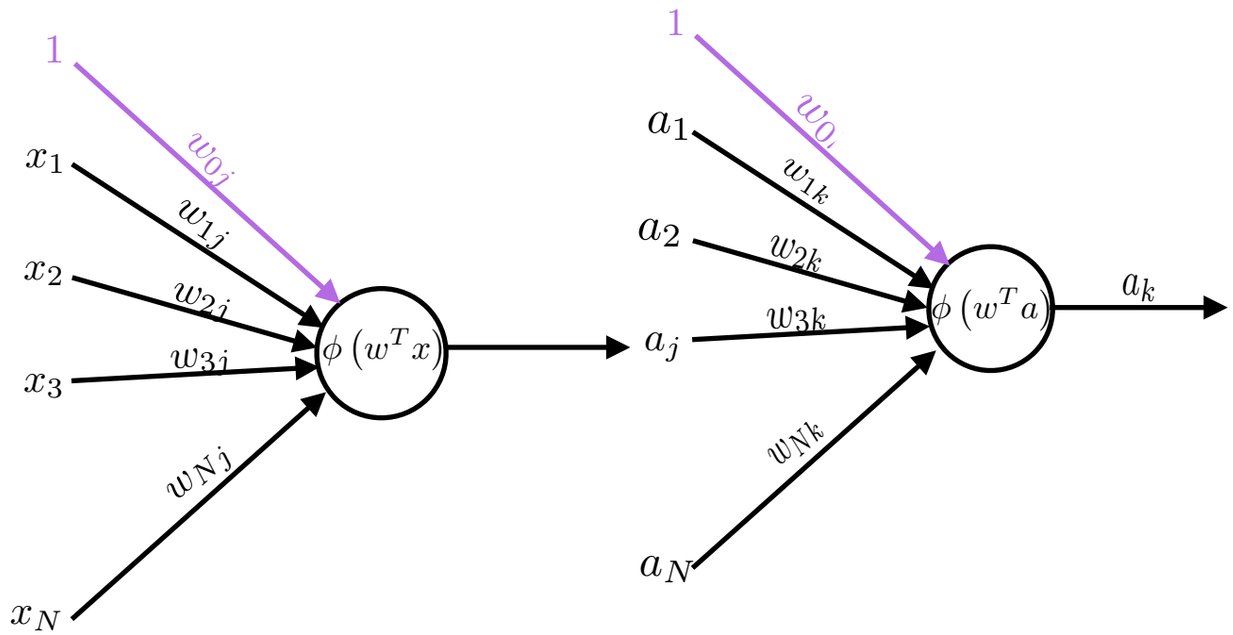
- Fit curves using a linear combination of: $\hat{f}(x) = \sum_i c_i B_i(x)$

$$
\begin{aligned}
y(x) &= 1.013 \\
&+ 1.198\ max(0, x - 0.485) \\
&- 1.803\ max(0, 0.485 - x) \\
&- 1.321\ max(0, x - 0.283) \\
&- 1.609\ max(0, x - 0.640) \\
&+ 1.591\ max(0, x - 0.907)
\end{aligned}
$$



Forward Propagation

- The output of a perceptron is determined by a sequence of steps:
  - obtain the inputs
  - multiply the inputs by the respective weights
  - calculate output using the activation function
- To create a multi-layer perceptron, you can simply use the output of one layer as the input to the next one.



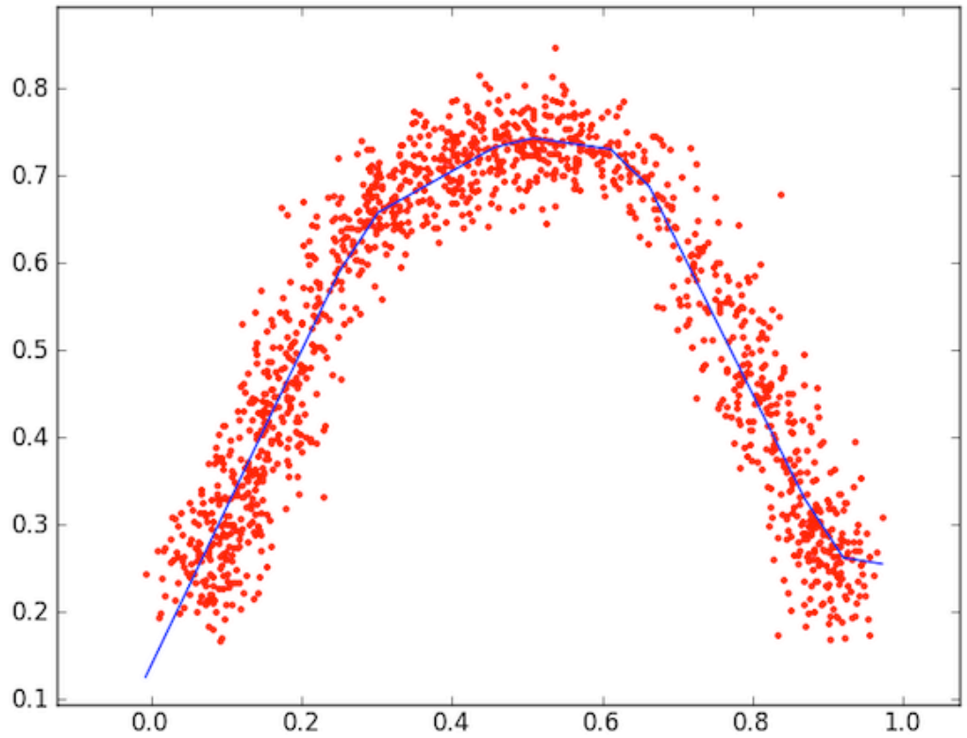
- But how can we propagate back the errors and update the weights?

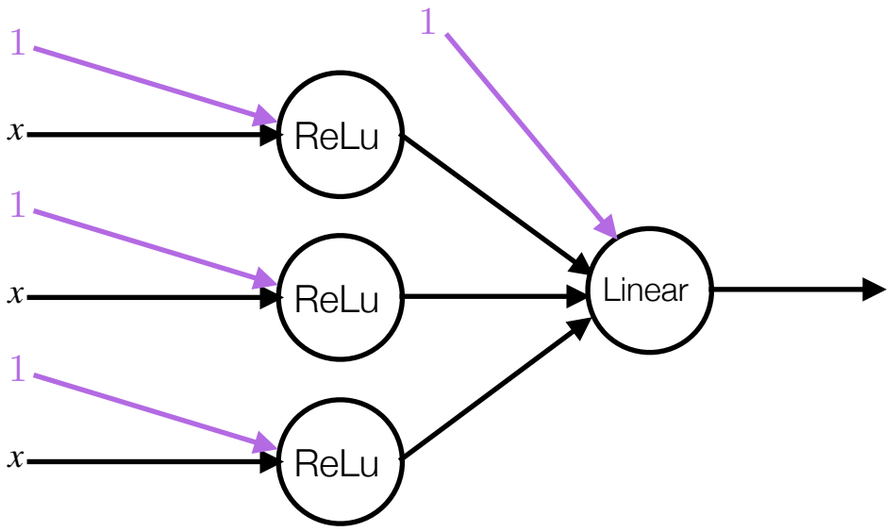
Stepwise Regression

https://en.wikipedia.org/wiki/Multivariate_adaptive_regression_spline

$$\hat{f}(x) = \sum_i c_i B_i(x)$$

$$\begin{aligned} y(x) &= 1.013 \\ &+ 1.198\ max(0, x - 0.485) \\ &- 1.803\ max(0, 0.485 - x) \\ &- 1.321\ max(0, x - 0.283) \\ &- 1.609\ max(0, x - 0.640) \\ &+ 1.591\ max(0, x - 0.907) \end{aligned}$$





Loss Functions

- For learning to occur, we must quantify how far off we are from the desired output. There are two common ways of doing this:

  - Quadratic error function:

$$E = \frac{1}{N} \sum_n |y_n - a_n|^2$$

  - Cross Entropy

$$J = -\frac{1}{N} \sum_n \left[y_n^T \log a_n + (1 - y_n)^T \log (1 - a_n) \right]$$

The Cross Entropy is complementary to sigmoid activation in the output layer and improves its stability.

Regularization

- Helps keep weights relatively small by adding a penalization to the cost function.

- Two common choices:

$$\hat{J}_w(X) = J_w(X) + \lambda \sum_{ij} \left| w_{ij} \right| \qquad \text{"Lasso"}$$

$$\hat{J}_w(X) = J_w(X) + \lambda \sum_{ij} w_{ij}^2 \qquad \text{L2}$$

- Lasso helps with feature selection by driving less important weights to zero

Backward Propagation of Errors (BackProp)

- BackProp operates in two phases:
    - Forward propagate the inputs and calculate the deltas
    - Update the weights
- The error at the output is a weighted average difference between predicted output and the observed one.
- For inner layers there is no “real output”!

BackProp

- Let $\delta^{(l)}$ be the error at each of the total L layers:

- Then:

$$\delta^{(L)} = h_w(X) - y$$

- And for every other layer, in reverse order:

$$\delta^{(l)} = W^{(l)T}\delta^{(l+1)} .* \phi^{\dagger}\left(z^{(l)}\right)$$

- Until:

$$\delta^{(1)} \equiv 0$$

as there's no error on the input layer.

- And finally:

$$\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)}\delta_i^{(l+1)}$$

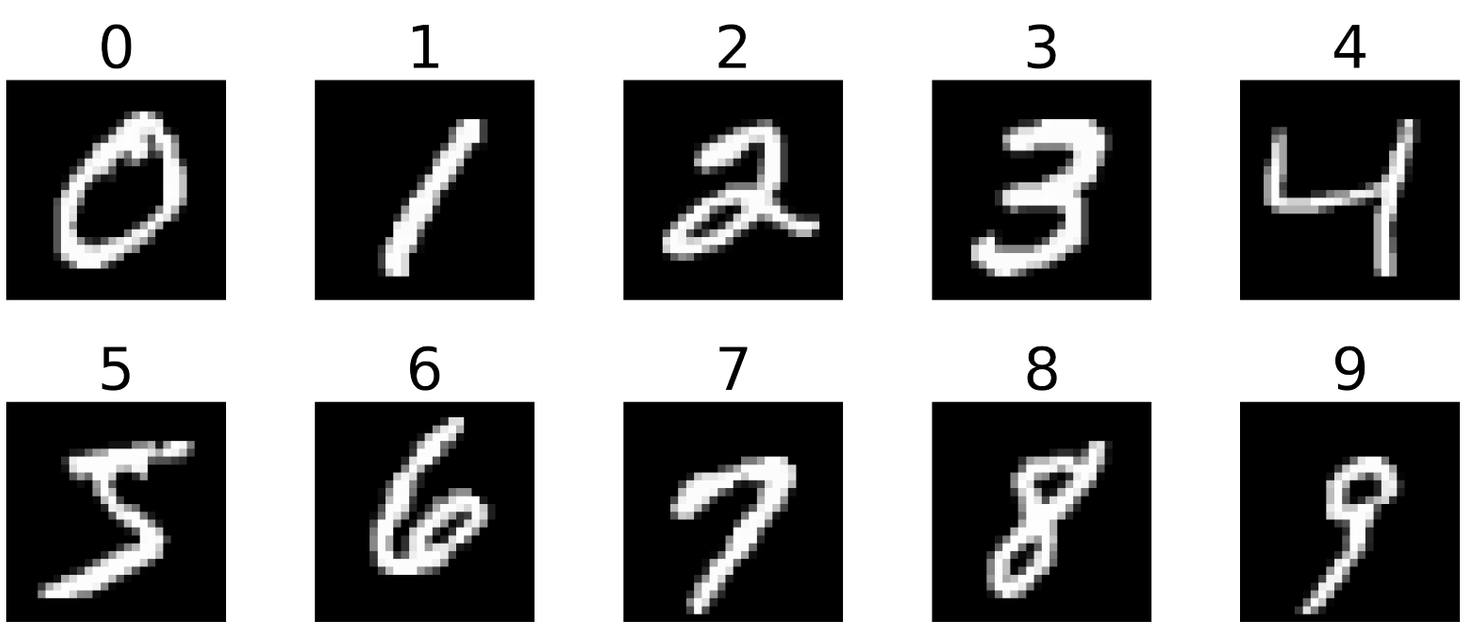
$$\frac{\partial}{\partial w_{ij}^{(l)}} J_w\left(X, \overrightarrow{y}\right) = \frac{1}{m}\Delta_{ij}^{(l)} + \lambda w_{ij}^{(l)}$$

