Avoiding Overfitting Through Regularization

Regularization is the process of adding information in order to solve an ill-posed problem or to prevent overfitting. Regularization can be applied to objective functions in ill-posed optimization problems.

L1 Regularization | Lasso | Least Absolute:

$j_n(\theta) = j_0(\theta) + \alpha \sum_{i=1}^{m} |\theta_i|$

L2 Regularization | Ridge

$j_n(\theta) = j_0(\theta) + \frac{\alpha}{2} \sum_{i=1}^{m} (\theta_i)^2$

L1 - L2 Regularization

$j_n(\theta) = j_0(\theta) + r\alpha \sum_{i=1}^{m} |\theta_i| + \frac{(1-r)}{2}\alpha \sum_{i=1}^{m} (\theta_i)^2$

Dropout:

[Refer the paper (https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf)](https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf)

ℓ_1 and ℓ_2 regularization

In [ ]:
```
from tensorflow import keras
```

In [ ]:
```
layer = keras.layers.Dense(100, activation="elu",
                           kernel_initializer="he_normal",
                           kernel_regularizer=keras.regularizers.l2(0.01))
# or l1(0.1) for ℓ1 regularization with a factor or 0.1
# or l1_l2(0.1, 0.01) for both ℓ1 and ℓ2 regularization, with factors 0.1 and 0.01 respe
```

In [ ]:
```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(300, activation="elu",
                       kernel_initializer="he_normal",
                       kernel_regularizer=keras.regularizers.l2(0.01)),
    keras.layers.Dense(100, activation="elu",
                       kernel_initializer="he_normal",
                       kernel_regularizer=keras.regularizers.l2(0.01)),
    keras.layers.Dense(10, activation="softmax",
                       kernel_regularizer=keras.regularizers.l2(0.01))
])
model.compile(loss="sparse_categorical_crossentropy", optimizer="nadam", metrics=["accur
# n_epochs = 2
# history = model.fit(X_train_scaled, y_train, epochs=n_epochs,
#                     validation_data=(X_valid_scaled, y_valid))
```

In [ ]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_4 (Dense)              (None, 300)               235500
_________________________________________________________________
dense_5 (Dense)              (None, 100)               30100
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1010
=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
from functools import partial

RegularizedDense = partial(keras.layers.Dense,
                           activation="elu",
                           kernel_initializer="he_normal",
                           kernel_regularizer=keras.regularizers.l2(0.01))

model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    RegularizedDense(300),
    RegularizedDense(100),
    RegularizedDense(10, activation="softmax")
])
model.compile(loss="sparse_categorical_crossentropy", optimizer="nadam", metrics=["accur
# n_epochs = 2
# history = model.fit(X_train_scaled, y_train, epochs=n_epochs,
#                     validation_data=(X_valid_scaled, y_valid))
```

In [ ]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_2 (Flatten)          (None, 784)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 300)               235500    
_________________________________________________________________
dense_8 (Dense)              (None, 100)               30100     
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1010      
=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
_________________________________________________________________
```

Max-Norm Regularization

In [ ]:
```
from functools import partial

RegularizedDense = partial(keras.layers.Dense,
                           activation="elu",
                           kernel_initializer="he_normal",
                           kernel_regularizer=keras.regularizers.l2(0.01),
                           kernel_constraint=keras.constraints.max_norm(1.))

model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    RegularizedDense(300),
    RegularizedDense(100),
    RegularizedDense(10, activation="softmax")
])
model.compile(loss="sparse_categorical_crossentropy", optimizer="nadam", metrics=["accur
# n_epochs = 2
# history = model.fit(X_train_scaled, y_train, epochs=n_epochs,
#                     validation_data=(X_valid_scaled, y_valid))
```

In [ ]:
```
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_6 (Flatten)          (None, 784)               0         
_________________________________________________________________
dense_13 (Dense)             (None, 300)               235500    
_________________________________________________________________
dense_14 (Dense)             (None, 100)               30100     
_________________________________________________________________
dense_15 (Dense)             (None, 10)                1010      
=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
_________________________________________________________________
```

Dropout

In [ ]:
```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dropout(rate=0.2),
    keras.layers.Dense(300, activation="elu", kernel_initializer="he_normal"),
    keras.layers.Dropout(rate=0.2),
    keras.layers.Dense(100, activation="elu", kernel_initializer="he_normal"),
    keras.layers.Dropout(rate=0.2),
    keras.layers.Dense(10, activation="softmax")
])
model.compile(loss="sparse_categorical_crossentropy", optimizer="nadam", metrics=["accur
# n_epochs = 2
# history = model.fit(X_train_scaled, y_train, epochs=n_epochs,
#                     validation_data=(X_valid_scaled, y_valid))
```

