In [0]:

```
## Idea for this code is taken from the kernel https://www.kaggle.com/lopuhin/mercari-golf-0-3875-
cv-in-75-loc-1900-s
## Thanks to the authors for their elegant idea of using MLPs for this problem.

import warnings
warnings.filterwarnings('ignore')
import os
import gc
import time
from datetime import datetime

from contextlib import contextmanager

import keras as ks
import pandas as pd
import numpy as np
import scipy
import tensorflow as tf
from keras.models import load_model

from sklearn.preprocessing import OneHotEncoder
from sklearn.feature_extraction.text import TfidfVectorizer as Tfidf


from scipy.sparse import hstack
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_log_error as msle
from sklearn.model_selection import KFold
```

```
Using TensorFlow backend.
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info.

In [0]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [0]:

```
os.chdir('/content/drive/My Drive/Colab Notebooks/Mercari_NN')
```

In [0]:

```
@contextmanager
def timer(name):
    t0 = time.time()
    yield
    print(f'[{name}] done in {time.time() - t0:.0f} s')
```

In [0]:

```
def preprocess(df: pd.DataFrame) -> pd.DataFrame:
    df['name'] = df['name'].fillna('') + ' ' + df['brand_name'].fillna('')
    df['text'] = df['item_description'].fillna('') + ' ' + df['name'] + ' ' +
df['category_name'].fillna('')
    return df[['name', 'text', 'shipping', 'item_condition_id']]
```

Loading train data and splitting for Cross Validation

In [0]:

```
def load_train():

    train = pd.read_table('train.tsv')
    train = train[train['price'] > 0].reset_index(drop=True)
    cv = KFold(n_splits=20, shuffle=True, random_state=42)
    train_ids, valid_ids = next(cv.split(train))
    train, valid = train.iloc[train_ids], train.iloc[valid_ids]

    global y_scaler
    y_scaler = StandardScaler()

    global train_price, valid_price
    train_price = train['price'].values.reshape(-1, 1)
    valid_price = valid['price'].values.reshape(-1, 1)

    y_train = y_scaler.fit_transform(np.log1p(train_price))

    y_valid = y_scaler.transform(np.log1p(valid_price))

    return train, valid, y_train, y_valid
```

Pre-processing and featurization of train data

In [0]:

```
def process_train(train, valid):

    global vectorizer1, vectorizer2, vectorizer3, vectorizer4
    with timer('process train'):
        train = preprocess(train)
        vectorizer1 = Tfidf(max_features=100000, token_pattern='\w+', dtype=np.float32)
        train_namevec  = vectorizer1.fit_transform(train['name'].values)

        vectorizer2 = Tfidf(max_features=100000, token_pattern='\w+', ngram_range=(1, 2), dtype=np.
float32)
        train_textvec  = vectorizer2.fit_transform(train['text'].values)

        vectorizer3 = OneHotEncoder(dtype=np.float32)
        train_shipvec = vectorizer3.fit_transform(train['shipping'].values.reshape(-1, 1))

        vectorizer4 = OneHotEncoder(dtype=np.float32)
        train_conditionvec = vectorizer4.fit_transform(train['item_condition_id'].values.reshape(-1
, 1))

        X_train = hstack((train_namevec, train_textvec, train_shipvec, train_conditionvec)).tocsr()

    with timer('process valid'):
        valid = preprocess(valid)

        valid_namevec  = vectorizer1.transform(valid['name'].values)

        valid_textvec  = vectorizer2.transform(valid['text'].values)

        valid_shipvec = vectorizer3.transform(valid['shipping'].values.reshape(-1, 1))

        valid_conditionvec = vectorizer4.transform(valid['item_condition_id'].values.reshape(-1, 1)
)

        X_valid = hstack((valid_namevec, valid_textvec, valid_shipvec, valid_conditionvec)).tocsr()

        # Binarizing input
        Xb_train, Xb_valid = [x.astype(np.bool).astype(np.float32) for x in [X_train, X_valid]]


    return X_train, X_valid, Xb_train, Xb_valid
```

Processing test data

In [0]:

```
def load_process_test():

    test = pd.read_table('test.tsv')

    global predictions
    predictions = pd.DataFrame(test['test_id'])

    with timer('process test'):
        test = preprocess(test)

        test_namevec  = vectorizer1.transform(test['name'].values)

        test_textvec  = vectorizer2.transform(test['text'].values)

        test_shipvec = vectorizer3.transform(test['shipping'].values.reshape(-1, 1))

        test_conditionvec = vectorizer4.transform(test['item_condition_id'].values.reshape(-1, 1))

