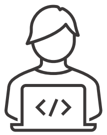
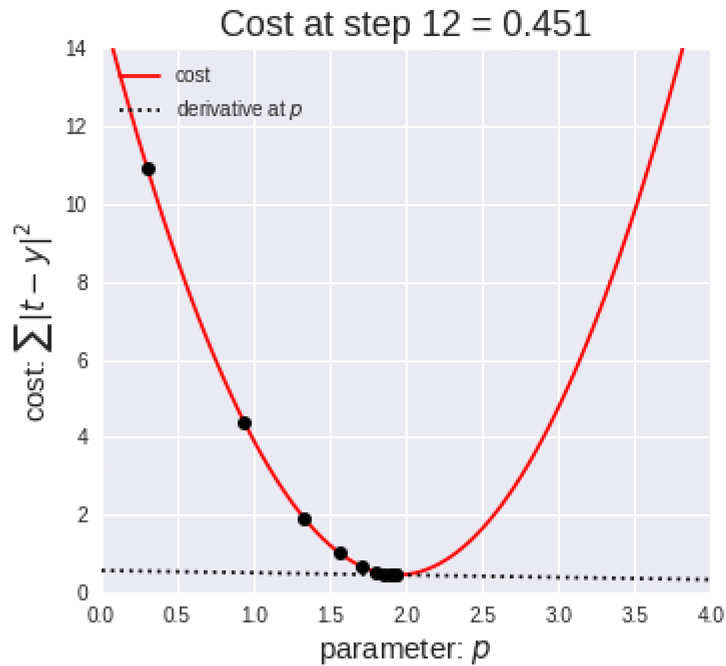
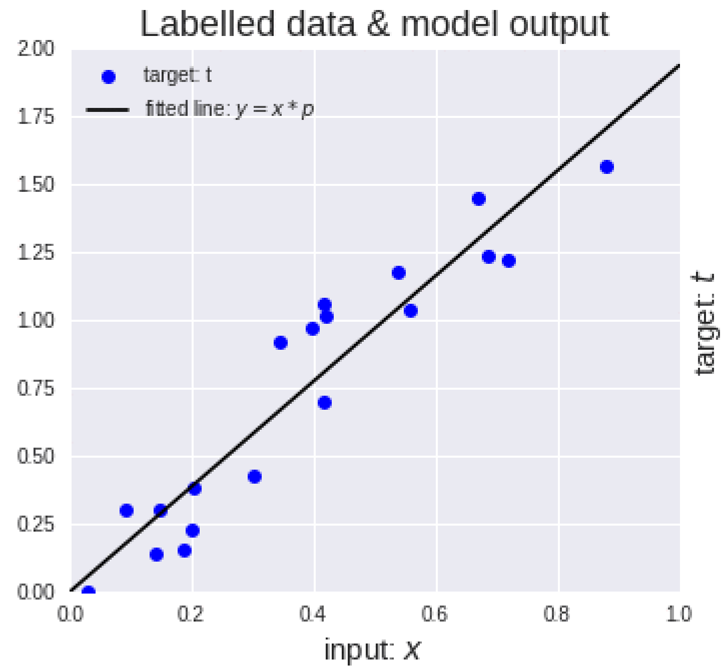
# DEEP LEARNING FOR INVESTING