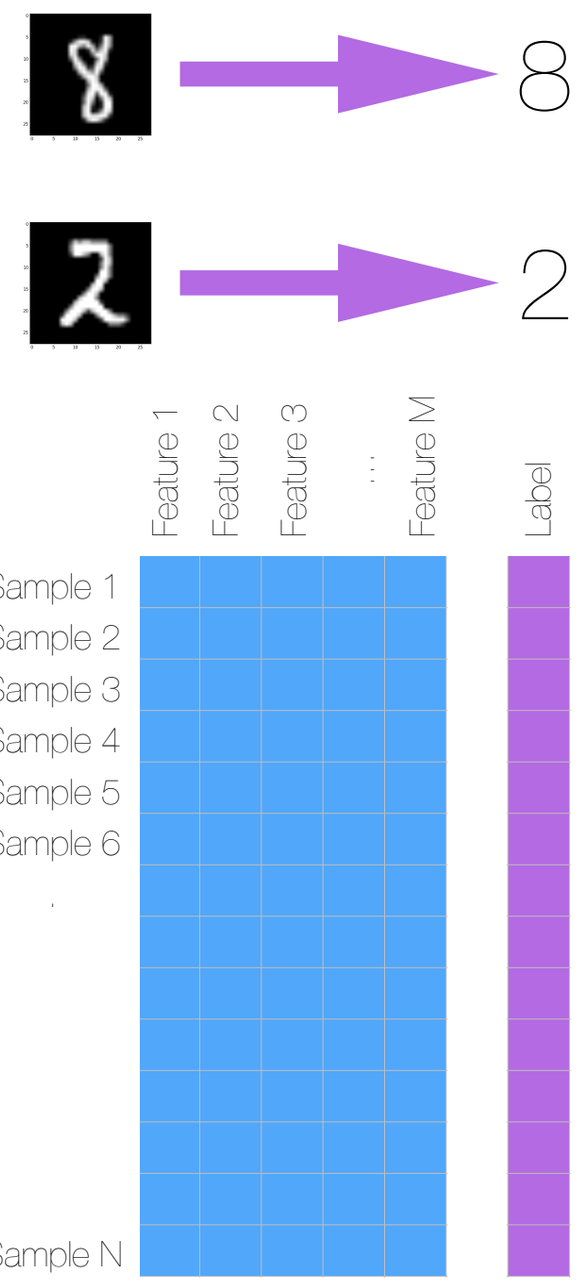
A practical example - MNIST

THE MNIST DATABASE

of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond





yann.lecun.com/exdb/mnist/

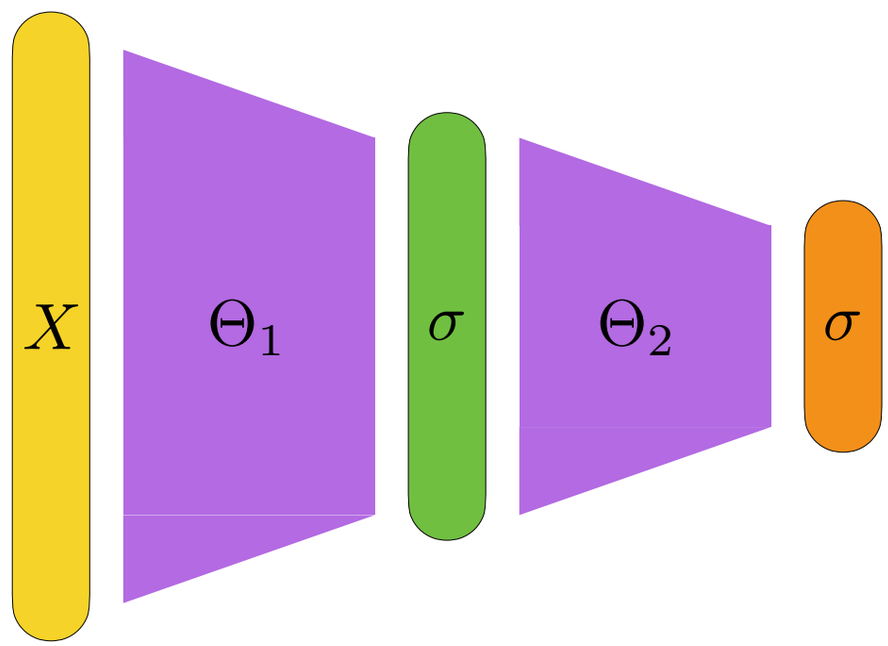
A practical example - MNIST

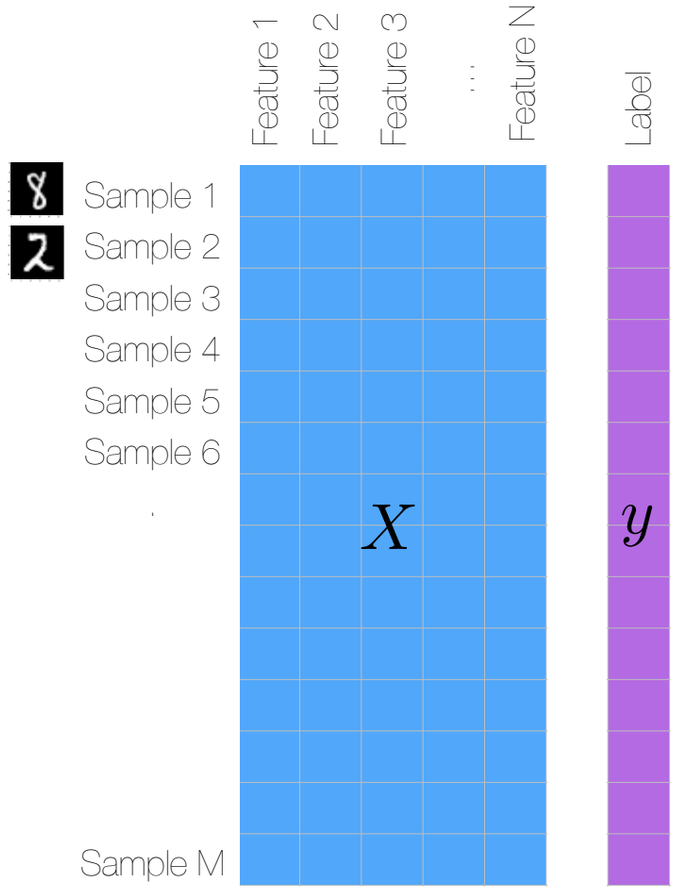
THE MNIST DATABASE

of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

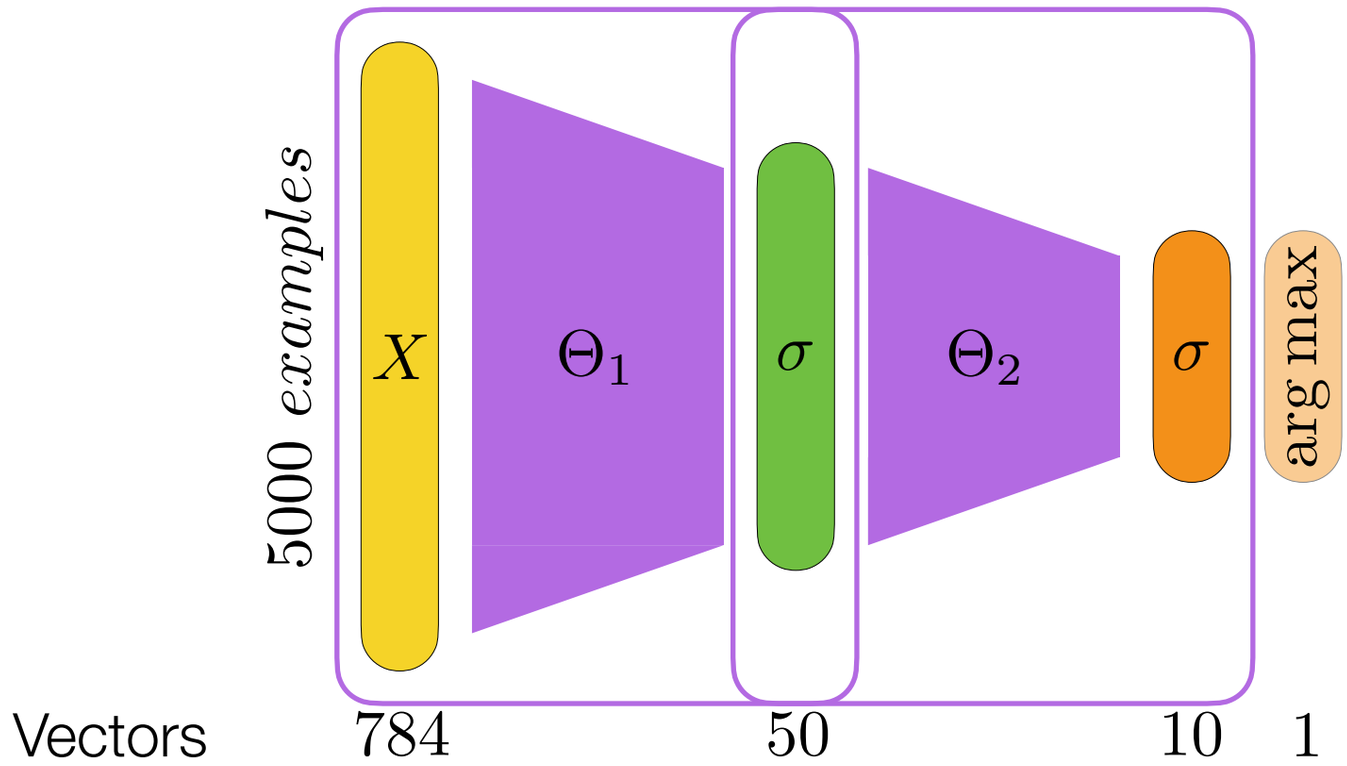
3 layers:
1 input layer
1 hidden layer
1 output layer





yann.lecun.com/exdb/mnist/

A practical example - MNIST




```python
def forward(Theta, X, active):
    N = X.shape[0]

    # Add the bias column
    X_ = np.concatenate((np.ones((N, 1)), X), 1)

    # Multiply by the weights
    z = np.dot(X_, Theta.T)

    # Apply the activation function
    a = active(z)

    return a
```

```python
def predict(Theta1, Theta2, X):
    h1 = forward(Theta1, X, sigmoid)
    h2 = forward(Theta2, h1, sigmoid)

    return np.argmax(h2, 1)
```

Code - Simple Network

https://github.com/DataForScience/DeepLearning

Practical Considerations

- So far we have looked at very idealized cases. Reality is never this simple!

- In practice, many details have to be considered:

    - Data normalization

    - Overfitting

    - Hyperparameters

    - Bias, Variance tradeoffs

    - etc…

Data Normalization

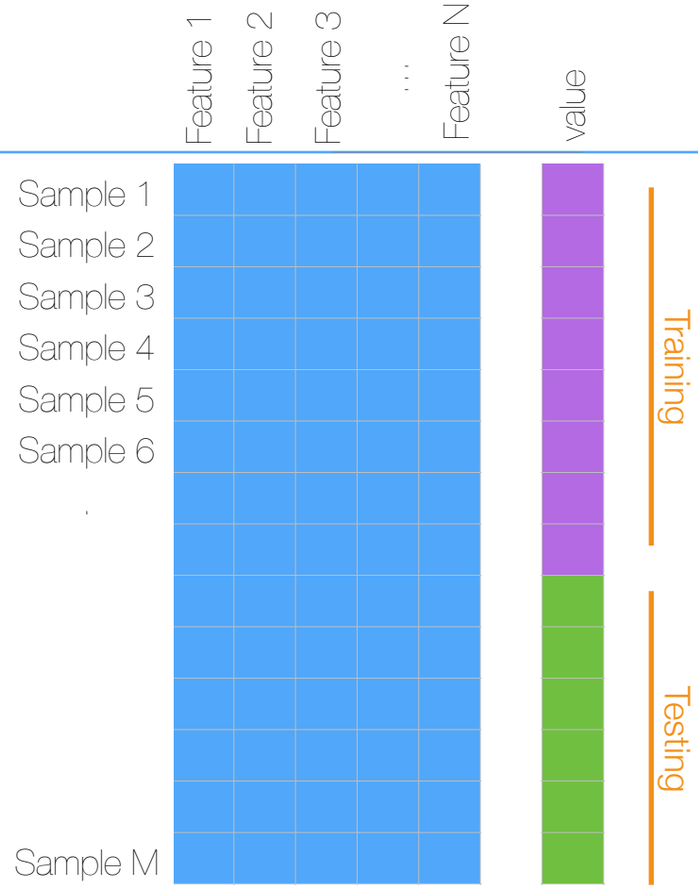
- The range of raw data values can vary widely.
- Using feature with very different ranges in the same analysis can cause numerical problems. Many algorithms are linear or use euclidean distances that are heavily influenced by the numerical values used (cm vs km, for example)
- To avoid difficulties, it's common to rescale the range of all features in such a way that each feature follows within the same range.
- Several possibilities:
  - Rescaling - $\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}}$
  - Standardization - $\hat{x} = \frac{x - \mu_x}{\sigma_x}$
  - Normalization - $\hat{x} = \frac{x}{||x||}$
  - In the rest of the discussion we will assume that the data has been normalized in some

