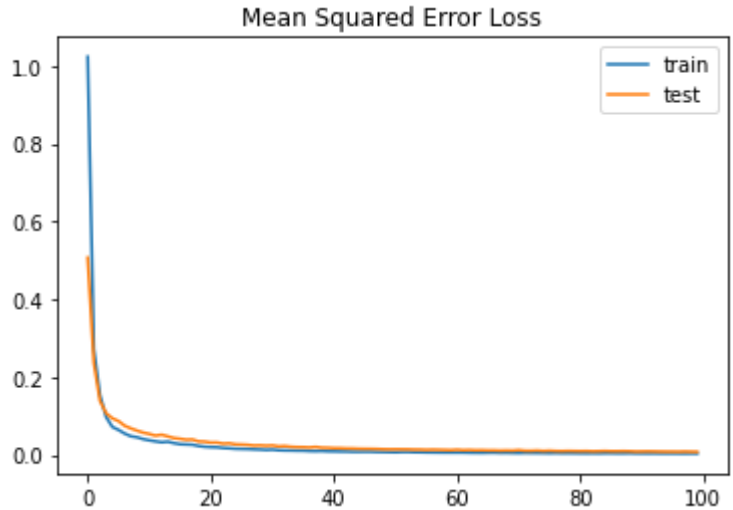
```
# mlp for regression with mse loss function
from sklearn.datasets import make_regression
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from matplotlib import pyplot
# generate regression dataset
X, y = make_regression(n_samples=1000, n_features=20, noise=0.1, random_state=
# standardize dataset
X = StandardScaler().fit_transform(X)
y = StandardScaler().fit_transform(y.reshape(len(y),1))[:,0]
# split into train and test
n_train = 500
trainX, testX = X[:n_train, :], X[n_train:, :]
trainy, testy = y[:n_train], y[n_train:]
# define model
model = Sequential()
model.add(Dense(25, input_dim=20, activation='relu', kernel_initializer='he_un
model.add(Dense(1, activation='linear'))
opt = SGD(lr=0.01, momentum=0.9)

model.compile(loss='mean_squared_error', optimizer=opt)
# fit model
history = model.fit(trainX, trainy, validation_data=(testX, testy), epochs=100,
# evaluate the model
train_mse = model.evaluate(trainX, trainy, verbose=0)
test_mse = model.evaluate(testX, testy, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_mse, test_mse))
# plot loss during training

pyplot.title('Mean Squared Error Loss')
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```

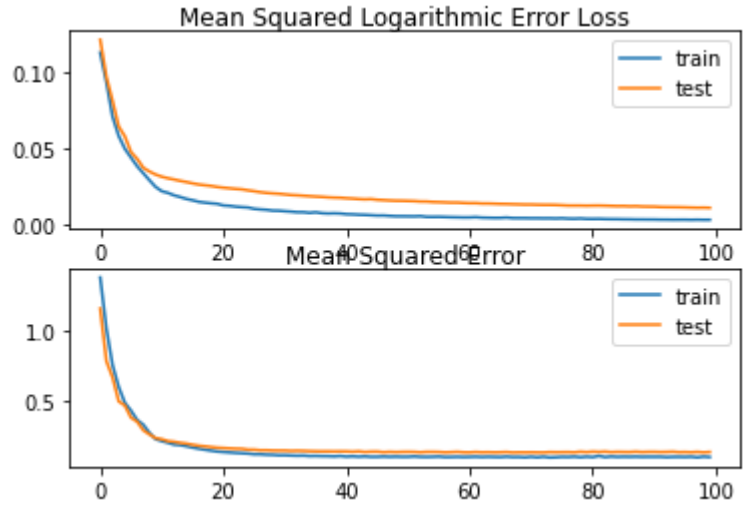
```
Epoch 80/100
16/16 [==============================] - 0s 4ms/step - loss: 0.0039 - val_loss
Epoch 81/100
16/16 [==============================] - 0s 4ms/step - loss: 0.0038 - val_loss
Epoch 82/100
16/16 [==============================] - 0s 4ms/step - loss: 0.0036 - val_loss
Epoch 83/100
16/16 [==============================] - 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
```



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

