Introduction to Machine Learning and Deep Learning

[v1]

Mason Brain (2019)

Dom Huh, Jay Deorukhkar

Logistics

We are going to cover the foundational algorithms of machine learning and deep learning and look under the hood to see how these models are working at a lower level.

Linear Regression, Logistic Regression, Single Layer Models, Multilayer Perceptron Models

Our hope is you will understand what the algorithms do, and how the algorithms work at a high level.

We will stop at the end of every slide for questions, so please wait until then.

Logistics (cont)

There will be some math.

Working with matrices.

Dot Product

$$\begin{bmatrix} a & b \end{bmatrix} \bullet \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \end{bmatrix}$$

Matrix Multiplication

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \bullet \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \bullet \begin{bmatrix} w & x \\ y & z \end{bmatrix} = \begin{bmatrix} aw + by & ax + bz \\ cw + dy & cx + dz \end{bmatrix}$$

Element-wise Product

$$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \odot \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_1 \cdot b_1 \\ a_2 \cdot b_2 \end{bmatrix}$$

Logistics (cont)

There will be some math.

Taking partial derivatives.

Given $Z = \frac{1}{2} * X^2 + 5 * Y$

$dZ/dX = X$

$dZ/dY = 5$

Chain rule:

$d(g(ax+b))/dx = g'(ax+b) * a$

Remember that partial derivative is equal to the rate of change (aka slope) of the function wrt the variable (X or Y)

Logistics (cont)

We are going to use Python

To download frameworks:

In terminal: pip3 install numpy

numpy: Framework for matrix calculation (vectorization)

Including frameworks:

```
import numpy as np
```

Note: using “as np” allows you to refer to that framework as np

What is Machine Learning?

Machine learning is giving computers the ability to learn.

Thus, these programs do not require explicit rule set/instructions, meaning minimal human intervention in developing these models.

In practice, machine learning is applied with data.

General Steps of Machine Learning

1) Obtain and process data

2) Split data into three/four sections
    a) (Training, Validation, Testing)

3) Define model architecture with learning algorithm

4) Train model with training dataset (aka Model is learning)

5) Check how good your model is with validation/testing dataset

6) Deploy your model

Dataset

What is a dataset?

Features:

Median Income, House Age, Average Rooms, Average Bedrooms, Population…

Labels:

Average house value in units of 100,000.

```
huh:conv dyh7$ more california_housing.py


from sklearn.datasets import *
import pandas as pd
ds = fetch_california_housing()
df = pd.DataFrame(data = ds.data, index = ds.target, columns = ds.feature_names)
print(df.head())

huh:conv dyh7$ python3 california_housing.py
       MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
4.526  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
3.585  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
3.521  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85
3.413  5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
3.422  3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85

       Longitude
4.526    -122.23
3.585    -122.22
3.521    -122.24
3.413    -122.25
3.422    -122.25
```

Labels

Dataset

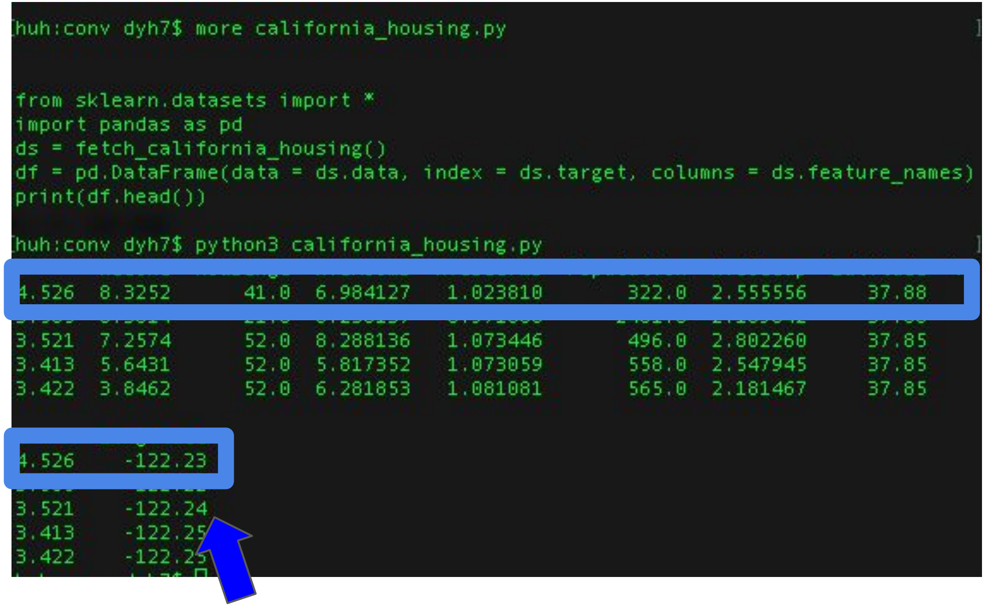
housing.csv (1.36 MB)

10 of 10 columns

	# longitu...	# latitude	# housin...	# total_ro...	# total_b...	# populat...	# househ...	# median...	# median...	A ocean_...
1	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
2	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
3	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
4	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
5	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY
6	-122.25	37.85	52.0	919.0	213.0	413.0	193.0	4.0368	269700.0	NEAR BAY
7	-122.25	37.84	52.0	2535.0	489.0	1094.0	514.0	3.6591	299200.0	NEAR BAY
8	-122.25	37.84	52.0	3104.0	687.0	1157.0	647.0	3.12	241400.0	NEAR BAY
9	-122.26	37.84	42.0	2555.0	665.0	1206.0	595.0	2.0804	226700.0	NEAR BAY
10	-122.25	37.84	52.0	3549.0	707.0	1551.0	714.0	3.6912	261100.0	NEAR BAY
11	-122.26	37.85	52.0	2202.0	434.0	910.0	402.0	3.2031	281500.0	NEAR BAY

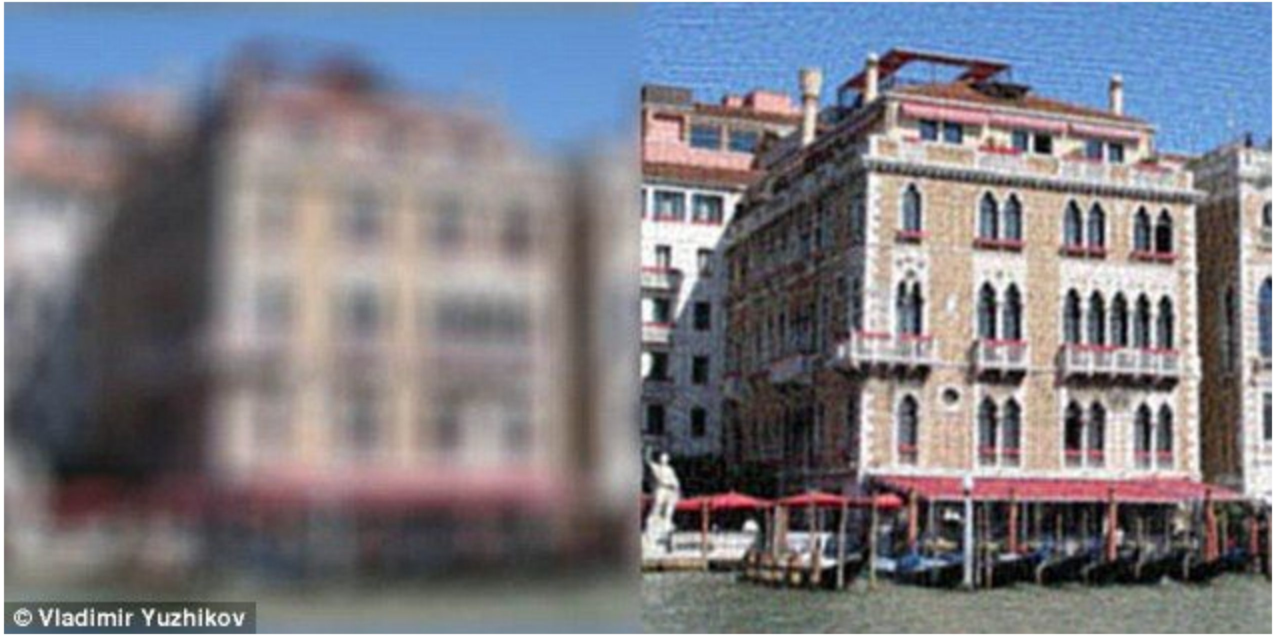
Dataset

A dataset can have many examples, and it is important to perform some sort of analysis on the dataset to ensure the distribution of the dataset of the split is optimal.

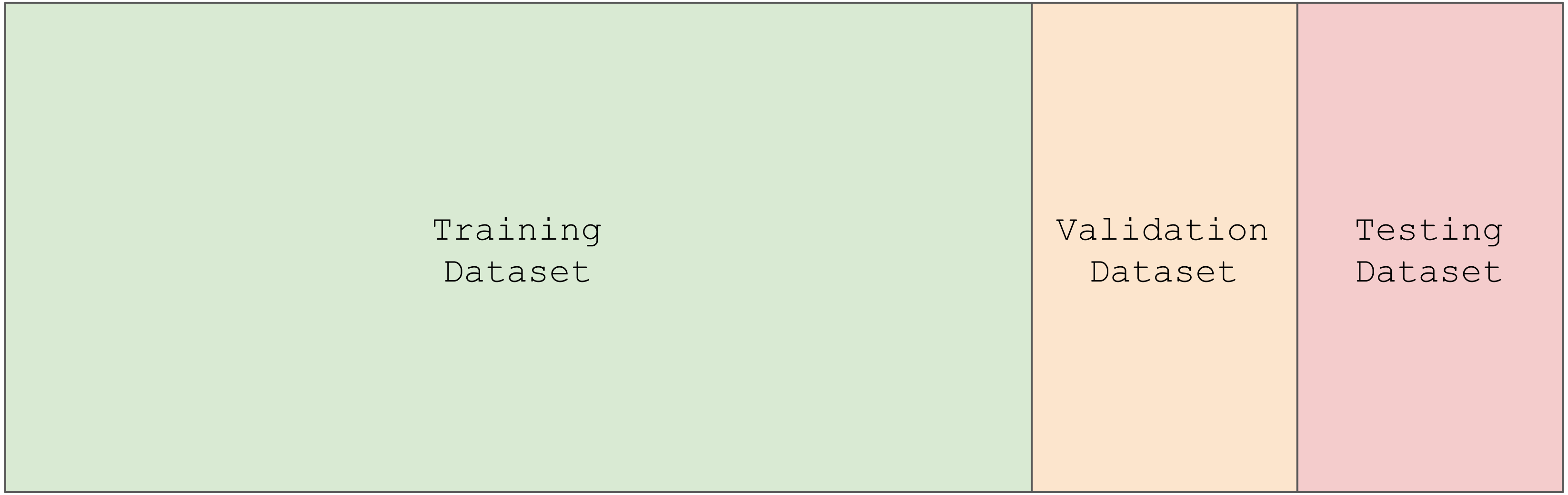


1 data example

Distribution of Dataset



Split of Dataset



Depend on the scale of the dataset.
If the dataset is very large, you can likely use 90% for just training.
If the dataset is smaller, you need a sufficient amount of validation and testing data to ensure model evaluation reliability.