- FOREX
- CRYPTO
- หุ้น
- กองทุน
- ทอง



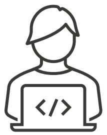
# Gradient Descent Optimization




Source: https://medium.com/onfido-tech/machine-learning-101-be2e0a86c96a

INVESTIC



# Gradient Descent Optimization

Cost

Initial
Weight

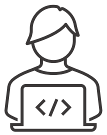
Gradient

Incremental
Step

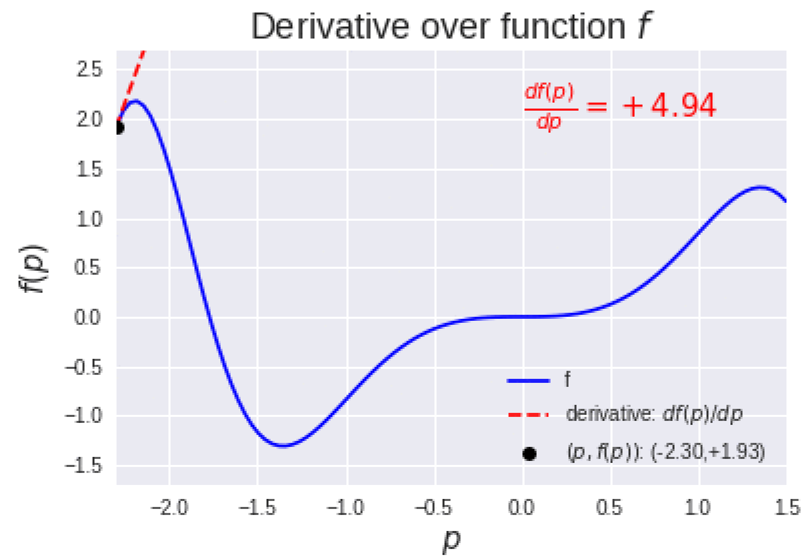
Minimum Cost

Derivative of Cost

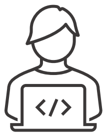
Weight



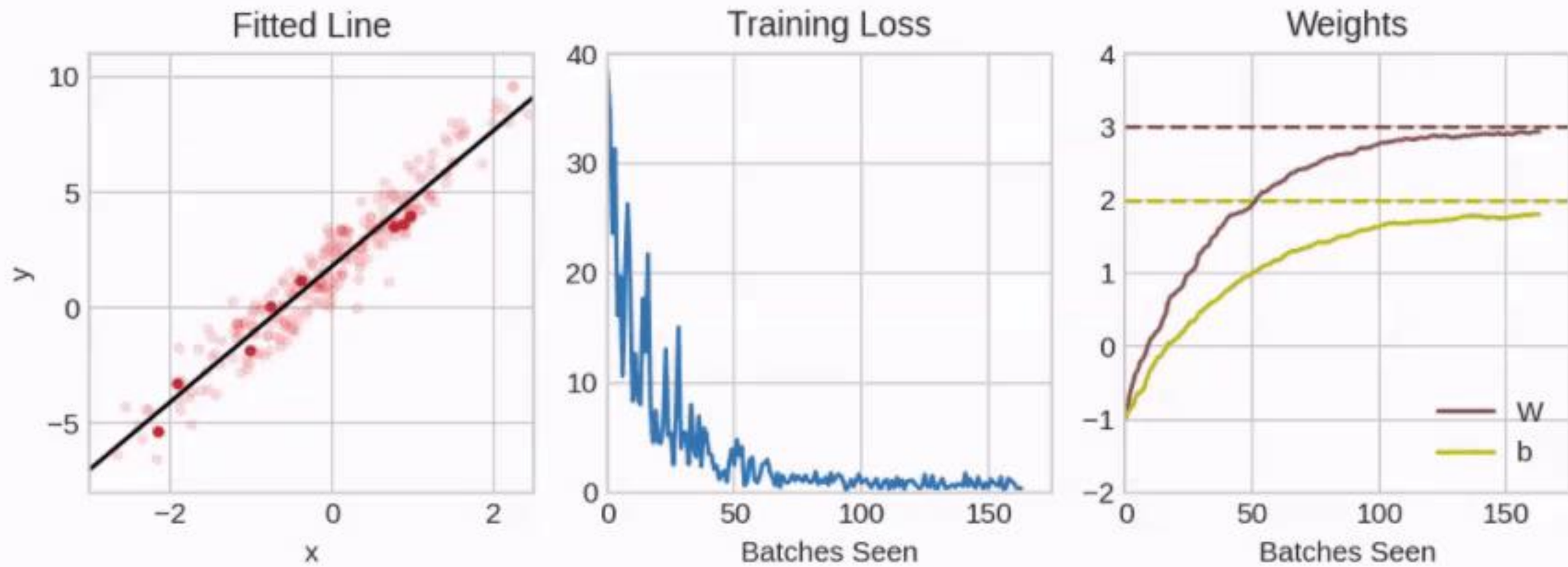
# Gradient Descent Optimization




INVESTIC

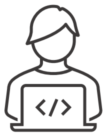


# Training Neural Network in action

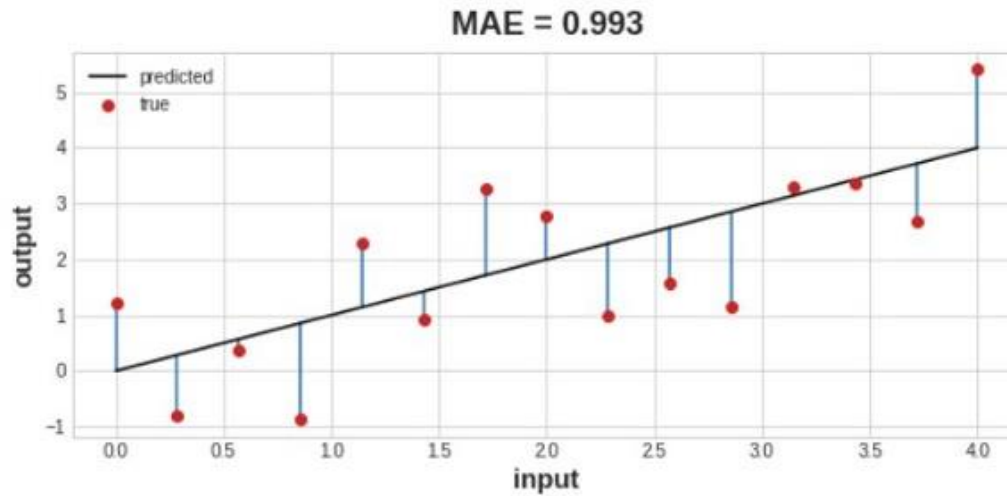



*Training a neural network with Stochastic Gradient Descent.*

INVESTIC

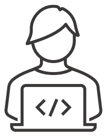


# Same2 loss function

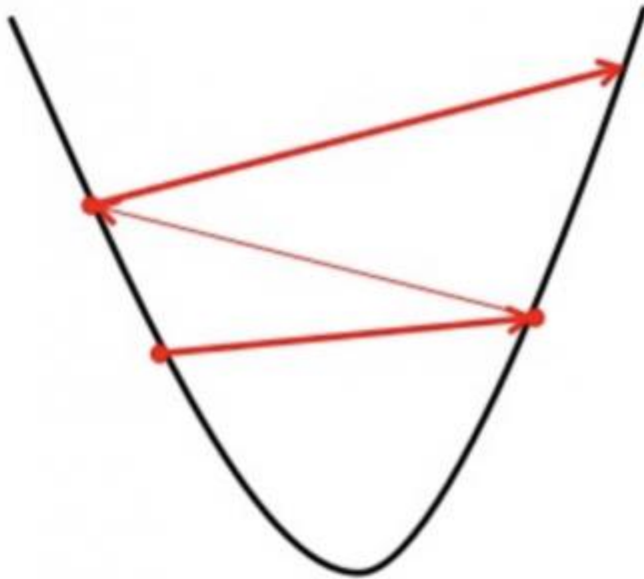


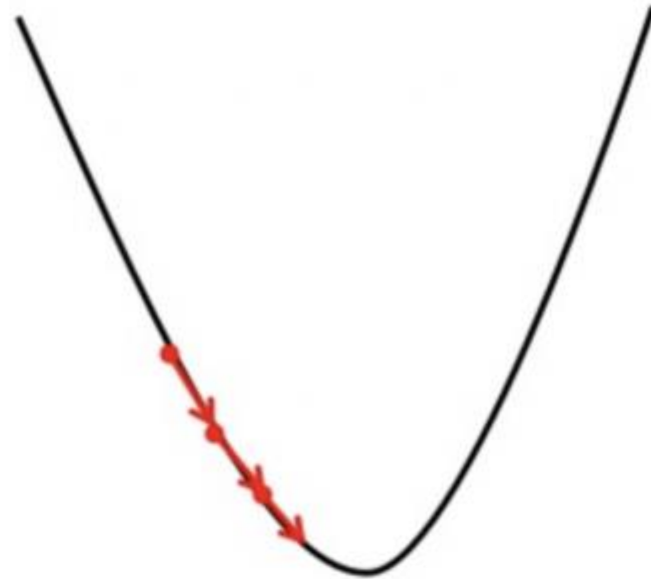

*The mean absolute error is the average length between the fitted curve and the data points.*