        X_test = hstack((test_namevec, test_textvec, test_shipvec, test_conditionvec)).tocsr()

        # Binarizing input
        Xb_test = X_test.astype(np.bool).astype(np.float32)

    return X_test, Xb_test
```

Model 1

In [0]:

```
def run_model1(X_train, y_train, X_valid, y_valid):
    '''- returns an MLP model trained on tfidf vectorized sparse input.
    - Does not perform best on binarized input.
    - Uses Adam optimizer with constant learning rate.
    - trains 2 epochs, Batch size is doubled at every epoch to speed up the optimization'''

    model_in = ks.Input(shape=(X_train.shape[1],), dtype='float32', sparse=True)
    out = ks.layers.Dense(256, activation='relu')(model_in)
    # out = ks.layers.Dropout(0.1)(out)      ## performance is better without dropouts
    out = ks.layers.Dense(64, activation='relu')(out)
    # out = ks.layers.Dropout(0.1)(out)
    out = ks.layers.Dense(64, activation='relu')(out)
    # out = ks.layers.Dropout(0.2)(out)
    out = ks.layers.Dense(32, activation='relu')(out)
    out = ks.layers.Dense(1)(out)
    model = ks.Model(model_in, out)

    model.compile(loss='mean_squared_error', optimizer=ks.optimizers.Adam(lr=3e-3))
    for i in range(2):
        with timer(f'epoch {i + 1}'):
            model.fit(x=X_train, y=y_train, batch_size=2**(9 + i), epochs=1, verbose=1, validation_c
ata=(X_valid, y_valid))

    return model
```

Model 2

In [0]:

```
def run_model2(Xb_train, y_train, Xb_valid, y_valid):
    '''- returns an MLP model trained on binarized sparse input.
    - Does not perform best on non-binarized(regular) input.
    - Uses Adam optimizer with constant learning rate.
    - trains 3 epochs, Batch size is doubled at every epoch to speed up the optimization'''

    model_in = ks.Input(shape=(Xb_train.shape[1],), dtype='float32', sparse=True)
    out = ks.layers.Dense(256, activation='relu')(model_in)
    # out = ks.layers.Dropout(0.1)(out)      ## performance is better without dropouts
    out = ks.layers.Dense(64, activation='relu')(out)
    # out = ks.layers.Dropout(0.1)(out)
    out = ks.layers.Dense(64, activation='relu')(out)
    # out = ks.layers.Dropout(0.2)(out)
    # out = ks.layers.Dropout(0.2)(out)
    out = ks.layers.Dense(32, activation='relu')(out)
    out = ks.layers.Dense(1)(out)
    model = ks.Model(model_in, out)

    model.compile(loss='mean_squared_error', optimizer=ks.optimizers.Adam(lr=3e-3))
    for i in range(3):
        with timer(f'epoch {i + 1}'):
            model.fit(x=Xb_train, y=y_train, batch_size=2**(9 + i), epochs=1, verbose=1, validation_
data=(Xb_valid, y_valid))

    return model
```

Main function

In [0]:

```
DEVELOP = True # Set to True for only trainng and validation

def main():

    start_time = datetime.now()

    print('\n\nLoading and processing train data.....')
    train, valid, y_train, y_valid = load_train()

    X_train, X_valid, Xb_train, Xb_valid = process_train(train, valid)
    print(X_train.shape, y_train.shape, X_valid.shape, y_valid.shape)

    del train, valid
    gc.collect()

    ## Running model1 on regualr data X_train, X_valid
    print('\n\nRunning model on regular (non-binary) input.....')
    model1 = run_model1(X_train, y_train, X_valid, y_valid)
    model1.save('model1.h5')
    model1 = load_model('model1.h5')
    model1.summary()
    pred1 = model1.predict(X_valid)[:, 0]

    y_pred = np.expm1(y_scaler.inverse_transform(pred1.reshape(-1, 1))[:, 0])
    print('1st run val RMSLE: {:.4f}'.format(np.sqrt(msle(valid_price, y_pred))))

    ## Running again
    model2 = run_model1(X_train, y_train, X_valid, y_valid)
    model2.save('model2.h5')
    model2 = load_model('model2.h5')
    model2.summary()
    pred2 = model2.predict(X_valid)[:, 0]

    y_pred = np.expm1(y_scaler.inverse_transform(pred2.reshape(-1, 1))[:, 0])
    print('2nd run val RMSLE: {:.4f}'.format(np.sqrt(msle(valid_price, y_pred))))

