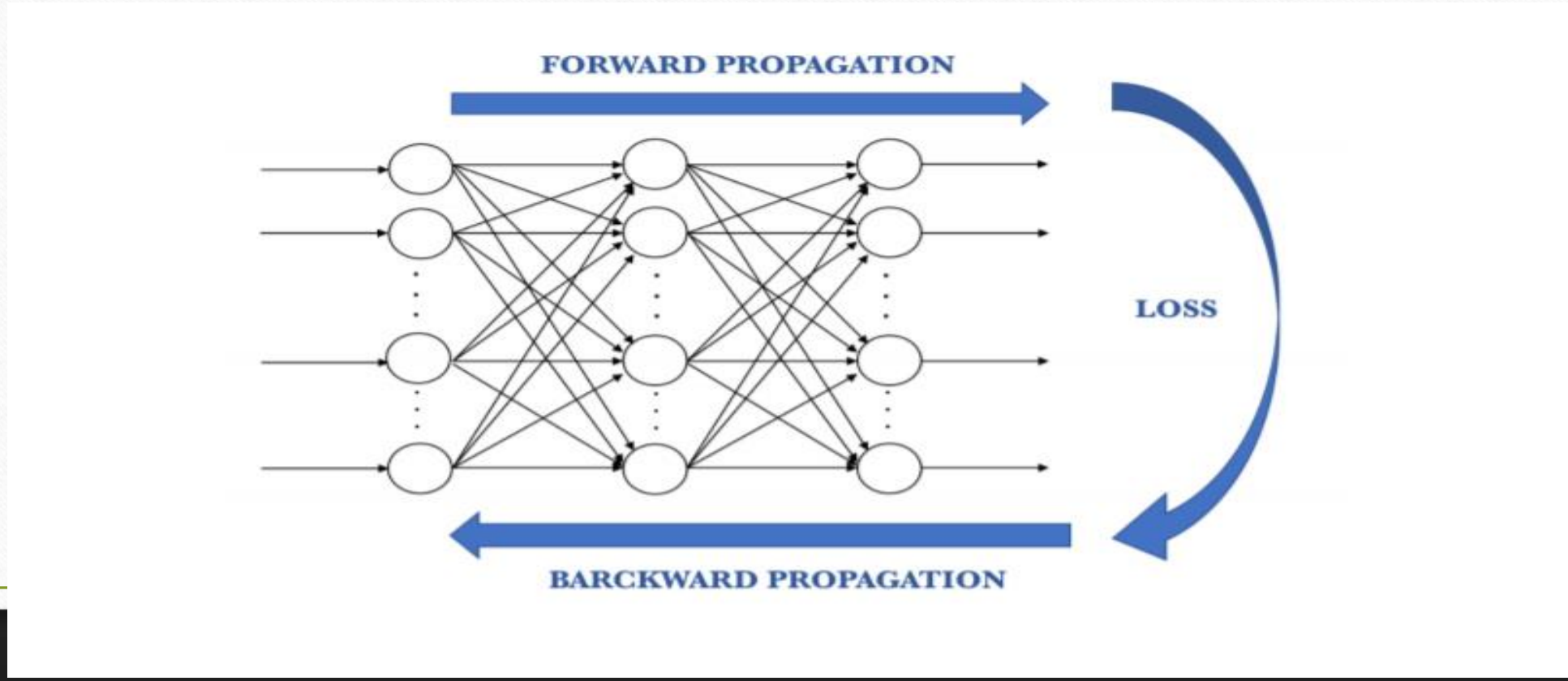