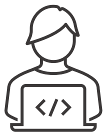
INVESTIC



# Learning Rate

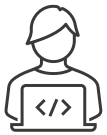




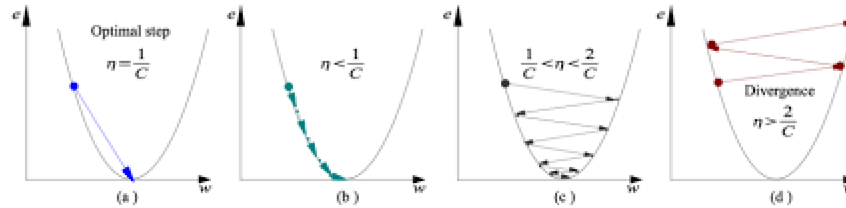
# Too Large Learning Rate

INVESTIC

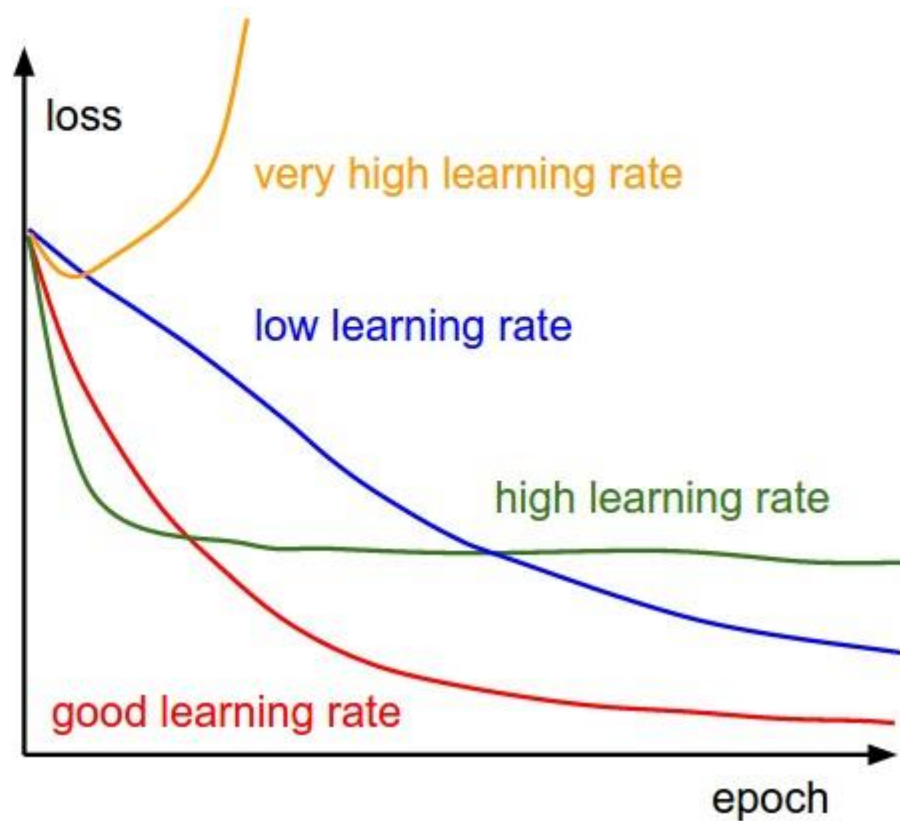


# Learning Rate

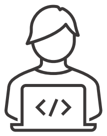
Source: Coursera



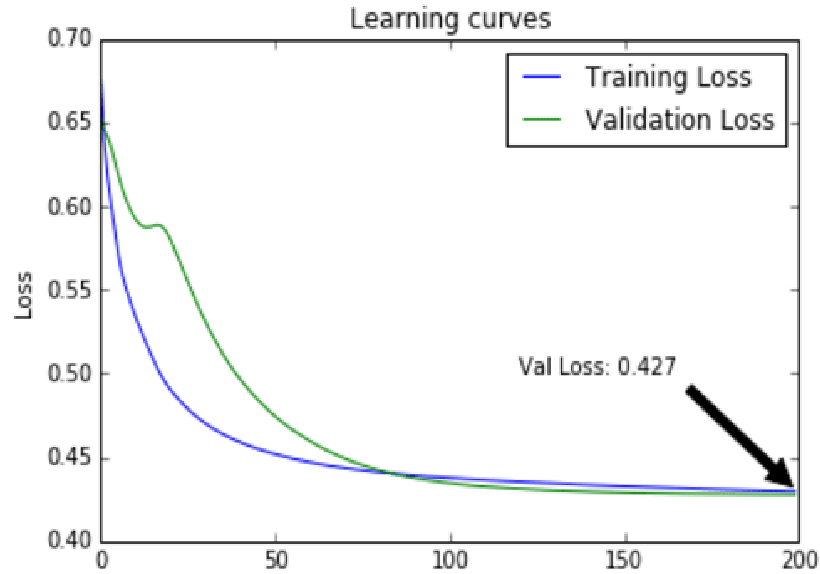

Source: researchgate

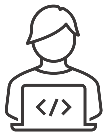



INVESTIC



# Visualized Training Model

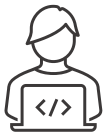




# Visualized Training Model

```
history = model.fit(
    X_train, y_train,
    validation_data=(X_valid, y_valid),
    batch_size=256,
    epochs=10,
)
```

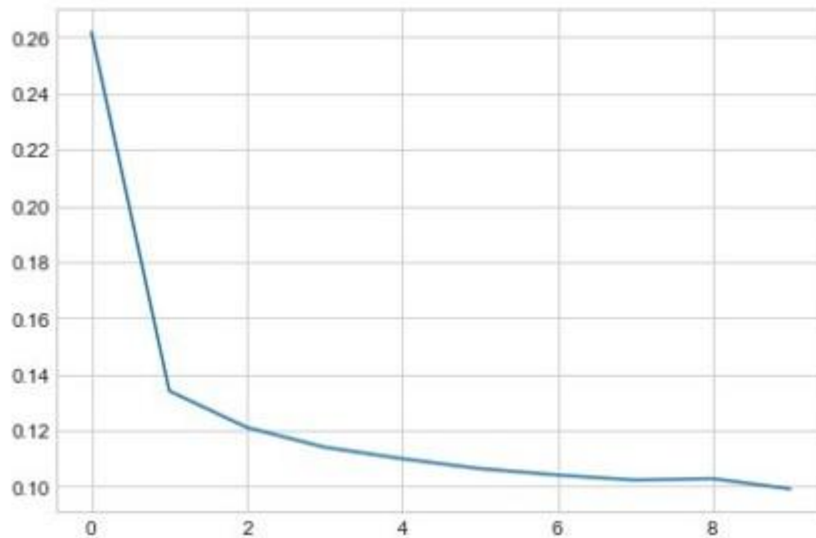
```
Epoch 1/10
5/5 [==============================] - 0s 40ms/step - loss: 0.2617 - val_loss: 0.1332
Epoch 2/10
5/5 [==============================] - 0s 17ms/step - loss: 0.1341 - val_loss: 0.1221
Epoch 3/10
5/5 [==============================] - 0s 18ms/step - loss: 0.1211 - val_loss: 0.1167
Epoch 4/10
5/5 [==============================] - 0s 17ms/step - loss: 0.1140 - val_loss: 0.1082
Epoch 5/10
5/5 [==============================] - 0s 24ms/step - loss: 0.1099 - val_loss: 0.1093
Epoch 6/10
5/5 [==============================] - 0s 22ms/step - loss: 0.1063 - val_loss: 0.1019
Epoch 7/10
5/5 [==============================] - 0s 22ms/step - loss: 0.1041 - val_loss: 0.1008
Epoch 8/10
5/5 [==============================] - 0s 16ms/step - loss: 0.1023 - val_loss: 0.1000
Epoch 9/10
5/5 [==============================] - 0s 18ms/step - loss: 0.1028 - val_loss: 0.1004
Epoch 10/10
5/5 [==============================] - 0s 18ms/step - loss: 0.0991 - val_loss: 0.1029
```

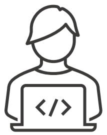


# Visualized Training Model

```
import pandas as pd

# convert the training history to a dataframe
history_df = pd.DataFrame(history.history)
# use Pandas native plot method
history_df['loss'].plot();
```

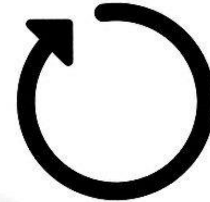




# In each loop

## Epoch :

An Epoch represent one iteration over the entire dataset.

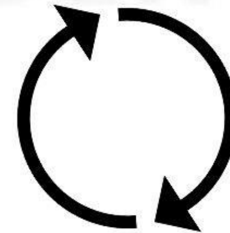


## Batch :

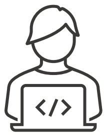
We cannot pass the entire dataset into the Neural Network at once. So, we divide the dataset into number of batches.

## Iteration :

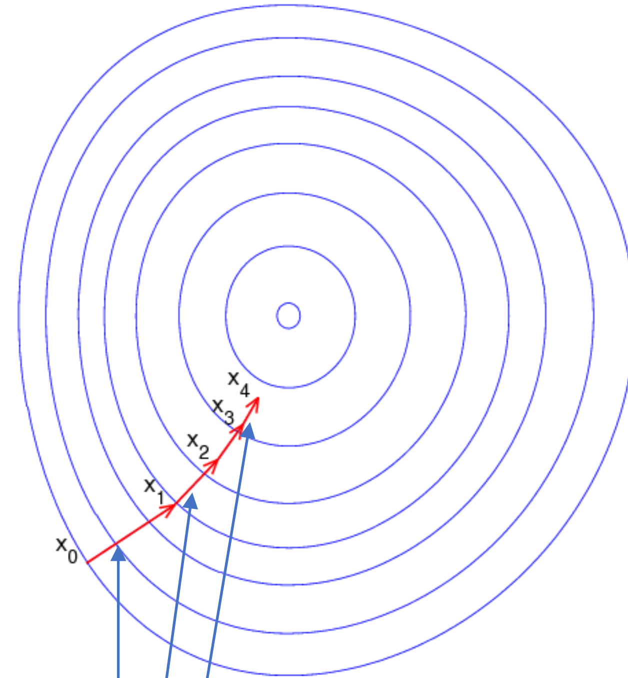
If we have 1000 images as Data ane a batch size of 20, then an Epoch should run 1000/20 = 50 iteration.



