LAPENNA Program
Q&A 2

Kwai Wong, Stan Tomov
Julian Halloy, Stephen Qiu, Eric Zhao

University of Tennessee, Knoxville

Sept. 29, 2021

Acknowledgements:

- Support from NSF, UTK, JICS, ICL, NICS
- LAPENNA, www.jics.utk.edu/lapenna, NSF award #202409
- www.icl.utk.edu, cfdlab.utk.edu, www.xsede.org, www.jics.utk.edu/recsem-reu,
- MagmaDNN is a project grown out from the RECSEM REU Summer program supported under NSF award #1659502
- Source code: www.bitbucket.org/icl/magmadnn
- www.bitbucket.org/cfdl/opendnnwheel

INNOVATIVE COMPUTING LABORATORY

Exercises
Multilayer Perceptron (MLP)

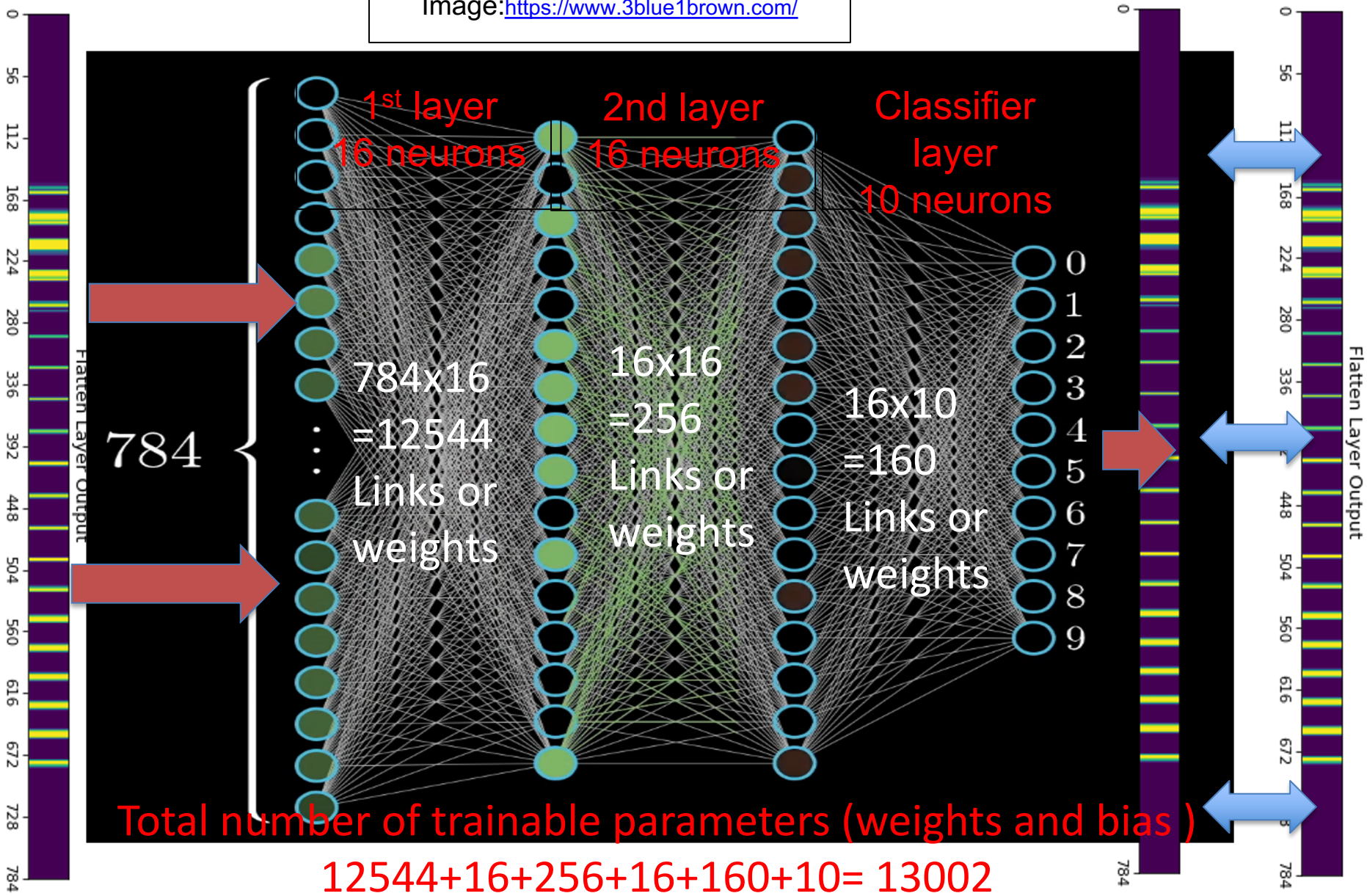
Simple MNIST MLP Neural Network

Input

output

labels

Image:https://www.3blue1brown.com/

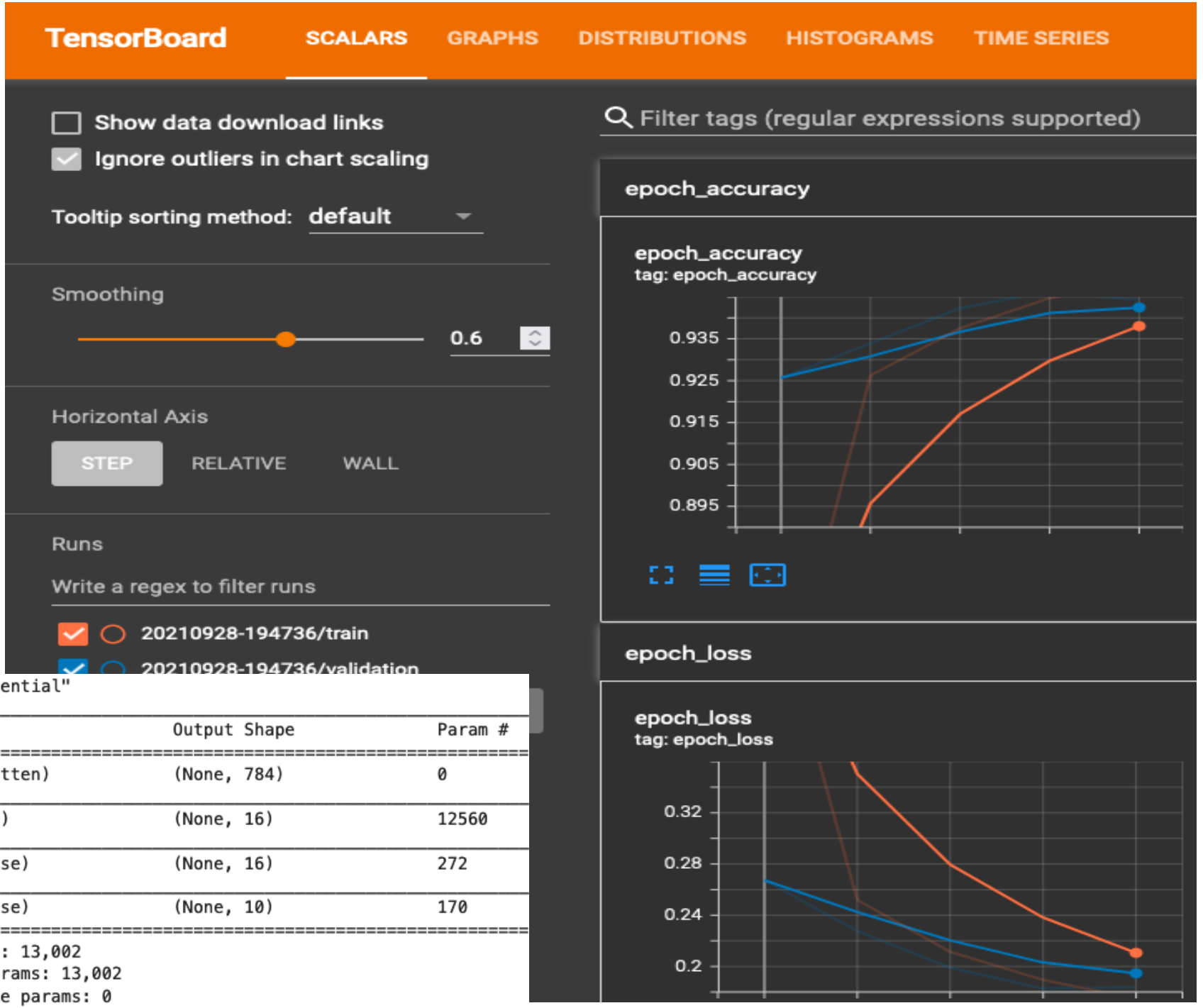


Total number of trainable parameters (weights and bias)

12544+16+256+16+160+10= 13002

MNIST Tensorflow code

```
import tensorflow as tf
print(tf.__version__)
import datetime, os

%load_ext tensorboard

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(10,activation='softmax')
])

model.compile(optimizer='sgd', loss='SparseCategoricalCrossentropy',
metrics=['accuracy'])
model.summary()

logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)

model.fit(x_train, y_train, epochs=5, batch_size=10,
    validation_data=(x_test, y_test), callbacks=[tensorboard_callback])

model.evaluate(x_test, y_test, verbose=2)

%tensorboard --logdir logs
```




```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 16)                12560
_________________________________________________________________
dense_1 (Dense)              (None, 16)                272
_________________________________________________________________
dense_2 (Dense)              (None, 10)                170
=================================================================
Total params: 13,002
Trainable params: 13,002
Non-trainable params: 0
```

Exercise

a) A neural network is used to classify a dataset composed of 5000 grayscale images of 0 to 9 digits. Each of the image has 32 x 32 pixels. The neural network contains 4 layers. The first layer is the input layer, the second layer has 20 neurons, the third one has 15 neurons, the last one is the softmax neuron of 10 classes. Please specify clearly the dimension of the matrices of each layer (both input weights and output) of the training process given a batch size of 50?

Layer one : input vector
Layer two:
- input matrix and
- weight matrix,
- output matrix

Layer three:
- input matrix
- weight matrix,
- output matrix

Layer four:
- input matrix,
- weight matrix, and
- output

b) What is the total number of trainable parameters?

c) How many batch iterations are needed to finish one epoch ?

Exercise : Answer

a) A neural network is used to classify a dataset composed of 5000 grayscale images of 0 to 9 digits. Each of the image has 32 x 32 pixels. The neural network contains 4 layers. The first layer is the input layer, the second layer has 20 neurons, the third one has 15 neurons, the last one is the softmax neuron of 10 classes. Please specify clearly the dimension of the matrices of each layer (both input weights and output) of the training process given a batch size of 50?

