មហាវិទ្យាល័យវិស្វកម្ម
FACULTY OF ENGINEERING

Introduction to Deep Learning Applications and Theory

Lecture 5
ANN Implementation in Python

Chhoeum Vantha, Ph.D.
Telecom & Electronic Engineering

Previous week: ANN's Theory

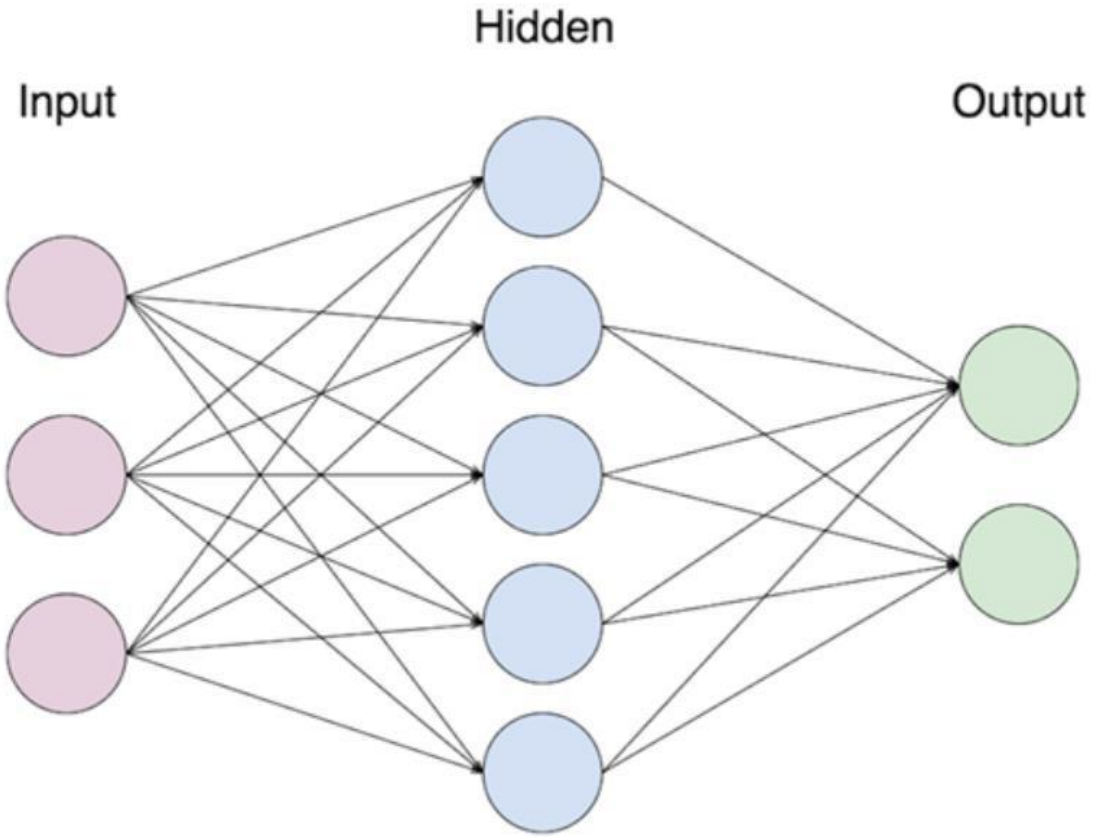
- Introduction to artificial neural network (ANN)
- The Perceptron

Outline

- Layers: Python Operators
- Optimizer
- Summarize Model
- Training Loss
- Validation Loss
- Designing Training Procedures