In [ ]:
```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_3 (Flatten)          (None, 784)               0         
_________________________________________________________________
dropout (Dropout)            (None, 784)               0         
_________________________________________________________________
dense_10 (Dense)             (None, 300)               235500    
_________________________________________________________________
dropout_1 (Dropout)          (None, 300)               0         
_________________________________________________________________
dense_11 (Dense)             (None, 100)               30100     
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0         
_________________________________________________________________
dense_12 (Dense)             (None, 10)                1010      
=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
_________________________________________________________________
```

Solving a Regression Problem using ANN:

In [1]:

```
import pandas as pd
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
housing = fetch_california_housing()
housing
```

Out[2]:
```
{'data': array([[   8.3252    ,   41.        ,    6.98412698, ...,    2.55555556,
          37.88      , -122.23      ],
        [   8.3014    ,   21.        ,    6.23813708, ...,    2.10984183,
          37.86      , -122.22      ],
        [   7.2574    ,   52.        ,    8.28813559, ...,    2.80225989,
          37.85      , -122.24      ],
        ...,
        [   1.7       ,   17.        ,    5.20554273, ...,    2.3256351 ,
          39.43      , -121.22      ],
        [   1.8672    ,   18.        ,    5.32951289, ...,    2.12320917,
          39.43      , -121.32      ],
        [   2.3886    ,   16.        ,    5.25471698, ...,    2.61698113,
          39.37      , -121.24      ]]),
 'target': array([4.526, 3.585, 3.521, ..., 0.923, 0.847, 0.894]),
 'frame': None,
 'target_names': ['MedHouseVal'],
 'feature_names': ['MedInc',
  'HouseAge',
  'AveRooms',
  'AveBedrms',
  'Population',
  'AveOccup',
  'Latitude',
  'Longitude'],
 'DESCR': '.. _california_housing_dataset:\n\nCalifornia Housing dataset\n-------------
-------------\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 20640\n\n
:Number of Attributes: 8 numeric, predictive attributes and the target\n\n    :Attribut
e Information:\n        - MedInc        median income in block group\n        - HouseAg
e      median house age in block group\n        - AveRooms      average number of rooms
per household\n        - AveBedrms     average number of bedrooms per household\n
- Population    block group population\n        - AveOccup      average number of house
hold members\n        - Latitude      block group latitude\n        - Longitude     blo
ck group longitude\n\n    :Missing Attribute Values: None\n\nThis dataset was obtained
from the StatLib repository.\nhttps://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.h
tml\n\nThe target variable is the median house value for California districts,\nexpress
ed in hundreds of thousands of dollars ($100,000).\n\nThis dataset was derived from the
1990 U.S. census, using one row per census\nblock group. A block group is the smallest
geographical unit for which the U.S.\nCensus Bureau publishes sample data (a block grou
p typically has a population\nof 600 to 3,000 people).\n\nAn household is a group of pe
ople residing within a home. Since the average\nnumber of rooms and bedrooms in this da
taset are provided per household, these\ncolumns may take surpinsingly large values for
block groups with few households\nand many empty houses, such as vacation resorts.\n\nI
t can be downloaded/loaded using the\n:func:`sklearn.datasets.fetch_california_housing`
function.\n\n.. topic:: References\n\n    - Pace, R. Kelley and Ronald Barry, Sparse Sp
atial Autoregressions,\n      Statistics and Probability Letters, 33 (1997) 291-297\n'}
```

