Keras_1

April 24, 2021

Nicholas Paisley

EPOCH 150

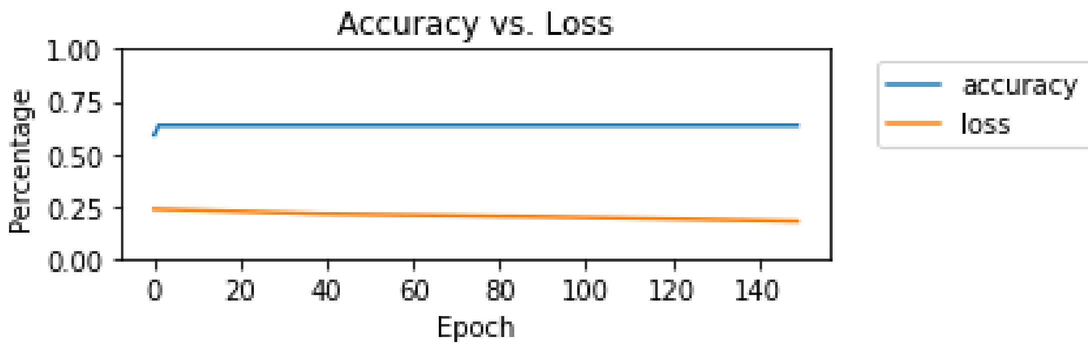
[1]:
```
#libraries to import and Keras
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers

#loading the data
features_and_targets = np.loadtxt('features_and_targets.csv',delimiter=',')
np.random.shuffle(features_and_targets)

# split into input (X) and output (Y) variables
X = features_and_targets[:,0:5]
Y = features_and_targets[:,5:7]

#makes it so the random variables don't change for each run
np.random.seed(7)

#creates model
model = Sequential() #A Sequential model is appropriate for a plain stack of␣
↪layers where each layer has exactly one input tensor and one output tensor
model.add(Dense(4, input_dim=5, activation='relu'))
#hidden layer: 4
#input_dim: specifying the number of elements within that first dimension only.␣
↪Initial amount of neurons
#activation: relu - > Applies the rectified linear unit activation function.␣
↪max(x, 0), the element-wise maximum of 0 and the input tensor.
model.add(Dense(3, activation='relu'))
model.add(Dense(2, activation='sigmoid'))
#activation: sigmoid -> sigmoid(x) = 1 / (1 + exp(-x)). For small values (<-5),␣
↪sigmoid returns a value close to zero, and for large values (>5) the result␣
↪of the function gets close to 1.

# Compile model
```

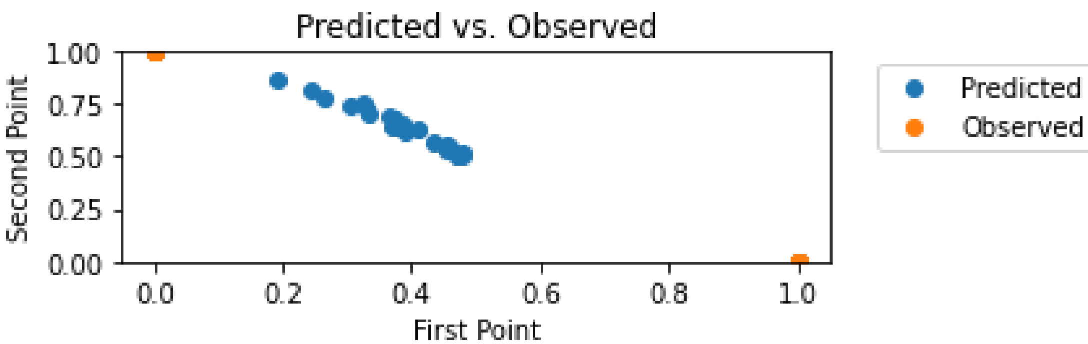
```python
model.compile(optimizer='adam', loss='mean_squared_error',metrics=['accuracy'])
#optimizer: Adam optimization is a stochastic gradient descent method that is␣
↪based on adaptive estimation of first-order and second-order moments.
#loss: The purpose of loss functions is to compute the quantity that a model␣
↪should seek to minimize during training.
#      mean_squared_error: Average of the square of the difference between␣
↪actual and estimated values.