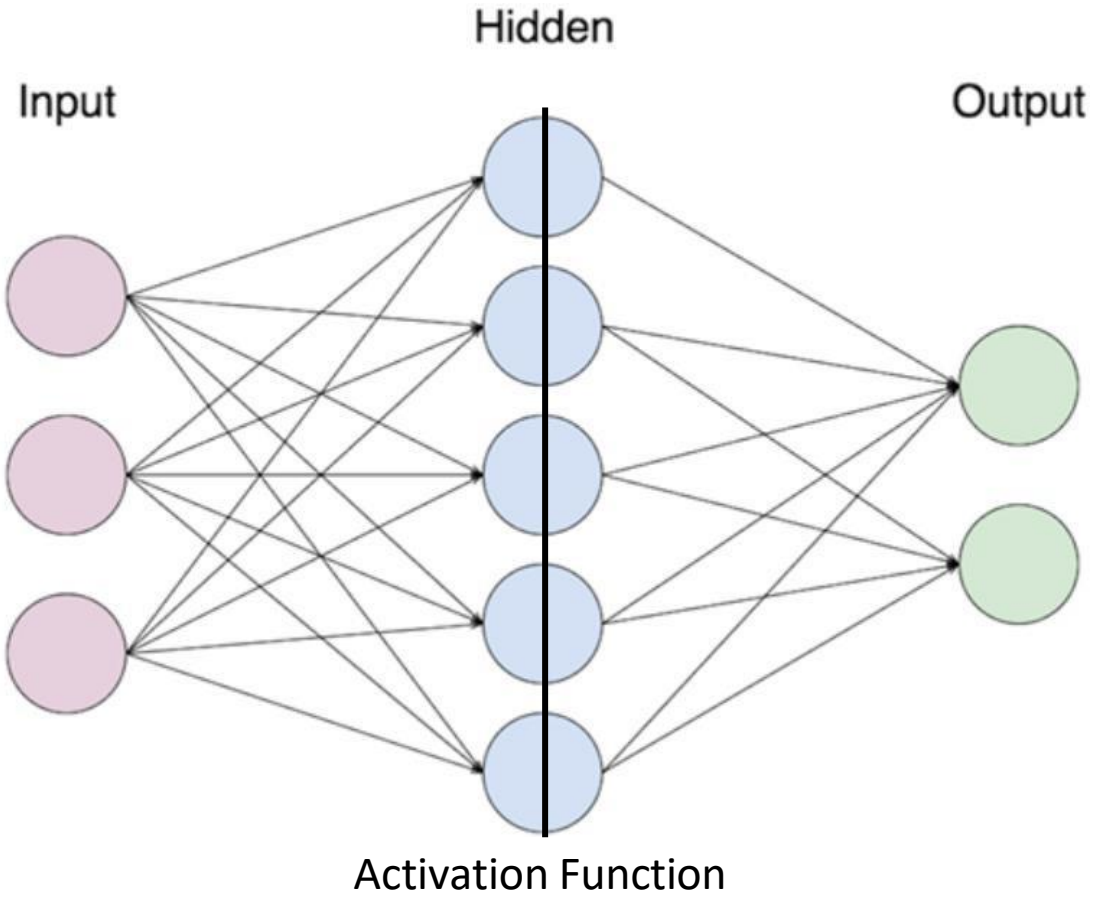
Layers: Python Operators

- Activation Functions
- Normalization
- Dropout
- Loss Functions



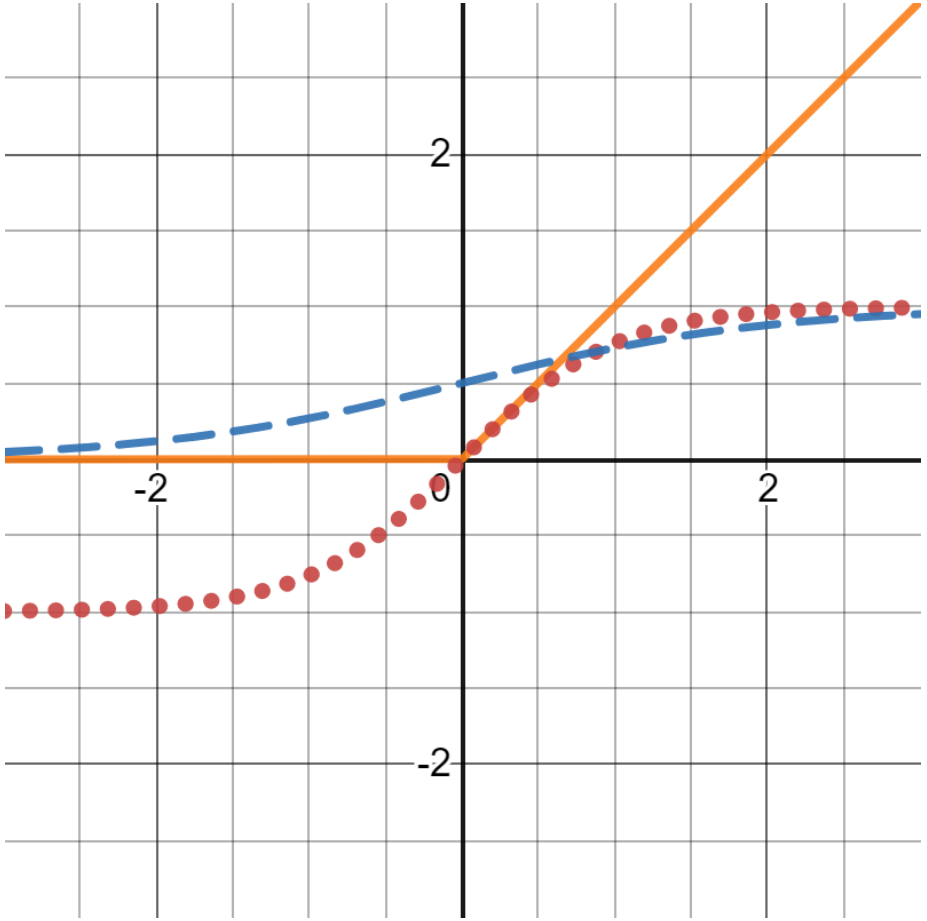
Layers: Python Operators

- Activation Functions
- Normalization
- Dropout
- Loss Functions



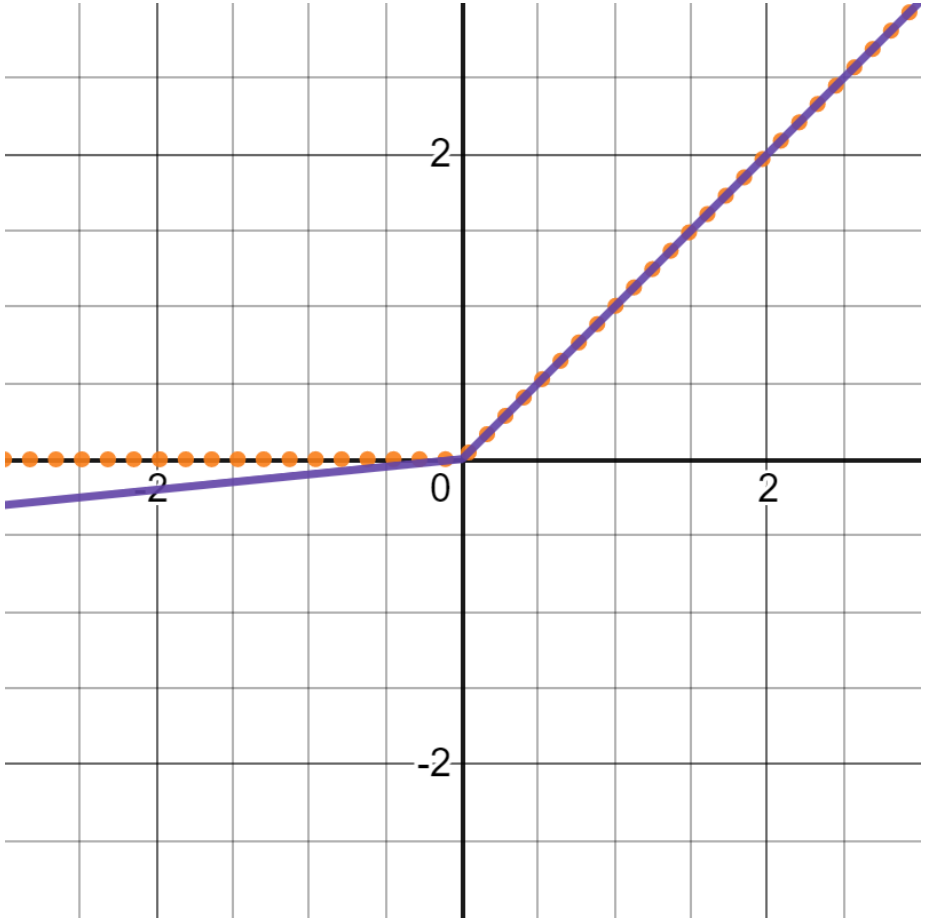
Activation Functions

- Non-linear functions performed by neurons
- ReLU - Rectified Linear Unit (nn.ReLU)
  - $y \geq 0$
- Tanh (nn.tanh)
  - $-1<y<1$
  - nn.Tanh
- Sigmoid (nn.Sigmoid)
  - $0<y<1$



