Neural Networks: Tensorflow model operation

Andrzej Kordecki

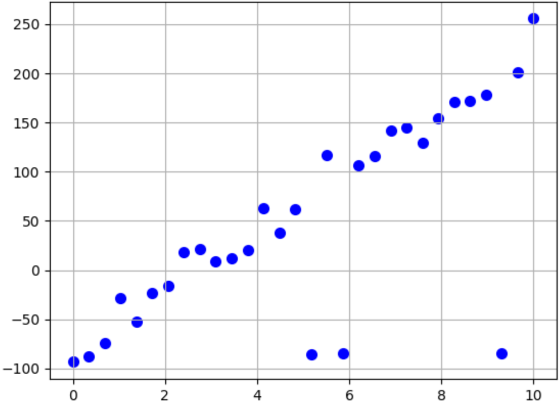
Neural Networks for Classification and Identification (ML.EM05): Exercise 11
Division of Theory of Machines and Robots
Institute of Aeronautics and Applied Mechanics
Faculty of Power and Aeronautical Engineering
Warsaw University of Technology

Table of Contents

Examples of network models

Regression model

Linear regression task:



Regression model

Dataset generation and standardization:

```
# Generate regression dataset
X = np.linspace(0, 10, 30)
Y = 2 * X + 4*np.random.rand(np.size(X))-2
X = X.reshape(-1, 1)
Y = Y.reshape(-1, 1)
# Data scaling
Y = (Y - np.mean(Y))/np.std(Y)
```

Regression model

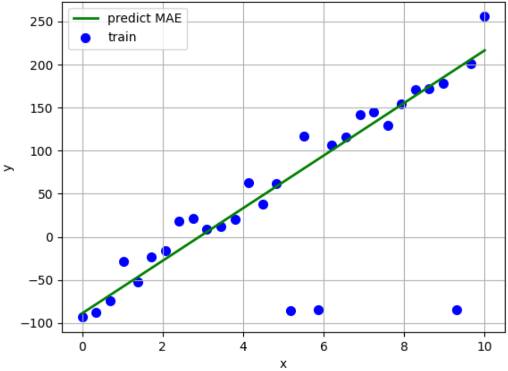
Building and training of neural network model:

```
# Define model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1, activation='linear',
                          input_shape=(1, ))
])

# Set training parameters
model.compile(optimizer='SGD',
              loss=tf.keras.losses.MAE)

# Train model
model.fit(X, Y, epochs=1000)
```

Regression model

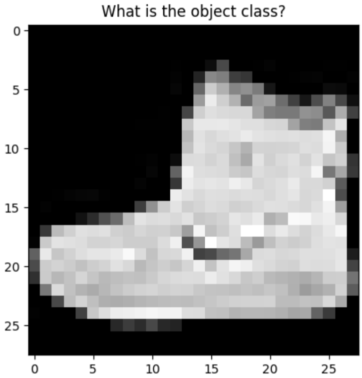
Predicting model results:

```
predX1 = np.linspace(min(X), max(X), 100)
predY1 = model.predict(predX1)
```



Classification model model

Image classification task:



Classification model model

Image dataset - MNIST dataset:

Classification model model

Dataset loading and normalization:

```
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_images, train_labels),
        (test_images, test_labels)
              = fashion_mnist.load_data()
class_names = ['T-shirt/top', 'Trouser', 'Pullover',
'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag',
'Ankle boot']

train_images = train_images / 255.0
test_images = test_images / 255.0
```

Classification model model

Building and training of neural network model:

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation=None)
])
model.summary()

model.compile(optimizer='adam',
              loss=tf.keras.losses
                   .SparseCategoricalCrossentropy(
                       from_logits=True),
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=10)
```

Classification model model

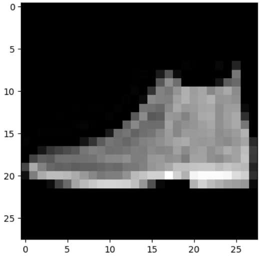
Model evaluation on train and test dataset:

```
test_loss, test_acc = model.evaluate(test_images,
                                      test_labels, verbose=2)
print('\nTest accuracy:', test_acc)

# Prints:
Test accuracy: 0.8813999891281128

# But, the training accuracy is:
Epoch 10/10
1875/1875 [==========================] - 3s 1ms/step
- loss: 0.2388 - accuracy: 0.9109
```

Classification model model

Model predictions:

```
# Input size of image must mach input size of model
# t_image.shape = (1,28,28) and image size = (28,28)
t_image = np.expand_dims(test_images[0, :, :], axis=0)
t_label = test_labels[0]
```



Classification model model

Model predictions with logits:

```
prediction = model.predict(t_image)
print(f"Logits: Prediction = {prediction},
      Label = {t_label}")
print(f"Prediction = {class_names[np.argmax(prediction)
      label = {class_names[t_label]}")

# Prints:
Logits: Prediction = [[ -8.693618  -10.190063
-11.337278   -11.141912   -9.663246   -2.8053892
-8.676396    2.8095415 -10.380032    6.2661247]],
Label = 9
Prediction = Ankle boot, label = Ankle boot
```

Classification model model

Attach a softmax layer to convert the logits to probabilities:

```
probability_model = tf.keras.Sequential(
                [model, tf.keras.layers.Softmax()])
```

Do we need to train model again?

The softmax layer rescales the outputs, so that the activations sum to 1 and all of them lie between 0 and 1:

$$y_k = \frac{\exp(v_k)}{\sum_k \exp(v_k)}$$

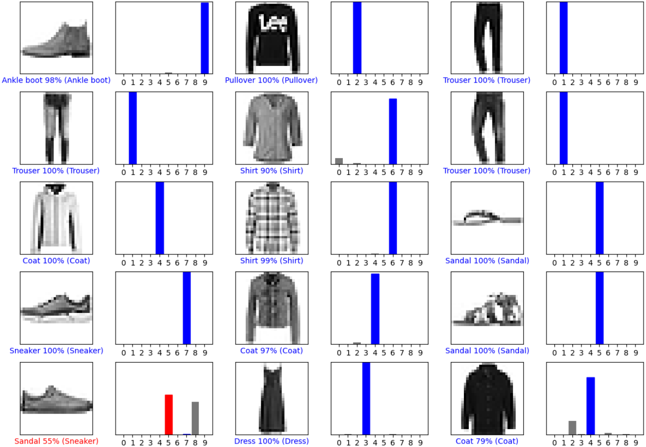
Classification model model

Model predictions with Softmax layer:

```
predictions = probability_model.predict(test_images)
print(f"Softmax: Prediction = {prediction},
                                Label = {t_label}")


# Prints:
Logits: Prediction = [[3.0869717e-07 6.9124908e-08
2.1948530e-08 2.6684052e-08 1.1706536e-07
1.1136732e-04 3.1405943e-07 3.0569704e-02
5.7165298e-08 9.6931803e-01]], Label = 9
Prediction = Ankle boot, label = Ankle boot
```

Classification model model

Evaluation of model on test dataset.



Improving model evaluation and training

Results evaluation

Results evaluation general approach:

- Test model after training on new data (at least validation data)
- Checking loss/measures value training history.

To evaluate the inference-mode loss and metrics for the data provided with use tensorflow function:

```
eval_model = model.evaluate(test_images,  test_labels)
print(f"Model evaluation = {eval_model}")

# Prints information about loss and accuracy
# - measures used in training:
# Model evaluation =
# [0.4173202514648437 5, 0.8860999941825867]
```

Results evaluation

Results evaluation from training history - Part 1. We will include information about validation data:

```
history = model.fit(train_images,
                    train_labels,
                    epochs=10,
                    validation_data=(test_images,
                                     test_labels))
```

Results evaluation

If we have one set we can use on of function arguments to split dataset into training and validation set.

```
history = model.fit(train_images,
                    train_labels,
                    epochs=10,
                    validation_split=0.2)
```

The validation data is selected from the last samples in the x and y data provided, before shuffling.

Results evaluation

Results evaluation from training history - Part 2. We will use default measures names.

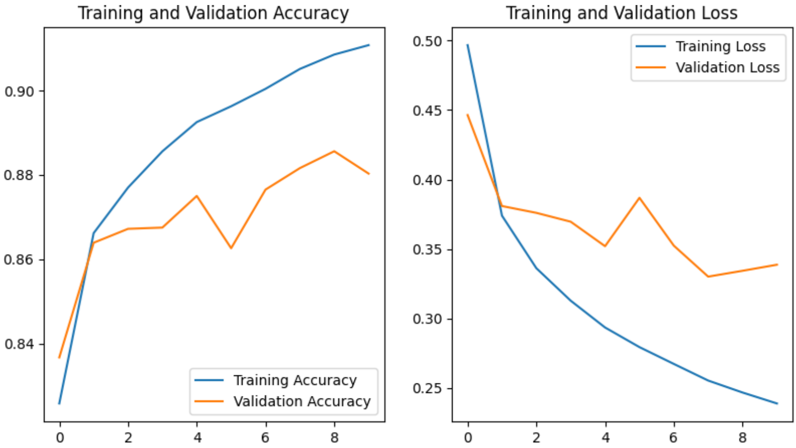
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(len(loss))
```

Results evaluation

Results evaluation from training history - Part 3:

```
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc,
            label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss,
            label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

Results evaluation



Results evaluation

TensorBoard is a tool for providing the measurements and visualizations needed during the machine learning workflow.