#metrics: accuracy: Calculates how often predictions equal labels.

#The summary is textual and includes information about:
#The layers and their order in the model.
#The output shape of each layer.
#The number of parameters (weights) in each layer.
#The total number of parameters (weights) in the model.
model.summary()

history = model.fit(X,Y, epochs=150, verbose=0) #Trains the model for a fixed␣
↪number of epochs (iterations on a dataset).
scores = model.evaluate(X,Y) #Evaluation is a process during development of the␣
↪model to check whether the model is best fit for the given problem and␣
↪corresponding data

#Plotting 1st graph which is accuracy vs loss
plt.subplot(2,1,1)
plt.plot(history.history['accuracy']) #plotting accuaray
plt.plot(history.history['loss']) #plotting loss
plt.title("Accuracy vs. Loss") #title
plt.xlabel("Epoch") #Making Epoch x- value and label
plt.ylabel("Percentage") # percentage (0.0 - 1.0)
plt.ylim((0.00,1.00)) #y limit
plt.legend(['accuracy','loss'], bbox_to_anchor=(1.05, 1.0), loc='upper left')␣
↪#Creating legend outside of graph
plt.tight_layout() #automatically adjust subplot parameters to give specified␣
↪padding

#printing the accuracy
print('\n%s: %.2f%%' % (model.metrics_names[1], scores[1]*100))

predicted_targets = model.predict(X)

#Printing predicted targets against the observed targets
for i in range(22):
    print('Predicted: ',predicted_targets[i,:],'Observed: ',Y[i,:])

#printing the predicted targets
print(predicted_targets)
```

```
#Plotting graph of predicted vs observed
plt.subplot(2,1,2)
plt.scatter(predicted_targets[:,0],predicted_targets[:,1]) #plotting predicted␣
↪points
plt.scatter(Y[:,0],Y[:,1]) #plotting observed values
plt.title("Predicted vs. Observed") #title of the graph
plt.xlabel("First Point") #x label
plt.ylabel("Second Point") #y label
plt.ylim((0.00,1.00)) #y limit
plt.legend(['Predicted','Observed'], bbox_to_anchor=(1.05, 1.0), loc='upper␣
↪left') #Creating legend outside of graph
plt.tight_layout() #automatically adjust subplot parameters to give specified␣
↪padding

print(history.history.keys())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 4)                 24
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 15
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 8
=================================================================
Total params: 47
Trainable params: 47
Non-trainable params: 0
_________________________________________________________________
1/1 [==============================] - 0s 2ms/step - loss: 0.1811 - accuracy:
0.6364

accuracy: 63.64%
Predicted:  [0.304353   0.7395478] Observed:  [0. 1.]
Predicted:  [0.33236787 0.7041497 ] Observed:  [0. 1.]
Predicted:  [0.47742462 0.5138783 ] Observed:  [1. 0.]
Predicted:  [0.47212344 0.51005805] Observed:  [1. 0.]
Predicted:  [0.37065208 0.64698607] Observed:  [0. 1.]
Predicted:  [0.24520051 0.8135823 ] Observed:  [0. 1.]
Predicted:  [0.45297068 0.53385806] Observed:  [0. 1.]
Predicted:  [0.38432375 0.6538129 ] Observed:  [0. 1.]
Predicted:  [0.43345144 0.5627089 ] Observed:  [0. 1.]
Predicted:  [0.37185562 0.67667145] Observed:  [0. 1.]
Predicted:  [0.47719097 0.50528836] Observed:  [1. 0.]
Predicted:  [0.39043853 0.6186756 ] Observed:  [0. 1.]
```

```
Predicted:  [0.2662159  0.77654827] Observed:  [0. 1.]
Predicted:  [0.37910601 0.6409492 ] Observed:  [1. 0.]
Predicted:  [0.18998319 0.8655764 ] Observed:  [0. 1.]
Predicted:  [0.36463326 0.68911976] Observed:  [0. 1.]
Predicted:  [0.4585702  0.54296887] Observed:  [1. 0.]
Predicted:  [0.4529259  0.55580103] Observed:  [1. 0.]
Predicted:  [0.32811403 0.73022974] Observed:  [0. 1.]
Predicted:  [0.32666573 0.7538046 ] Observed:  [0. 1.]
Predicted:  [0.4097159 0.634783 ] Observed:  [1. 0.]