Activation Functions

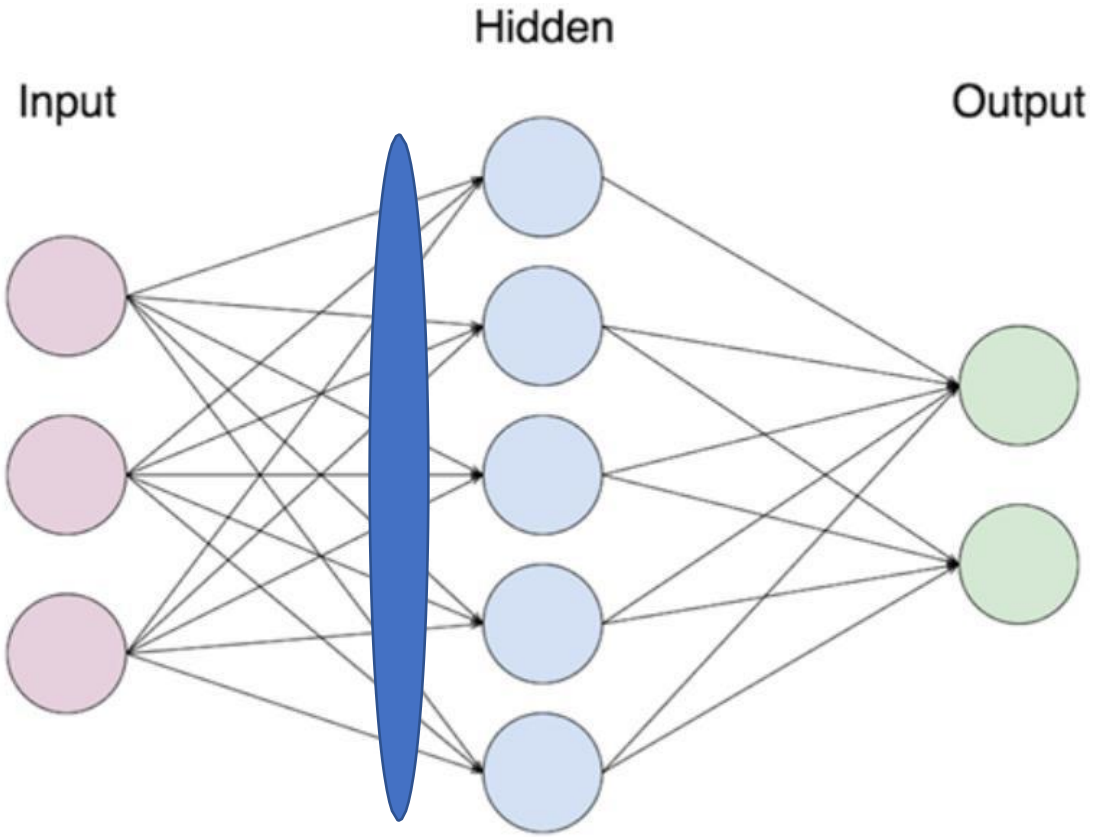
- Leaky ReLU
  - Similar to ReLU, but has non-zero values for negative x
  - Takes argument *negative_slope*, which determines the slope for x<0.
- For full list of activation functions, see: https://pytorch.org/docs/stable/nn.html



Leaky ReLU with negative slope = 0.1

Layers: Python Operators

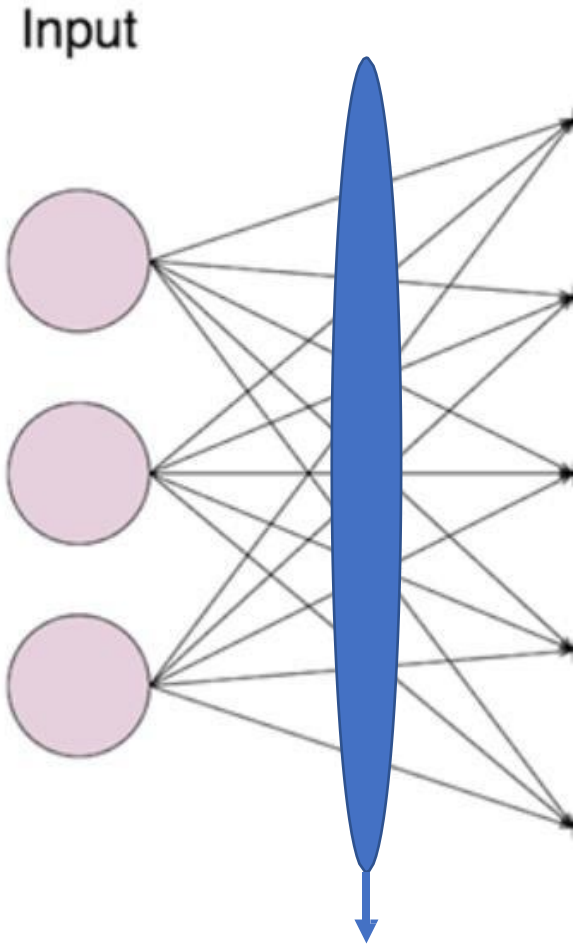
- Activation Functions
- Normalization
- Dropout
- Loss Functions



Normalization

Input Normalization: Batch Normalization

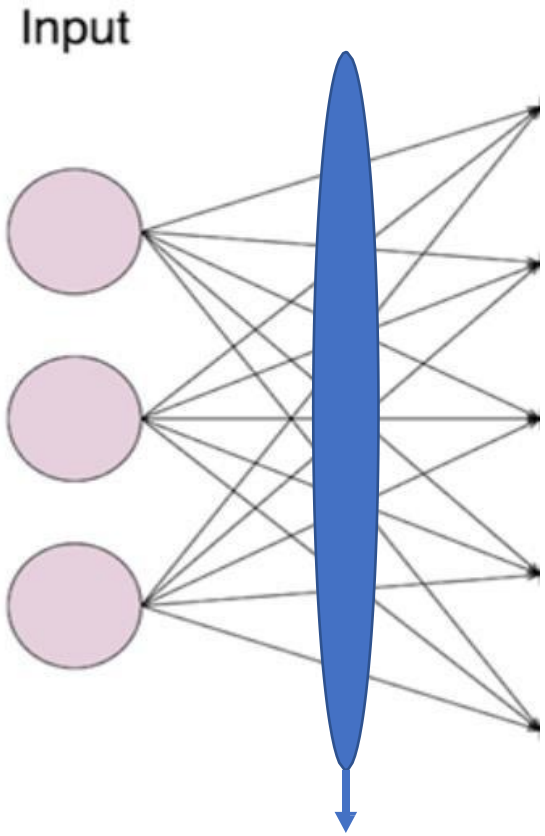
- Normalizes input into each layer for each training mini-batch
- Addresses issue of shifting input distributions over training
- Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1