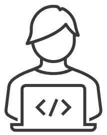
INFINITYCODEX



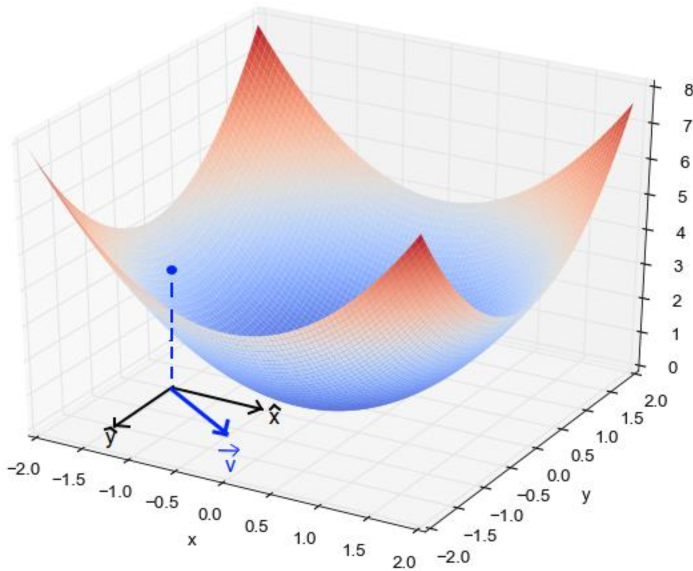
# Real life

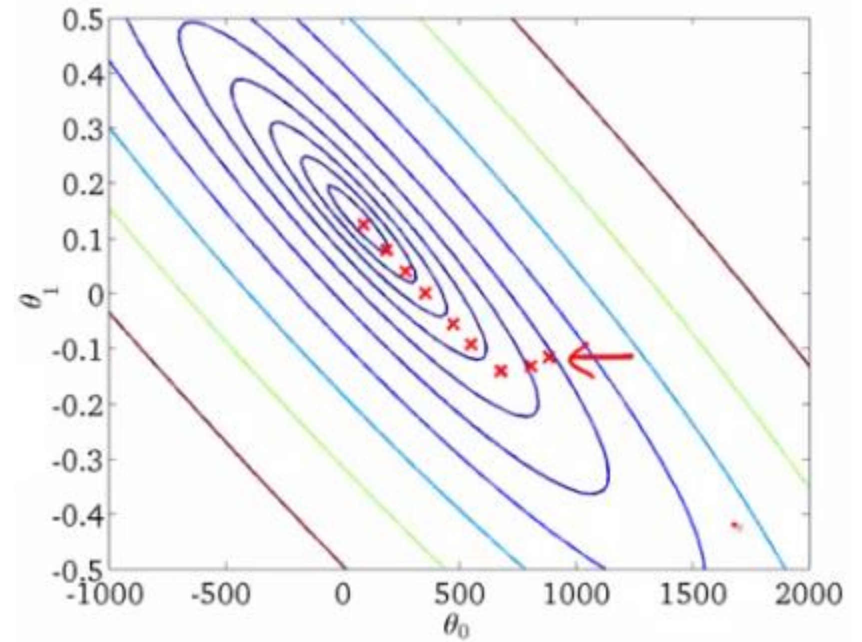


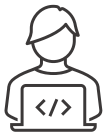

Different in
Steep/Slope



# 3D vs 2D .. Contour plot





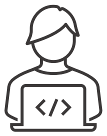
# Trapped !!??

