GROUP NAME: MI4

Machine Intelligence 1
WS 2020/21, Obermayer/Kashef

Exercise Sheet 62
due: 22.12.2020 at 23:55

In [ ]:
```
import numpy as np
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import  Input, Dense, LSTM
from keras.optimizers import Adam
```

(a) Create the train and validation data as follows:

- Draw 10,000 different series, each consisting of 30 integer numbers (i.e. digits) from 0 to 9, where each digit is uniformly distributed and independent from the others, that is, $\left(x_1^{(\alpha)}, \ldots, x_{30}^{(\alpha)}\right) \in \{0, 1, \ldots, 9\}^{30}$ with $x_t^{(\alpha)} \overset{\text{iid}}{\sim} \mathcal{U}_{\text{int}}(0, 9)$ for $t = 1, \ldots, 30,\ \alpha = 1, \ldots, 10000$.
- A series gets the label 1 if its sum is greater or equal to 100 and the label 0 otherwise.
- Use 8,000 series as training set and 2,000 series as your test set.

In [ ]:
```
X = np.random.randint(0, 9, size = [10000, 30, 1])
y = np.asarray([1 if np.sum(i)>=100 else 0 for i in X])
y = y.reshape((10000, 1))

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=12)
```

(b) Build a recurrent network for number series classification as follows:

- The network is composed of 200 LSTM cells yielding an output vector $\underline{\mathbf{h}}^{(t)}$ in each time step t of the 30 time steps within a sequence.
- On top of the LSTM layer there is a single linear output neuron (receiving input from the 200 LSTM cells. The output neuron uses the logistic sigmoidal as its non-linearity, i.e, $y(\underline{\mathbf{h}}^{(t)}) \in (0, 1)$. $y(\underline{\mathbf{h}}^{(30)})$. The output neuron's activity should be interpreted as the probability that the sum of the number series is greater or equal to 100.
  Hints for Keras users: Apply a `Dense` layer with one output neuron and sigmoid activation after the `LSTM` layer.

In [ ]:
```
model = Sequential()
model.add(LSTM(200, input_shape=(30, 1), return_sequences=False))
model.add(Dense(1, activation='sigmoid'))
```

(c) Use *cross-entropy* between the labels of the training data set and the output y of the network *after the last time step* as the loss function for the learning process.

(d) Use classification accuracy as the performance measure to compare training and generalization performance for the trained model.

(e) As many frameworks differ in their implementation of LSTM and allow for different levels of tweaking the parameters you are free to choose parameters which seem reasonable to you (maybe the default values)[1].

(f) For the training procedure iterate over the data 60 times (i.e. use 60 epochs) in random fashion and use a mini-batch size of 50.

(g) Train your model using the *Adam* algorithm. Use the following parameters: $\eta = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1 * 10^{-8}$.

Deliverables:

1. (8 points) Evaluate the final accuracy of the model on the validation data.