Input Normalization: Batch Normalization

- Batch Normalization

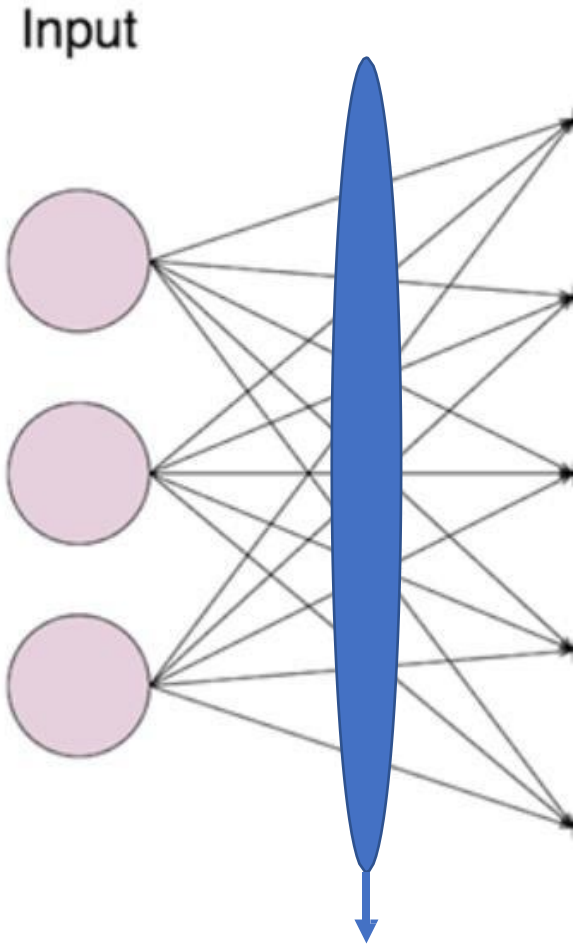
```
tf.keras.layers.BatchNormalization(
    axis=-1,
    momentum=0.99,
    epsilon=0.001,
    center=True,
    scale=True,
    beta_initializer=&#x27;zeros',
    gamma_initializer=&#x27;ones',
    moving_mean_initializer=&#x27;zeros',
    moving_variance_initializer=&#x27;ones'
    beta_regularizer=None,
    gamma_regularizer=None,
    beta_constraint=None,
    gamma_constraint=None,
    synchronized=False,
    **kwargs
)
```



Input Normalization: Batch Normalization

- Inputs:
  - num_features: Number of features in the input vector
  - eps: numerical stability parameter

```
from keras.models import Sequential
from keras.layers import Dense, BatchNormalization

model = Sequential()
model.add(Dense(32, input_shape=(64,)))
#This is the Batch normalization layer
model.add(BatchNormalization())
model.add(Dense(32))
model.add(BatchNormalization())
model.add(Dense(10, activation='softmax'))
```



Input Normalization

Other normalization procedures include:

- Layer Norm: Transposes Batch Norm.
- Normalizes over all summed inputs to a layer
  - https://arxiv.org/abs/1607.06450

- Group Norm: Normalizes by grouped channels instead of batches
  - https://arxiv.org/abs/1803.08494

Layers: Python Operators

- Activation Functions
- Normalization
- Dropout
- Loss Functions

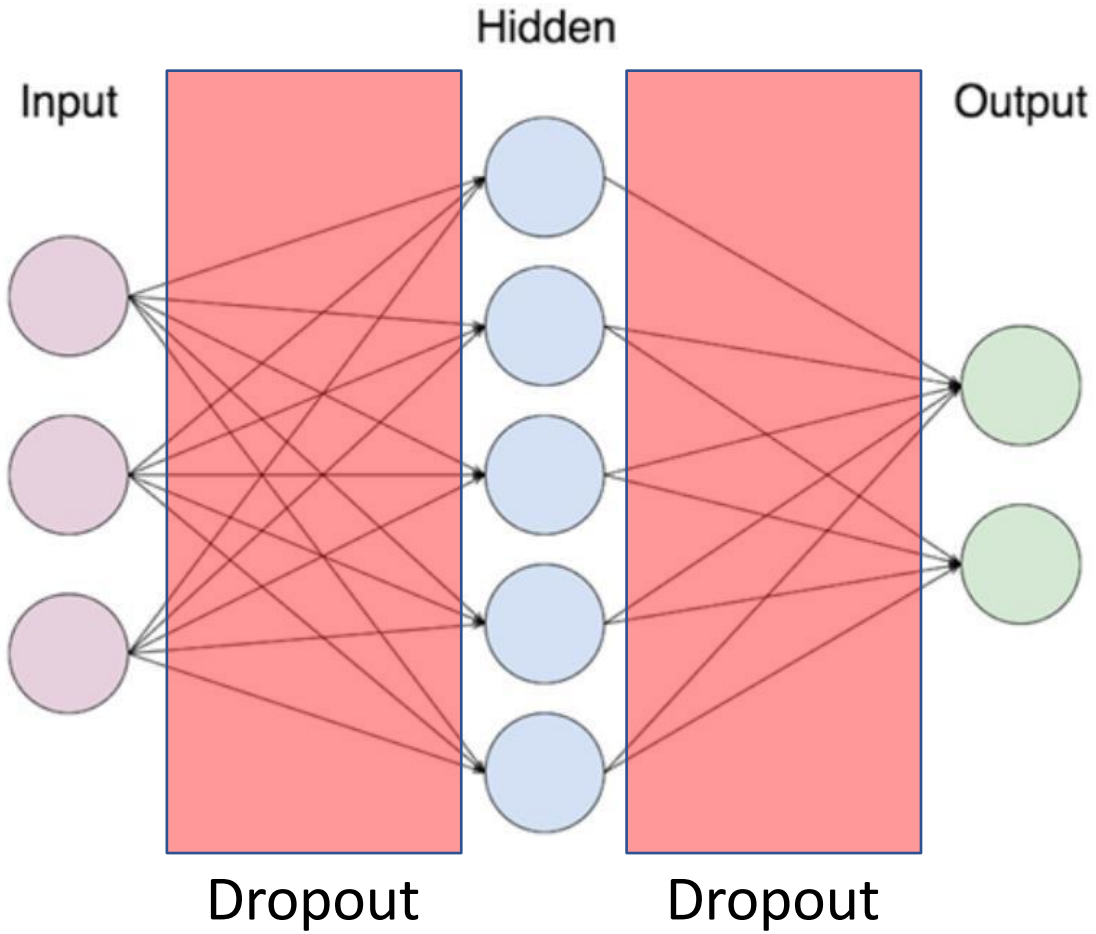
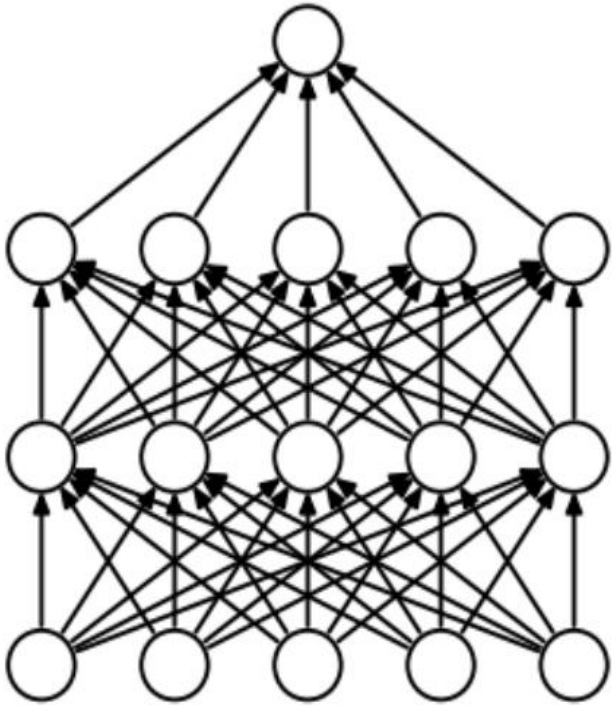


