lfr-transformer-training-inference

July 7, 2023

Hello Fellow Kagglers,

This competition has been challenging to say the least, especially the submission.

After a few weeks I finally got a working training + inference pipeline which is shared through this notebook.

The model consists of a transformer embedding + encoder + decoder.

Inference is performed by starting with an SOS token and predicting one character at a time using the previous prediction.

Feel free to ask for clarafications or comment.

Notebook will be updated periodically.

Preprocessing Notebook

V6

This competition has an inference limit of 5 hours which requires careful allocation of computational resources in the model. Most changes are based on the assymetrical number of encoder/deocder calls during inference.

Inference requires the encoder to encode the input frames and subsequently use that encoding to predict the 1st character by inputting the encoding and SOS (Start of Sentence) token. Next, the encoding, SOS token and 1st predicted token are used to predict the 2nd character. Inference thus requires 1 call to the encoder and multiple calls to the encoder. On average a phrase is 18 characters long, requiring 18+1(SOS token) calls to the decoder. To stay within the 5 hour inference limit the encoder can be computationally heavy, however the decoder should be light.

Some inspiration is taken from the 1st place solution - training from the last Google - Isolated Sign Language Recognition competition.

- Increased training epochs 30 -> 100
- Using all data for training, no validation set
- Increased number of decoder blocks 2 -> 3
- Increased encoder dimensions 256 -> 384
- Halved attention dimension to decrease computational intensity of Multi Head Attention
- Added 20% dropout to multi head attention output
- Batch size 128 -> 64
- Classification layer linear activation for logits in loss function

Helpful Tutorials

English-to-Spanish translation with a sequence-to-sequence Transformer

Lecture 12.1 Self-attention

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sn
import tensorflow as tf
import tensorflow_addons as tfa

from tqdm.notebook import tqdm
from sklearn.model_selection import train_test_split, GroupShuffleSplit
from leven import levenshtein

import glob
import sys
import os
import math
import gc
import sys
import sklearn
import time
import json

# TQDM Progress Bar With Pandas Apply Function
tqdm.pandas()

print(f'Tensorflow Version {tf.__version__}')
print(f'Python Version: {sys.version}')
```

```
/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/__init__.py:98:
UserWarning: unable to load libtensorflow_io_plugins.so: unable to open file:
libtensorflow_io_plugins.so, from paths: ['/opt/conda/lib/python3.10/site-
packages/tensorflow_io/python/ops/libtensorflow_io_plugins.so']
caused by: ['/opt/conda/lib/python3.10/site-
packages/tensorflow_io/python/ops/libtensorflow_io_plugins.so: undefined symbol:
_ZN3tsl6StatusC1EN10tensorflow5error4CodeESt17basic_string_viewIcSt11char_traits
IcEENS_14SourceLocationE']
  warnings.warn(f"unable to load libtensorflow_io_plugins.so: {e}")
/opt/conda/lib/python3.10/site-
packages/tensorflow_io/python/ops/__init__.py:104: UserWarning: file system
plugins are not loaded: unable to open file: libtensorflow_io.so, from paths:
['/opt/conda/lib/python3.10/site-
packages/tensorflow_io/python/ops/libtensorflow_io.so']
caused by: ['/opt/conda/lib/python3.10/site-
packages/tensorflow_io/python/ops/libtensorflow_io.so: undefined symbol:
```

```
_ZTVN10tensorflow13GcsFileSystemE']
  warnings.warn(f"file system plugins are not loaded: {e}")
/opt/conda/lib/python3.10/site-
packages/tensorflow_addons/utils/tfa_eol_msg.py:23: UserWarning:

TensorFlow Addons (TFA) has ended development and introduction of new features.
TFA has entered a minimal maintenance and release mode until a planned end of
life in May 2024.
Please modify downstream libraries to take dependencies from other repositories
in our TensorFlow community (e.g. Keras, Keras-CV, and Keras-NLP).

For more information see: https://github.com/tensorflow/addons/issues/2807

  warnings.warn(

Tensorflow Version 2.12.0
Python Version: 3.10.10 | packaged by conda-forge | (main, Mar 24 2023,
20:08:06) [GCC 11.3.0]
```

1 Character 2 Ordinal Encoding

[2]:
```
# Read Character to Ordinal Encoding Mapping
with open('/kaggle/input/asl-fingerspelling/character_to_prediction_index.
  ↪json') as json_file:
    CHAR2ORD = json.load(json_file)

# Ordinal to Character Mapping
ORD2CHAR = {j:i for i,j in CHAR2ORD.items()}

# Character to Ordinal Encoding Mapping
display(pd.Series(CHAR2ORD).to_frame('Ordinal Encoding'))
```

	Ordinal Encoding
	0
!	1
#	2
$	3
%	4
&	5
'	6
(	7
)	8
*	9
+	10
,	11
-	12
.	13

/	14
0	15
1	16
2	17
3	18
4	19
5	20
6	21
7	22
8	23
9	24
:	25
;	26
=	27
?	28
@	29
[	30
_	31
a	32
b	33
c	34
d	35
e	36
f	37
g	38
h	39
i	40
j	41
k	42
l	43
m	44
n	45
o	46
p	47
q	48
r	49
s	50
t	51
u	52
v	53
w	54
x	55
y	56
z	57
~	58

2 Global Config

[3]:
```
# If Notebook Is Run By Committing or In Interactive Mode For Development
IS_INTERACTIVE = os.environ['KAGGLE_KERNEL_RUN_TYPE'] == 'Interactive'
# Verbose Setting during training
VERBOSE = 1 if IS_INTERACTIVE else 2
# Global Random Seed
SEED = 42
# Number of Frames to resize recording to
N_TARGET_FRAMES = 128
# Global debug flag, takes subset of train
DEBUG = False
# Number of Unique Characters To Predict + Pad Token + SOS Token + EOS Token
N_UNIQUE_CHARACTERS0 = len(CHAR2ORD)
N_UNIQUE_CHARACTERS = len(CHAR2ORD) + 1 + 1 + 1
PAD_TOKEN = len(CHAR2ORD) # Padding
SOS_TOKEN = len(CHAR2ORD) + 1 # Start Of Sentence
EOS_TOKEN = len(CHAR2ORD) + 2 # End Of Sentence
# Whether to use 10% of data for validation
USE_VAL = False
# Batch Size
BATCH_SIZE = 64
# Number of Epochs to Train for
N_EPOCHS = 2 if IS_INTERACTIVE else 100
# Number of Warmup Epochs in Learning Rate Scheduler
N_WARMUP_EPOCHS = 10
# Maximum Learning Rate
LR_MAX = 1e-3
# Weight Decay Ratio as Ratio of Learning Rate
WD_RATIO = 0.05
# Length of Phrase + EOS Token
MAX_PHRASE_LENGTH = 31 + 1
# Whether to Train The model
TRAIN_MODEL = True
# Whether to Load Pretrained Weights
LOAD_WEIGHTS = False
# Learning Rate Warmup Method [log,exp]
WARMUP_METHOD = 'exp'
```

3 Plot Config

[4]:
```
# MatplotLib Global Settings
mpl.rcParams.update(mpl.rcParamsDefault)
mpl.rcParams['xtick.labelsize'] = 16
mpl.rcParams['ytick.labelsize'] = 16
mpl.rcParams['axes.labelsize'] = 18
```

```
mpl.rcParams['axes.titlesize'] = 24
```

4 Train

[5]:
```
# Read Train DataFrame
if DEBUG:
    train = pd.read_csv('/kaggle/input/asl-fingerspelling/train.csv').head(5000)
else:
    train = pd.read_csv('/kaggle/input/asl-fingerspelling/train.csv')

# Set Train Indexed By sqeuence_id
train_sequence_id = train.set_index('sequence_id')

# Number Of Train Samples
N_SAMPLES = len(train)
print(f'N_SAMPLES: {N_SAMPLES}')

display(train.info())
display(train.head())
```

```
N_SAMPLES: 67208
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 67208 entries, 0 to 67207
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   path            67208 non-null  object
 1   file_id         67208 non-null  int64
 2   sequence_id     67208 non-null  int64
 3   participant_id  67208 non-null  int64
 4   phrase          67208 non-null  object
dtypes: int64(3), object(2)
memory usage: 2.6+ MB
```

```
None
```

	path	file_id	sequence_id	participant_id	phrase
0	train_landmarks/5414471.parquet	5414471	1816796431	217	3 creekhouse
1	train_landmarks/5414471.parquet	5414471	1816825349	107	scales/kuhaylah
2	train_landmarks/5414471.parquet	5414471	1816909464	1	1383 william lanier
3	train_landmarks/5414471.parquet	5414471	1816967051	63	
4	train_landmarks/5414471.parquet	5414471	1817123330	89	

```
3          988 franklin lane
4  6920 northeast 661st road
```

5 File Path

[6]:
```
# Get complete file path to file
def get_file_path(path):
    return f'/kaggle/input/asl-fingerspelling/{path}'

train['file_path'] = train['path'].apply(get_file_path)
```

6 Example File Paths

[7]:
```
# Unique Parquet Files
INFERENCE_FILE_PATHS = pd.Series(
        glob.glob('/kaggle/input/aslfr-preprocessing-dataset/
 ↪train_landmark_subsets/*')
    )

print(f'Found {len(INFERENCE_FILE_PATHS)} Inference Pickle Files')
```

```
Found 10 Inference Pickle Files
```

7 Load X/y

[8]:
```
# Train/Validation
if USE_VAL:
    # TRAIN
    X_train = np.load('/kaggle/input/aslfr-preprocessing-dataset/X_train.npy')
    y_train = np.load('/kaggle/input/aslfr-preprocessing-dataset/y_train.npy')[:
 ↪,:MAX_PHRASE_LENGTH]
    N_TRAIN_SAMPLES = len(X_train)
    # VAL
    X_val = np.load('/kaggle/input/aslfr-preprocessing-dataset/X_val.npy')
    y_val = np.load('/kaggle/input/aslfr-preprocessing-dataset/y_val.npy')[:,:
 ↪MAX_PHRASE_LENGTH]
    N_VAL_SAMPLES = len(X_val)
    # Shapes
    print(f'X_train shape: {X_train.shape}, X_val shape: {X_val.shape}')
# Train On All Data
else:
    # TRAIN
    X_train = np.load('/kaggle/input/aslfr-preprocessing-dataset/X.npy')
    y_train = np.load('/kaggle/input/aslfr-preprocessing-dataset/y.npy')[:,:
 ↪MAX_PHRASE_LENGTH]
```

```
    N_TRAIN_SAMPLES = len(X_train)
    print(f'X_train shape: {X_train.shape}')
```

```
X_train shape: (61955, 128, 164)
```

8 Example Batch

[9]:
```
# Example Batch For Debugging
N_EXAMPLE_BATCH_SAMPLES = 1024
N_EXAMPLE_BATCH_SAMPLES_SMALL = 32
# Example Batch
X_batch = {
    'frames': np.copy(X_train[:N_EXAMPLE_BATCH_SAMPLES]),
    'phrase': np.copy(y_train[:N_EXAMPLE_BATCH_SAMPLES]),
#     'phrase_type': np.copy(y_phrase_type_train[:N_EXAMPLE_BATCH_SAMPLES]),
}
y_batch = np.copy(y_train[:N_EXAMPLE_BATCH_SAMPLES])
# Small Example Batch
X_batch_small = {
    'frames': np.copy(X_train[:N_EXAMPLE_BATCH_SAMPLES_SMALL]),
    'phrase': np.copy(y_train[:N_EXAMPLE_BATCH_SAMPLES_SMALL]),
#     'phrase_type': np.copy(y_phrase_type_train[:
 ↪N_EXAMPLE_BATCH_SAMPLES_SMALL]),
}
y_batch_small = np.copy(y_train[:N_EXAMPLE_BATCH_SAMPLES_SMALL])
```

9 Example Parquet

[10]:
```
# Read First Parquet File
# example_parquet_df = pd.read_parquet(train['file_path'][0])
example_parquet_df = pd.read_parquet(INFERENCE_FILE_PATHS[0])

# Each parquet file contains 1000 recordings
print(f'# Unique Recording: {example_parquet_df.index.nunique()}')
# Display DataFrame layout
display(example_parquet_df.head())
```

```
# Unique Recording: 1000
```

sequence_id	x_left_hand_0	x_left_hand_1	x_left_hand_2	x_left_hand_3	\
1816796431	NaN	NaN	NaN	NaN	
1816796431	NaN	NaN	NaN	NaN	
1816796431	NaN	NaN	NaN	NaN	
1816796431	NaN	NaN	NaN	NaN	
1816796431	NaN	NaN	NaN	NaN	

```
             x_left_hand_4  x_left_hand_5  x_left_hand_6  x_left_hand_7  \
sequence_id
1816796431             NaN            NaN            NaN            NaN
1816796431             NaN            NaN            NaN            NaN
1816796431             NaN            NaN            NaN            NaN
1816796431             NaN            NaN            NaN            NaN
1816796431             NaN            NaN            NaN            NaN

             x_left_hand_8  x_left_hand_9  …  y_face_314  y_face_317  \
sequence_id                                …
1816796431             NaN            NaN  …    0.551424    0.538415
1816796431             NaN            NaN  …    0.550706    0.538216
1816796431             NaN            NaN  …    0.550613    0.537836
1816796431             NaN            NaN  …    0.549740    0.536994
1816796431             NaN            NaN  …    0.550614    0.538677

             y_face_318  y_face_321  y_face_324  y_face_375  y_face_402  \
sequence_id
1816796431     0.539000    0.546458    0.539715    0.543958    0.538425
1816796431     0.538723    0.545990    0.539296    0.543357    0.538225
1816796431     0.538564    0.545949    0.539212    0.543279    0.537961
1816796431     0.538449    0.545622    0.539666    0.543694    0.537328
1816796431     0.540376    0.547104    0.541524    0.545222    0.539203

             y_face_405  y_face_409  y_face_415
sequence_id
1816796431     0.549351    0.538230    0.540015
1816796431     0.548827    0.537376    0.539256
1816796431     0.548796    0.537360    0.539332
1816796431     0.548015    0.538301    0.539954
1816796431     0.549211    0.539734    0.541707

