In [ ]:
```
## 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
```

In [ ]:
```
np.set_printoptions(precision = 2)
```

In [ ]:
```
import tensorflow as tf
```

```
WARNING:tensorflow:From c:\Users\vp140\.conda\envs\pycaretenv\lib\site-packages\k
eras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is depre
cated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [ ]:
```
tf.__version__
```

Out[ ]:
```
'2.15.0'
```

In [ ]:
```
# Generate artificial data with 5 samples, 4 features per sample
# and 3 output classes
num_samples = 10 # number of samples
num_features = 5 # 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), repla
# 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)
```

```
[[5 5 3 9 9 4 8 4 3 8]
 [4 9 5 9 7 4 8 8 6 6]
 [6 3 6 9 7 9 4 4 6 3]
 [3 4 3 3 9 4 7 8 6 3]
 [6 7 3 5 4 8 8 5 8 9]]
------
[2 0 2 0 1 0 1 1 1 0]
------
[[0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [1. 0. 0.]]
```

A generic layer class with forward and backward methods

In [ ]:
```
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 $\nabla_{\hat{\mathbf{y}}}(L)$:

$$\nabla_{\hat{\mathbf{y}}}(L) = \begin{bmatrix}\nabla_{\hat{y}_0}(L)\\ \nabla_{\hat{y}_1}(L)\\ \nabla_{\hat{y}_2}(L)\end{bmatrix} = \begin{bmatrix}-y_0/\hat{y}_0\\ -y_1/\hat{y}_1\\ -y_2/\hat{y}_2.\end{bmatrix}$$

8. Now focus on the second to last term $\nabla_{\mathbf{z}}(\mathbf{a})$:

$$\begin{aligned}\nabla_{\mathbf{z}}(\mathbf{a}) &= \nabla_{\mathbf{z}}\left(\begin{bmatrix}a_0\\ a_1\\ a_2\end{bmatrix}\right)\\ &= \begin{bmatrix}\nabla_{\mathbf{z}}(a_0) & \nabla_{\mathbf{z}}(a_1) & \nabla_{\mathbf{z}}(a_2)\end{bmatrix}\\ &= \begin{bmatrix}\nabla_{z_0}(a_0) & \nabla_{z_0}(a_1) & \nabla_{z_0}(a_2)\\ \nabla_{z_1}(a_0) & \nabla_{z_1}(a_1) & \nabla_{z_1}(a_2)\\ \nabla_{z_2}(a_0) & \nabla_{z_2}(a_1) & \nabla_{z_2}(a_2)\end{bmatrix}\\ &= \begin{bmatrix}a_0(1-a_0) & -a_1a_0 & -a_2a_0\\ -a_0a_1 & a_1(1-a_1) & -a_2a_1\\ -a_0a_2 & -a_1a_2 & a_2(1-a_2)\end{bmatrix}.\end{aligned}$$

9. On Monday, we will focus on the first term to complete the gradient calculation using the computation graph.

CCE loss and its gradient

$$L = L\left(\mathbf{y}, \hat{\mathbf{y}}\right) = \sum_{k=0}^{2} -y_k \log(\hat{y}_k)$$

$$\nabla_{\hat{\mathbf{y}}}(L) = \begin{bmatrix}\nabla_{\hat{y}_0}(L)\\ \nabla_{\hat{y}_1}(L)\\ \nabla_{\hat{y}_2}(L)\end{bmatrix} = \begin{bmatrix}-y_0/\hat{y}_0\\ -y_1/\hat{y}_1\\ -y_2/\hat{y}_2\end{bmatrix}.$$

In [ ]:
```
## Define the loss function and its gradient
def cce(y, yhat):
  return(-np.sum(y*np.log(yhat),axis=0))

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

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

Softmax activation layer class

$$\begin{aligned}\text{forward: } \mathbf{a} &= \text{softmax}(\mathbf{z}),\\ \text{backward: } \nabla_{\mathbf{z}}(L) &= \nabla_{\mathbf{z}}(\mathbf{a}) \times \nabla_{\mathbf{a}}(L) = \nabla_{\mathbf{z}}(\mathbf{a}) \times \nabla_{\hat{\mathbf{y}}}(L)\\ &= \begin{bmatrix} a_0(1-a_0) & -a_1a_0 & -a_2a_0 \\ -a_0a_1 & a_1(1-a_1) & -a_2a_1 \\ -a_0a_2 & -a_1a_2 & a_2(1-a_2)\end{bmatrix}\begin{bmatrix}-y_0/\hat{y}_0\\ -y_1/\hat{y}_1\\ -y_2/\hat{y}_2\end{bmatrix}.\end{aligned}$$

In [ ]:
```
## 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 = None):
    return(np.dot((np.identity(np.size(self.output))-self.output.T) * self.outpu
```

In [ ]:
```
# Step-1: add the bias feature to all the samples
X = np.vstack([X, np.ones((1, num_samples))])
```

Calculate the gradient of the losstill the second to last term (that is, the gradient w.r.t. the input of the softmax activation layer) :

$$\begin{aligned}\Rightarrow \nabla_{\mathbf{W}}(L) &= \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\mathbf{a}) \times \nabla_{\mathbf{a}}(L)\\ &= \underbrace{\nabla_{\mathbf{W}}(\mathbf{z})}_{\text{first term}} \times \underbrace{\nabla_{\mathbf{z}}(\mathbf{a})}_{\text{second to last term}} \times \underbrace{\nabla_{\hat{\mathbf{y}}}(L)}_{\text{last term}}.\end{aligned}$$

In [ ]:
```
## 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-2: initialize the entries of the weights matrix randomly
W = np.random.normal(0, 1, (num_labels, num_features))
W = np.hstack([W, 0.01*np.ones((num_labels, 1))])

