```
## Load libraries
import numpy as np
import sys
import matplotlib.pyplot as plt
import matplotlib.cm as cm
plt.style.use('dark_background')
%matplotlib inline
```

```
import tensorflow as tf
```

```
tf.__version__
```

```
# Generate artificial data with 5 samples, 4 features per sample
# and 3 output classes
num_samples = 5 # number of samples
num_features = 4 # number of features (a.k.a. dimensionality)
num_labels = 3 # number of output labels
# Data matrix (each column = single sample)
X = np.random.choice(np.arange(3, 10), size = (num_features, num_samples), replace = True)
# Class labels
y = np.random.choice([0, 1, 2], size = num_samples, replace = True)
print(X)
print('------')
print(y)
print('------')
# One-hot encode class labels
y = tf.keras.utils.to_categorical(y)
print(y)
```

A generic layer class with forward and backward methods

```
class Layer:
  def __init__(self):
    self.input = None
    self.output = None

  def forward(self, input):
    pass

  def backward(self, output_gradient, learning_rate):
    pass
```

The softmax classifier steps for a generic sample $\mathbf{x}$ with (one-hot encoded) true label $\mathbf{y}$ (3 possible categories) using a randomly initialized weights matrix (with bias abosrbed as its last last column):

1. Calculate raw scores vector for a generic sample $\mathbf{x}$ (bias feature added):
$$\mathbf{z} = \mathbf{W}\mathbf{x}.$$
2. Calculate softmax probabilities (that is, softmax-activate the raw scores)
$$\mathbf{a} = \mathrm{softmax}(\mathbf{z}) \Rightarrow \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \mathrm{softmax}\left(\begin{bmatrix} z_0 \\ z_1 \\ z_2 \end{bmatrix}\right) = \begin{bmatrix} \frac{e^{z_0}}{e^{z_0}+e^{z_1}+e^{z_2}} \\ \frac{e^{z_1}}{e^{z_0}+e^{z_1}+e^{z_2}} \\ \frac{e^{z_2}}{e^{z_0}+e^{z_1}+e^{z_2}} \end{bmatrix}$$
3. Softmax loss for this sample is (where output label y is not yet one-hot encoded)
$$\begin{aligned} L &= -\log([a]_y) \\ &= -\log\Big([\mathrm{softmax}(\mathbf{z})]_y\Big) \\ &= -\log\Big([\mathrm{softmax}(\mathbf{W}\mathbf{x})]_y\Big). \end{aligned}$$

4. Predicted probability vector that the sample belongs to each one of the output categories is given a new name
$$\hat{\mathbf{y}} = \mathbf{a}.$$
5. One-hot encoding the output label
$$\underbrace{y \rightarrow \mathbf{y}}_{\text{e.g. } 2 \rightarrow \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}$$
results in the following representation for the softmax loss for the sample which is also referred to as the categorical crossentropy (CCE) loss:
$$L = L\left(\mathbf{y}, \hat{\mathbf{y}}\right) = \sum_{k=0}^{2} -y_k \log(\hat{y}_k)\,.$$
6. Calculate the gradient of the loss for the sample w.r.t. weights by following the computation graph from top to bottom (that is, backward):
$$\begin{array}{c} L \\ \downarrow \\ \hat{\mathbf{y}} = \mathbf{a} \\ \downarrow \\ \mathbf{z} \\ \downarrow \\ \mathbf{W} \end{array}$$
$$\begin{aligned} \Rightarrow \nabla_{\mathbf{W}}(L) &= \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\mathbf{a}) \times \nabla_{\mathbf{a}}(L) \\ &= \underbrace{\nabla_{\mathbf{W}}(\mathbf{z})}_{\text{first term}} \times \underbrace{\nabla_{\mathbf{z}}(\mathbf{a})}_{\text{second to last term}} \times \underbrace{\nabla_{\hat{\mathbf{y}}}(L).}_{\text{last term}} \end{aligned}$$
7. Now focus on the last term $\nabla_{\hat{\mathbf{y}}}(L)$:
$$\nabla_{\hat{\mathbf{y}}}(L) = \begin{bmatrix} \nabla_{\hat{y}_0}(L) \\ \nabla_{\hat{y}_1}(L) \\ \nabla_{\hat{y}_2}(L) \end{bmatrix} = \begin{bmatrix} -y_0/\hat{y}_0 \\ -y_1/\hat{y}_2 \\ -y_0/\hat{y}_2. \end{bmatrix}$$
8. Now focus on the second to last term $\nabla_{\mathbf{z}}(\mathbf{a})$:
$$\begin{aligned} \nabla_{\mathbf{z}}(\mathbf{a}) &= \nabla_{\mathbf{z}}\left(\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}\right) \\ &= \left[\,\nabla_{\mathbf{z}}(a_0) \quad \nabla_{\mathbf{z}}(a_1) \quad \nabla_{\mathbf{z}}(a_2)\,\right] \\ &= \begin{bmatrix} \nabla_{z_0}(a_0) & \nabla_{z_0}(a_1) & \nabla_{z_0}(a_2) \\ \nabla_{z_1}(a_0) & \nabla_{z_1}(a_1) & \nabla_{z_1}(a_2) \\ \nabla_{z_2}(a_0) & \nabla_{z_2}(a_1) & \nabla_{z_2}(a_2) \end{bmatrix} \\ &= \begin{bmatrix} a_0(1-a_0) & -a_1 a_0 & -a_2 a_0 \\ -a_0 a_1 & a_1(1-a_1) & -a_1 a_1 \\ -a_0 a_2 & -a_1 a_2 & a_2(1-a_2) \end{bmatrix}. \end{aligned}$$
9. On Monday, we will focus on the first term to complete the gradient calculation using the computation graph.

```
## Softmax activation class
class Softmax(Layer):
  def forward(self, input):
    self.output = np.array(tf.nn.softmax(input))

  def backward(self, output_gradient, learning_rate):
    return(np.dot((np.identity(np.size(self.output))-self.output.T) * self.output, output_gradient))


## Define the loss function and its gradient
def cce(y, yhat):
  return(-np.sum(y*np.log(yhat)))
```

```
def cce_gradient(y, yhat):
  return(-y/yhat)

# TensorFlow in-built function for categorical crossentropy loss
#cce = tf.keras.losses.CategoricalCrossentropy()


## Train the 0-layer neural network using batch training with batch size = 1

# Steps: run over each sample, calculate loss, gradient of loss,
# and update weights.

# Step-1: add the bias feature to all the samples
# Step-2: initialize the entries of the weights matrix randomly
# Step-3: create softmax layer object softmax

# Step-4: run over each sample
for i in range(X.shape[1]):
  # Step-5: forward step
  # (a) Raw scores z = Wx = np.dot(W, x[:, i])
  # (b) Softmax activation: softmax.forward(z)
  # (c) Calculate cce loss for sample: cce(y[i, :], softmax.output)
  # (d) Print cce loss

  # Step-6: backward step
  # (a) Calculate the gradient of the sample loss w.r.t. input of the
  # softmax layer: softmax.backward(output_gradient = cce_gradient(y[i, :], yhat))
  # (d) Print gradient
```