Layer one : input vector : 32 x32 = 1024

Layer two:

- input matrix = 50 x 1024
- weight matrix = 1024 x 20
- output matrix = 50 x20

Layer three:

- input matrix = 50 x 20
- weight matrix = 20 x 15
- output matrix = 50 x 15

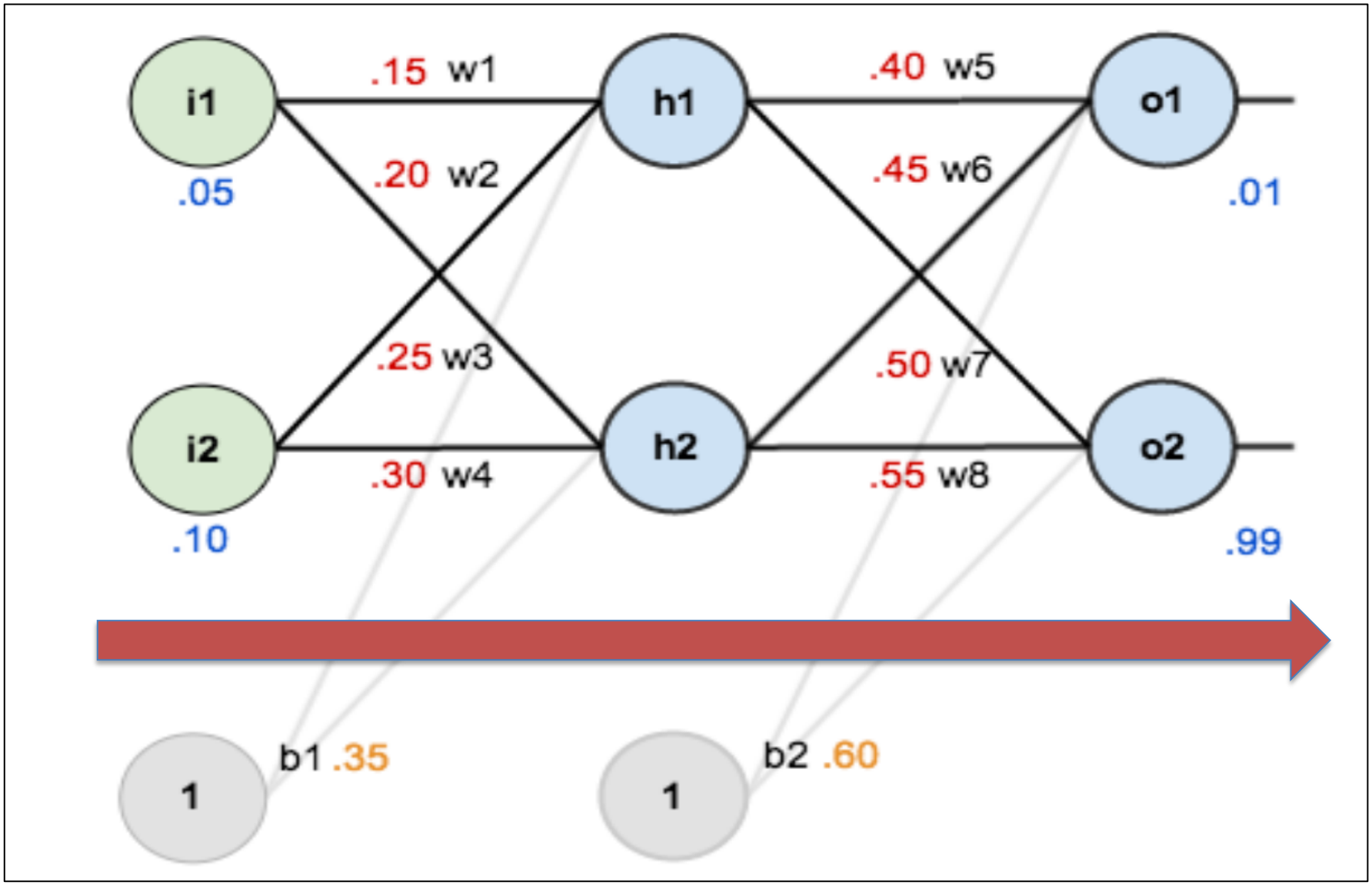
Layer four:

- input matrix = 50 x15
- weight matrix = 15 x 10
- output = 50 x 10

b) Total number of trainable parameters = (1024x20 + 20) + (20x15+15) + (15x10+10) =20975

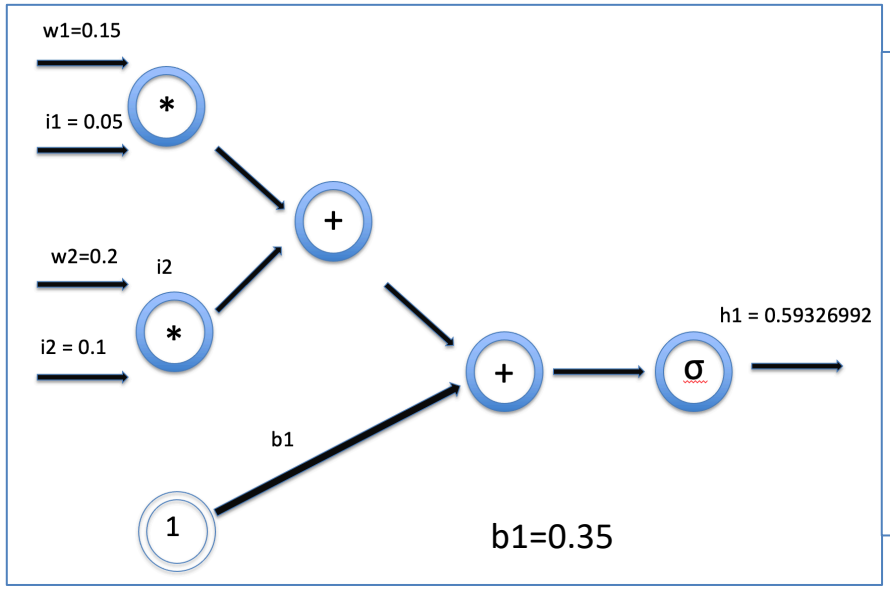
c) How many batch iterations are needed to finish one epoch ? = 5000/50 = 100

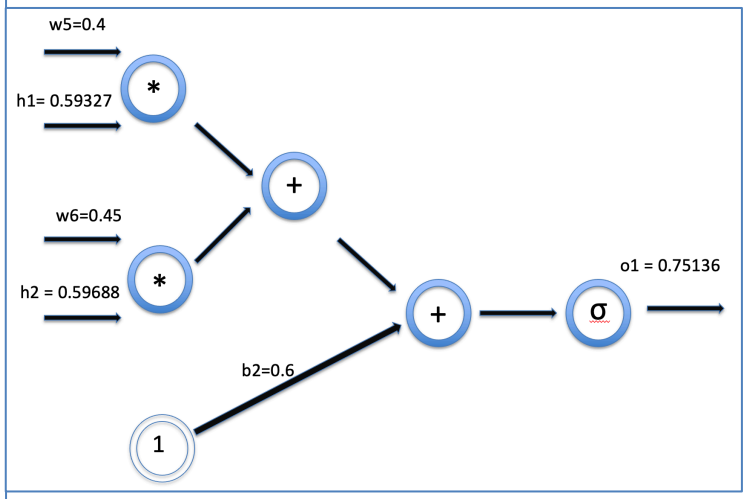
Multilayer Perceptron : Forward Path Calculation



$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$





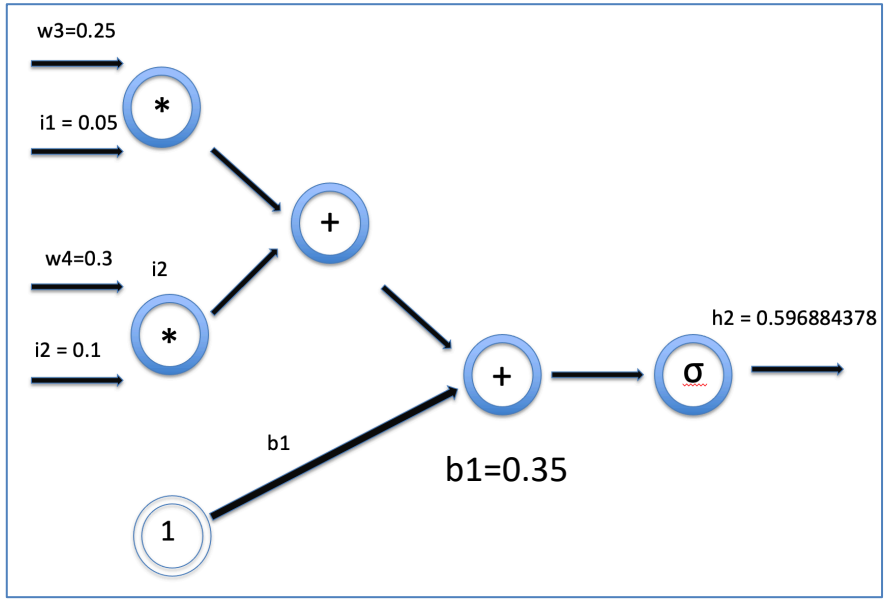
target=0.01

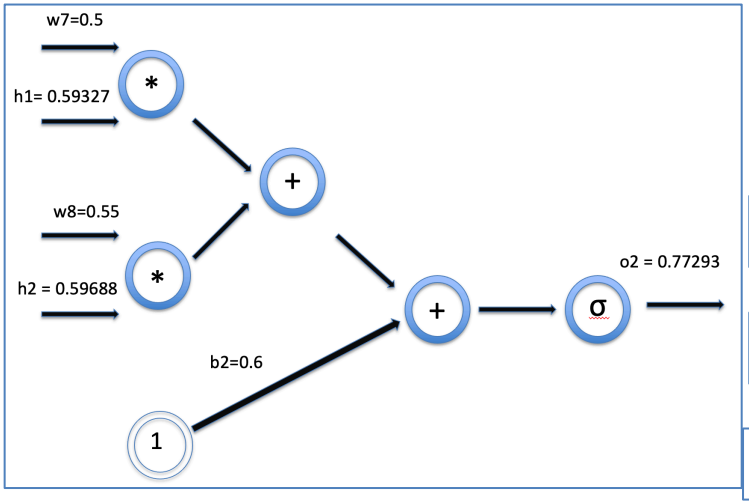
o1=0.75136

Eo1=0.27481

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Etotal = Eo1 + Eo2 = 0.298371109





Eo2=0.02356

o2=0.77293

target=0.99

$$E_{o2} = 0.023560026$$

Python Code
Forward path calculation

```
# e constant
e = 2.7182818284
# initial values
i1 = 0.05
i2 = 0.10
# initial weights
w1 = 0.15
w2 = 0.20
w3 = 0.25
w4 = 0.30
w5 = 0.40
w6 = 0.45
w7 = 0.50
w8 = 0.55
# bias
b1 = 0.35
b2 = 0.60
# targets
To1 = 0.01
To2 = 0.99
```

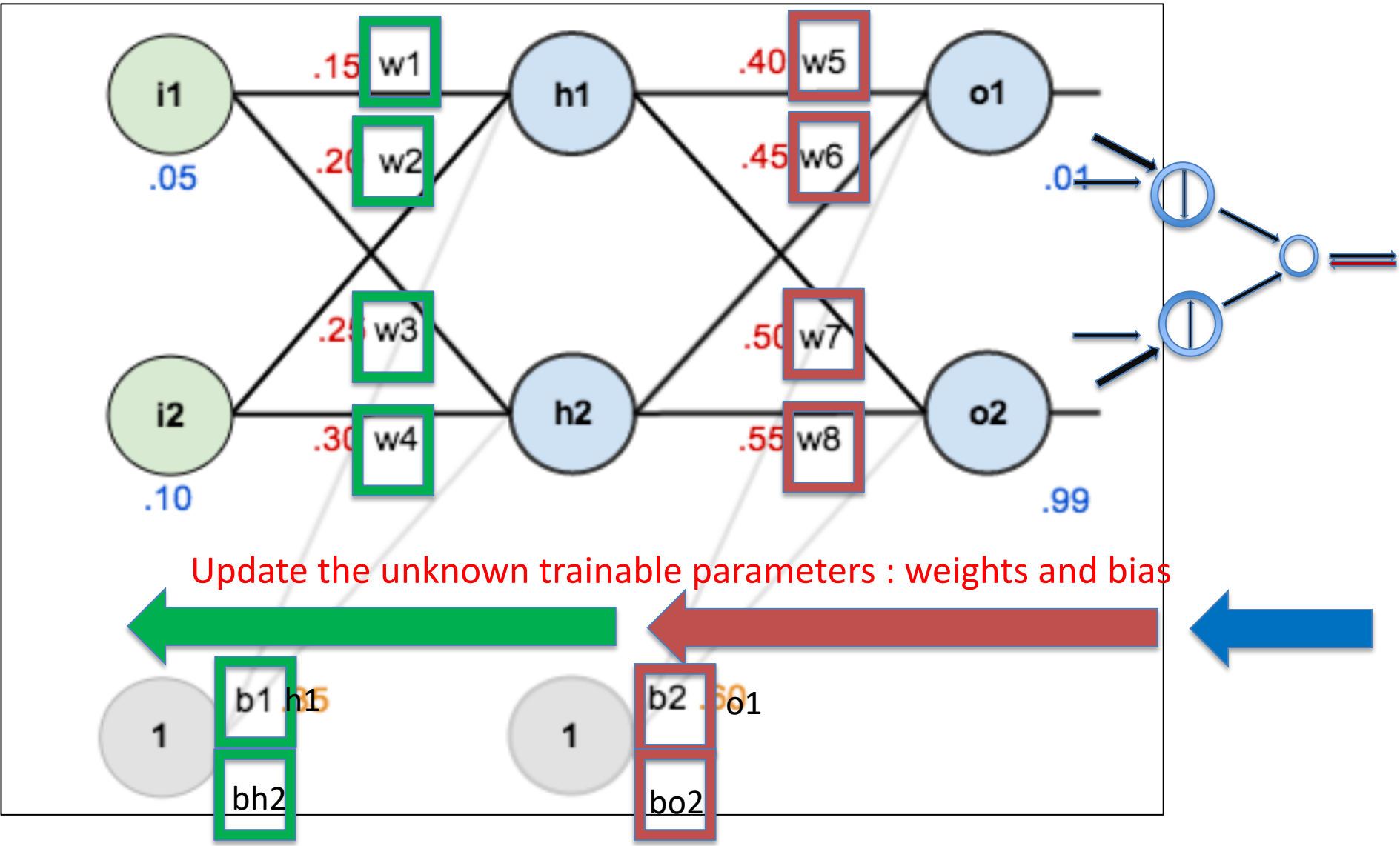
```
# forward propagation
h1 = 1/(1+e**(-(w1*i1 + w2*i2+b1)))
print("h1: " + str(h1))

h2 = 1/(1+e**(-(w3*i1 + w4*i2+b1)))
print("h2: " + str(h2))

o1 = 1/(1+e**(-(w5*h1 + w6*h2+b2)))
print("o1: " + str(o1))

o2 = 1/(1+e**(-(w7*h1 + w8*h2+b2)))
print("o2: " + str(o2))
```

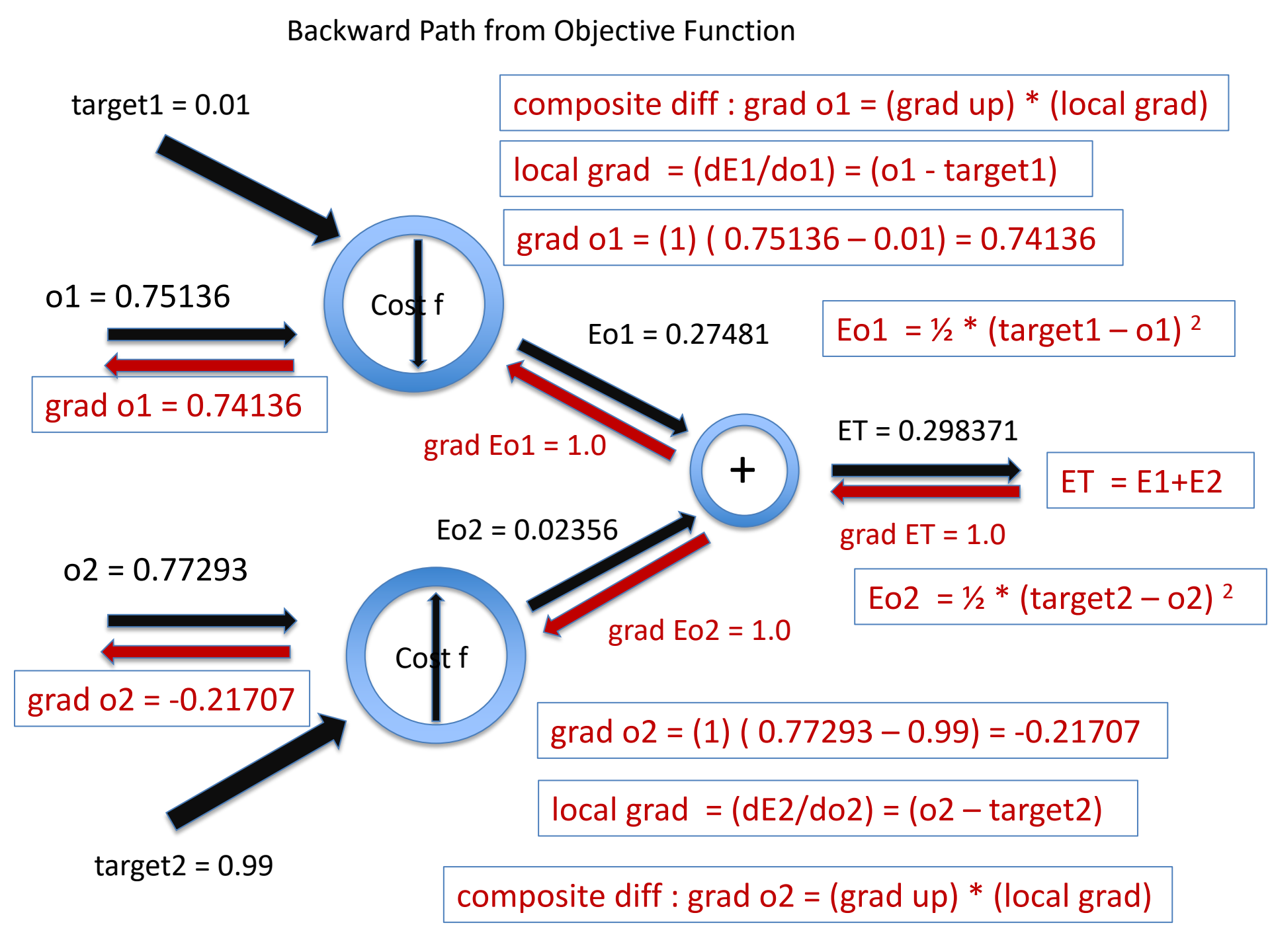
```
h1: 0.5932699921052087 h2: 0.5968843782577157
o1: 0.7513650695475076 o2: 0.772928465316421
```

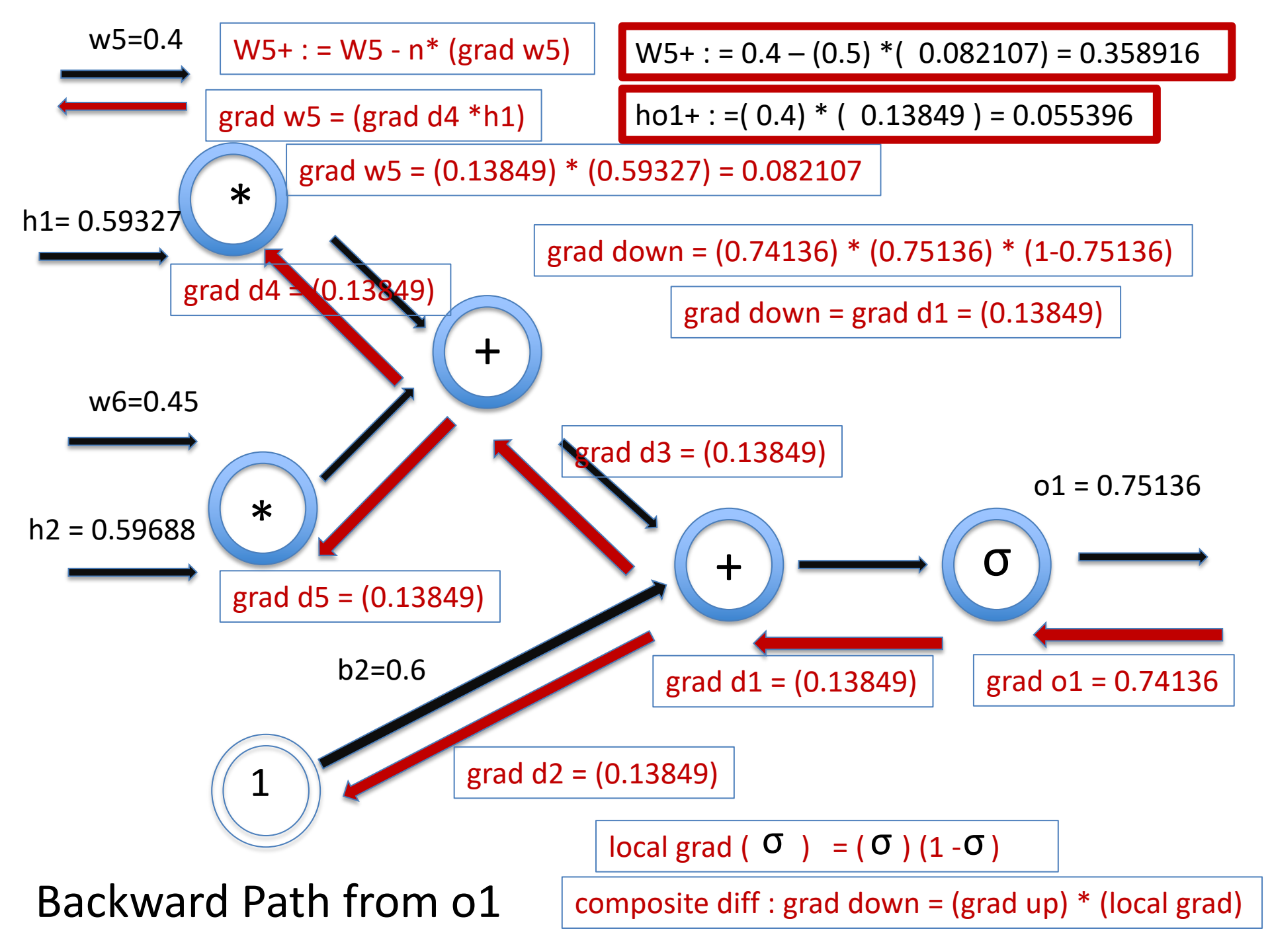
```
# Error
Eo1 = 0.5*(To1-o1)**2
print("Error o1: " + str(Eo1))
Eo2 = 0.5*(To2-o2)**2
print("Error o2: " + str(Eo2))

E = Eo1 + Eo2
print("Total Error: " + str(E))
```

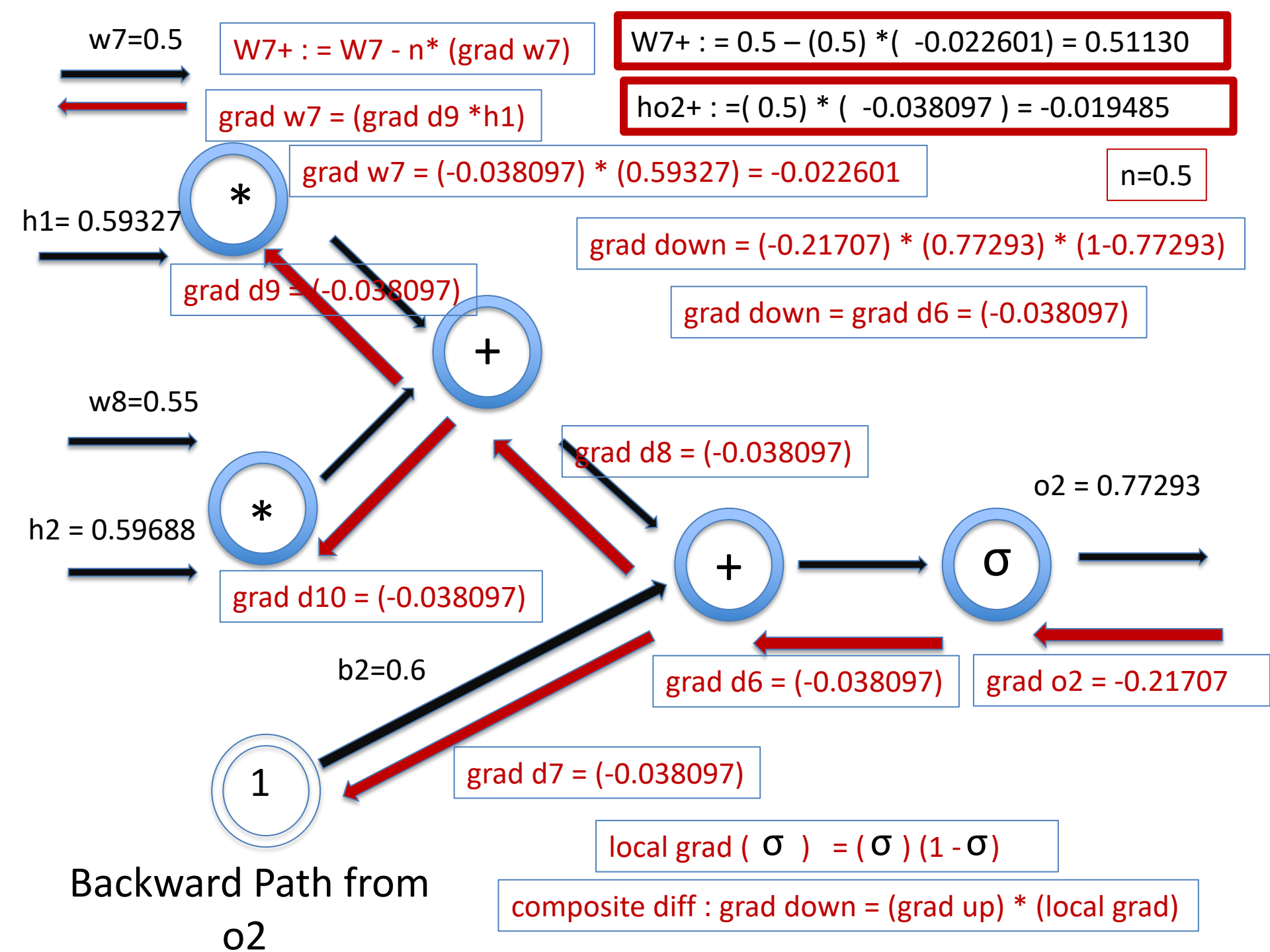
```
Error o1: 0.2748110831725904
Error o2: 0.023560025584942117
Total Error: 0.29837110875753253
```

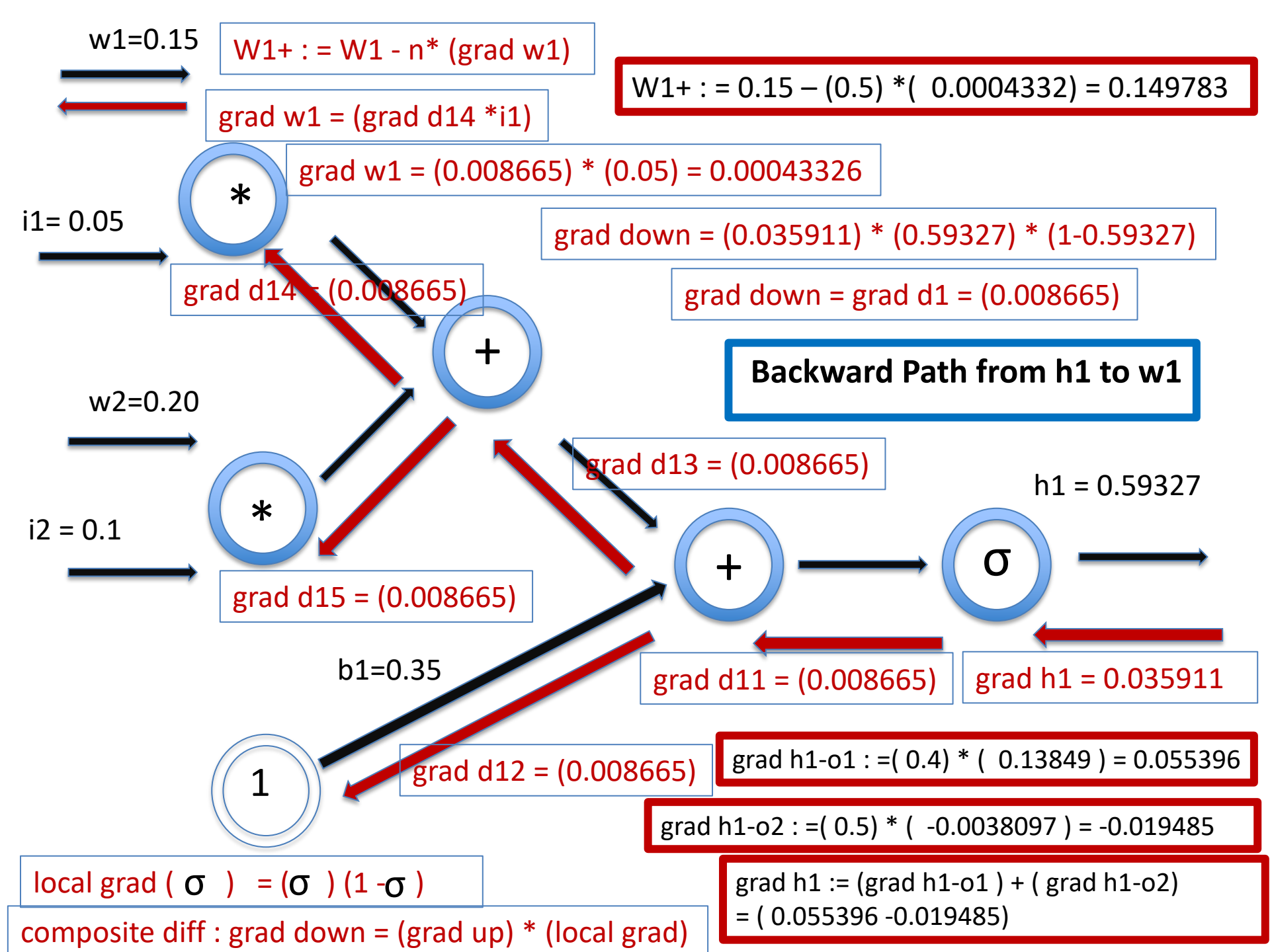
Multilayer Perceptron : Backward Path Calculation




Backward Path from Objective Function
target1 = 0.01
composite diff : grad o1 = (grad up) * (local grad)
local grad = (dE1/do1) = (o1 - target1)
grad o1 = (1) (0.75136 – 0.01) = 0.74136
o1 = 0.75136
Cost f
Eo1 = 0.27481
Eo1 = ½ * (target1 – o1) 2
grad o1 = 0.74136
grad Eo1 = 1.0
+
ET = 0.298371
ET = E1+E2
grad ET = 1.0
Eo2 = 0.02356
Eo2 = ½ * (target2 – o2) 2
o2 = 0.77293
grad Eo2 = 1.0
Cost f
grad o2 = -0.21707
grad o2 = (1) (0.77293 – 0.99) = -0.21707
local grad = (dE2/do2) = (o2 – target2)
target2 = 0.99
composite diff : grad o2 = (grad up) * (local grad)


w5=0.4
W5+ : = W5 - n* (grad w5)
W5+ : = 0.4 – (0.5) *(0.082107) = 0.358916
grad w5 = (grad d4 *h1)
ho1+ : =(0.4) * (0.13849) = 0.055396
*
grad w5 = (0.13849) * (0.59327) = 0.082107
h1= 0.59327
grad down = (0.74136) * (0.75136) * (1-0.75136)
grad d4 = (0.13849)
grad down = grad d1 = (0.13849)
+
w6=0.45
grad d3 = (0.13849)
o1 = 0.75136
*
h2 = 0.59688
+
σ
grad d5 = (0.13849)
b2=0.6
grad d1 = (0.13849)
grad o1 = 0.74136
1
grad d2 = (0.13849)
local grad (σ) = (σ) (1 -σ)
Backward Path from o1
composite diff : grad down = (grad up) * (local grad)


w7=0.5
W7+ : = W7 - n* (grad w7)
W7+ : = 0.5 – (0.5) *(-0.022601) = 0.51130
grad w7 = (grad d9 *h1)
ho2+ : =(0.5) * (-0.038097) = -0.019485
*
grad w7 = (-0.038097) * (0.59327) = -0.022601
n=0.5
h1= 0.59327
grad down = (-0.21707) * (0.77293) * (1-0.77293)
grad d9 = (-0.038097)
grad down = grad d6 = (-0.038097)
+
w8=0.55
grad d8 = (-0.038097)
o2 = 0.77293
*
h2 = 0.59688
+
σ
grad d10 = (-0.038097)
b2=0.6
grad d6 = (-0.038097)
grad o2 = -0.21707
1
grad d7 = (-0.038097)
local grad (σ) = (σ) (1 - σ)
composite diff : grad down = (grad up) * (local grad)
Backward Path from o2


w1=0.15
W1+ : = W1 - n* (grad w1)
W1+ : = 0.15 – (0.5) *(0.0004332) = 0.149783
grad w1 = (grad d14 *i1)
grad w1 = (0.008665) * (0.05) = 0.00043326
*
i1= 0.05
grad down = (0.035911) * (0.59327) * (1-0.59327)
grad d14 = (0.008665)
grad down = grad d1 = (0.008665)
+
Backward Path from h1 to w1
w2=0.20
grad d13 = (0.008665)
h1 = 0.59327
*
i2 = 0.1
+
σ
grad d15 = (0.008665)
b1=0.35
grad d11 = (0.008665)
grad h1 = 0.035911
1
grad d12 = (0.008665)
grad h1-o1 : =(0.4) * (0.13849) = 0.055396
grad h1-o2 : =(0.5) * (-0.0038097) = -0.019485
local grad (σ) = (σ) (1 -σ)
grad h1 := (grad h1-o1) + (grad h1-o2)
= (0.055396 -0.019485)
composite diff : grad down = (grad up) * (local grad)

```
# Backpropagation
n = 0.5

grad_o1 = o1 - To1
grad_o2 = o2 - To2

print("grad_o1: "+str(grad_o1))
print("grad_o2: "+str(grad_o2))

grad_d1 = (grad_o1)*o1*(1-o1)
grad_d2 = (grad_o2)*o2*(1-o2)

print("grad_d1: "+str(grad_d1))
print("grad_d2: "+str(grad_d2))

grad_w5 = grad_d1*h1
print("grad_w5: "+str(grad_w5))

w5_final = w5 - n*grad_w5

grad_w6 = grad_d1*h2
print("grad_w6: "+str(grad_w6))
w6_final = w6 - n*grad_w6

grad_w7 = grad_d2*h1
print("grad_w7: "+str(grad_w7))
w7_final = w7 - n*grad_w7

grad_w8 = grad_d2*h2
print("grad_w8: "+str(grad_w8))
w8_final = w8 - n*grad_w8
```

```
grad_h1 = w5*grad_d1 + w7*grad_d2
print("grad_h1: "+str(grad_h1))
grad_d11 = grad_h1*h1*(1-h1)
print("grad_d11: "+str(grad_d11))

grad_w1 = grad_d11*i1
print("grad_w1: "+str(grad_w1))
w1_final = w1 - n*grad_w1

grad_w2 = grad_d11*i1
print("grad_w2: "+str(grad_w2))
w2_final = w2 - n*grad_w2

grad_h2 = w6*grad_d1 + w8*grad_d2
print("grad_h2: "+str(grad_h2))
grad_d22 = grad_h2*h2*(1-h2)
print("grad_d22: "+str(grad_d22))

grad_w3 = grad_d22*i1
print("grad_w3: "+str(grad_w3))
w3_final = w3 - n*grad_w3

grad_w4 = grad_d22*i2
print("grad_w4: "+str(grad_w4))
w4_final = w4 - n*grad_w4
```

```
print("w1+: "+str(w1_final))
print("w2+: "+str(w2_final))
print("w3+: "+str(w3_final))
print("w4+: "+str(w4_final))
print("w5+: "+str(w5_final))
print("w6+: "+str(w6_final))
print("w7+: "+str(w7_final))
print("w8+: "+str(w8_final))
```

w1+: 0.1497807161327648
w2+: 0.1997807161327648 2
w3+: 0.2497511436323716 4
w4+: 0.2995022872647432 6
w5+: 0.3589164797177565 3
w6+: 0.4086661860761087
w7+: 0.5113012702391395
w8+: 0.5613701211083925

grad_o1: 0.7413650695475076
grad_o2: -0.2170715346835789 8
grad_d1: 0.1384985616294507 6
grad_d2: -0.0380982365180384 4
grad_w5: 0.0821670405644870 1
grad_w6: 0.0826676278477826 3
grad_w7: -0.0226025404782790 4
grad_w8: -0.0227402422167847 7
grad_h1: 0.0363530639276108 6
grad_d11: 0.0087713546894077 9
grad_w1: 0.0004385677344703895
grad_w2: 0.0004385677344703895
grad_h2: 0.0413703226483317 1
grad_d22: 0.0099542547051342 71
grad_w3: 0.0004977127352567136
grad_w4: 0.0009954254705134271

```
# Tensorflow code for the example
import tensorflow as tf
import keras

# sets the input [i1, i2]
x = tf.constant([0.05, 0.1],shape=[1,2],
dtype=tf.float32)

#setting the weight matrix for first fully
connected layer [w1, w3];[w2, w4]
w1 = tf.constant([[0.15,0.25],[0.2,0.3]])

#setting the weight matrix for second fc layer
[w5, w7];[w6, w8]
w2 = tf.constant([[0.4,0.5],[0.45,0.55]])

# sets the target matrix [t1, t2]
t = tf.constant([0.01,0.99])

#sets the bias1.
b1 = tf.constant([0.35, 0.35])

#sets the bias2.
b2 = tf.constant([0.6, 0.6])

def forward(x,w1,w2,b1,b2):
     sum1=tf.matmul(x,w1)+b1
     H = tf.nn.sigmoid(sum1)
     sum2=tf.matmul(H,w2)+b2
     O = tf.nn.sigmoid(sum2)
     return O
```

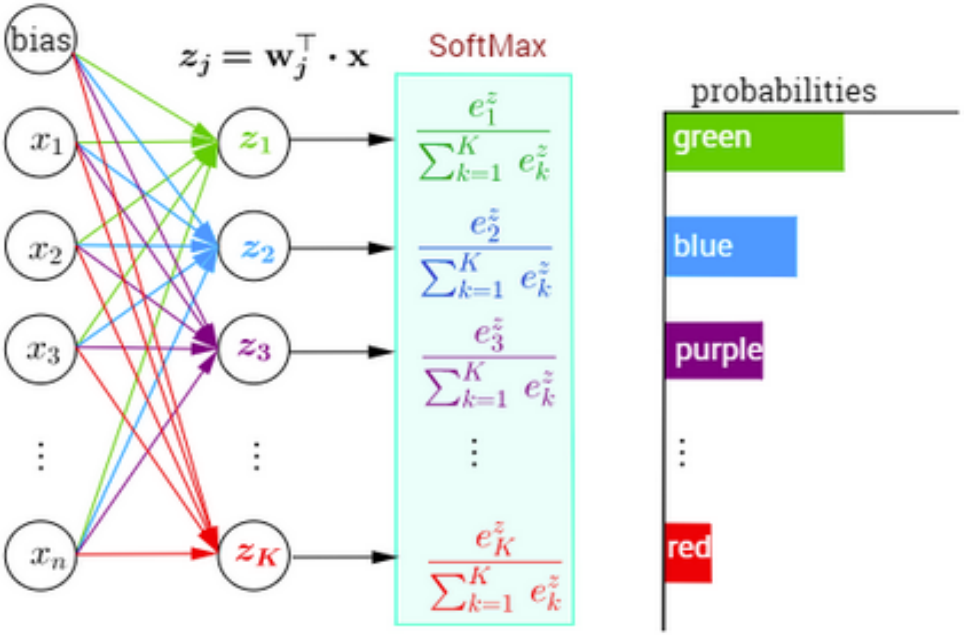
```
def backward(x,w1,w2,b1,b2,t):
     with tf.GradientTape() as g:
        g.watch([w1,w2,x,b1,b2])
        O = forward(x, w1, w2, b1, b2)
        Etotal = tf.losses.mse(t, O)
        print("Total loss=" + str(Etotal))

#calculates the gradients for each part
     dEtotal_dw = g.gradient(Etotal,
     [w1,w2,x,b1,b2])
     dEtotal_dw1, dEtotal_dw2,
     dEtotal_dx,    dEtotal_db1,   dEtotal_db2
= dEtotal_dw
     return dEtotal_dw1, dEtotal_dw2,
     dEtotal_dx,dEtotal_db1,dEtotal_db2

dEtotal_dw1,dEtotal_dw2,dEtotal_dx,dEtotal_db1,
dEtotal_db2=backward(x,w1,w2,b1,b2,t)

#the learning rate is set to be 0.5
w1 = w1 - 0.5*dEtotal_dw1
w2 = w2 - 0.5*dEtotal_dw2

b1 = b1 - 0.5*dEtotal_db1
b2 = b2 - 0.5*dEtotal_db2

print('The new w1 is',w1.numpy())
print('The new w2 is',w2.numpy())
print('The new b1 is',b1.numpy())
print('The new b2 is',b2.numpy())
```

```
Total loss = tf.Tensor([0.2983711], shape=(1,), dtype=float32)
The new w1 is [[0.14978072   0.24975115] [0.19956143   0.2995023 ]]
The new w2 is [[0.3589165    0.5113013 ] [0.40866616    0.56137013]]
The new b1 is [0.3456143   0.34502286] The new b2 is [0.53075075   0.61904913]
```

Multi-Class Classification with NN and SoftMax Function

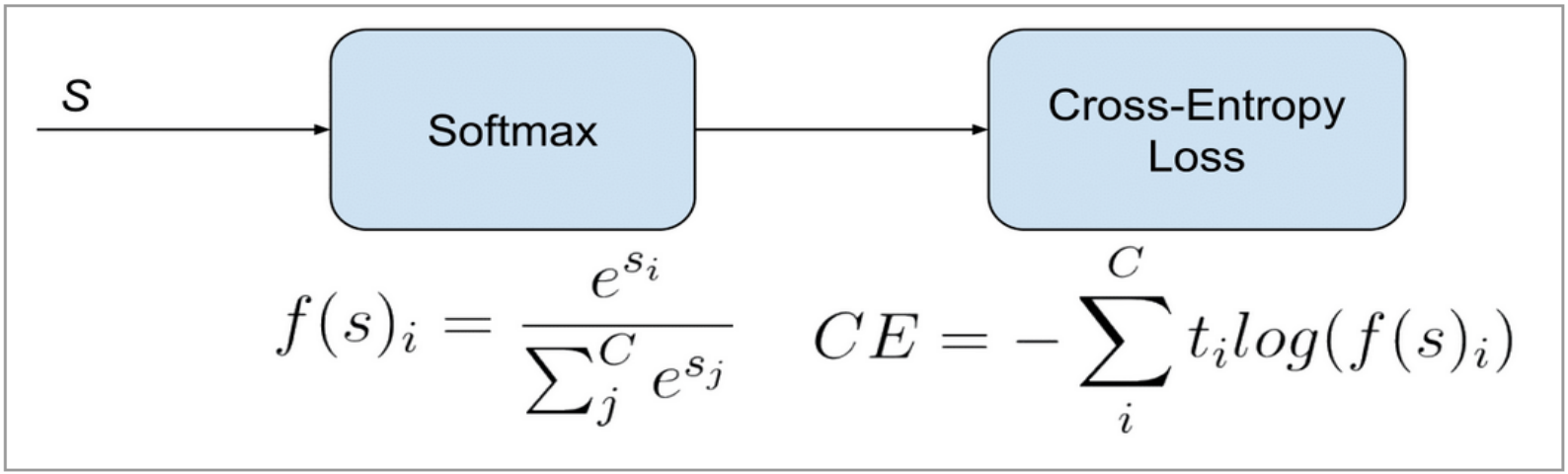
$$S(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

$$\mathbf{z} = \begin{bmatrix} z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

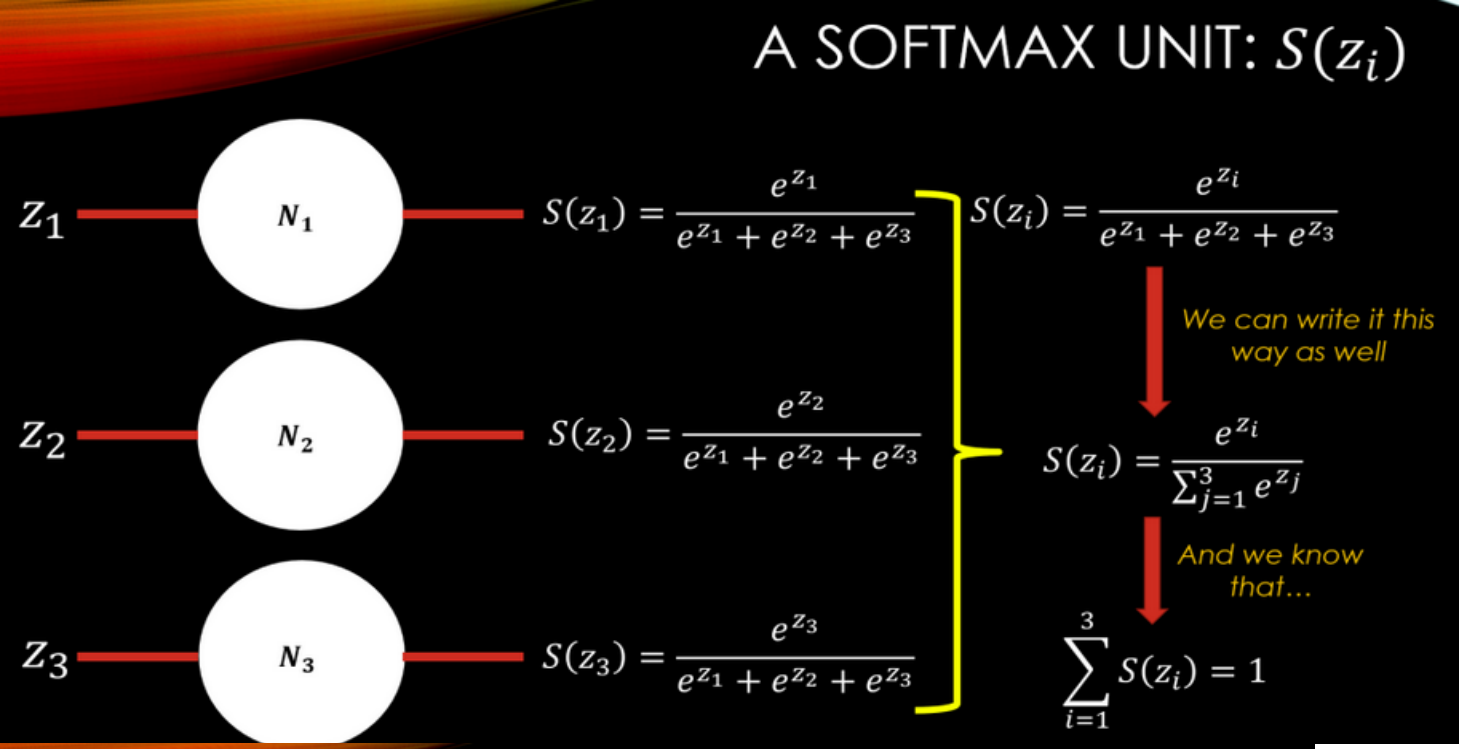


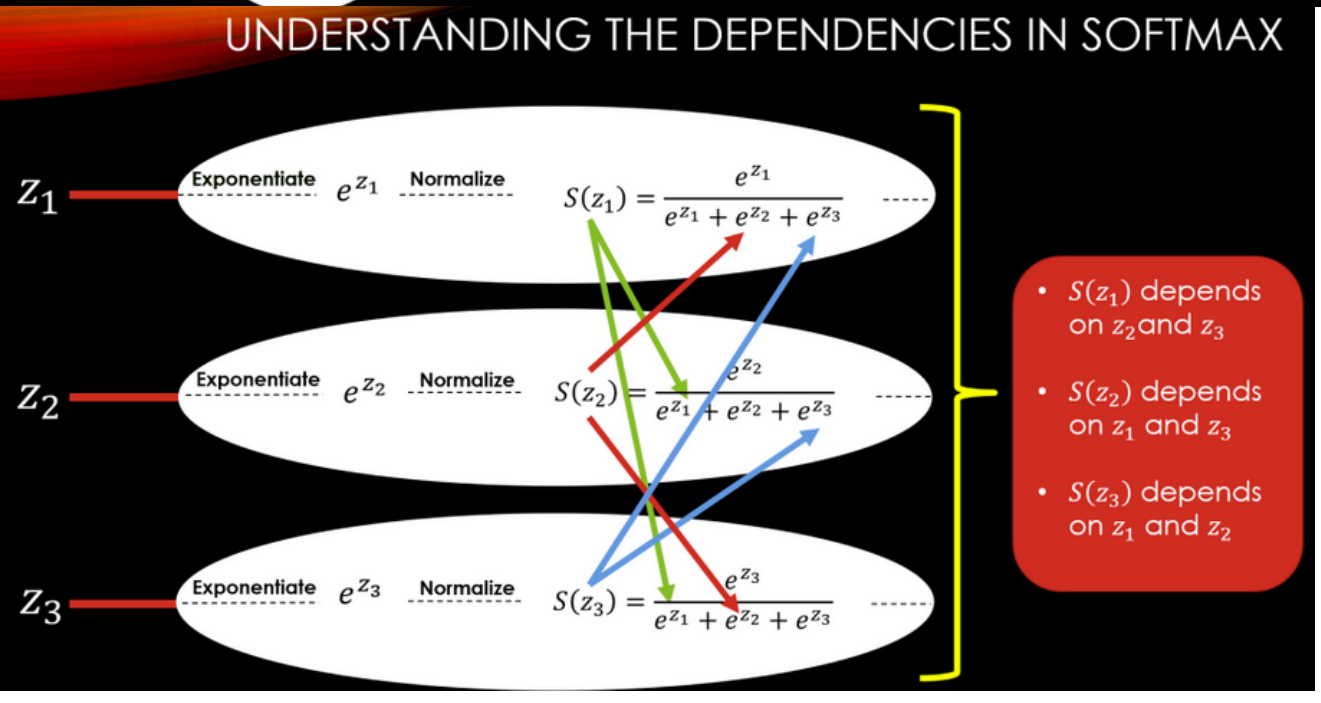
Softmax Function

Categorical (Softmax) Cross Entropy Loss (Statistical Learning)

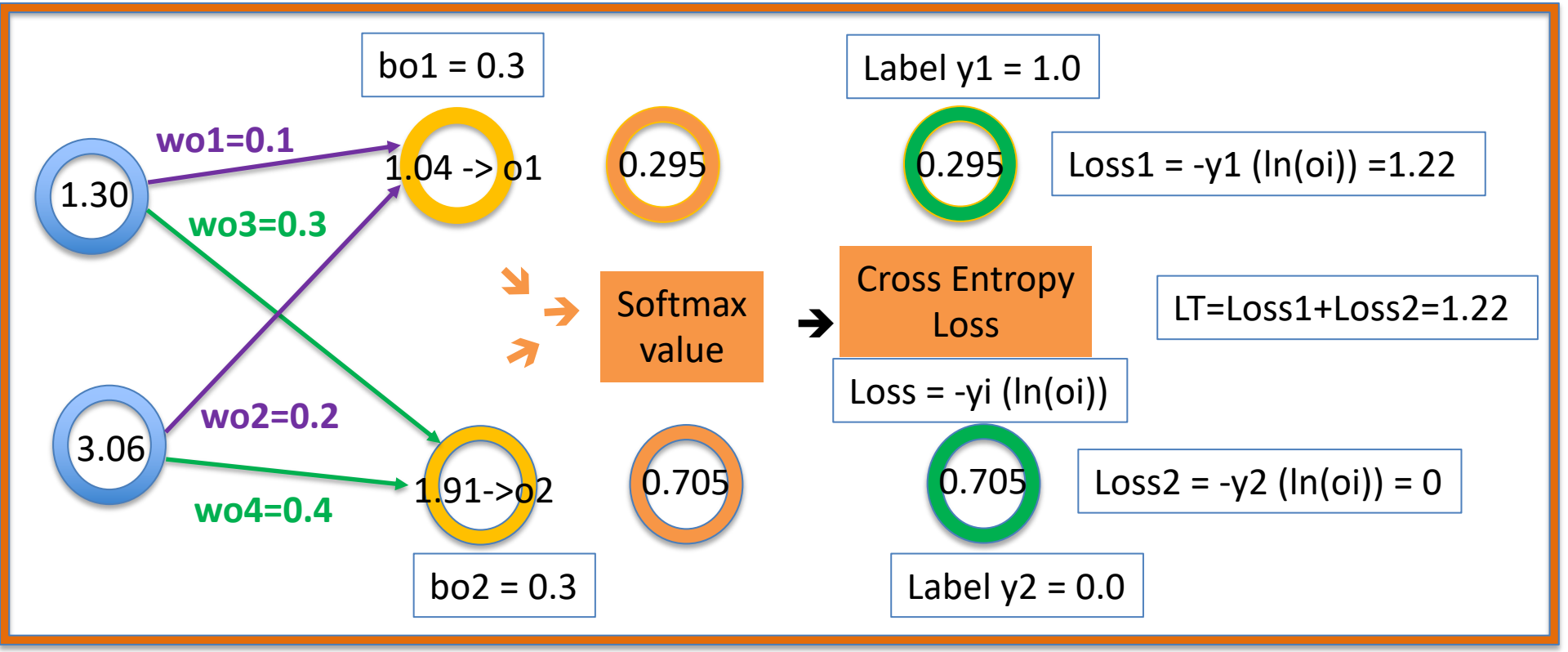


dE/dz=d(E=cross entropy)/d(z=w^Tx+b) = d(yi(lnoi)/dz = (oi – yi)





https://www.mldawn.com/the-derivative-of-softmaxz-function-w-r-t-z/



CNN : Backward Path (derivatives)

Learning rate = n = 0.5

$$\frac{\partial S(z_i)}{\partial z_j} = \begin{cases} S(z_i) \times (1 - S(z_i)) & \text{if } i = j \\ -S(z_i) \times S(z_j) & \text{if } i \neq j \end{cases}$$

$$\frac{\partial E}{\partial Z_k} = O_k - y_k$$

weights bias

w+ = w – n * (dE/dw)
dE/dw = (dE/dS) * (dS/dw)

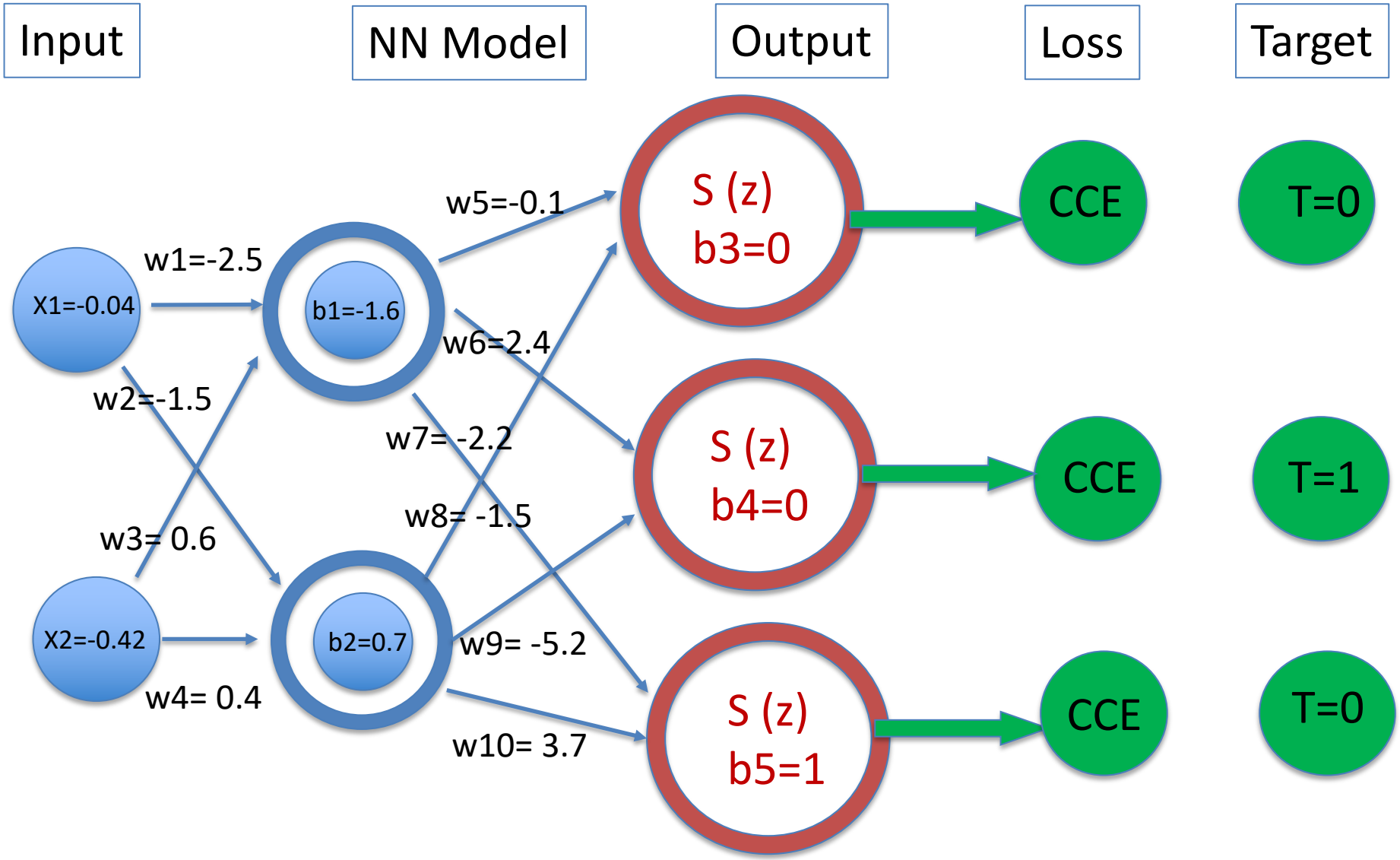
Softmax value

dL/dS

Cross Entropy Loss (L)

dL/dL = 1

Exercises : Represent the following NN model as a computational graph and build a python code to update w and b



Categorical (Softmax) Cross Entropy Loss

```
# the code for updating the parameters in question 2
import tensorflow as tf
import keras

x = tf.constant([-0.04, -0.42],shape=[1, 2],dtype=tf.float32) # sets the input
[X1, X2]

w1 = tf.constant([[-2.5, -1.5],[0.6, 0.4]]) #setting the weight matrix for first
fully connected layer [w1, w2];[w3, w4]
w2 = tf.constant([[-0.1, 2.4, -2.2],[-1.5, -5.2, 3.7]]) #setting the weight matrix
for second fc layer [w5, w6, w7];[w8, w9, w10]

t = tf.constant([[0, 1, 0]], dtype=tf.float32) # sets the target matrix [t1 t2,t3]

b1 = tf.constant([-1.6, 0.7]) #sets the bias1.
b2 = tf.constant([0, 0, 1], dtype=tf.float32) #sets the bias2.

def forward(x,w1,w2,b1,b2):
    sum1=tf.matmul(x,w1)+b1
    H = tf.nn.sigmoid(sum1)
    print(H)
    sum2=tf.matmul(H,w2)+b2
    O = tf.nn.softmax(sum2)
    return O

O = forward(x, w1, w2, b1, b2)

print('Forward path output is:', O.numpy())
```

```
def backward(x,w1,w2,b1,b2,t):
with tf.GradientTape() as g:
    g.watch([w1,w2,x,b1,b2])
        O = forward(x, w1, w2, b1, b2)
        Etotal = tf.losses.categorical_crossentropy(t, O)
        print("Total loss = " , Etotal.numpy())
    dEtotal_dw = g.gradient(Etotal, [w1,w2,x,b1,b2]) #calculates the gradients
    dEtotal_dw1, dEtotal_dw2, dEtotal_dx, dEtotal_db1, dEtotal_db2 = dEtotal_dw
    return dEtotal_dw1,dEtotal_dw2,dEtotal_dx,dEtotal_db1,dEtotal_db2

dEtotal_dw1,dEtotal_dw2,dEtotal_dx,dEtotal_db1,dEtotal_db2 =
backward(x,w1,w2,b1,b2,t)

w1 = w1 - 0.5*dEtotal_dw1 #the learning rate is set to be 0.5
w2 = w2 - 0.5*dEtotal_dw2

b1 = b1 - 0.5*dEtotal_db1
b2 = b2 - 0.5*dEtotal_db2

print('The new w1 is:')
print(w1.numpy())
print('The new w2 is:')
print(w2.numpy())
print('The new b1 is:')
print(b1.numpy())
print('The new b2 is:')
print(b2.numpy())
```

```
tf.Tensor([[0.14779511 0.64382386]], shape=(1, 2), dtype=float32)
Forward path output is: [[0.01729538   0.00231123.   0.9803934 ]]
tf.Tensor([[0.14779511   0.64382386]], shape=(1, 2), dtype=float32)
Total loss = [6.069976]
The new w1 is: [[-2.5114694   -1.4596888 ]
                [ 0.47957253   0.8232676 ]]
The new w2 is: [[-0.10127809 2.4737267 -2.2724488 ]
                [-1.5055676 -4.878832 3.3843997 ]]
The new b1 is: [-1.313268 -0.30777997]
The new b2 is: [-0.00864769 0.4988444 0.5098033 ]
```

Exercise : write a simple python code for the example

Homework

Question

CIFAR stands for Canadian Institute For Advanced Research and 10 refers to 10 classes. The CIFIR 10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

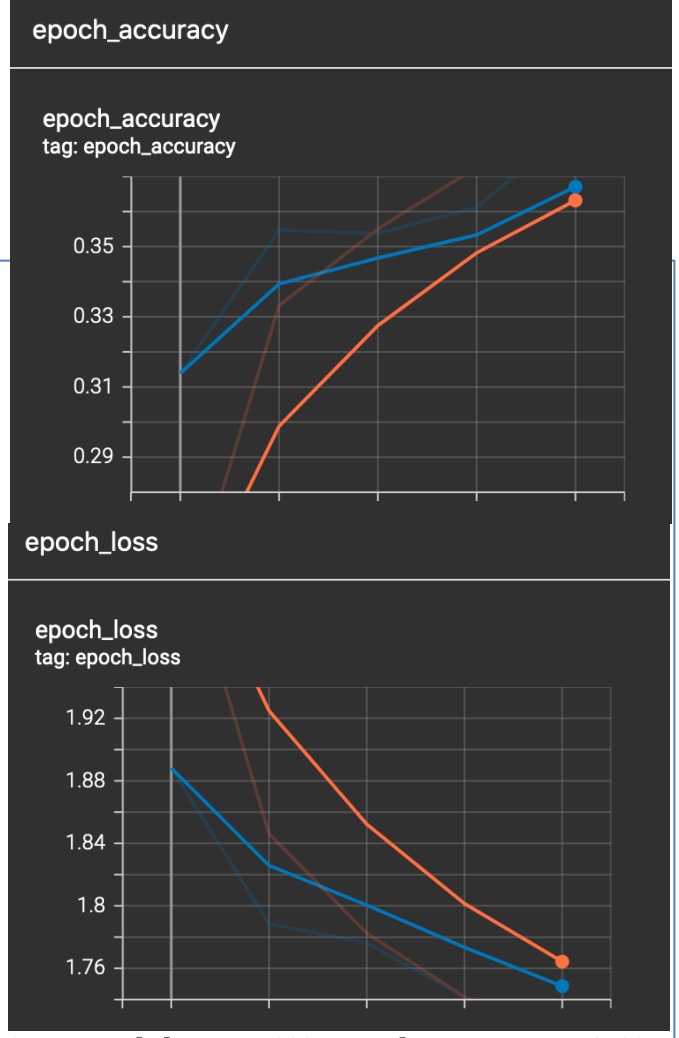
Convert CIFIR-10 images to greyscale images or reshape the CIFIR-10 images to vectors similar to that of MNIST dataset.

Modify the simple MLP TensorFlow tutorial which uses the MNIST data to train the modified CIFIR 10 dataset. Plot the loss and accuracy of the training and testing results.

Helpful web links:
https://www.tensorflow.org/tutorials/quickstart/beginner
https://www.tensorflow.org/api_docs/python/tf/image/rgb_to_grayscale
https://datascience.stackexchange.com/questions/24459/how-to-give-cifar-10-as-an-input-to-mlp

```
import tensorflow as tf
print(tf.__version__)
import datetime, os

%load_ext tensorboard

cifar10 = tf.keras.datasets.cifar10 # mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(32, 32, 3)),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(10,activation='softmax')
])

model.compile(optimizer='sgd',loss='SparseCategoricalCrossentropy',
metrics=['accuracy'])
model.summary()

logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)

model.fit(x_train, y_train, epochs=5, batch_size=50,validation_data=(x_test,
y_test), callbacks=[tensorboard_callback])

model.evaluate(x_test, y_test, verbose=2)

%tensorboard --logdir logs
```

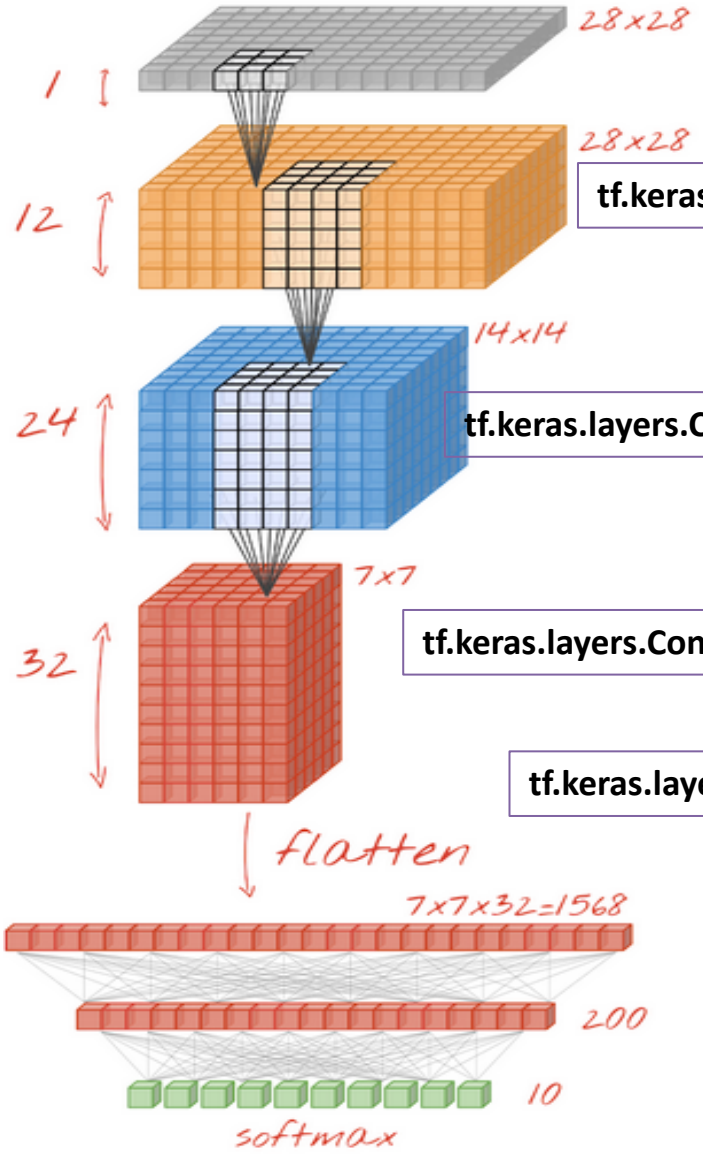
```
2.6.0
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 2s 0us/step
170508288/170498071 [==============================] - 2s 0us/step
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 3072)              0
_________________________________________________________________
dense (Dense)                (None, 16)                49168
_________________________________________________________________
dense_1 (Dense)              (None, 16)                272
_________________________________________________________________
dense_2 (Dense)              (None, 10)                170
=================================================================
Total params: 49,610
Trainable params: 49,610
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
1000/1000 [==============================] - 4s 3ms/step - loss: 2.0560 - accuracy: 0.2415 - val_loss: 1.8881 - val_accuracy: 0.3139
Epoch 2/5
1000/1000 [==============================] - 3s 3ms/step - loss: 1.8462 - accuracy: 0.3332 - val_loss: 1.7885 - val_accuracy: 0.3547
Epoch 3/5
1000/1000 [==============================] - 3s 3ms/step - loss: 1.7825 - accuracy: 0.3550 - val_loss: 1.7762 - val_accuracy: 0.3538
Epoch 4/5
1000/1000 [==============================] - 3s 3ms/step - loss: 1.7417 - accuracy: 0.3726 - val_loss: 1.7417 - val_accuracy: 0.3611
Epoch 5/5
1000/1000 [==============================] - 3s 3ms/step - loss: 1.7157 - accuracy: 0.3827 - val_loss: 1.7162 - val_accuracy: 0.3849
313/313 - 1s - loss: 1.7162 - accuracy: 0.3849
```



Convolution NN : MNIST 28x28 Pixels

[FW, FH, C, H] = [filter size ? X ?, input channel (Depth), output channel (no. of filter)]

tf.keras.layers.Reshape(input_shape=(28*28,), target_shape=(28, 28, 1))



Convolutional 3x3 filters=12

$W_1[3, 3, 1, 12]$

tf.keras.layers.Conv2D(kernel_size=3, filters=12, padding='same', activation='relu')

Convolutional 6x6 filters=24

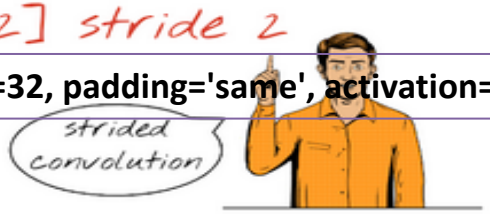
$W_2[6, 6, 12, 24]$ stride 2

tf.keras.layers.Conv2D(kernel_size=6, filters=24, padding='same', activation='relu',strides=2)

Convolutional 6x6 filters=32

$W_2[6, 6, 24, 32]$ stride 2

tf.keras.layers.Conv2D(kernel_size=6, filters=32, padding='same', activation='relu', strides=2)



tf.keras.layers.Flatten()

https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#0

Dense layer

$W_4[1568, 200]$

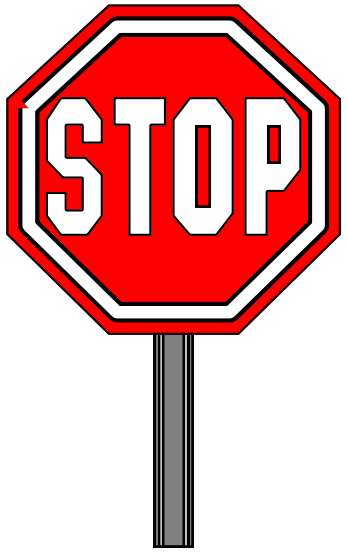
tf.keras.layers.Dense(200, activation='relu')

Softmax dense layer

$W_5[200, 10]$

tf.keras.layers.Dense(10, activation='softmax')

The End



- The End!