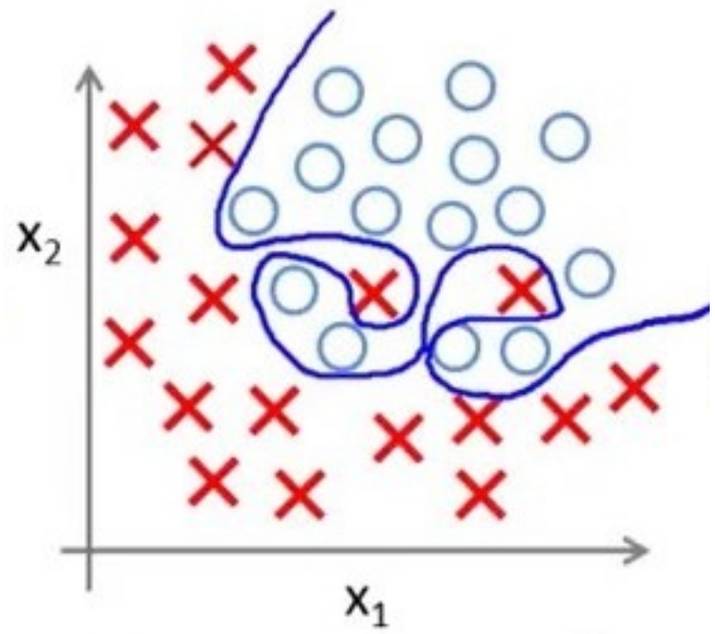
Data Normalization

- The range of raw data values can vary widely.
- Using feature with very different ranges in the same analysis can cause numerical problems. Many algorithms are linear or use euclidean distances that are heavily influenced by the numerical values used (cm vs km, for example)
- To avoid difficulties, it's common to rescale the range of all features in such a way that each feature follows within the same range.
- Several possibilities:
  - Rescaling - $\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}}$
  - Standardization - $\hat{x} = \frac{x - \mu_x}{\sigma_x}$
  - Normalization - $\hat{x} = \frac{x}{||x||}$
  - In the rest of the discussion we will assume that the data has been normalized in some

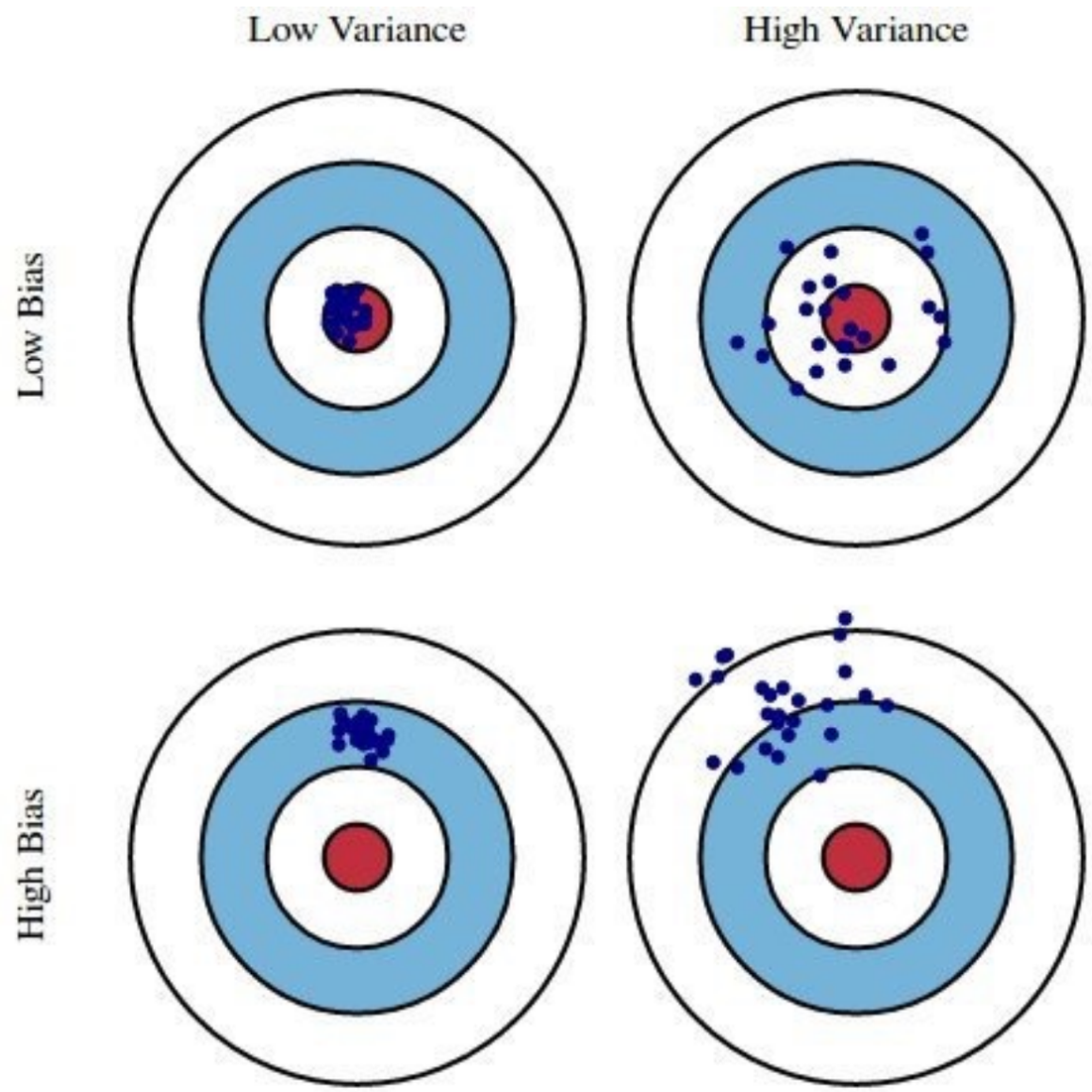
Supervised Learning - Overfitting

- "Learning the noise"
- "Memorization" instead of "generalization"
- How can we prevent it?
  - Split dataset into two subsets: Training and Testing
  - Train model using only the Training dataset and evaluate results in the previously unseen Testing dataset.
- Different heuristics on how to split:
  - Single split
  - k-fold cross validation: split dataset in k parts, train in k-1 and evaluate in 1, repeat k times and average results.

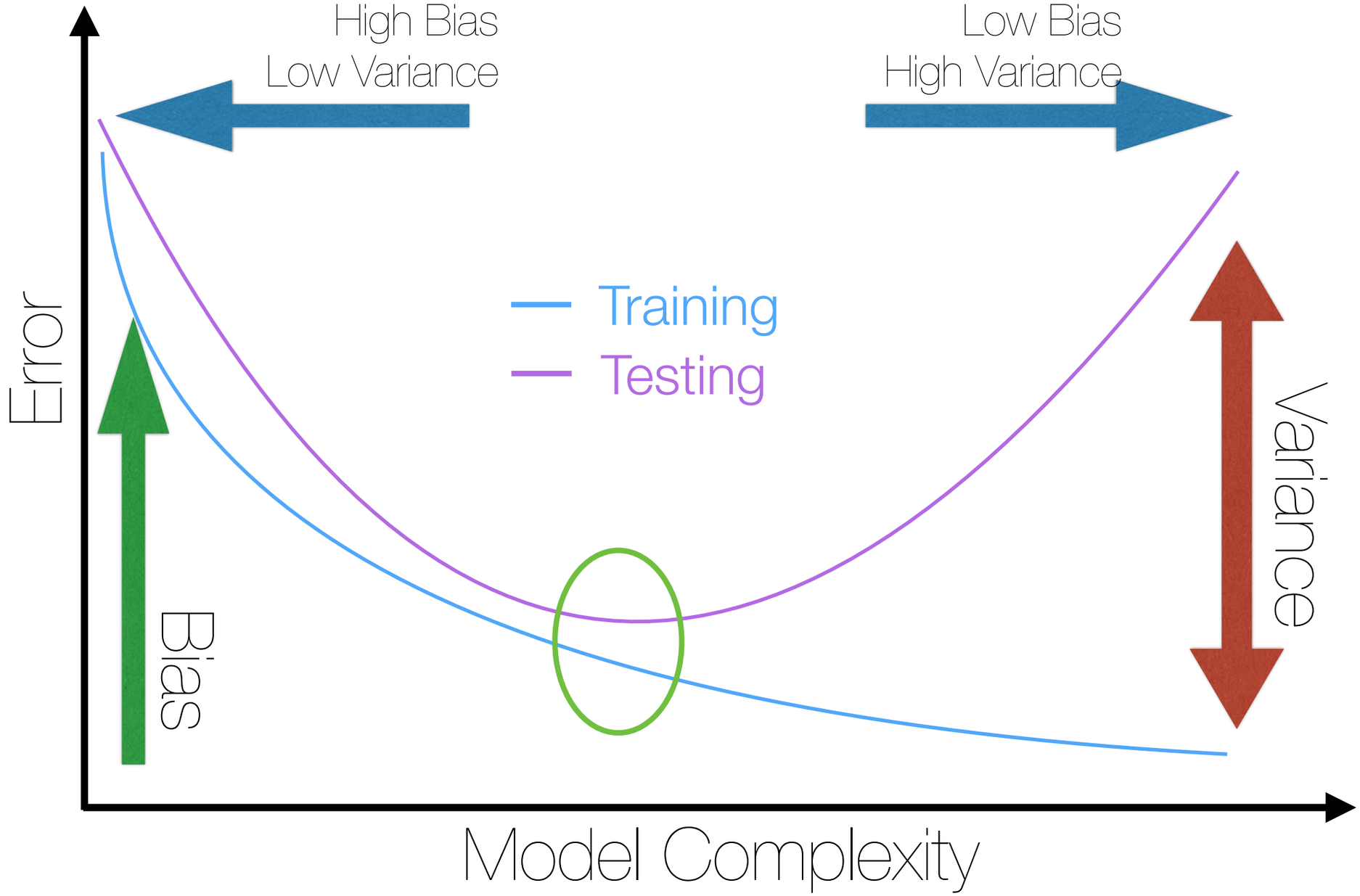




Bias-Variance Tradeoff



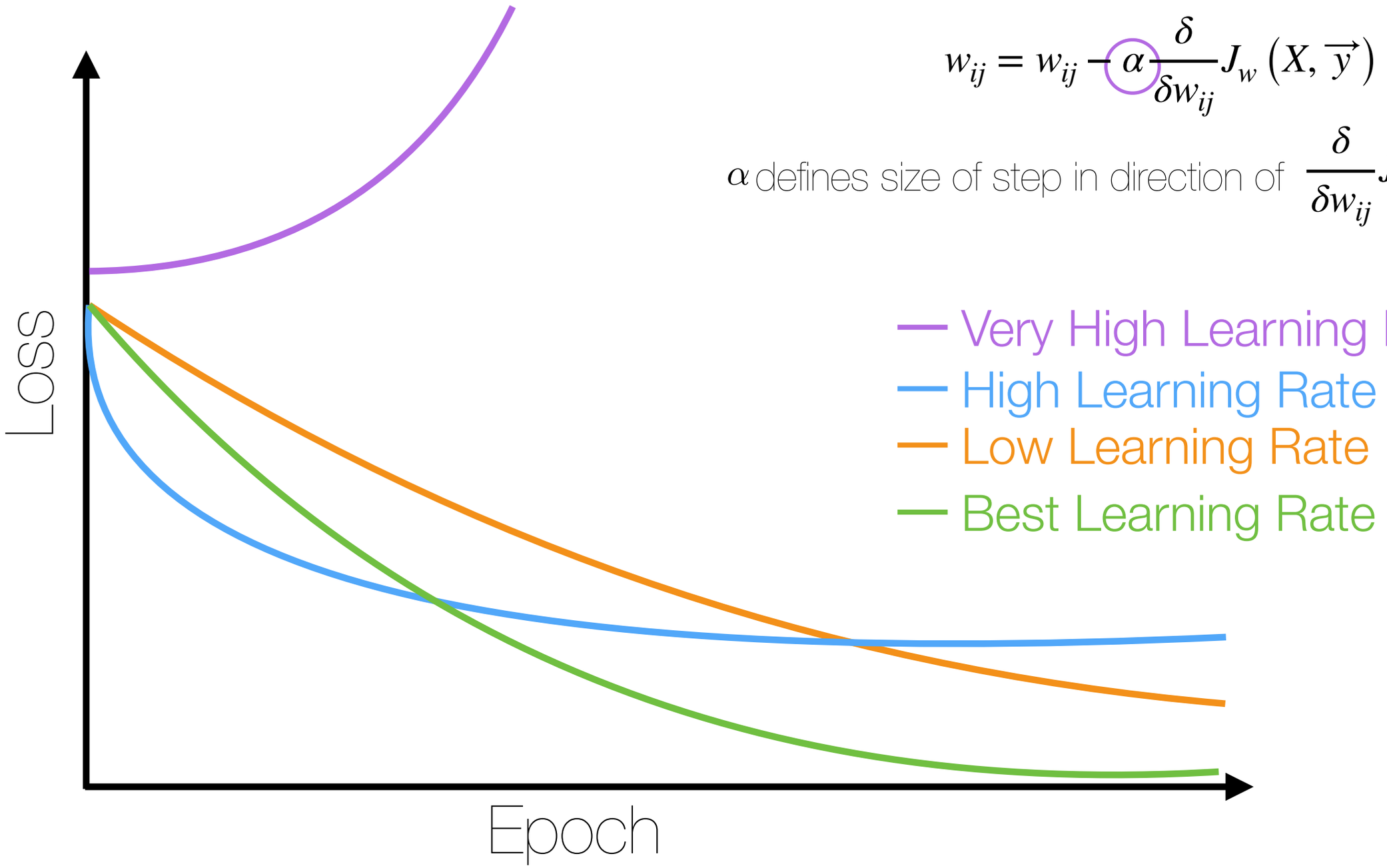
Bias-Variance Tradeoff



Learning Rate

$$w_{ij} = w_{ij} - \alpha \frac{\delta}{\delta w_{ij}} J_w\left(X, \vec{y}\right)$$