# Step-3: create softmax layer object softmax
softmax = Softmax()
```

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

  # Step-6: backward step
  # (a) Calculate the gradient of the sample loss w.r.t. input of the
  # softmax layer:
  grad = cce_gradient(y[i, :], softmax.output)
  grad = softmax.backward(output_gradient = grad)
  grad = grad.reshape(-1, 1) * X[:, i].reshape(-1, 1).T
  # (d) Print gradient
  print(grad)
  # Gradient descent step
  learning_rate = 1e-03
  W = W + learning_rate * (-grad)
```

```
0.09276563687585779
[[ 4.56  3.65  5.47  2.73  5.47  0.91]
 [ 4.56  3.65  5.47  2.73  5.47  0.91]
 [-0.44 -0.35 -0.53 -0.27 -0.53 -0.09]]
18.787388616005654
[[-5.00e+00 -9.00e+00 -3.00e+00 -4.00e+00 -7.00e+00 -1.00e+00]
 [ 3.47e-08  6.24e-08  2.08e-08  2.77e-08  4.85e-08  6.93e-09]
 [ 3.47e-08  6.24e-08  2.08e-08  2.77e-08  4.85e-08  6.93e-09]]
0.024964662214391625
[[ 2.93  4.88  5.85  2.93  2.93  0.98]
 [ 2.93  4.88  5.85  2.93  2.93  0.98]
 [-0.07 -0.12 -0.15 -0.07 -0.07 -0.02]]
7.7307672495905395
[[-9.00e+00 -9.00e+00 -9.00e+00 -3.00e+00 -5.00e+00 -1.00e+00]
 [ 3.95e-03  3.95e-03  3.95e-03  1.32e-03  2.20e-03  4.39e-04]
 [ 3.95e-03  3.95e-03  3.95e-03  1.32e-03  2.20e-03  4.39e-04]]
13.91003020821659
[[ 8.19e-06  6.37e-06  6.37e-06  8.19e-06  3.64e-06  9.10e-07]
 [-9.00e+00 -7.00e+00 -7.00e+00 -9.00e+00 -4.00e+00 -1.00e+00]
 [ 8.19e-06  6.37e-06  6.37e-06  8.19e-06  3.64e-06  9.10e-07]]
16.88552860065912
[[-4.00e+00 -4.00e+00 -9.00e+00 -4.00e+00 -8.00e+00 -1.00e+00]
 [ 1.86e-07  1.86e-07  4.18e-07  1.86e-07  3.71e-07  4.64e-08]
 [ 1.86e-07  1.86e-07  4.18e-07  1.86e-07  3.71e-07  4.64e-08]]
13.30788695865515
[[ 1.33e-05  1.33e-05  6.65e-06  1.16e-05  1.33e-05  1.66e-06]
 [-8.00e+00 -8.00e+00 -4.00e+00 -7.00e+00 -8.00e+00 -1.00e+00]
 [ 1.33e-05  1.33e-05  6.65e-06  1.16e-05  1.33e-05  1.66e-06]]
13.216881055973108
[[ 7.28e-06  1.46e-05  7.28e-06  1.46e-05  9.10e-06  1.82e-06]
 [-4.00e+00 -8.00e+00 -4.00e+00 -8.00e+00 -5.00e+00 -1.00e+00]
 [ 7.28e-06  1.46e-05  7.28e-06  1.46e-05  9.10e-06  1.82e-06]]
5.429831732373401
[[ 1.32e-02  2.63e-02  2.63e-02  2.63e-02  3.51e-02  4.38e-03]
 [-2.99e+00 -5.97e+00 -5.97e+00 -5.97e+00 -7.96e+00 -9.96e-01]
 [ 1.32e-02  2.63e-02  2.63e-02  2.63e-02  3.51e-02  4.38e-03]]
10.583802580031405
[[-8.00e+00 -6.00e+00 -3.00e+00 -3.00e+00 -9.00e+00 -1.00e+00]
 [ 2.03e-04  1.52e-04  7.60e-05  7.60e-05  2.28e-04  2.53e-05]
 [ 2.03e-04  1.52e-04  7.60e-05  7.60e-05  2.28e-04  2.53e-05]]
```