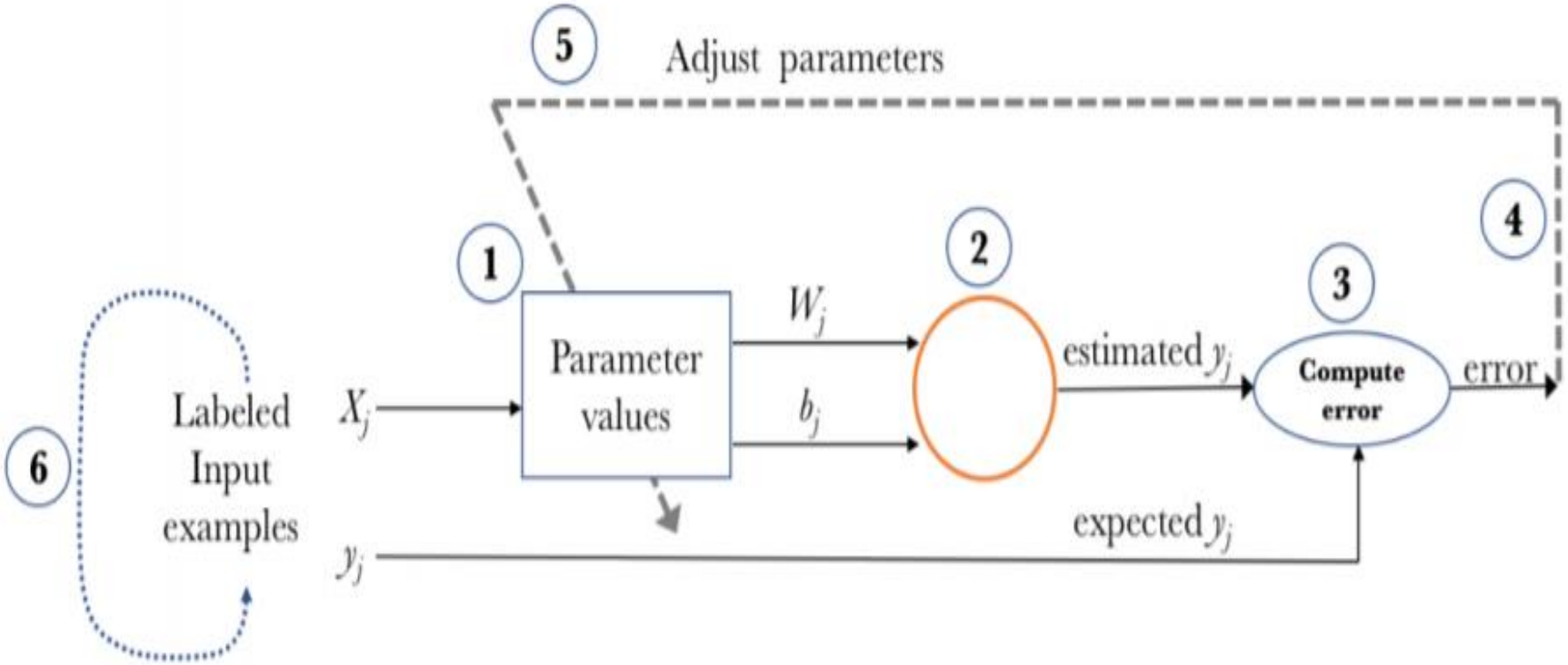
BACK