```
log_dir="logs/fit/program_results"
tensorboard_callback = tf.keras.callbacks.TensorBoard(
        log_dir=log_dir, histogram_freq=1)
model.fit(x=x_train, y=y_train, epochs=5,
          validation_data=(x_test, y_test),
          callbacks=[tensorboard_callback])
```

Start TensorBoard through the command line:

```
tensorboard --logdir logs/fit
# Print: TensorBoard 1.5.1 at
         http://localhost:6006 (Press CTRL+C to quit)
```

Open address in internet browser.

Save and load models

Model progress can be saved:

- after training,
- during training (e.g. after each epoch),
- before training.

This means a model can resume where it left off and avoid long training times. You can save:

- model structure,
- model structure and weights.

Save and load models

Before and after training:

- Load model or weights:

```
# Loads the weights
model.load_weights(checkpoint_path)
# Load the entire model from a HDF5 file.
models.load_model('my_model.h5')
```

- Save model or weights:

```
# Save the weights
model.save_weights(path)
# Save the entire model to a HDF5 file.
model.save('my_model.h5')
```

Save and load models

Saving during training with use of Checkpoint callback:

```
checkpoint_path = "c:/saves/cp_model.h5"
checkpoint_dir = os.path.dirname(checkpoint_path)

# Create a callback that saves the model's weights
cp_callback = tf.keras.callbacks.ModelCheckpoint(
                        filepath=checkpoint_path,
                        save_weights_only=True,
                        verbose=1)
# Train the model with callback
model.fit(train_images, train_labels, epochs=10,
          validation_data=(test_images,test_labels),
          callbacks=[cp_callback])  # Pass callback
```

Overfit and underfit

Overfit and underfit in model training:

- Overfitting occurs when the accuracy of our model on the validation data would peak after training for a number of epochs, and would then stagnate or start decreasing.
- Underfitting occurs when there is still room for improvement on the train data. In can happen due to: too simple model or too short training time.

Overfit and underfit

To prevent overfitting we performe:

- Add more data or new different dataset to complete our original training dataset. The dataset should cover the full range of inputs that the model is expected to handle. Additional data may only be useful if it covers new and uncommon cases.
- Use regularization methods. These methods place constraints on the quantity and type of information your model can store. If a network can only afford to memorize a small number of patterns, the optimization process will force it to focus on the most prominent patterns, which have a better chance of generalizing well.

Overfit and underfit

The simplest way to prevent overfitting is to start with a small model with a small number of learnable parameters (small model capacity):

- Large model with more parameters will have more "memorization capacity" and therefore will be able to easily learn a perfect dictionary-like mapping between training samples and their targets.
- Deep learning models tend to be good at fitting to the training data, but the real challenge is generalization, not fitting.

Unfortunately, there is no magical formula to determine the right size or architecture/structure of your model. You will have to experiment using a series of different architectures.

Overfit and underfit

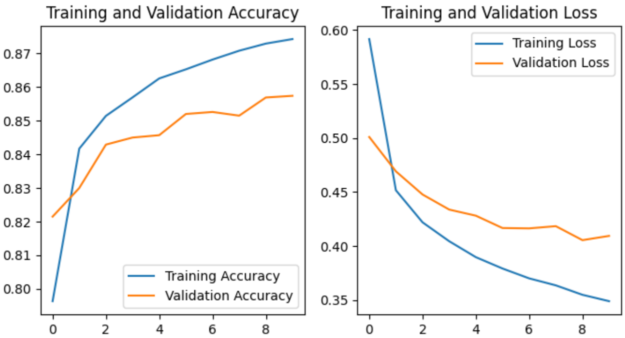
To find an appropriate model size, it's best to start with relatively few layers and parameters, then begin increasing the size of the layers or adding new layers until you see diminishing returns on the validation loss.

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(10, activation=None)
])
history = model.fit(train_images, train_labels,
    epochs=10, validation_data=(test_images,
                                test_labels))
```

