Ex3- Machine learning to medical data

Due to: 30.5.2016

עידכונים מ22.5.2016

מטרת תרגיל היא להתנסות בסיווג תמונות ותמונות רפואיות תוך שימוש neural network

הנתונים לתרגיל

(1) MNIST
http://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html

(2) CAFIR-10
https://www.cs.toronto.edu/~kriz/cifar.html

(3) הנתונים כאמור יהיו תמונות Cyst. אפשר להוריד את סט התמונת מכאן.
https://www.dropbox.com/sh/dg89oqt90gfjq9b/AACjXkbLF-wLkkU0HAlF1ojha?dl=0
כאן כל תמונה היא patch קטן בגודל של [39x39] מתוך סריקת CT שלמה.

טיפ לא להגשה: ודעו לעצמכם שאתם שולטים בקוד ומבינים בו כל פסיק. (זה חשוב...)

קוד התרגיל מבוסס על ההסבר באתר הזה:
/http://peterroelants.github.io/posts/neural_network_implementation_part01
תשתמשו בזה, זה יכול לעזור לכם

הקלט לתוכנית:

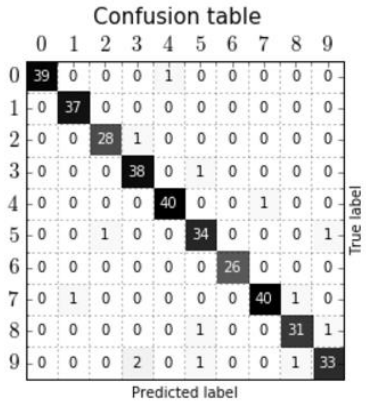
התוכנית תקלות שני מספרים

(1) מספר שיציין איזה סט תמונת לטעון (1,2 או 3).
(2) ומספר קונפיגורציה (יש לנסות 9 שיטות שונות – 3 לכול סט).
בקונפיגורציה הכוונה: קצב לימוש שונה, גודל רשת שונה (עומק רוחב) וכו'

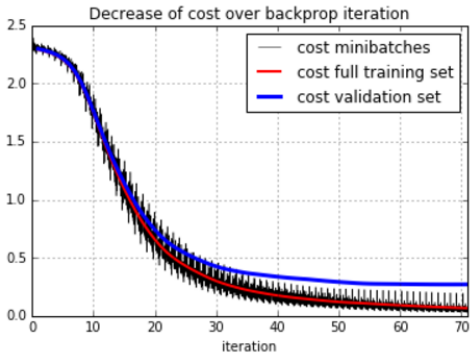
יש ליצר בעצמכם Main לתרגיל

הפלט יהיה:

(1) גרף של אחוז הדיוק הסופי לכול קבוצה על קבוצת הtest (אותם 20% שהשארנו לגמרי בצד)
נדגים עבור 10 קבוצות. (הנתונים של MNIST) העמודות והשורות מציינים את הקטגוריה (סיפרה) 0 עד 9. והערכים בתוך הטבלה מציינים כמה איברים סווגו לתוך כל אחת מהקבוצות. למשל אנו רואים שבשורה 2 עמודה 3 רשום 1. הכוונה שמדובר בספרה 2 שתויגה (בטעות) כספרה 2. לעומת זאת בשורה 2 עמודה 2 רשום 28 כלומר 28 ספרות תיוגו בצורה נכונה.



(2) גרף של הloss/cost במהלך האימון – צריך להדגים גרף ללא overfitting לפחות עבור קונפיגורצה אחת לכול דאטא סט. לדוגמא:



זהו קורס מדעי בנושא מערכות לומדות ולא קורס תכנות בלבד, על כן התרגיל אינו תרגיל תכנות "רגיל". חלק חשוב בהגשה (ובציון) הוא הדוח המלווה, אותו יש להגיש בפורמט PDF.

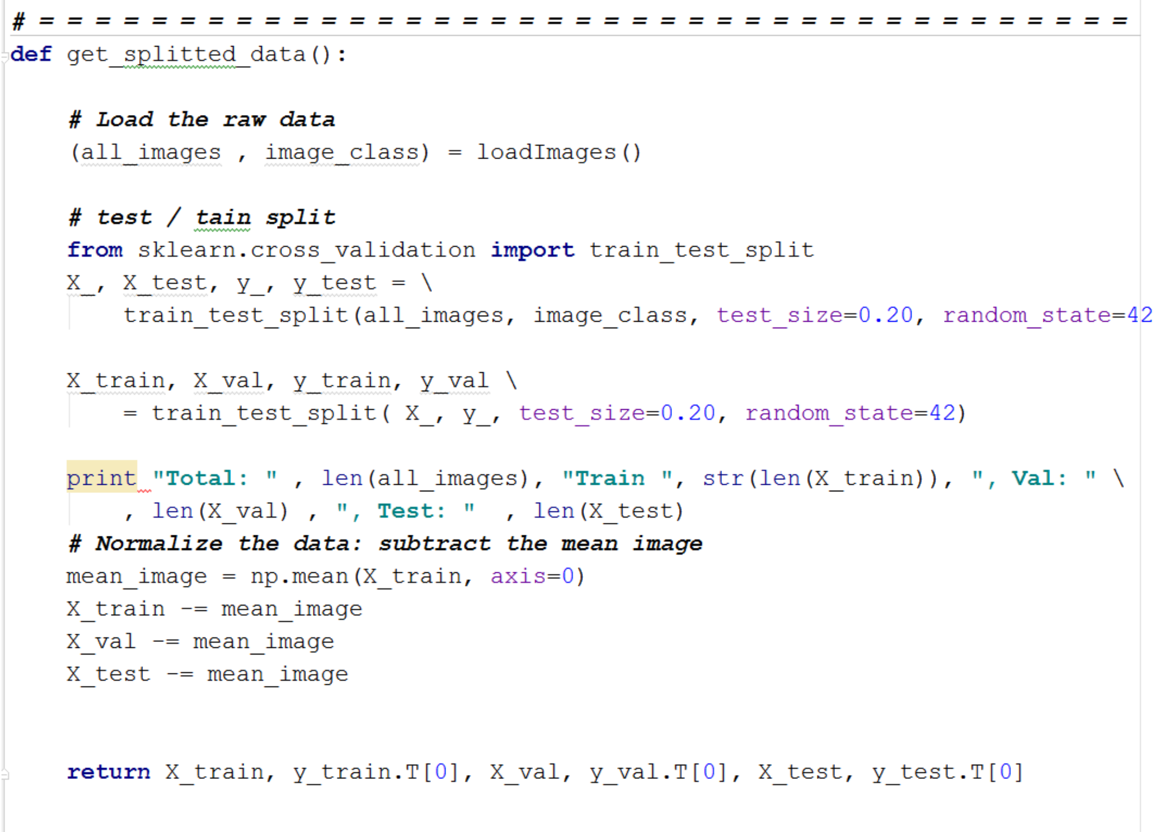
a. מה יהיה בדוח? הסברים! הקובץ יכלול את קטעי הקוד המרכזים והסברים + פלט וגרפים.
b. נבהיר, כל הפלט, התוצאות והגרפים צריכים להיות כלולים בדוח, כך שנוכל להעריך אותם בקריאה רצופה, בלי להריץ את הקוד שלכם בזמן הקריאה. כמובן שאחר כך נריץ ונסתכל על הקוד, וגם זה נלקח בחשבון בציון.
c. על ההסברים בדוח להיות עד 10 עמודים (חסם עליון ביותר) -- לא כולל הקוד (אותו אפשר להוסיף בסוף כנספח).

הסבר על הקוד

הקוד על-מנת לטעון את התמונות לתוך datafaem ב python הוא:

```python
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
def loadImagesByList( file_pattern ):
    image_list = map(Image.open,glob.glob(file_pattern))
    imSizeAsVector = image_list[0].size[0] * image_list[0].size[1]
    images = np.zeros([len(image_list),imSizeAsVector])
    for idx, im in enumerate(image_list):
        images[idx,:] = np.array(im, np.uint8).reshape(imSizeAsVector,1).T

    return images

# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
def loadImages():
    """..."""
    OK_file_pattern = '*OK*axial.png'
    Cyst_file_pattern = '*Cyst*axial.png'


    # Ok-images
    OK_image = loadImagesByList(OK_file_pattern )

    # Cyst-images
    Cyst_image = loadImagesByList(Cyst_file_pattern )

    #concatenate the two types
    image_class = np.concatenate( ( np.zeros([OK_image.shape[0],1]) ,
                                    np.ones([Cyst_image.shape[0],1]) ) )
    all_images = np.concatenate((OK_image,Cyst_image ))
    return (all_images , image_class)
```

הקוד אשר מפצל את הנתונים ל test ו train ו val



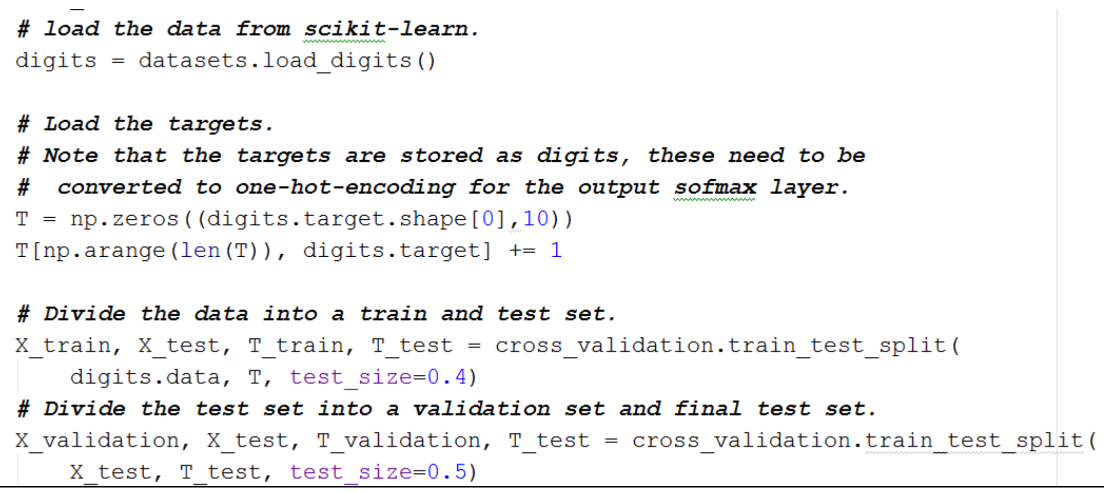
```
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
def get_splitted_data():

    # Load the raw data
    (all_images , image_class) = loadImages()

    # test / tain split
    from sklearn.cross_validation import train_test_split
    X_, X_test, y_, y_test = \
        train_test_split(all_images, image_class, test_size=0.20, random_state=42

    X_train, X_val, y_train, y_val \
        = train_test_split( X_, y_, test_size=0.20, random_state=42)

    print "Total: " , len(all_images), "Train ", str(len(X_train)), ", Val: " \
        , len(X_val) , ", Test: "  , len(X_test)
    # Normalize the data: subtract the mean image
    mean_image = np.mean(X_train, axis=0)
    X_train -= mean_image
    X_val -= mean_image
    X_test -= mean_image


    return X_train, y_train.T[0], X_val, y_val.T[0], X_test, y_test.T[0]
```

הנתונים של MNIST

הקוד שלהלן טוען את הנתונים של MNIST, (אילו נתונים שמגיעים מוכנים בscikit-learn הם נמצאים תחת הclass של datasets)

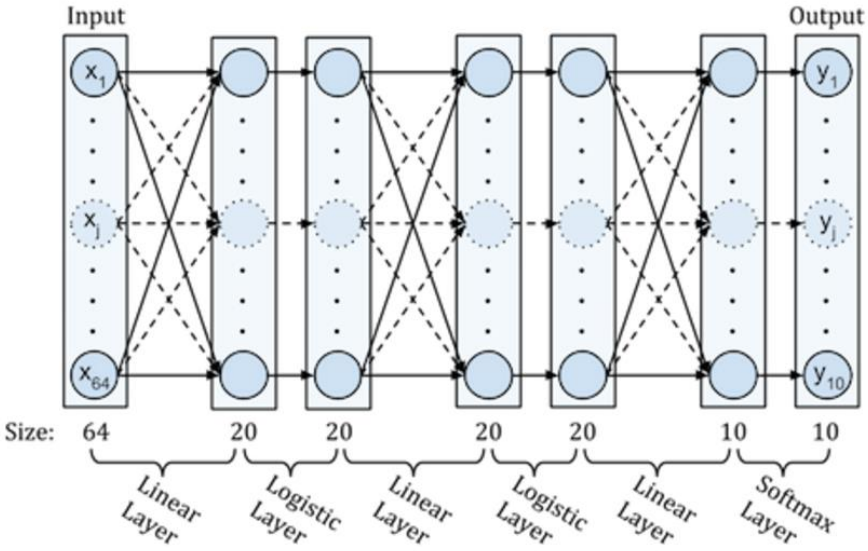
כאשר ברצונכם לטעון את הנתונים של MNIST צריך להחליף את תוכן הפונקציה get_splited_data בקוד זה.

Import scikit



```
# load the data from scikit-learn.
digits = datasets.load_digits()

# Load the targets.
# Note that the targets are stored as digits, these need to be
#  converted to one-hot-encoding for the output sofmax layer.
T = np.zeros((digits.target.shape[0],10))
T[np.arange(len(T)), digits.target] += 1

# Divide the data into a train and test set.
X_train, X_test, T_train, T_test = cross_validation.train_test_split(
    digits.data, T, test_size=0.4)
# Divide the test set into a validation set and final test set.
X_validation, X_test, T_validation, T_test = cross_validation.train_test_split(
    X_test, T_test, test_size=0.5)
```

הרשת


Input
Output
x_1
x_j
x_{64}
y_1
y_j
y_{10}
Size: 64 20 20 20 20 10 10
Linear Layer
Logistic Layer
Linear Layer
Logistic Layer
Linear Layer
Softmax Layer

ראשית נגדיר את הפונקציה הלוגיסטית ואת הנגזרת שלה
(את הנגזרת אנו נצטרך לשלב האימון)

```
def logistic(z):
    return 1 / (1 + np.exp(-z))

def logistic_deriv(y):  # Derivative of logistic function
    return np.multiply(y, (1 - y))

def softmax(z):
    return np.exp(z) / np.sum(np.exp(z), axis=1, keepdims=True)
```

נגדיר מחלקה אבסטראקטית בשם layer. מחלקה אבסטראקטית ראשונה.

```
class Layer(object):
    """Base class for the different layers.
    Defines base methods and documentation of methods."""

    def get_params_iter(self):
        """Return an iterator over the parameters (if any).
        The iterator has the same order as get_params_grad.
        The elements returned by the iterator are editable in-place."""
        return []

    def get_params_grad(self, X, output_grad):
        """Return a list of gradients over the parameters.
        The list has the same order as the get_params_iter iterator.
        X is the input. output_grad is the gradient at the output of this layer.
        """
        return []

    def get_output(self, X):
        """Perform the forward step linear transformation. X is the input."""
        pass

    def get_input_grad(self, Y, output_grad=None, T=None):
        """Return the gradient at the inputs of this layer.
        Y is the pre-computed output of this layer (not needed in this case).
        output_grad is the gradient at the output of this layer
         (gradient at input of next layer).   Output layer uses targets T to comput
         output error instead of output_grad"""
        pass
```

מימוש רשת לינארית.

(1) ה__init__ מאתחל רשת ע"פ גודל הפרמטרים שקיבל.

(2) get_params_iter – כשמו כן הוא..מחזיר איטרטור עבור הפרמטרים של השיכבה שהם: המשקולות W והbias – b.

(3) get_output – זה שלב הforward – כמו שאנו כבר מכירים מהניורון הבודד \ סיווג נתון חדש. פשוט מכפילים את הנתון אשר נמצא x במשקולות w ומוסיפים את הbias.

(4) get_params_grad – מחזיר רשימה עם גרדיאנטים של הפרמטרים

(5) get_input_grad – פונקציה זו משמשת את תהליך ה back-propgation בהינתן הגרדיאנט ביציאה של השיכבה היא מכפילה במשקולות על מנת לחשב את הגראדיאנטים בכניסה של השכיבה. (בשפה חופשית נגיד שהפונקציה "מפעפעת" את השגיאה מהפלט לכניסות שלה.) במקרה של שיכבה לינארית מדובר למעשה בהכפלה של הפלט במשקולות.
זו פונקציה שלמעשה עובדת בכיוון ההפוך של get_output

```
class LinearLayer(Layer):
    """The linear layer performs a linear transformation to its input."""

    def __init__(self, n_in, n_out):
        """Initialize hidden layer parameters.
        n_in is the number of input variables.
        n_out is the number of output variables."""
        self.W = np.random.randn(n_in, n_out) * 0.1
        self.b = np.zeros(n_out)

    def get_params_iter(self):
        """Return an iterator over the parameters."""
        return itertools.chain(np.nditer(self.W, op_flags=['readwrite']),
                               np.nditer(self.b, op_flags=['readwrite']))

    def get_output(self, X):
        """Perform the forward step linear transformation."""
        return X.dot(self.W) + self.b

    def get_params_grad(self, X, output_grad):
        """Return a list of gradients over the parameters."""
        JW = X.T.dot(output_grad)
        Jb = np.sum(output_grad, axis=0)
        return [g for g in itertools.chain(np.nditer(JW), np.nditer(Jb))]

    def get_input_grad(self, Y, output_grad):
        """Return the gradient at the inputs of this layer."""
        return output_grad.dot(self.W.T)
```

מימוש השיכבה הלוגיסטית. לשיכבה זו שתי פונקציות לשלב הforward ולשלב הbackword

(1) ה get_output זה הפעלה של הפונק' הלוגיסטית קדימה בשלב

(2) הget_input_grad – פונק' אשר מחשב את הנגזרות בכניסה של השיכבה. בהינתן הפלט (Y) והשגיאה (output_grad) – הפונקציה מחשב את הגרדיאנט בכניסה. הסבירו את הפיתוח של פונק' זו.

```
class LogisticLayer(Layer):
    """The logistic layer applies the logistic function to its inputs."""

    def get_output(self, X):
        """Perform the forward step transformation."""
        return logistic(X)

    def get_input_grad(self, Y, output_grad):
        """Return the gradient at the inputs of this layer."""
        return np.multiply(logistic_deriv(Y), output_grad)
```

מימוש שיכבת softMAX

- get_output הוא למעשה שלב הforward בו מעברים את הנתונים דרך פונקצית הsoft שהגדרנו לעיל.
- get_input_grad זו הנגזרת של הפונקצית ה- softmax אותה אנו צריכים לשלב הלימוד - שלב הback-propagation.

נזכור שב softmax אנו משתמשים רק בסוף הרשת פעם אחד. אם נסתכל על הגרדיאנט שחושב (השגיאה) נראה שלמעשה זה הפלט של הרשת (Y) פחות הפלט המצופה (T) – שזה בדיוק מה שהיו מצפים לשגיאה של הרשת כולה ואכן זה כך.

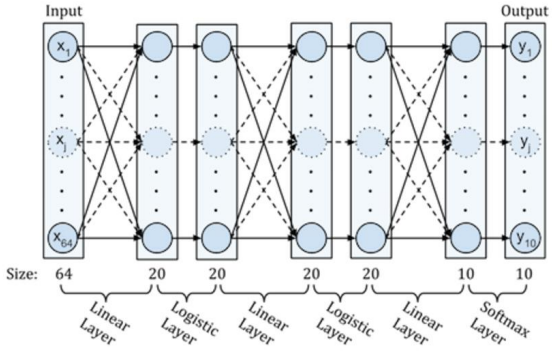
```
class SoftmaxOutputLayer(Layer):
    """
    The softmax output layer computes the classification
    propabilities at the output.
    """

    def get_output(self, X):
        """Perform the forward step transformation."""
        return softmax(X)

    def get_input_grad(self, Y, T):
        """Return the gradient at the inputs of this layer."""
        return (Y - T) / Y.shape[0]

    def get_cost(self, Y, T):
        """Return the cost at the output of this output layer."""
        dim_ = 1
        return - np.multiply(T, np.log(Y)).sum() / Y.shape[dim_]
```

בניית הרשת עצמה ע"פ התרשים לעיל.

שברשת המוצגת, השכבה הראשונה ברשת היא בעלת 64 כניסות והאחרונה בעלת 10 יציאות. זו רשת המתאימה לנתונים של MNIST. שם כל תמונה מורכבת מ64 פיקסלים וישנם 10 קטגוריות של תמונות.

שימו לב למשל שבנתונים של הCyst יש צורך לעדכן את הרשת ל1521 כניסות – כי כל patch הוא בגודל 39*39 ומספר היציאות הינו 2 (ריקמה בריאה או חולה)

כמה כניסות ויציאות יש ברשת של CIFAR ?



בניית הרשת: המערך layers מכיל את השכבות שכבות השונות. ואנו מכניסים לו שיכבה אחרי שיכבה (בלולאה).
בהתחלה את שתי השכבות המלאות כאשר בינהם ישנה שיכה 'לוגיסטית'. והשכבה האחרונה מוציאה ניורונים כמספר המחלקות – בסוף נעשה סיווג סופי בעזרת softmax. אנו רואים כיצד הכתיבה האבסטראקטית של המחלקות עזרה לנו, וכעת בניית הרשת מאוד פשוטה.

(זה בmain)

```
# Define a sample model to be trained on the data
hidden_neurons_1 = 20  # Number of neurons in the first hidden-layer
hidden_neurons_2 = 20  # Number of neurons in the second hidden-layer
# Create the model
layers = [] # Define a list of layers
# Add first hidden layer
layers.append(LinearLayer(X_train.shape[1], hidden_neurons_1))
layers.append(LogisticLayer())
# Add second hidden layer
layers.append(LinearLayer(hidden_neurons_1, hidden_neurons_2))
layers.append(LogisticLayer())
# Add output layer
layers.append(LinearLayer(hidden_neurons_2, T_train.shape[1]))
layers.append(SoftmaxOutputLayer())
```

שלב הforward בו אנו מעבירים את הנתונים שיכבה אחרי שיכבה מהכינסה עד היציאה.

```python
# Define the forward propagation step as a method.
def forward_step(input_samples, layers):
    """..."""
    activations = [input_samples] # List of layer activations
    # Compute the forward activations for each layer starting from the first
    X = input_samples
    for layer in layers:
        Y = layer.get_output(X)   # Get the output of the current layer
        activations.append(Y)     # Store the output for future processing
        X = activations[-1]       # Set the current input as the
                                  # activations of the previous layer
    return activations            # Return the activations of each layer
```

שלב הbackword זה השלב החשוב ביותר בו אנו מעדכנים את השגיאות (הגרדיאנטים) בכל הרשת. הגרדיאנטים נשמרים בתוך המערך בשם param_grads.

בפונקציה זו אנו עוברים על השכבות בסדר הפוך מהאחרונה לראשונה. עבור כל שיכבה אנו "מפעפעים" את השגיאה מהפלט לכניסות שלה.

(א) מחשבים בכל שיכבה את ה גרדיאנט (נגזרת) ביחס לפלט המצופה - בעזרת הפונק' get_input_grad

(ב)

```python
# Define the backward propagation step as a method
def backward_step(activations, targets, layers):
    """..."""
    param_grads = collections.deque()  # List of parameter gradients for each layer
    output_grad = targets  # The error gradient at the output of the current layer
    # Propagate the error backwards through all the layers.
    #  Use reversed to iterate backwards over the list of layers.
    for layer in reversed(layers):
        Y = activations.pop()  # Get the activations of the last layer on the stack
        # Compute the error at the each layer.
        input_grad = layer.get_input_grad(Y, output_grad)
        # Get the input of this layer (activations of the previous layer)
        X = activations[-1]
        # Compute the layer parameter gradients used to update the parameters
        grads = layer.get_params_grad(X, output_grad)
        param_grads.appendleft(grads)
        # Compute gradient at output of previous layer (input of current layer):
        output_grad = input_grad
    return list(param_grads)  # Return the parameter gradients
```

```python
# Create the minibatches
batch_size = 25  # Approximately 25 samples per batch
nb_of_batches = X_train.shape[0] / batch_size  # Number of batches
XT_batches = zip(
    np.array_split(X_train, nb_of_batches, axis=0),  # X samples
    np.array_split(T_train, nb_of_batches, axis=0))  # Y targets

# Define a method to update the parameters
def update_params(layers, param_grads, learning_rate):
    """..."""
    for layer, layer_backprop_grads in zip(layers, param_grads):
        for param, grad in itertools.izip(layer.get_params_iter()
                , layer_backprop_grads):
            # The parameter returned by the iterator point to the memory space of
            #  the original layer and can thus be modified inplace.
            param -= learning_rate * grad  # Update each parameter
```

```python
# Perform backpropagation
# initalize some lists to store the cost for future analysis
minibatch_costs = []
training_costs = []
validation_costs = []
max_nb_of_iterations = 300  # Train for a maximum of 300 iterations
learning_rate = 0.1  # Gradient descent learning rate

# Train for the maximum number of iterations
for iteration in range(max_nb_of_iterations):
    for X, T in XT_batches:  # For each minibatch sub-iteration
        activations = forward_step(X, layers)  # Get the activations
        minibatch_cost = layers[-1].get_cost(activations[-1], T)  # Get cost
        minibatch_costs.append(minibatch_cost)
        param_grads = backward_step(activations, T, layers)  # Get the gradients
        update_params(layers, param_grads, learning_rate)  # Update the parameters
    # Get full training cost for future analysis (plots)
    activations = forward_step(X_train, layers)
    train_cost = layers[-1].get_cost(activations[-1], T_train)
    training_costs.append(train_cost)
    # Get full validation cost
    activations = forward_step(X_validation, layers)
    validation_cost = layers[-1].get_cost(activations[-1], T_validation)
    validation_costs.append(validation_cost)
    if len(validation_costs) > 3:
        # Stop training if the cost on the validation set doesn't decrease
        #  for 3 iterations
        if validation_costs[-1] >= validation_costs[-2] >= validation_costs[-3]:
            break
nb_of_iterations = iteration + 1  # The number of iterations that have been execute
```

נהלי הגשה כללים:

- יש לכתוב בpython בלבד!

- שם קובץ הקוד יהיה exXq.py
  - **X = מספר תרגיל**
  - **q = מספר השאלה בתרגיל.**

- להיכן לשלוח ?
  - לאתר
- מה לשלוח ?
  - על כל הקבצים להימצא בתיקיה בשם exX-<login-name1> :
    - לדוגמא ex1- assafsp.
    - את התיקייה הזו יש להגיש באופן מכווץ על-ידי ZIP.
  - למייל יש לצרף את הקבצים הבאים:

1. **קובץ ZIP**:

שם הzip הוא exX-<login-name1>. zip.

לדוגמא ex1-assafsp.zip.

2. **קובץ PDF** המכל את הקוד שהגשתם.

- שם קובץ הPDF יהיה exXq.pdf

ואת הREADME המפורט לעיל

(מספר קבצי הPDF כמספר קבצי הקוד אשר הגשתם + REDEME)

- איך ממרים ל PDF ? אפשר ע"י התוכנה הבאה:

http://sourceforge.net/projects/pdfcreator

(במכללה כבר מותקן acrobatPDF) **או בלינק הבא:**

http://www.freepdfconvert.com/pdf-email

- **כאשר**
  - **X = מספר תרגיל**
  - **Y = מספר הסמסטר = 2**
  - **Z = 2016**

- איחור: 2^y, y = (מספר ימי האיחור + 1).
- מי שלא מפריד בקו בין הפונ': 3-
- ב. מי שלא מתעד פונ': 5-
- ה. מי שהפלט שלו קשה להבנה: 5-.
- ו. מי שלא מתעד את הקובץ: 5-