PROPAGATION

Backpropagation

Back-propagation is just a way of propagating the total loss **back** into the **neural network** to know how much of the loss every node is responsible for, and subsequently updating the weights in such a way that minimizes the loss by giving the nodes with higher error rates lower weights and vice versa.




5
Adjust parameters
1
Parameter values
W_j
b_j
2
4
3
estimated y_j
Compute error
error
6
Labeled Input examples
X_j
y_j
expected y_j

learning algorithm consists of:

1. Start with values (often random) for the network parameters (wij weights and bj biases)
2. Take a set of input data and pass them through the network to obtain their prediction.
3. Compare these predictions obtained with the values of expected labels and calculate the loss with them.
4. Perform the backpropagation in order to propagate this loss to each and every one of the parameters that make up the model of the neural network.
5. Use this propagated information to update the parameters of the neural network with the gradient descent in a way that the total loss is reduced and a better model is obtained.
6. Continue iterating in the previous steps until we consider that we have a good model.

Loss function

A loss function is one of the parameters required to quantify how close a particular neural network is to the ideal weight during the training process

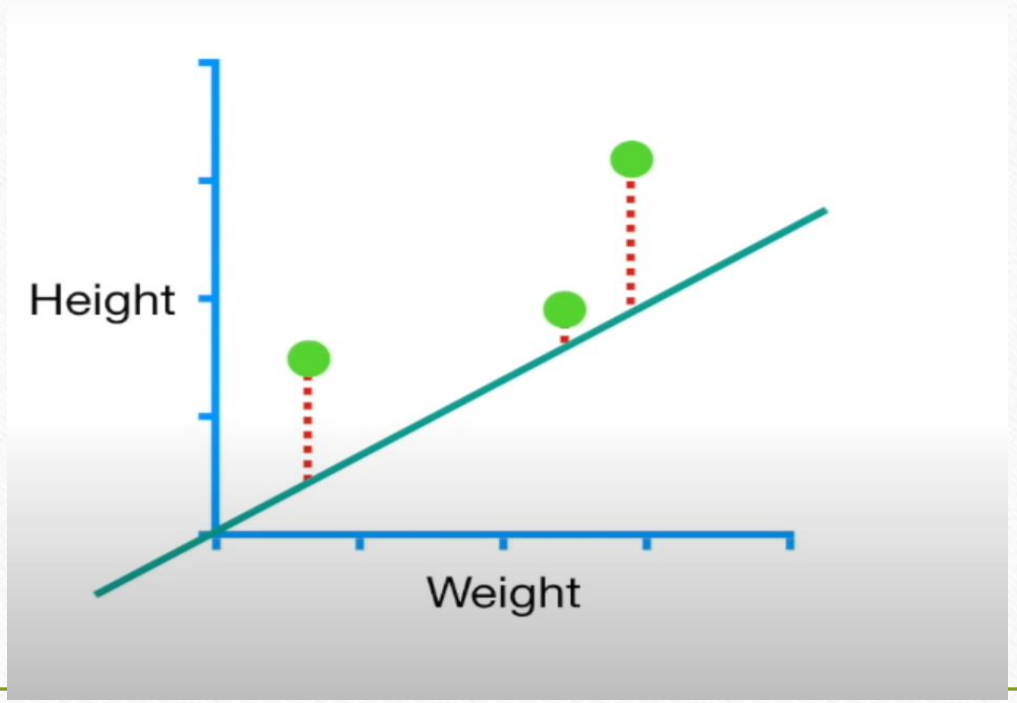
Optimizers

Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate

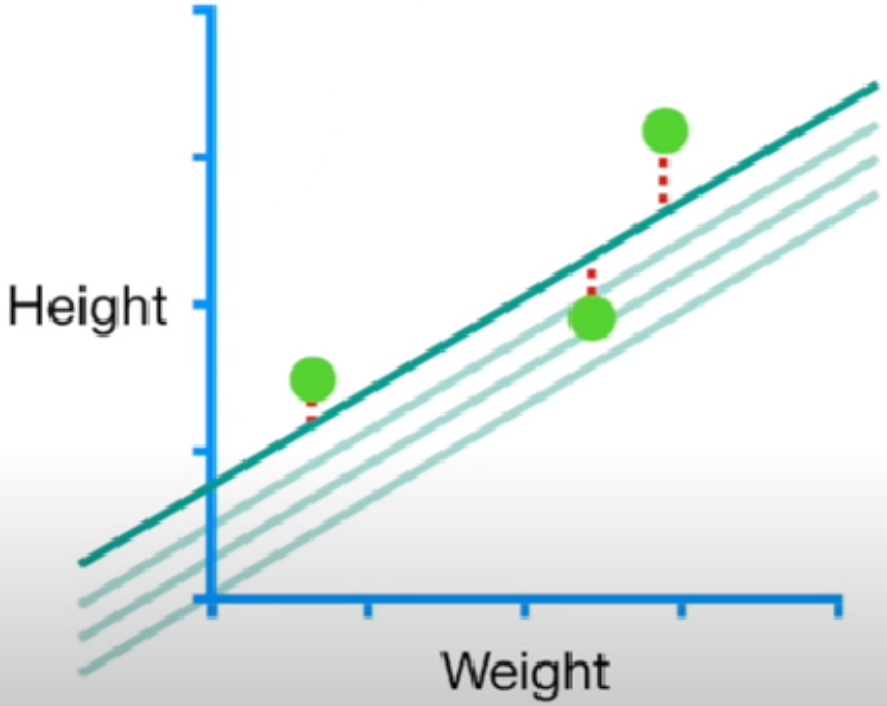
order to reduce the losses.