In [3]:
```
housing.keys()
```

Out[3]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'feature_names', 'DESCR'])
```

In [4]:
```
X = pd.DataFrame(housing.data, columns= housing.feature_names)
X.head()
```

Out[4]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25

In [5]:
```
y = pd.DataFrame(housing.target, columns=['target'])
y.head()
```

Out[5]:

	target
0	4.526
1	3.585
2	3.521
3	3.413
4	3.422

In [6]:
```
X.shape
```

Out[6]:
```
(20640, 8)
```

In [7]:
```
y.shape
```

Out[7]:
```
(20640, 1)
```

In [8]:
```
X_train_full, X_test, y_train_full, y_test = train_test_split(X,y, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full,y_train_full, random_
```

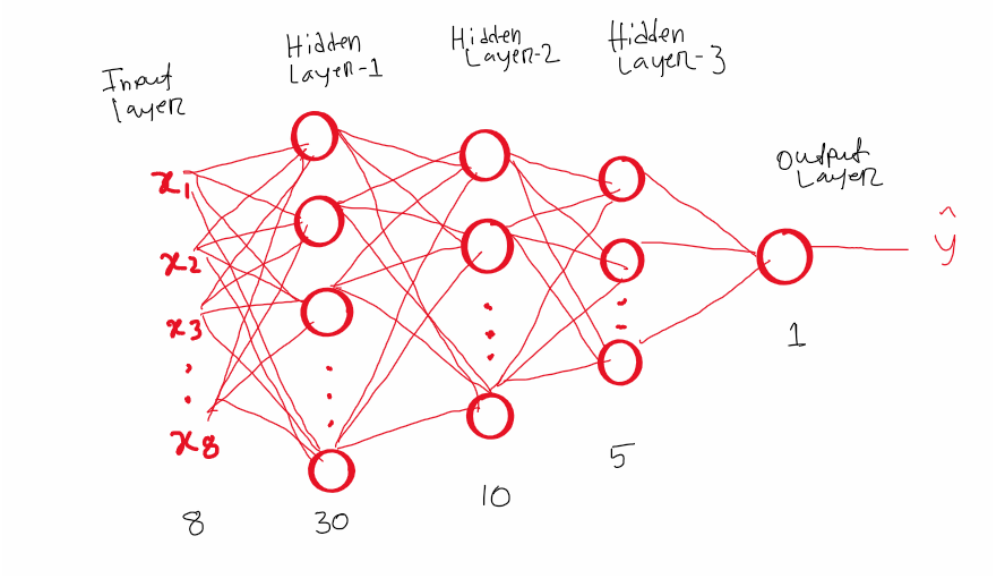
In [9]:
```
print(X_train_full.shape)
print(X_test.shape)
print(X_train.shape)
print(X_valid.shape)
```

```
(15480, 8)
(5160, 8)
(11610, 8)
(3870, 8)
```

In [10]:
```
X_train.shape[1:]
```

Out[10]:
```
(8,)
```

Architecture used:



In [11]:
```
LAYERS = [
         tf.keras.layers.Dense(30, activation="relu", input_shape = X_train.shape[1:]),
         tf.keras.layers.Dense(10, activation="relu"),
         tf.keras.layers.Dense(5, activation='relu'),
         tf.keras.layers.Dense(1)

]
```

Q)while defining the layer in classification you didn't applied Activation function and used Flatten ,but here you directly started from dense and applied RELU in the very first layer ,why?

The choice of layer architecture and activation functions in a neural network can vary depending on the specific task and the desired behavior of the model. Better option is, add relu activation function in dense layers and in output layer if it is binary classification add sigmoid otherwise add softmax.

In [12]:
```
model = tf.keras.models.Sequential(LAYERS)
```

In [13]:
```
LOSS = "mse"
OPTIMIZER = "sgd"

model.compile(optimizer= OPTIMIZER, loss= LOSS)
```

In [14]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 30)                270       
                                                                 
 dense_1 (Dense)             (None, 10)                310       
                                                                 
 dense_2 (Dense)             (None, 5)                 55        
                                                                 
 dense_3 (Dense)             (None, 1)                 6         
                                                                 
=================================================================
Total params: 641
Trainable params: 641
Non-trainable params: 0
_________________________________________________________________
```

In [15]:
```
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_valid = scaler.transform(X_valid)
X_test = scaler.transform(X_test)
```