A

Local Minima

Global Minima

Saddle Point



# Trapped !!??

$x_{t+1}$ $x^*$

$y_{t+1}$

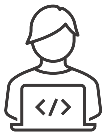
$x_t$

$y_t$

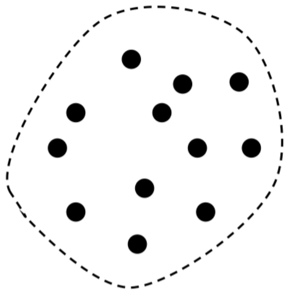
3 2 1 0 −1

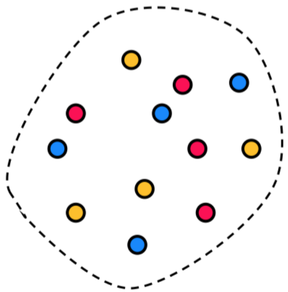
−1 −0.5 0

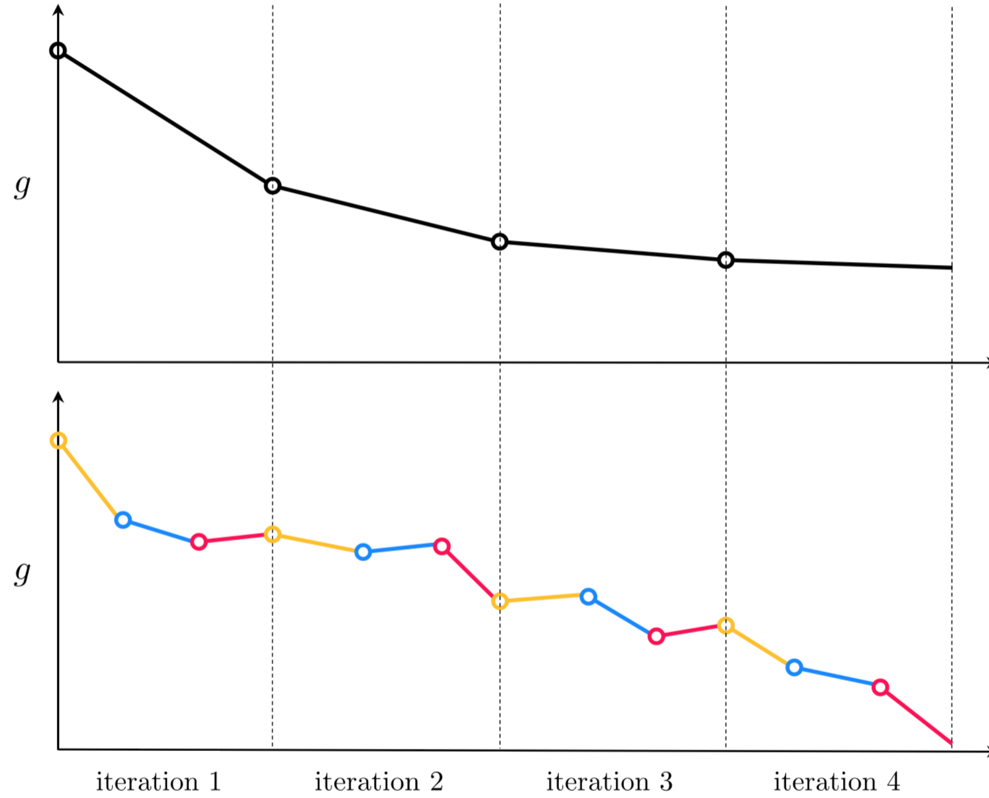
0 −0.5

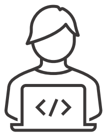


# RNG them???



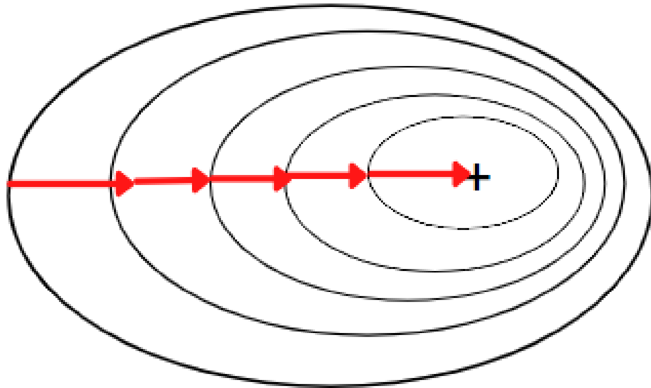




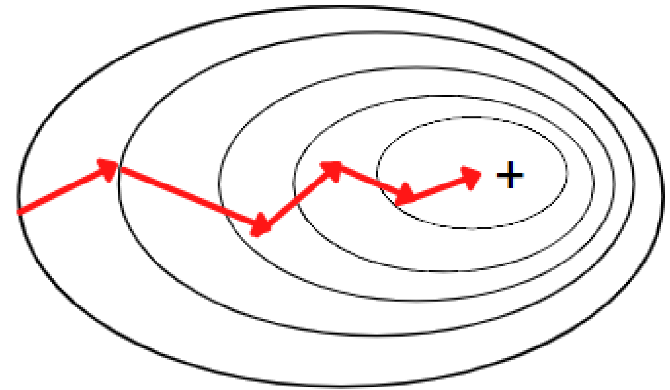


# Batch

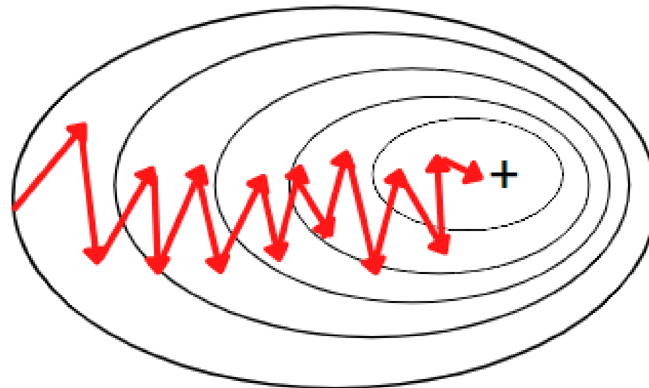
## Batch Gradient Descent

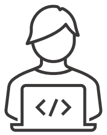



## Mini-Batch Gradient Descent

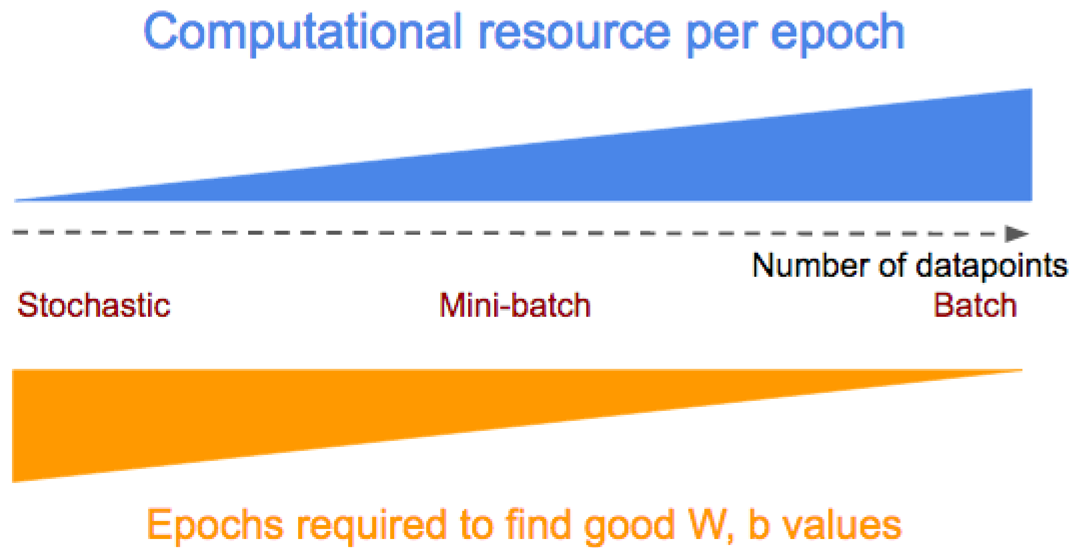



## Stochastic Gradient Descent

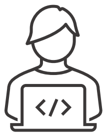




# Batch




INVESTIC



# Batch

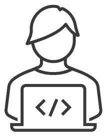
Stochastic
Mini-batch
Batch

$\theta_1$

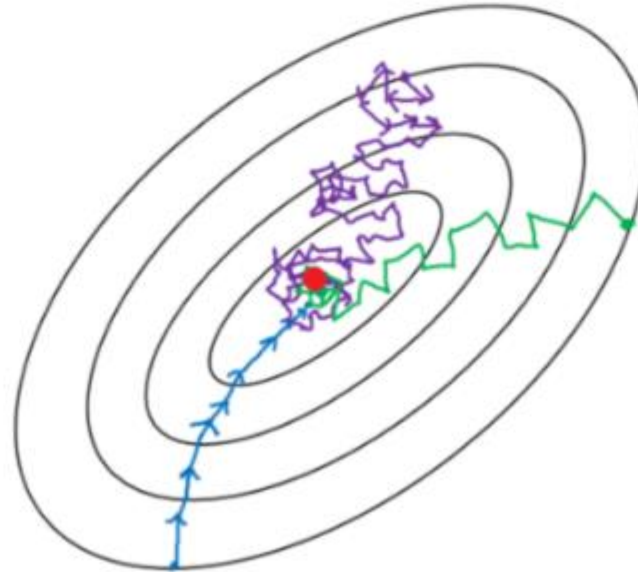
3.8
3.6
3.4
3.2
3.0
2.8
2.6
2.4

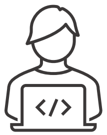
2.5 3.0 3.5 4.0 4.5

$\theta_0$



# Batch





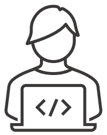
# Batch

Needed GPU memory

Desired Batch Size

Max Batch Size

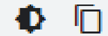
INVESTIC



# Compiler

### compile

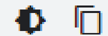
View source



```
compile(
    optimizer='rmsprop', loss=None, metrics=None, loss_weights=None,
    weighted_metrics=None, run_eagerly=None, steps_per_execution=None, **kwargs
)
```

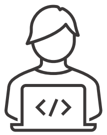
Configures the model for training.

**Example:**



```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=[tf.keras.metrics.BinaryAccuracy(),
                       tf.keras.metrics.FalseNegatives()])
```

INVESTIC



# Compiler

## Optimizer

### Classes

`class Adadelta` : Optimizer that implements the Adadelta algorithm.

`class Adagrad` : Optimizer that implements the Adagrad algorithm.

`class Adam` : Optimizer that implements the Adam algorithm.

`class Adamax` : Optimizer that implements the Adamax algorithm.

`class Ftrl` : Optimizer that implements the FTRL algorithm.

`class Nadam` : Optimizer that implements the NAdam algorithm.

`class Optimizer` : Base class for Keras optimizers.

`class RMSprop` : Optimizer that implements the RMSprop algorithm.

`class SGD` : Gradient descent (with momentum) optimizer.

## Loss

### Classes

`class BinaryCrossentropy` : Computes the cross-entropy loss between true labels and predicted labels.

`class CategoricalCrossentropy` : Computes the crossentropy loss between the labels and predictions.

`class CategoricalHinge` : Computes the categorical hinge loss between `y_true` and `y_pred`.

`class CosineSimilarity` : Computes the cosine similarity between labels and predictions.

`class Hinge` : Computes the hinge loss between `y_true` and `y_pred`.

`class Huber` : Computes the Huber loss between `y_true` and `y_pred`.

`class KLDivergence` : Computes Kullback-Leibler divergence loss between `y_true` and `y_pred`.

`class LogCosh` : Computes the logarithm of the hyperbolic cosine of the prediction error.

`class Loss` : Loss base class.

`class MeanAbsoluteError` : Computes the mean of absolute difference between labels and predictions.

`class MeanAbsolutePercentageError` : Computes the mean absolute percentage error between `y_true` and `y_pred`.

`class MeanSquaredError` : Computes the mean of squares of errors between labels and predictions.

`class MeanSquaredLogarithmicError` : Computes the mean squared logarithmic error between `y_true` and `y_pred`.

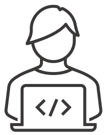
`class Poisson` : Computes the Poisson loss between `y_true` and `y_pred`.

`class Reduction` : Types of loss reduction.

`class SparseCategoricalCrossentropy` : Computes the crossentropy loss between the labels and predictions.

`class SquaredHinge` : Computes the squared hinge loss between `y_true` and `y_pred`.

INVESTIC

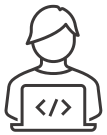


# Compiler

Rmsprop quit saddle first

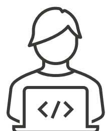
SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop

INVESTIC

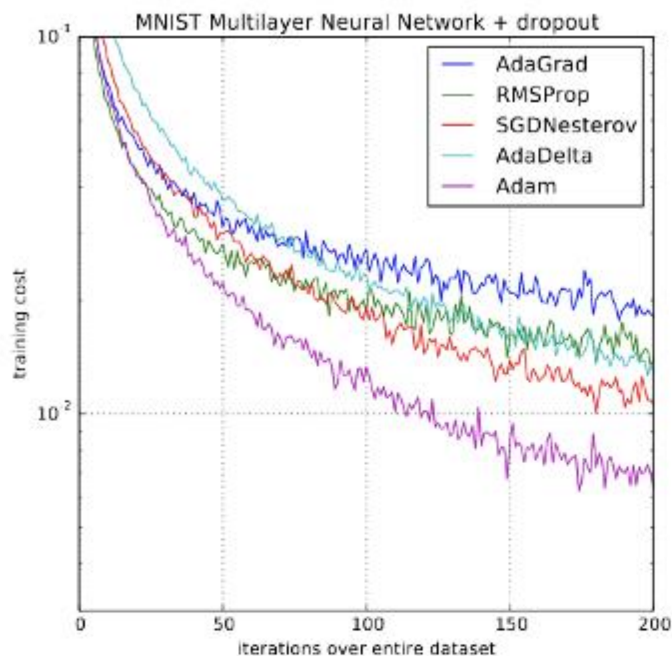


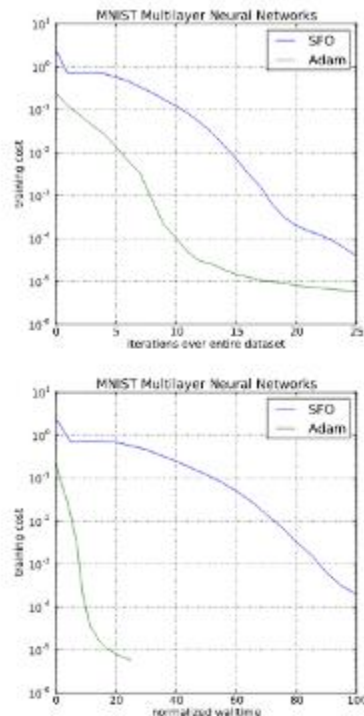
# Compiler

SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop



# Compiler

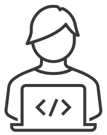



(a) (b)