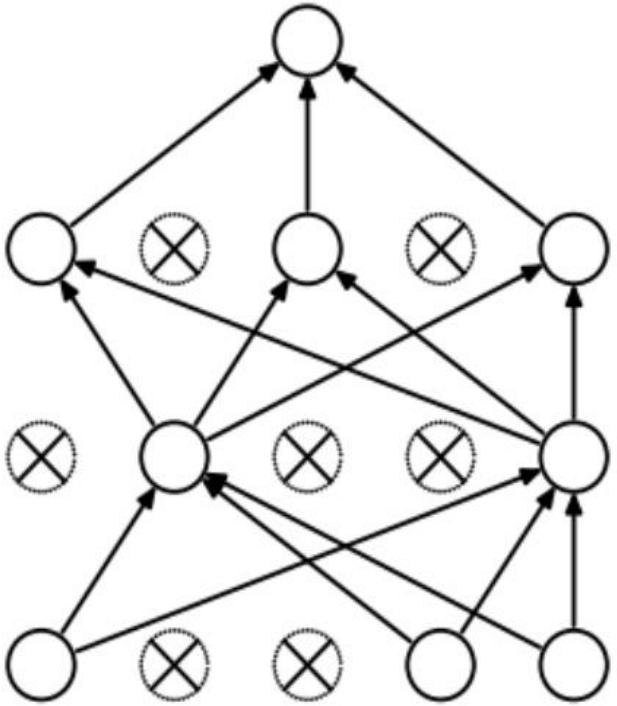
Image source: https://towardsdatascience.com/step-by-step-building-your-own-neural-network-from-scratch-df64b1c5ab

Dropout

- The term “dropout” refers to dropping out the nodes (input and hidden layer) in a neural network.
- All the forward and backwards connections with a dropped node are temporarily removed, thus creating a new network architecture out of the parent network.
- The nodes are dropped by a dropout probability of p.



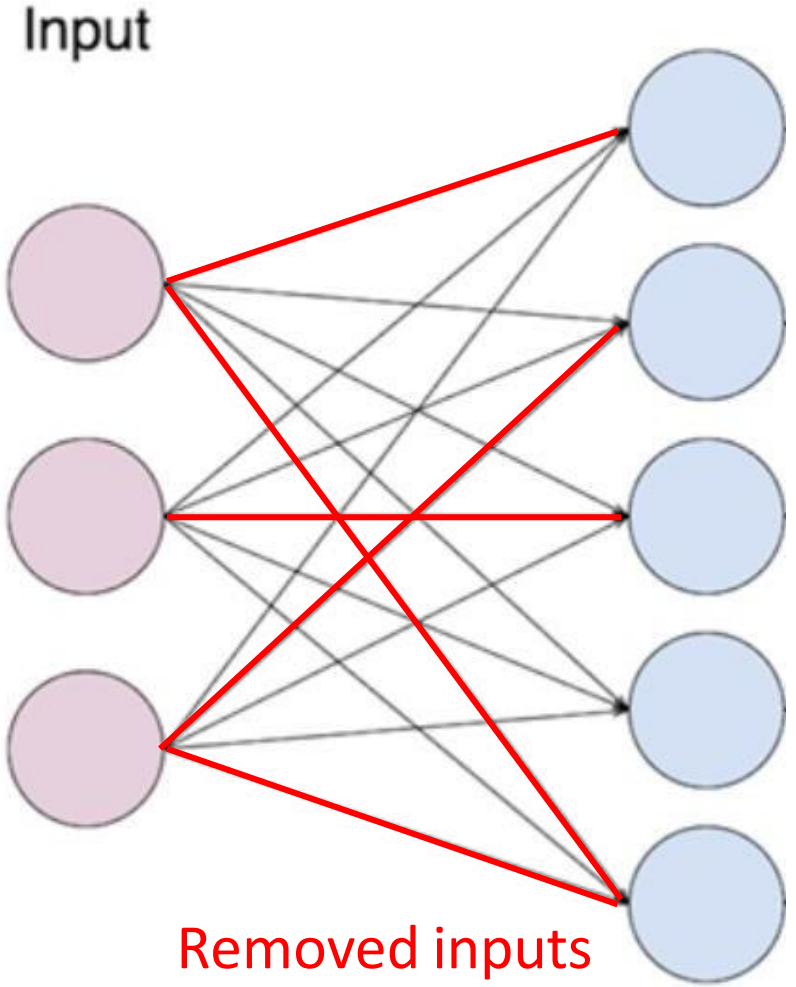
(a) Standard Neural Net



(b) After applying dropout.

Dropout

- Randomly zeroes some elements of input tensor with probability *p*
- Effective technique for regularization Outputs scaled by 1/1-p
- Treated as identity during evaluation



