Batch normalisation

July 30, 2021

1 Batch normalisation layers

In this reading we will look at incorporating batch normalisation into our models and look at an example of how we do this in practice.

As usual, let's first import tensorflow.

```
In [1]: import tensorflow as tf
        print(tf.__version__)
```

```
2.0.0
```

We will be working with the diabetes dataset that we have been using in this week's screencasts.

Let's load and pre-process the dataset.

```
In [2]: # Load the dataset

        from sklearn.datasets import load_diabetes
        diabetes_dataset = load_diabetes()
```

```
In [3]: # Save the input and target variables

        from sklearn.model_selection import train_test_split

        data = diabetes_dataset['data']
        targets = diabetes_dataset['target']
```

```
In [4]: # Normalise the target data (this will make clearer training curves)

        targets = (targets - targets.mean(axis=0)) / (targets.std())
```

```
In [5]: # Split the dataset into training and test datasets

        train_data, test_data, train_targets, test_targets = train_test_split(data, targets, te
```

1.0.1 Batch normalisation - defining the model

We can implement batch normalisation into our model by adding it in the same way as any other layer.

```
In [6]: from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Flatten, Dense, Conv2D, MaxPooling2D, BatchNormali
```

```
In [7]: # Build the model

        model = Sequential([
            Dense(64, input_shape=[train_data.shape[1],], activation="relu"),
            BatchNormalization(),  # <- Batch normalisation layer
            Dropout(0.5),
            BatchNormalization(),  # <- Batch normalisation layer
            Dropout(0.5),
            Dense(256, activation='relu'),
        ])

        # NB: We have not added the output layer because we still have more layers to add!
```

```
In [8]: # Print the model summary

        model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                704
_________________________________________________________________
batch_normalization (BatchNo (None, 64)                256
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
batch_normalization_1 (Batch (None, 64)                256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               16640
=================================================================
Total params: 17,856
Trainable params: 17,600
Non-trainable params: 256
_________________________________________________________________
```

Recall that there are some parameters and hyperparameters associated with batch normalisation.

- The hyperparameter **momentum** is the weighting given to the previous running mean when re-computing it with an extra minibatch. By **default**, it is set to 0.99.
- The hyperparameter ϵ is used for numeric stability when performing the normalisation over the minibatch. By **default** it is set to 0.001.
- The parameters β and γ are used to implement an affine transformation after normalisation. By **default**, β is an all-zeros vector, and γ is an all-ones vector.

1.0.2 Customising parameters

These can all be changed (along with various other properties) by adding optional arguments to `tf.keras.layers.BatchNormalization()`.

We can also specify the axis for batch normalisation. By default, it is set as -1.

Let's see an example.

In [9]:
```
# Add a customised batch normalisation layer

        model.add(tf.keras.layers.BatchNormalization(
            momentum=0.95,
            epsilon=0.005,
            axis = -1,
            beta_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.05),
            gamma_initializer=tf.keras.initializers.Constant(value=0.9)
        ))
```

In [10]:
```
# Add the output layer

         model.add(Dense(1))
```

1.1 Compile and fit the model

Let's now compile and fit our model with batch normalisation, and track the progress on training and validation sets.

First we compile our model.

In [11]:
```
# Compile the model

         model.compile(optimizer='adam',
                       loss='mse',
                       metrics=['mae'])
```

Now we fit the model to the data.