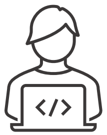
Figure 2: Training of multilayer neural networks on MNIST images. (a) Neural networks using dropout stochastic regularization. (b) Neural networks with deterministic cost function. We compare with the sum-of-functions (SFO) optimizer (Sohl-Dickstein et al., 2014)

INVESTIC

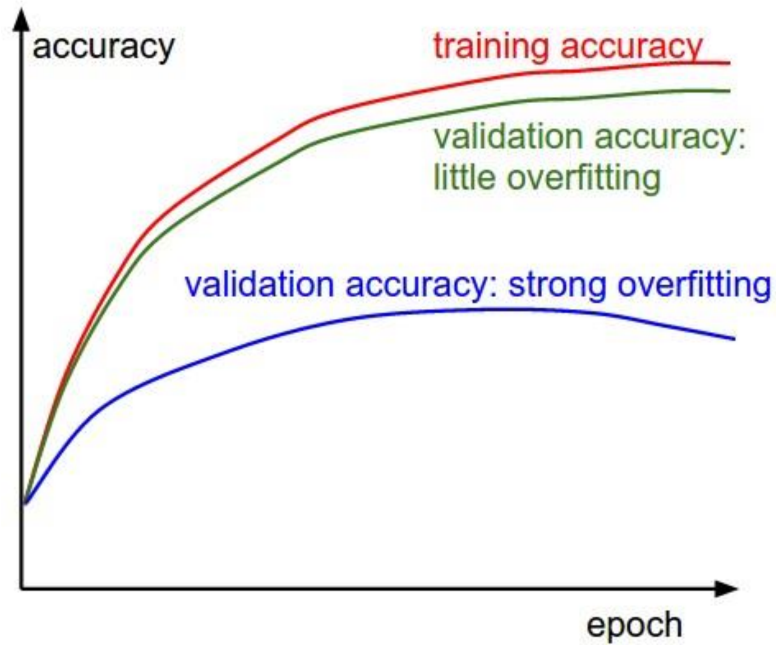


# Overfit Underfit??

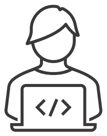
| | Under-fitting | Optimal-fitting | Over-fitting |
|---|---|---|---|
| Regression | | | |
| Classification | | | |
| Deep learning | Error<br>Validation<br>Training<br>Epochs | Error<br>Validation<br>Training<br>Epochs | Error<br>Validation<br>Training<br>Epochs |



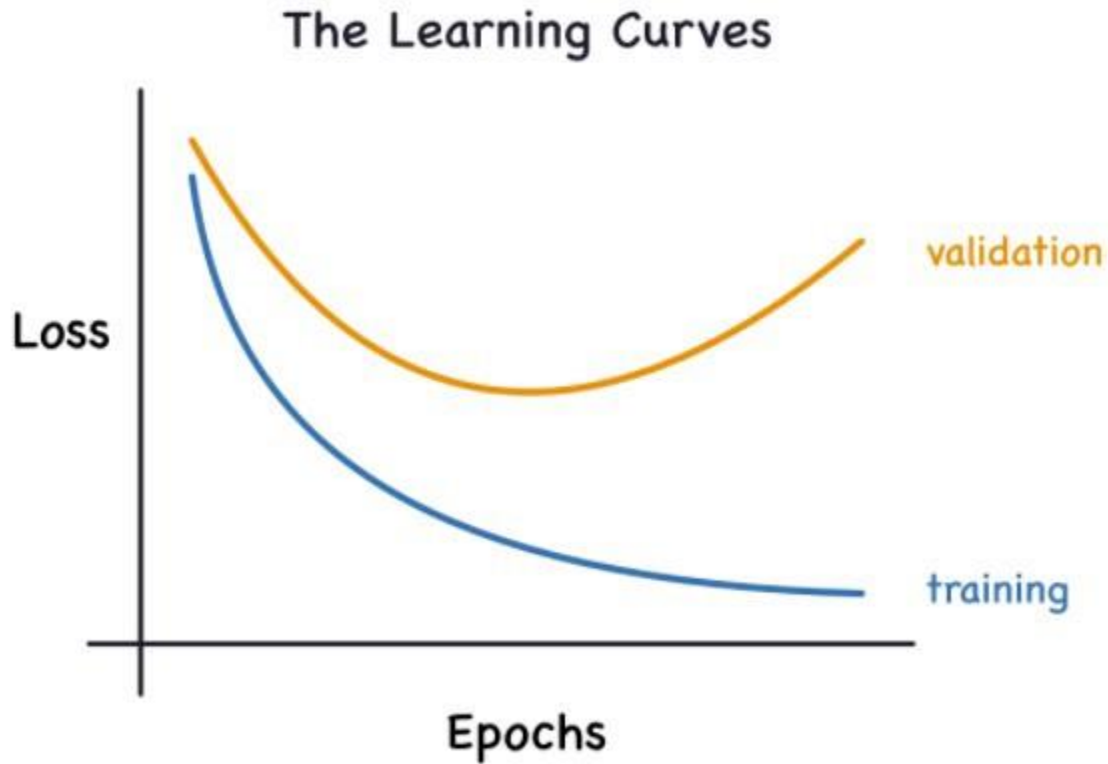
# Overfit Underfit??