Removed inputs
for p = 0.33

Layers: Python Operators

- Activation Functions
- Normalization
- Dropout
- Loss Functions

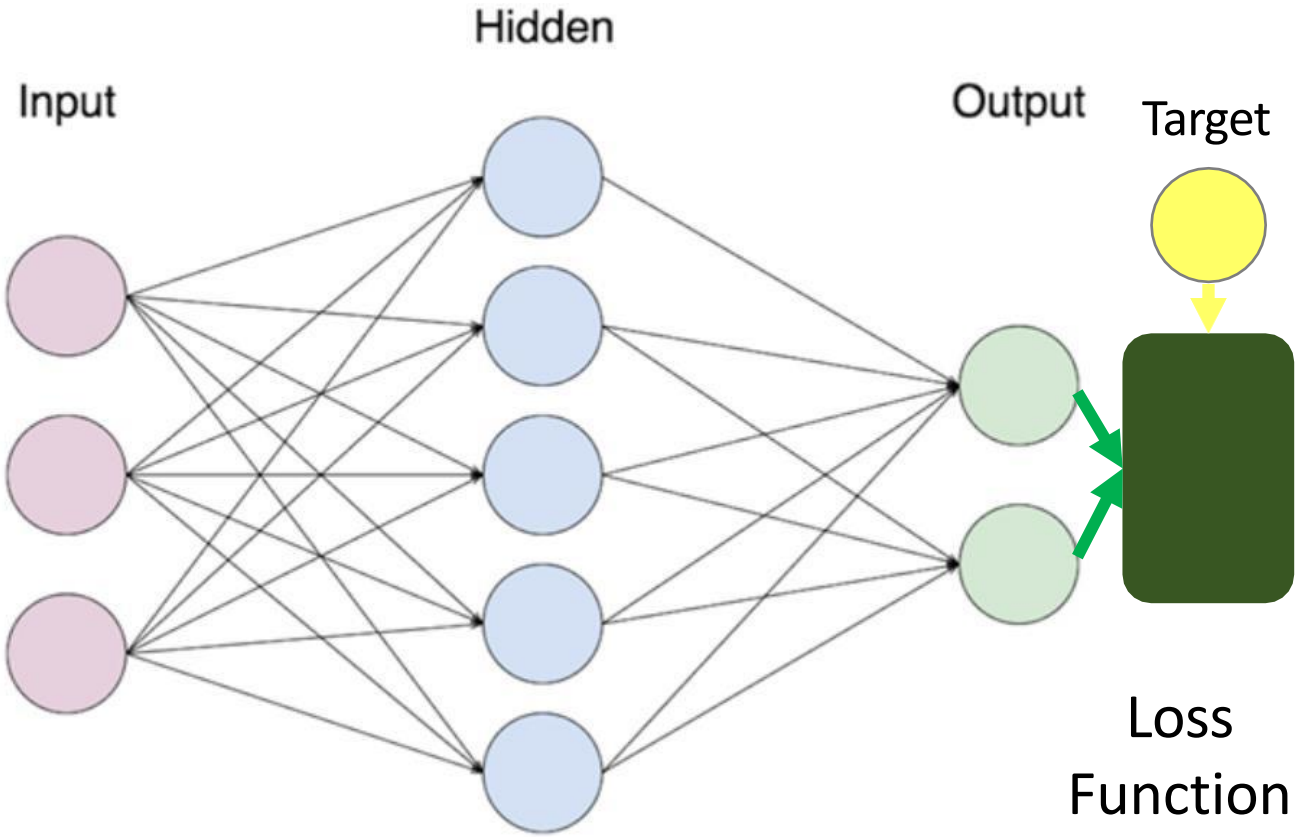
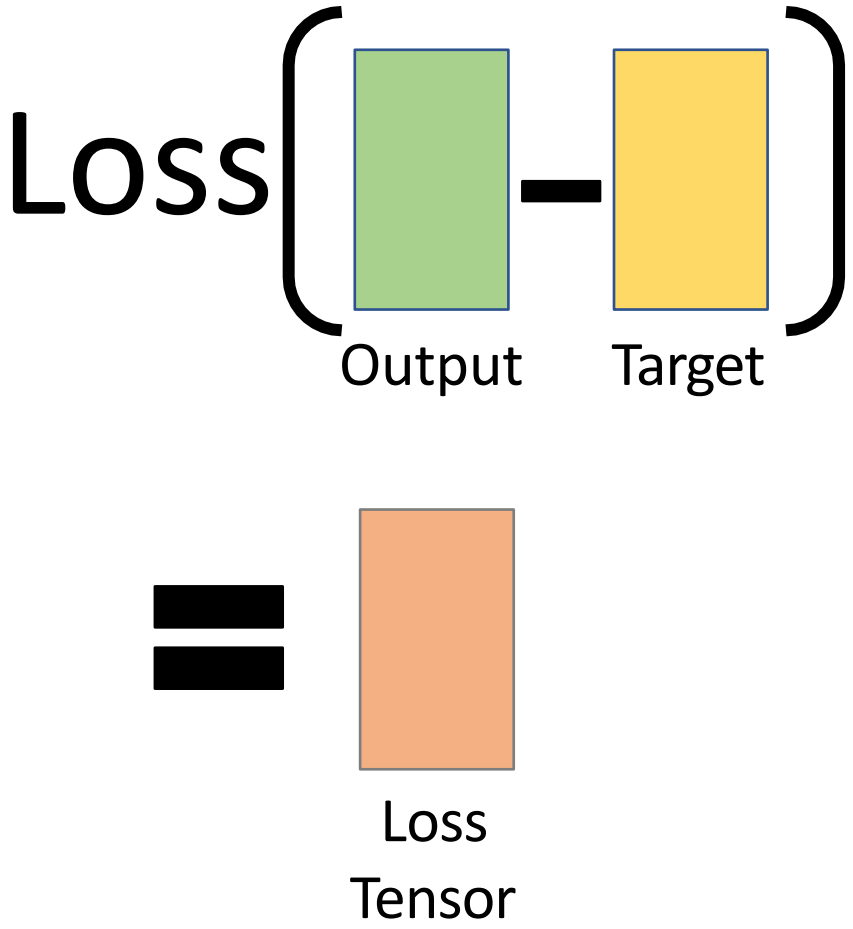


Image source: https://towardsdatascience.com/step-by-step-guide-to-building-your-own-neural-network-from-scratch-df64b1c5ab6e

Loss Functions

- Loss function parameters:
  - Reduction: how the output will be reduced in dimension:
    - None: Gives entire Loss Tensor with no reduction over batches
    - Sum: Takes sum of the loss tensor across batches, returning a single number
    - Mean: Same as sum, but divides by the number of batches to get the mean



Loss Functions: keras

Probabilistic losses

- BinaryCrossentropy class
- CategoricalCrossentropy class
- SparseCategoricalCrossentropy class
- Poisson class
- binary_crossentropy function
- categorical_crossentropy function
- sparse_categorical_crossentropy functio
- poisson function
- KLDivergence class
- kl_divergence function

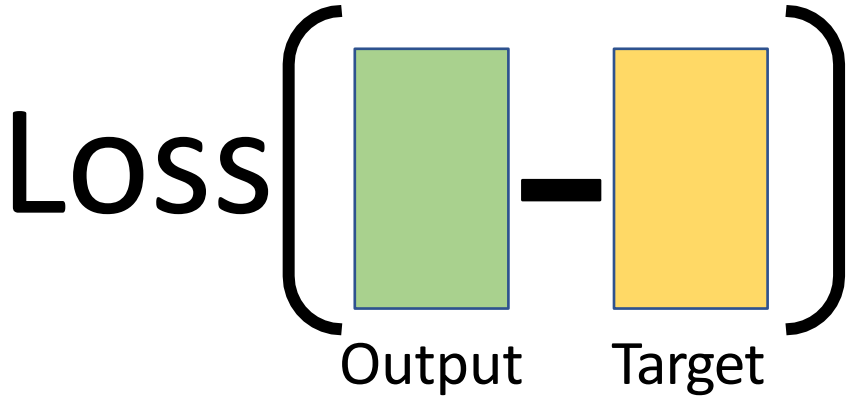
Regression losses

- MeanSquaredError class
- MeanAbsoluteError class
- MeanAbsolutePercentageError class
- MeanSquaredLogarithmicError class
- CosineSimilarity class
- mean_squared_error function
- mean_absolute_error function
- mean_absolute_percentage_error function
- mean_squared_logarithmic_error function
- cosine_similarity function
- Huber class
- huber function
- LogCosh class
- log_cosh function