[5 rows x 164 columns]
```

10 Landmark Indices

[11]:
```python
# Get indices in original dataframe
def get_idxs(df, words_pos, words_neg=[], ret_names=True, idxs_pos=None):
    idxs = []
    names = []
    for w in words_pos:
        for col_idx, col in enumerate(example_parquet_df.columns):
            # Exclude Non Landmark Columns
            if col in ['frame']:
                continue
```

```
            col_idx = int(col.split('_')[-1])
            # Check if column name contains all words
            if (w in col) and (idxs_pos is None or col_idx in idxs_pos) and␣
 ↪all([w not in col for w in words_neg]):
                idxs.append(col_idx)
                names.append(col)
    # Convert to Numpy arrays
    idxs = np.array(idxs)
    names = np.array(names)
    # Returns either both column indices and names
    if ret_names:
        return idxs, names
    # Or only columns indices
    else:
        return idxs
```

[12]:
```
# Lips Landmark Face Ids
LIPS_LANDMARK_IDXS = np.array([
        61, 185, 40, 39, 37, 0, 267, 269, 270, 409,
        291, 146, 91, 181, 84, 17, 314, 405, 321, 375,
        78, 191, 80, 81, 82, 13, 312, 311, 310, 415,
        95, 88, 178, 87, 14, 317, 402, 318, 324, 308,
    ])

# Landmark Indices for Left/Right hand without z axis in raw data
LEFT_HAND_IDXS0, LEFT_HAND_NAMES0 = get_idxs(example_parquet_df, ['left_hand'],␣
 ↪['z'])
RIGHT_HAND_IDXS0, RIGHT_HAND_NAMES0 = get_idxs(example_parquet_df,␣
 ↪['right_hand'], ['z'])
LIPS_IDXS0, LIPS_NAMES0 = get_idxs(example_parquet_df, ['face'], ['z'],␣
 ↪idxs_pos=LIPS_LANDMARK_IDXS)
COLUMNS0 = np.concatenate((LEFT_HAND_NAMES0, RIGHT_HAND_NAMES0, LIPS_NAMES0))
N_COLS0 = len(COLUMNS0)
# Only X/Y axes are used
N_DIMS0 = 2

print(f'N_COLS0: {N_COLS0}')
```

```
N_COLS0: 164
```

[13]:
```
# Landmark Indices in subset of dataframe with only COLUMNS selected
LEFT_HAND_IDXS = np.argwhere(np.isin(COLUMNS0, LEFT_HAND_NAMES0)).squeeze()
RIGHT_HAND_IDXS = np.argwhere(np.isin(COLUMNS0, RIGHT_HAND_NAMES0)).squeeze()
LIPS_IDXS = np.argwhere(np.isin(COLUMNS0, LIPS_NAMES0)).squeeze()
HAND_IDXS = np.concatenate((LEFT_HAND_IDXS, RIGHT_HAND_IDXS), axis=0)
N_COLS = N_COLS0
# Only X/Y axes are used
```

```
N_DIMS = 2

print(f'N_COLS: {N_COLS}')
```

```
N_COLS: 164
```

[14]:
```
# Indices in processed data by axes with only dominant hand
HAND_X_IDXS = np.array(
        [idx for idx, name in enumerate(LEFT_HAND_NAMES0) if 'x' in name]
    ).squeeze()
HAND_Y_IDXS = np.array(
        [idx for idx, name in enumerate(LEFT_HAND_NAMES0) if 'y' in name]
    ).squeeze()
# Names in processed data by axes
HAND_X_NAMES = LEFT_HAND_NAMES0[HAND_X_IDXS]
HAND_Y_NAMES = LEFT_HAND_NAMES0[HAND_Y_IDXS]
```

11 Mean/STD Loading

[15]:
```
# Mean/Standard Deviations of data used for normalizing
MEANS = np.load('/kaggle/input/aslfr-preprocessing-dataset/MEANS.npy').
 ↪reshape(-1)
STDS = np.load('/kaggle/input/aslfr-preprocessing-dataset/STDS.npy').reshape(-1)
```

12 Tensorflow Preprocessing Layer

[16]:
```
"""
    Tensorflow layer to process data in TFLite
    Data needs to be processed in the model itself, so we can not use Python
"""
class PreprocessLayer(tf.keras.layers.Layer):
    def __init__(self):
        super(PreprocessLayer, self).__init__()
        self.normalisation_correction = tf.constant(
                    # Add 0.50 to x coordinates of left hand (original right␣
 ↪hand) and substract 0.50 of right hand (original left hand)
                    [0.50 if 'x' in name else 0.00 for name in␣
 ↪LEFT_HAND_NAMES0],
                dtype=tf.float32,
            )

    @tf.function(
        input_signature=(tf.TensorSpec(shape=[None,N_COLS0], dtype=tf.
 ↪float32),),
    )
```

```
    def call(self, data0, resize=True):
        # Fill NaN Values With 0
        data = tf.where(tf.math.is_nan(data0), 0.0, data0)

        # Hacky
        data = data[None]

        # Empty Hand Frame Filtering
        hands = tf.slice(data, [0,0,0], [-1, -1, 84])
        hands = tf.abs(hands)
        mask = tf.reduce_sum(hands, axis=2)
        mask = tf.not_equal(mask, 0)
        data = data[mask][None]

        # Pad Zeros
        N_FRAMES = len(data[0])
        if N_FRAMES < N_TARGET_FRAMES:
            data = tf.concat((
                data,
                tf.zeros([1,N_TARGET_FRAMES-N_FRAMES,N_COLS], dtype=tf.float32)
            ), axis=1)
        # Downsample
        data = tf.image.resize(
            data,
            [1, N_TARGET_FRAMES],
            method=tf.image.ResizeMethod.BILINEAR,
        )

        # Squeeze Batch Dimension
        data = tf.squeeze(data, axis=[0])

        return data

preprocess_layer = PreprocessLayer()
```

[17]:
```
# Function To Test Preprocessing Layer
def test_preprocess_layer():
    demo_sequence_id = example_parquet_df.index.unique()[15]
    demo_raw_data = example_parquet_df.loc[demo_sequence_id, COLUMNS0]
    data = preprocess_layer(demo_raw_data)

    print(f'demo_raw_data shape: {demo_raw_data.shape}')
    print(f'data shape: {data.shape}')

    return data

if IS_INTERACTIVE:
```

```
    data = test_preprocess_layer()
```

13 Train Dataset

[18]:
```
# Train Dataset Iterator
def get_train_dataset(X, y, batch_size=BATCH_SIZE):
    sample_idxs = np.arange(len(X))
    while True:
        # Get random indices
        random_sample_idxs = np.random.choice(sample_idxs, batch_size)

        inputs = {
            'frames': X[random_sample_idxs],
            'phrase': y[random_sample_idxs],
        }
        outputs = y[random_sample_idxs]

        yield inputs, outputs
```

[19]:
```
# Train Dataset
train_dataset = get_train_dataset(X_train, y_train)
```

[20]:
```
# Training Steps Per Epoch
TRAIN_STEPS_PER_EPOCH = math.ceil(N_TRAIN_SAMPLES / BATCH_SIZE)
print(f'TRAIN_STEPS_PER_EPOCH: {TRAIN_STEPS_PER_EPOCH}')
```

```
TRAIN_STEPS_PER_EPOCH: 969
```

14 Validation Dataset

[21]:
```
# Validation Set
def get_val_dataset(X, y, batch_size=BATCH_SIZE):
    offsets = np.arange(0, len(X), batch_size)
    while True:
        # Iterate over whole validation set
        for offset in offsets:
            inputs = {
                'frames': X[offset:offset+batch_size],
                'phrase': y[offset:offset+batch_size],
            }
            outputs = y[offset:offset+batch_size]

            yield inputs, outputs
```

[22]:
```
# Validation Dataset
if USE_VAL:
    val_dataset = get_val_dataset(X_val, y_val)
```

[23]:
```
if USE_VAL:
    N_VAL_STEPS_PER_EPOCH = math.ceil(N_VAL_SAMPLES / BATCH_SIZE)
    print(f'N_VAL_STEPS_PER_EPOCH: {N_VAL_STEPS_PER_EPOCH}')
```

15 Model Config

[24]:
```
# Epsilon value for layer normalisation
LAYER_NORM_EPS = 1e-6

# final embedding and transformer embedding size
UNITS_ENCODER = 384
UNITS_DECODER = 256

# Transformer
NUM_BLOCKS_ENCODER = 3
NUM_BLOCKS_DECODER = 2
NUM_HEADS = 4
MLP_RATIO = 2

# Dropout
EMBEDDING_DROPOUT = 0.00
MLP_DROPOUT_RATIO = 0.30
MHA_DROPOUT_RATIO = 0.20
CLASSIFIER_DROPOUT_RATIO = 0.10

# Initiailizers
INIT_HE_UNIFORM = tf.keras.initializers.he_uniform
INIT_GLOROT_UNIFORM = tf.keras.initializers.glorot_uniform
INIT_ZEROS = tf.keras.initializers.constant(0.0)
# Activations
GELU = tf.keras.activations.gelu
```

16 Landmark Embedding

[25]:
```
# Embeds a landmark using fully connected layers
class LandmarkEmbedding(tf.keras.Model):
    def __init__(self, units, name):
        super(LandmarkEmbedding, self).__init__(name=f'{name}_embedding')
        self.units = units
        self.supports_masking = True
```

```
    def build(self, input_shape):
        # Embedding for missing landmark in frame, initizlied with zeros
        self.empty_embedding = self.add_weight(
            name=f'{self.name}_empty_embedding',
            shape=[self.units],
            initializer=INIT_ZEROS,
        )
        # Embedding
        self.dense = tf.keras.Sequential([
            tf.keras.layers.Dense(self.units, name=f'{self.name}_dense_1',␣
↪use_bias=False, kernel_initializer=INIT_GLOROT_UNIFORM, activation=GELU),
            tf.keras.layers.Dense(self.units, name=f'{self.name}_dense_2',␣
↪use_bias=False, kernel_initializer=INIT_HE_UNIFORM),
        ], name=f'{self.name}_dense')

    def call(self, x):
        return tf.where(
                # Checks whether landmark is missing in frame
                tf.reduce_sum(x, axis=2, keepdims=True) == 0,
                # If so, the empty embedding is used
                self.empty_embedding,
                # Otherwise the landmark data is embedded
                self.dense(x),
            )
```

17 Embedding

[26]:
```
# Creates embedding for each frame
class Embedding(tf.keras.Model):
    def __init__(self):
        super(Embedding, self).__init__()
        self.supports_masking = True

    def build(self, input_shape):
        # Positional embedding for each frame index
        self.positional_embedding = tf.Variable(
            initial_value=tf.zeros([N_TARGET_FRAMES, UNITS_ENCODER], dtype=tf.
↪float32),
            trainable=True,
            name='embedding_positional_encoder',
        )
        # Embedding layer for Landmarks
        self.dominant_hand_embedding = LandmarkEmbedding(UNITS_ENCODER,␣
↪'dominant_hand')

    def call(self, x, training=False):
```

```
        # Normalize
        x = tf.where(
                tf.math.equal(x, 0.0),
                0.0,
                (x - MEANS) / STDS,
            )
        # Dominant Hand
        x = self.dominant_hand_embedding(x)
        # Add Positional Encoding
        x = x + self.positional_embedding

        return x
```

18 Transformer

[27]:
```
# based on: https://stackoverflow.com/questions/67342988/
 ↪verifying-the-implementation-of-multihead-attention-in-transformer
# replaced softmax with softmax layer to support masked softmax
def scaled_dot_product(q,k,v, softmax, attention_mask):
    #calculates Q . K(transpose)
    qkt = tf.matmul(q,k,transpose_b=True)
    #caculates scaling factor
    dk = tf.math.sqrt(tf.cast(q.shape[-1],dtype=tf.float32))
    scaled_qkt = qkt/dk
    softmax = softmax(scaled_qkt, mask=attention_mask)
    z = tf.matmul(softmax,v)
    #shape: (m,Tx,depth), same shape as q,k,v
    return z

class MultiHeadAttention(tf.keras.layers.Layer):
    def __init__(self,d_model, num_of_heads, dropout, d_out=None):
        super(MultiHeadAttention,self).__init__()
        self.d_model = d_model
        self.num_of_heads = num_of_heads
        self.depth = d_model//num_of_heads
        self.wq = [tf.keras.layers.Dense(self.depth//2, use_bias=False) for i␣
 ↪in range(num_of_heads)]
        self.wk = [tf.keras.layers.Dense(self.depth//2, use_bias=False) for i␣
 ↪in range(num_of_heads)]
        self.wv = [tf.keras.layers.Dense(self.depth//2, use_bias=False) for i␣
 ↪in range(num_of_heads)]
        self.wo = tf.keras.layers.Dense(d_model if d_out is None else d_out,␣
 ↪use_bias=False)
        self.softmax = tf.keras.layers.Softmax()
        self.do = tf.keras.layers.Dropout(dropout)
        self.supports_masking = True
```

```
    def call(self, q, k, v, attention_mask=None, training=False):

        multi_attn = []
        for i in range(self.num_of_heads):
            Q = self.wq[i](q)
            K = self.wk[i](k)
            V = self.wv[i](v)
            multi_attn.append(scaled_dot_product(Q,K,V, self.softmax,␣
 ↪attention_mask))

        multi_head = tf.concat(multi_attn, axis=-1)
        multi_head_attention = self.wo(multi_head)
        multi_head_attention = self.do(multi_head_attention, training=training)

        return multi_head_attention
```

19 Encoder

source

[28]:
```
# Encoder based on multiple transformer blocks
class Encoder(tf.keras.Model):
    def __init__(self, num_blocks):
        super(Encoder, self).__init__(name='encoder')
        self.num_blocks = num_blocks
        self.supports_masking = True

    def build(self, input_shape):
        self.ln_1s = []
        self.mhas = []
        self.ln_2s = []
        self.mlps = []
        # Make Transformer Blocks
        for i in range(self.num_blocks):
            # First Layer Normalisation
            self.ln_1s.append(tf.keras.layers.
 ↪LayerNormalization(epsilon=LAYER_NORM_EPS))
            # Multi Head Attention
            self.mhas.append(MultiHeadAttention(UNITS_ENCODER, NUM_HEADS,␣
 ↪MHA_DROPOUT_RATIO))
            # Second Layer Normalisation
            self.ln_2s.append(tf.keras.layers.
 ↪LayerNormalization(epsilon=LAYER_NORM_EPS))
            # Multi Layer Perception
            self.mlps.append(tf.keras.Sequential([
```