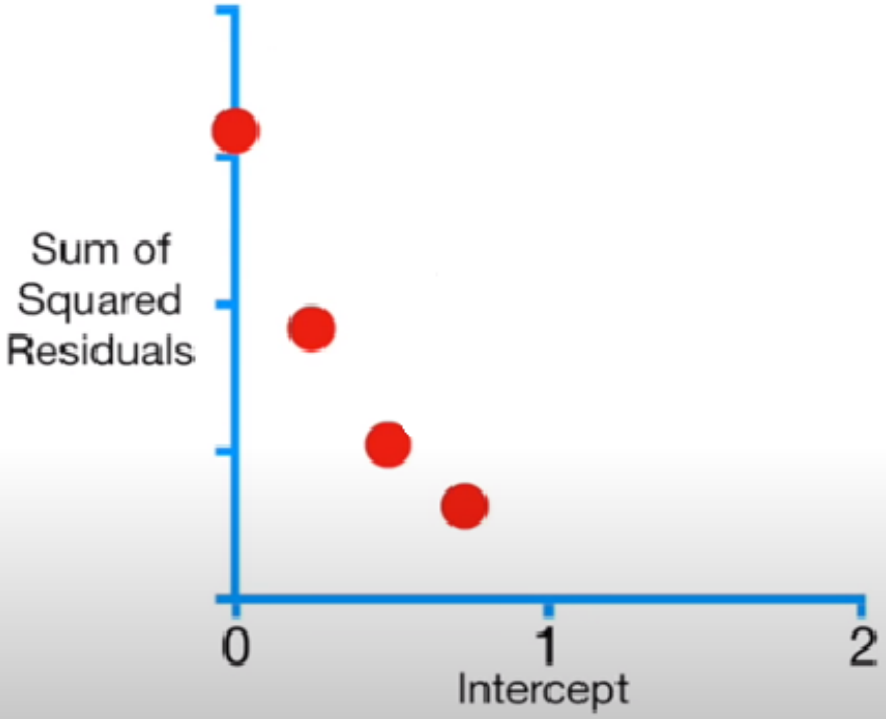
Gradient Decent

Gradient Descent is the most basic but most used optimization algorithm. It's used heavily in linear regression and classification algorithms.



PREDICTED HEIGHT=slope * weight + height





METRICS :

Metrics is used to evaluate the performance of your model. It is similar to loss function, but not used in training process. Keras provides quite a few metrics as a module, **metrics** and they are as follows

- accuracy
- binary_accuracy

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

Trains the **model** for a fixed number of epochs

Model**.**fit(x=x_train, y=y_train, epochs=5, batch_size=100)

- **x**: Input data.
- **y**: Target data.
- **epochs** tells us the number of times all the training data have passed through the neural network in the training process.
- **Batch size**
- we can partition the training data in mini batches to pass them through the network. In Keras, the batch_size is the argument that indicates the size of these batches that will be used in the fit() method in an iteration of the training to update the gradient.

EVALUATE:

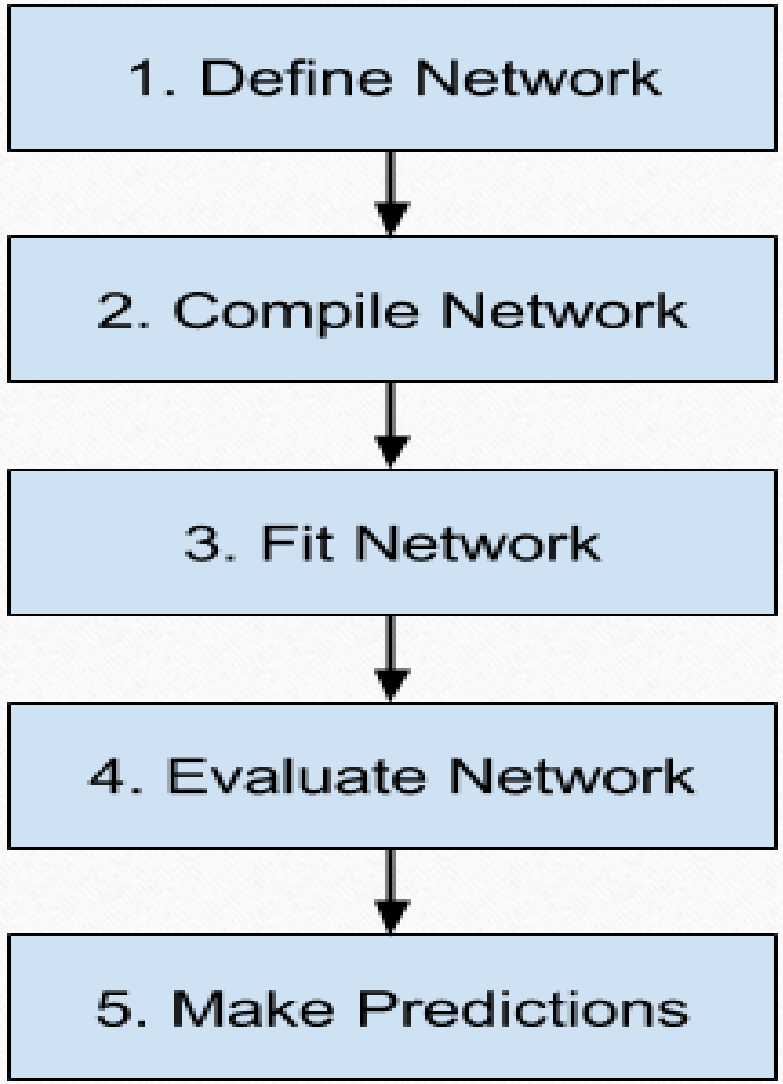
Returns the loss value & metrics values for test set.

model.evaluate(x_test , y_test)

PREDICT:

Returns the predicted value for the input given.

Model.predict(x_test)



LOAD DATA

```
import tensorflow as tf
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

SPLITTING DATA:

Further the data is split into training and testing set based on size of the dataset .