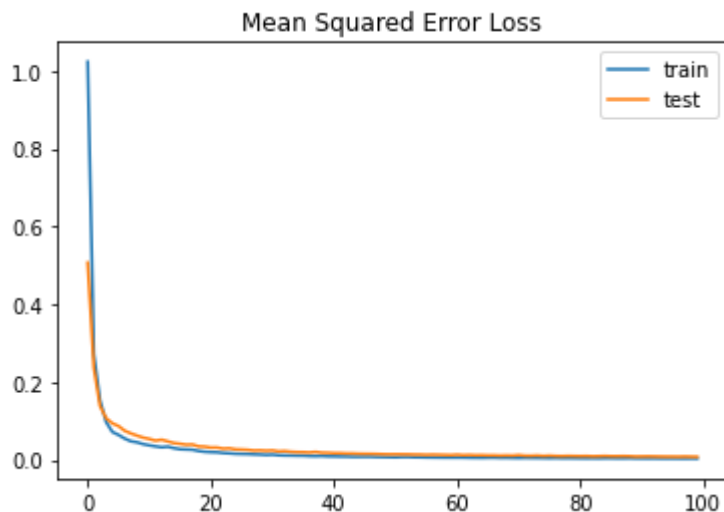


```
1 # mlp for regression with mse loss function
2 from sklearn.datasets import make_regression
3 from sklearn.preprocessing import StandardScaler
4 from tensorflow.keras.models import Sequential
5 from tensorflow.keras.layers import Dense
6 from tensorflow.keras.optimizers import SGD
7 from matplotlib import pyplot
8 # generate regression dataset
9 X, y = make_regression(n_samples=1000, n_features=20, noise=0.1, random_state=
10 # standardize dataset
11 X = StandardScaler().fit_transform(X)
12 y = StandardScaler().fit_transform(y.reshape(len(y),1))[:,0]
13 # split into train and test
14 n_train = 500
15 trainX, testX = X[:n_train, :], X[n_train:, :]
16 trainy, testy = y[:n_train], y[n_train:]
17 # define model
18 model = Sequential()
19 model.add(Dense(25, input_dim=20, activation='relu', kernel_initializer='he_un
20 model.add(Dense(1, activation='linear'))
21 opt = SGD(lr=0.01, momentum=0.9)
22
23 model.compile(loss='mean_squared_error', optimizer=opt)
24 # fit model
25 history = model.fit(trainX, trainy, validation_data=(testX, testy), epochs=100,
26 # evaluate the model
27 train_mse = model.evaluate(trainX, trainy, verbose=0)
28 test_mse = model.evaluate(testX, testy, verbose=0)
29 print('Train: %.3f, Test: %.3f' % (train_mse, test_mse))
30 # plot loss during training
31
32 pyplot.title('Mean Squared Error Loss')
33 pyplot.plot(history.history['loss'], label='train')
34 pyplot.plot(history.history['val_loss'], label='test')
35 pyplot.legend()
36 pyplot.show()
```



```
Epoch 80/100
16/16 [=====] - 0s 4ms/step - loss: 0.0039 - val_loss
Epoch 81/100
16/16 [=====] - 0s 4ms/step - loss: 0.0038 - val_loss
Epoch 82/100
16/16 [=====] - 0s 4ms/step - loss: 0.0036 - val_loss
Epoch 83/100
16/16 [=====] - 0s 4ms/step - loss: 0.0037 - val_loss
Epoch 84/100
16/16 [=====] - 0s 4ms/step - loss: 0.0036 - val_loss
Epoch 85/100
16/16 [=====] - 0s 4ms/step - loss: 0.0039 - val_loss
Epoch 86/100
16/16 [=====] - 0s 4ms/step - loss: 0.0039 - val_loss
Epoch 87/100
16/16 [=====] - 0s 4ms/step - loss: 0.0037 - val_loss
Epoch 88/100
16/16 [=====] - 0s 4ms/step - loss: 0.0036 - val_loss
Epoch 89/100
16/16 [=====] - 0s 4ms/step - loss: 0.0039 - val_loss
Epoch 90/100
16/16 [=====] - 0s 4ms/step - loss: 0.0035 - val_loss
Epoch 91/100
16/16 [=====] - 0s 4ms/step - loss: 0.0034 - val_loss
Epoch 92/100
16/16 [=====] - 0s 4ms/step - loss: 0.0033 - val_loss
Epoch 93/100
16/16 [=====] - 0s 4ms/step - loss: 0.0034 - val_loss
Epoch 94/100
16/16 [=====] - 0s 4ms/step - loss: 0.0032 - val_loss
Epoch 95/100
16/16 [=====] - 0s 4ms/step - loss: 0.0032 - val_loss
Epoch 96/100
16/16 [=====] - 0s 4ms/step - loss: 0.0032 - val_loss
Epoch 97/100
16/16 [=====] - 0s 4ms/step - loss: 0.0031 - val_loss
Epoch 98/100
16/16 [=====] - 0s 4ms/step - loss: 0.0031 - val_loss
Epoch 99/100
16/16 [=====] - 0s 4ms/step - loss: 0.0032 - val_loss
Epoch 100/100
16/16 [=====] - 0s 4ms/step - loss: 0.0031 - val_loss
Train: 0.003, Test: 0.007
```