In [16]:

```
EPOCHS = 20

history = model.fit( X_train, y_train, epochs= EPOCHS, validation_data=(X_valid, y_valid
```

```
Epoch 1/20
363/363 [==============================] - 3s 5ms/step - loss: 0.7655 - val_loss: 0.603
9
Epoch 2/20
363/363 [==============================] - 1s 3ms/step - loss: 0.4610 - val_loss: 0.392
6
Epoch 3/20
363/363 [==============================] - 1s 3ms/step - loss: 0.4087 - val_loss: 0.434
6
Epoch 4/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3891 - val_loss: 0.379
9
Epoch 5/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3752 - val_loss: 0.356
9
Epoch 6/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3666 - val_loss: 0.355
4
Epoch 7/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3619 - val_loss: 0.377
0
Epoch 8/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3593 - val_loss: 0.386
4
Epoch 9/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3572 - val_loss: 0.349
5
Epoch 10/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3527 - val_loss: 0.348
6
Epoch 11/20
363/363 [==============================] - 1s 2ms/step - loss: 0.3498 - val_loss: 0.348
5
Epoch 12/20
363/363 [==============================] - 1s 2ms/step - loss: 0.3467 - val_loss: 0.337
2
Epoch 13/20
363/363 [==============================] - 1s 2ms/step - loss: 0.3429 - val_loss: 0.361
6
Epoch 14/20
363/363 [==============================] - 1s 2ms/step - loss: 0.3407 - val_loss: 0.328
2
Epoch 15/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3392 - val_loss: 0.338
2
Epoch 16/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3389 - val_loss: 0.338
5
Epoch 17/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3336 - val_loss: 0.353
1
Epoch 18/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3334 - val_loss: 0.388
9
Epoch 19/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3309 - val_loss: 0.366
5
Epoch 20/20
```

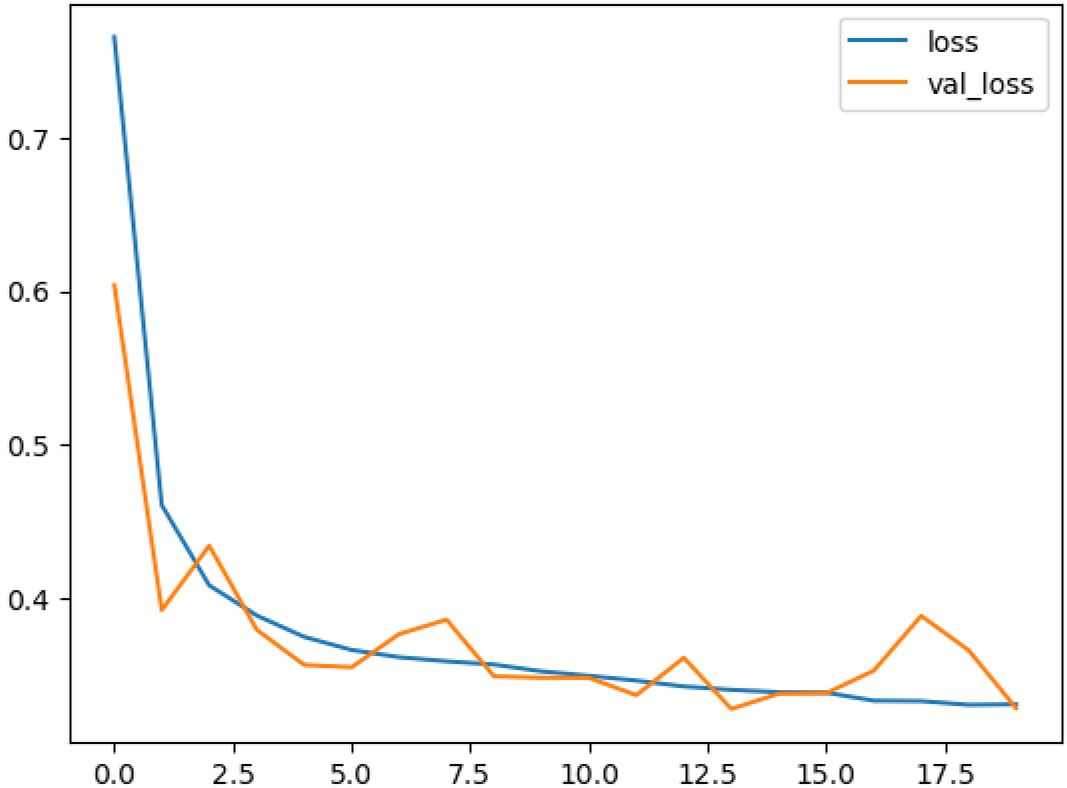
```
363/363 [==============================] - 1s 3ms/step - loss: 0.3313 - val_loss: 0.328
7
```

In [17]:
```
pd.DataFrame(history.history)
```

Out[17]:

	loss	val_loss
0	0.765455	0.603858
1	0.461002	0.392607
2	0.408732	0.434563
3	0.389118	0.379874
4	0.375180	0.356915
5	0.366648	0.355371
6	0.361943	0.376984
7	0.359288	0.386361
8	0.357164	0.349506
9	0.352737	0.348626
10	0.349832	0.348474
11	0.346712	0.337208
12	0.342892	0.361606
13	0.340700	0.328218
14	0.339156	0.338185
15	0.338889	0.338527
16	0.333643	0.353137
17	0.333382	0.388871
18	0.330911	0.366475
19	0.331260	0.328746

