Introduction to Deep Learning

Name: **Due Date:** Apr. 26, 2021
Programming Assignment: Number 4

Problem Description

Interpreting 3 dimensional poses of humans from real images has broad area of applications in self-driving cars, entertainment, environmental awareness and daily human interaction and emotion prediction. The Human3.6M contains 3.6 Million 3D human poses acquired through video recordings for training dynamic regression models to predict the 3D body joints positions using the image sequences.

Model Description

The dynamic regression model consist of a convolutional layer, a Long-short-term memory (LSTM) and a fully connected dense layer. The structure is as follows:

- An input layer with shape [Batch-size, $8, 224, 224, 3$]
- A Convolution layer with filter size $[5, 5, 32]$ and an output shape of [Batch-size, $8, 220, 220, 32$] and a ReLU activation function.
- A 3D Max pooling layer with pool size of $(1, 2, 2)$ and stride $[1, 2, 2]$. The output shape is [Batch-size, $8, 110, 110, 32$].
- A reshape layer to flatten the the last 3 dimensions of the tensor to shape $[batch_size, 8, -1]$.
- An LSTM layer with 50 units and return sequence set to true. Default activation functions of standard tensorflow LSTM module was used for the gates.
- A fully connected layer of 51 nodes. With a sigmoid activation function.
- a reshape layer to reshape the tensor to $[batch_size, 8, 17, 3]$

A forward pass of this model is implemented by feeding the image data into the network through the input layer. The convolution layers were implemented using tf.keras.layers.Conv2D() and the filter size, padding and activation arguments are passed along.
The pooling layers were implemented using tf.keras.layers.MaxPool3D(), the reshape layer was implemented using tf.keras.layers.Reshape(), the LSTM layer was implemented using tf.keras.layers.LSTM, while the fully connected layer was implemented using tf.keras.layers.Dense().

Loss function equations

$$\text{loss} = (ytrue - ypred)^2 \tag{1}$$

$$\text{Average loss} = \frac{1}{M}\sum_{i=1}^{M}(\frac{1}{8}\sum_{j=1}^{8}(\frac{1}{17}\sum_{k=1}^{17}(\frac{1}{3}\sum(\text{loss})))) \tag{2}$$

$$M = \text{ batch size}$$

$$\text{If } x, y, z = \text{ the 3D coordinates,}$$

$$\text{MPJPE} = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{8}\sum_{j=1}^{8}\sqrt{x_i^2 + y_i^2 + z_i^2}\ \forall i = 1, 2, \ldots 17 \tag{3}$$

Mean Per Joint Position Error (MPJPE)

MPJPE	
Joint	MPJPE (mm)
1	17.38
2	40.59
3	83.66
4	167.78
5	37.50
6	85.08
7	164.06
8	53.92
9	46.95
10	63.31
11	60.47
12	57.83
13	106.52
14	170.22
15	71.60
16	112.17
17	186.93