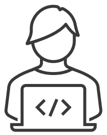



INVESTIC

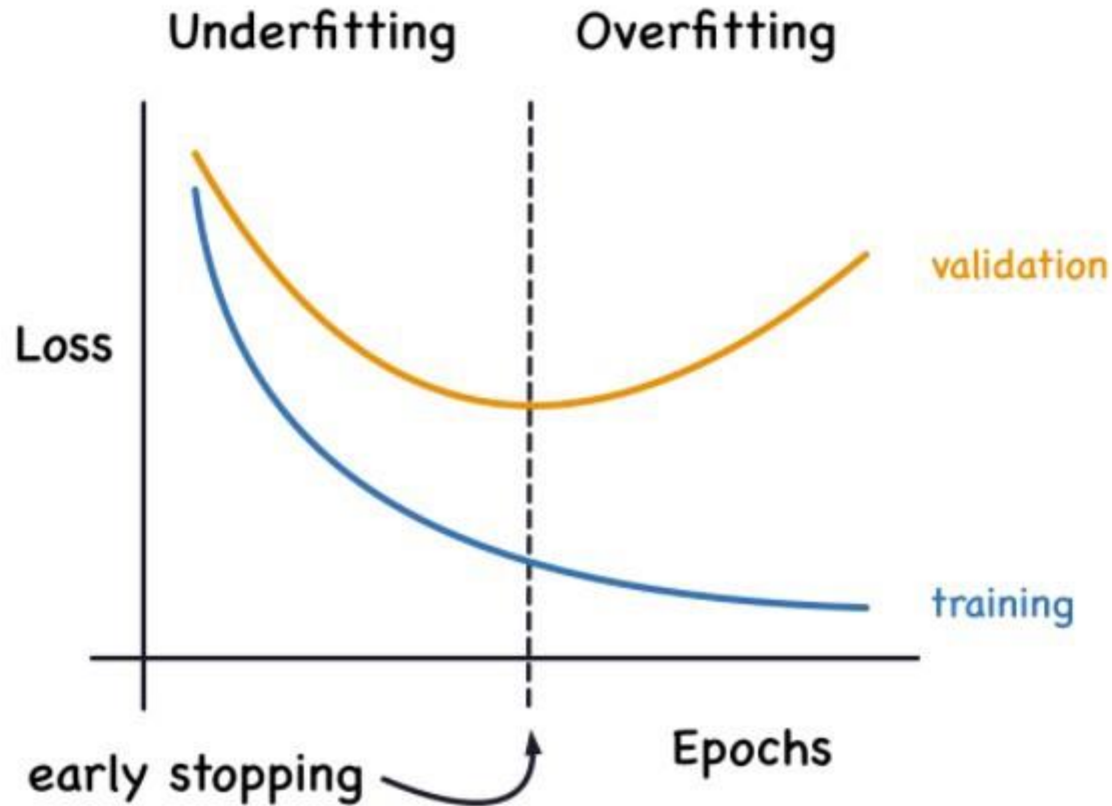


# Validating during training

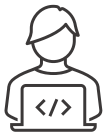




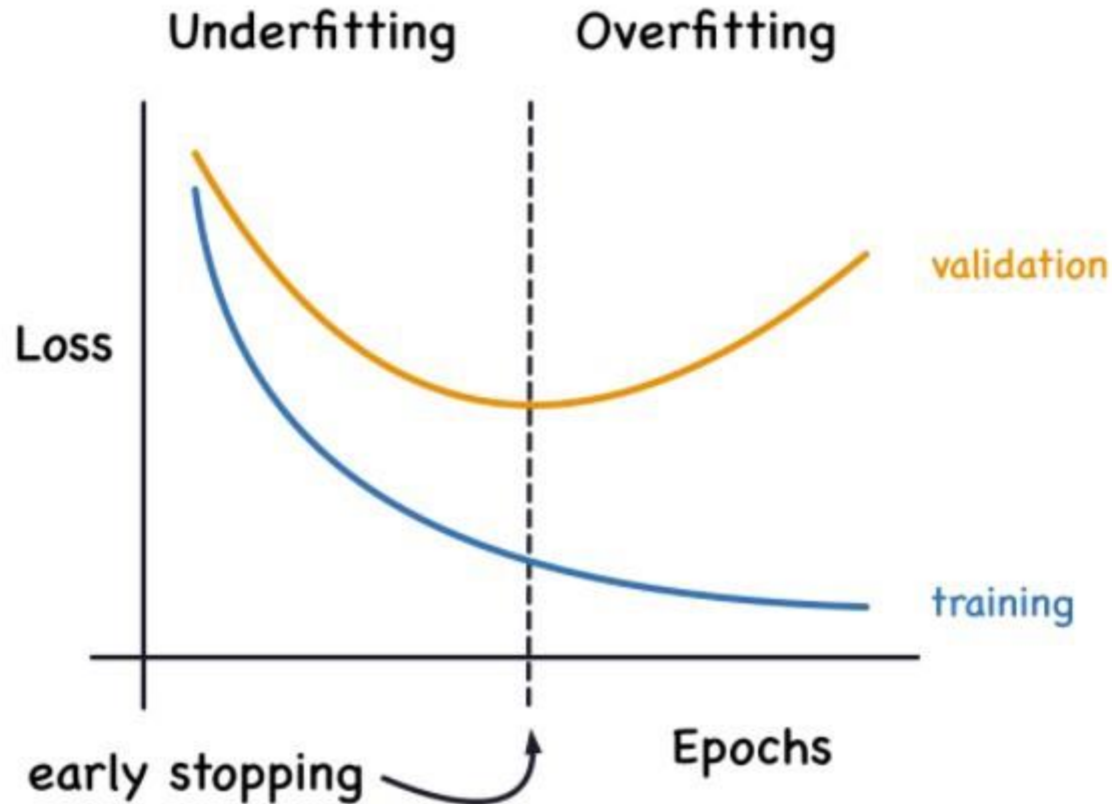
# STOP !



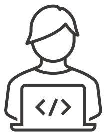

INVESTIC



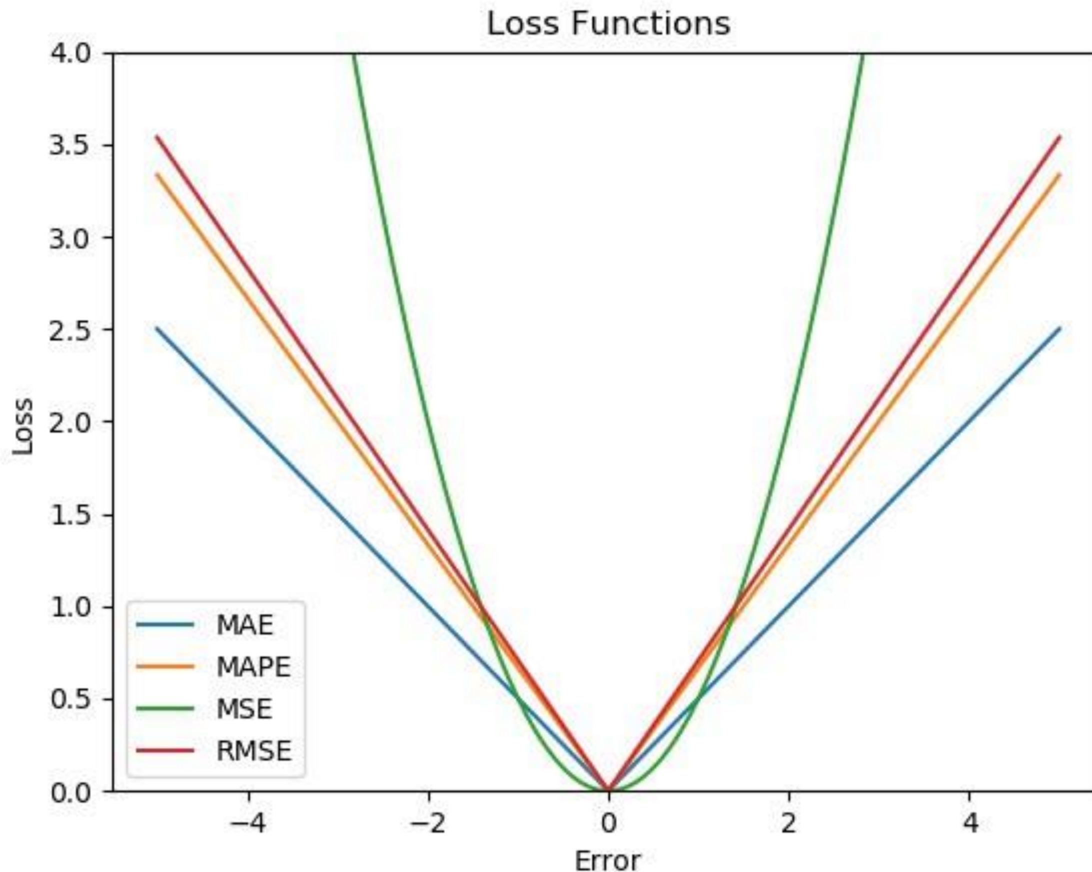
# STOP !




*We keep the model where the validation loss is at a minimum.*



# MAE vs MSE vs RMSE vs MAPE



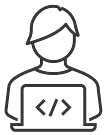

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

test set · predicted vaue · actual value

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{y_i}.100\%$$

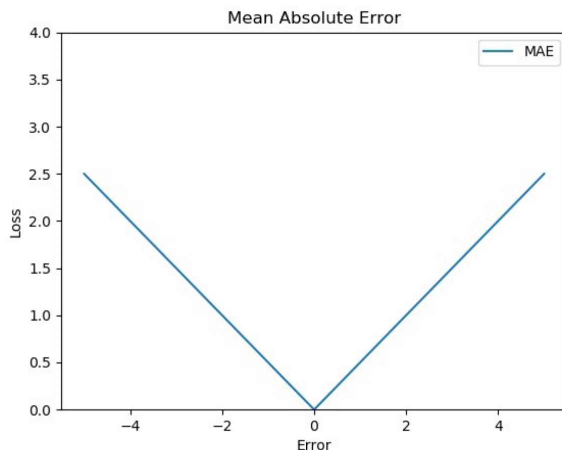


# More on Loss Function for Regression

## Mean Absolute Error (MAE)

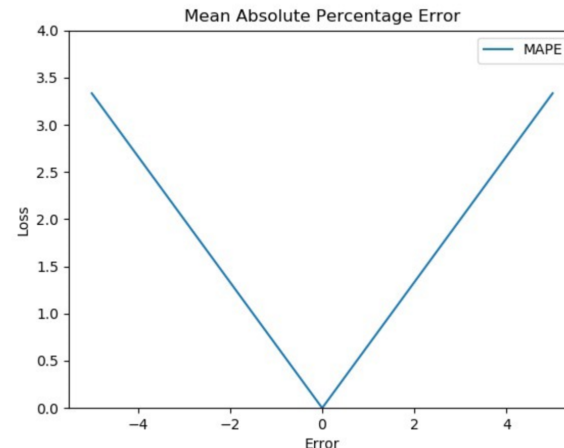
considering all the errors on the same scale

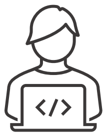
MAE is a linear scoring method, all the errors are weighted equally. This means that while backpropagation, we may just jump past the minima due to MAE's steep nature.