Overfit and underfit

The obtained results:

```
Epoch 10/10
loss: 0.3489 - accuracy: 0.8743
- val_loss: 0.4093 - val_accuracy: 0.8574
```



Overfit and underfit

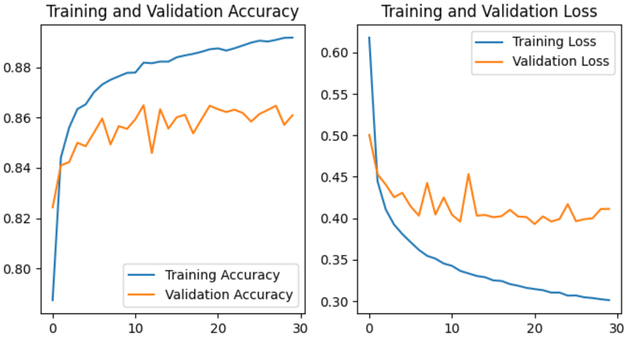
We increase total number of epochs to 30.

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(10, activation=None)
])
history = model.fit(train_images, train_labels,
    epochs=30, validation_data=(test_images,
                                test_labels))
```

Overfit and underfit

The obtained results:

```
Epoch 30/30
loss: 0.3012 - accuracy: 0.8917
- val_loss: 0.4113 - val_accuracy: 0.8609
```



Overfit and underfit

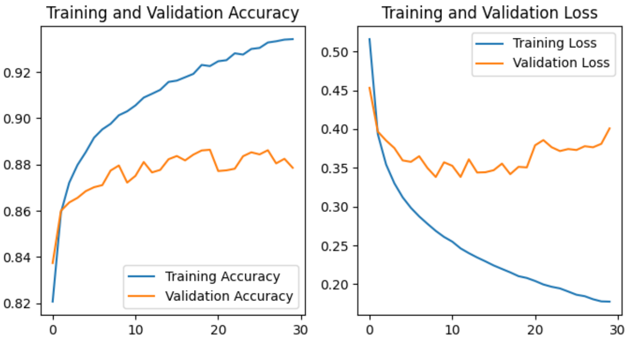
We increase total number of neurons in hidden layer to 64.

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation=None)
])
history = model.fit(train_images, train_labels,
    epochs=30, validation_data=(test_images,
                                test_labels))
```

Overfit and underfit

The obtained results:

```
Epoch 30/30
loss: 0.1774 - accuracy: 0.9343
- val_loss: 0.4009 - val_accuracy: 0.8786
```



Overfit and underfit

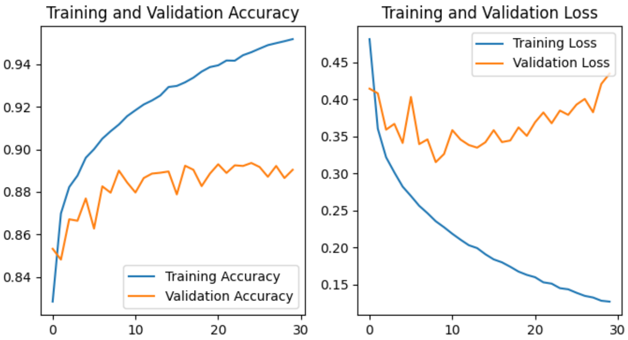
What happen if we further increase number of neurons to 256.

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(10, activation=None)
])
history = model.fit(train_images, train_labels,
    epochs=30, validation_data=(test_images,
                                test_labels))
```

Overfit and underfit

The obtained results:

```
Epoch 30/30
loss: 0.1774 - accuracy: 0.9343 \
- val_loss: 0.4009 - val_accuracy: 0.8786
```



Overfit and underfit

Conclusions for avoiding overfiting and underfitting:

- Compare the validation metrics to the training metrics.
- If both metrics are moving in the same direction, the model is too small or too small number epochs.
- If the validation metric begins to stagnate while the training metric continues to improve, you are probably close to overfitting. There is a small difference between training and validation metrics. We have found a good model structure.
- If the validation metric is going in the wrong direction, the model is clearly overfitting.

Overfit and underfit

If we check the big model loss and accuracy, these models have as good values as simple model at some epoch. The big model have also better capacity to improve results. How can we improve big models?

We can also avoiding overfiting and underfitting in case of big model:

- Early stopping criterion.
- Dynamic learning rate and momentum methods.
- Weight regularization.
- Adding generalization layers: dropout and batch normalization.