α defines size of step in direction of $\frac{\delta}{\delta w_{ij}} J_w\left(X, \vec{y}\right)$



Tips

- online learning - update weights after each case
  - might be useful to update model as new data is obtained
  - subject to fluctuations

- mini-batch - update weights after a "small" number of cases
  - batches should be balanced
  - if dataset is redundant, the gradient estimated using only a fraction of the data is a good approximation to the full gradient.

- momentum - let gradient change the velocity of weight change instead of the value directly

- rmsprop - divide learning rate for each weight by a running average of "recent" gradients

- learning rate - vary over the course of the training procedure and use different learning rates for each weight

Generalization

- Neural Networks are extremely modular in their design with
- Fortunately, we can write code that is also modular and can easily handle arbitrary numbers of layers
- Let's describe the structure of our network as a list of weight matrices and activation functions
- We also need to keep track of the gradients of the activation functions so let us define a simple class:

```
class Activation(object):
    def f(z):
        pass

    def df(z):
        pass

class Linear(Activation):
    def f(z):
        return z

    def df(z):
        return np.ones(z.shape)

class Sigmoid(Activation):
    def f(z):
        return 1./(1+np.exp(-z))

    def df(z):
        h = Sigmoid.f(z)
        return h*(1-h)
```

Generalization

- Now we can describe our simple MNIST model with:

```
Thetas = []
Thetas.append(init_weights(input_layer_size, hidden_layer_size))
Thetas.append(init_weights(hidden_layer_size, num_labels))

model = []

model.append(Thetas[0])
model.append(Sigmoid)
model.append(Thetas[1])
model.append(Sigmoid)
```

- Where Sigmoid is an object that contains both the sigmoid function and its gradient as was defined in the previous slide.

Generalization - Forward propagation

```python
def forward(Theta, X, active):
    N = X.shape[0]

    # Add the bias column
    X_ = np.concatenate((np.ones((N, 1)), X), 1)

    # Multiply by the weights
    z = np.dot(X_, Theta.T)

    # Apply the activation function
    a = active.f(z)

    return a


def predict(model, X):
    h = X.copy()

    for i in range(0, len(model), 2):
        theta = model[i]
        activation = model[i+1]

        h = forward(theta, h, activation)

    return np.argmax(h, 1)
```

```python
def backprop(model, X, y):
    M = X.shape[0]

    Thetas = model[0::2]
    activations = model[1::2]
    layers = len(Thetas)

    K = Thetas[-1].shape[0]
    J = 0
    Deltas = []

    for i in range(layers):
        Deltas.append(np.zeros(Thetas[i].shape))

    deltas = [0, 0, 0, 0]

    for i in range(M):
        As = []
        Zs = [0]
        Hs = [X[i]]
        # Forward propagation, saving intermediate results
        As.append(np.concatenate(([1], Hs[0])))  # Input layer

        for l in range(1, layers+1):
            Zs.append(np.dot(Thetas[l-1], As[l-1]))
            Hs.append(activations[l-1].f(Zs[l]))
            As.append(np.concatenate(([1], Hs[l])))

        y0 = one_hot(K, y[i])

        # Cross entropy
        J -= np.dot(y0.T, np.log(Hs[2]))+np.dot((1-y0).T, np.log(1-Hs[2]))

        # Calculate the weight deltas
        deltas[layers] = Hs[layers]-y0

        for l in range(layers-1, 1, -1):
            deltas[l] = np.dot(Thetas[l-1].T, deltas[l+1])[1:]*activations[l-1].df(Zs[l-1])
            Deltas[l] += np.outer(deltas[l+1], As[l])

    J /= M
    grads = []
    grads.append(Deltas[0]/M)
    grads.append(Deltas[1]/M)

    return [J, grads]
```

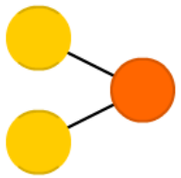
Code - Modular Network

https://github.com/DataForScience/DeepLearning

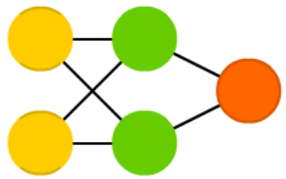
Neural Network Architectures

- Backfed Input Cell
- Input Cell
- Noisy Input Cell
- Hidden Cell
- Probablistic Hidden Cell
- Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Different Memory Cell
- Kernel
- Convolution or Pool

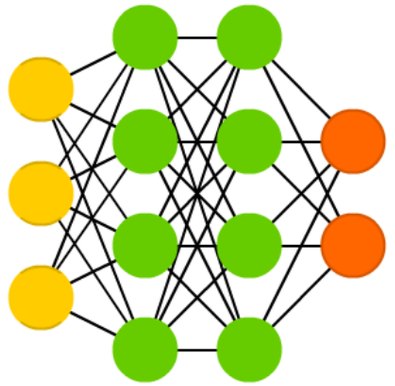
Perceptron (P)



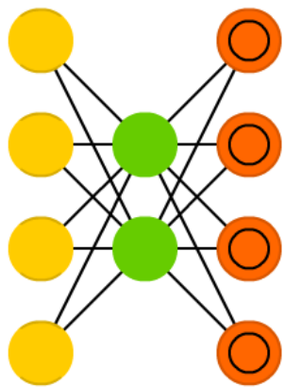
Feed Forward (FF)



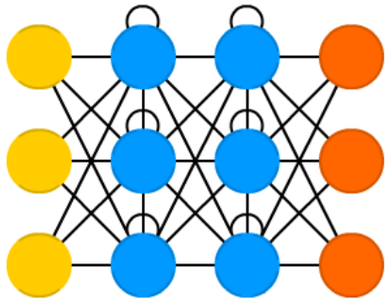
Deep Feed Forward (DFF)



Auto Encoder (AE)



Recurrent Neural Network (RNN)