In [18]:
```
pd.DataFrame(history.history).plot()
```
Out[18]: <Axes: >



In [19]:
```
model.evaluate(X_test, y_test)
```
```
162/162 [==============================] - 0s 2ms/step - loss: 0.3218
```
Out[19]: 0.3217606842517853

In [20]:
```
X_test.shape
```
Out[20]: (5160, 8)

In [21]:
```
new = X_test[0]
```
In [29]:
```
new2 = X_test[1]
```
In [30]:
```
new
```
Out[30]:
```
array([-1.15780104, -0.28673138, -0.49550877, -0.16618097, -0.02946012,
        0.38899735,  0.19374821,  0.2870474 ])
```
In [23]:
```
new.shape
```
Out[23]: (8,)

In [24]:
```
X_test[0]
```

Out[24]:
```
array([-1.15780104, -0.28673138, -0.49550877, -0.16618097, -0.02946012,
        0.38899735,  0.19374821,  0.2870474 ])
```

In [25]:
```
new.reshape((1,8))
```

Out[25]:
```
array([[-1.15780104, -0.28673138, -0.49550877, -0.16618097, -0.02946012,
         0.38899735,  0.19374821,  0.2870474 ]])
```

In [31]:
```
new2.reshape((1,8))
```

Out[31]:
```
array([[-0.7125531 ,  0.10880952, -0.16332973,  0.20164652,  0.12842117,
        -0.11818174, -0.23725261,  0.06215231]])
```

In [26]:
```
model.predict(new.reshape((1,8)))
```

```
1/1 [==============================] - 0s 206ms/step
```

Out[26]:
```
array([[0.77056193]], dtype=float32)
```

In [32]:
```
model.predict(new2.reshape((1,8)))
```

```
1/1 [==============================] - 0s 41ms/step
```

Out[32]:
```
array([[1.4972047]], dtype=float32)
```

Model with callback

In [33]:
```
model_2 = tf.keras.models.Sequential(LAYERS)

LOSS = "mse"
OPTIMIZER = tf.keras.optimizers.SGD(learning_rate=1e-3)

model_2.compile(loss=LOSS , optimizer=OPTIMIZER)

EPOCHS = 20

checkpoint_cb = tf.keras.callbacks.ModelCheckpoint("my_keras_model.h5", save_best_only=T
early_stopping_cb = tf.keras.callbacks.EarlyStopping(patience=5, restore_best_weights=Tr
tensorboard_cb = tf.keras.callbacks.TensorBoard(log_dir="logs")

CALLBACKS = [checkpoint_cb, early_stopping_cb, tensorboard_cb]


history = model_2.fit(X_train, y_train, epochs = EPOCHS, validation_data=(X_valid, y_val
```

```
Epoch 1/20
363/363 [==============================] - 2s 4ms/step - loss: 0.3214 - val_loss: 0.323
4
Epoch 2/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3196 - val_loss: 0.323
5
Epoch 3/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3190 - val_loss: 0.321
9
Epoch 4/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3188 - val_loss: 0.322
1
Epoch 5/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3183 - val_loss: 0.320
9
Epoch 6/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3179 - val_loss: 0.319
8
Epoch 7/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3177 - val_loss: 0.320
0
Epoch 8/20
363/363 [==============================] - 2s 5ms/step - loss: 0.3173 - val_loss: 0.321
6
Epoch 9/20
363/363 [==============================] - 1s 4ms/step - loss: 0.3169 - val_loss: 0.320
7
Epoch 10/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3170 - val_loss: 0.319
4
Epoch 11/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3167 - val_loss: 0.319
8
Epoch 12/20
363/363 [==============================] - 1s 3ms/step - loss: 0.3164 - val_loss: 0.320
5
Epoch 13/20
363/363 [==============================] - 1s 4ms/step - loss: 0.3161 - val_loss: 0.319
6
Epoch 14/20
363/363 [==============================] - 1s 4ms/step - loss: 0.3157 - val_loss: 0.320
8
Epoch 15/20
363/363 [==============================] - 1s 4ms/step - loss: 0.3155 - val_loss: 0.322
5
```

In [34]:

```
%load_ext tensorboard
```