```
# fit model
history = model.fit(trainX, trainy, validation_data=(testX, testy), epochs=100,
# evaluate the model
_, train_mse = model.evaluate(trainX, trainy, verbose=0)
_, test_mse = model.evaluate(testX, testy, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_mse, test_mse))
# plot loss during training
pyplot.subplot(211)
pyplot.title('Mean Squared Logarithmic Error Loss', pad=-20)
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
# plot mse during training
pyplot.subplot(212)
pyplot.title('Mean Squared Error', pad=-20)
pyplot.plot(history.history['mse'], label='train')
pyplot.plot(history.history['val_mse'], label='test')
pyplot.legend()
pyplot.show()
```

```
Train: 0.110, Test: 0.145
```




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

```
_, train_mse = model.evaluate(trainX, trainy, verbose=0)
_, test_mse = model.evaluate(testX, testy, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_mse, test_mse))
# plot loss during training
pyplot.subplot(211)
pyplot.title('Mean Absolute Error Loss', pad=-20)
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
# plot mse during training
pyplot.subplot(212)
pyplot.title('Mean Squared Error', pad=-20)
pyplot.plot(history.history['mse'], label='train')
pyplot.plot(history.history['val_mse'], label='test')
pyplot.legend()
pyplot.show()
```

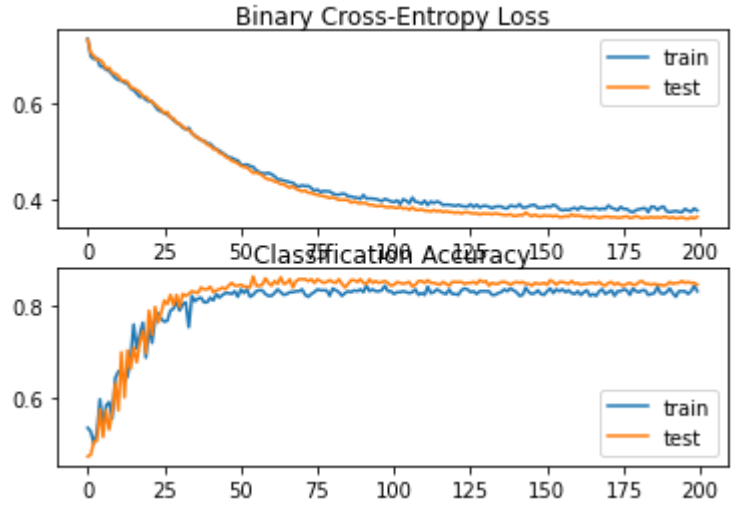
```
Train: 0.004, Test: 0.005
```

Binary Classification Loss Functions Case Study

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

```
model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
# fit model
history = model.fit(trainX, trainy, validation_data=(testX, testy), epochs=200,
# evaluate the model
_, train_acc = model.evaluate(trainX, trainy, verbose=0)
_, test_acc = model.evaluate(testX, testy, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
# plot loss during training
pyplot.subplot(211)
pyplot.title('Binary Cross-Entropy Loss', pad=-20)
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
# plot accuracy during training
pyplot.subplot(212)
pyplot.title('Classification Accuracy', pad=-40)
pyplot.plot(history.history['accuracy'], label='train')
pyplot.plot(history.history['val_accuracy'], label='test')
pyplot.legend()
pyplot.show()
```

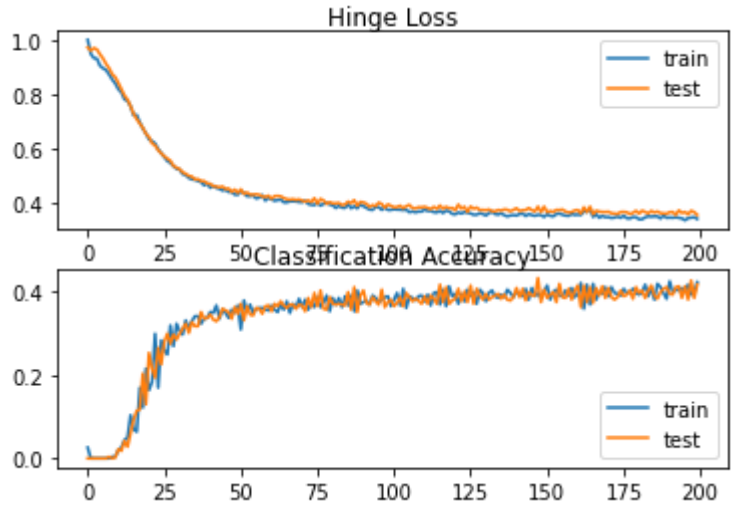
```
Train: 0.836, Test: 0.844
```




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

