מבוא ללמידה עמוקה – תרגיל 1

מטרת התרגיל:

לאפשר לסטודנט להתנסות בבניית רשת פשוטה "מאפס" ולהקנות היכרות מעמיקה עם הרכיבים השונים של תהליכי ה-forward/backward propagation.

הגדרות והגשה:

1) מועד הגשת התרגיל: 26/04/2018
2) התרגיל יתבצע בשפת התכנות python (גירסה 3.6). ההגשה תתבצע בזוגות אלא אם ניתן אישור מוקדם ע"י המרצה. תרגילים שיוגשו באופן אחר לא יבדקו.
3) חל איסור להעתיק קוד מוכן ממקורות קיימים (GitHub וכו').
4) ניתן ליישם פונקציות עזר נוספות מעבר לאלה שהוגדרו. יש להוסיף הסברים לפונקציות אלה.
5) יש ליישם את האלגוריתם תוך שימוש ב-vectorization כפי שהוגדר בהרצאה 1.
6) הקוד כולו יוגש כקובץ יחיד.

הוראות:

1) ישמו את הפונקציות הבאות המיישמות את שלב ה-forward propagation:

a) initialize_parameters(layer_dims)

input: an array of the dimensions of each layer in the network (layer 0 is the size of the flattened input, layer L is the output sigmoid)

output: a dictionary containing the initialized *W* and *b* parameters of each layer (W1…WL, b1…bL).

Hint: Use the *randn* and *zeros* functions of numpy to initialize W and b, respectively

b) linear_forward(A, W, b)

Description:
Implement the linear part of a layer's forward propagation.

input:
A – the activations of the previous layer
W – the weight matrix of the current layer (of shape [size of current layer, size of previous layer])
B – the bias vector of the current layer (of shape [size of current layer, 1])

Output:
Z – the linear component of the activation function (i.e., the value before applying the non-linear function)

linear_cache – a dictionary containing *A*, *W*, *b* and *Z* (stored for making the backpropagation easier to compute)

c) sigmoid(Z)

Input:
Z – the linear component of the activation function

Output:
A – the activations of the layer
activation_cache – returns Z, which will be useful for the backpropagation

d) relu(Z)

Input:
Z – the linear component of the activation function

Output:
A – the activations of the layer
activation_cache – returns Z, which will be useful for the backpropagation

e) linear_activation_forward(A_prev, W, B, activation)

Description:
Implement the forward propagation for the LINEAR->ACTIVATION layer

Input:
A_prev – activations of the previous layer
W – the weights matrix of the current layer
B – the bias vector of the current layer
Activation – the activation function to be used (a string, either "sigmoid" or "relu")

Output:
A – the activations of the current layer
linear_cache – the dictionary generated by the *linear_forward* function

f) L_model_forward(X, parameters)

Description:
Implement forward propagation for the [LINEAR->RELU]*(L-1)->LINEAR->SIGMOID computation

Input:

X – the data, numpy array of shape (input size, number of examples)
parameters – the initialized W and b parameters of each layer

Output:

AL – the last post-activation value

caches – a list of all the cache objects generated by the *linear_forward* function

g) compute_cost(AL, Y)

Description:
Implement the cost function defined by equation
$cost = -\frac{1}{m} * \sum_1^m [(y^i * \log(AL)) + ((1 - y^i) * (1 - AL))]$ (see the slides of the first lecture for additional information if needed).

Input:
AL – probability vector corresponding to your label predictions, shape (1, number of examples)
Y – the labels vector (i.e. the ground truth)

Output:
cost – the cross-entropy cost

2) יישמו את הפונקציות הבאות המיישמות את שלב ה-backward propagation:

a) linear_backward(dZ, cache)

Description:
Implements the linear part of the backward propagation process for a single layer

Input:
dZ – the gradient of the cost with respect to the linear output of the current layer (layer l)
cache – tuple of values (A_prev, W, b) coming from the forward propagation in the current layer

Output:
dA_prev -- Gradient of the cost with respect to the activation (of the previous layer l-1), same shape as A_prev
dW -- Gradient of the cost with respect to W (current layer l), same shape as W
db -- Gradient of the cost with respect to b (current layer l), same shape as b

b) linear_activation_backward(dA, cache, activation)

Description:
Implements the backward propagation for the LINEAR->ACTIVATION layer. The function first computes dZ and then applies the linear_backward function.

Some comments:

- The derivative of ReLU is $f'(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & otherwise \end{cases}$
- The Sigmoid function is $\sigma(x) = \frac{1}{1+e^{-x}}$ and its derivative is $\sigma'(x) = \sigma(x) * (1 - \sigma(x))$
- You should use the activations cache created earlier for the calculation of the activation derivative and the linear cache should be fed to the linear_backward function

Input:

dA – post activation gradient of the current layer

cache – contains both the linear cache and the activations cache

Output:

dA_prev – Gradient of the cost with respect to the activation (of the previous layer l-1), same shape as A_prev
dW – Gradient of the cost with respect to W (current layer l), same shape as W
db – Gradient of the cost with respect to b (current layer l), same shape as b

c) relu_backward (dA, activation_cache)

Description:
Implements backward propagation for a ReLU unit

Input:
dA – the post-activation gradient
activation_cache – contains Z (stored during the forward propagation)

Output:
dZ – gradient of the cost with respect to Z

d) sigmoid_backward (dA, activation_cache)

Description:
Implements backward propagation for a sigmoid unit

Input:
dA – the post-activation gradient
activation_cache – contains Z (stored during the forward propagation)

Output:
dZ – gradient of the cost with respect to Z

e) L_model_backward(AL, Y, caches)

Description:

Implement the backward propagation process for the entire network.
Some comments:

- The backpropagation for the Sigmoid should be done separately (because there is only one like it), and the process for the ReLU layers should be done in a loop
- The derivative for the output of the softmax layer can be calculated using:
  dAL = - (np.divide(Y, AL) - np.divide(1 - Y, 1 - AL))

Input:

AL – the probabilities vector, the output of the forward propagation (L_model_forward)

Y – the true labels vector (the "ground truth" – true classifications)

Caches – list of caches containing for each layer: a) the linear cache; b) the activation cache

Output:

Grads – a dictionary with the gradients

grads["dA" + str(l)] = ...

grads["dW" + str(l)] = ...

grads["db" + str(l)] = ...

f) Update_parameters(parameters, grads, learning_rate)

Description:

Updates parameters using gradient descent

Input:

parameters – a python dictionary containing the DNN architecture's parameters

grads – a python dictionary containing the gradients (generated by L_model_backward)

learning_rate – the learning rate used to update the parameters (the "alpha")

Output:

parameters – the updated values of the parameters object provided as input

3) יישמו את הפונקציות המשתמשות בפונקציונאליות שיישמתם בשלבים הקודמים על מנת לאמן את הרשת בשלמותה ולספק תחזיות:

a) L_layer_model(X, Y, layers_dims, learning_rate, num_iterations)

Description:
Implements a L-layer neural network. All layers but the last should have the ReLU activation function, and the final layer will apply the sigmoid activation function. The network should only address binary classification.

Hint: the function should use the earlier functions in the following order: initialize -> L_model_forward -> compute_cost -> L_model_backward -> update parameters

Input:
X – the input data, a numpy array of shape (height*width , number_of_examples)
*Comment: since the input is in grayscale we only have height and width, otherwise it would have been height*width*3*

Y – the "real" labels of the data, a vector of shape (1, number of examples)

Layer_dims – a list containing the dimensions of each layer, including the input

Output:
parameters – the parameters learnt by the system during the training (the same parameters that were updated in the update_parameters function).

costs – the values of the cost function (calculated by the compute_cost function). One value is to be saved after each 100 training iterations (e.g. 3000 iterations -> 30 values).

b) Predict(X, Y, parameters)

Description:
The function receives an input data and the true labels and calculates the accuracy of the trained neural network on the data.

Input:
X – the input data, a numpy array of shape (height*width, number_of_examples)

Y – the "real" labels of the data, a vector of shape (1, number of examples)

Parameters – a python dictionary containing the DNN architecture's parameters

Output:
accuracy – the accuracy measure of the neural net on the provided data

4) הפעילו את הקוד שכתבתם על מנת לסווג את הדאטסט MNIST והציגו דו"ח סיכום
א) הורידו את הנתונים מהכתובת http://yann.lecun.com/exdb/mnist/

ב) מאחר שבניתם רשת לסיווג בינארי (שני קלאסים בלבד), יש להריץ שני סטים של ניסויים – אחד הספרות 3 ו-8, השני על הספרות 7 ו-9. <u>יש לחלץ את הדגימות הרלוונטיות מתוך סט האימון והמבחן</u>. את כל הפלט המתבקש בסעיפים הבאים יש להוציא עבור כל אחד משני הניסויים.

ג) הריצו את הרשת שלכם עם הקונפיגורציה הבאה:

- 4 שכבות (לא כולל שכבת הקלט) בגדלים הבאים: 20,7,5,1
- את הקלט יש "לשטח" לכדי וקטור מהצורה [784,m], כאשר m מייצג את מספר הדוגמאות
- יש להריץ את הקוד 3000 איטרציות עם learning_rate=0.009

ד) עבור כל אחד משני הניסויים המפורטים, יש להוציא את הפלטים הבאים:

- ערכי ה-accuracy עבור סט האימון וסט המבחן
- ערך ה-cost עבור כל 100 איטרציות אימון (נא לדאוג שיופיע גם אינדקס האיטרציה בנוסף לערך). נא לבצע את ההדפסה מתוך פונקציית L_layer_model

ה) את הנתונים המבוקשים למעלה יש להציג בדו"ח מסכם (מסמך וורד) שיועלה יחד עם הקוד.

בהצלחה!