Predicted:  [0.4712695 0.5108622] Observed:  [1. 0.]
[[0.304353   0.7395478 ]
 [0.33236787 0.7041497 ]
 [0.47742462 0.5138783 ]
 [0.47212344 0.51005805]
 [0.37065208 0.64698607]
 [0.24520051 0.8135823 ]
 [0.45297068 0.53385806]
 [0.38432375 0.6538129 ]
 [0.43345144 0.5627089 ]
 [0.37185562 0.67667145]
 [0.47719097 0.50528836]
 [0.39043853 0.6186756 ]
 [0.2662159  0.77654827]
 [0.37910601 0.6409492 ]
 [0.18998319 0.8655764 ]
 [0.36463326 0.68911976]
 [0.4585702  0.54296887]
 [0.4529259  0.55580103]
 [0.32811403 0.73022974]
 [0.32666573 0.7538046 ]
 [0.4097159  0.634783  ]
 [0.4712695  0.5108622 ]]
dict_keys(['loss', 'accuracy'])
```





EPOCH 500

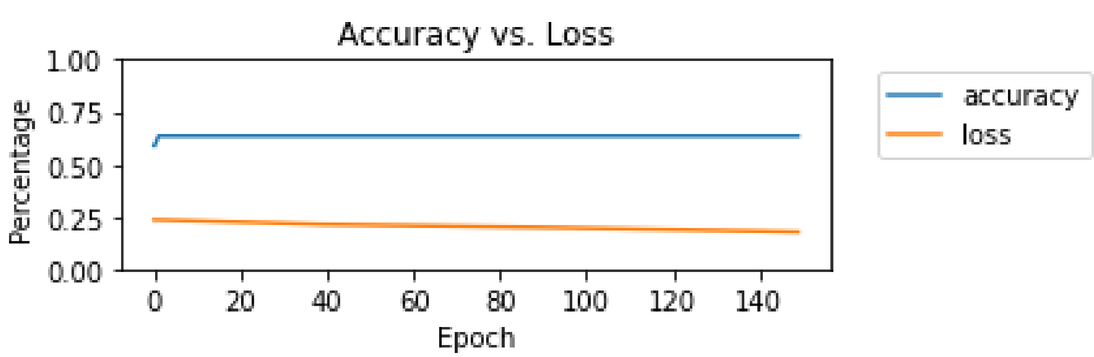
[2]:
```
model = Sequential() #A Sequential model is appropriate for a plain stack of␣
↪layers where each layer has exactly one input tensor and one output tensor
model.add(Dense(4, input_dim=5, activation='relu'))
#hidden layer: 4
#input_dim: specifying the number of elements within that first dimension only.␣
↪Initial amount of neurons
#activation: relu - > Applies the rectified linear unit activation function.␣
↪max(x, 0), the element-wise maximum of 0 and the input tensor.
model.add(Dense(3, activation='relu'))
model.add(Dense(2, activation='sigmoid'))
#activation: sigmoid -> sigmoid(x) = 1 / (1 + exp(-x)). For small values (<-5),␣
↪sigmoid returns a value close to zero, and for large values (>5) the result␣
↪of the function gets close to 1.

# Compile model
model.compile(optimizer='adam', loss='mean_squared_error',metrics=['accuracy'])
#optimizer: Adam optimization is a stochastic gradient descent method that is␣
↪based on adaptive estimation of first-order and second-order moments.
#loss: The purpose of loss functions is to compute the quantity that a model␣
↪should seek to minimize during training.
#      mean_squared_error: Average of the square of the difference between␣
↪actual and estimated values.