In [ ]:
```
adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1*10**(-8))
model.compile(optimizer=adam, loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, batch_size=50, epochs=60, verbose=1, validation_split=0.1, shuffle= Tr
```

```
Epoch 1/60
144/144 [==============================] - 11s 66ms/step - loss: 0.2622 - accuracy: 0.8978 - val_loss: 0.112
3 - val_accuracy: 0.9438
Epoch 2/60
144/144 [==============================] - 9s 61ms/step - loss: 0.1315 - accuracy: 0.9459 - val_loss: 0.0785
- val_accuracy: 0.9638
Epoch 3/60
144/144 [==============================] - 9s 62ms/step - loss: 0.0756 - accuracy: 0.9696 - val_loss: 0.0586
- val_accuracy: 0.9825
Epoch 4/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0865 - accuracy: 0.9642 - val_loss: 0.0548
- val_accuracy: 0.9887
Epoch 5/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0582 - accuracy: 0.9777 - val_loss: 0.0464
- val_accuracy: 0.9887
Epoch 6/60
144/144 [==============================] - 9s 65ms/step - loss: 0.0571 - accuracy: 0.9785 - val_loss: 0.0530
- val_accuracy: 0.9737
Epoch 7/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0528 - accuracy: 0.9775 - val_loss: 0.0389
- val_accuracy: 0.9887
Epoch 8/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0489 - accuracy: 0.9781 - val_loss: 0.0279
- val_accuracy: 0.9887
Epoch 9/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0466 - accuracy: 0.9799 - val_loss: 0.0439
- val_accuracy: 0.9850
Epoch 10/60
144/144 [==============================] - 9s 66ms/step - loss: 0.0518 - accuracy: 0.9790 - val_loss: 0.0306
- val_accuracy: 0.9925
Epoch 11/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0418 - accuracy: 0.9844 - val_loss: 0.0246
- val_accuracy: 0.9912
Epoch 12/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0405 - accuracy: 0.9852 - val_loss: 0.0539
- val_accuracy: 0.9787
Epoch 13/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0497 - accuracy: 0.9808 - val_loss: 0.0242
- val_accuracy: 0.9937
Epoch 14/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0346 - accuracy: 0.9867 - val_loss: 0.0247
- val_accuracy: 0.9875
Epoch 15/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0403 - accuracy: 0.9826 - val_loss: 0.0315
- val_accuracy: 0.9887
Epoch 16/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0327 - accuracy: 0.9870 - val_loss: 0.0287
- val_accuracy: 0.9875
Epoch 17/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0325 - accuracy: 0.9860 - val_loss: 0.0220
- val_accuracy: 0.9962
Epoch 18/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0304 - accuracy: 0.9906 - val_loss: 0.0282
- val_accuracy: 0.9850
Epoch 19/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0372 - accuracy: 0.9822 - val_loss: 0.0189
- val_accuracy: 0.9937
Epoch 20/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0342 - accuracy: 0.9869 - val_loss: 0.0247
- val_accuracy: 0.9937
Epoch 21/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0294 - accuracy: 0.9887 - val_loss: 0.0659
- val_accuracy: 0.9688
Epoch 22/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0470 - accuracy: 0.9809 - val_loss: 0.0251
- val_accuracy: 0.9937
Epoch 23/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0341 - accuracy: 0.9857 - val_loss: 0.0223
- val_accuracy: 0.9925
Epoch 24/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0315 - accuracy: 0.9874 - val_loss: 0.0255
- val_accuracy: 0.9887
Epoch 25/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0238 - accuracy: 0.9913 - val_loss: 0.0602
- val_accuracy: 0.9737
Epoch 26/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0369 - accuracy: 0.9829 - val_loss: 0.0333
- val_accuracy: 0.9825
Epoch 27/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0361 - accuracy: 0.9844 - val_loss: 0.1040
- val_accuracy: 0.9538
Epoch 28/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0306 - accuracy: 0.9869 - val_loss: 0.0394
- val_accuracy: 0.9787
Epoch 29/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0336 - accuracy: 0.9843 - val_loss: 0.0411
- val_accuracy: 0.9812
Epoch 30/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0235 - accuracy: 0.9911 - val_loss: 0.0209
- val_accuracy: 0.9950
Epoch 31/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0235 - accuracy: 0.9921 - val_loss: 0.0253
- val_accuracy: 0.9875
Epoch 32/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0256 - accuracy: 0.9891 - val_loss: 0.0389
- val_accuracy: 0.9775
Epoch 33/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0296 - accuracy: 0.9876 - val_loss: 0.0929
- val_accuracy: 0.9538
Epoch 34/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0412 - accuracy: 0.9829 - val_loss: 0.0172
- val_accuracy: 0.9937
Epoch 35/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0224 - accuracy: 0.9909 - val_loss: 0.0178
- val_accuracy: 0.9937
Epoch 36/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0208 - accuracy: 0.9919 - val_loss: 0.0205
- val_accuracy: 0.9925
Epoch 37/60
144/144 [==============================] - 9s 65ms/step - loss: 0.0228 - accuracy: 0.9918 - val_loss: 0.0234
- val_accuracy: 0.9887
Epoch 38/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0315 - accuracy: 0.9873 - val_loss: 0.0178
- val_accuracy: 0.9950
Epoch 39/60
144/144 [==============================] - 9s 65ms/step - loss: 0.0191 - accuracy: 0.9940 - val_loss: 0.0158
- val_accuracy: 0.9950
Epoch 40/60
144/144 [==============================] - 9s 66ms/step - loss: 0.0203 - accuracy: 0.9917 - val_loss: 0.0177
- val_accuracy: 0.9900
Epoch 41/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0212 - accuracy: 0.9931 - val_loss: 0.0158
- val_accuracy: 0.9950
Epoch 42/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0222 - accuracy: 0.9907 - val_loss: 0.0153
- val_accuracy: 0.9950
Epoch 43/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0257 - accuracy: 0.9912 - val_loss: 0.1133
- val_accuracy: 0.9550
Epoch 44/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0280 - accuracy: 0.9896 - val_loss: 0.0165
- val_accuracy: 0.9925
Epoch 45/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0196 - accuracy: 0.9926 - val_loss: 0.0182
- val_accuracy: 0.9925
Epoch 46/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0250 - accuracy: 0.9889 - val_loss: 0.0173
- val_accuracy: 0.9950
Epoch 47/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0184 - accuracy: 0.9927 - val_loss: 0.0279
- val_accuracy: 0.9850
Epoch 48/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0223 - accuracy: 0.9888 - val_loss: 0.0987
- val_accuracy: 0.9663
Epoch 49/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0302 - accuracy: 0.9884 - val_loss: 0.0270
- val_accuracy: 0.9862
Epoch 50/60
144/144 [==============================] - 9s 65ms/step - loss: 0.0187 - accuracy: 0.9930 - val_loss: 0.0521
- val_accuracy: 0.9762
Epoch 51/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0255 - accuracy: 0.9884 - val_loss: 0.0268
- val_accuracy: 0.9875
Epoch 52/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0190 - accuracy: 0.9927 - val_loss: 0.0266
- val_accuracy: 0.9812
Epoch 53/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0200 - accuracy: 0.9908 - val_loss: 0.0152
- val_accuracy: 0.9937
Epoch 54/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0185 - accuracy: 0.9927 - val_loss: 0.0188
- val_accuracy: 0.9925
Epoch 55/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0153 - accuracy: 0.9946 - val_loss: 0.0421
- val_accuracy: 0.9775
Epoch 56/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0176 - accuracy: 0.9920 - val_loss: 0.0163
- val_accuracy: 0.9925
Epoch 57/60
144/144 [==============================] - 9s 64ms/step - loss: 0.0164 - accuracy: 0.9929 - val_loss: 0.0187
- val_accuracy: 0.9912
Epoch 58/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0280 - accuracy: 0.9885 - val_loss: 0.0225
- val_accuracy: 0.9887
Epoch 59/60
144/144 [==============================] - 9s 63ms/step - loss: 0.0268 - accuracy: 0.9886 - val_loss: 0.0170
- val_accuracy: 0.9937
Epoch 60/60
144/144 [==============================] - 9s 65ms/step - loss: 0.0203 - accuracy: 0.9919 - val_loss: 0.0255
- val_accuracy: 0.9875
```