Categories of Machine Learning

Supervised Learning

  Uses dataset with labels

Unsupervised Learning

  Uses dataset without labels

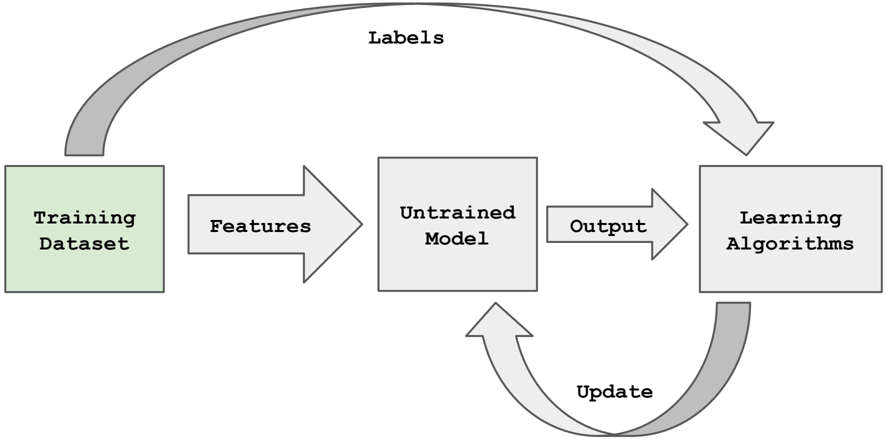
Reinforcement Learning

  Uses policy and value function to learn from environment

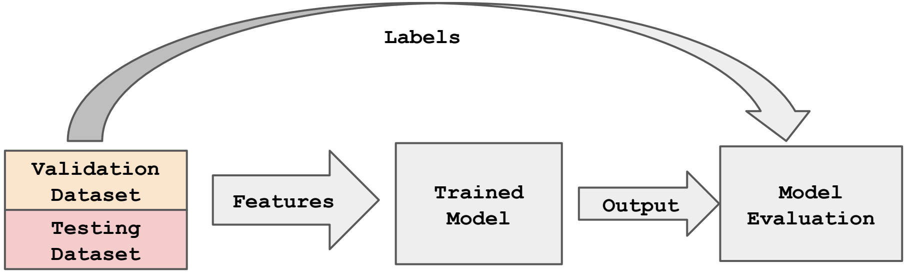
Supervised Learning

Dataset is fed into the model, and the model learns the mapping between the features and labels by using a learning algorithm like gradient descent optimizer. Validation and testing would provide insight on how good on model.

Training



Validation/Testing

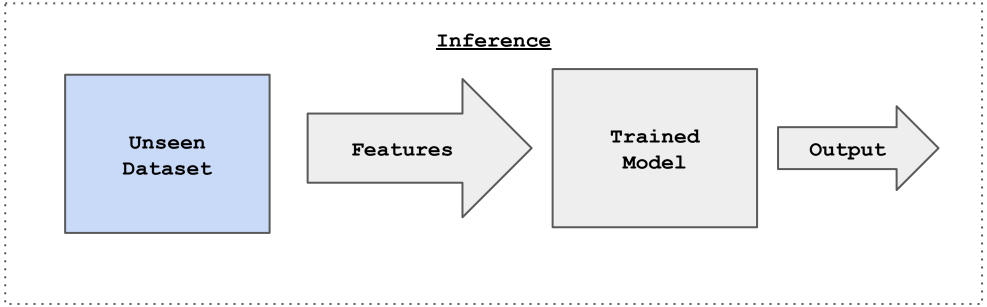


Supervised Learning

Once the model is trained, it can be used to predict on unseen data. Typically for supervised learning, there are two main use cases: regression and classification.

Regression is used if the desired output is real-valued.

Classification is used if the desired output is discrete.

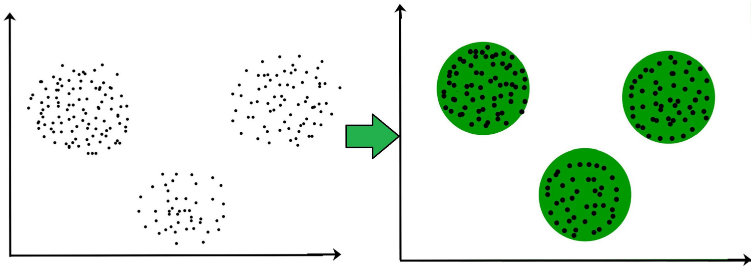


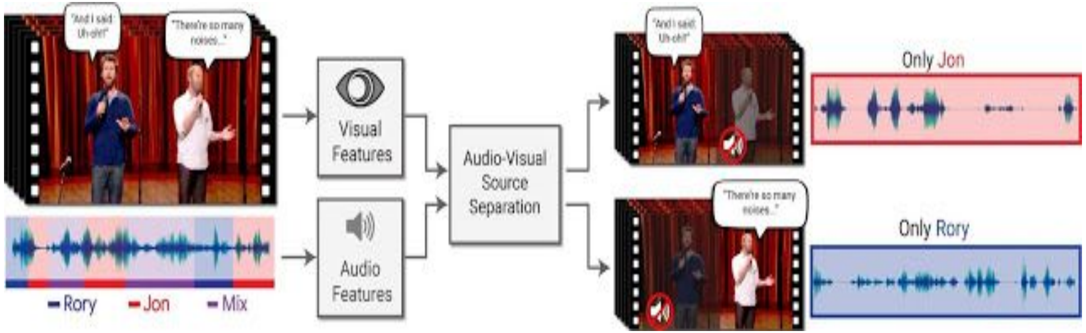
House Price		
<10K	<50K	>100K
0	0	1

House Price	124,102

Unsupervised Learning

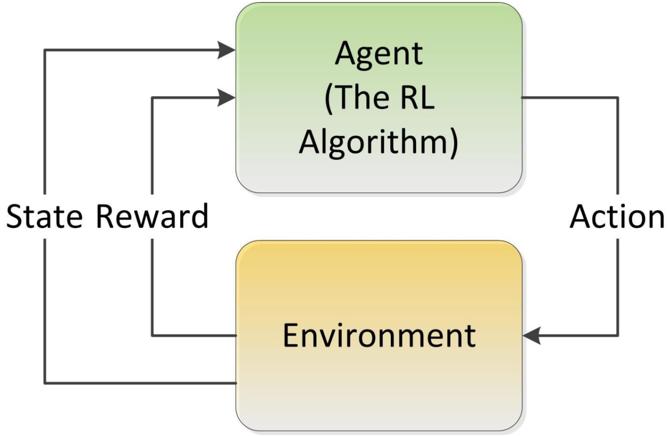
Model is trained only on the features, and derive patterns, such as similarities, correlation, within the dataset. Clustering and non-clustering are common use cases of unsupervised learning.

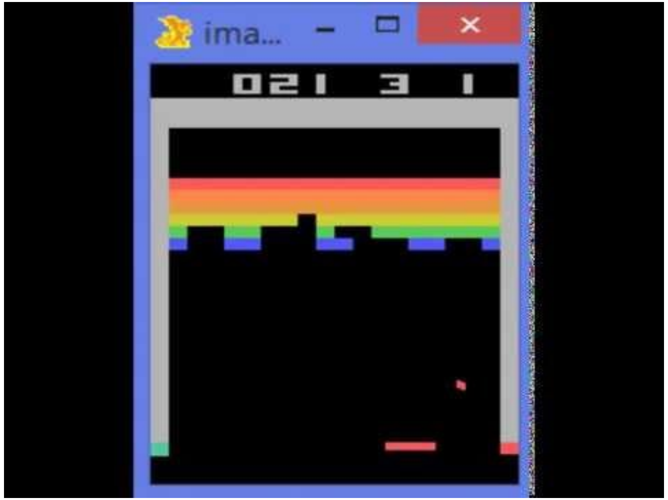




Reinforcement Learning

Agent learns from the environment with policy, which is the agent's behavior function(how the agent picks its action), value function, which determines how good each state and action is, and the model, which is the agent's representation of the environment.





Overview

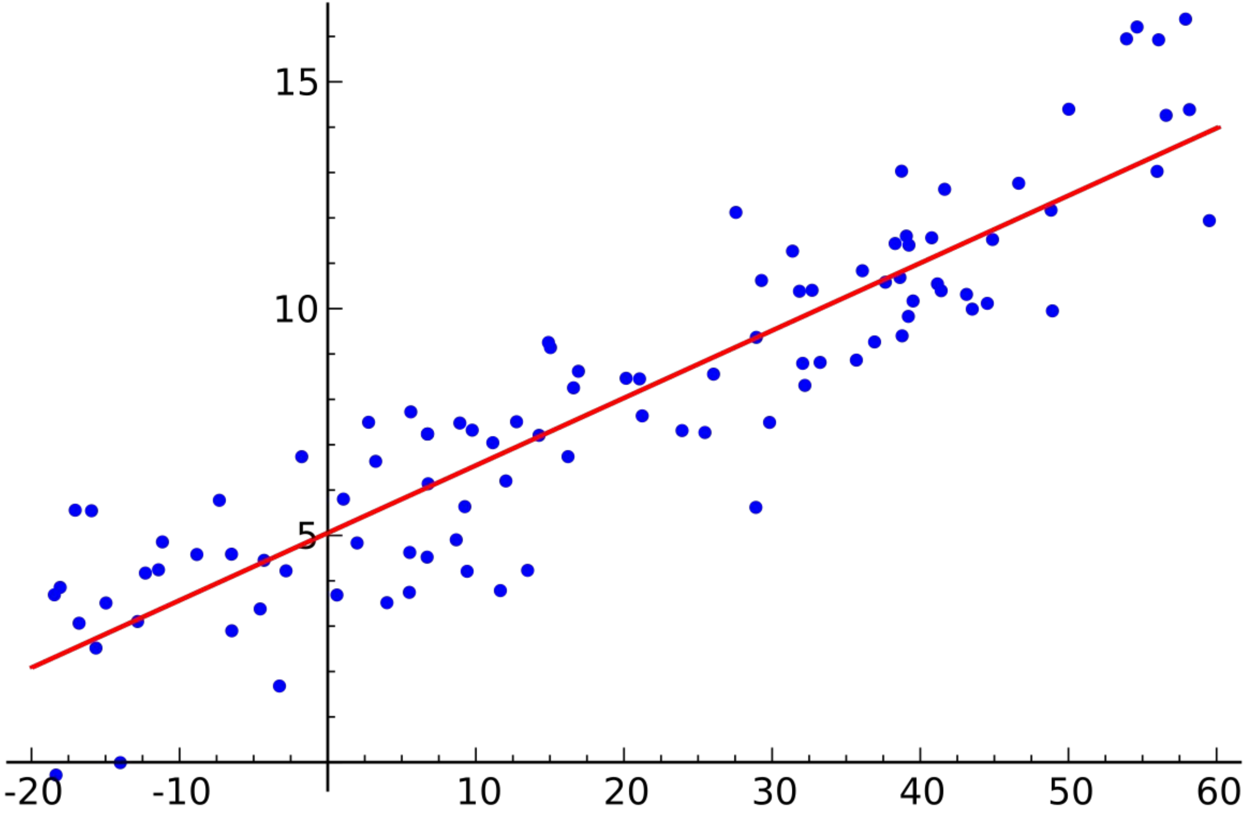
Linear Regression

Logistic Regression

Single Layer Model

Multi-Layer Perceptron Model

Linear Regression



Linear Regression

Goal: Find the best fit line given a dataset

Input: Features (X), Labels (Y)

Parameters: Weight(W), bias(b)

Output: Prediction = $\hat{y}$ = WX + b

Loss Function: L = $\frac{1}{2}(\hat{y} - Y)^2$

Cost Function: J = 1/m * sum(L) over all examples (m) (MSE)

Linear Regression

Goal: Find the best fit line given a dataset

Hyperparameter: learning rate (α)

Loss Function: $L = \frac{1}{2}(\hat{y} - Y)^2$

Cost Function: $J = 1/m * \text{sum}(L)$ over all examples (m)

Updating our Weights and Bias:

$W = W - \alpha * dJ/dW$

$b = b - \alpha * dJ/db$

This is gradient descent optimizer. Takes the partial derivative wrt the Parameter being updated.

Linear Regression (cont)

Input: Features (X), Labels (Y)

X = [1 3 6 5], Y = [4 8 14 13]

Parameters: Weight, bias (W, b) are randomly initialized

W = [5], b = [1 1 1 1] or [1] (w/ broadcasting)

Output: Prediction (Y_pred = WX + b)

Y_pred = [5]*[1 3 6 5] + [1] = [11 16 31 26]

Loss/Cost (m=4)

J = ¼*sum(½([11 16 31 26] - [4 8 14 13])^2) = 11.25 >> 0

Linear Regression (cont)

Loss/Cost (m=4)

$$J = \frac{1}{4}*\frac{1}{2}([11\ 16\ 31\ 26] - [4\ 8\ 14\ 13])^2 >> 0$$

Update weights (a = 0.01)

$$J = \frac{1}{4}*\frac{1}{2}(Y\_pred - Y)^2 = \frac{1}{4}*\frac{1}{2}(WX+b - Y)^2$$

$$dJ/dW = \frac{1}{4}(Y\_pred - Y) * X$$

$$dJ/db = \frac{1}{4}(Y\_pred - Y)$$

Linear Regression (cont)

Update weights (a = 0.01)

J = ¼*½(Y_pred - Y)^2 = ¼*½(WX+b - Y)^2

dJ/dW = ¼(Y_pred - Y) * X

dJ/db = ¼(Y_pred - Y)

W = W - α * dJ/dW

b = b - α * dJ/db

W = [5] - 0.01 * (¼ * [1 3 6 5] * [7 8 17 13])

B = [1 1 1 1] - 0.01 * (¼ * [7 8 17 13])

Linear Regression (cont)

Update weights (a = 0.01)

W = [5] - [1.98] = [4.505]

b = [0.9825 0.98 0.9575 0.9675]

Repeat for n epochs

(number of times the model goes through all examples)

Linear Regression (cont)

Why does this work?

Plot J(W,b) >> cost function

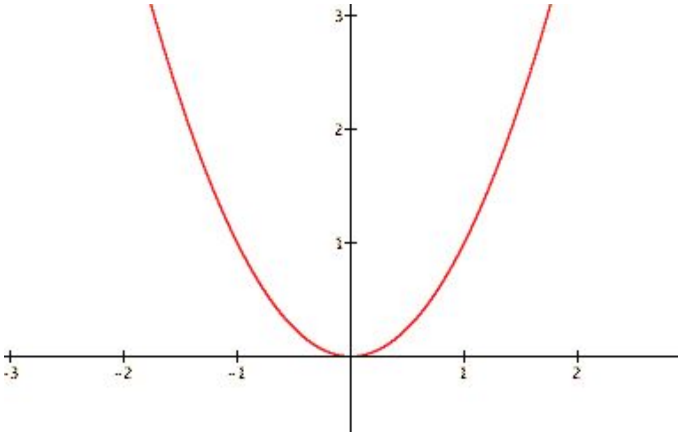
½(Y_pred - Y)^2

Plot dJ/dW

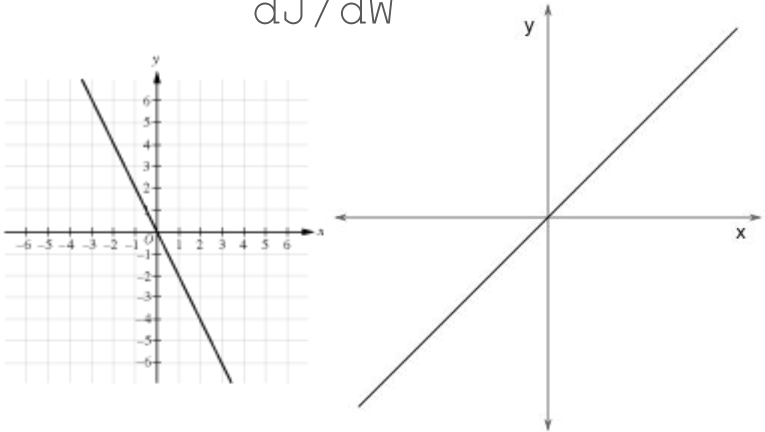
(Y_pred - Y) * X

W = W - a * dJ/dW

J(W,b)



dJ/dW



Linear Regression (cont)

Why does this work?

Plot J(W,b) >> cost function

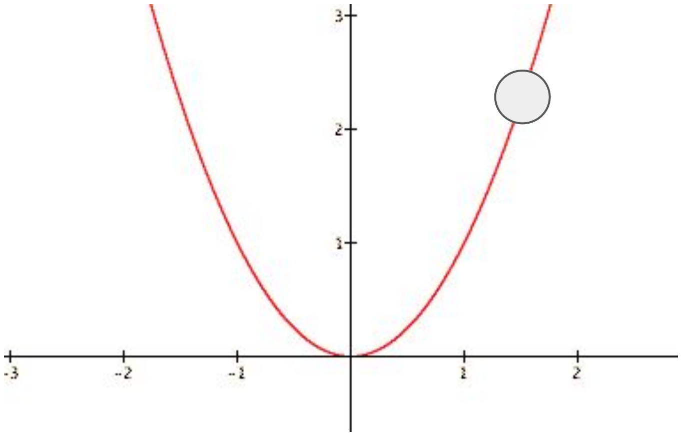
½(Y_pred - Y)^2

Plot dJ/dW

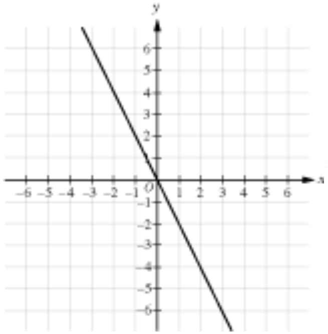
(Y_pred - Y) * X

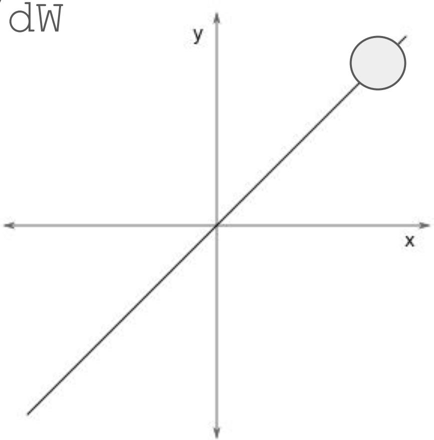
W = W - a * dJ/dW

J(W,b)



dJ/dW





Linear Regression (cont)

Why does this work?

Plot J(W,b) >> cost function

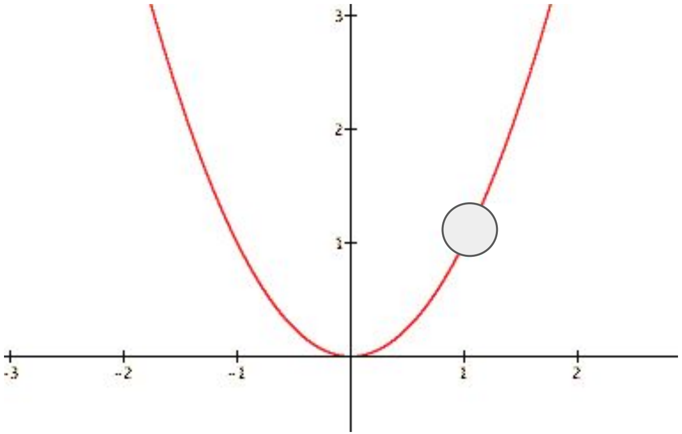
½(Y_pred - Y)^2

Plot dJ/dW

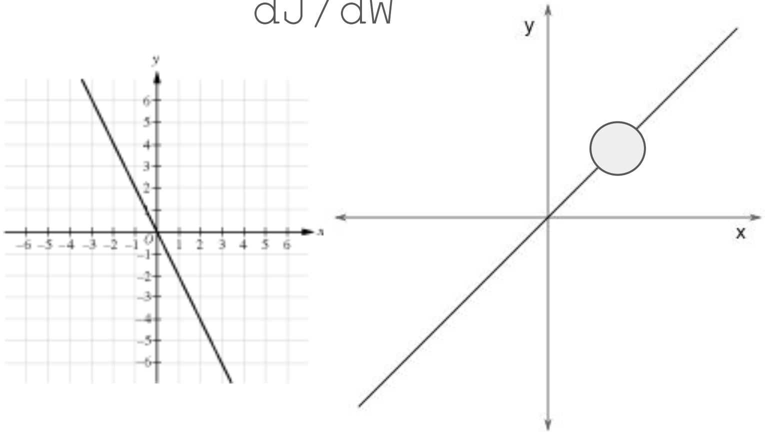
(Y_pred - Y) * X

W = W - a * dJ/dW

J(W,b)



dJ/dW



Linear Regression (cont)

Why does this work?

Plot J(W,b) >> cost function

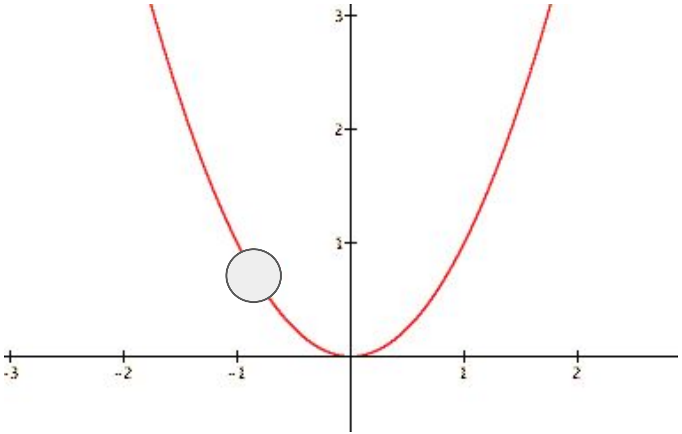
½(Y_pred - Y)^2

Plot dJ/dW

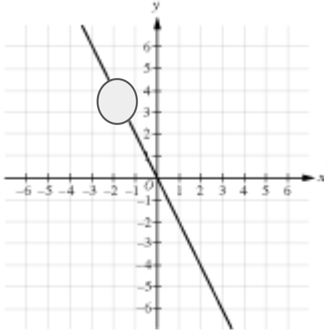
(Y_pred - Y) * X

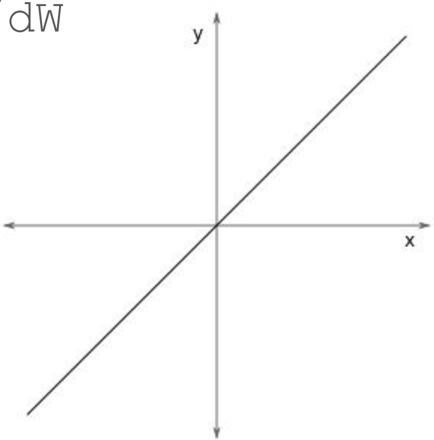
W = W - a * dJ/dW

J(W,b)



dJ/dW





Linear Regression (cont)

Why does this work?