```

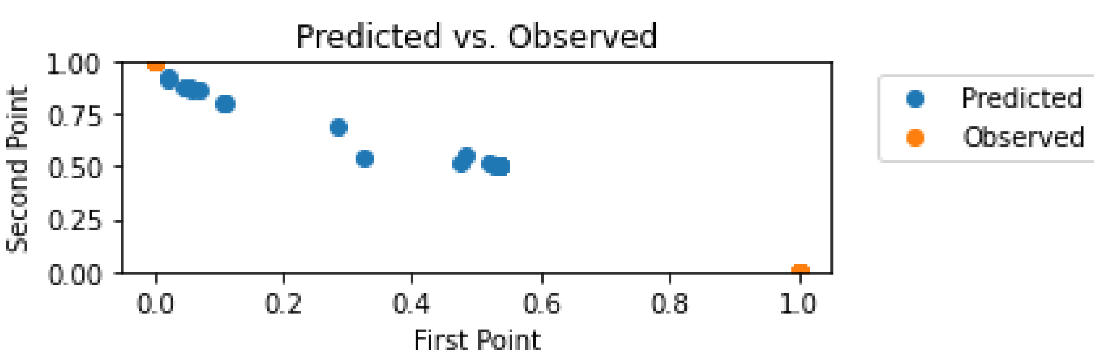
```
#metrics: accuracy: Calculates how often predictions equal labels.

#The summary is textual and includes information about:
#The layers and their order in the model.
#The output shape of each layer.
#The number of parameters (weights) in each layer.
#The total number of parameters (weights) in the model.
model.summary()


history2 = model.fit(X,Y, epochs=500, verbose=0) #Trains the model for a fixed␣
↪number of epochs (iterations on a dataset).
scores2 = model.evaluate(X,Y) #Evaluation is a process during development of␣
↪the model to check whether the model is best fit for the given problem and␣
↪corresponding data

#Plotting 1st graph which is accuracy vs loss
plt.subplot(2,1,1)
plt.plot(history.history['accuracy']) #plotting accuaray
plt.plot(history.history['loss']) #plotting loss
plt.title("Accuracy vs. Loss") #title
plt.xlabel("Epoch") #Making Epoch x- value and label
plt.ylabel("Percentage") # percentage (0.0 - 1.0)
plt.ylim((0.00,1.00)) #y limit
plt.legend(['accuracy','loss'], bbox_to_anchor=(1.05, 1.0), loc='upper left')␣
↪#Creating legend outside of graph
plt.tight_layout() #automatically adjust subplot parameters to give specified␣
↪padding

#printing the accuracy
print('\n%s: %.2f%%' % (model.metrics_names[1], scores[1]*100))

predicted_targets = model.predict(X)

#Printing predicted targets against the observed targets
for i in range(22):
    print('Predicted: ',predicted_targets[i,:],'Observed: ',Y[i,:])

#printing the predicted targets
print(predicted_targets)

#Plotting graph of predicted vs observed
plt.subplot(2,1,2)
plt.scatter(predicted_targets[:,0],predicted_targets[:,1]) #plotting predicted␣
↪points
plt.scatter(Y[:,0],Y[:,1]) #plotting observed values
plt.title("Predicted vs. Observed") #title of the graph
```

```
plt.xlabel("First Point") #x label
plt.ylabel("Second Point") #y label
plt.ylim((0.00,1.00)) #y limit
plt.legend(['Predicted','Observed'], bbox_to_anchor=(1.05, 1.0), loc='upper␣
↪left') #Creating legend outside of graph
plt.tight_layout() #automatically adjust subplot parameters to give specified␣
↪padding

print(history.history.keys())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 4)                 24
_________________________________________________________________
dense_4 (Dense)              (None, 3)                 15
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 8
=================================================================
Total params: 47
Trainable params: 47
Non-trainable params: 0
_________________________________________________________________
1/1 [==============================] - 0s 1ms/step - loss: 0.1240 - accuracy:
1.0000

accuracy: 63.64%
Predicted:  [0.10887912 0.7995602 ] Observed:  [0. 1.]
Predicted:  [0.05474904 0.87792784] Observed:  [0. 1.]
Predicted:  [0.53414446 0.5098747 ] Observed:  [1. 0.]
Predicted:  [0.5342533 0.5096788] Observed:  [1. 0.]
Predicted:  [0.04714996 0.87380004] Observed:  [0. 1.]
Predicted:  [0.02108562 0.9218284 ] Observed:  [0. 1.]
Predicted:  [0.05927566 0.8665958 ] Observed:  [0. 1.]
Predicted:  [0.28616878 0.68456054] Observed:  [0. 1.]
Predicted:  [0.48261774 0.5585737 ] Observed:  [0. 1.]
Predicted:  [0.4749964 0.518608 ] Observed:  [0. 1.]
Predicted:  [0.5342533 0.5096788] Observed:  [1. 0.]
Predicted:  [0.07029212 0.8626903 ] Observed:  [0. 1.]