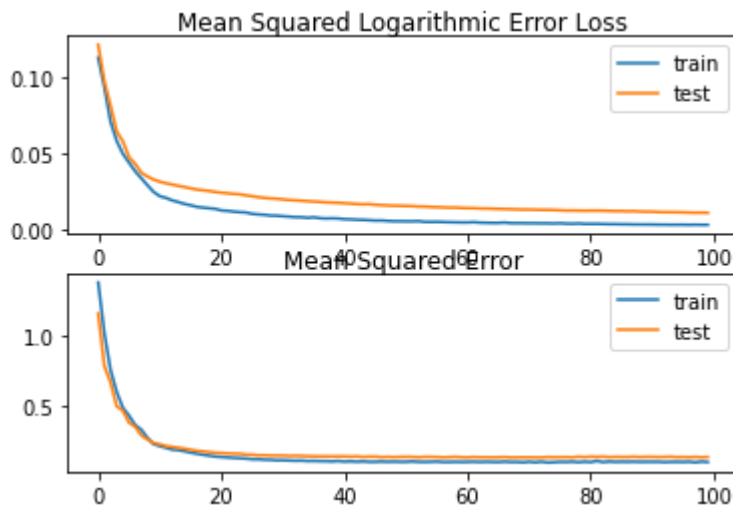

```
1 # mlp for regression with msle loss function
2 from sklearn.datasets import make_regression
3 from sklearn.preprocessing import StandardScaler
4 from tensorflow.keras.models import Sequential
5 from tensorflow.keras.layers import Dense
6 from tensorflow.keras.optimizers import SGD
7 from matplotlib import pyplot
8 # generate regression dataset
9 X, y = make_regression(n_samples=1000, n_features=20, noise=0.1, random_state=
10 # standardize dataset
11 X = StandardScaler().fit_transform(X)
12 y = StandardScaler().fit_transform(y.reshape(len(y),1))[:,0]
13 # split into train and test
14 n_train = 500
15 trainX, testX = X[:n_train, :], X[n_train:, :]
16 trainy, testy = y[:n_train], y[n_train:]
17 # define model
18 model = Sequential()
19 model.add(Dense(25, input_dim=20, activation='relu', kernel_initializer='he_un
20 model.add(Dense(1, activation='linear'))
21 opt = SGD(lr=0.01, momentum=0.9)
22 model.compile(loss='mean_squared_logarithmic_error', optimizer=opt, metrics=['
```

```

23 # fit model
24 history = model.fit(trainX, trainy, validation_data=(testX, testy), epochs=100,
25 # evaluate the model
26 _, train_mse = model.evaluate(trainX, trainy, verbose=0)
27 _, test_mse = model.evaluate(testX, testy, verbose=0)
28 print('Train: %.3f, Test: %.3f' % (train_mse, test_mse))
29 # plot loss during training
30 pyplot.subplot(211)
31 pyplot.title('Mean Squared Logarithmic Error Loss', pad=-20)
32 pyplot.plot(history.history['loss'], label='train')
33 pyplot.plot(history.history['val_loss'], label='test')
34 pyplot.legend()
35 # plot mse during training
36 pyplot.subplot(212)
37 pyplot.title('Mean Squared Error', pad=-20)
38 pyplot.plot(history.history['mse'], label='train')
39 pyplot.plot(history.history['val_mse'], label='test')
40 pyplot.legend()
41 pyplot.show()