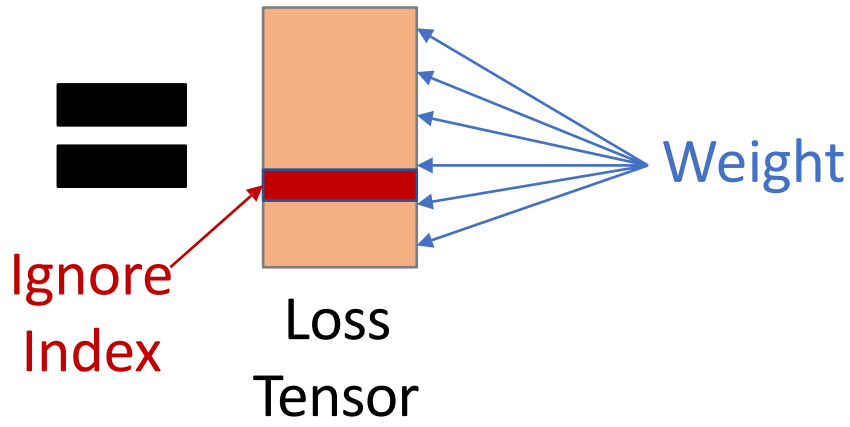
https://keras.io/api/losses/

Loss Functions – Cross Entropy Loss

- Cross Entropy Parameters
  - Weight
    - 1D tensor assigning weights to each class, which is helpful if you have an unbalanced training set
  - ignore_index
    - Specifies a target value that is ignored and does not contribute to the input gradient




```
keras_core.losses.BinaryCrossentropy(
    from_logits=False,
    label_smoothing=0.0,
    axis=-1,
    reduction="sum_over_batch_size",
    name="binary_crossentropy",
)
```

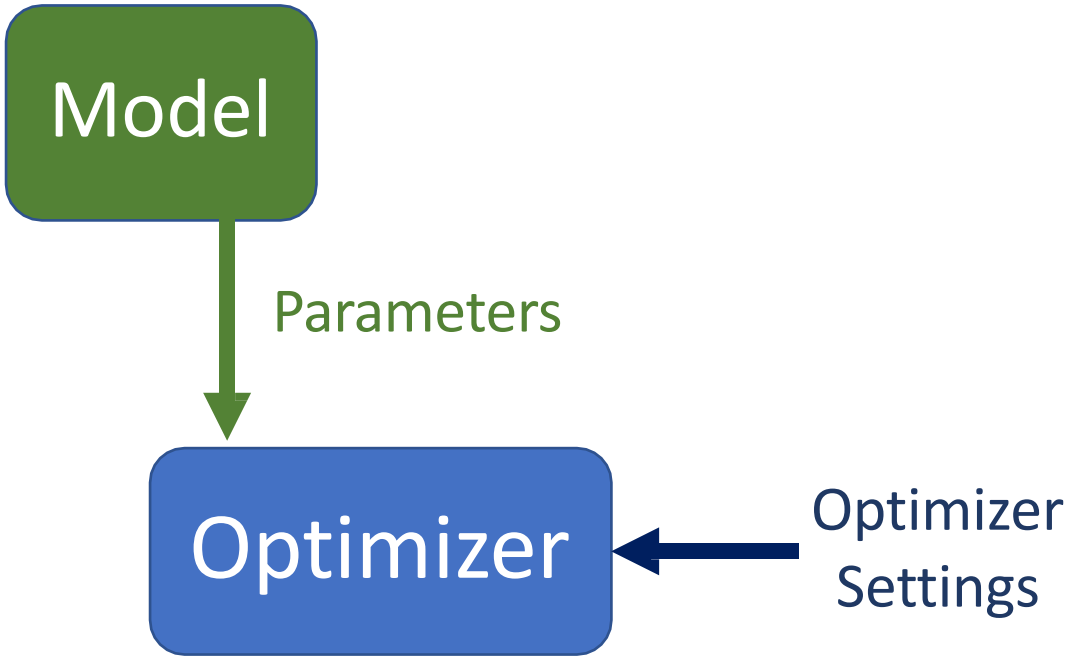


Optimizer

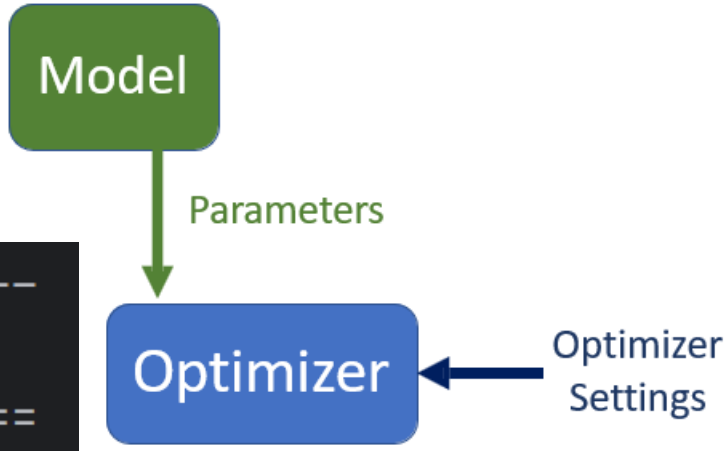
- Optimizers are algorithms or methods used to minimize an error function(loss function)or to maximize the efficiency of production.
- Optimizers are mathematical functions which are dependent on model's learnable parameters i.e Weights & Biases.
- Optimizers help to know how to change weights and learning rate of neural network to reduce the losses.

Optimizer: Initialization

- Parameters:
  - Should be iterable containing parameters to optimize
  - E.g., model.parameters() or [var1, var2]
  - Parameters must be defined BEFORE the optimizer
- Optimizer Settings
  - Learning rate, weight decay, etc.



