**Day 9**

**What to do?**

Learn about loss functions and how loss is calculated.

**Loss function:**

The main objective of any ML/DL algorithm is to choose a model with low cost/loss/error. In NN, the model is evaluated based on a loss function. While model error is calculated, a loss function must be chosen. As there are various models in ML (regression and classification), each type has its own set of loss functions.

**Regression Loss functions:**

1. **MSE:**

Default loss function for all regression algorithms. Usually chosen when the target variable is Gaussian. It is calculated as the average of squared differences between predicted and actual values. The most optimal value is 0.0 (lowest MSE).

# mlp for regression with mse loss function

from sklearn.datasets import make\_regression

from sklearn.preprocessing import StandardScaler

from keras.models import Sequential

from keras.layers import Dense

from 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=1)

# 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\_uniform'))

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, verbose=0)

# 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('Loss / Mean Squared Error')

pyplot.plot(history.history['loss'], label='train')

pyplot.plot(history.history['val\_loss'], label='test')

pyplot.legend()

pyplot.show()

1. **MSLE:**

Used when target variable has large values and are spread. First calculate the logarithm of each predicted value and then calculate the mean squared error (MSE). This is called as mean squared log error.

# mlp for regression with msle loss function

from sklearn.datasets import make\_regression

from sklearn.preprocessing import StandardScaler

from keras.models import Sequential

from keras.layers import Dense

from 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=1)

# 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\_uniform'))

model.add(Dense(1, activation='linear'))

opt = SGD(lr=0.01, momentum=0.9)

model.compile(loss='mean\_squared\_logarithmic\_error', optimizer=opt, metrics=['mse'])

# fit model

history = model.fit(trainX, trainy, validation\_data=(testX, testy), epochs=100, verbose=0)

# 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('Loss')

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')

pyplot.plot(history.history['mean\_squared\_error'], label='train')

pyplot.plot(history.history['val\_mean\_squared\_error'], label='test')

pyplot.legend()

pyplot.show()

1. **MAE:**

Mean absolute error is qualified to use when the target variable is both Gaussian and have outliers. It is calculated as the average of absolute difference between predicted and actual values.

# mlp for regression with mae loss function

from sklearn.datasets import make\_regression

from sklearn.preprocessing import StandardScaler

from keras.models import Sequential

from keras.layers import Dense

from 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=1)

# 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\_uniform'))

model.add(Dense(1, activation='linear'))

opt = SGD(lr=0.01, momentum=0.9)

model.compile(loss='mean\_absolute\_error', optimizer=opt, metrics=['mse'])

# fit model

history = model.fit(trainX, trainy, validation\_data=(testX, testy), epochs=100, verbose=0)

# 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('Loss')

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')

pyplot.plot(history.history['mean\_squared\_error'], label='train')

pyplot.plot(history.history['val\_mean\_squared\_error'], label='test')

pyplot.legend()

pyplot.show()

**Binary Classification loss functions:**

1. **Binary cross entropy:**

It is the default loss function used for binary classification. Cross entropy calculates score that summarizes the difference between actual and predicted probability distribution for positive class. Here, the target values are in the set {0, 1}.

# mlp for the circles problem with cross entropy loss

from sklearn.datasets import make\_circles

from keras.models import Sequential

from keras.layers import Dense

from 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\_uniform'))

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, verbose=0)

# 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('Loss')

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('Accuracy')

pyplot.plot(history.history['accuracy'], label='train')

pyplot.plot(history.history['val\_accuracy'], label='test')

pyplot.legend()

pyplot.show()

1. **Hinge loss:**

This loss function is an alternative for cross-entropy loss, designed specifically for SVMs. Here, the target values are in the set {-1, 1}. This function assigns more error when there’s difference in sign between predicted and actual values.

# mlp for the circles problem with hinge loss

from sklearn.datasets import make\_circles

from keras.models import Sequential

from keras.layers import Dense

from 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\_uniform'))

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, verbose=0)

# 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('Loss')

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('Accuracy')

pyplot.plot(history.history['accuracy'], label='train')

pyplot.plot(history.history['val\_accuracy'], label='test')

pyplot.legend()

pyplot.show()

1. **Square hinge loss:**

Like hinge loss, but the additional step is to square the hinge loss.

# mlp for the circles problem with squared hinge loss

from sklearn.datasets import make\_circles

from keras.models import Sequential

from keras.layers import Dense

from 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\_uniform'))

model.add(Dense(1, activation='tanh'))

opt = SGD(lr=0.01, momentum=0.9)

model.compile(loss='squared\_hinge', optimizer=opt, metrics=['accuracy'])

# fit model

history = model.fit(trainX, trainy, validation\_data=(testX, testy), epochs=200, verbose=0)

# 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('Loss')

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('Accuracy')

pyplot.plot(history.history['accuracy'], label='train')

pyplot.plot(history.history['val\_accuracy'], label='test')

pyplot.legend()

pyplot.show()

**Multi-Class Classification loss functions:**

1. **Multi-class cross-entropy loss:**

It is default for multi-class classification problems that may have n classes. It is calculated in the same way as binary cross-entropy loss.

# mlp for the blobs multi-class classification problem with cross-entropy loss

from sklearn.datasets import make\_blobs

from keras.layers import Dense

from keras.models import Sequential

from keras.optimizers import SGD

from 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, random\_state=2)

# 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\_uniform'))

model.add(Dense(3, activation='softmax'))

# compile model

opt = SGD(lr=0.01, momentum=0.9)

model.compile(loss='categorical\_crossentropy', optimizer=opt, metrics=['accuracy'])

# fit model

history = model.fit(trainX, trainy, validation\_data=(testX, testy), epochs=100, verbose=0)

# 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('Loss')

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('Accuracy')

pyplot.plot(history.history['accuracy'], label='train')

pyplot.plot(history.history['val\_accuracy'], label='test')

pyplot.legend()

pyplot.show()

1. **Sparse multi-class cross-entropy loss:**

This loss function is used when you have hundreds/thousands of classes (vocabulary words), which also does not require one-hot encoding of target variable, and moreover, cross-entropy loss may suffer from significant memory usage.

# mlp for the blobs multi-class classification problem with sparse cross-entropy loss

from sklearn.datasets import make\_blobs

from keras.layers import Dense

from keras.models import Sequential

from keras.optimizers import SGD

from matplotlib import pyplot

# generate 2d classification dataset

X, y = make\_blobs(n\_samples=1000, centers=3, n\_features=2, cluster\_std=2, random\_state=2)

# 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\_uniform'))

model.add(Dense(3, activation='softmax'))

# compile model

opt = SGD(lr=0.01, momentum=0.9)

model.compile(loss='sparse\_categorical\_crossentropy', optimizer=opt, metrics=['accuracy'])

# fit model

history = model.fit(trainX, trainy, validation\_data=(testX, testy), epochs=100, verbose=0)

# 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('Loss')

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('Accuracy')

pyplot.plot(history.history['accuracy'], label='train')

pyplot.plot(history.history['val\_accuracy'], label='test')

pyplot.legend()

pyplot.show()