```

☞ Train: 0.110, Test: 0.145



```

1 model.compile(loss='mean_absolute_error', optimizer=opt, metrics=['mse'])
2 # fit model
3 history = model.fit(trainX, trainy, validation_data=(testX, testy), epochs=100,
4 # evaluate the model

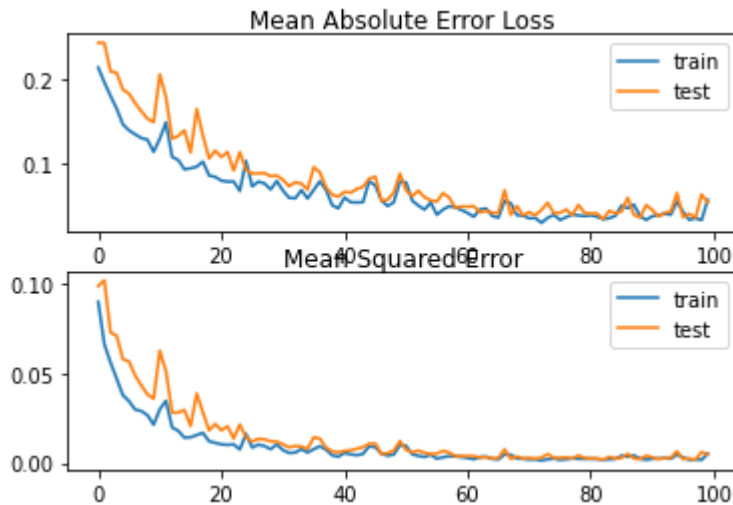
```

```

5  _, train_mse = model.evaluate(trainX, trainy, verbose=0)
6  _, test_mse = model.evaluate(testX, testy, verbose=0)
7  print('Train: %.3f, Test: %.3f' % (train_mse, test_mse))
8  # plot loss during training
9  pyplot.subplot(211)
10 pyplot.title('Mean Absolute Error Loss', pad=-20)
11 pyplot.plot(history.history['loss'], label='train')
12 pyplot.plot(history.history['val_loss'], label='test')
13 pyplot.legend()
14 # plot mse during training
15 pyplot.subplot(212)
16 pyplot.title('Mean Squared Error', pad=-20)
17 pyplot.plot(history.history['mse'], label='train')
18 pyplot.plot(history.history['val_mse'], label='test')
19 pyplot.legend()
20 pyplot.show()