    ## Running model2 on binarized data Xb_train, Xb_valid
    print('\n\nRunning model on binarized input.....')
    model3 = run_model2(Xb_train, y_train, Xb_valid, y_valid)
    model3.save('model3.h5')
    model3 = load_model('model3.h5')
    model3.summary()
    pred3 = model3.predict(Xb_valid)[:, 0]

    y_pred = np.expm1(y_scaler.inverse_transform(pred3.reshape(-1, 1))[:, 0])
    print('3rd run val RMSLE: {:.4f}'.format(np.sqrt(msle(valid_price, y_pred))))

    ## Running again
    model4 = run_model2(Xb_train, y_train, Xb_valid, y_valid)
    model4.save('model4.h5')
    model4 = load_model('model4.h5')
    model4.summary()
    pred4 = model4.predict(Xb_valid)[:, 0]

    y_pred = np.expm1(y_scaler.inverse_transform(pred4.reshape(-1, 1))[:, 0])
    print('4th run val RMSLE: {:.4f}'.format(np.sqrt(msle(valid_price, y_pred))))
    ## Final Prediction = weighted average of predictions of 4 models/runs
    print('\n\nEnsemble (weighted average of predictions from 4 models/runs).....')
    y_pred = np.average([pred1, pred2, pred3, pred4], weights=[0.33, 0.33, 0.17, 0.17], axis=0)
    y_pred = np.expm1(y_scaler.inverse_transform(y_pred.reshape(-1, 1))[:, 0])
    print('Final valid RMSLE: {:.4f}'.format(np.sqrt(msle(valid_price, y_pred))))

    if DEVELOP==False:
        ## This block loads and predicts on test data if DEVELOP is not set

        # print('\n\nLoading and processing test data.....')
        X_test, Xb_test = load_process_test()
        print(X_test.shape, Xb_test.shape)

        with timer('predict test'):
            test_pred1 = model1.predict(X_test)[:, 0]
            test_pred2 = model2.predict(X_test)[:, 0]
            test_pred3 = model3.predict(Xb_test)[:, 0]
            test_pred4 = model4.predict(Xb_test)[:, 0]


        test_pred = np.average([test_pred1, test_pred2, test_pred3, test_pred4], weights=[0.33, 0.3
3, 0.17, 0.17], axis=0)
        test_pred = np.expm1(y_scaler.inverse_transform(test_pred.reshape(-1, 1))[:, 0])

        print('\n\nCreating submisssion file.....')
        predictions['price'] = test_pred

        predictions.to_csv('predictions.csv', index=False)


    print(f'Code finished execution in {datetime.now() - start_time}')

if __name__ == '__main__':
    main()
```

```
Loading and processing train data.....
[process train] done in 152 s
[process valid] done in 7 s
(1407577, 200007) (1407577, 1) (74084, 200007) (74084, 1)


Running model on regular (non-binary) input.....
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:66: The name tf.get_default_graph is deprecated. Plea
se use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:539: The name tf.sparse_placeholder is deprecated. Pl
ease use tf.compat.v1.sparse_placeholder instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:4432: The name tf.random_uniform is deprecated. Pleas
e use tf.random.uniform instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:1133: The name tf.sparse_tensor_dense_matmul is depre
cated. Please use tf.sparse.sparse_dense_matmul instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:793: The name t
f.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:541: The name tf.placeholder is deprecated. Please us
e tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:1033: The name tf.assign_add is deprecated. Please us
e tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:1020: The name tf.assign is deprecated. Please use tf
.compat.v1.assign instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:3005: The name tf.Session is deprecated. Please use t
f.compat.v1.Session instead.
```

```
f.compat.v1.Session instead.

Train on 1407577 samples, validate on 74084 samples
Epoch 1/1
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:190: The name tf.get_default_session is deprecated. P
lease use tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:197: The name tf.ConfigProto is deprecated. Please us
e tf.compat.v1.ConfigProto instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:207: The name tf.global_variables is deprecated. Plea
se use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:216: The name tf.is_variable_initialized is
deprecated. Please use tf.compat.v1.is_variable_initialized instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:223: The name tf.variables_initializer is deprecated.
Please use tf.compat.v1.variables_initializer instead.