Predicted:  [0.05030382 0.8702805 ] Observed:  [0. 1.]
Predicted:  [0.5281301 0.5106043] Observed:  [1. 0.]
Predicted:  [0.02253717 0.919525  ] Observed:  [0. 1.]
Predicted:  [0.05342168 0.8796252 ] Observed:  [0. 1.]
Predicted:  [0.5342533 0.5096788] Observed:  [1. 0.]
Predicted:  [0.5183224  0.51449305] Observed:  [1. 0.]
Predicted:  [0.1050995  0.80211306] Observed:  [0. 1.]
```

```
Predicted:  [0.32426497 0.5423933 ] Observed:  [0. 1.]
Predicted:  [0.5342533 0.5096788] Observed:  [1. 0.]
Predicted:  [0.5342533 0.5096788] Observed:  [1. 0.]
[[0.10887912 0.7995602 ]
 [0.05474904 0.87792784]
 [0.53414446 0.5098747 ]
 [0.5342533  0.5096788 ]
 [0.04714996 0.87380004]
 [0.02108562 0.9218284 ]
 [0.05927566 0.8665958 ]
 [0.28616878 0.68456054]
 [0.48261774 0.5585737 ]
 [0.4749964  0.518608  ]
 [0.5342533  0.5096788 ]
 [0.07029212 0.8626903 ]
 [0.05030382 0.8702805 ]
 [0.5281301  0.5106043 ]
 [0.02253717 0.919525  ]
 [0.05342168 0.8796252 ]
 [0.5342533  0.5096788 ]
 [0.5183224  0.51449305]
 [0.1050995  0.80211306]
 [0.32426497 0.5423933 ]
 [0.5342533  0.5096788 ]
 [0.5342533  0.5096788 ]]
dict_keys(['loss', 'accuracy'])
```





EPOCH 1500

[3]:
```
model = Sequential() #A Sequential model is appropriate for a plain stack of␣
 ↪layers where each layer has exactly one input tensor and one output tensor
model.add(Dense(4, input_dim=5, activation='relu'))
#hidden layer: 4
#input_dim: specifying the number of elements within that first dimension only.␣
 ↪Initial amount of neurons
#activation: relu - > Applies the rectified linear unit activation function.␣
 ↪max(x, 0), the element-wise maximum of 0 and the input tensor.
model.add(Dense(3, activation='relu'))
model.add(Dense(2, activation='sigmoid'))
#activation: sigmoid -> sigmoid(x) = 1 / (1 + exp(-x)). For small values (<-5),␣
 ↪sigmoid returns a value close to zero, and for large values (>5) the result␣
 ↪of the function gets close to 1.

# Compile model
model.compile(optimizer='adam', loss='mean_squared_error',metrics=['accuracy'])
#optimizer: Adam optimization is a stochastic gradient descent method that is␣
 ↪based on adaptive estimation of first-order and second-order moments.
#loss: The purpose of loss functions is to compute the quantity that a model␣
 ↪should seek to minimize during training.
#      mean_squared_error: Average of the square of the difference between␣
 ↪actual and estimated values.
#metrics: accuracy: Calculates how often predictions equal labels.

#The summary is textual and includes information about:
#The layers and their order in the model.
#The output shape of each layer.
#The number of parameters (weights) in each layer.
#The total number of parameters (weights) in the model.
model.summary()


history3 = model.fit(X,Y, epochs=1500, verbose=0) #Trains the model for a fixed␣
 ↪number of epochs (iterations on a dataset).