```

☞ Train: 0.004, Test: 0.005



▼ Binary Classification Loss Functions Case Study

```

1  # mlp for the circles problem with cross-entropy loss
2  from sklearn.datasets import make_circles
3  from tensorflow.keras.models import Sequential
4  from tensorflow.keras.layers import Dense
5  from tensorflow.keras.optimizers import SGD
6  from matplotlib import pyplot
7  # generate 2d classification dataset
8  X, y = make_circles(n_samples=1000, noise=0.1, random_state=1)
9  # split into train and test
10 n_train = 500
11 trainX, testX = X[:n_train, :], X[n_train:, :]
12 trainy, testy = y[:n_train], y[n_train:]
13 # define model
14 model = Sequential()
15 model.add(Dense(50, input_dim=2, activation='relu', kernel_initializer='he_uni
16 model.add(Dense(1, activation='sigmoid'))
17 opt = SGD(lr=0.01, momentum=0.9)

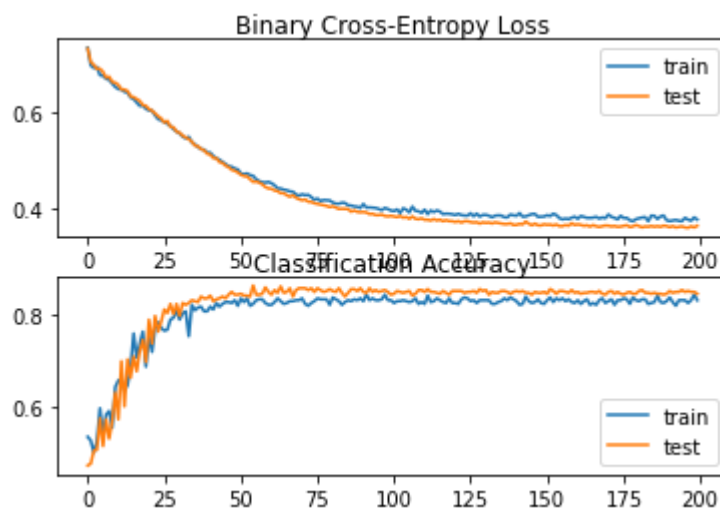
```

```

18 model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
19 # fit model
20 history = model.fit(trainX, trainy, validation_data=(testX, testy), epochs=200,
21 # evaluate the model
22 _, train_acc = model.evaluate(trainX, trainy, verbose=0)
23 _, test_acc = model.evaluate(testX, testy, verbose=0)
24 print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
25 # plot loss during training
26 pyplot.subplot(211)
27 pyplot.title('Binary Cross-Entropy Loss', pad=-20)
28 pyplot.plot(history.history['loss'], label='train')
29 pyplot.plot(history.history['val_loss'], label='test')
30 pyplot.legend()
31 # plot accuracy during training
32 pyplot.subplot(212)
33 pyplot.title('Classification Accuracy', pad=-40)
34 pyplot.plot(history.history['accuracy'], label='train')
35 pyplot.plot(history.history['val_accuracy'], label='test')
36 pyplot.legend()
37 pyplot.show()

```

☞ Train: 0.836, Test: 0.844



```

1 # mlp for the circles problem with hinge loss
2 from sklearn.datasets import make_circles
3 from tensorflow.keras.models import Sequential
4 from tensorflow.keras.layers import Dense

```

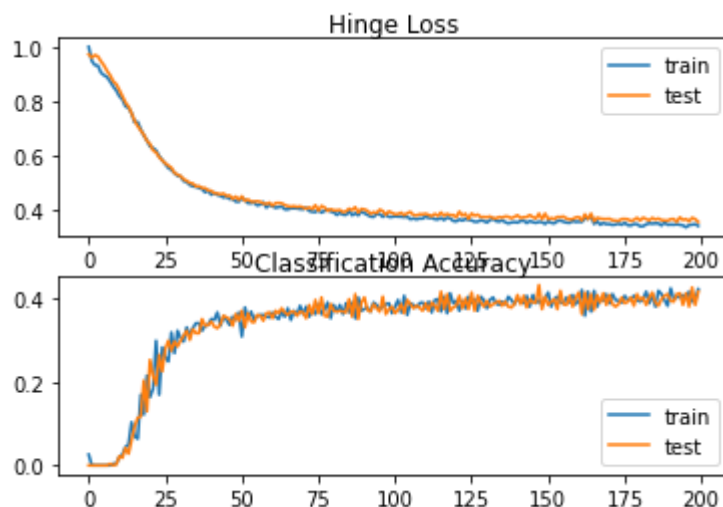


```

5  from tensorflow.keras.optimizers import SGD
6  from matplotlib import pyplot
7  from numpy import where
8  # generate 2d classification dataset
9  X, y = make_circles(n_samples=1000, noise=0.1, random_state=1)
10 # change y from {0,1} to {-1,1}
11 y[where(y == 0)] = -1
12 # split into train and test
13 n_train = 500
14 trainX, testX = X[:n_train, :], X[n_train:, :]
15 trainy, testy = y[:n_train], y[n_train:]
16 # define model
17 model = Sequential()
18 model.add(Dense(50, input_dim=2, activation='relu', kernel_initializer='he_uni
19 model.add(Dense(1, activation='tanh'))
20 opt = SGD(lr=0.01, momentum=0.9)
21 model.compile(loss='hinge', optimizer=opt, metrics=['accuracy'])
22 # fit model
23 history = model.fit(trainX, trainy, validation_data=(testX, testy), epochs=200,
24 # evaluate the model
25 _, train_acc = model.evaluate(trainX, trainy, verbose=0)
26 _, test_acc = model.evaluate(testX, testy, verbose=0)
27 print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
28 # plot loss during training
29 pyplot.subplot(211)
30 pyplot.title('Hinge Loss', pad=-20)
31 pyplot.plot(history.history['loss'], label='train')
32 pyplot.plot(history.history['val_loss'], label='test')
33 pyplot.legend()
34 # plot accuracy during training
35 pyplot.subplot(212)
36 pyplot.title('Classification Accuracy', pad=-40)
37 pyplot.plot(history.history['accuracy'], label='train')
38 pyplot.plot(history.history['val_accuracy'], label='test')
39 pyplot.legend()
40 pyplot.show()