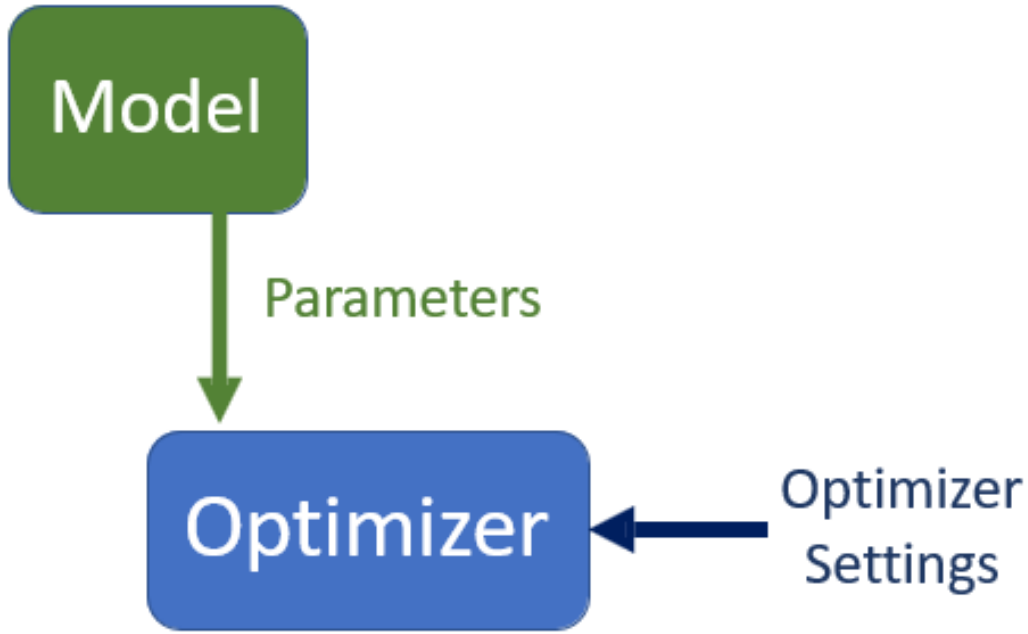
Optimizer: Initialization-Param




```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 250)               25250

 dense_1 (Dense)             (None, 150)               37650

 dense_2 (Dense)             (None, 250)               37750

 dense_3 (Dense)             (None, 100)               25100

=================================================================
Total params: 125750 (491.21 KB)
Trainable params: 125750 (491.21 KB)
Non-trainable params: 0 (0.00 Byte)
```

Optimizer – Functions in Keras

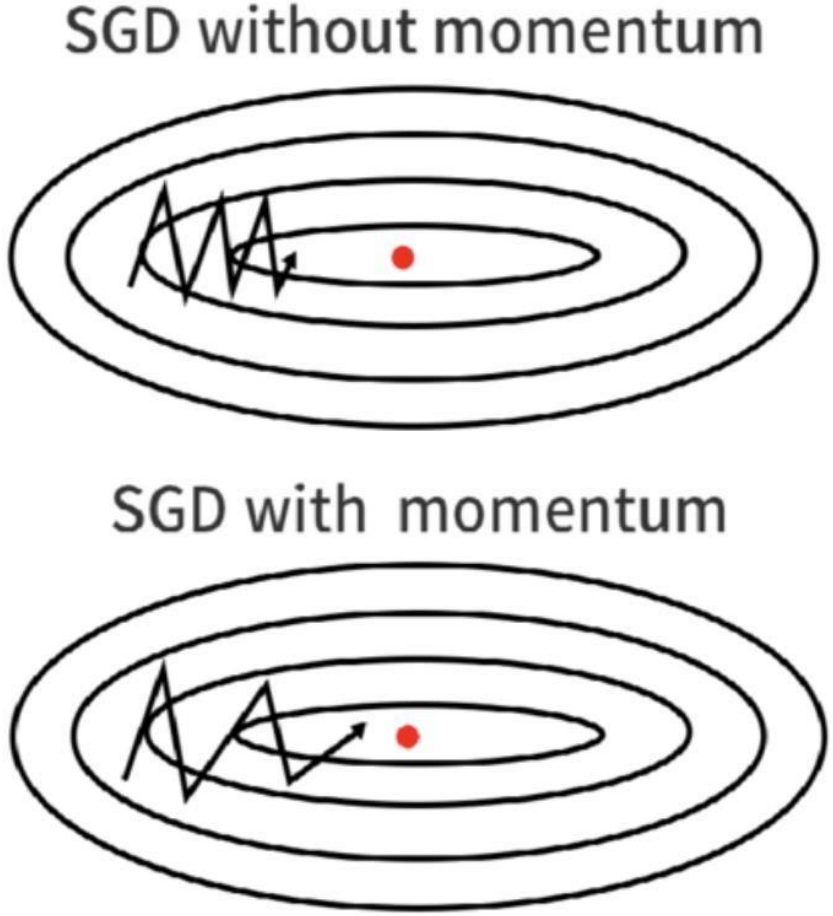
- SGD
- RMSprop
- Adam
- AdamW
- Adadelta
- Adagrad
- Adamax
- Adafactor
- Nadam
- Ftrl



https://keras.io/api/optimizers/

Optimizers

- Stochastic Gradient Descent (`optim.SGD`)
  - params: Model parameters
  - lr: Learning rate (required)
  - momentum: momentum factor (default: 0)
  - weight_decay: (default: 0)



Optimizers

- Adam ()
  - params: Model parameters
  - lr: Learning rate (default: 0.001)
  - betas: coefficients (tuple) used for computing running averages of gradient and its square (default: (0.9, 0.999))
  - eps: term added to denominator to improve numerical stability (default: 1e-8)
  - weight_decay: (default: 0)

Other Common Optimizers

- AdaDelta()
  - Precursor to Adam which uses first-order estimates to adapt learning rate
- Adamax()
  - Variant on Adam based on infinity norm
- RMSProp()
  - Take the square root of the gradient average before adding epsilon to normalization of LR

```
# Add Optimizer to minimize the loss

model1.compile(loss='mean_squared_error', optimizer=RMSprop(lr=0.007), metrics=['accuracy'])
```

Summarize Model

- Keras provides a way to summarize a model.
- The summary is textual and includes information about:
  - The layers and their order in the model.
  - The output shape of each layer.
  - The number of parameters (weights) in each layer.
  - The total number of parameters (weights) in the model.

Training Loss

- The training loss is a metric used to assess how a deep learning model fits the training data
- It assesses the error of the model on the training set.
- The training loss is calculated by taking the sum of errors for each example in the training set.

Validation Loss

- Validation loss is a metric used to assess the performance of a deep learning model on the validation set.
- The validation loss is similar to the training loss and is calculated from a sum of the errors for each example in the validation set.

Practice 5

Deep Neural Network procedure in Python code:

```
# 1. Import the library
# 2.Load data
        # 2.1 Input data
        # 2.2 Target Data
# 3. Pre-processing data
        # 3.1 Convert data to array
        # 3.2 Transpose the matrix
# 4. Normalize the data
        # 4.1 formula to normalize
```

Deep Neural Network procedure in Python code:

```
# 5. Label the target output
        # 5.1 formula to label the target output
# 6. Split data 80% for train and 20% for validation
# 7. Create deep learning model
# 8. Compile and train model
# 9. Visualize loss in deep learning
# 10. Visualize accuracy in deep learning
# 11. Save the models
# 11. Test model
# 12. Check the performance of model with actual output and predict output
```

Thanks!