1407577/1407577 [==============================] - 64s 45us/step - loss: 0.3439 - val_loss: 0.2992
[epoch 1] done in 64 s
Train on 1407577 samples, validate on 74084 samples
Epoch 1/1
1407577/1407577 [==============================] - 32s 23us/step - loss: 0.2029 - val_loss: 0.2910
[epoch 2] done in 32 s
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (None, 200007)            0         
_________________________________________________________________
dense_1 (Dense)              (None, 256)               51202048  
_________________________________________________________________
dense_2 (Dense)              (None, 64)                16448     
_________________________________________________________________
dense_3 (Dense)              (None, 64)                4160      
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2080      
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 33        
=================================================================
Total params: 51,224,769
Trainable params: 51,224,769
Non-trainable params: 0
_________________________________________________________________
1st run val RMSLE: 0.4023
Train on 1407577 samples, validate on 74084 samples
Epoch 1/1
1407577/1407577 [==============================] - 63s 45us/step - loss: 0.3438 - val_loss: 0.3002
[epoch 1] done in 63 s
Train on 1407577 samples, validate on 74084 samples
Epoch 1/1
1407577/1407577 [==============================] - 33s 23us/step - loss: 0.2021 - val_loss: 0.2871
[epoch 2] done in 33 s
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_2 (InputLayer)         (None, 200007)            0         
_________________________________________________________________
dense_6 (Dense)              (None, 256)               51202048  
_________________________________________________________________
dense_7 (Dense)              (None, 64)                16448     
_________________________________________________________________
dense_8 (Dense)              (None, 64)                4160      
_________________________________________________________________
dense_9 (Dense)              (None, 32)                2080      
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 33        
=================================================================
Total params: 51,224,769
Trainable params: 51,224,769
Non-trainable params: 0
```

```
Non-trainable params: 0
_________________________________________________________________
2nd run val RMSLE: 0.3996


Running model on binarized input.....
Train on 1407577 samples, validate on 74084 samples
Epoch 1/1
1407577/1407577 [==============================] - 63s 45us/step - loss: 0.3500 - val_loss: 0.3060
[epoch 1] done in 63 s
Train on 1407577 samples, validate on 74084 samples
Epoch 1/1
1407577/1407577 [==============================] - 32s 23us/step - loss: 0.2077 - val_loss: 0.2934
[epoch 2] done in 32 s
Train on 1407577 samples, validate on 74084 samples
Epoch 1/1
1407577/1407577 [==============================] - 18s 13us/step - loss: 0.1187 - val_loss: 0.2949
[epoch 3] done in 18 s
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 200007)            0
_________________________________________________________________
dense_11 (Dense)             (None, 256)               51202048
_________________________________________________________________
dense_12 (Dense)             (None, 64)                16448
_________________________________________________________________
dense_13 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_14 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 33
=================================================================
Total params: 51,224,769
Trainable params: 51,224,769
Non-trainable params: 0
_________________________________________________________________
3rd run val RMSLE: 0.4050
Train on 1407577 samples, validate on 74084 samples
Epoch 1/1
1407577/1407577 [==============================] - 65s 46us/step - loss: 0.3501 - val_loss: 0.3043
[epoch 1] done in 66 s
Train on 1407577 samples, validate on 74084 samples
Epoch 1/1
1407577/1407577 [==============================] - 37s 26us/step - loss: 0.2089 - val_loss: 0.2937
[epoch 2] done in 37 s
Train on 1407577 samples, validate on 74084 samples
Epoch 1/1
1407577/1407577 [==============================] - 19s 14us/step - loss: 0.1202 - val_loss: 0.2958
[epoch 3] done in 19 s
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         (None, 200007)            0
_________________________________________________________________
dense_16 (Dense)             (None, 256)               51202048
_________________________________________________________________
dense_17 (Dense)             (None, 64)                16448
_________________________________________________________________
dense_18 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_19 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_20 (Dense)             (None, 1)                 33
=================================================================
Total params: 51,224,769
Trainable params: 51,224,769
Non-trainable params: 0
_________________________________________________________________
4th run val RMSLE: 0.4056


Ensemble (weighted average of predictions from 4 models/runs).....
Final valid RMSLE: 0.3848
Code finished execution in 0:10:29.996681
```

- **Compared to the original code, I have made some changes in the MLP architecture as well as parameters such as learning rate and batch size to get better results.**
- **I also tried using dropouts(0.1, 0.2, 0.3, .. 0.5) but the models performed better without dropouts (I got a validation RMSLE of 0.3872 with dropouts). So I have removed dropouts in the final code.**
- **I also experimented with diffrent activation units ('tanh', 'sigmoid', 'linear', 'relu'). 'relu' performs significantly better that rest all.**
- **I have also changed the number of epochs from 3 to 2 for model1 (non-binary data), as the model starts overfitting from 3rd epoch.**
- **Instead of taking simple mean, I have taken weighted average of predictions from 4 different models/runs.**
- **For simplicity of the code, and also because I have used Google Colab(training is faster with GPUs than with multi-core CPUs), I have trained the models one after another unlike pool processing in the original kernel. The code finishes runnng in decent amount of time on Colab.**
- **The validation RMSLE I got was 0.3848 as compared to 0.3875 in the source kernel.**