Backpropagation

```
import math

w1 = 0.4
w2 = 0.6
w3 = 0.1
i1 = 0.1
i2 = 0.5
out = 0.2
eta = 0.5

def sigmoid (x):
    s = 1.0  / (1.0 + math.exp (-x))
    return s
```

Backpropagation

```
def forward (w1, w2, w3, i1, i2):
    neth = w1 * i1 + w2 * i2
    outh = sigmoid (neth)
    neto = w3 * outh
    outo = sigmoid (neto)
    return outo
```

Backpropagation

```
for i in range (1000):
    neth = w1 * i1 + w2 * i2
    outh = sigmoid (neth)
    neto = w3 * outh
    o = sigmoid (neto)
    err = 0.5 * (out - o) ** 2
    print (err, o)
    dw3 = (o - out) * o * (1.0 - o) * outh
    dw2 = (o - out) * o * (1.0 - o) * w3 * outh * (1.0 - outh) * i2
    dw1 = (o - out) * o * (1.0 - o) * w3 * outh * (1.0 - outh) * i1
    w3 = w3 - eta * dw3
    w2 = w2 - eta * dw2
    w1 = w1 - eta * dw1
```

XOR

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD
import numpy as np

X = np.array([[0.,0.],[0.,1.],[1.,0.],[1.,1.]])
y = np.array([[0.],[1.],[1.],[0.]])
```

XOR

```
model = Sequential()
model.add(Dense(8, input_dim=2))
model.add(Activation('tanh'))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

XOR

```
sgd = SGD(lr=0.1)

model.compile(loss='mse', optimizer=sgd)

model.fit(X, y, verbose=1, batch_size=1,
epochs=1000)

print(model.predict_classes(X))
```

MNIST

- Preparing the data :

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
from keras.utils import to_categorical
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

MNIST

- Defining the network :

```
from keras import models

from keras import layers

network = models.Sequential()

network.add(layers.Dense(512, activation='relu',
input_shape=(28 * 28,)))

network.add(layers.Dense(10, activation='softmax'))
```

MNIST

- Defining the optimizer and the loss :

```
network.compile(optimizer='rmsprop',
loss='categorical_crossentropy',
metrics=['accuracy'])
```

- Training the network :

```
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

MNIST

- Testing the network :

```
test_loss, test_acc = network.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
```

MNIST

- Predicting the class of an example :

```
import matplotlib.pyplot as plt
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
plt.imshow(test_images [0])
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
img = test_images [0].reshape ((1, 28*28))
print (network.predict_classes(img))
```

Preparing the data

```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i in range (len (sequences)):
        for j in range (len (sequences [i])):
            results [i] [sequences [i] [j]] = 1.
    return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

- You should also convert your labels from integer to numeric, which is straightforward:

```
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

IMDB

- Defining the network :

```
from keras import models
from keras import layers
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

IMDB

- Defining the optimizer and the loss :

```
model.compile(optimizer='rmsprop',
loss='binary_crossentropy',
metrics=['accuracy'])
```

IMDB

- Defining a validation set :

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

IMDB

- Training with a validation set :

```
model.compile (optimizer='rmsprop',
loss='binary_crossentropy', metrics=['acc'])

history = model.fit (partial_x_train, partial_y_train,epochs=20,
batch_size=512, validation_data=(x_val, y_val))
```

IMDB

- Visualize the training loss :

```
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

IMDB

- Visualize the training accuracy:

```
plt.clf() #Clears the figure
acc = history_dict['acc']
val_acc = history_dict['val_acc']
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```