## Mean Absolute Percentage Error (MAPE)

MAPE is similar to that of MAE, with one key difference, that it calculates error in terms of **percentage**, instead of raw values. Due to this, MAPE is independent of the scale of our variables.



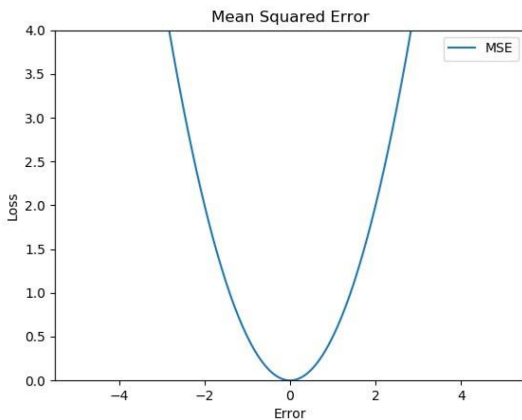


# More on Loss Function for Regression

## Mean Squared Error (MSE)

For small errors, MSE helps converge to the minima efficiently, as the gradient reduces gradually.

a **quadratic scoring** method, meaning, the penalty is proportional to not the error (like in MAE) but to the **square of the error**, which gives relatively higher weight (penalty) to large errors/outliers, while smoothening the gradient for smaller errors.

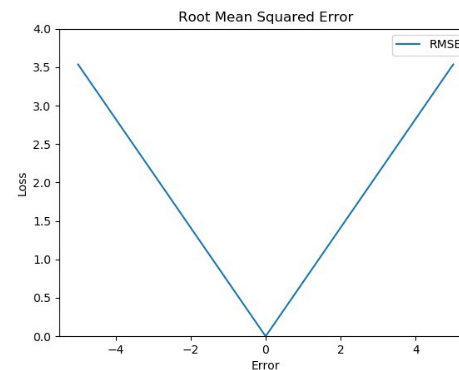


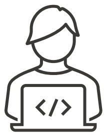

## Root Mean Squared Error (RMSE)

RMSE is just the **square root** of MSE, which means, it is again, a linear scoring method, but still better than MAE as it gives comparatively more weightage to larger errors.

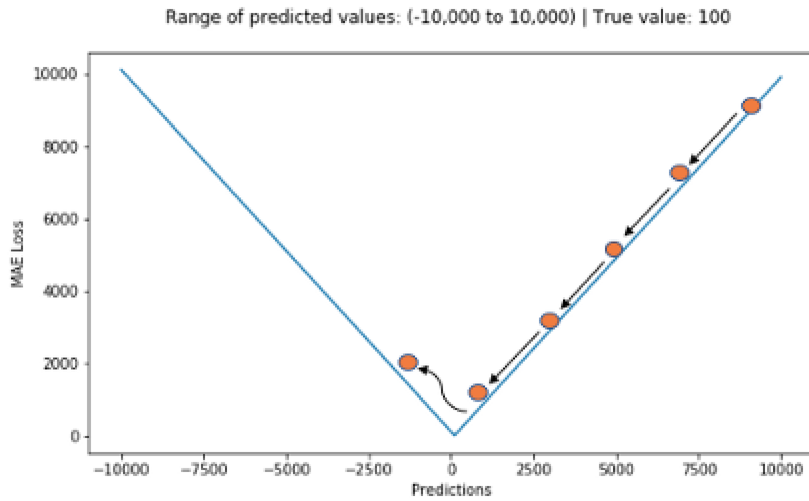
RMSE is still a linear scoring function, so again, near minima, the gradient is sudden.

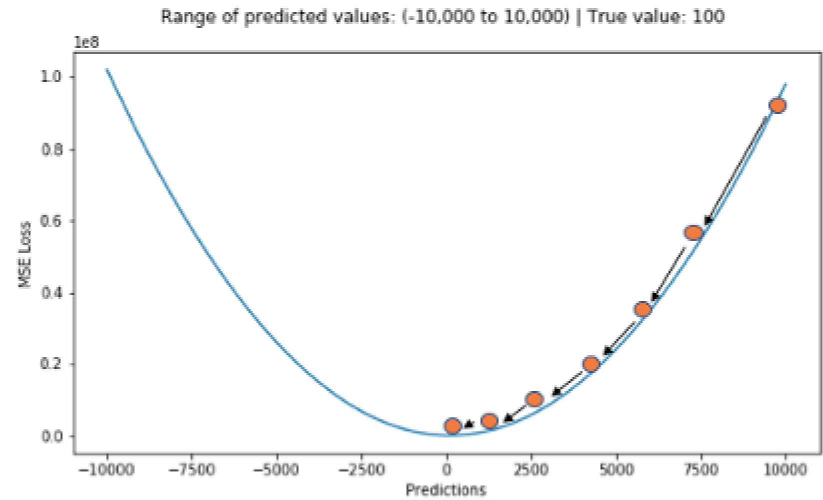
Less extreme losses even for larger values.



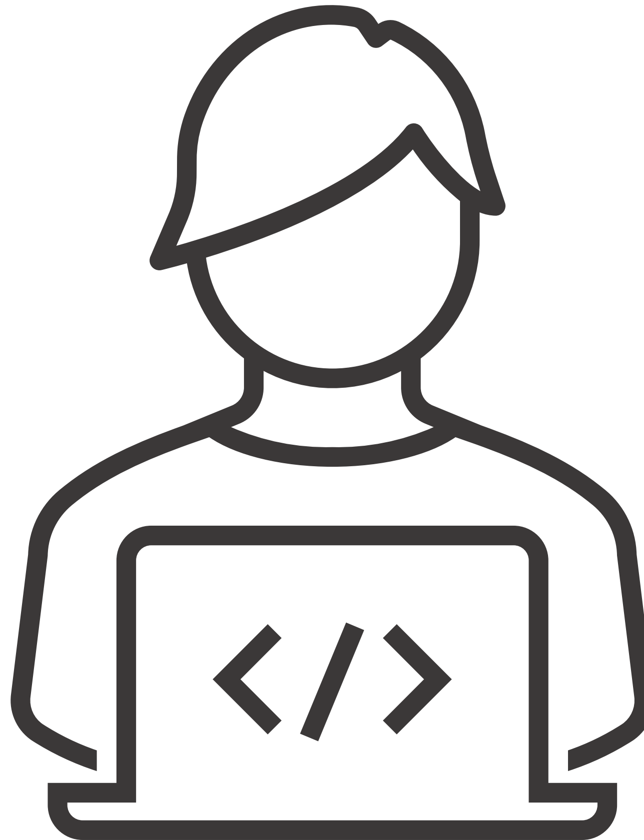


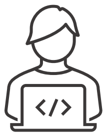
# More on Loss Function for Regression



# WORKSHOP TIME





# Workshop Time !

**Workshop I**

Pick your asset class – Stock, Crypto, Forex … Choose it yourself !!

Try regression with neural network (1 layer, no activation function)

**Workshop II**

Based on workshop I, add 2 more layers and activation function as relu

a) No activation function at output node
b) Use sigmoid as activation function at output node

Recommend: ADAM as optimizer

**Workshop III – False EMA cross over signal check with Deep Learning**

Pick 1 asset, create ema-5 to ema-20 cross over, RSI-14, MACD. Check whether
This strategy profit in next 1 month or not …

Design your own network .. Just try it

INVESTIC