```
                tf.keras.layers.Dense(UNITS_ENCODER * MLP_RATIO,␣
↪activation=GELU, kernel_initializer=INIT_GLOROT_UNIFORM, use_bias=False),
                tf.keras.layers.Dropout(MLP_DROPOUT_RATIO),
                tf.keras.layers.Dense(UNITS_ENCODER,␣
↪kernel_initializer=INIT_HE_UNIFORM, use_bias=False),
            ]))
            # Optional Projection to Decoder Dimension
            if UNITS_ENCODER != UNITS_DECODER:
                self.dense_out = tf.keras.layers.Dense(UNITS_DECODER,␣
↪kernel_initializer=INIT_GLOROT_UNIFORM, use_bias=False)
                self.apply_dense_out = True
            else:
                self.apply_dense_out = False

    def call(self, x, x_inp, training=False):
        # Attention mask to ignore missing frames
        attention_mask = tf.where(tf.math.reduce_sum(x_inp, axis=[2]) == 0.0, 0.
↪0, 1.0)
        attention_mask = tf.expand_dims(attention_mask, axis=1)
        attention_mask = tf.repeat(attention_mask, repeats=N_TARGET_FRAMES,␣
↪axis=1)
        # Iterate input over transformer blocks
        for ln_1, mha, ln_2, mlp in zip(self.ln_1s, self.mhas, self.ln_2s, self.
↪mlps):
            x = ln_1(x + mha(x, x, x, attention_mask=attention_mask))
            x = ln_2(x + mlp(x))

        # Optional Projection to Decoder Dimension
        if self.apply_dense_out:
            x = self.dense_out(x)

        return x
```

20 Decoder

[29]:
```
# Decoder based on multiple transformer blocks
class Decoder(tf.keras.Model):
    def __init__(self, num_blocks):
        super(Decoder, self).__init__(name='decoder')
        self.num_blocks = num_blocks
        self.supports_masking = True

    def build(self, input_shape):
        # Positional Embedding, initialized with zeros
        self.positional_embedding = tf.Variable(
```

```
            initial_value=tf.zeros([N_TARGET_FRAMES, UNITS_DECODER], dtype=tf.
↪float32),
            trainable=True,
            name='embedding_positional_encoder',
        )
        # Character Embedding
        self.char_emb = tf.keras.layers.Embedding(N_UNIQUE_CHARACTERS,␣
↪UNITS_DECODER, embeddings_initializer=INIT_ZEROS)
        # Positional Encoder MHA
        self.pos_emb_mha = MultiHeadAttention(UNITS_DECODER, NUM_HEADS,␣
↪MHA_DROPOUT_RATIO)
        self.pos_emb_ln = tf.keras.layers.
↪LayerNormalization(epsilon=LAYER_NORM_EPS)
        # First Layer Normalisation
        self.ln_1s = []
        self.mhas = []
        self.ln_2s = []
        self.mlps = []
        # Make Transformer Blocks
        for i in range(self.num_blocks):
            # First Layer Normalisation
            self.ln_1s.append(tf.keras.layers.
↪LayerNormalization(epsilon=LAYER_NORM_EPS))
            # Multi Head Attention
            self.mhas.append(MultiHeadAttention(UNITS_DECODER, NUM_HEADS,␣
↪MHA_DROPOUT_RATIO))
            # Second Layer Normalisation
            self.ln_2s.append(tf.keras.layers.
↪LayerNormalization(epsilon=LAYER_NORM_EPS))
            # Multi Layer Perception
            self.mlps.append(tf.keras.Sequential([
                tf.keras.layers.Dense(UNITS_DECODER * MLP_RATIO,␣
↪activation=GELU, kernel_initializer=INIT_GLOROT_UNIFORM, use_bias=False),
                tf.keras.layers.Dropout(MLP_DROPOUT_RATIO),
                tf.keras.layers.Dense(UNITS_DECODER,␣
↪kernel_initializer=INIT_HE_UNIFORM, use_bias=False),
            ]))

    def get_causal_attention_mask(self, B):
        i = tf.range(N_TARGET_FRAMES)[:, tf.newaxis]
        j = tf.range(N_TARGET_FRAMES)
        mask = tf.cast(i >= j, dtype=tf.int32)
        mask = tf.reshape(mask, (1, N_TARGET_FRAMES, N_TARGET_FRAMES))
        mult = tf.concat(
            [tf.expand_dims(B, -1), tf.constant([1, 1], dtype=tf.int32)],
            axis=0,
```

```
        )
        mask = tf.tile(mask, mult)
        mask = tf.cast(mask, tf.float32)
        return mask

    def call(self, encoder_outputs, phrase, training=False):
        # Batch Size
        B = tf.shape(encoder_outputs)[0]
        # Cast to INT32
        phrase = tf.cast(phrase, tf.int32)
        # Prepend SOS Token
        phrase = tf.pad(phrase, [[0,0], [1,0]], constant_values=SOS_TOKEN,␣
↪name='prepend_sos_token')
        # Pad With PAD Token
        phrase = tf.pad(phrase, [[0,0],␣
↪[0,N_TARGET_FRAMES-MAX_PHRASE_LENGTH-1]], constant_values=PAD_TOKEN,␣
↪name='append_pad_token')
        # Causal Mask
        causal_mask = self.get_causal_attention_mask(B)
        # Positional Embedding
        x = self.positional_embedding + self.char_emb(phrase)
        # Causal Attention
        x = self.pos_emb_ln(x + self.pos_emb_mha(x, x, x,␣
↪attention_mask=causal_mask))
        # Iterate input over transformer blocks
        for ln_1, mha, ln_2, mlp in zip(self.ln_1s, self.mhas, self.ln_2s, self.
↪mlps):
            x = ln_1(x + mha(x, encoder_outputs, encoder_outputs,␣
↪attention_mask=causal_mask))
            x = ln_2(x + mlp(x))
        # Slice 31 Characters
        x = tf.slice(x, [0, 0, 0], [-1, MAX_PHRASE_LENGTH, -1])

        return x
```

[30]:
```
# Causal Attention to make decoder not attent to future characters which it␣
↪needs to predict
def get_causal_attention_mask(B):
    i = tf.range(N_TARGET_FRAMES)[:, tf.newaxis]
    j = tf.range(N_TARGET_FRAMES)
    mask = tf.cast(i >= j, dtype=tf.int32)
    mask = tf.reshape(mask, (1, N_TARGET_FRAMES, N_TARGET_FRAMES))
    mult = tf.concat(
        [tf.expand_dims(B, -1), tf.constant([1, 1], dtype=tf.int32)],
        axis=0,
    )
```

```
    mask = tf.tile(mask, mult)
    mask = tf.cast(mask, tf.float32)
    return mask

get_causal_attention_mask(1)
```

[30]:
```
<tf.Tensor: shape=(1, 128, 128), dtype=float32, numpy=
array([[[1., 0., 0., …, 0., 0., 0.],
        [1., 1., 0., …, 0., 0., 0.],
        [1., 1., 1., …, 0., 0., 0.],
        …,
        [1., 1., 1., …, 1., 0., 0.],
        [1., 1., 1., …, 1., 1., 0.],
        [1., 1., 1., …, 1., 1., 1.]]], dtype=float32)>
```

21 Non Pad/SOS/EOS Token Accuracy

[31]:
```
# TopK accuracy for multi dimensional output
class TopKAccuracy(tf.keras.metrics.Metric):
    def __init__(self, k, **kwargs):
        super(TopKAccuracy, self).__init__(name=f'top{k}acc', **kwargs)
        self.top_k_acc = tf.keras.metrics.SparseTopKCategoricalAccuracy(k=k)

    def update_state(self, y_true, y_pred, sample_weight=None):
        y_true = tf.reshape(y_true, [-1])
        y_pred = tf.reshape(y_pred, [-1, N_UNIQUE_CHARACTERS])
        character_idxs = tf.where(y_true < N_UNIQUE_CHARACTERS0)
        y_true = tf.gather(y_true, character_idxs, axis=0)
        y_pred = tf.gather(y_pred, character_idxs, axis=0)
        self.top_k_acc.update_state(y_true, y_pred)

    def result(self):
        return self.top_k_acc.result()

    def reset_state(self):
        self.top_k_acc.reset_state()
```

22 Loss Weights

[32]:
```
# Create Initial Loss Weights All Set To 1
loss_weights = np.ones(N_UNIQUE_CHARACTERS, dtype=np.float32)
# Set Loss Weight Of Pad Token To 0
loss_weights[PAD_TOKEN] = 0
```

23 Sparse Categorical Crossentropy With Label Smoothing¶

[33]:
```
# source:: https://stackoverflow.com/questions/60689185/
 ↪label-smoothing-for-sparse-categorical-crossentropy
def scce_with_ls(y_true, y_pred):
    # Filter Pad Tokens
    idxs = tf.where(y_true != PAD_TOKEN)
    y_true = tf.gather_nd(y_true, idxs)
    y_pred = tf.gather_nd(y_pred, idxs)
    # One Hot Encode Sparsely Encoded Target Sign
    y_true = tf.cast(y_true, tf.int32)
    y_true = tf.one_hot(y_true, N_UNIQUE_CHARACTERS, axis=1)
    # Categorical Crossentropy with native label smoothing support
    loss = tf.keras.losses.categorical_crossentropy(y_true, y_pred,␣
 ↪label_smoothing=0.25, from_logits=True)
    loss = tf.math.reduce_mean(loss)
    return loss
```

24 Model

[34]:
```
def get_model():
    # Inputs
    frames_inp = tf.keras.layers.Input([N_TARGET_FRAMES, N_COLS], dtype=tf.
 ↪float32, name='frames')
    phrase_inp = tf.keras.layers.Input([MAX_PHRASE_LENGTH], dtype=tf.int32,␣
 ↪name='phrase')
    # Frames
    x = frames_inp

    # Masking
    x = tf.keras.layers.Masking(mask_value=0.0, input_shape=(N_TARGET_FRAMES,␣
 ↪N_COLS))(x)

    # Embedding
    x = Embedding()(x)

    # Encoder Transformer Blocks
    x = Encoder(NUM_BLOCKS_ENCODER)(x, frames_inp)

    # Decoder
    x = Decoder(NUM_BLOCKS_DECODER)(x, phrase_inp)

    # Classifier
    x = tf.keras.Sequential([
        # Dropout
        tf.keras.layers.Dropout(CLASSIFIER_DROPOUT_RATIO),
```

```
        # Output Neurons
        tf.keras.layers.Dense(N_UNIQUE_CHARACTERS, activation=tf.keras.
↪activations.linear, kernel_initializer=INIT_HE_UNIFORM, use_bias=False),
    ], name='classifier')(x)

    outputs = x

    # Create Tensorflow Model
    model = tf.keras.models.Model(inputs=[frames_inp, phrase_inp],␣
↪outputs=outputs)

    # Categorical Crossentropy Loss With Label Smoothing
    loss = scce_with_ls

    # Adam Optimizer
    optimizer = tfa.optimizers.RectifiedAdam(sma_threshold=4)
    optimizer = tfa.optimizers.Lookahead(optimizer, sync_period=5)

    # TopK Metrics
    metrics = [
        TopKAccuracy(1),
        TopKAccuracy(5),
    ]

    model.compile(
        loss=loss,
        optimizer=optimizer,
        metrics=metrics,
        loss_weights=loss_weights,
    )

    return model
```

[35]:
```
# Input data
for k, v in X_batch.items():
    print(f'{k}: {v.shape}')
```

```
frames: (1024, 128, 164)
phrase: (1024, 32)
```

[36]:
```
tf.keras.backend.clear_session()

model = get_model()
```

[37]:
```
# Plot model summary
model.summary(expand_nested=True)
```

```
Model: "model"
__________________________________________________________________________________
__________________
 Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
==================
 frames (InputLayer)             [(None, 128, 164)]   0           []

 masking (Masking)               (None, 128, 164)     0
['frames[0][0]']

 embedding (Embedding)           (None, 128, 384)     259968
['masking[0][0]']
|--------------------------------------------------------------------------------
-----------------|
| dominant_hand_embedding (Landm  multiple          210816      []
|
| arkEmbedding)
|
||------------------------------------------------------------------------------
----------------||
|| dominant_hand_embedding_dense   (None, 128, 384)  210432    []
||
|| (Sequential)
||
|||----------------------------------------------------------------------------
---------------|||
||| dominant_hand_embedding_dense_  (None, 128, 384)  62976   []
|||
||| 1 (Dense)
|||
|||
|||
||| dominant_hand_embedding_dense_  (None, 128, 384)  147456  []
|||
||| 2 (Dense)
|||
||------------------------------------------------------------------------------
----------------||
|--------------------------------------------------------------------------------
-----------------|
__________________________________________________________________________________
__________________
 encoder (Encoder)               (None, 128, 256)     2757120
['embedding[0][0]',
'frames[0][0]']
|--------------------------------------------------------------------------------
-----------------|
```

```
| layer_normalization (LayerNorm  multiple          768         []
|
| alization)
|
|
|
| layer_normalization_2 (LayerNo  multiple          768         []
|
| rmalization)
|
|
|
| layer_normalization_4 (LayerNo  multiple          768         []
|
| rmalization)
|
|
|
| multi_head_attention (MultiHea  multiple          294912      []
|
| dAttention)
|
|
|
| multi_head_attention_1 (MultiH  multiple          294912      []
|
| eadAttention)
|
|
|
| multi_head_attention_2 (MultiH  multiple          294912      []
|
| eadAttention)
|
|
|
| layer_normalization_1 (LayerNo  multiple          768         []
|
| rmalization)
|
|
|
| layer_normalization_3 (LayerNo  multiple          768         []
|
| rmalization)
|
|
|
```

```
| layer_normalization_5 (LayerNo  multiple          768         []
|
| rmalization)
|
|
|
| sequential (Sequential)      (None, 128, 384)     589824      []
|
||--------------------------------------------------------------------------
----------------||
|| dense_13 (Dense)           (None, 128, 768)     294912      []
||
||
||
|| dropout_1 (Dropout)        (None, 128, 768)     0           []
||
||
||
|| dense_14 (Dense)           (None, 128, 384)     294912      []
||
|--------------------------------------------------------------------------
-----------------|
| sequential_1 (Sequential)    (None, 128, 384)     589824      []
|
||--------------------------------------------------------------------------
----------------||
|| dense_29 (Dense)           (None, 128, 768)     294912      []
||
||
||
|| dropout_3 (Dropout)        (None, 128, 768)     0           []
||
||
||
|| dense_30 (Dense)           (None, 128, 384)     294912      []
||
|--------------------------------------------------------------------------
-----------------|
| sequential_2 (Sequential)    (None, 128, 384)     589824      []
|
||--------------------------------------------------------------------------
----------------||
|| dense_45 (Dense)           (None, 128, 768)     294912      []
||
||
||
|| dropout_5 (Dropout)        (None, 128, 768)     0           []
||
```

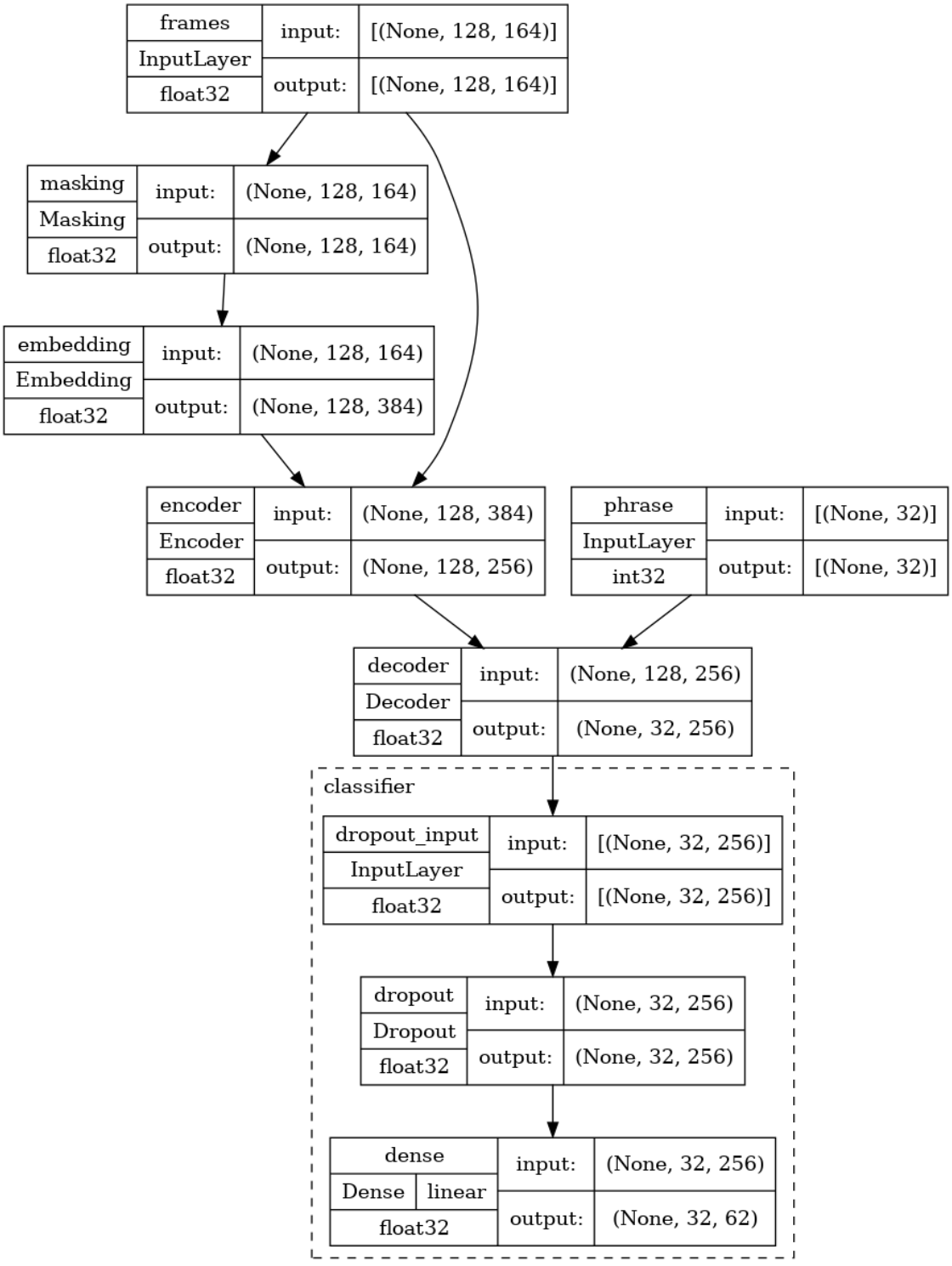
```
||
||
|| dense_46 (Dense)           (None, 128, 384)     294912      []
||
|-------------------------------------------------------------------------------
-----------------|
| dense_47 (Dense)             multiple             98304       []
|
-------------------------------------------------------------------------------
------------------
 phrase (InputLayer)            [(None, 32)]         0           []

 decoder (Decoder)              (None, 32, 256)      968704
['encoder[0][0]',
'phrase[0][0]']
|-------------------------------------------------------------------------------
-----------------|
| embedding (Embedding)        multiple             15872       []
|
|
|
| multi_head_attention (MultiHea  multiple          131072      []
|
| dAttention)
|
|
|
| layer_normalization (LayerNorm  multiple          512         []
|
| alization)
|
|
|
| layer_normalization_1 (LayerNo  multiple          512         []
|
| rmalization)
|
|
|
| layer_normalization_3 (LayerNo  multiple          512         []
|
| rmalization)
|
|
|
| multi_head_attention_1 (MultiH  multiple          131072      []
|
| eadAttention)
```

```
|
|
|
| multi_head_attention_2 (MultiH  multiple          131072      []
|
| eadAttention)
|
|
|
| layer_normalization_2 (LayerNo  multiple          512         []
|
| rmalization)
|
|
|
| layer_normalization_4 (LayerNo  multiple          512         []
|
| rmalization)
|
|
|
| sequential (Sequential)      (None, 128, 256)     262144      []
|
||-------------------------------------------------------------------------
----------------||
|| dense_26 (Dense)           (None, 128, 512)     131072      []
||
||
||
|| dropout_2 (Dropout)        (None, 128, 512)     0           []
||
||
||
|| dense_27 (Dense)           (None, 128, 256)     131072      []
||
|-------------------------------------------------------------------------
-----------------|
| sequential_1 (Sequential)    (None, 128, 256)     262144      []
|
||-------------------------------------------------------------------------
----------------||
|| dense_41 (Dense)           (None, 128, 512)     131072      []
||
||
||
|| dropout_4 (Dropout)        (None, 128, 512)     0           []
||
||
```

```
||
|| dense_42 (Dense)            (None, 128, 256)     131072      []
||
|-----------------------------------------------------------------------------------
-----------------|
-----------------------------------------------------------------------------------
------------------
 classifier (Sequential)        (None, 32, 62)       15872
['decoder[0][0]']
|-----------------------------------------------------------------------------------
-----------------|
| dropout (Dropout)             (None, 32, 256)      0           []
|
|
|
| dense (Dense)                 (None, 32, 62)       15872       []
|
-----------------------------------------------------------------------------------
------------------
===================================================================================
==================
Total params: 4,001,664
Trainable params: 4,001,664
Non-trainable params: 0
-----------------------------------------------------------------------------------
------------------
```

[38]:
```
# Plot Model Architecture
tf.keras.utils.plot_model(model, show_shapes=True, show_dtype=True,␣
 ↪show_layer_names=True, expand_nested=True, show_layer_activations=True)
```

[38]:


frames
InputLayer
float32
input:
output:
[(None, 128, 164)]
[(None, 128, 164)]
masking
Masking
float32
input:
output:
(None, 128, 164)
(None, 128, 164)
embedding
Embedding
float32
input:
output:
(None, 128, 164)
(None, 128, 384)
encoder
Encoder
float32
input:
output:
(None, 128, 384)
(None, 128, 256)
phrase
InputLayer
int32
input:
output:
[(None, 32)]
[(None, 32)]
decoder
Decoder
float32
input:
output:
(None, 128, 256)
(None, 32, 256)
classifier
dropout_input
InputLayer
float32
input:
output:
[(None, 32, 256)]
[(None, 32, 256)]
dropout
Dropout
float32
input:
output:
(None, 32, 256)
(None, 32, 256)
dense
Dense
linear
float32
input:
output:
(None, 32, 256)
(None, 32, 62)

25 Verify Training Flag

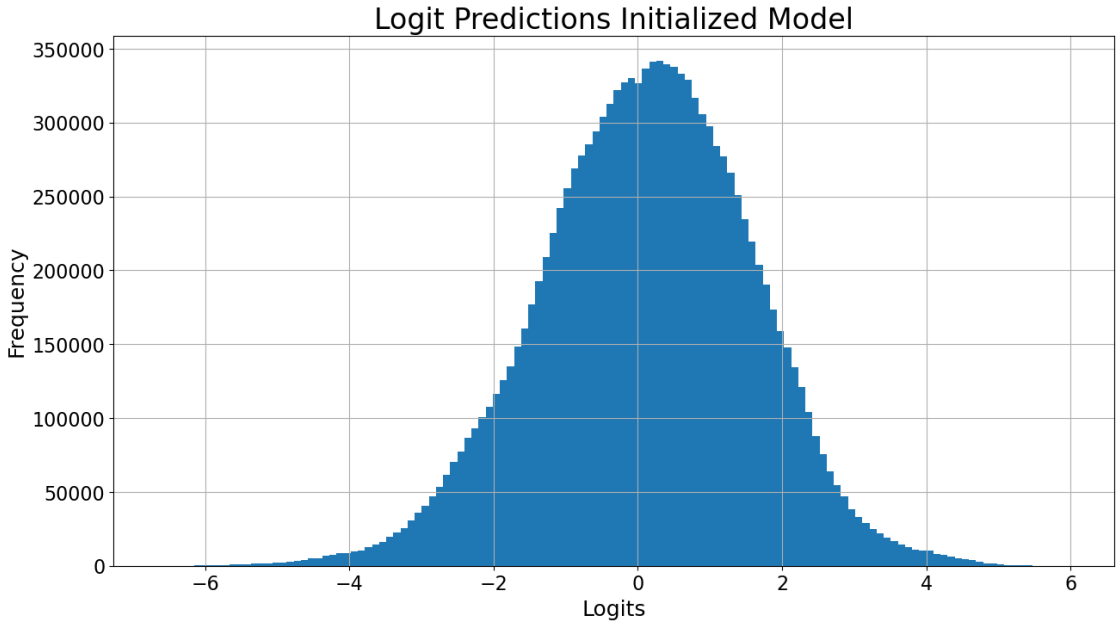
[39]:
```
def verify_correct_training_flag():
    # Verify static output for inference
    pred = model(X_batch_small, training=False)
    for _ in tqdm(range(10)):
        assert tf.reduce_min(tf.cast(pred == model(X_batch_small,␣
 ↪training=False), tf.int8)) == 1

    # Verify at least 99% varying output due to dropout during training
    for _ in tqdm(range(10)):
        assert tf.reduce_mean(tf.cast(pred != model(X_batch_small,␣
 ↪training=True), tf.float32)) > 0.99

verify_correct_training_flag()
```

```
  0%|          | 0/10 [00:00<?, ?it/s]
  0%|          | 0/10 [00:00<?, ?it/s]
```

26 Verify No NaN Predictions

[40]:
```
# Verify No NaN predictions
def verify_no_nan_predictions():
    y_pred = model.predict(
        val_dataset if USE_VAL else train_dataset,
        steps=N_VAL_STEPS_PER_EPOCH if USE_VAL else 100,
        verbose=VERBOSE,
    )

    print(f'# NaN Values In Predictions: {np.isnan(y_pred).sum()}')

    plt.figure(figsize=(15,8))
    plt.title(f'Logit Predictions Initialized Model')
    pd.Series(y_pred.flatten()).plot(kind='hist', bins=128)
    plt.xlabel('Logits')
    plt.grid()
    plt.show()

verify_no_nan_predictions()
```

```
100/100 - 7s - 7s/epoch - 67ms/step
# NaN Values In Predictions: 0
```

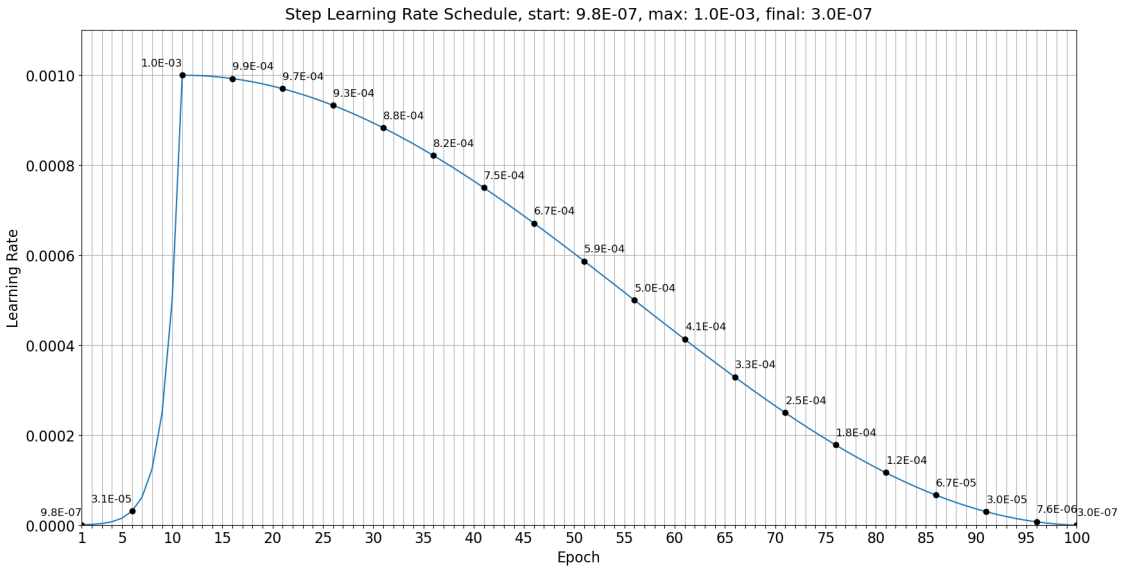


27 Learning Rate Scheduler

[41]:
```
def lrfn(current_step, num_warmup_steps, lr_max, num_cycles=0.50,␣
↪num_training_steps=N_EPOCHS):

    if current_step < num_warmup_steps:
        if WARMUP_METHOD == 'log':
            return lr_max * 0.10 ** (num_warmup_steps - current_step)
        else:
            return lr_max * 2 ** -(num_warmup_steps - current_step)
    else:
        progress = float(current_step - num_warmup_steps) / float(max(1,␣
↪num_training_steps - num_warmup_steps))

        return max(0.0, 0.5 * (1.0 + math.cos(math.pi * float(num_cycles) * 2.0␣
↪* progress))) * lr_max
```

[42]:
```
def plot_lr_schedule(lr_schedule, epochs):
    fig = plt.figure(figsize=(20, 10))
    plt.plot([None] + lr_schedule + [None])
    # X Labels
    x = np.arange(1, epochs + 1)
    x_axis_labels = [i if epochs <= 40 or i % 5 == 0 or i == 1 else None for i␣
↪in range(1, epochs + 1)]
    plt.xlim([1, epochs])
```