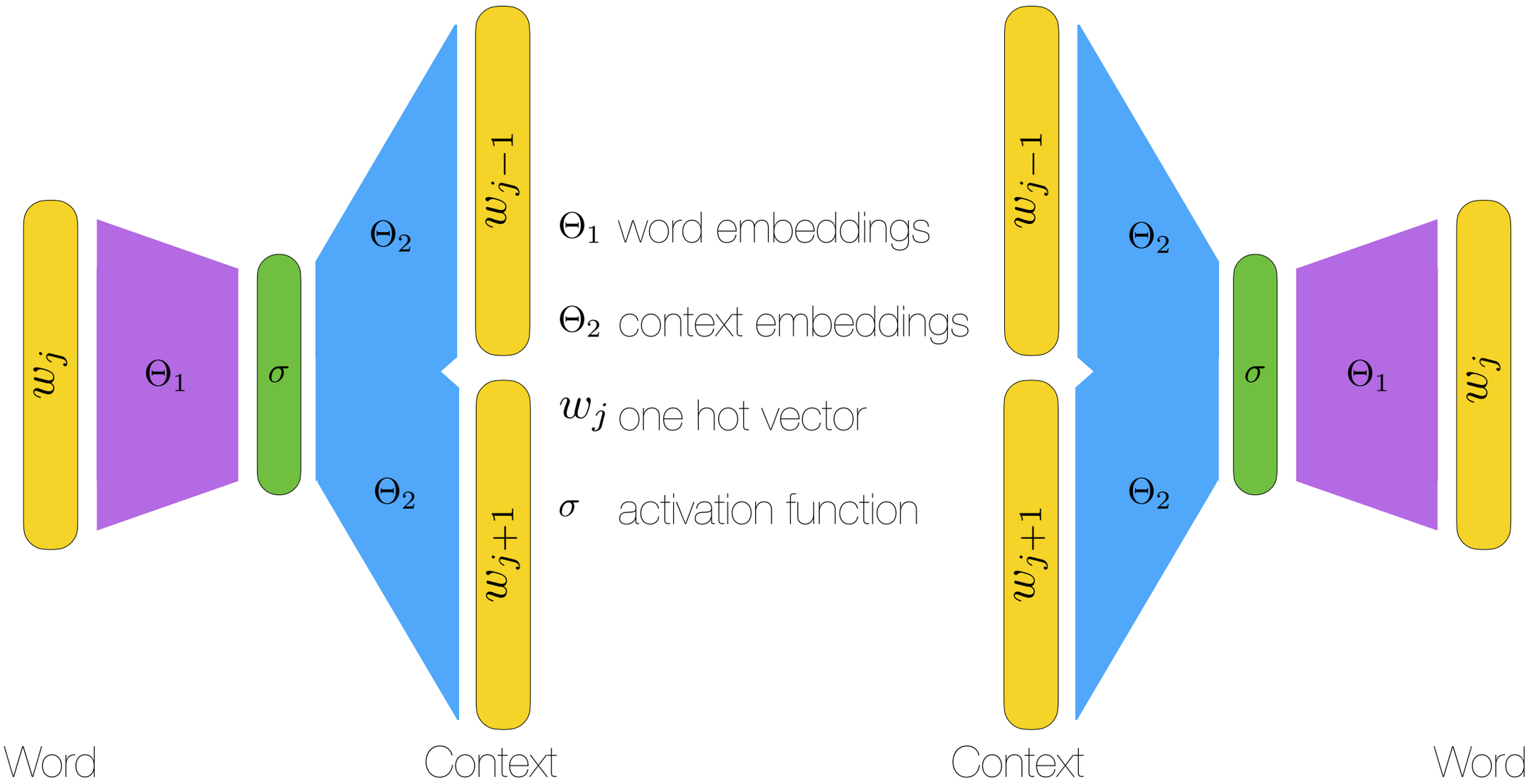
Mikolov 2013

Skipgram

$$\max p\left(C|w\right)$$

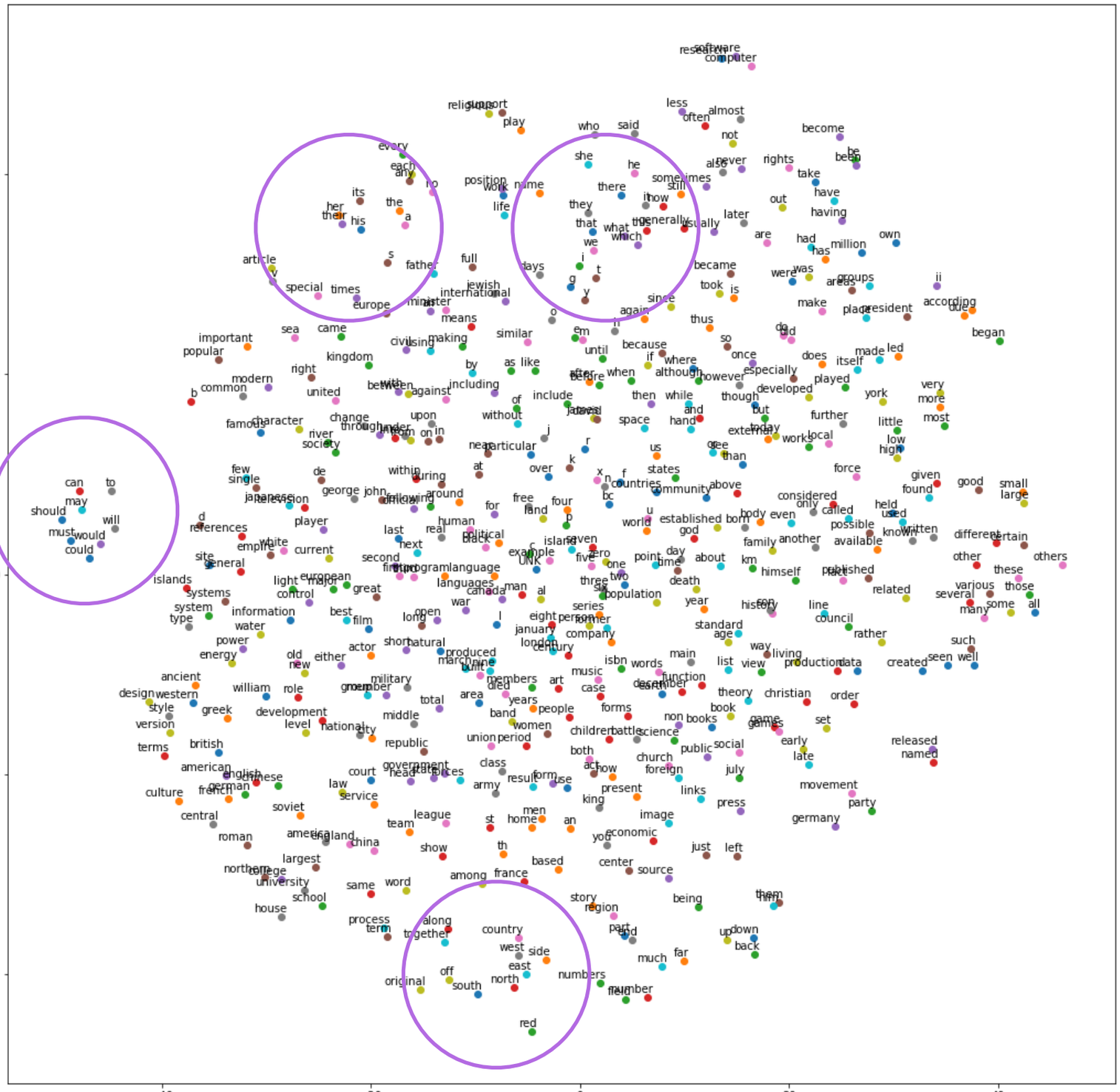
Continuous Bag of Words

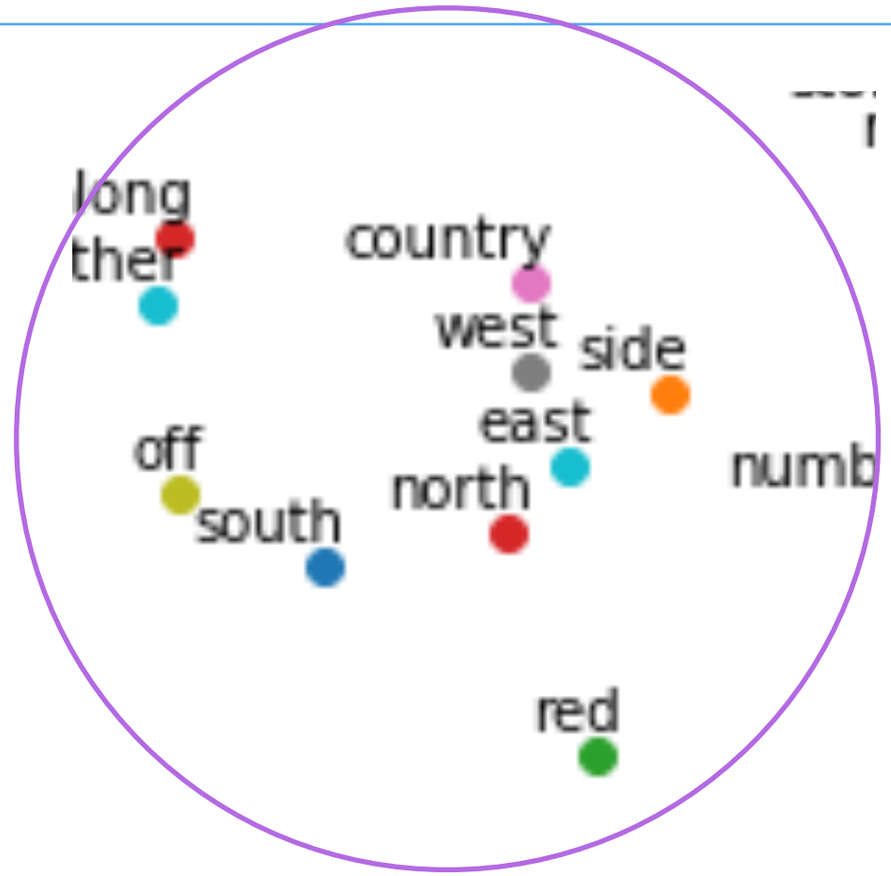
$$\max p\left(w|C\right)$$



Word Context Context Word

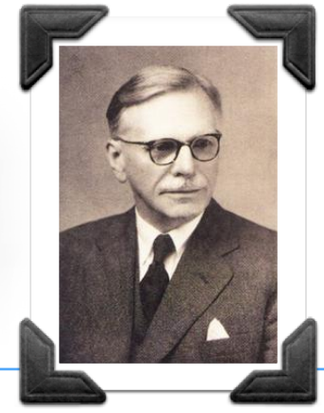
Visualization





Analogies

"You shall know a word by the company it keeps"
(J. R. Firth)



- The embedding of each word is a function of the context it appears in:

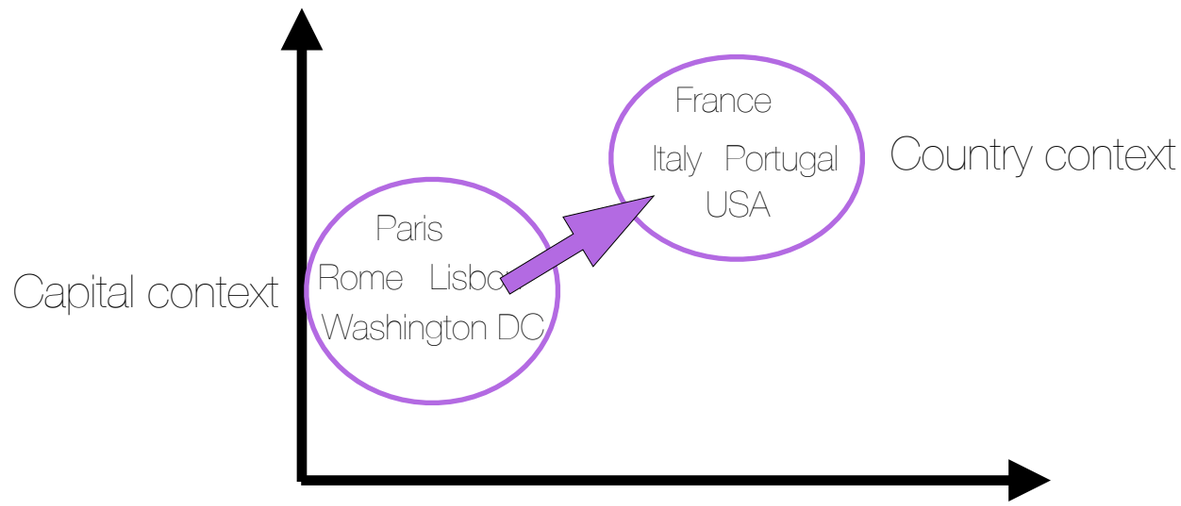
$$\sigma\left(red\right) = f\left(context\left(red\right)\right)$$

- words that appear in similar contexts will have similar embeddings:

$$context\left(red\right) \approx context\left(blue\right) \implies \sigma\left(red\right) \approx \sigma\left(blue\right)$$

- "Distributional hypotesis" in linguistics

Geometrical relations between contexts imply semantic relations between words!



$$\sigma\left(France\right) - \sigma\left(Paris\right) + \sigma\left(Rome\right) = \sigma\left(Italy\right)$$

$$\vec{b} - \vec{a} + \vec{c} = \vec{d}$$

Analogies

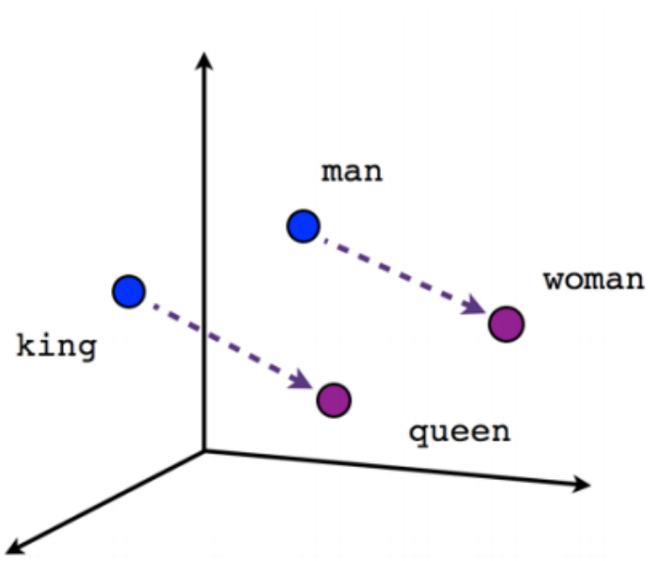
https://www.tensorflow.org/tutorials/word2vec

What is the word d that is most similar to b and c and most dissimilar to a?

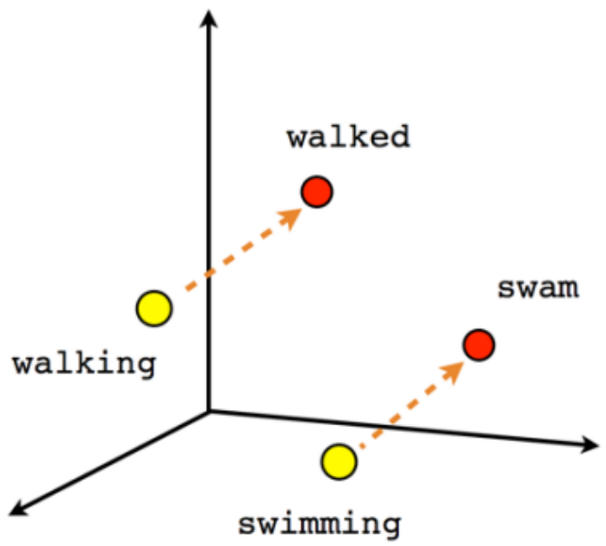
$$\vec{b} - \vec{a} + \vec{c} = \vec{d}$$

$$d^{\dagger} = \underset{x}{\text{argmax}} \frac{\left(\vec{b} - \vec{a} + \vec{c}\right)^T}{\left|\left|\vec{b} - \vec{a} + \vec{c}\right|\right|} \vec{x}$$

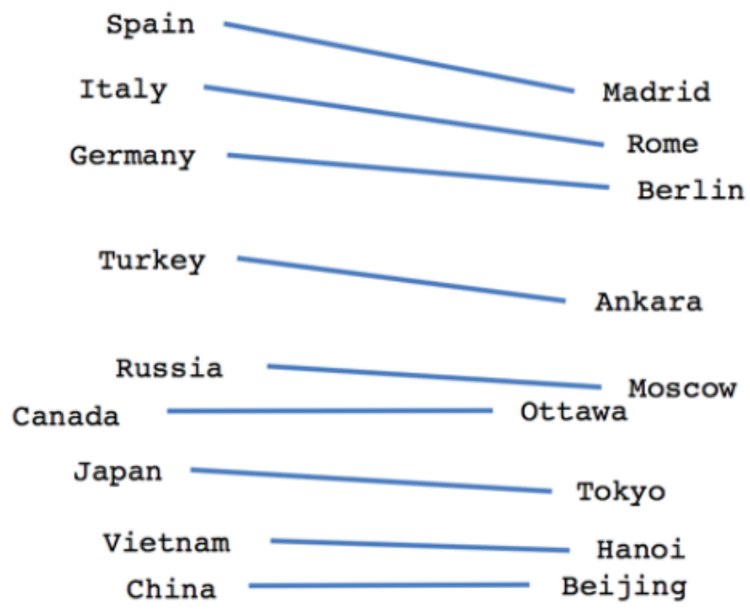
$$d^{\dagger} \sim \underset{x}{\text{argmax}} \left(\vec{b}^T \vec{x} - \vec{a}^T \vec{x} + \vec{c}^T \vec{x}\right)$$