```
from tensorflow.keras.optimizers import SGD
from matplotlib import pyplot
from numpy import where
# generate 2d classification dataset
X, y = make_circles(n_samples=1000, noise=0.1, random_state=1)
# change y from {0,1} to {-1,1}
y[where(y == 0)] = -1
# split into train and test
n_train = 500
trainX, testX = X[:n_train, :], X[n_train:, :]
trainy, testy = y[:n_train], y[n_train:]
# define model
model = Sequential()
model.add(Dense(50, input_dim=2, activation='relu', kernel_initializer='he_uni
model.add(Dense(1, activation='tanh'))
opt = SGD(lr=0.01, momentum=0.9)
model.compile(loss='hinge', optimizer=opt, metrics=['accuracy'])
# fit model
history = model.fit(trainX, trainy, validation_data=(testX, testy), epochs=200,
# evaluate the model
_, train_acc = model.evaluate(trainX, trainy, verbose=0)
_, test_acc = model.evaluate(testX, testy, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
# plot loss during training
pyplot.subplot(211)
pyplot.title('Hinge Loss', pad=-20)
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
# plot accuracy during training
pyplot.subplot(212)
pyplot.title('Classification Accuracy', pad=-40)
pyplot.plot(history.history['accuracy'], label='train')
pyplot.plot(history.history['val_accuracy'], label='test')
pyplot.legend()
pyplot.show()
```

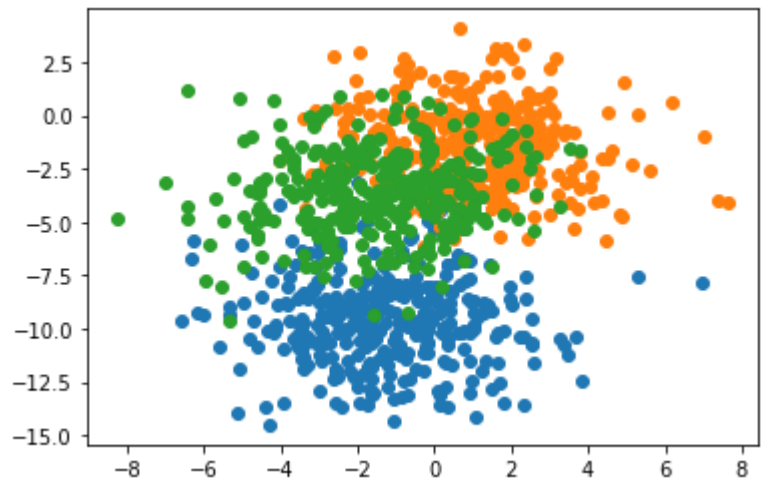
```
Train: 0.416, Test: 0.418
```



Multiclass Classication Loss Functions Case Study

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

```




```
# mlp for the blobs multi-class classification problem with cross-entropy loss
from sklearn.datasets.samples_generator import make_blobs
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.utils import to_categorical
from matplotlib import pyplot
# generate 2d classification dataset
X, y = make_blobs(n_samples=1000, centers=3, n_features=2, cluster_std=2, rand
# one hot encode output variable
y = to_categorical(y)
# split into train and test
n_train = 500
trainX, testX = X[:n_train, :], X[n_train:, :]
trainy, testy = y[:n_train], y[n_train:]
# define model
model = Sequential()
model.add(Dense(50, input_dim=2, activation='relu', kernel_initializer='he_uni
model.add(Dense(3, activation='softmax'))
# compile model
opt = SGD(lr=0.01, momentum=0.9)
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accura
# fit model
history = model.fit(trainX, trainy, validation_data=(testX, testy), epochs=100,
# evaluate the model
_, train_acc = model.evaluate(trainX, trainy, verbose=0)
    test_acc = model.evaluate(testX, testy, verbose=0)
```