In [14]:
```
# Train the model

         history = model.fit(train_data, train_targets, epochs=100, validation_split=0.15, bat
```

```
Train on 337 samples, validate on 60 samples
Epoch 1/100
337/337 - 0s - loss: 0.6150 - mae: 0.6247 - val_loss: 0.7754 - val_mae: 0.7192
Epoch 2/100
337/337 - 0s - loss: 0.5311 - mae: 0.5997 - val_loss: 0.7706 - val_mae: 0.7169
Epoch 3/100
337/337 - 0s - loss: 0.5471 - mae: 0.6071 - val_loss: 0.7799 - val_mae: 0.7162
Epoch 4/100
337/337 - 0s - loss: 0.5252 - mae: 0.5753 - val_loss: 0.8090 - val_mae: 0.7260
Epoch 5/100
337/337 - 0s - loss: 0.5724 - mae: 0.6149 - val_loss: 0.7875 - val_mae: 0.7164
Epoch 6/100
337/337 - 0s - loss: 0.5827 - mae: 0.6150 - val_loss: 0.7812 - val_mae: 0.7077
Epoch 7/100
337/337 - 0s - loss: 0.5689 - mae: 0.6215 - val_loss: 0.7770 - val_mae: 0.7092
Epoch 8/100
337/337 - 0s - loss: 0.5911 - mae: 0.6106 - val_loss: 0.7544 - val_mae: 0.7069
Epoch 9/100
337/337 - 0s - loss: 0.5589 - mae: 0.5964 - val_loss: 0.7438 - val_mae: 0.6991
Epoch 10/100
337/337 - 0s - loss: 0.5304 - mae: 0.5781 - val_loss: 0.7604 - val_mae: 0.7041
Epoch 11/100
337/337 - 0s - loss: 0.5458 - mae: 0.5979 - val_loss: 0.7494 - val_mae: 0.6969
Epoch 12/100
337/337 - 0s - loss: 0.5328 - mae: 0.5965 - val_loss: 0.7390 - val_mae: 0.6912
Epoch 13/100
337/337 - 0s - loss: 0.5292 - mae: 0.5804 - val_loss: 0.7392 - val_mae: 0.6936
Epoch 14/100
337/337 - 0s - loss: 0.5372 - mae: 0.6048 - val_loss: 0.7211 - val_mae: 0.6867
Epoch 15/100
337/337 - 0s - loss: 0.5366 - mae: 0.5986 - val_loss: 0.7143 - val_mae: 0.6848
Epoch 16/100
337/337 - 0s - loss: 0.5204 - mae: 0.5850 - val_loss: 0.7215 - val_mae: 0.6902
Epoch 17/100
337/337 - 0s - loss: 0.6051 - mae: 0.6313 - val_loss: 0.7432 - val_mae: 0.6994
Epoch 18/100
337/337 - 0s - loss: 0.5525 - mae: 0.5885 - val_loss: 0.7543 - val_mae: 0.7001
Epoch 19/100
337/337 - 0s - loss: 0.5438 - mae: 0.5871 - val_loss: 0.7608 - val_mae: 0.6977
Epoch 20/100
337/337 - 0s - loss: 0.5147 - mae: 0.5857 - val_loss: 0.7529 - val_mae: 0.6926
Epoch 21/100
337/337 - 0s - loss: 0.5351 - mae: 0.5905 - val_loss: 0.7582 - val_mae: 0.6961
Epoch 22/100
337/337 - 0s - loss: 0.5441 - mae: 0.5873 - val_loss: 0.7320 - val_mae: 0.6897
Epoch 23/100
337/337 - 0s - loss: 0.5301 - mae: 0.5937 - val_loss: 0.7047 - val_mae: 0.6749
Epoch 24/100
```

```
337/337 - 0s - loss: 0.5116 - mae: 0.5692 - val_loss: 0.6976 - val_mae: 0.6662
Epoch 25/100
337/337 - 0s - loss: 0.5309 - mae: 0.5911 - val_loss: 0.6939 - val_mae: 0.6619
Epoch 26/100
337/337 - 0s - loss: 0.5026 - mae: 0.5726 - val_loss: 0.7060 - val_mae: 0.6680
Epoch 27/100
337/337 - 0s - loss: 0.5446 - mae: 0.6027 - val_loss: 0.7060 - val_mae: 0.6688
Epoch 28/100
337/337 - 0s - loss: 0.5551 - mae: 0.5969 - val_loss: 0.7147 - val_mae: 0.6734
Epoch 29/100
337/337 - 0s - loss: 0.5240 - mae: 0.5954 - val_loss: 0.7241 - val_mae: 0.6762
Epoch 30/100
337/337 - 0s - loss: 0.5589 - mae: 0.5945 - val_loss: 0.6970 - val_mae: 0.6676
Epoch 31/100
337/337 - 0s - loss: 0.5426 - mae: 0.5860 - val_loss: 0.6760 - val_mae: 0.6638
Epoch 32/100
337/337 - 0s - loss: 0.4845 - mae: 0.5666 - val_loss: 0.6755 - val_mae: 0.6643
Epoch 33/100
337/337 - 0s - loss: 0.5025 - mae: 0.5677 - val_loss: 0.7021 - val_mae: 0.6706
Epoch 34/100
337/337 - 0s - loss: 0.5290 - mae: 0.5890 - val_loss: 0.7181 - val_mae: 0.6742
Epoch 35/100
337/337 - 0s - loss: 0.5259 - mae: 0.5802 - val_loss: 0.7226 - val_mae: 0.6782