Male-Female

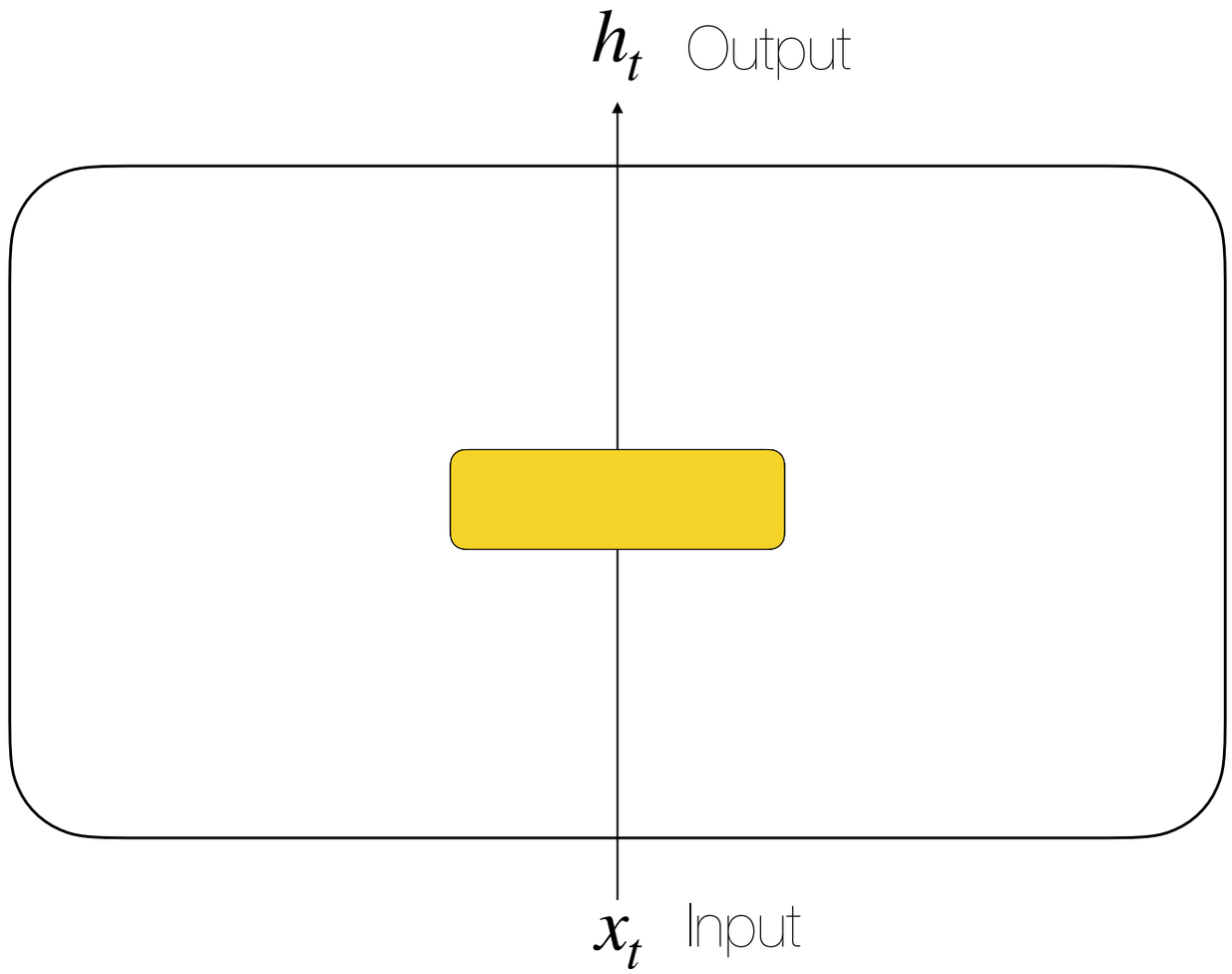


Verb tense



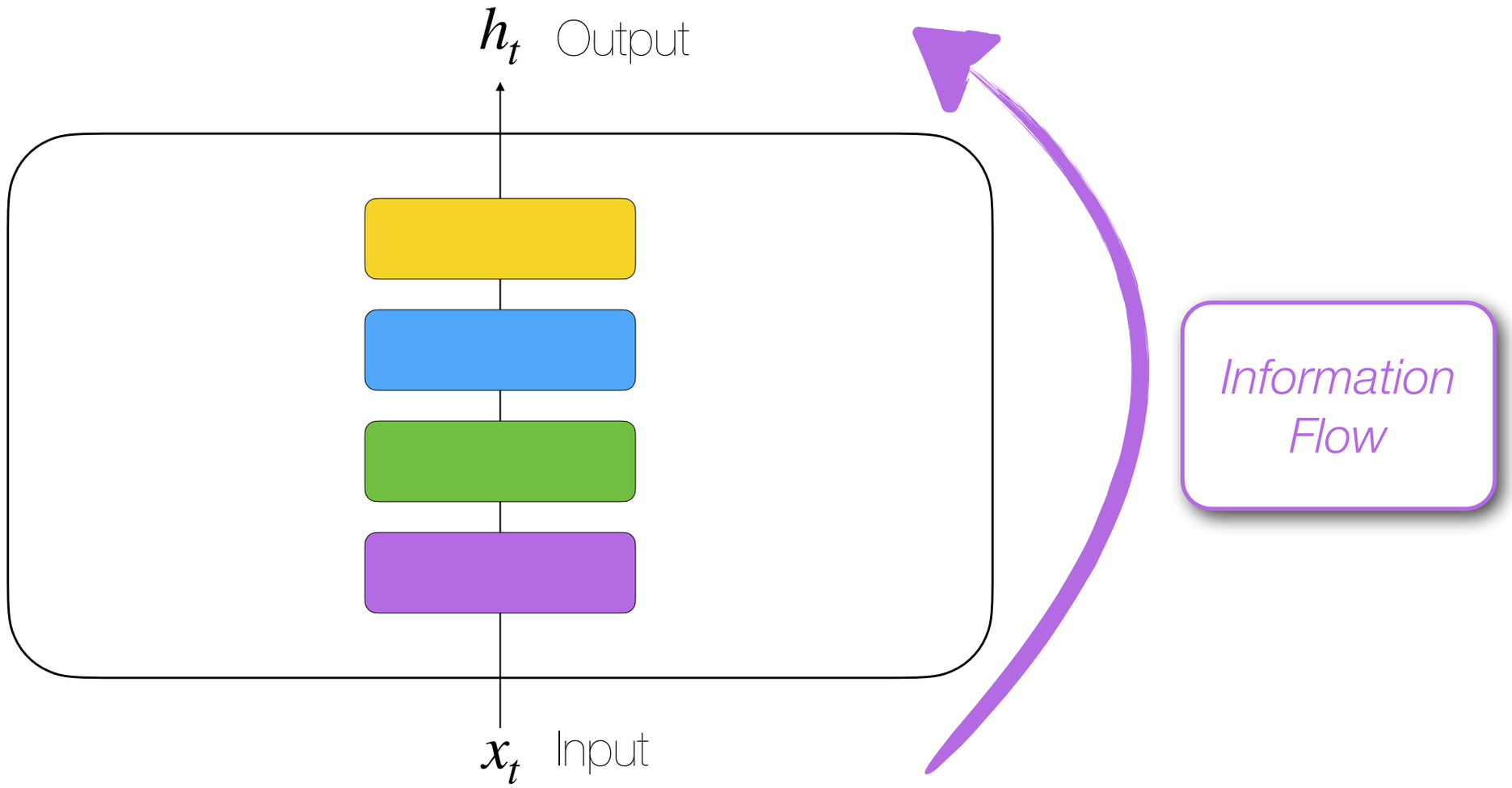
Country-Capital

Feed Forward Networks



$$h_t = f\left(x_t\right)$$

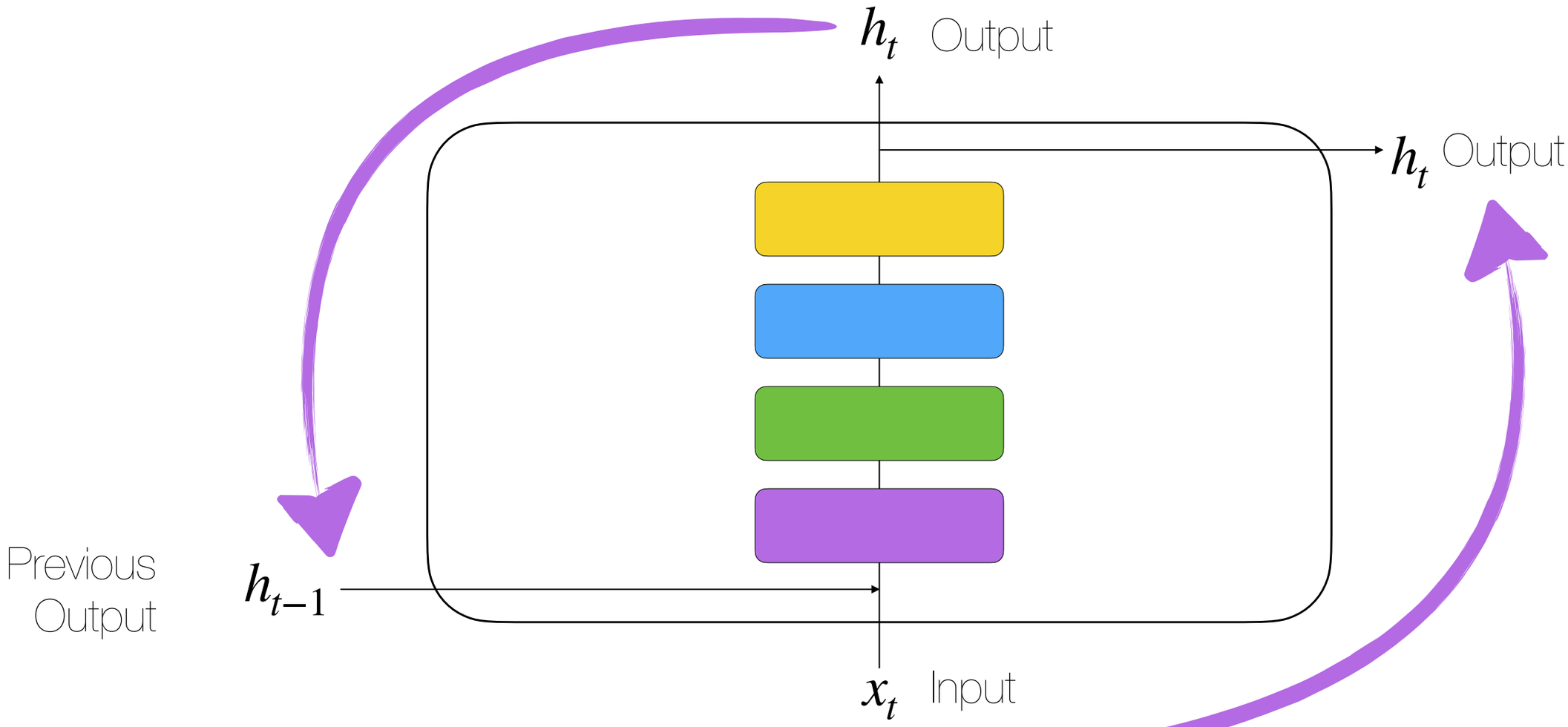
Feed Forward Networks



$$h_t = f\left(x_t\right)$$

Recurrent Neural Network (RNN)