Plot J(W,b) >> cost function

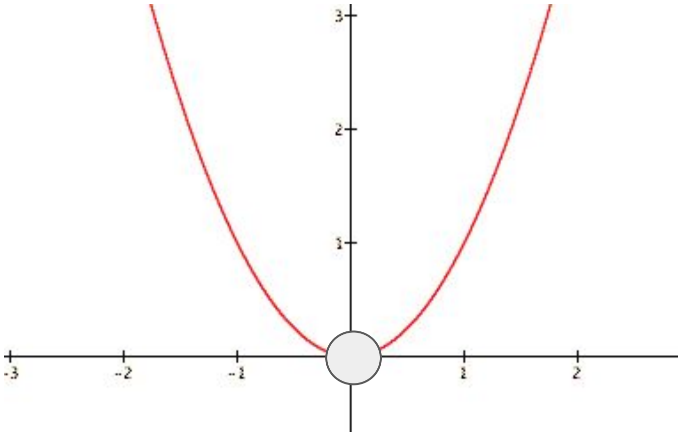
½(Y_pred - Y)^2

Plot dJ/dW

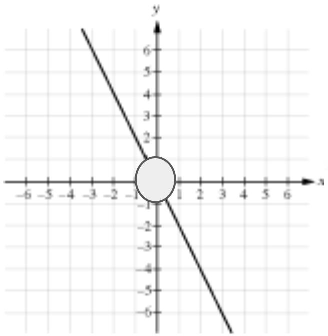
(Y_pred - Y) * X

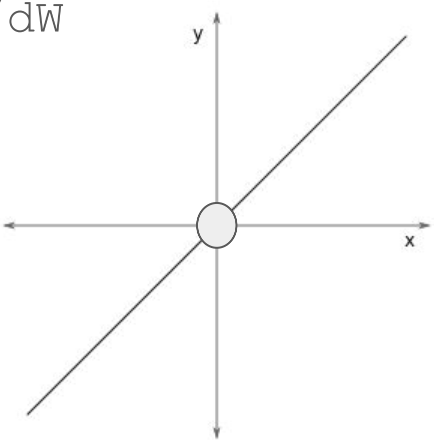
W = W - a * dJ/dW

J(W,b)



dJ/dW





Linear Regression (cont)

Example code:

https://colab.research.google.com/drive/16gDXqJlncbGhisdEryWthlcHl4WUZ2PI#scrollTo=IMG_q8aSxZmt

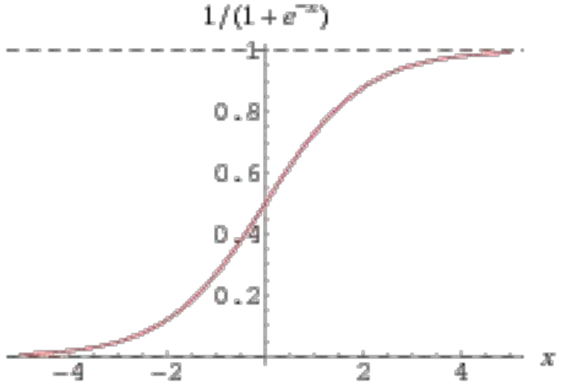
Logistic Regression

Not an independent algorithm, often used to non-linearize the model or used for classification

First run linear regression then pipeline the model to a logistic function

Logistic function outputs a 'logit'

ie) Sigmoid = 1/(1+exp(-y)) where y = WX + b



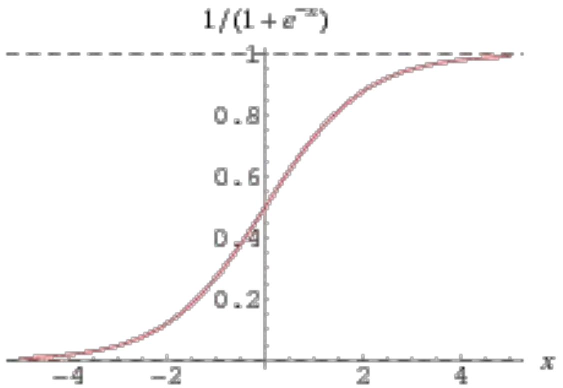
Logistic Regression

Sigmoid = 1/(1+exp(-x)) = σ(x) = g(x)

Note the bounds: This is used for binary classification

(0 or 1)

Use cases: Yes-no classification, Detection



Logistic Regression (cont)

Logistic Cost function:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

Logistic Regression (cont)

Logistic Cost function:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))]$$

Generalized Update Equations:

W = W - a * dJ/dW

dJ/dW = g’(y) * dy/dW

b = b - a * dJ/db

dJ/db = g’(y) * dy/dW

Logistic Regression (cont)

Logistic Cost function:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

Why does this work?

If y=1:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)}))$$

If y=0:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[\quad (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

Logistic Regression (cont)

Why does this work?

If y=1:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)}))$$

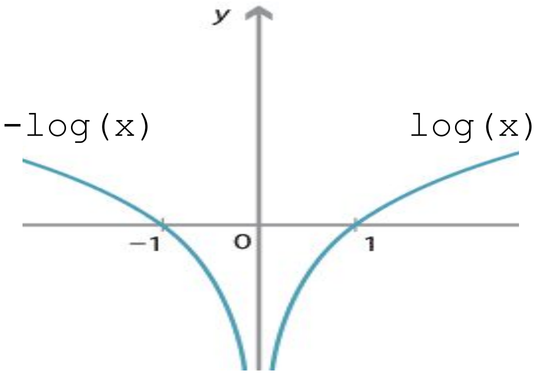
If y=0:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[\quad (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

Lets minimize the cost (to zero)...

If y=1: -y(log(y_pred)) = 0

If y=0: -(1-y)(log(1-y_pred)) = 0



Logistic Regression (cont)

Update Equations for Sigmoid:

```
    W = W - a * dJ/dW

dJ(g)/dW = 1/m * σ’(y_pred) * dy_pred/dW

         = 1/m * (σ(y_pred)-y) * x

    b = b - a * dJ/db

dJ(g)/db = 1/m * σ’(y_pred) * dy_pred/db

         = 1/m * (σ(y_pred)-y)
```

Repeat for n epochs

Logistic Regression (cont)

Proof using computation graph:

$$dJ(g)/dW = \sigma'(y\_pred) * dy\_pred/dW = (\sigma(y\_pred)-y) * x$$

$$dJ(g)/db = \sigma'(y\_pred) * dy\_pred/db = (\sigma(y\_pred)-y)$$

$$\boxed{y\_pred=W*X+b} \longrightarrow \boxed{A = \sigma(y\_pred)} \longrightarrow \boxed{J(A,y)}$$

$$dJ/dy\_pred = dJ/dA * dA/dy\_pred$$

$$dA/dy\_pred = A(1-A)$$

$$dJ/dW = dJ/y\_pred * dy\_pred/dW$$

$$dJ/dA= -y/A + (1-y)/(1-A)$$

Logistic Regression (cont)

Example code:

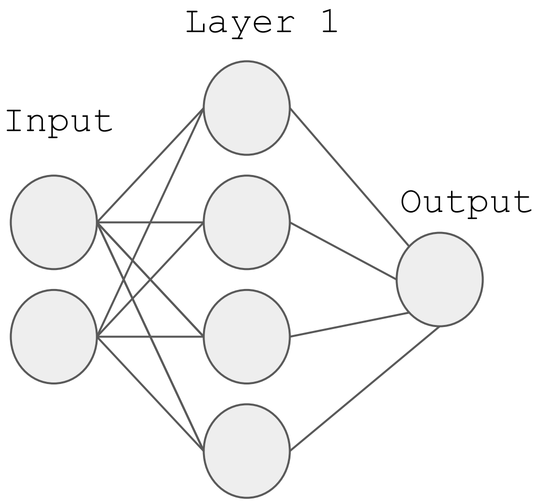
https://colab.research.google.com/drive/1NyhtWWhOu2xauZYDqB1wblGptS6jW_Mq

Single Layer Model

Each node has a weight and bias associated to it

Layer 1 can have an activation function, which is the logistic function (ie. sigmoid), thus each node within a layer has the same activation function.

Output has a weight and bias with an activation function(ie. sigmoid)



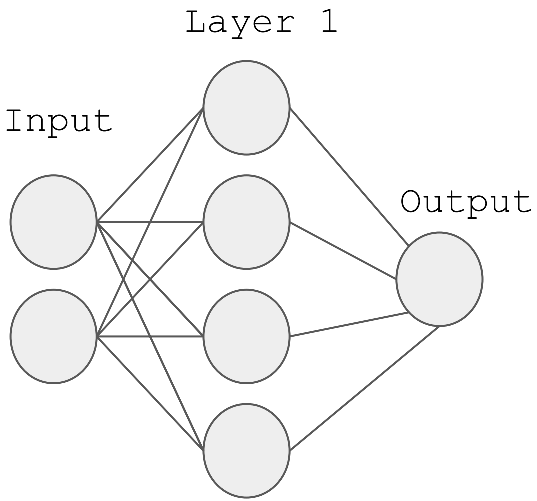
Single Layer Model

A model that has a layer with n nodes that computes logistic regression.

Input shape := (2,)

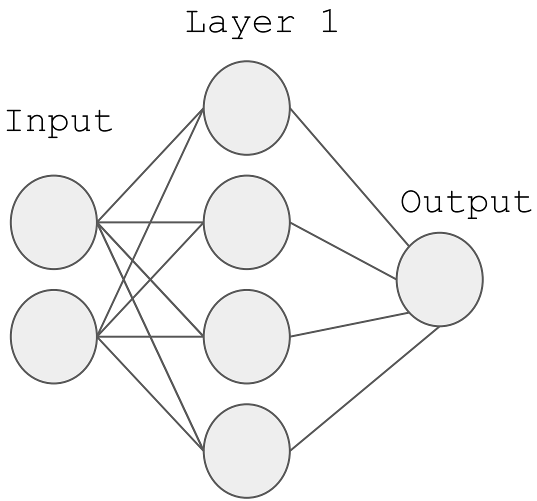
Layer 1 shape := (4,2) [[x1,x2][x1,x2][x1,x2][x1,x2]]

Output shape := (1,4)



Single Layer Model

Weights should be initialized randomly to avoid redundancy/symmetry, where bias does not matter too much, thus can be initialized to be all zeros or ones.

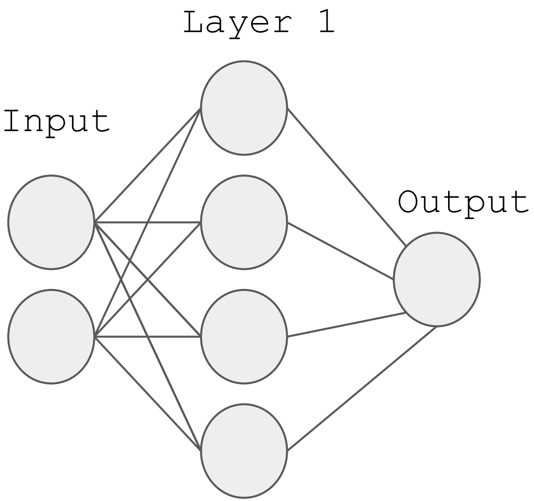


Single Layer Model

Propagation:

Forward - Data is being passed through the model to compute the output

Back - Model parameters is being updated via cost function optimization



Single Layer Model

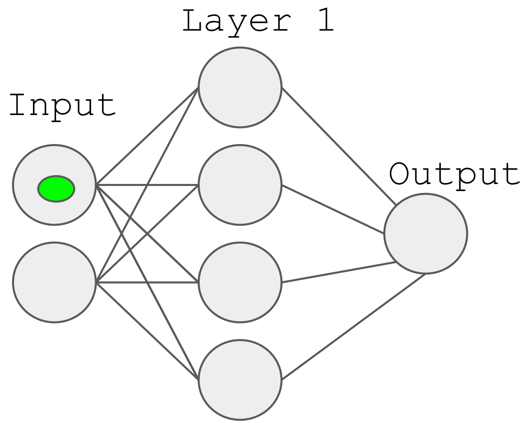
Propagation:

Forward - Data is fed into the input layer

Data.shape = (2,1)

Weight is initialized randomly (W1=np.random.rand(4,2))

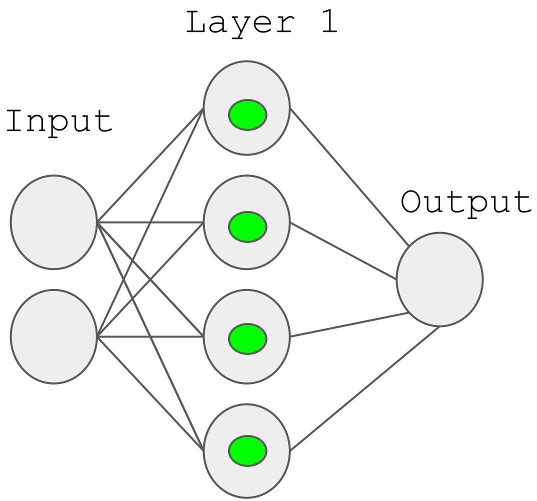
Bias is initialized as ones (b = np.ones(4,1))



Single Layer Model

Propagation:

Forward - Layer 1 applies weights and bias with an sigmoid activation

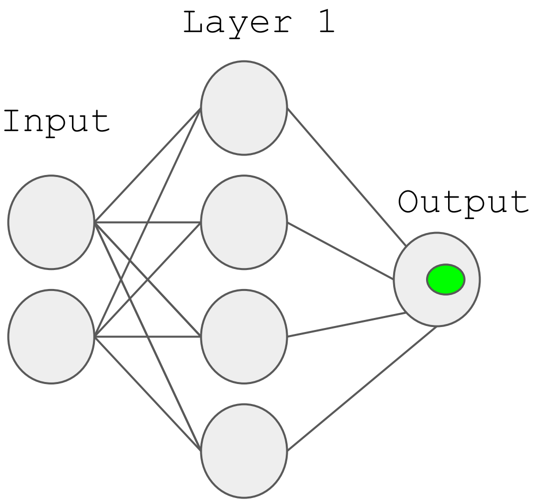


Single Layer Model

Propagation:

Forward - Output is computed

```
Output = 1/(1+np.exp(-(np.transpose(W)L1+b)))
```

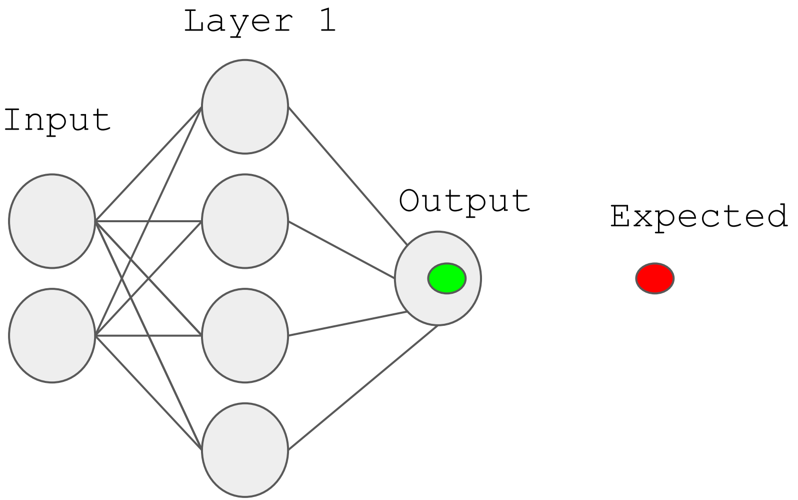


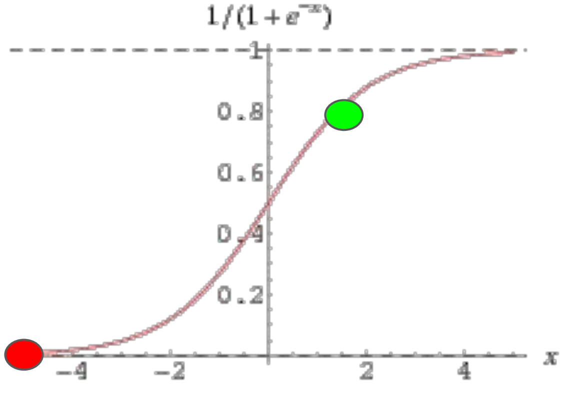
Single Layer Model

Backpropagation:

Cost is computed (assume m=1)

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

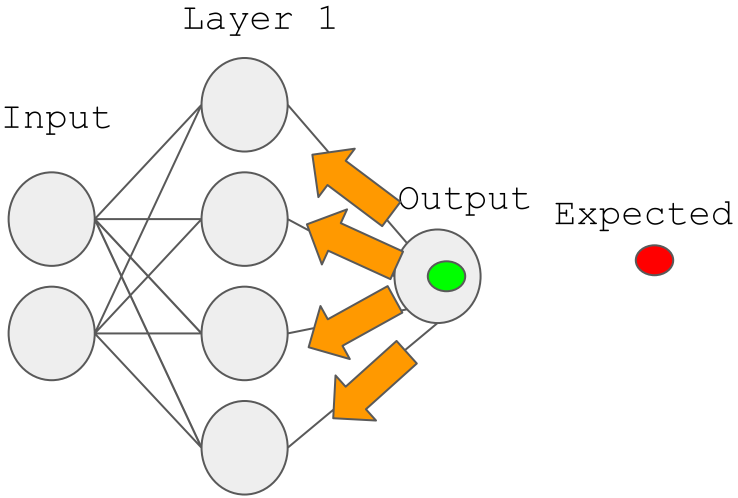




Single Layer Model

Backpropagation:

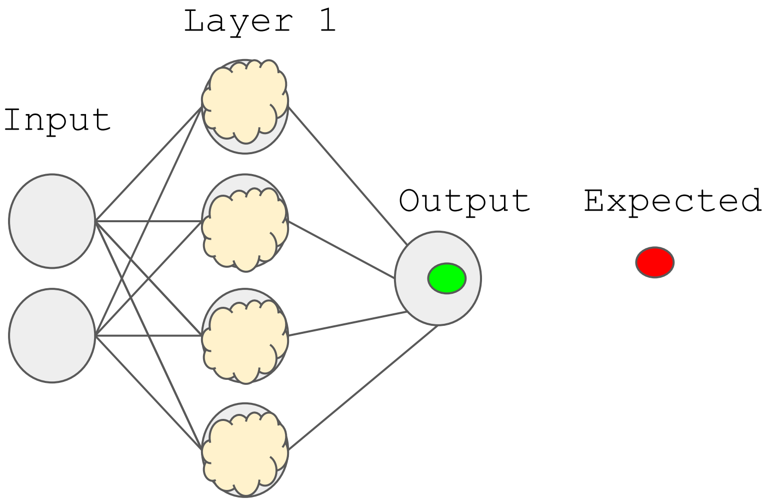
Calculate the gradient wrt parameters in output is first computed, then each node wrt each weight and bias in layer 1. With vectorization, W and b are matrices so that you don’t have to loop through each node in the layer.



Single Layer Model

Backpropagation:

Update the parameters of the model



Wout = Wout + a * dJ/dWout
bout = bout + a * dJ/dbout

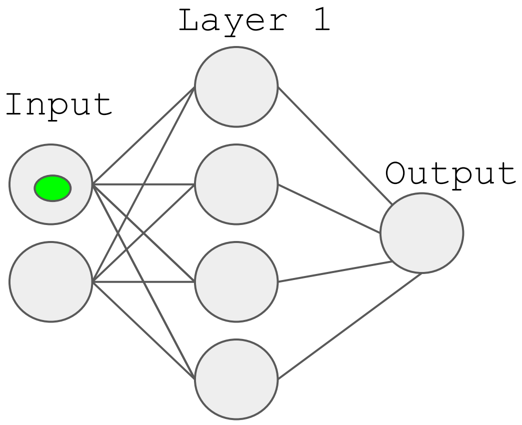
W1 = W1 + a * dJ/dW1
b1 = b1 + a * dJ/db1

Single Layer Model

Propagation:

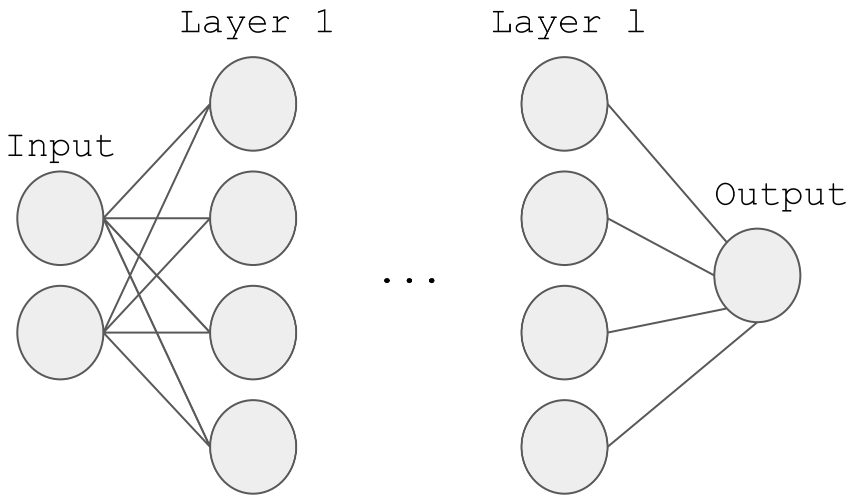
Repeat for n epochs

Forward - Data is fed into the input layer



Multi-Layer Perceptron

A model with l layers, each with their own independent number of nodes(perceptron) that will compute linear or logistic regression. The preceding layer output will be fed as the next layer’s input.



Multi-Layer Perceptron

Dimensions are important to note.

Input shape := (2,)

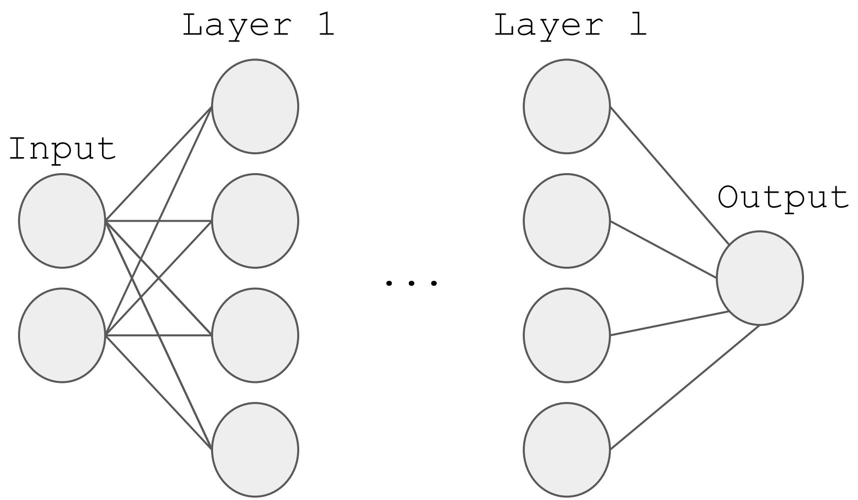
Layer 1 shape := (4,2)

Layer 2 shape := (n(2),4)

...

Layer l shape := (4,n(l-1))

Output shape := (1,4)

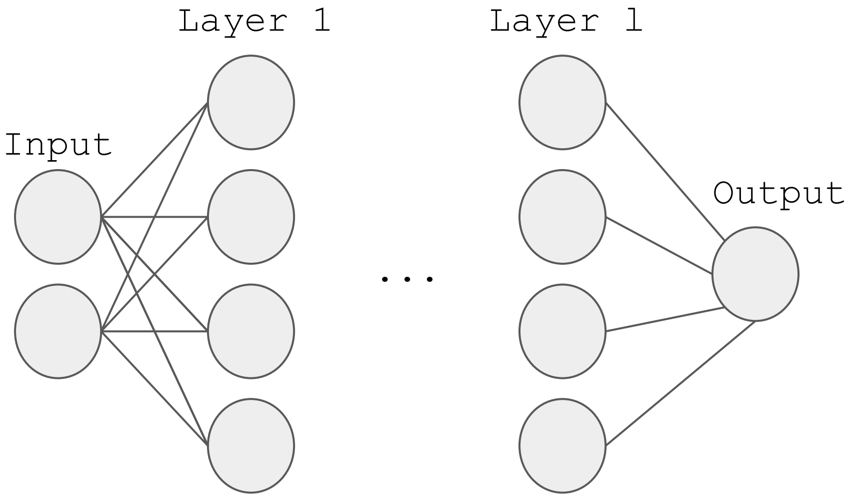


Multi-Layer Perceptron

Propagation

Forward - Data is fed into the model via input layer, is propagated to throughout the layers to compute the output.

Back - The model parameters are adjusted from the output layer all the way back to the first layer.