Epoch 36/100
337/337 - 0s - loss: 0.5293 - mae: 0.5797 - val_loss: 0.7133 - val_mae: 0.6749
Epoch 37/100
337/337 - 0s - loss: 0.4902 - mae: 0.5668 - val_loss: 0.7242 - val_mae: 0.6752
Epoch 38/100
337/337 - 0s - loss: 0.5116 - mae: 0.5747 - val_loss: 0.7407 - val_mae: 0.6803
Epoch 39/100
337/337 - 0s - loss: 0.5140 - mae: 0.5920 - val_loss: 0.7484 - val_mae: 0.6840
Epoch 40/100
337/337 - 0s - loss: 0.5116 - mae: 0.5819 - val_loss: 0.7576 - val_mae: 0.6891
Epoch 41/100
337/337 - 0s - loss: 0.5312 - mae: 0.6087 - val_loss: 0.7665 - val_mae: 0.6944
Epoch 42/100
337/337 - 0s - loss: 0.5030 - mae: 0.5737 - val_loss: 0.7356 - val_mae: 0.6797
Epoch 43/100
337/337 - 0s - loss: 0.4893 - mae: 0.5721 - val_loss: 0.7127 - val_mae: 0.6669
Epoch 44/100
337/337 - 0s - loss: 0.4718 - mae: 0.5515 - val_loss: 0.7091 - val_mae: 0.6652
Epoch 45/100
337/337 - 0s - loss: 0.4926 - mae: 0.5705 - val_loss: 0.7073 - val_mae: 0.6658
Epoch 46/100
337/337 - 0s - loss: 0.5099 - mae: 0.5741 - val_loss: 0.7161 - val_mae: 0.6713
Epoch 47/100
337/337 - 0s - loss: 0.4886 - mae: 0.5623 - val_loss: 0.7291 - val_mae: 0.6735
Epoch 48/100
```

```
337/337 - 0s - loss: 0.4849 - mae: 0.5592 - val_loss: 0.7348 - val_mae: 0.6684
Epoch 49/100
337/337 - 0s - loss: 0.5290 - mae: 0.5850 - val_loss: 0.7133 - val_mae: 0.6639
Epoch 50/100
337/337 - 0s - loss: 0.5061 - mae: 0.5784 - val_loss: 0.7115 - val_mae: 0.6790
Epoch 51/100
337/337 - 0s - loss: 0.5261 - mae: 0.5924 - val_loss: 0.6995 - val_mae: 0.6748
Epoch 52/100
337/337 - 0s - loss: 0.5103 - mae: 0.5744 - val_loss: 0.7048 - val_mae: 0.6742
Epoch 53/100
337/337 - 0s - loss: 0.4999 - mae: 0.5718 - val_loss: 0.7223 - val_mae: 0.6766
Epoch 54/100
337/337 - 0s - loss: 0.4833 - mae: 0.5674 - val_loss: 0.7227 - val_mae: 0.6744
Epoch 55/100
337/337 - 0s - loss: 0.5134 - mae: 0.5795 - val_loss: 0.7246 - val_mae: 0.6782
Epoch 56/100
337/337 - 0s - loss: 0.5042 - mae: 0.5704 - val_loss: 0.7462 - val_mae: 0.6837
Epoch 57/100
337/337 - 0s - loss: 0.5023 - mae: 0.5739 - val_loss: 0.7513 - val_mae: 0.6873
Epoch 58/100
337/337 - 0s - loss: 0.5045 - mae: 0.5717 - val_loss: 0.7491 - val_mae: 0.6834
Epoch 59/100
337/337 - 0s - loss: 0.4948 - mae: 0.5611 - val_loss: 0.7419 - val_mae: 0.6746
Epoch 60/100
337/337 - 0s - loss: 0.5305 - mae: 0.5919 - val_loss: 0.7529 - val_mae: 0.6794
Epoch 61/100
337/337 - 0s - loss: 0.5139 - mae: 0.5695 - val_loss: 0.7524 - val_mae: 0.6847
Epoch 62/100
337/337 - 0s - loss: 0.4804 - mae: 0.5604 - val_loss: 0.7289 - val_mae: 0.6787
Epoch 63/100
337/337 - 0s - loss: 0.5131 - mae: 0.5851 - val_loss: 0.7295 - val_mae: 0.6788
Epoch 64/100
337/337 - 0s - loss: 0.5325 - mae: 0.6023 - val_loss: 0.7555 - val_mae: 0.6845
Epoch 65/100
337/337 - 0s - loss: 0.5544 - mae: 0.5942 - val_loss: 0.7751 - val_mae: 0.6923
Epoch 66/100
337/337 - 0s - loss: 0.4939 - mae: 0.5705 - val_loss: 0.7635 - val_mae: 0.6917
Epoch 67/100
337/337 - 0s - loss: 0.5119 - mae: 0.5761 - val_loss: 0.7468 - val_mae: 0.6847
Epoch 68/100
337/337 - 0s - loss: 0.5079 - mae: 0.5667 - val_loss: 0.7503 - val_mae: 0.6836
Epoch 69/100
337/337 - 0s - loss: 0.4601 - mae: 0.5407 - val_loss: 0.7404 - val_mae: 0.6788
Epoch 70/100
337/337 - 0s - loss: 0.4829 - mae: 0.5683 - val_loss: 0.7365 - val_mae: 0.6797
Epoch 71/100
337/337 - 0s - loss: 0.5127 - mae: 0.5751 - val_loss: 0.7392 - val_mae: 0.6830
Epoch 72/100
```