```
    plt.xticks(x, x_axis_labels) # set tick step to 1 and let x axis start at 1

    # Increase y-limit for better readability
    plt.ylim([0, max(lr_schedule) * 1.1])

    # Title
    schedule_info = f'start: {lr_schedule[0]:.1E}, max: {max(lr_schedule):.1E},␣
 ↪final: {lr_schedule[-1]:.1E}'
    plt.title(f'Step Learning Rate Schedule, {schedule_info}', size=18, pad=12)

    # Plot Learning Rates
    for x, val in enumerate(lr_schedule):
        if epochs <= 40 or x % 5 == 0 or x is epochs - 1:
            if x < len(lr_schedule) - 1:
                if lr_schedule[x - 1] < val:
                    ha = 'right'
                else:
                    ha = 'left'
            elif x == 0:
                ha = 'right'
            else:
                ha = 'left'
            plt.plot(x + 1, val, 'o', color='black');
            offset_y = (max(lr_schedule) - min(lr_schedule)) * 0.02
            plt.annotate(f'{val:.1E}', xy=(x + 1, val + offset_y), size=12,␣
 ↪ha=ha)

    plt.xlabel('Epoch', size=16, labelpad=5)
    plt.ylabel('Learning Rate', size=16, labelpad=5)
    plt.grid()
    plt.show()

# Learning rate for encoder
LR_SCHEDULE = [lrfn(step, num_warmup_steps=N_WARMUP_EPOCHS, lr_max=LR_MAX,␣
 ↪num_cycles=0.50) for step in range(N_EPOCHS)]
# Plot Learning Rate Schedule
plot_lr_schedule(LR_SCHEDULE, epochs=N_EPOCHS)
# Learning Rate Callback
lr_callback = tf.keras.callbacks.LearningRateScheduler(lambda step:␣
 ↪LR_SCHEDULE[step], verbose=0)
```



28 Weight Decay Callback

[43]:
```
# Custom callback to update weight decay with learning rate
class WeightDecayCallback(tf.keras.callbacks.Callback):
    def __init__(self, wd_ratio=WD_RATIO):
        self.step_counter = 0
        self.wd_ratio = wd_ratio

    def on_epoch_begin(self, epoch, logs=None):
        model.optimizer.weight_decay = model.optimizer.learning_rate * self.
 ↪wd_ratio
        print(f'learning rate: {model.optimizer.learning_rate.numpy():.2e},␣
 ↪weight decay: {model.optimizer.weight_decay.numpy():.2e}')
```

29 Evaluate Initialized Model

[44]:
```
# Evaluate Initialized Model On Validation Data
y_pred = model.evaluate(
    val_dataset if USE_VAL else train_dataset,
    steps=N_VAL_STEPS_PER_EPOCH if USE_VAL else TRAIN_STEPS_PER_EPOCH,
    verbose=VERBOSE,
)
```

```
969/969 - 41s - loss: 5.2113 - top1acc: 0.0056 - top5acc: 0.0540 - 41s/epoch -
43ms/step
```

30 Baseline

[45]:
```
# baseline accuracy when only pad token is predicted
if USE_VAL:
    baseline_accuracy = np.mean(y_val == PAD_TOKEN)
else:
    baseline_accuracy = np.mean(y_train == PAD_TOKEN)
print(f'Baseline Accuracy: {baseline_accuracy:.4f}')
```

```
Baseline Accuracy: 0.4100
```

31 Train

[46]:
```
gc.collect()
```

[46]:
```
25963
```

[47]:
```
if TRAIN_MODEL:
    # Clear all models in GPU
    tf.keras.backend.clear_session()

    # Get new fresh model
    model = get_model()

    # Sanity Check
    model.summary()

    # Actual Training
    history = model.fit(
            x=train_dataset,
            steps_per_epoch=TRAIN_STEPS_PER_EPOCH,
            epochs=N_EPOCHS,
            # Only used for validation data since training data is a generator
            validation_data=val_dataset if USE_VAL else None,
            validation_steps=N_VAL_STEPS_PER_EPOCH if USE_VAL else None,
            callbacks=[
                lr_callback,
                WeightDecayCallback(),
            ],
            verbose = VERBOSE,
        )
```

```
Model: "model"
__________________________________________________________________________________
__________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================
```

```
==================
 frames (InputLayer)            [(None, 128, 164)]   0           []

 masking (Masking)              (None, 128, 164)     0
['frames[0][0]']

 embedding (Embedding)          (None, 128, 384)     259968
['masking[0][0]']

 encoder (Encoder)              (None, 128, 256)     2757120
['embedding[0][0]',
'frames[0][0]']

 phrase (InputLayer)            [(None, 32)]         0           []

 decoder (Decoder)              (None, 32, 256)      968704
['encoder[0][0]',
'phrase[0][0]']

 classifier (Sequential)        (None, 32, 62)       15872
['decoder[0][0]']

================================================================================
==================
Total params: 4,001,664
Trainable params: 4,001,664
Non-trainable params: 0
--------------------------------------------------------------------------------
------------------
learning rate: 9.77e-07, weight decay: 4.88e-08
Epoch 1/100
969/969 - 162s - loss: 4.6122 - top1acc: 0.0361 - top5acc: 0.1450 - lr:
9.7656e-07 - 162s/epoch - 167ms/step
learning rate: 1.95e-06, weight decay: 9.77e-08
Epoch 2/100
969/969 - 122s - loss: 4.0207 - top1acc: 0.0896 - top5acc: 0.2885 - lr:
1.9531e-06 - 122s/epoch - 126ms/step
learning rate: 3.91e-06, weight decay: 1.95e-07
Epoch 3/100
969/969 - 122s - loss: 3.6777 - top1acc: 0.1410 - top5acc: 0.4200 - lr:
3.9063e-06 - 122s/epoch - 126ms/step
learning rate: 7.81e-06, weight decay: 3.91e-07
Epoch 4/100
969/969 - 122s - loss: 3.4956 - top1acc: 0.1658 - top5acc: 0.4813 - lr:
7.8125e-06 - 122s/epoch - 126ms/step
learning rate: 1.56e-05, weight decay: 7.81e-07
Epoch 5/100
969/969 - 122s - loss: 3.3993 - top1acc: 0.1827 - top5acc: 0.5097 - lr:
```

```
1.5625e-05 - 122s/epoch - 126ms/step
learning rate: 3.13e-05, weight decay: 1.56e-06
Epoch 6/100
969/969 - 122s - loss: 3.3298 - top1acc: 0.1978 - top5acc: 0.5319 - lr:
3.1250e-05 - 122s/epoch - 126ms/step
learning rate: 6.25e-05, weight decay: 3.13e-06
Epoch 7/100
969/969 - 122s - loss: 3.2676 - top1acc: 0.2132 - top5acc: 0.5548 - lr:
6.2500e-05 - 122s/epoch - 126ms/step
learning rate: 1.25e-04, weight decay: 6.25e-06
Epoch 8/100
969/969 - 122s - loss: 3.1972 - top1acc: 0.2355 - top5acc: 0.5893 - lr:
1.2500e-04 - 122s/epoch - 126ms/step
learning rate: 2.50e-04, weight decay: 1.25e-05
Epoch 9/100
969/969 - 122s - loss: 3.1032 - top1acc: 0.2700 - top5acc: 0.6449 - lr:
2.5000e-04 - 122s/epoch - 126ms/step
learning rate: 5.00e-04, weight decay: 2.50e-05
Epoch 10/100
969/969 - 122s - loss: 2.9655 - top1acc: 0.3349 - top5acc: 0.7153 - lr:
5.0000e-04 - 122s/epoch - 126ms/step
learning rate: 1.00e-03, weight decay: 5.00e-05
Epoch 11/100
969/969 - 122s - loss: 2.8061 - top1acc: 0.4148 - top5acc: 0.7753 - lr: 0.0010 -
122s/epoch - 126ms/step
learning rate: 1.00e-03, weight decay: 5.00e-05
Epoch 12/100
969/969 - 122s - loss: 2.6406 - top1acc: 0.4967 - top5acc: 0.8215 - lr:
9.9970e-04 - 122s/epoch - 126ms/step
learning rate: 9.99e-04, weight decay: 4.99e-05
Epoch 13/100
969/969 - 122s - loss: 2.5450 - top1acc: 0.5440 - top5acc: 0.8424 - lr:
9.9878e-04 - 122s/epoch - 126ms/step
learning rate: 9.97e-04, weight decay: 4.99e-05
Epoch 14/100
969/969 - 122s - loss: 2.4808 - top1acc: 0.5747 - top5acc: 0.8553 - lr:
9.9726e-04 - 122s/epoch - 126ms/step
learning rate: 9.95e-04, weight decay: 4.98e-05
Epoch 15/100
969/969 - 122s - loss: 2.4287 - top1acc: 0.5987 - top5acc: 0.8650 - lr:
9.9513e-04 - 122s/epoch - 126ms/step
learning rate: 9.92e-04, weight decay: 4.96e-05
Epoch 16/100
969/969 - 122s - loss: 2.3917 - top1acc: 0.6167 - top5acc: 0.8707 - lr:
9.9240e-04 - 122s/epoch - 126ms/step
learning rate: 9.89e-04, weight decay: 4.95e-05
Epoch 17/100
969/969 - 122s - loss: 2.3610 - top1acc: 0.6308 - top5acc: 0.8759 - lr:
```

```
9.8907e-04 - 122s/epoch - 126ms/step
learning rate: 9.85e-04, weight decay: 4.93e-05
Epoch 18/100
969/969 - 122s - loss: 2.3355 - top1acc: 0.6420 - top5acc: 0.8802 - lr:
9.8515e-04 - 122s/epoch - 126ms/step
learning rate: 9.81e-04, weight decay: 4.90e-05
Epoch 19/100
969/969 - 122s - loss: 2.3175 - top1acc: 0.6501 - top5acc: 0.8829 - lr:
9.8063e-04 - 122s/epoch - 126ms/step
learning rate: 9.76e-04, weight decay: 4.88e-05
Epoch 20/100
969/969 - 122s - loss: 2.2995 - top1acc: 0.6581 - top5acc: 0.8864 - lr:
9.7553e-04 - 122s/epoch - 126ms/step
learning rate: 9.70e-04, weight decay: 4.85e-05
Epoch 21/100
969/969 - 122s - loss: 2.2807 - top1acc: 0.6671 - top5acc: 0.8897 - lr:
9.6985e-04 - 122s/epoch - 126ms/step
learning rate: 9.64e-04, weight decay: 4.82e-05
Epoch 22/100
969/969 - 122s - loss: 2.2646 - top1acc: 0.6745 - top5acc: 0.8928 - lr:
9.6359e-04 - 122s/epoch - 126ms/step
learning rate: 9.57e-04, weight decay: 4.78e-05
Epoch 23/100
969/969 - 122s - loss: 2.2579 - top1acc: 0.6769 - top5acc: 0.8932 - lr:
9.5677e-04 - 122s/epoch - 126ms/step
learning rate: 9.49e-04, weight decay: 4.75e-05
Epoch 24/100
969/969 - 122s - loss: 2.2454 - top1acc: 0.6825 - top5acc: 0.8952 - lr:
9.4940e-04 - 122s/epoch - 126ms/step
learning rate: 9.41e-04, weight decay: 4.71e-05
Epoch 25/100
969/969 - 122s - loss: 2.2367 - top1acc: 0.6871 - top5acc: 0.8969 - lr:
9.4147e-04 - 122s/epoch - 125ms/step
learning rate: 9.33e-04, weight decay: 4.67e-05
Epoch 26/100
969/969 - 122s - loss: 2.2255 - top1acc: 0.6918 - top5acc: 0.8989 - lr:
9.3301e-04 - 122s/epoch - 125ms/step
learning rate: 9.24e-04, weight decay: 4.62e-05
Epoch 27/100
969/969 - 121s - loss: 2.2125 - top1acc: 0.6976 - top5acc: 0.9014 - lr:
9.2402e-04 - 121s/epoch - 125ms/step
learning rate: 9.15e-04, weight decay: 4.57e-05
Epoch 28/100
969/969 - 122s - loss: 2.2068 - top1acc: 0.7006 - top5acc: 0.9022 - lr:
9.1452e-04 - 122s/epoch - 125ms/step
learning rate: 9.05e-04, weight decay: 4.52e-05
Epoch 29/100
969/969 - 121s - loss: 2.1960 - top1acc: 0.7052 - top5acc: 0.9042 - lr:
```

```
9.0451e-04 - 121s/epoch - 125ms/step
learning rate: 8.94e-04, weight decay: 4.47e-05
Epoch 30/100
969/969 - 122s - loss: 2.1914 - top1acc: 0.7075 - top5acc: 0.9039 - lr:
8.9401e-04 - 122s/epoch - 126ms/step
learning rate: 8.83e-04, weight decay: 4.42e-05
Epoch 31/100
969/969 - 122s - loss: 2.1787 - top1acc: 0.7135 - top5acc: 0.9071 - lr:
8.8302e-04 - 122s/epoch - 125ms/step
learning rate: 8.72e-04, weight decay: 4.36e-05
Epoch 32/100
969/969 - 122s - loss: 2.1743 - top1acc: 0.7150 - top5acc: 0.9072 - lr:
8.7157e-04 - 122s/epoch - 125ms/step
learning rate: 8.60e-04, weight decay: 4.30e-05
Epoch 33/100
969/969 - 122s - loss: 2.1678 - top1acc: 0.7185 - top5acc: 0.9082 - lr:
8.5967e-04 - 122s/epoch - 125ms/step
learning rate: 8.47e-04, weight decay: 4.24e-05
Epoch 34/100
969/969 - 121s - loss: 2.1615 - top1acc: 0.7213 - top5acc: 0.9089 - lr:
8.4733e-04 - 121s/epoch - 125ms/step
learning rate: 8.35e-04, weight decay: 4.17e-05
Epoch 35/100
969/969 - 122s - loss: 2.1588 - top1acc: 0.7221 - top5acc: 0.9097 - lr:
8.3457e-04 - 122s/epoch - 125ms/step
learning rate: 8.21e-04, weight decay: 4.11e-05
Epoch 36/100
969/969 - 121s - loss: 2.1473 - top1acc: 0.7277 - top5acc: 0.9116 - lr:
8.2139e-04 - 121s/epoch - 125ms/step
learning rate: 8.08e-04, weight decay: 4.04e-05
Epoch 37/100
969/969 - 121s - loss: 2.1449 - top1acc: 0.7288 - top5acc: 0.9120 - lr:
8.0783e-04 - 121s/epoch - 125ms/step
learning rate: 7.94e-04, weight decay: 3.97e-05
Epoch 38/100
969/969 - 121s - loss: 2.1367 - top1acc: 0.7324 - top5acc: 0.9135 - lr:
7.9389e-04 - 121s/epoch - 125ms/step
learning rate: 7.80e-04, weight decay: 3.90e-05
Epoch 39/100
969/969 - 121s - loss: 2.1328 - top1acc: 0.7342 - top5acc: 0.9144 - lr:
7.7960e-04 - 121s/epoch - 125ms/step
learning rate: 7.65e-04, weight decay: 3.82e-05
Epoch 40/100
969/969 - 122s - loss: 2.1275 - top1acc: 0.7365 - top5acc: 0.9152 - lr:
7.6496e-04 - 122s/epoch - 125ms/step
learning rate: 7.50e-04, weight decay: 3.75e-05
Epoch 41/100
969/969 - 121s - loss: 2.1243 - top1acc: 0.7383 - top5acc: 0.9157 - lr:
```

```
7.5000e-04 - 121s/epoch - 125ms/step
learning rate: 7.35e-04, weight decay: 3.67e-05
Epoch 42/100
969/969 - 122s - loss: 2.1191 - top1acc: 0.7403 - top5acc: 0.9166 - lr:
7.3474e-04 - 122s/epoch - 125ms/step
learning rate: 7.19e-04, weight decay: 3.60e-05
Epoch 43/100
969/969 - 122s - loss: 2.1122 - top1acc: 0.7435 - top5acc: 0.9180 - lr:
7.1919e-04 - 122s/epoch - 125ms/step
learning rate: 7.03e-04, weight decay: 3.52e-05
Epoch 44/100
969/969 - 121s - loss: 2.1114 - top1acc: 0.7440 - top5acc: 0.9174 - lr:
7.0337e-04 - 121s/epoch - 125ms/step
learning rate: 6.87e-04, weight decay: 3.44e-05
Epoch 45/100
969/969 - 121s - loss: 2.1056 - top1acc: 0.7463 - top5acc: 0.9192 - lr:
6.8730e-04 - 121s/epoch - 125ms/step
learning rate: 6.71e-04, weight decay: 3.36e-05
Epoch 46/100
969/969 - 122s - loss: 2.0996 - top1acc: 0.7496 - top5acc: 0.9196 - lr:
6.7101e-04 - 122s/epoch - 126ms/step
learning rate: 6.55e-04, weight decay: 3.27e-05
Epoch 47/100
969/969 - 122s - loss: 2.0943 - top1acc: 0.7517 - top5acc: 0.9206 - lr:
6.5451e-04 - 122s/epoch - 125ms/step
learning rate: 6.38e-04, weight decay: 3.19e-05
Epoch 48/100
969/969 - 121s - loss: 2.0912 - top1acc: 0.7529 - top5acc: 0.9213 - lr:
6.3782e-04 - 121s/epoch - 125ms/step
learning rate: 6.21e-04, weight decay: 3.10e-05
Epoch 49/100
969/969 - 121s - loss: 2.0864 - top1acc: 0.7549 - top5acc: 0.9215 - lr:
6.2096e-04 - 121s/epoch - 125ms/step
learning rate: 6.04e-04, weight decay: 3.02e-05
Epoch 50/100
969/969 - 121s - loss: 2.0854 - top1acc: 0.7560 - top5acc: 0.9220 - lr:
6.0396e-04 - 121s/epoch - 125ms/step
learning rate: 5.87e-04, weight decay: 2.93e-05
Epoch 51/100
969/969 - 121s - loss: 2.0793 - top1acc: 0.7582 - top5acc: 0.9235 - lr:
5.8682e-04 - 121s/epoch - 125ms/step
learning rate: 5.70e-04, weight decay: 2.85e-05
Epoch 52/100
969/969 - 121s - loss: 2.0775 - top1acc: 0.7590 - top5acc: 0.9233 - lr:
5.6959e-04 - 121s/epoch - 125ms/step
learning rate: 5.52e-04, weight decay: 2.76e-05
Epoch 53/100
969/969 - 121s - loss: 2.0662 - top1acc: 0.7643 - top5acc: 0.9258 - lr:
```

```
5.5226e-04 - 121s/epoch - 125ms/step
learning rate: 5.35e-04, weight decay: 2.67e-05
Epoch 54/100
969/969 - 121s - loss: 2.0622 - top1acc: 0.7657 - top5acc: 0.9263 - lr:
5.3488e-04 - 121s/epoch - 125ms/step
learning rate: 5.17e-04, weight decay: 2.59e-05
Epoch 55/100
969/969 - 121s - loss: 2.0592 - top1acc: 0.7674 - top5acc: 0.9268 - lr:
5.1745e-04 - 121s/epoch - 125ms/step
learning rate: 5.00e-04, weight decay: 2.50e-05
Epoch 56/100
969/969 - 121s - loss: 2.0576 - top1acc: 0.7682 - top5acc: 0.9267 - lr:
5.0000e-04 - 121s/epoch - 125ms/step
learning rate: 4.83e-04, weight decay: 2.41e-05
Epoch 57/100
969/969 - 121s - loss: 2.0574 - top1acc: 0.7680 - top5acc: 0.9269 - lr:
4.8255e-04 - 121s/epoch - 125ms/step
learning rate: 4.65e-04, weight decay: 2.33e-05
Epoch 58/100
969/969 - 122s - loss: 2.0485 - top1acc: 0.7721 - top5acc: 0.9284 - lr:
4.6512e-04 - 122s/epoch - 125ms/step
learning rate: 4.48e-04, weight decay: 2.24e-05
Epoch 59/100
969/969 - 122s - loss: 2.0460 - top1acc: 0.7733 - top5acc: 0.9280 - lr:
4.4774e-04 - 122s/epoch - 125ms/step
learning rate: 4.30e-04, weight decay: 2.15e-05
Epoch 60/100
969/969 - 121s - loss: 2.0406 - top1acc: 0.7759 - top5acc: 0.9296 - lr:
4.3041e-04 - 121s/epoch - 125ms/step
learning rate: 4.13e-04, weight decay: 2.07e-05
Epoch 61/100
969/969 - 121s - loss: 2.0339 - top1acc: 0.7784 - top5acc: 0.9309 - lr:
4.1318e-04 - 121s/epoch - 125ms/step
learning rate: 3.96e-04, weight decay: 1.98e-05
Epoch 62/100
969/969 - 121s - loss: 2.0326 - top1acc: 0.7794 - top5acc: 0.9310 - lr:
3.9604e-04 - 121s/epoch - 125ms/step
learning rate: 3.79e-04, weight decay: 1.90e-05
Epoch 63/100
969/969 - 121s - loss: 2.0288 - top1acc: 0.7811 - top5acc: 0.9317 - lr:
3.7904e-04 - 121s/epoch - 125ms/step
learning rate: 3.62e-04, weight decay: 1.81e-05
Epoch 64/100
969/969 - 121s - loss: 2.0246 - top1acc: 0.7830 - top5acc: 0.9322 - lr:
3.6218e-04 - 121s/epoch - 125ms/step
learning rate: 3.45e-04, weight decay: 1.73e-05
Epoch 65/100
969/969 - 122s - loss: 2.0202 - top1acc: 0.7852 - top5acc: 0.9332 - lr:
```

```
3.4549e-04 - 122s/epoch - 125ms/step
learning rate: 3.29e-04, weight decay: 1.64e-05
Epoch 66/100
969/969 - 121s - loss: 2.0178 - top1acc: 0.7861 - top5acc: 0.9335 - lr:
3.2899e-04 - 121s/epoch - 125ms/step
learning rate: 3.13e-04, weight decay: 1.56e-05
Epoch 67/100
969/969 - 121s - loss: 2.0151 - top1acc: 0.7878 - top5acc: 0.9340 - lr:
3.1270e-04 - 121s/epoch - 125ms/step
learning rate: 2.97e-04, weight decay: 1.48e-05
Epoch 68/100
969/969 - 121s - loss: 2.0110 - top1acc: 0.7893 - top5acc: 0.9346 - lr:
2.9663e-04 - 121s/epoch - 125ms/step
learning rate: 2.81e-04, weight decay: 1.40e-05
Epoch 69/100
969/969 - 122s - loss: 2.0084 - top1acc: 0.7905 - top5acc: 0.9348 - lr:
2.8081e-04 - 122s/epoch - 126ms/step
learning rate: 2.65e-04, weight decay: 1.33e-05
Epoch 70/100
969/969 - 121s - loss: 2.0054 - top1acc: 0.7921 - top5acc: 0.9353 - lr:
2.6526e-04 - 121s/epoch - 125ms/step
learning rate: 2.50e-04, weight decay: 1.25e-05
Epoch 71/100
969/969 - 121s - loss: 2.0022 - top1acc: 0.7935 - top5acc: 0.9361 - lr:
2.5000e-04 - 121s/epoch - 125ms/step
learning rate: 2.35e-04, weight decay: 1.18e-05
Epoch 72/100
969/969 - 121s - loss: 1.9969 - top1acc: 0.7956 - top5acc: 0.9369 - lr:
2.3504e-04 - 121s/epoch - 125ms/step
learning rate: 2.20e-04, weight decay: 1.10e-05
Epoch 73/100
969/969 - 121s - loss: 1.9977 - top1acc: 0.7953 - top5acc: 0.9367 - lr:
2.2040e-04 - 121s/epoch - 125ms/step
learning rate: 2.06e-04, weight decay: 1.03e-05
Epoch 74/100
969/969 - 121s - loss: 1.9919 - top1acc: 0.7980 - top5acc: 0.9376 - lr:
2.0611e-04 - 121s/epoch - 125ms/step
learning rate: 1.92e-04, weight decay: 9.61e-06
Epoch 75/100
969/969 - 121s - loss: 1.9895 - top1acc: 0.7993 - top5acc: 0.9380 - lr:
1.9217e-04 - 121s/epoch - 125ms/step
learning rate: 1.79e-04, weight decay: 8.93e-06
Epoch 76/100
969/969 - 121s - loss: 1.9856 - top1acc: 0.8010 - top5acc: 0.9389 - lr:
1.7861e-04 - 121s/epoch - 125ms/step
learning rate: 1.65e-04, weight decay: 8.27e-06
Epoch 77/100
969/969 - 122s - loss: 1.9875 - top1acc: 0.8000 - top5acc: 0.9383 - lr:
```

```
1.6543e-04 - 122s/epoch - 125ms/step
learning rate: 1.53e-04, weight decay: 7.63e-06
Epoch 78/100
969/969 - 121s - loss: 1.9811 - top1acc: 0.8033 - top5acc: 0.9390 - lr:
1.5267e-04 - 121s/epoch - 125ms/step
learning rate: 1.40e-04, weight decay: 7.02e-06
Epoch 79/100
969/969 - 121s - loss: 1.9761 - top1acc: 0.8056 - top5acc: 0.9401 - lr:
1.4033e-04 - 121s/epoch - 125ms/step
learning rate: 1.28e-04, weight decay: 6.42e-06
Epoch 80/100
969/969 - 122s - loss: 1.9772 - top1acc: 0.8052 - top5acc: 0.9399 - lr:
1.2843e-04 - 122s/epoch - 126ms/step
learning rate: 1.17e-04, weight decay: 5.85e-06
Epoch 81/100
969/969 - 122s - loss: 1.9758 - top1acc: 0.8056 - top5acc: 0.9397 - lr:
1.1698e-04 - 122s/epoch - 126ms/step
learning rate: 1.06e-04, weight decay: 5.30e-06
Epoch 82/100
969/969 - 121s - loss: 1.9718 - top1acc: 0.8076 - top5acc: 0.9410 - lr:
1.0599e-04 - 121s/epoch - 125ms/step
learning rate: 9.55e-05, weight decay: 4.77e-06
Epoch 83/100
969/969 - 121s - loss: 1.9726 - top1acc: 0.8068 - top5acc: 0.9403 - lr:
9.5492e-05 - 121s/epoch - 125ms/step
learning rate: 8.55e-05, weight decay: 4.27e-06
Epoch 84/100
969/969 - 121s - loss: 1.9699 - top1acc: 0.8079 - top5acc: 0.9411 - lr:
8.5481e-05 - 121s/epoch - 125ms/step
learning rate: 7.60e-05, weight decay: 3.80e-06
Epoch 85/100
969/969 - 121s - loss: 1.9629 - top1acc: 0.8112 - top5acc: 0.9423 - lr:
7.5976e-05 - 121s/epoch - 125ms/step
learning rate: 6.70e-05, weight decay: 3.35e-06
Epoch 86/100
969/969 - 121s - loss: 1.9621 - top1acc: 0.8120 - top5acc: 0.9421 - lr:
6.6987e-05 - 121s/epoch - 125ms/step
learning rate: 5.85e-05, weight decay: 2.93e-06
Epoch 87/100
969/969 - 122s - loss: 1.9612 - top1acc: 0.8123 - top5acc: 0.9422 - lr:
5.8526e-05 - 122s/epoch - 125ms/step
learning rate: 5.06e-05, weight decay: 2.53e-06
Epoch 88/100
969/969 - 122s - loss: 1.9624 - top1acc: 0.8122 - top5acc: 0.9421 - lr:
5.0603e-05 - 122s/epoch - 126ms/step
learning rate: 4.32e-05, weight decay: 2.16e-06
Epoch 89/100
969/969 - 121s - loss: 1.9623 - top1acc: 0.8119 - top5acc: 0.9423 - lr:
```

```
4.3227e-05 - 121s/epoch - 125ms/step
learning rate: 3.64e-05, weight decay: 1.82e-06
Epoch 90/100
969/969 - 121s - loss: 1.9598 - top1acc: 0.8131 - top5acc: 0.9426 - lr:
3.6408e-05 - 121s/epoch - 125ms/step
learning rate: 3.02e-05, weight decay: 1.51e-06
Epoch 91/100
969/969 - 121s - loss: 1.9591 - top1acc: 0.8136 - top5acc: 0.9429 - lr:
3.0154e-05 - 121s/epoch - 125ms/step
learning rate: 2.45e-05, weight decay: 1.22e-06
Epoch 92/100
969/969 - 122s - loss: 1.9619 - top1acc: 0.8122 - top5acc: 0.9421 - lr:
2.4472e-05 - 122s/epoch - 126ms/step
learning rate: 1.94e-05, weight decay: 9.68e-07
Epoch 93/100
969/969 - 121s - loss: 1.9598 - top1acc: 0.8131 - top5acc: 0.9422 - lr:
1.9369e-05 - 121s/epoch - 125ms/step
learning rate: 1.49e-05, weight decay: 7.43e-07
Epoch 94/100
969/969 - 121s - loss: 1.9587 - top1acc: 0.8134 - top5acc: 0.9426 - lr:
1.4852e-05 - 121s/epoch - 125ms/step
learning rate: 1.09e-05, weight decay: 5.46e-07
Epoch 95/100
969/969 - 121s - loss: 1.9585 - top1acc: 0.8136 - top5acc: 0.9425 - lr:
1.0926e-05 - 121s/epoch - 125ms/step
learning rate: 7.60e-06, weight decay: 3.80e-07
Epoch 96/100
969/969 - 122s - loss: 1.9566 - top1acc: 0.8147 - top5acc: 0.9427 - lr:
7.5961e-06 - 122s/epoch - 125ms/step
learning rate: 4.87e-06, weight decay: 2.43e-07
Epoch 97/100
969/969 - 122s - loss: 1.9557 - top1acc: 0.8151 - top5acc: 0.9430 - lr:
4.8660e-06 - 122s/epoch - 126ms/step
learning rate: 2.74e-06, weight decay: 1.37e-07
Epoch 98/100
969/969 - 121s - loss: 1.9566 - top1acc: 0.8143 - top5acc: 0.9427 - lr:
2.7391e-06 - 121s/epoch - 125ms/step
learning rate: 1.22e-06, weight decay: 6.09e-08
Epoch 99/100
969/969 - 121s - loss: 1.9548 - top1acc: 0.8153 - top5acc: 0.9432 - lr:
1.2180e-06 - 121s/epoch - 125ms/step
learning rate: 3.05e-07, weight decay: 1.52e-08
Epoch 100/100
969/969 - 121s - loss: 1.9529 - top1acc: 0.8162 - top5acc: 0.9439 - lr:
3.0459e-07 - 121s/epoch - 125ms/step
```

[48]:
```
# Load Weights
if LOAD_WEIGHTS:
    model.load_weights('/kaggle/input/aslfr-training-python37/model.h5')
    print(f'Successfully Loaded Pretrained Weights')
```

[49]:
```
# Save Model Weights
model.save_weights('model.h5')
```

[50]:
```
# Verify Model is Loaded Correctly
model.evaluate(
    val_dataset if USE_VAL else train_dataset,
    steps=N_VAL_STEPS_PER_EPOCH if USE_VAL else TRAIN_STEPS_PER_EPOCH,
    batch_size=BATCH_SIZE,
    verbose=VERBOSE,
)
```

```
969/969 - 41s - loss: 1.8608 - top1acc: 0.8571 - top5acc: 0.9561 - 41s/epoch -
43ms/step
```

[50]:
```
[1.8608417510986328, 0.8570798635482788, 0.9560850262641907]
```

32 Levenshtein Distance

[51]:
```
# Output Predictions to string
def outputs2phrase(outputs):
    if outputs.ndim == 2:
        outputs = np.argmax(outputs, axis=1)

    return ''.join([ORD2CHAR.get(s, '') for s in outputs])
```

[52]:
```
@tf.function()
def predict_phrase(frames):
    # Add Batch Dimension
    frames = tf.expand_dims(frames, axis=0)
    # Start Phrase
    phrase = tf.fill([1,MAX_PHRASE_LENGTH], PAD_TOKEN)

    for idx in tf.range(MAX_PHRASE_LENGTH):
        # Cast phrase to int8
        phrase = tf.cast(phrase, tf.int8)
        # Predict Next Token
        outputs = model({
            'frames': frames,
            'phrase': phrase,
        })
```

```
        # Add predicted token to input phrase
        phrase = tf.cast(phrase, tf.int32)
        phrase = tf.where(
            tf.range(MAX_PHRASE_LENGTH) < idx + 1,
            tf.argmax(outputs, axis=2, output_type=tf.int32),
            phrase,
        )

    # Squeeze outputs
    outputs = tf.squeeze(phrase, axis=0)
    outputs = tf.one_hot(outputs, N_UNIQUE_CHARACTERS)

    # Return a dictionary with the output tensor
    return outputs

    # Return a dictionary with the output tensor
    return outputs
```

33 Levenstein Distance Train

[53]:
```
# Compute Levenstein Distances
def get_ld_train():
    N = 100 if IS_INTERACTIVE else 1000
    LD_TRAIN = []
    for idx, (frames, phrase_true) in enumerate(zip(tqdm(X_train, total=N),␣
 ↪y_train)):
        # Predict Phrase and Convert to String
        phrase_pred = predict_phrase(frames).numpy()
        phrase_pred = outputs2phrase(phrase_pred)
        # True Phrase Ordinal to String
        phrase_true = outputs2phrase(phrase_true)
        # Add Levenstein Distance
        LD_TRAIN.append({
            'phrase_true': phrase_true,
            'phrase_pred': phrase_pred,
            'levenshtein_distance': levenshtein(phrase_pred, phrase_true),
        })
        # Take subset in interactive mode
        if idx == N:
            break

    # Convert to DataFrame
    LD_TRAIN_DF = pd.DataFrame(LD_TRAIN)

    return LD_TRAIN_DF
```

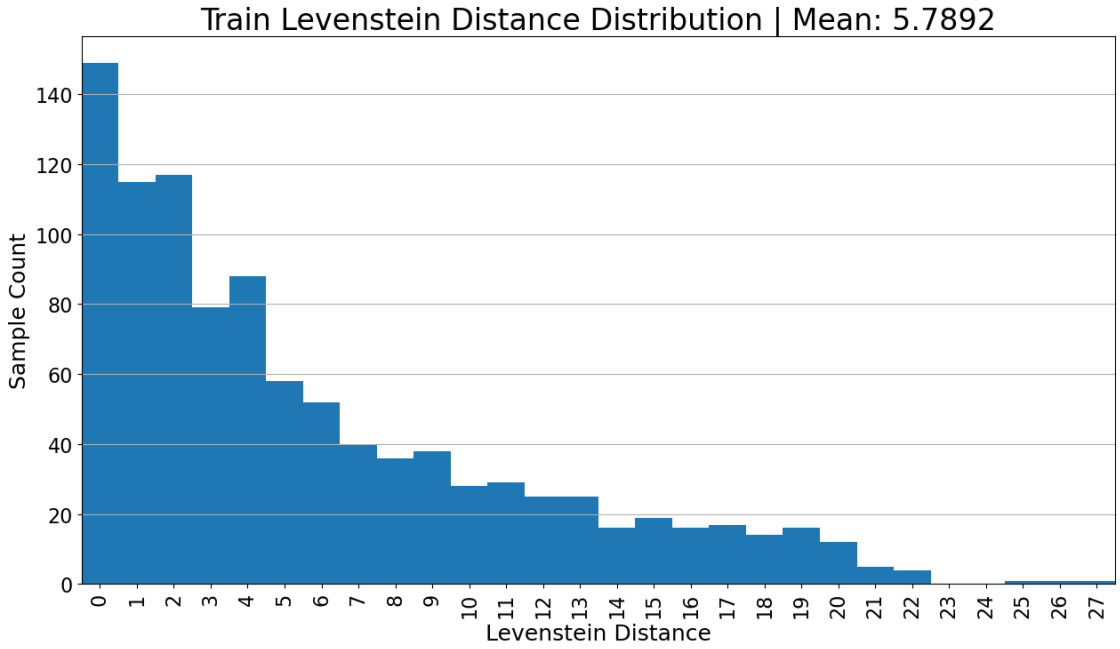
[54]:
```
LD_TRAIN_DF = get_ld_train()

# Display Errors
display(LD_TRAIN_DF.head(30))
```

```
  0%|          | 0/1000 [00:00<?, ?it/s]
```

	phrase_true	phrase_pred	levenshtein_distance
0	3 creekhouse	3 creek housese.de	6
1	scales/kuhaylah	scales/kuhaylaaa	2
2	hentaihubs.com	hentaihubs.com.com	4
3	1383 william lanier	1383 william lanier	0
4	988 franklin lane	988 franklin lane lanee	6
5	6920 northeast 661st road	6920 northeast 661st roadd	1
6	www.freem.ne.jp	www.freem.me.jp.jp	4
7	https://jsi.is/hukuoka	https://jsi.is/htkuoka	1
8	239613 stolze street	23961a3 stolze streets stre	
9	242-197-6202	217-686-0202	
10	271097 bayshore boulevard	271097 bayhore boulevard	
11	federico pearson	federico pearonon	
12	/carpina/hope_&_faith/litle	/carpina/hope_z_fith/litlee	
13	dine-in/code/	dine-cin/code/code/code//	
14	+264-97-568-217-145	+264-97-5482-2745	
15	+51-2721-208-63	+51-271-208-6363	
16	wildberries_ru	wilders-murces-lu	
17	leona owens	leona owenslens	
18	+220-557-859-04	+220-557-859-0404	
19	kati castro	kati castro	
20	5566 hellertown road	56 cll rock	
21	6867 granville drive	6867 granville drive	
22	1600 fire water	1600 willia ster	
23	+45-39-007-1887	+45-39-07-188787	
24	65634/tennessee%20river	65634/tenesseeo 2000-rive	
25	596-033-4046	515-000-4848	
26	18 cutter ridge road	18 cutter ridge road	
27	tampa fl	tampa fl	
28	492288 west 28th terrace south	492288 west 28th terrace south	
29	166 water power	www.santow.com/tarticles	

```
8                    7
9                    6
10                   1
11                   2
12                   3
13                  12
14                   5
15                   3
16                   8
17                   4
18                   2
19                   0
20                  12
21                   0
22                   7
23                   3
24                   7
25                   6
26                   0
27                   0
28                   0
29                  20
```

[55]:
```
# Value Counts
LD_TRAIN_VC = dict([(i, 0) for i in range(LD_TRAIN_DF['levenshtein_distance'].
  ↪max()+1)])
for ld in LD_TRAIN_DF['levenshtein_distance']:
    LD_TRAIN_VC[ld] += 1

plt.figure(figsize=(15,8))
pd.Series(LD_TRAIN_VC).plot(kind='bar', width=1)
plt.title(f'Train Levenstein Distance Distribution | Mean: {LD_TRAIN_DF.
  ↪levenshtein_distance.mean():.4f}')
plt.xlabel('Levenstein Distance')
plt.ylabel('Sample Count')
plt.xlim(-0.50, LD_TRAIN_DF.levenshtein_distance.max()+0.50)
plt.grid(axis='y')
plt.savefig('temp.png')
plt.show()
```



34 Levenstein Distance Evaluation

[56]:
```
# Compute Levenstein Distances
def get_ld_val():
    N = 100 if IS_INTERACTIVE else 1000
    LD_VAL = []
    for idx, (frames, phrase_true) in enumerate(zip(tqdm(X_val, total=N),␣
 ↪y_val)):
        # Predict Phrase and Convert to String
        phrase_pred = predict_phrase(frames).numpy()
        phrase_pred = outputs2phrase(phrase_pred)
        # True Phrase Ordinal to String
        phrase_true = outputs2phrase(phrase_true)
        # Add Levenstein Distance
        LD_VAL.append({
            'phrase_true': phrase_true,
            'phrase_pred': phrase_pred,
            'levenshtein_distance': levenshtein(phrase_pred, phrase_true),
        })
        # Take subset in interactive mode
        if idx == N:
            break

    # Convert to DataFrame
    LD_VAL_DF = pd.DataFrame(LD_VAL)
```

```
    return LD_VAL_DF
```

[57]:
```
if USE_VAL:
    LD_VAL_DF = get_ld_val()

    # Display Errors
    display(LD_VAL_DF.head(30))
```

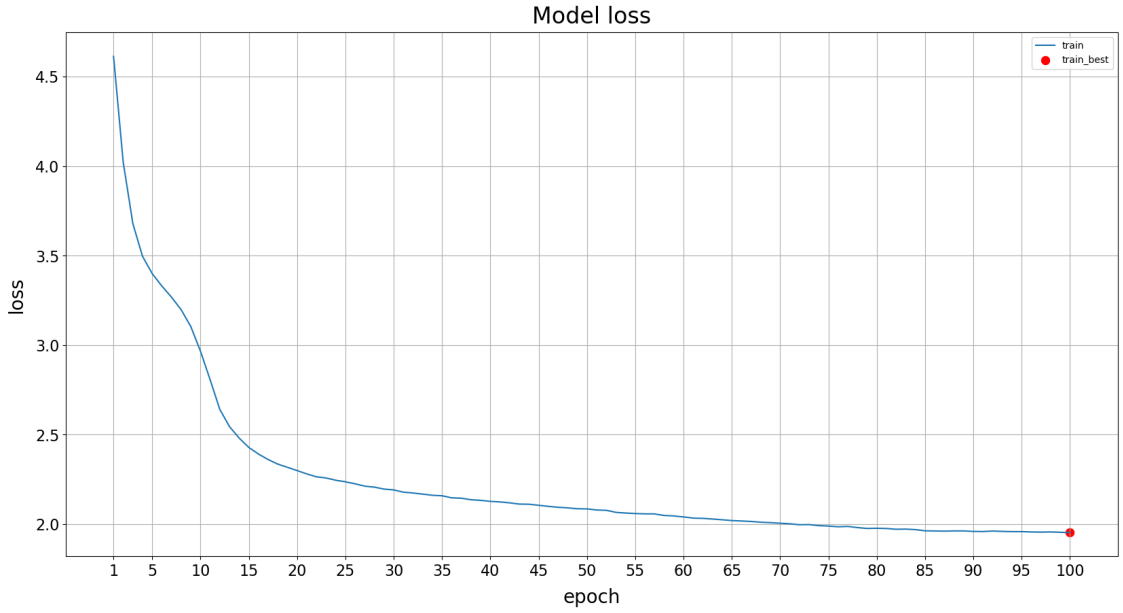
[58]:
```
# Value Counts
if USE_VAL:
    LD_VAL_VC = dict([(i, 0) for i in range(LD_VAL_DF['levenshtein_distance'].
 ↪max()+1)])
    for ld in LD_VAL_DF['levenshtein_distance']:
        LD_VAL_VC[ld] += 1

    plt.figure(figsize=(15,8))
    pd.Series(LD_VAL_VC).plot(kind='bar', width=1)
    plt.title(f'Validation Levenstein Distance Distribution | Mean: {LD_VAL_DF.
 ↪levenshtein_distance.mean():.4f}')
    plt.xlabel('Levenstein Distance')
    plt.ylabel('Sample Count')
    plt.xlim(0-0.50, LD_VAL_DF.levenshtein_distance.max()+0.50)
    plt.grid(axis='y')
    plt.savefig('temp.png')
    plt.show()
```

35 Training History

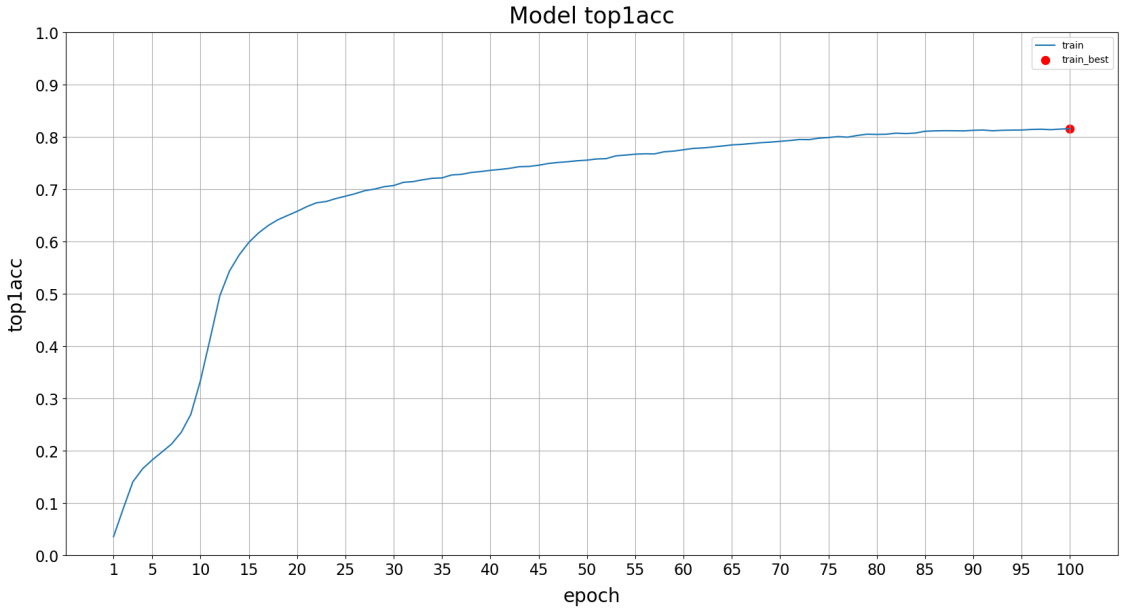
[59]:
```
def plot_history_metric(metric, f_best=np.argmax, ylim=None, yscale=None,␣
 ↪yticks=None):
    # Only plot when training
    if not TRAIN_MODEL:
        return

    plt.figure(figsize=(20, 10))

    values = history.history[metric]
    N_EPOCHS = len(values)
    val = 'val' in ''.join(history.history.keys())
    # Epoch Ticks
    if N_EPOCHS <= 20:
        x = np.arange(1, N_EPOCHS + 1)
    else:
        x = [1, 5] + [10 + 5 * idx for idx in range((N_EPOCHS - 10) // 5 + 1)]
```

```
    x_ticks = np.arange(1, N_EPOCHS+1)

    # Validation
    if val:
        val_values = history.history[f'val_{metric}']
        val_argmin = f_best(val_values)
        plt.plot(x_ticks, val_values, label=f'val')

    # summarize history for accuracy
    plt.plot(x_ticks, values, label=f'train')
    argmin = f_best(values)
    plt.scatter(argmin + 1, values[argmin], color='red', s=75, marker='o',␣
↪label=f'train_best')
    if val:
        plt.scatter(val_argmin + 1, val_values[val_argmin], color='purple',␣
↪s=75, marker='o', label=f'val_best')

    plt.title(f'Model {metric}', fontsize=24, pad=10)
    plt.ylabel(metric, fontsize=20, labelpad=10)

    if ylim:
        plt.ylim(ylim)

    if yscale is not None:
        plt.yscale(yscale)

    if yticks is not None:
        plt.yticks(yticks, fontsize=16)

    plt.xlabel('epoch', fontsize=20, labelpad=10)
    plt.tick_params(axis='x', labelsize=8)
    plt.xticks(x, fontsize=16) # set tick step to 1 and let x axis start at 1
    plt.yticks(fontsize=16)

    plt.legend(prop={'size': 10})
    plt.grid()
    plt.show()
```

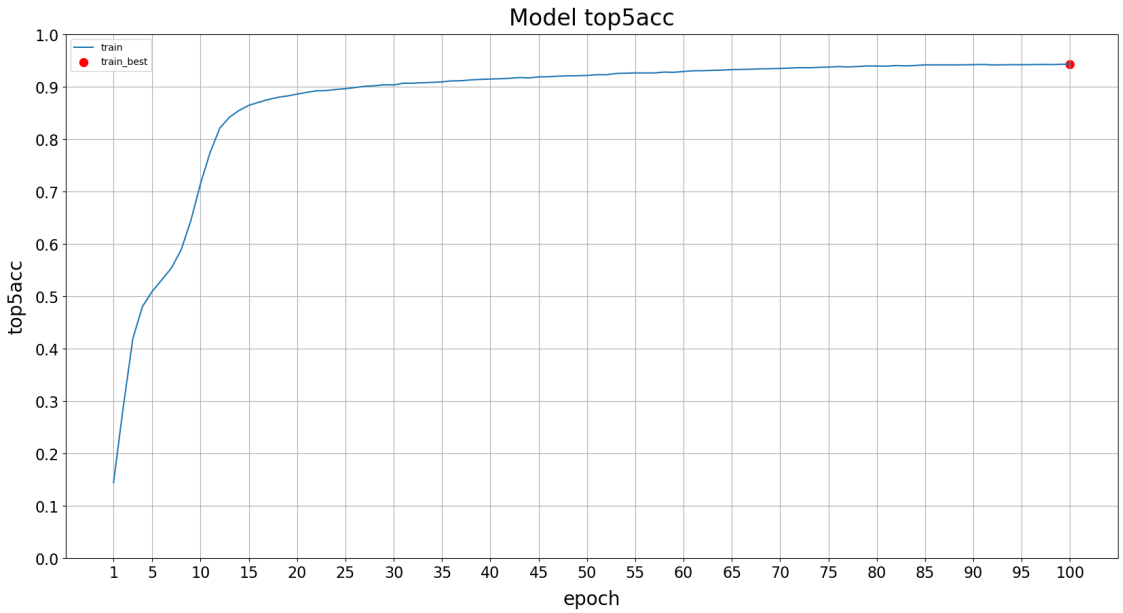
[60]:
```
plot_history_metric('loss', f_best=np.argmin)
```



[61]:
```
plot_history_metric('top1acc', ylim=[0,1], yticks=np.arange(0.0, 1.1, 0.1))
```



[62]:
```
plot_history_metric('top5acc', ylim=[0,1], yticks=np.arange(0.0, 1.1, 0.1))
```



36 Inference

[63]:
```
# Model Layer Names
for l in model.layers:
    print(l.name)
```

```
frames
masking
embedding
encoder
phrase
decoder
classifier
```

[64]:
```
# TFLite model for submission
class TFLiteModel(tf.Module):
    def __init__(self, model):
        super(TFLiteModel, self).__init__()

        # Load the feature generation and main models
        self.preprocess_layer = preprocess_layer
        self.model = model

    @tf.function(jit_compile=True)
    def encoder(self, x, frames_inp):
        x = self.model.get_layer('embedding')(x)
```

```
        x = self.model.get_layer('encoder')(x, frames_inp)

        return x

    @tf.function(jit_compile=True)
    def decoder(self, x, phrase_inp):
        x = self.model.get_layer('decoder')(x, phrase_inp)
        x = self.model.get_layer('classifier')(x)

        return x

    @tf.function(input_signature=[tf.TensorSpec(shape=[None, N_COLS0], dtype=tf.
↪float32, name='inputs')])
    def __call__(self, inputs):
        # Number Of Input Frames
        N_INPUT_FRAMES = tf.shape(inputs)[0]
        # Preprocess Data
        frames_inp = self.preprocess_layer(inputs)
        # Add Batch Dimension
        frames_inp = tf.expand_dims(frames_inp, axis=0)
        # Get Encoding
        encoding = self.encoder(frames_inp, frames_inp)
        # Make Prediction
        phrase = tf.fill([1,MAX_PHRASE_LENGTH], PAD_TOKEN)
        # Predict One Token At A Time
        stop = False
        for idx in tf.range(MAX_PHRASE_LENGTH):
            # Cast phrase to int8
            phrase = tf.cast(phrase, tf.int8)
            # If EOS token is predicted, stop predicting
            outputs = tf.cond(
                stop,
                lambda: tf.one_hot(tf.cast(phrase, tf.int32),␣
↪N_UNIQUE_CHARACTERS),
                lambda: self.decoder(encoding, phrase)
            )
            # Add predicted token to input phrase
            phrase = tf.cast(phrase, tf.int32)
            # Replcae PAD token with predicted token up to idx
            phrase = tf.where(
                tf.range(MAX_PHRASE_LENGTH) < idx + 1,
                tf.argmax(outputs, axis=2, output_type=tf.int32),
                phrase,
            )
            # Predicted Token
            predicted_token = phrase[0,idx]
            # If EOS (End Of Sentence) token is predicted stop
```

```
            if not stop:
                stop = predicted_token == EOS_TOKEN

        # Squeeze outputs
        outputs = tf.squeeze(phrase, axis=0)
        outputs = tf.one_hot(outputs, N_UNIQUE_CHARACTERS)

        # Return a dictionary with the output tensor
        return {'outputs': outputs }

# Define TF Lite Model
tflite_keras_model = TFLiteModel(model)

# Sanity Check
# demo_sequence_id = 1816796431
demo_sequence_id = example_parquet_df.index.unique()[0]
demo_raw_data = example_parquet_df.loc[demo_sequence_id, COLUMNS0].values
demo_phrase_true = train_sequence_id.loc[demo_sequence_id, 'phrase']
print(f'demo_raw_data shape: {demo_raw_data.shape}, dtype: {demo_raw_data.
 ↪dtype}')
demo_output = tflite_keras_model(demo_raw_data)['outputs'].numpy()
print(f'demo_output shape: {demo_output.shape}, dtype: {demo_output.dtype}')
print(f'demo_outputs phrase decoded: {outputs2phrase(demo_output)}')
print(f'phrase true: {demo_phrase_true}')
```

```
demo_raw_data shape: (123, 164), dtype: float32
demo_output shape: (32, 62), dtype: float32
demo_outputs phrase decoded: 3 creek house
phrase true: 3 creekhouse
```

[65]:
```
# Create Model Converter
keras_model_converter = tf.lite.TFLiteConverter.
 ↪from_keras_model(tflite_keras_model)
# Convert Model
tflite_model = keras_model_converter.convert()
# Write Model
with open('/kaggle/working/model.tflite', 'wb') as f:
    f.write(tflite_model)
```

[66]:
```
# Add selected_columns json to only select specific columns from input frames
with open('inference_args.json', 'w') as f:
     json.dump({ 'selected_columns': COLUMNS0.tolist() }, f)
```

[67]:
```
# Zip Model
!zip submission.zip /kaggle/working/model.tflite /kaggle/working/inference_args.
 ↪json
```

```
adding: kaggle/working/model.tflite (deflated 9%)
adding: kaggle/working/inference_args.json (deflated 83%)
```