Multi-Layer Perceptron

Forward Propagation

Z1 = W1*Input + b1

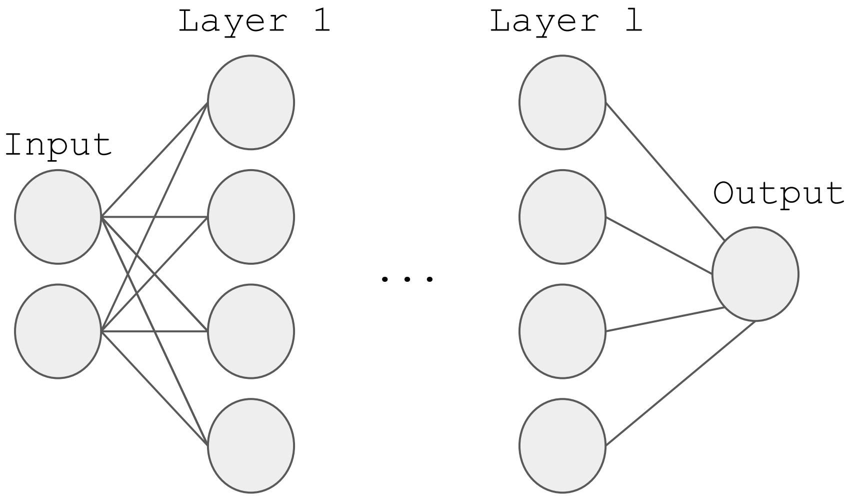
A1 = g(Z1)

...

Zl = Wl*Z(l-1) + bl

Al = g(Zl)

Output = g(Wout*Z(l) + bout)

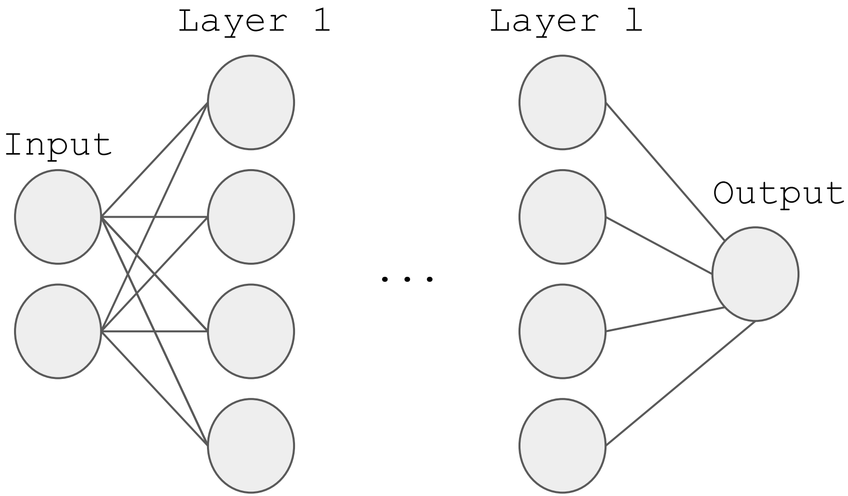


Multi-Layer Perceptron

Backpropagation

Compute Cost function…

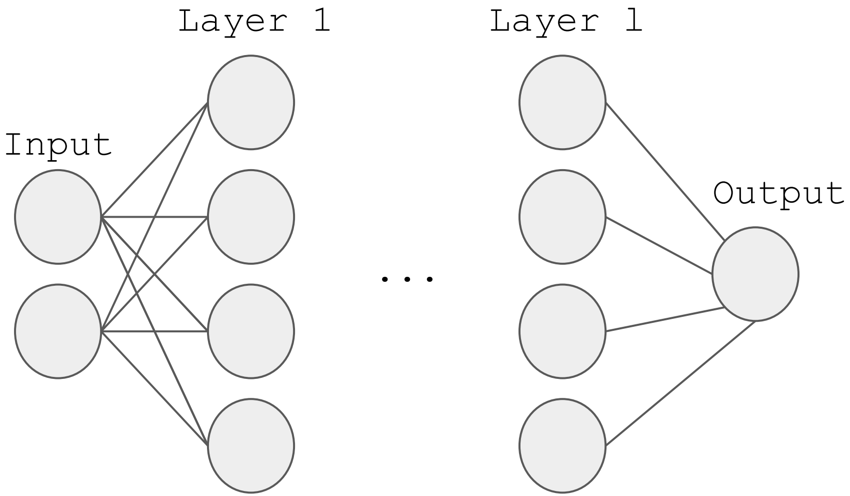
$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$



Multi-Layer Perceptron

Backpropagation

Compute gradient of the cost function wrt output parameters(Wout, bout), then layer l, then layer (l-1),... then layer 1.



Multi-Layer Perceptron

Backpropagation Equations

https://www.ics.uci.edu/~pjsadows/notes.pdf

Typically done with automatic/computational differentiation by frameworks via computational graphs which will allow abstraction from the calculations.

https://dlsys.cs.washington.edu/pdf/lecture4.pdf

Multi-Layer Perceptron

Backpropagation

Update the parameters of the model

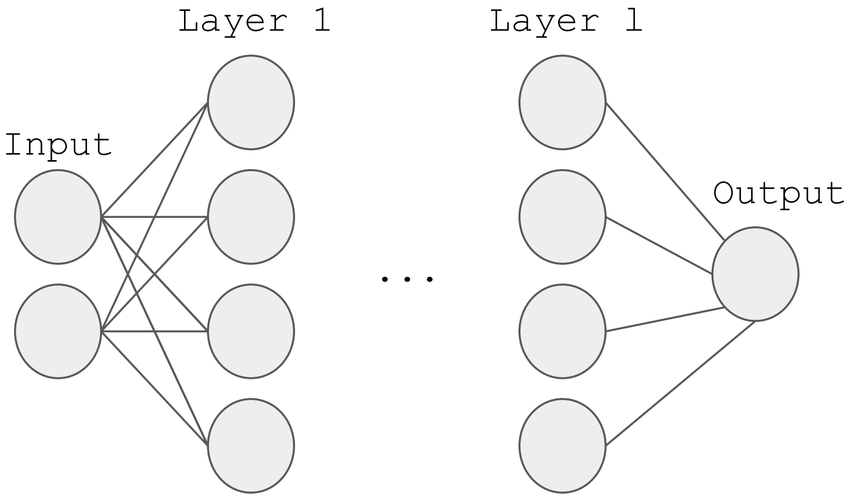
Wout = Wout + a * dJ/dWout

bout = bout + a * dJ/dbout

…

dJ/dW1 = W1 + a * dJ/dW1

dJ/db1 = b1 + a * dJ/db1



Multi-Layer Perceptron

Implementation:

https://colab.research.google.com/drive/1H-XRgRRrrKpmJQ92ZaF4TsnqDjEFd5TZ#scrollTo=27qsh0ye6u0O

Multi-class classification

Change the output activation to softmax function (aka multinomial logistic equation).

Takes the log probability of each label over the sum of all log probabilities.

Thus, determines the model’s confidence of each class.

$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

Loss Functions

Cross Entropy

- Binary
- Sparse

MSE

MAE

Hinge

Huber

And many more

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

$$L(y,\hat{y}) = -\sum y_{(i)} \log \hat{y}_{(i)}$$

$$\frac{1}{n}\Sigma\left(y - \widehat{y}\right)^2$$

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

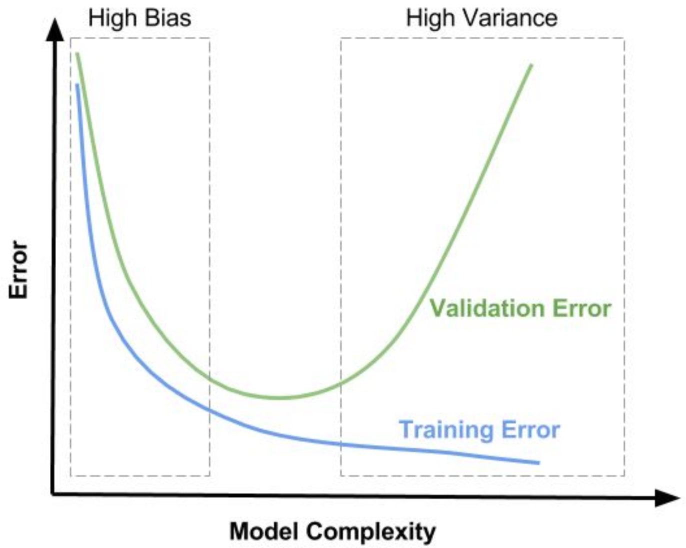
$$\frac{1}{n}\Sigma\left(y - \widehat{y}\right)$$

$$L = \begin{cases} \frac{1}{2}\left\|y - \bar{x}^T\bar{\theta}\right\|_2^2 & if \quad \left|y - \bar{x}^T\bar{\theta}\right| \leqslant t_H \\ t_H\left|y - \bar{x}^T\bar{\theta}\right| - \frac{t_H}{2} & otherwise \end{cases}$$

Issues with Deep Models

Notice how many parameters this model consist, and how computationally expensive it is to have that many parameters, especially with large datasets (>1gB of audio data)

The deeper the network, the greater complexity is going to be. This can lead to high training time or underfitting the data (high bias).

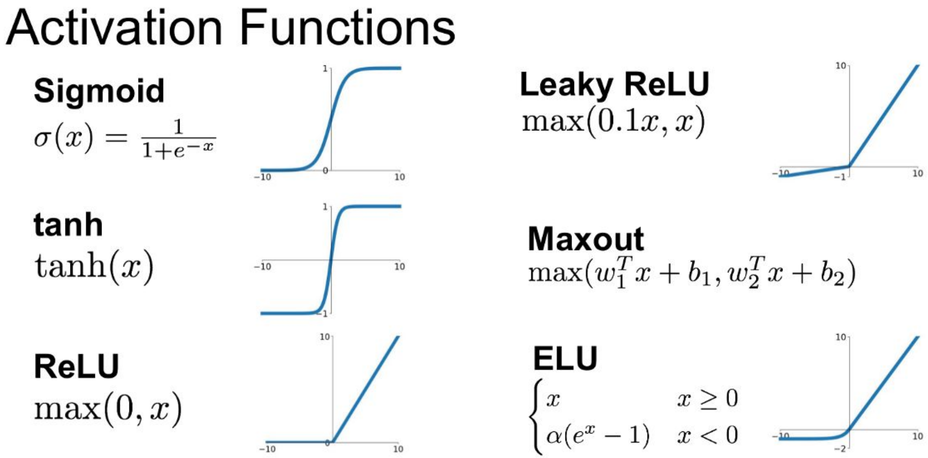


Issues with Models

To speed up training, you can normalize the data, tweak the hyperparameters, initialize model parameters more optimally, adjust activation functions ...

Activation Functions

Sigmoid, Tanh >> ReLU

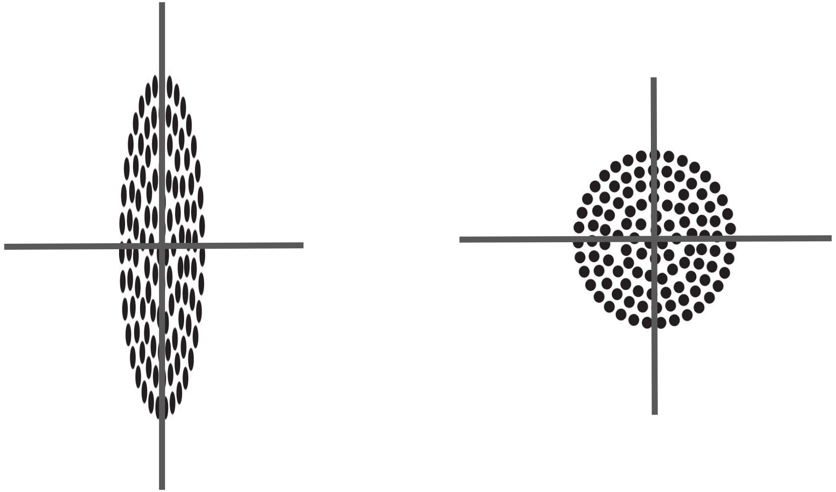


Issues with Models

Feature scaling/Normalization:

Mean normalization: $$x' = \frac{x - \text{average}(x)}{\max(x) - \min(x)}$$

Standardization: $$x' = \frac{x - \bar{x}}{\sigma}$$



Batch Normalization:

Normalizes within hidden layers

Issues with Models

What if your model does really well on the training dataset, but does poorly on the validation and testing dataset?

Your model is overfitted to the training dataset, and has a variance problem.

Use regularization by introducing a new regularizing variable (R) scaled by a hyperparameter lambda into the loss function

- L2 regularization
  - Penalizes the weights from being too big
    - R = sum(W^2) over all W

Issues with Models

L2 regularization

Penalizes the weights from being too big

R = mean(W^2) over all W

J = ½ mean(y_pred-y)^2 + lambda * R

W := W*(1-a*lambda/m) - a * dJ/dW (w/o lambda * R)

How does this help?

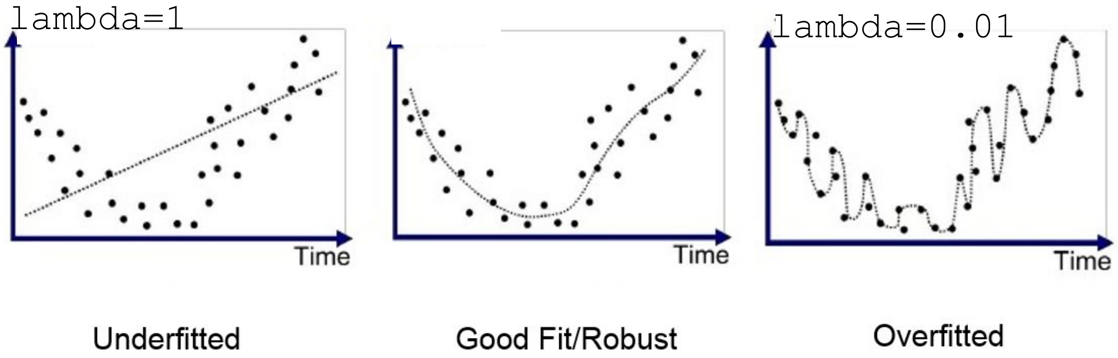
Weights are discounted as much as possible as the loss is being minimized. With larger lambda, the less the model will fit to your data (underfitting). With smaller lambda, the more the model will fit to your data (overfitting).

Issues with Models

L2 regularization

How does this help?

Weights are discounted as much as possible as the loss is being minimized. With larger lambda, the less the model will fit to your data (underfitting). With smaller lambda, the more the model will fit to your data (overfitting).

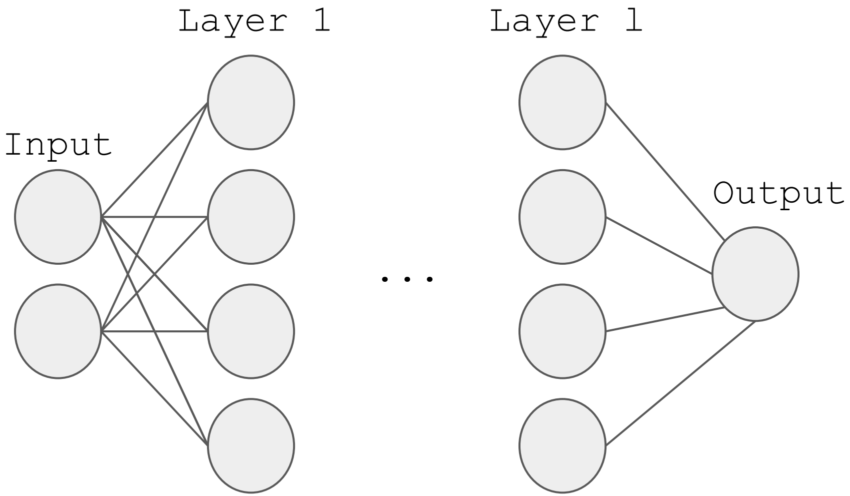


Issues with Deep Models

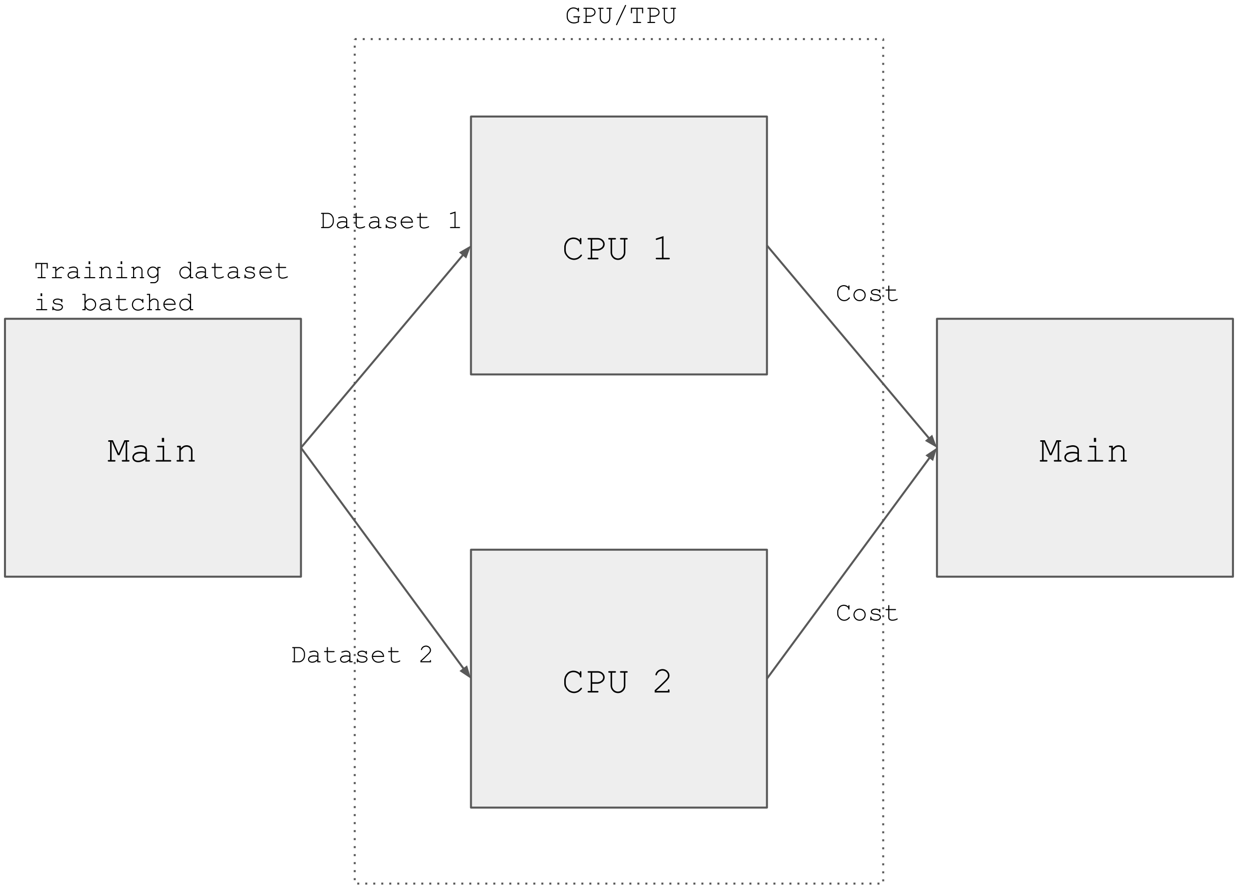
Notice how many parameters this model consist, and how computationally expensive it is to have that many parameters, especially with large datasets (>1gB of audio data)

GPUs/TPUs to alleviate cost computation:

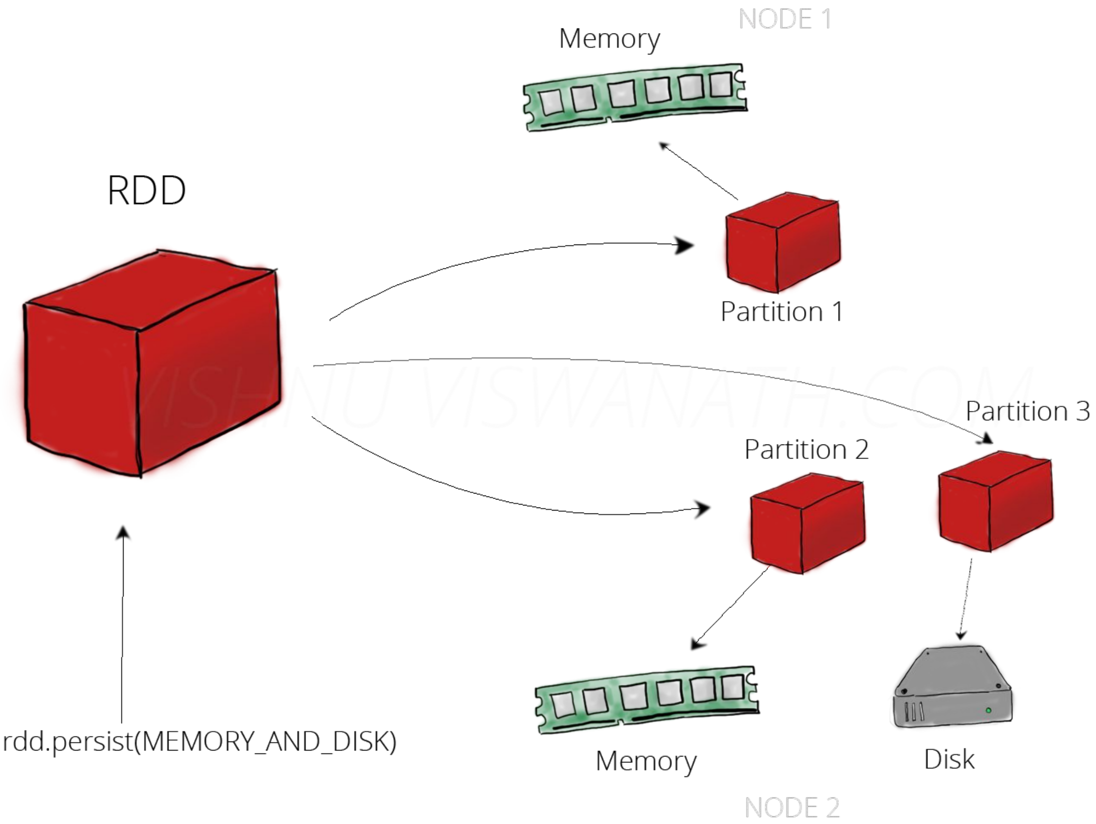
Multithreading and parallel computing



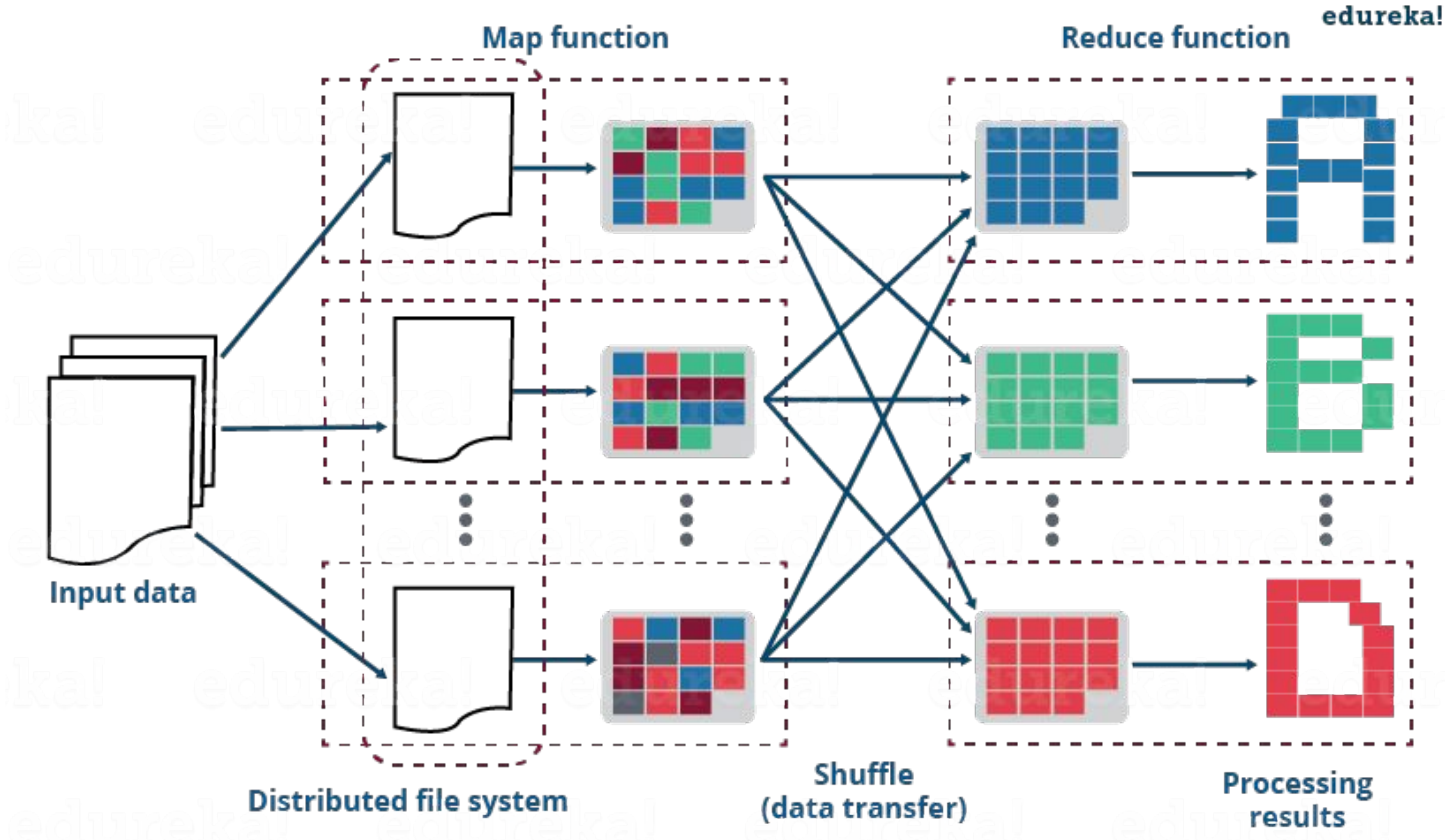
Parallel computing



Resilient Distributed Data (Spark)



MapReduce (Hadoop)



Batching our Data

Batch Gradient Descent (n=m)

Update after going through every example

Mini-batch Gradient Descent (1<n<m)

Update after going through n examples

Stochastic Gradient Descent (n=1)

Update after going through 1 example

Batching our Data

Mini-batch Gradient Descent ($1<n<m$)

Update after going through n examples

Mini-batching is considered to be optimal since it takes advantage of vectorization and parallel computing

Batching our Data

What if we want to do real-time learning?

Use online learning approach

Use stochastic learning on incoming data, then disregard the data after a single training iteration.

Batching our Data

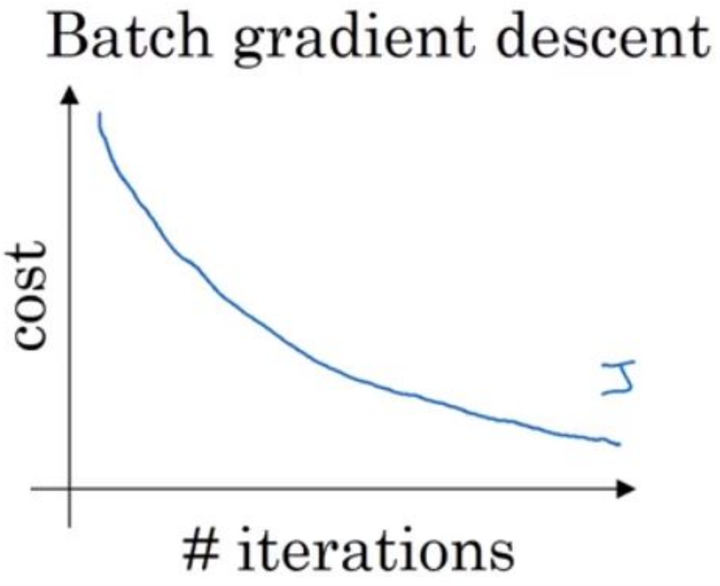
Comparison between stochastic, minibatch, and batch learning

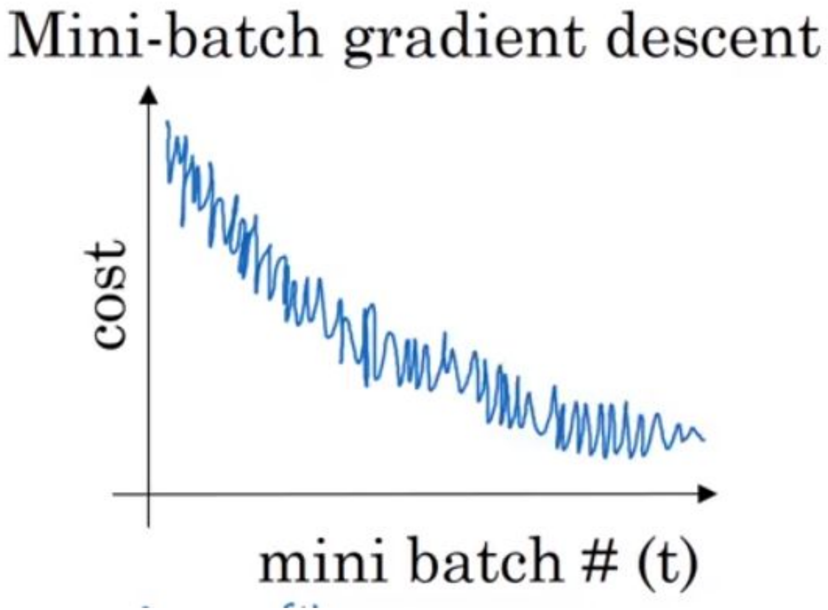
Stochastic has fastest update, but convergence of cost is difficult to know since the cost function graph is messy

Minibatch utilizes vectorization, and convergence of cost can generally be seen.

Batch has the slowest updates, but cost will always decrease if implementation is correct.

Batching our Data





Optimization Algorithms

Momentum

$$Vd_w = \beta_1 * Vd_w + (1-\beta_1) * d_w$$
$$Vd_b = \beta_1 * Vd_b + (1-\beta_1) * d_b$$

$\Big\}$ "momentum"-like update

RMSprop

$$Sd_w = \beta_2 * Sd_w + (1-\beta_2) * {d_w}^2$$
$$Sd_b = \beta_2 * Sd_b + (1-\beta_2) * {d_b}^2$$

$\Big\}$ "RMSprop"

ADAM

Standard:
Beta1 = 0.9
Beta2 = 0.999
E = 10*10**-8

$$V_{d_w}^{corrected} = Vd_w \,/\, (1-\beta_1^t)\ , \qquad V_{d_b}^{corrected} = Vd_b \,/\, (1-\beta_1^t)$$
$$S_{d_w}^{corrected} = Sd_w \,/\, (1-\beta_2^t)\ , \qquad S_{d_b}^{corrected} = Sd_b \,/\, (1-\beta_2^t)$$

$$w := w - \alpha * \frac{V_{d_w}^{corrected}}{\sqrt{S_{d_w}^{corrected} + \varepsilon}}$$

$$b := b - \alpha * \frac{V_{d_b}^{corrected}}{\sqrt{S_{d_b}^{corrected} + \varepsilon}}$$

ML/Deep Learning Frameworks

Python:

- Keras
- Tensorflow
- PyTorch
- Caffe (and Matlab)
- Scikit Learn

Java/Scala:

- Deeplearning4j
- MOA

ML/Deep Learning Frameworks

Why use frameworks?

Capability to use GPUs

Optimal implementation

Easier to build models

For Keras…

```
import keras

model = keras.models.Sequential()
model.add(keras.layers.Dense(units = 10, input_shape=(16,)))
model.add(keras.layers.Activation(‘sigmoid’))
model.compile(optimizer='sgd', loss='mean_squared_error',metrics=['accuracy'])
model.fit(x=features, y=labels, batch_size=16, epochs=100)
score = model.evaluate(x_test, y_test, batch_size=16)
prediction = model.predict(unseen_features) #inference
```

ML/Deep Learning Frameworks

For SciKit Learn

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(x=features, y=labels)
model.score(x=x_test, y=y_test)
model.predict(unseen_features)
```

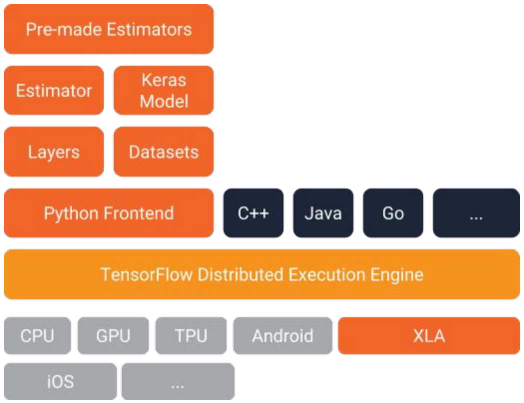
For Tensorflow

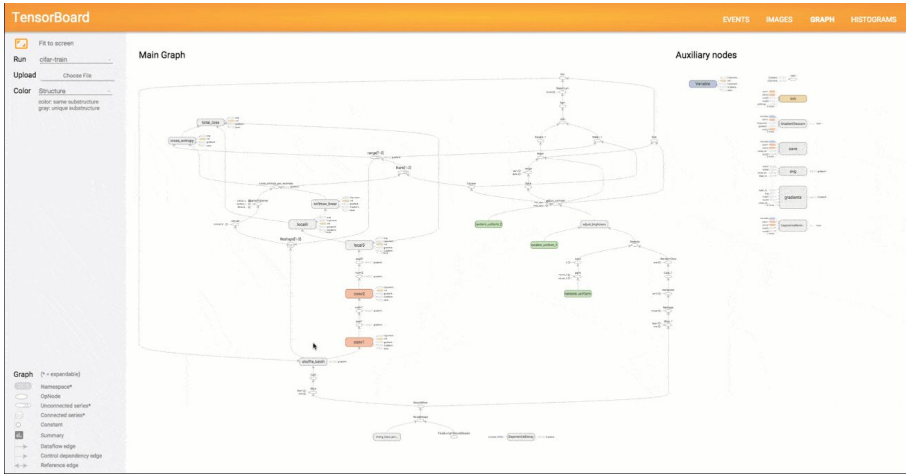
```
import tensorflow as tf

X = tf.placeholder(shape=(16,), dtype=tf.float16)
Y = tf.placeholder(shape=(10,), dtype=tf.float16)
W1 = tf.get_variable("W1", [25, 12288], initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.get_variable("b1", [25, 1], initializer = tf.zeros_initializer())
Z1 = tf.add(tf.matmul(W1, X), b1)
A1 = tf.nn.sigmoid_cross_entropy_with_logits(Z1)
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=A1, labels=labels))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
init = tf.global_variables_initializer()
with tf.Session() as sess:
      sess.run(init)
      for epoch in range(num_epochs):
            _ , epoch_cost = sess.run([optimizer, cost], feed_dict={X:features, Y: labels})
prediction = tf.nn.sigmoid_cross_entropy_with_logits(tf.add(tf.matmul(W1, unseen_X), b1))
```

ML/Deep Learning Frameworks

Tensorflow as Backend can be used for Keras models with:
from tensorflow.python import keras

Why Tensorflow?
The core of Tensorflow is written in a combination of highly-optimized C++ and CUDA, thus it is optimized for GPU capability. Based on TensorFlow computation graphs, computation is executed by fast C++ code





Building a classification model on MNIST

Frameworks: Tensorflow, numpy, *matplotlib

Link: https://drive.google.com/open?id=1vR3Cs4xQdOXmU5QyLeF_A02PE2LCqqxo

Shortened: **https://goo.gl/hmCm1q**

URL: https://goo.gl/

Next time...

Introduction to Convolutional Neural Networks

ConvNet, YOLO, ResNet, Inception, and more

Introduction to Sequential Models

Recurrent Net, LSTMs, GRUs, Encoder-Decoder, and more