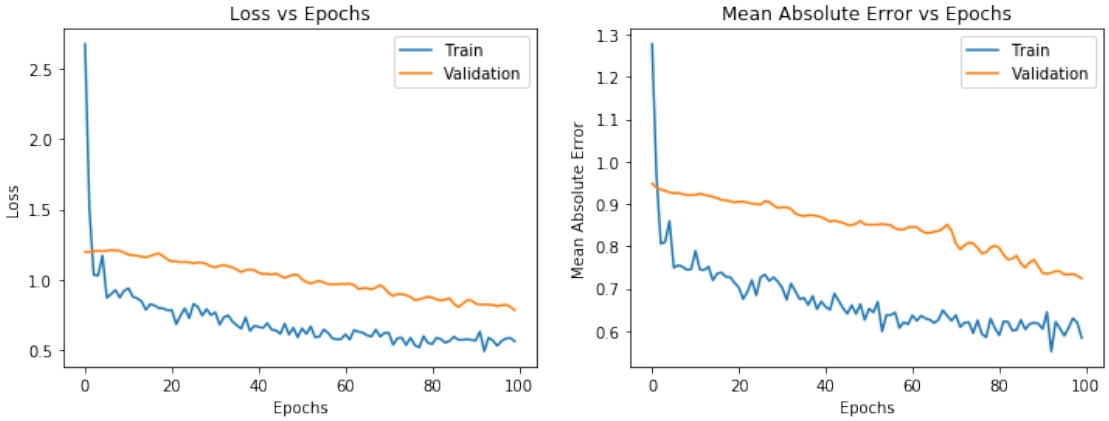
```
337/337 - 0s - loss: 0.4762 - mae: 0.5635 - val_loss: 0.7507 - val_mae: 0.6871
Epoch 73/100
337/337 - 0s - loss: 0.5516 - mae: 0.6026 - val_loss: 0.7538 - val_mae: 0.6918
Epoch 74/100
337/337 - 0s - loss: 0.4864 - mae: 0.5734 - val_loss: 0.7401 - val_mae: 0.6890
Epoch 75/100
337/337 - 0s - loss: 0.5009 - mae: 0.5699 - val_loss: 0.7373 - val_mae: 0.6856
Epoch 76/100
337/337 - 0s - loss: 0.5374 - mae: 0.5827 - val_loss: 0.7410 - val_mae: 0.6846
Epoch 77/100
337/337 - 0s - loss: 0.5042 - mae: 0.5770 - val_loss: 0.7436 - val_mae: 0.6843
Epoch 78/100
337/337 - 0s - loss: 0.5287 - mae: 0.5813 - val_loss: 0.7494 - val_mae: 0.6880
Epoch 79/100
337/337 - 0s - loss: 0.5082 - mae: 0.5683 - val_loss: 0.7460 - val_mae: 0.6814
Epoch 80/100
337/337 - 0s - loss: 0.4905 - mae: 0.5653 - val_loss: 0.7529 - val_mae: 0.6857
Epoch 81/100
337/337 - 0s - loss: 0.5535 - mae: 0.6055 - val_loss: 0.7488 - val_mae: 0.6877
Epoch 82/100
337/337 - 0s - loss: 0.5251 - mae: 0.5771 - val_loss: 0.7685 - val_mae: 0.6929
Epoch 83/100
337/337 - 0s - loss: 0.5211 - mae: 0.5851 - val_loss: 0.7856 - val_mae: 0.6991
Epoch 84/100
337/337 - 0s - loss: 0.5231 - mae: 0.5958 - val_loss: 0.7981 - val_mae: 0.7040
Epoch 85/100
337/337 - 0s - loss: 0.5137 - mae: 0.5670 - val_loss: 0.7726 - val_mae: 0.6914
Epoch 86/100
337/337 - 0s - loss: 0.5181 - mae: 0.5656 - val_loss: 0.7441 - val_mae: 0.6756
Epoch 87/100
337/337 - 0s - loss: 0.4803 - mae: 0.5649 - val_loss: 0.7401 - val_mae: 0.6754
Epoch 88/100
337/337 - 0s - loss: 0.4534 - mae: 0.5440 - val_loss: 0.7489 - val_mae: 0.6816
Epoch 89/100
337/337 - 0s - loss: 0.5331 - mae: 0.5883 - val_loss: 0.7696 - val_mae: 0.6969
Epoch 90/100
337/337 - 0s - loss: 0.5174 - mae: 0.5687 - val_loss: 0.7755 - val_mae: 0.7015
Epoch 91/100
337/337 - 0s - loss: 0.5041 - mae: 0.5809 - val_loss: 0.7711 - val_mae: 0.6961
Epoch 92/100