```

☞ Train: 0.416, Test: 0.418

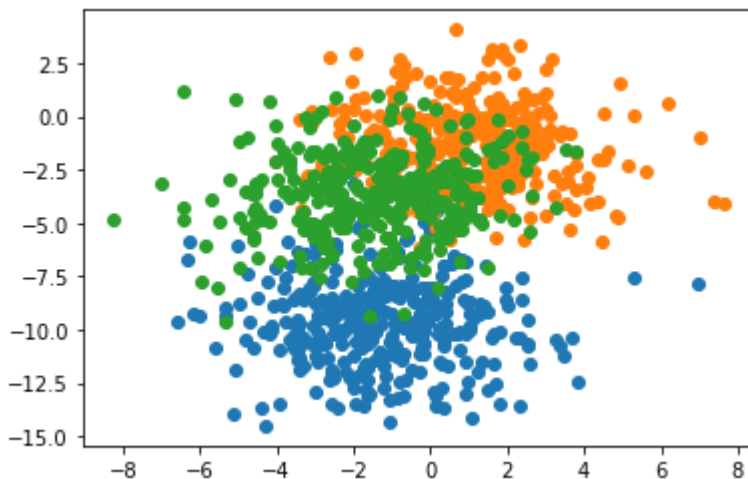


▼ Multiclass Classification Loss Functions Case Study

```

1  # scatter plot of blobs dataset
2  from sklearn.datasets.samples_generator import make_blobs
3  from numpy import where
4  from matplotlib import pyplot
5  # generate dataset
6  X, y = make_blobs(n_samples=1000, centers=3, n_features=2, cluster_std=2, rand
7  # select indices of points with each class label
8  for i in range(3):
9      samples_ix = where(y == i)
10     pyplot.scatter(X[samples_ix, 0], X[samples_ix, 1])
11 pyplot.show()
12

```



```

1  # mlp for the blobs multi-class classification problem with cross-entropy loss
2  from sklearn.datasets.samples_generator import make_blobs
3  from tensorflow.keras.layers import Dense
4  from tensorflow.keras.models import Sequential
5  from tensorflow.keras.optimizers import SGD
6  from tensorflow.keras.utils import to_categorical
7  from matplotlib import pyplot
8  # generate 2d classification dataset
9  X, y = make_blobs(n_samples=1000, centers=3, n_features=2, cluster_std=2, rand
10 # one hot encode output variable
11 y = to_categorical(y)
12 # split into train and test
13 n_train = 500
14 trainX, testX = X[:n_train, :], X[n_train:, :]
15 trainy, testy = y[:n_train], y[n_train:]
16 # define model
17 model = Sequential()
18 model.add(Dense(50, input_dim=2, activation='relu', kernel_initializer='he_uni
19 model.add(Dense(3, activation='softmax'))
20 # compile model
21 opt = SGD(lr=0.01, momentum=0.9)
22 model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accura
23 # fit model
24 history = model.fit(trainX, trainy, validation_data=(testX, testy), epochs=100,
25 # evaluate the model
26 _, train_acc = model.evaluate(trainX, trainy, verbose=0)
27 test_acc = model.evaluate(testX, testy, verbose=0)

```