Information Flow

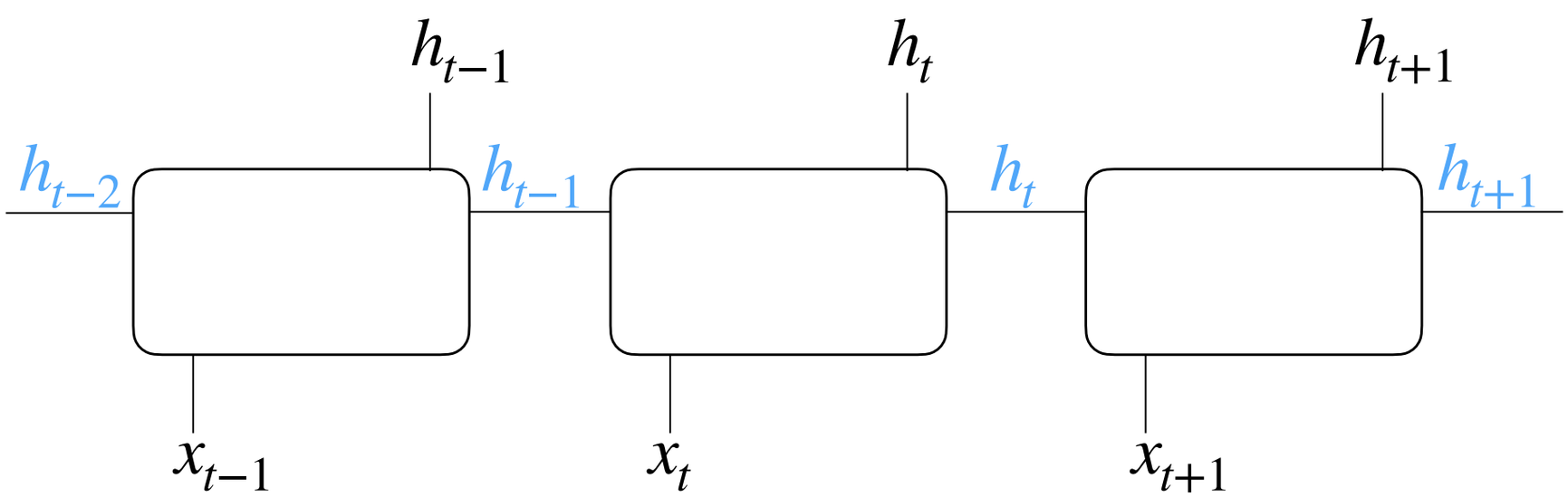


$$h_t = f\left(x_t, h_{t-1}\right)$$

Recurrent Neural Network (RNN)

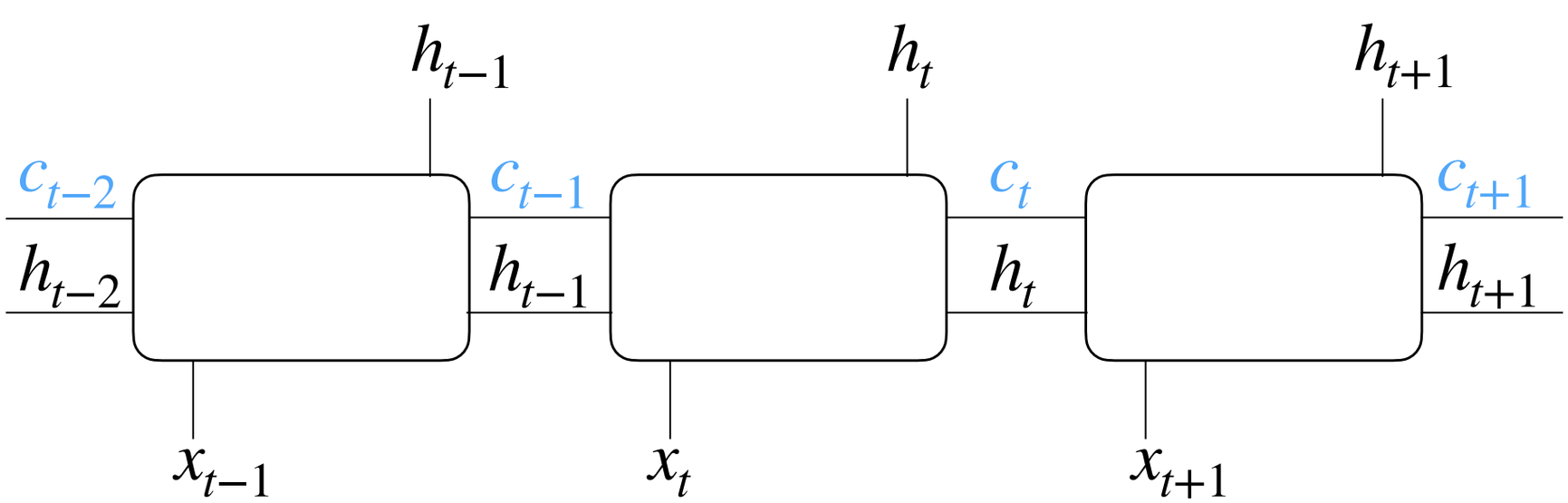
- Each output depends (implicitly) on all previous outputs.

- Input sequences generate output sequences (seq2seq)

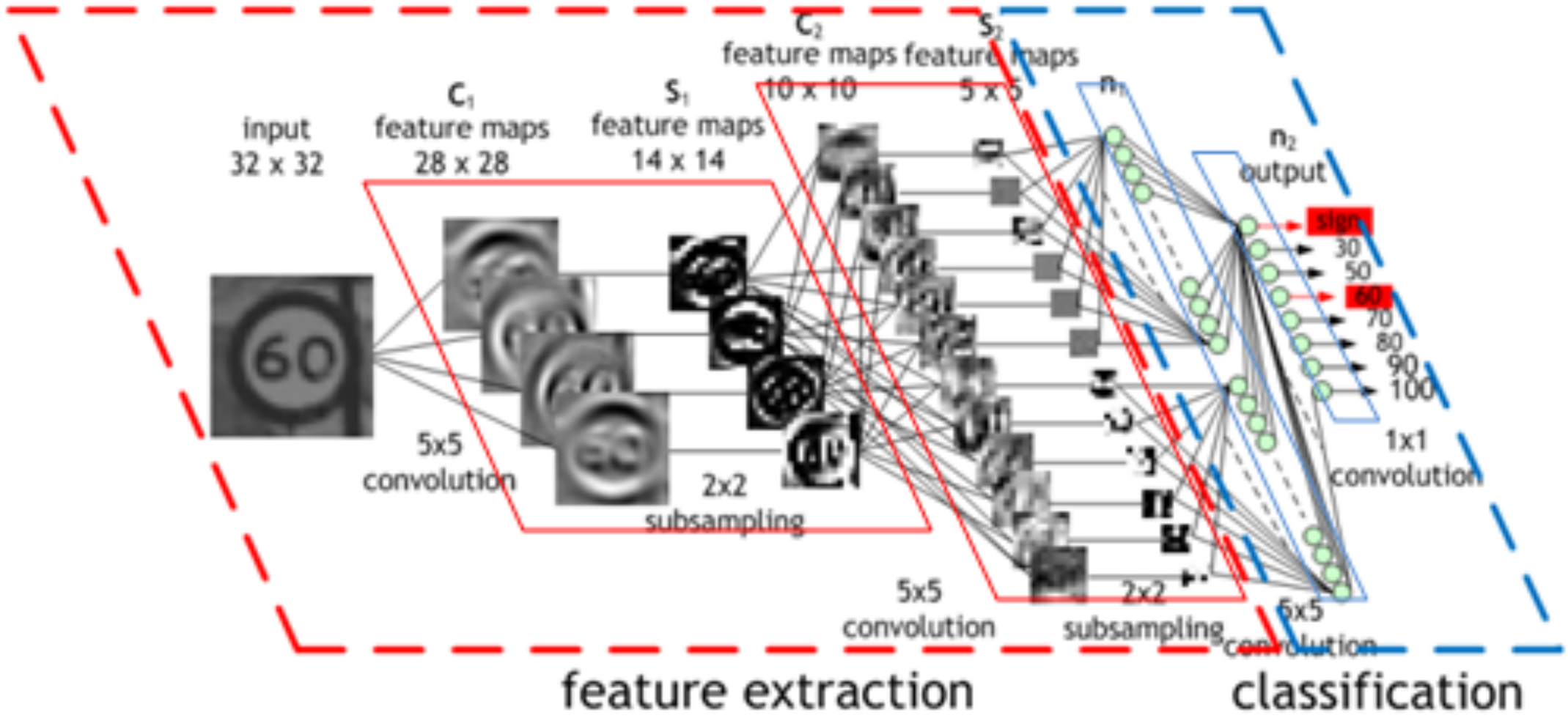


Long-Short Term Memory (LSTM)

- What if we want to keep explicit information about previous states (memory)?
- How much information is kept, can be controlled through gates.
- LSTMs were first introduced in 1997 by Hochreiter and Schmidhuber

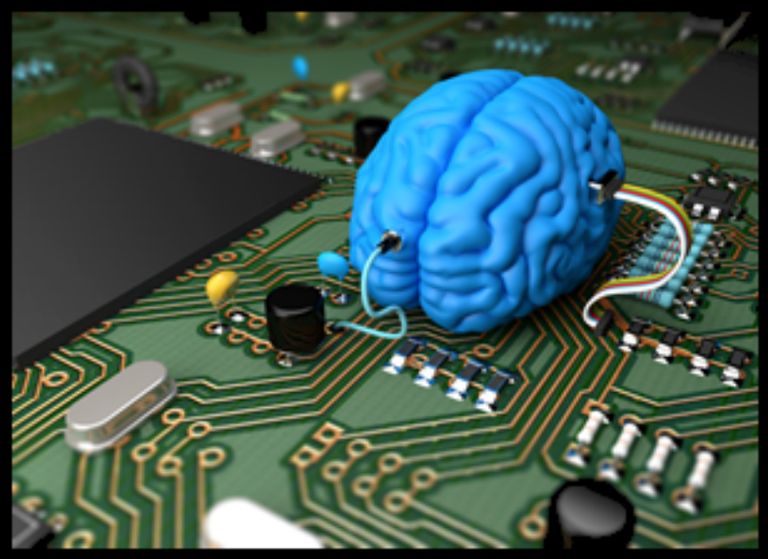


Convolutional Neural Networks



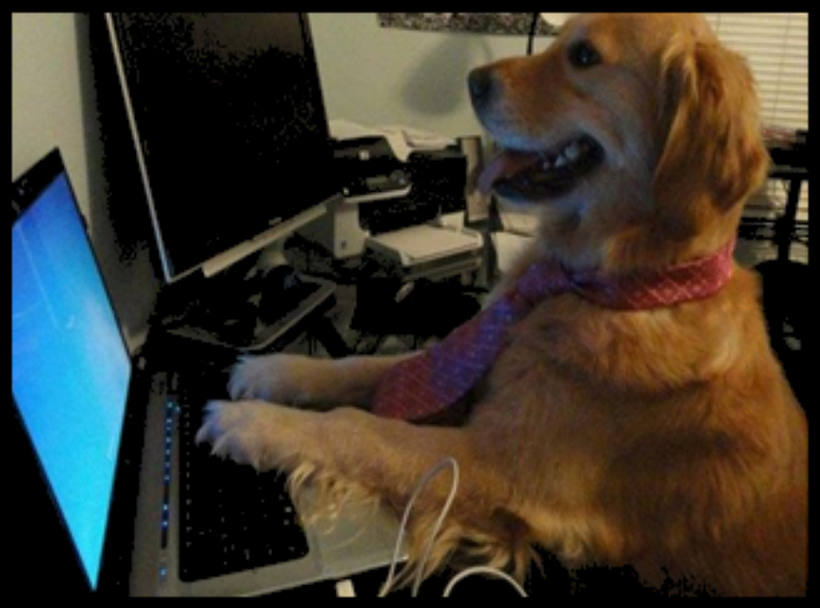
Deep Learning

What society thinks I do



What my friends think I do

What other computer scientists think I do



What mathematicians think I do

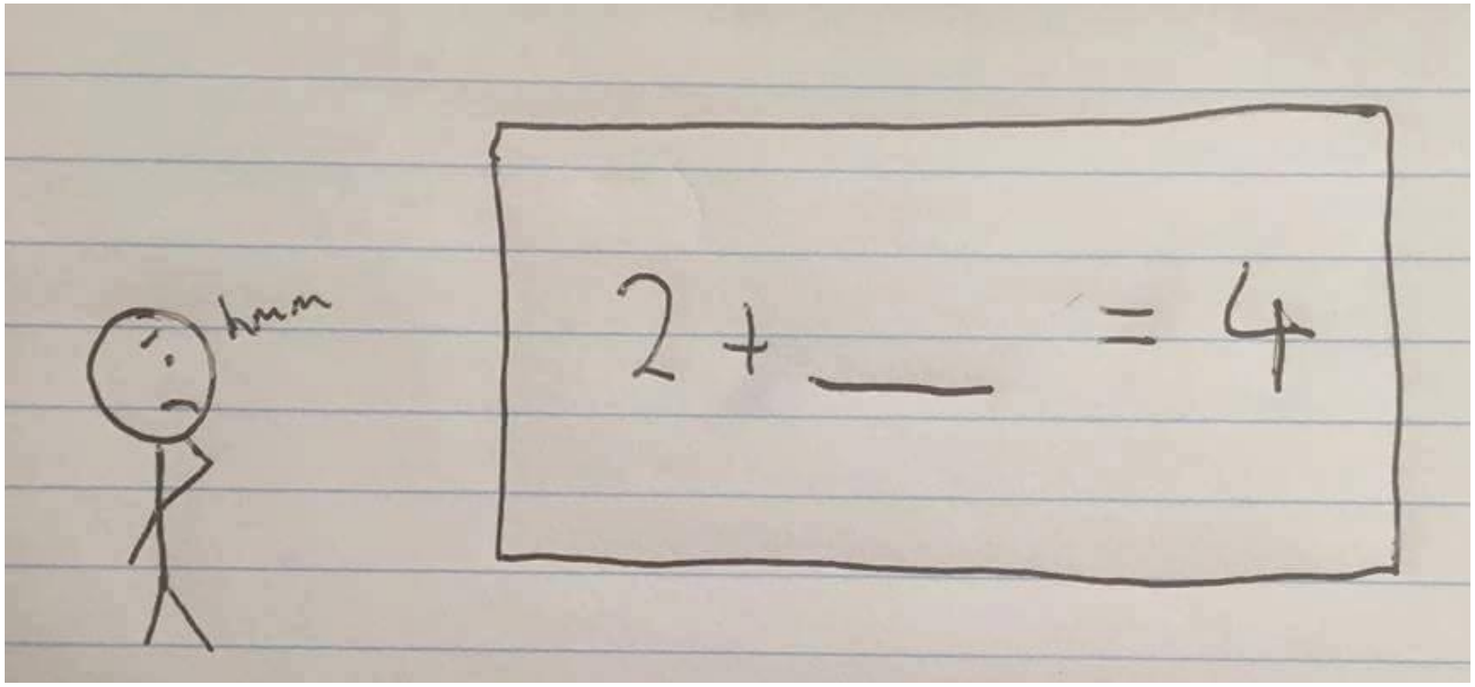
What I think I do

```
In [1]:
import keras
Using TensorFlow backend.
```