scores3 = model.evaluate(X,Y) #Evaluation is a process during development of␣
 ↪the model to check whether the model is best fit for the given problem and␣
 ↪corresponding data

#Plotting 1st graph which is accuracy vs loss
plt.subplot(2,1,1)
plt.plot(history.history['accuracy']) #plotting accuaray
plt.plot(history.history['loss']) #plotting loss
```

```python
plt.title("Accuracy vs. Loss") #title
plt.xlabel("Epoch") #Making Epoch x- value and label
plt.ylabel("Percentage") # percentage (0.0 - 1.0)
plt.ylim((0.00,1.00)) #y limit
plt.legend(['accuracy','loss'], bbox_to_anchor=(1.05, 1.0), loc='upper left')␣
 ↪#Creating legend outside of graph
plt.tight_layout() #automatically adjust subplot parameters to give specified␣
 ↪padding

#printing the accuracy
print('\n%s: %.2f%%' % (model.metrics_names[1], scores[1]*100))

predicted_targets = model.predict(X)

#Printing predicted targets against the observed targets
for i in range(22):
    print('Predicted: ',predicted_targets[i,:],'Observed: ',Y[i,:])

#printing the predicted targets
print(predicted_targets)

#Plotting graph of predicted vs observed
plt.subplot(2,1,2)
plt.scatter(predicted_targets[:,0],predicted_targets[:,1]) #plotting predicted␣
 ↪points
plt.scatter(Y[:,0],Y[:,1]) #plotting observed values
plt.title("Predicted vs. Observed") #title of the graph
plt.xlabel("First Point") #x label
plt.ylabel("Second Point") #y label
plt.ylim((0.00,1.00)) #y limit
plt.legend(['Predicted','Observed'], bbox_to_anchor=(1.05, 1.0), loc='upper␣
 ↪left') #Creating legend outside of graph
plt.tight_layout() #automatically adjust subplot parameters to give specified␣
 ↪padding

print(history.history.keys())
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 4)                 24
_________________________________________________________________
dense_7 (Dense)              (None, 3)                 15
_________________________________________________________________
dense_8 (Dense)              (None, 2)                 8
=================================================================
```

```
Total params: 47
Trainable params: 47
Non-trainable params: 0
_________________________________________________________________
1/1 [==============================] - 0s 1ms/step - loss: 0.0269 - accuracy:
1.0000

accuracy: 63.64%
Predicted:  [0.20726132 0.79716516] Observed:  [0. 1.]
Predicted:  [0.20726132 0.79716516] Observed:  [0. 1.]
Predicted:  [0.988214   0.02734095] Observed:  [1. 0.]
Predicted:  [0.9885967  0.02673069] Observed:  [1. 0.]
Predicted:  [0.20726132 0.79716516] Observed:  [0. 1.]
Predicted:  [0.20726132 0.79716516] Observed:  [0. 1.]
Predicted:  [0.20726132 0.79716516] Observed:  [0. 1.]
Predicted:  [0.20726132 0.79716516] Observed:  [0. 1.]
Predicted:  [0.20726132 0.79716516] Observed:  [0. 1.]
Predicted:  [0.20726132 0.79716516] Observed:  [0. 1.]
Predicted:  [0.9869437  0.02869537] Observed:  [1. 0.]
Predicted:  [0.20726132 0.79716516] Observed:  [0. 1.]
Predicted:  [0.20726132 0.79716516] Observed:  [0. 1.]
Predicted:  [0.9880882  0.02766582] Observed:  [1. 0.]
Predicted:  [0.20726132 0.79716516] Observed:  [0. 1.]
Predicted:  [0.20726132 0.79716516] Observed:  [0. 1.]
Predicted:  [0.9865585  0.02923307] Observed:  [1. 0.]
Predicted:  [0.9861835  0.02988473] Observed:  [1. 0.]
Predicted:  [0.20726132 0.79716516] Observed:  [0. 1.]
Predicted:  [0.20726132 0.79716516] Observed:  [0. 1.]
Predicted:  [0.98610836 0.0298987 ] Observed:  [1. 0.]
Predicted:  [0.98892415 0.02623782] Observed:  [1. 0.]
[[0.20726132 0.79716516]
 [0.20726132 0.79716516]
 [0.988214   0.02734095]
 [0.9885967  0.02673069]
 [0.20726132 0.79716516]
 [0.20726132 0.79716516]
 [0.20726132 0.79716516]
 [0.20726132 0.79716516]
 [0.20726132 0.79716516]
 [0.20726132 0.79716516]
 [0.9869437  0.02869537]
 [0.20726132 0.79716516]
 [0.20726132 0.79716516]
 [0.9880882  0.02766582]
 [0.20726132 0.79716516]
 [0.20726132 0.79716516]
 [0.9865585  0.02923307]
 [0.9861835  0.02988473]
```

```
 [0.20726132 0.79716516]
 [0.20726132 0.79716516]
 [0.98610836 0.0298987 ]
 [0.98892415 0.02623782]]
dict_keys(['loss', 'accuracy'])
```