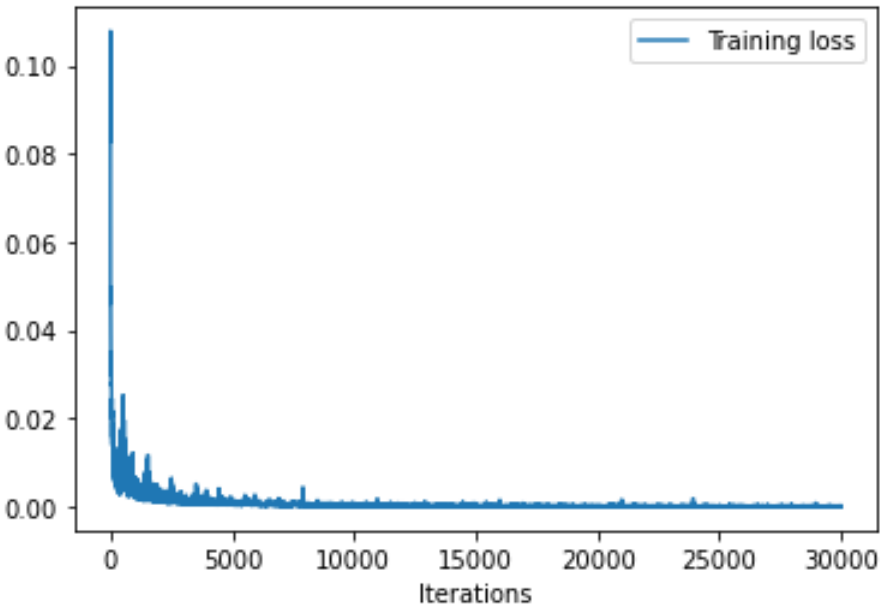

Figure 1: Training loss

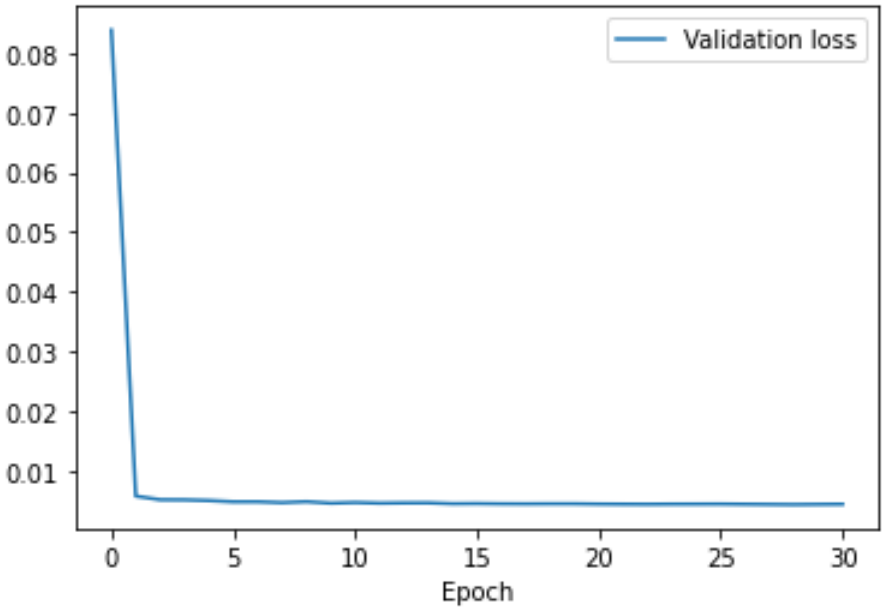

Figure 2: Validation loss

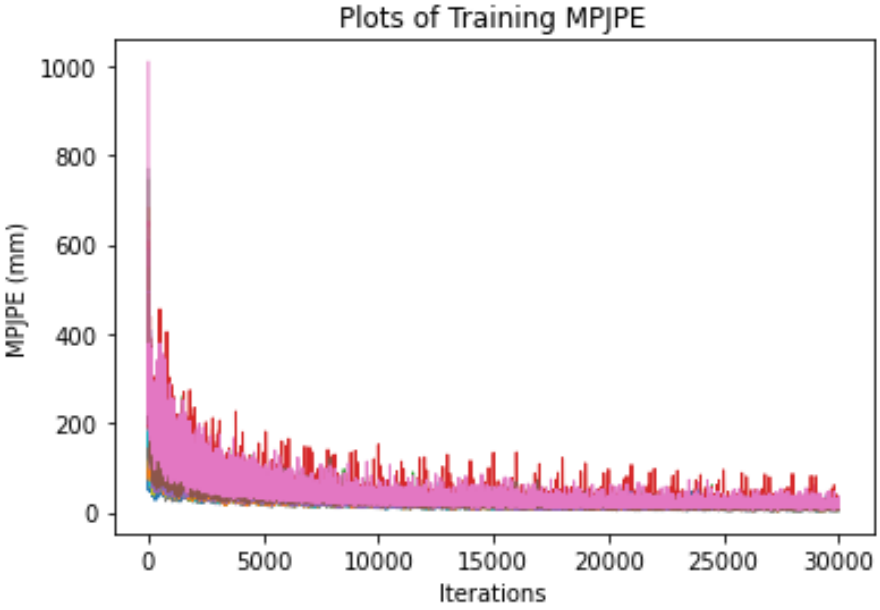

Figure 3: Training MPJPE

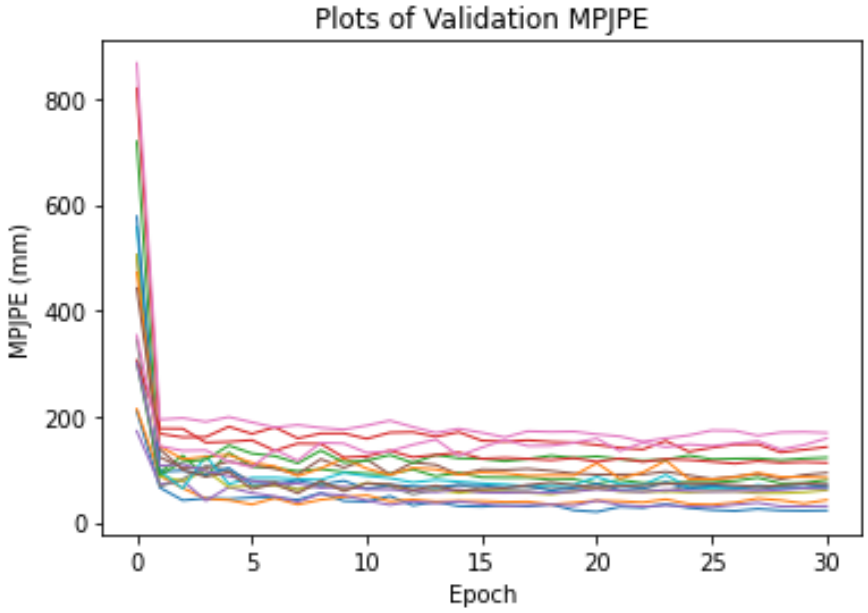

Figure 4: Validation MPJPE

Experimental setting

- A laptop with 16gb CPU and Nvidia RTX 2070 (8gb) was used.
- Python 3.8 was used for coding.
- Training and testing data were imported and processed using the hints provided. The data were normalized by dividing with 255.0. The training data was split into 10 sub training data since the local machine could not hold the entire training data in it memory and still train.
- Tensorflow version 2.4.0 was to build the model function using tf.keras.models.Sequential
- the model architecture was set during call of function Model with the input dimension
- The gradients of the weights were computed using tensorflow's tf.GradientTape() function. The loss function used is the mean squared error; tf.nn.mean_squared_error()
- Adam optimizer with learning rate $\eta = 0.0001$ was initially used for training. The optimizer was also used to apply the gradients to the model's trainable variables.
- Model is initialized by calling the Model function.
- Each of the sub-training data is shuffled and batches are generated using tf.data.Dataset.from_tensor_slices function.
- Although training loss was calculated during each weight update, the training and testing losses and accuracy were recorded after each epoch.

Hyper-parameters Used

- batch size = 5, No of batches = $590/5 \times 10 = 1180$
- Epoch = 30
- Number of iterations: No of batches $\times$ Epoch = $1180 \times 30 = 35400$
- learning rate $\eta = 0.0001$

Discussion

Batch size of 1 was initialy used and then slowly increased up to 10. Batch_sizes above 8 caused the out of memory (OOM) errors hence 5 was chosen. Using a learning rate of 0.001, the training converges slightly faster than the initially chosen rate.