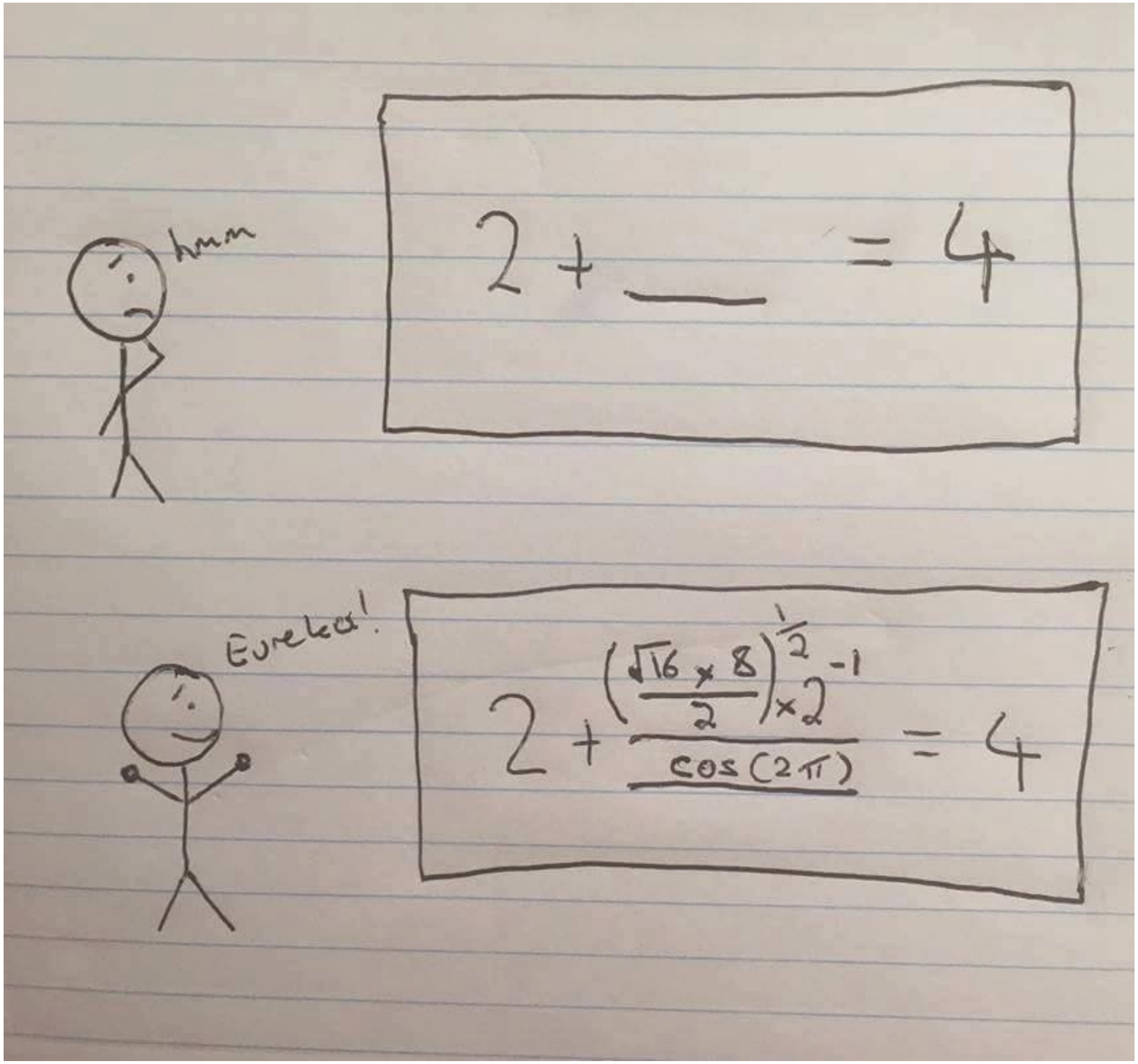
What I actually do

Curve Fitting?

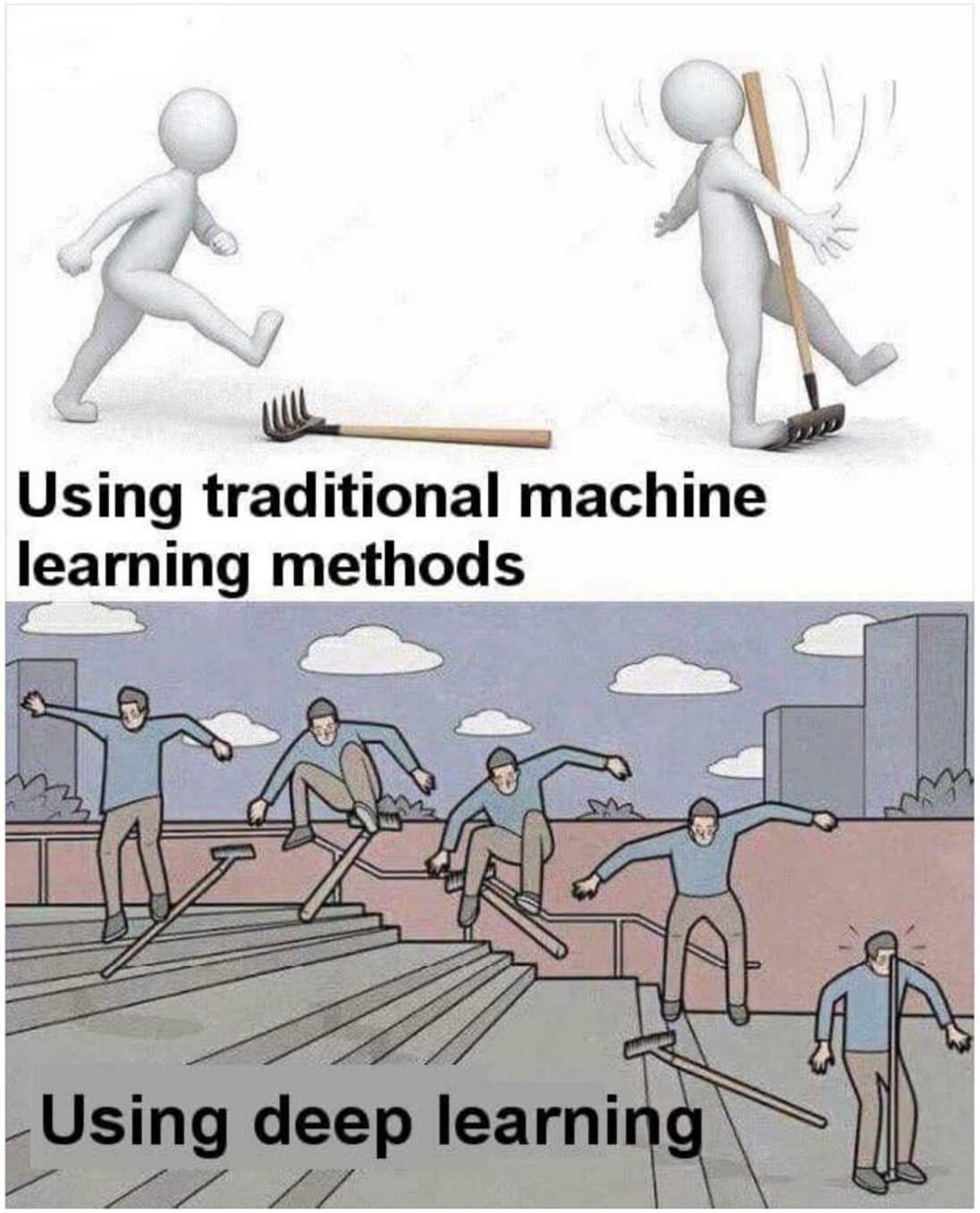
Interpretability?



Interpretability?



“Deep” learning



Events

graphs4sci.substack.com

Time Series for Everyone

Mar 4, 2022 - 10am-4pm (PST)

Advanced Time Series for Everyone

Mar 7, 2022 - 10am-4pm (PST)

Natural Language Processing (NLP) for Everyone

Apr 20, 2022 -10am-4pm (PST)

NLP with Deep Learning for Everyone

Apr 24, 2022 - 10am-4pm (PST)

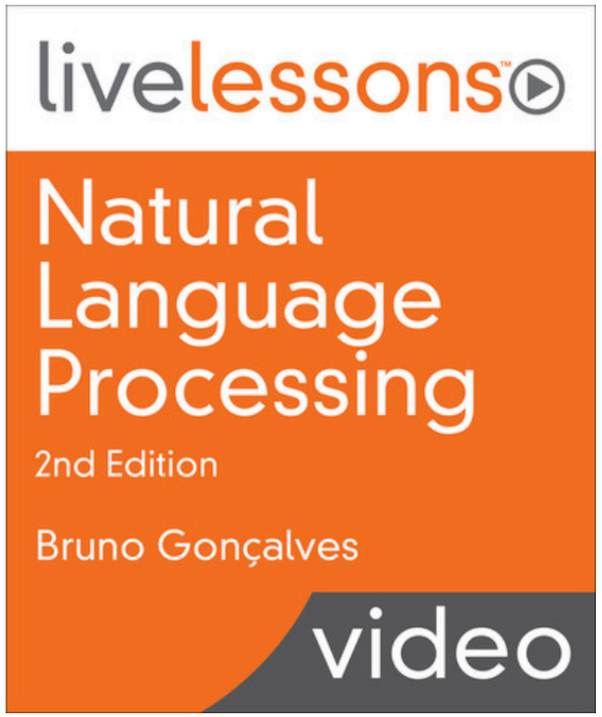
VIDEO

Natural Language Processing, 2nd Edition

Write the first review

By Bruno Goncalves

https://bit.ly/NLP_LL



TIME TO COMPLETE:
5h 23m

TOPICS:
Natural Language Processing

PUBLISHED BY:
Addison-Wesley Professional

PUBLICATION DATE:
October 2021

Start

5 Hours of Video Instruction

Overview

Natural Language Processing LiveLessons covers thefundamentals of Natural Language Processing in a simple and intuitive way,empowering you to add NLP to your toolkit. Using the powerful NLTK package, itgradually moves from the basics of text representation, cleaning, topicdetection, regular expressions, and sentiment analysis before moving on to theKeras deep learning framework to explore more advanced topics such as textclassification and sequence-to-sequence models. After successfully completingthese lessons you'll be equipped with a fundamental and practical understandingof state-of-the-art Natural Language Processing tools and algorithms.

VIDEO

Times Series Analysis for Everyone

★★★★★ 1 review

By Bruno Goncalves

https://bit.ly/Timeseries_LL

TIME TO COMPLETE:
6h

TOPICS:
Time Series

PUBLISHED BY:
Pearson

PUBLICATION DATE:
November 2021

Start

6 Hours of Video Instruction

The perfect introduction to time-based analytics

Overview

Times Series Analysis for Everyone LiveLessons covers the fundamental tools and techniques for the analysis of time series data. These lessons introduce you to the basic concepts, ideas, and algorithms necessary to develop your own time series applications in a step-by-step, intuitive fashion. The lessons follow a gradual progression, from the more specific to the more abstract, taking you from the very basics to some of the most recent and sophisticated algorithms by leveraging the statsmodels, arch, and Keras state-of-the-art models.

END