In [ ]:
```
scores = model.evaluate(X_test, y_test, verbose=1, batch_size=50)
print('Accurracy: {}'.format(scores[1]))
```

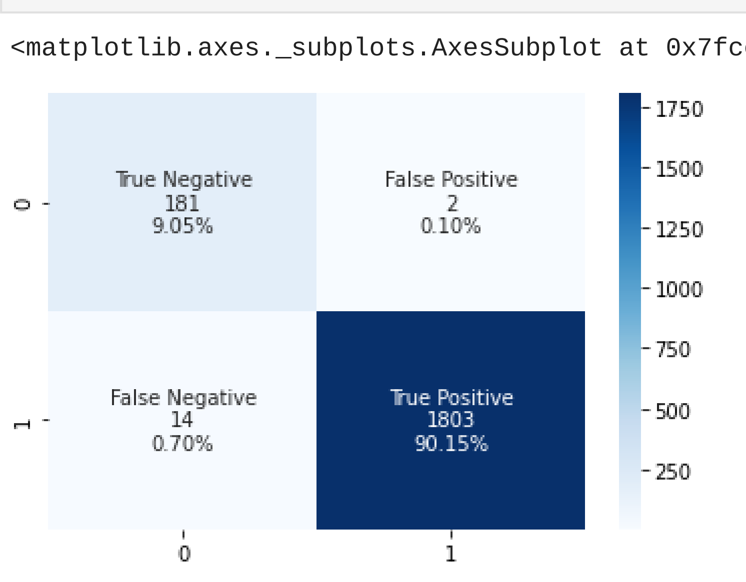
```
40/40 [==============================] - 1s 22ms/step - loss: 0.0186 - accuracy: 0.9920
Accurracy: 0.9919999837875366
```

In [ ]:
```
y_pred = model.predict(X_test).round()
cf_matrix = confusion_matrix(y_test, y_pred)
```

```
[[ 181    2]
 [  14 1803]]
```

In [ ]:
```
group_names = ['True Negative','False Positive','False Negative','True Positive']
group_counts = ['{0:0.0f}'.format(value) for value in cf_matrix.flatten()]
group_percentages = ['{0:.2%}'.format(value) for value in cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fced733c898>



2. (2 points) Inspect the class distribution (no. of positive examples vs. no. of negative examples). Briefly disucss the validity of the accuracy measure in relation to that distribution and possible alternatives.

The class distribution is skewed toward number of positives. (number of ones is higher)

The acurracy measure which is used in the above steps shows the performance of the model based on the number of correct classifications. As the distribution is skewed toward one type of class (around 91 percent of the datapoints belong to one class), its better to know how well the model performs in terms of false negatives and false positives. This gives the idea of the model performance within each class.

So an alternative would be using confusion matrix as shown above.

Loading [MathJax]/extensions/Safe.js