337/337 - 0s - loss: 0.4701 - mae: 0.5520 - val_loss: 0.7424 - val_mae: 0.6808
Epoch 93/100
337/337 - 0s - loss: 0.4878 - mae: 0.5569 - val_loss: 0.7140 - val_mae: 0.6680
Epoch 94/100
337/337 - 0s - loss: 0.4897 - mae: 0.5585 - val_loss: 0.6918 - val_mae: 0.6611
Epoch 95/100
337/337 - 0s - loss: 0.4914 - mae: 0.5693 - val_loss: 0.6811 - val_mae: 0.6587
Epoch 96/100
```

```
337/337 - 0s - loss: 0.5304 - mae: 0.5800 - val_loss: 0.7025 - val_mae: 0.6696
Epoch 97/100
337/337 - 0s - loss: 0.4453 - mae: 0.5471 - val_loss: 0.7001 - val_mae: 0.6697
Epoch 98/100
337/337 - 0s - loss: 0.5041 - mae: 0.5749 - val_loss: 0.6777 - val_mae: 0.6628
Epoch 99/100
337/337 - 0s - loss: 0.4840 - mae: 0.5622 - val_loss: 0.6586 - val_mae: 0.6565
Epoch 100/100
337/337 - 0s - loss: 0.4790 - mae: 0.5648 - val_loss: 0.6657 - val_mae: 0.6595
```

Finally, we plot training and validation loss and accuracy to observe how the accuracy of our model improves over time.

In [13]:
```
# Plot the learning curves

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

frame = pd.DataFrame(history.history)
epochs = np.arange(len(frame))

fig = plt.figure(figsize=(12,4))

# Loss plot
ax = fig.add_subplot(121)
ax.plot(epochs, frame['loss'], label="Train")
ax.plot(epochs, frame['val_loss'], label="Validation")
ax.set_xlabel("Epochs")
ax.set_ylabel("Loss")
ax.set_title("Loss vs Epochs")
ax.legend()

# Accuracy plot
ax = fig.add_subplot(122)
ax.plot(epochs, frame['mae'], label="Train")
ax.plot(epochs, frame['val_mae'], label="Validation")
ax.set_xlabel("Epochs")
ax.set_ylabel("Mean Absolute Error")
ax.set_title("Mean Absolute Error vs Epochs")
ax.legend()
```

Out[13]:
```
<matplotlib.legend.Legend at 0x7f80d82abb38>
```



1.2 Further reading and resources

- https://keras.io/layers/normalization/
- https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/layers/BatchNormalization

In [ ]: