(/)

You have a captain's log due before 2020-06-07 (in 1 day)! Log it now!
(/captain_logs/35487/edit)

0x00. Binary Classification

Specializations - Machine Learning — Supervised Learning

by Alexa Orrico, Software Engineer at Holberton School

weight: 3

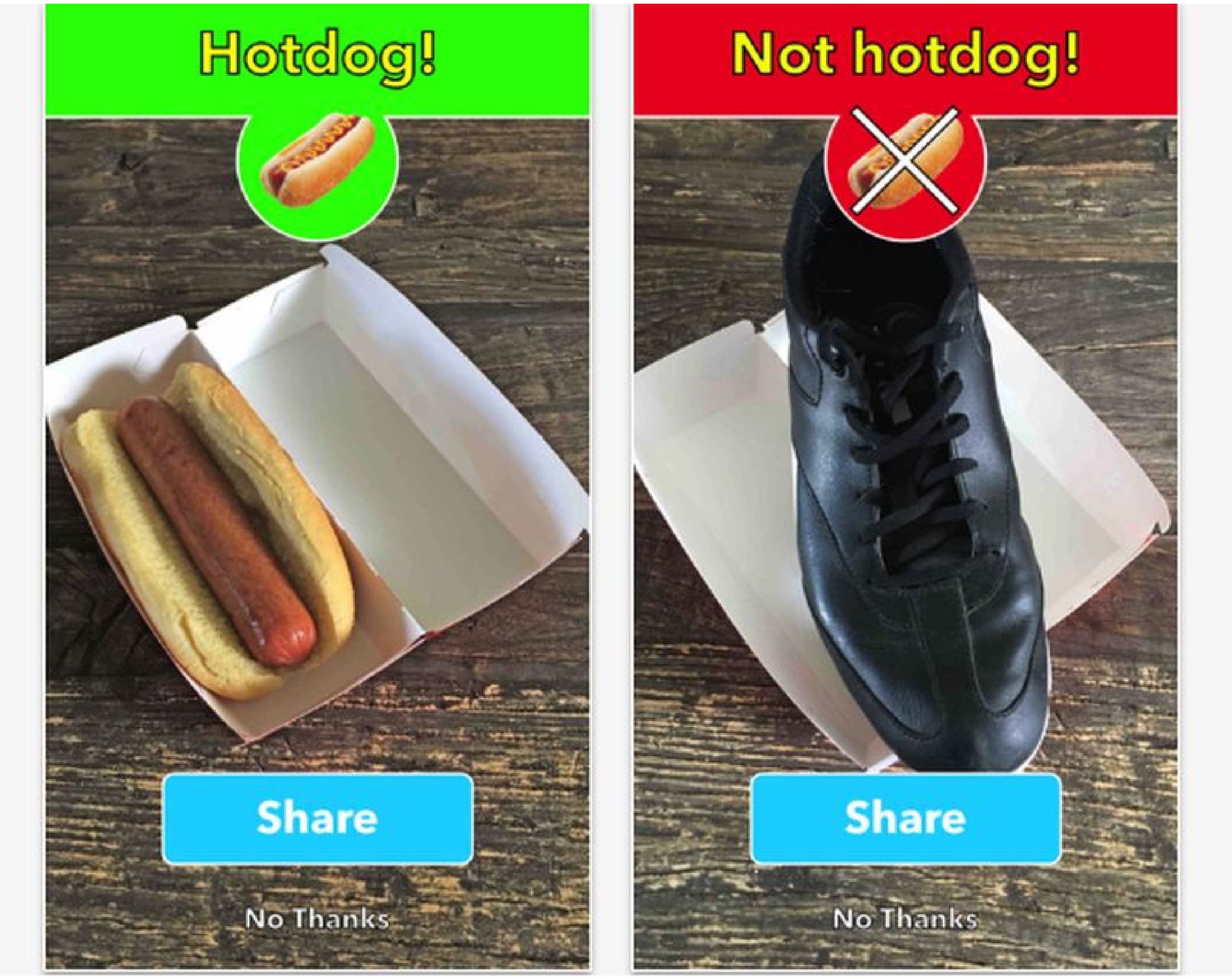
Project over - took place from 02-10-2020 to 02-19-2020 - you're done with 79% of tasks.

☑ Manual QA review was done by Abdel Giovanny Perez on 02-25-2020.

☑ QA review fully automated.

In a nutshell…

- **Manual QA review:** 8.5/42 mandatory
- **Auto QA review:** 142.0/161 mandatory
- **Altogether: 74.14%**
    - Mandatory: 74.14%
    - Optional: no optional tasks



Background Context

Welcome to your first project on supervised learning! At the end of this project, you should be able to build your own binary image classifier from scratch using `numpy`. As you might already see, there are a **LOT** of resources for you to read/watch. It may be tempting to dive into the projects right away, but it is **HIGHLY RECOMMENDED** that you spend **AT LEAST** 1 whole day going over the following materials. You should only start the project once you have a decent understanding of all the topics mentioned in **Learning Objectives**. You may also notice that there are multiple resources that cover the same topic, with some more technical than others. If you find yourself getting lost in a resource, move on to another and come back to the more technical one after you intuitively understand that topic. Good luck and have fun!

Resources

Read or watch:

- Supervised vs. Unsupervised Machine Learning (/rltoken/16mL_gwlZqRa3NAv325YiQ)
- How would you explain neural networks to someone who knows very little about AI or neurology? (/rltoken/A1B1qLJgpKvA4RySpI6r7g)
- Using Neural Nets to Recognize Handwritten Digits (/rltoken/EjjENEVXJKiAZsqSqWMI-w) (*until “A simple network to classify handwritten digits” (excluded)*)
- Forward propagation (/rltoken/da0lo6JbjEDxCbffjRHc7g)
- Understanding Activation Functions in Neural Networks (/rltoken/hz77ChKoiSjMFzi7mgMzBA)
- Loss function (/rltoken/KgRV0-I2LBdQciUXGIeCqQ)
- Gradient descent (/rltoken/7iSJelYELwy7C8cCsGk5hw)
- Calculus on Computational Graphs: Backpropagation (/rltoken/BONZS65eZnIMjngFhr7dPA)
- Backpropagation calculus (/rltoken/Arpa6EFk9q5gD9aJ4Wl5qA)
- What is a Neural Network? (/rltoken/EkncpxTwCUJztJsYI0wciA)
- Supervised Learning with a Neural Network (/rltoken/LoWxJZN-JA0VkV13QQrw1g)
- Binary Classification (/rltoken/cFuQ0hUHg_SpVCHvKrXMzg)
- Logistic Regression (/rltoken/sGIlY030fFNX4nNQidq5fw)
- Logistic Regression Cost Function (/rltoken/xZTVYTU5pSnSK3a7o3OuIQ)
- Gradient Descent (/rltoken/M3YbEr_BqYcILJNz7YzLaQ)
- Computation Graph (/rltoken/X5CelY1ajZt3wHrjtls6Fg)
- Logistic Regression Gradient Descent (/rltoken/HLxYo6tgVumVNRysPUxKNA)
- Vectorization (/rltoken/Zxdbe_-GWZRfXKfs5JM9ig)
- Vectorizing Logistic Regression (/rltoken/HQ9VuO9c4XPJgIm6Nxyn5Q)
- Vectorizing Logistic Regression’s Gradient Computation (/rltoken/RaswXJ2G9LHswV0CypjA0A)
- A Note on Python/Numpy Vectors (/rltoken/wKRb7J-yeA92EF5aEJd3oA)
- Neural Network Representations (/rltoken/JyhRr98YlhACYERu0GncNQ)
- Computing Neural Network Output (/rltoken/Lpcj3uH_6hh8Fp1dXzKkCw)
- Vectorizing Across Multiple Examples (/rltoken/uWY4JFKkT58mrHSBig_f5A)
- Gradient Descent For Neural Networks (/rltoken/Q583jTfE8BfU5hPW1xlDiQ)
- Random Initialization (/rltoken/2uZWU7WaWSQfwGNKlcgZig)
- Deep L-Layer Neural Network (/rltoken/iMyExMqGZGcawyK51FcdzQ)
- Train/Dev/Test Sets (/rltoken/varxWT03Dy39WlyrZBGAVQ)
- Random Initialization For Neural Networks : A Thing Of The Past (/rltoken/gMbQAIqJulDEppMx5rESBQ)
- Initialization of deep networks (/rltoken/ymVrn0IFwxnzoo3WwEZzcQ)
- numpy.zeros (/rltoken/Ho3q2XEingriAApbvjTy6w)
- numpy.random.randn (/rltoken/JFRl_j_ebMhXasgQ_geUug)
- numpy.exp (/rltoken/AjRCO7Yh0JW4lC0xuGmbOg)
- numpy.log (/rltoken/AtD0YS-8Q3Oc0TN1MQ6Wuw)
- numpy.sqrt (/rltoken/Xb26cP7K6Bqtoc_NhQXosg)
- numpy.where (/rltoken/6owcKbc6MWk7JlnlM0AaWA)

Optional:

- Predictive analytics (/rltoken/Bp_21TFndH5cSWLBObZuzQ)
- Maximum Likelihood Estimation (/rltoken/1gmRZd9vF7l3bcIPMnTPaw)

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/yKtIe6fSEklenpymZ18Xrw), **without the help of Google**:

General

- What is a model?
- What is supervised learning?
- What is a prediction?
- What is a node?
- What is a weight?
- What is a bias?
- What are activation functions?
    - Sigmoid?
    - Tanh?
    - Relu?
    - Softmax?
- What is a layer?
- What is a hidden layer?
- What is Logistic Regression?
- What is a loss function?
- What is a cost function?
- What is forward propagation?
- What is Gradient Descent?
- What is back propagation?
- What is a Computation Graph?
- How to initialize weights/biases
- The importance of vectorization
- How to split up your data

Requirements

General

- Allowed editors: `vi`, `vim`, `emacs`
- All your files will be interpreted/compiled on Ubuntu 16.04 LTS using `python3` (version 3.5)
- Your files will be executed with `numpy` (version 1.15)
- All your files should end with a new line
- The first line of all your files should be exactly `#!/usr/bin/env python3`
- A `README.md` file, at the root of the folder of the project, is mandatory
- Your code should use the `pycodestyle` style (version 2.4)
- All your modules should have documentation (`python3 -c 'print(__import__("my_module").__doc__)'`)

- All your classes should have documentation (`python3 -c 'print(__import__("my_module").MyClass.__doc__)'`)
- All your functions (inside and outside a class) should have documentation (`python3 -c 'print(__import__("my_module").my_function.__doc__)'` and `python3 -c 'print(__import__("my_module").MyClass.my_function.__doc__)'`)
- Unless otherwise noted, you are not allowed to import any module except `import numpy as np`
- Unless otherwise noted, you are not allowed to use any loops (`for` , `while` , etc.)
- All your files must be executable
- The length of your files will be tested using `wc`

More Info

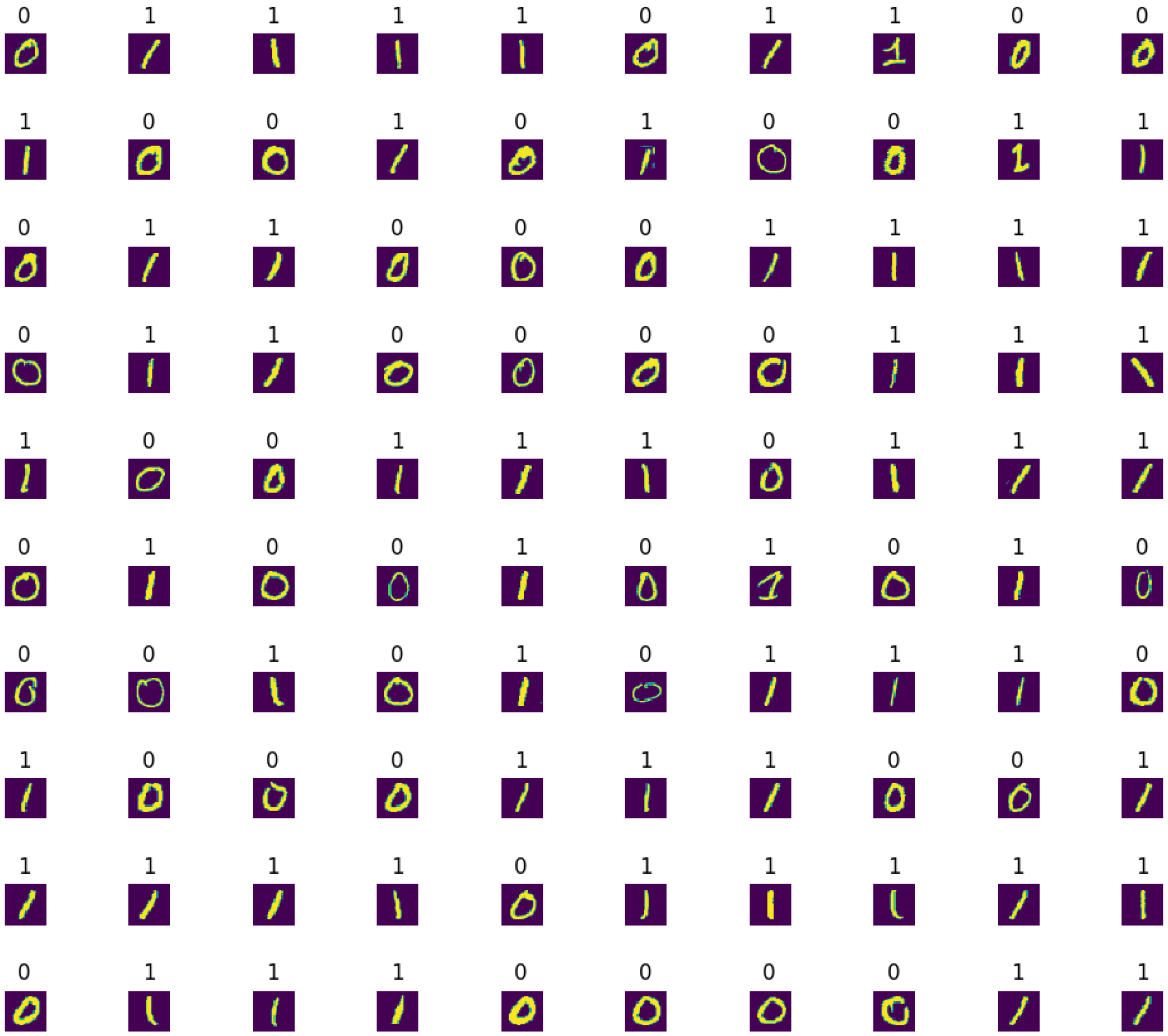
Matrix Multiplications

For all matrix multiplications in the following tasks, please use numpy.matmul (/rltoken/Ox8bY8ogmUftzjR96IrMDw)

Testing your code

In order to test your code, you'll need DATA! Please download these datasets (Binary_Train.npz (https://s3.amazonaws.com/intranet-projects-files/holbertonschool-ml/Binary_Train.npz), Binary_Dev.npz (https://s3.amazonaws.com/intranet-projects-files/holbertonschool-ml/Binary_Dev.npz)) to go along with all of the following main files. You **do not** need to upload these files to GitHub. Your code will not necessarily be tested with these datasets. All of the following code assumes that you have stored all of your datasets in a separate `data` directory.

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat show_data.py
#!/usr/bin/env python3

import matplotlib.pyplot as plt
import numpy as np

lib_train = np.load('../data/Binary_Train.npz')
X_3D, Y = lib_train['X'], lib_train['Y']

fig = plt.figure(figsize=(10, 10))
for i in range(100):
    fig.add_subplot(10, 10, i + 1)
    plt.imshow(X_3D[i])
    plt.title(Y[0, i])
    plt.axis('off')
plt.tight_layout()
plt.show()
alexa@ubuntu-xenial:0x00-binary_classification$ ./show_data.py
```



Tasks

0. Neuron

mandatory

Score: 100.00% (*Checks completed: 100.00%*)

☑ Done!

Help

Write a class `Neuron` that defines a single neuron performing binary classification:

- class constructor: `def __init__(self, nx):`

    - `nx` is the number of input features to the neuron
        - If `nx` is not an integer, raise a `TypeError` with the exception: `nx must be an integer`
        - If `nx` is less than 1, raise a `ValueError` with the exception: `nx must be a positive integer`
    - All exceptions should be raised in the order listed above
- Public instance attributes:
    - `W`: The weights vector for the neuron. Upon instantiation, it should be initialized using a random normal distribution.
    - `b`: The bias for the neuron. Upon instantiation, it should be initialized to 0.
    - `A`: The activated output of the neuron (prediction). Upon instantiation, it should be initialized to 0.

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 0-main.py
#!/usr/bin/env python3

import numpy as np

Neuron = __import__('0-neuron').Neuron

lib_train = np.load('../data/Binary_Train.npz')
X_3D, Y = lib_train['X'], lib_train['Y']
X = X_3D.reshape((X_3D.shape[0], -1)).T

np.random.seed(0)
neuron = Neuron(X.shape[0])
print(neuron.W)
print(neuron.W.shape)
print(neuron.b)
print(neuron.A)
neuron.A = 10
print(neuron.A)
alexa@ubuntu-xenial:0x00-binary_classification$ ./0-main.py
[[ 1.76405235e+00  4.00157208e-01  9.78737984e-01  2.24089320e+00
   1.86755799e+00 -9.77277880e-01  9.50088418e-01 -1.51357208e-01

...

  -5.85865511e-02 -3.17543094e-01 -1.63242330e+00 -6.71341546e-02
   1.48935596e+00  5.21303748e-01  6.11927193e-01 -1.34149673e+00]]
(1, 784)
0
0
10
alexa@ubuntu-xenial:0x00-binary_classification$
```

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `0-neuron.py`

Check your code? QA Review

1. Privatize Neuron mandatory

Score: 100.00% (*Checks completed: 100.00%*)

☑ Done!

Help

Write a class `Neuron` that defines a single neuron performing binary classification (Based on `0-neuron.py`):

- class constructor: `def __init__(self, nx):`
  - `nx` is the number of input features to the neuron
    - If `nx` is not an integer, raise a `TypeError` with the exception: `nx must be a integer`
    - If `nx` is less than 1, raise a `ValueError` with the exception: `nx must be positive`
  - All exceptions should be raised in the order listed above
- **Private** instance attributes:
  - `__W`: The weights vector for the neuron. Upon instantiation, it should be initialized using a random normal distribution.
  - `__b`: The bias for the neuron. Upon instantiation, it should be initialized to 0.
  - `__A`: The activated output of the neuron (prediction). Upon instantiation, it should be initialized to 0.
  - Each private attribute should have a corresponding getter function (no setter function).

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 1-main.py
#!/usr/bin/env python3

import numpy as np

Neuron = __import__('1-neuron').Neuron

lib_train = np.load('../data/Binary_Train.npz')
X_3D, Y = lib_train['X'], lib_train['Y']
X = X_3D.reshape((X_3D.shape[0], -1)).T

np.random.seed(0)
neuron = Neuron(X.shape[0])
print(neuron.W)
print(neuron.b)
print(neuron.A)
neuron.A = 10
print(neuron.A)
alexa@ubuntu-xenial:0x00-binary_classification$ ./1-main.py
[[ 1.76405235e+00  4.00157208e-01  9.78737984e-01  2.24089320e+00
   1.86755799e+00 -9.77277880e-01  9.50088418e-01 -1.51357208e-01

...

  -5.85865511e-02 -3.17543094e-01 -1.63242330e+00 -6.71341546e-02
   1.48935596e+00  5.21303748e-01  6.11927193e-01 -1.34149673e+00]]
0
0
Traceback (most recent call last):
  File "./1-main.py", line 16, in <module>
    neuron.A = 10
AttributeError: can't set attribute
alexa@ubuntu-xenial:0x00-binary_classification$
```

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `1-neuron.py`

Check your code? QA Review

2. Neuron Forward Propagation mandatory

☑ Done!

Score: 100.00% (*Checks completed: 100.00%*)

Help

Write a class `Neuron` that defines a single neuron performing binary classification (Based on `1-neuron.py`):

- Add the public method `def forward_prop(self, X):`
  - Calculates the forward propagation of the neuron
  - `X` is a `numpy.ndarray` with shape (`nx` , `m`) that contains the input data
    - `nx` is the number of input features to the neuron
    - `m` is the number of examples
  - Updates the private attribute `__A`
  - The neuron should use a sigmoid activation function
  - Returns the private attribute `__A`

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 2-main.py
#!/usr/bin/env python3

import numpy as np

Neuron = __import__('2-neuron').Neuron

lib_train = np.load('../data/Binary_Train.npz')
X_3D, Y = lib_train['X'], lib_train['Y']
X = X_3D.reshape((X_3D.shape[0], -1)).T

np.random.seed(0)
neuron = Neuron(X.shape[0])
neuron._Neuron__b = 1
A = neuron.forward_prop(X)
if (A is neuron.A):
        print(A)
vagrant@ubuntu-xe
alexa@ubuntu-xenial:0x00-binary_classification$ ./2-main.py
[[5.34775247e-10 7.24627778e-04 4.52416436e-07 ... 8.75691930e-05
  1.13141966e-06 6.55799932e-01]]
alexa@ubuntu-xenial:0x00-binary_classification$
```

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `2-neuron.py`

Check your code? QA Review

3. Neuron Cost

mandatory

Score: 100.00% (*Checks completed: 100.00%*)

Write a class `Neuron` that defines a single neuron performing binary classification (Based on `2-neuron.py`):

☑ Done!

Help

- Add the public method `def cost(self, Y, A):`
  - Calculates the cost of the model using logistic regression
  - `Y` is a `numpy.ndarray` with shape (1, `m`) that contains the correct labels for the input data
  - `A` is a `numpy.ndarray` with shape (1, `m`) containing the activated output of the neuron for each example
  - To avoid division by zero errors, please use `1.0000001 - A` instead of `1 - A`
  - Returns the cost

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 3-main.py
#!/usr/bin/env python3

import numpy as np

Neuron = __import__('3-neuron').Neuron

lib_train = np.load('../data/Binary_Train.npz')
X_3D, Y = lib_train['X'], lib_train['Y']
X = X_3D.reshape((X_3D.shape[0], -1)).T

np.random.seed(0)
neuron = Neuron(X.shape[0])
A = neuron.forward_prop(X)
cost = neuron.cost(Y, A)
print(cost)
alexa@ubuntu-xenial:0x00-binary_classification$ ./3-main.py
4.365104944262272
alexa@ubuntu-xenial:0x00-binary_classification$
```

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `3-neuron.py`

Check your code?

QA Review

4. Evaluate Neuron

mandatory

Score: 100.00% (*Checks completed: 100.00%*)

☑ Done!

Help

Write a class `Neuron` that defines a single neuron performing binary classification (Based on `3-neuron.py`):

- Add the public method `def evaluate(self, X, Y):`
    - Evaluates the neuron's predictions
    - `X` is a `numpy.ndarray` with shape (`nx` , `m`) that contains the input data
        - `nx` is the number of input features to the neuron
        - `m` is the number of examples
    - `Y` is a `numpy.ndarray` with shape (1, `m`) that contains the correct labels for the input data
    - Returns the neuron's prediction and the cost of the network, respectively
        - The prediction should be a `numpy.ndarray` with shape (1, `m`) containing the predicted labels for each example
        - The label values should be 1 if the output of the network is >= 0.5 and 0 otherwise

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 4-main.py
#!/usr/bin/env python3

import numpy as np

Neuron = __import__('4-neuron').Neuron

lib_train = np.load('../data/Binary_Train.npz')
X_3D, Y = lib_train['X'], lib_train['Y']
X = X_3D.reshape((X_3D.shape[0], -1)).T

np.random.seed(0)
neuron = Neuron(X.shape[0])
A, cost = neuron.evaluate(X, Y)
print(A)
print(cost)
alexa@ubuntu-xenial:0x00-binary_classification$ ./4-main.py
[[0 0 0 ... 0 0 0]]
4.365104944262272
alexa@ubuntu-xenial:0x00-binary_classification$
```

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `4-neuron.py`

Check your code? QA Review

5. Neuron Gradient Descent mandatory

☑ Done!

Score: 100.00% (*Checks completed: 100.00%*)

Help

Write a class `Neuron` that defines a single neuron performing binary classification (Based on `4-neuron.py`):

- Add the public method `def gradient_descent(self, X, Y, A, alpha=0.05):`
    - Calculates one pass of gradient descent on the neuron
    - `X` is a `numpy.ndarray` with shape (`nx` , `m`) that contains the input data
        - `nx` is the number of input features to the neuron
        - `m` is the number of examples
    - `Y` is a `numpy.ndarray` with shape (1, `m`) that contains the correct labels for the input data
    - `A` is a `numpy.ndarray` with shape (1, `m`) containing the activated output of the neuron for each example
    - `alpha` is the learning rate
    - Updates the private attributes `__W` and `__b`

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 5-main.py
#!/usr/bin/env python3

import numpy as np

Neuron = __import__('5-neuron').Neuron

lib_train = np.load('../data/Binary_Train.npz')
X_3D, Y = lib_train['X'], lib_train['Y']
X = X_3D.reshape((X_3D.shape[0], -1)).T

np.random.seed(0)
neuron = Neuron(X.shape[0])
A = neuron.forward_prop(X)
neuron.gradient_descent(X, Y, A, 0.5)
print(neuron.W)
print(neuron.b)
alexa@ubuntu-xenial:0x00-binary_classification$ ./5-main.py
[[ 1.76405235e+00  4.00157208e-01  9.78737984e-01  2.24089320e+00
   1.86755799e+00 -9.77277880e-01  9.50088418e-01 -1.51357208e-01

...

  -5.85865511e-02 -3.17543094e-01 -1.63242330e+00 -6.71341546e-02
   1.48935596e+00  5.21303748e-01  6.11927193e-01 -1.34149673e+00]]
0.2579495783615682
alexa@ubuntu-xenial:0x00-binary_classification$
```

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `5-neuron.py`

Check your code? QA Review

6. Train Neuron

mandatory

Score: 100.00% (*Checks completed: 100.00%*)

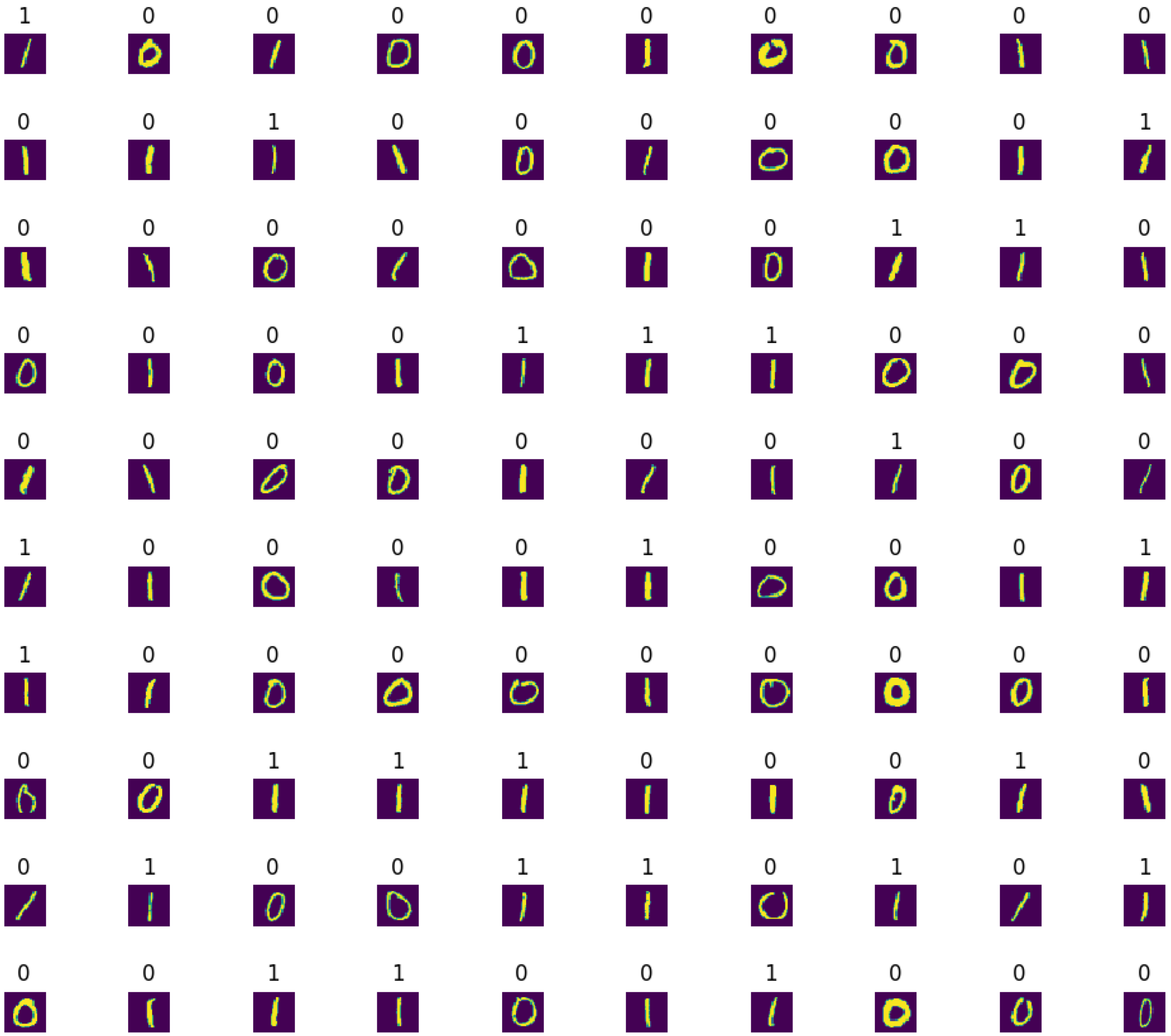
☑ Done!

Help

Write a class `Neuron` that defines a single neuron performing binary classification (Based on `5-neuron.py`):

- Add the public method `def train(self, X, Y, iterations=5000, alpha=0.05):`
    - Trains the neuron
    - `X` is a `numpy.ndarray` with shape (`nx`, `m`) that contains the input data
        - `nx` is the number of input features to the neuron
        - `m` is the number of examples
    - `Y` is a `numpy.ndarray` with shape (1, `m`) that contains the correct labels for the input data
    - `iterations` is the number of iterations to train over
        - if `iterations` is not an integer, raise a `TypeError` with the exception `iterations must be an integer`
        - if `iterations` is not positive, raise a `ValueError` with the exception `iterations must be a positive integer`
    - `alpha` is the learning rate
        - if `alpha` is not a float, raise a `TypeError` with the exception `alpha must be a float`
        - if `alpha` is not positive, raise a `ValueError` with the exception `alpha must be positive`
    - All exceptions should be raised in the order listed above
    - Updates the private attributes `__W`, `__b`, and `__A`
    - You are allowed to use one loop
    - Returns the evaluation of the training data after `iterations` of training have occurred

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 6-main.py
#!/usr/bin/env python3

import matplotlib.pyplot as plt
import numpy as np

Neuron = __import__('6-neuron').Neuron

lib_train = np.load('../data/Binary_Train.npz')
X_train_3D, Y_train = lib_train['X'], lib_train['Y']
X_train = X_train_3D.reshape((X_train_3D.shape[0], -1)).T
lib_dev = np.load('../data/Binary_Dev.npz')
X_dev_3D, Y_dev = lib_dev['X'], lib_dev['Y']
X_dev = X_dev_3D.reshape((X_dev_3D.shape[0], -1)).T

np.random.seed(0)
neuron = Neuron(X_train.shape[0])
A, cost = neuron.train(X_train, Y_train, iterations=10)
accuracy = np.sum(A == Y_train) / Y_train.shape[1] * 100
print("Train cost:", cost)
print("Train accuracy: {}%".format(accuracy))
A, cost = neuron.evaluate(X_dev, Y_dev)
accuracy = np.sum(A == Y_dev) / Y_dev.shape[1] * 100
print("Dev cost:", cost)
print("Dev accuracy: {}%".format(accuracy))
fig = plt.figure(figsize=(10, 10))
for i in range(100):
    fig.add_subplot(10, 10, i + 1)
    plt.imshow(X_dev_3D[i])
    plt.title(A[0, i])
    plt.axis('off')
plt.tight_layout()
plt.show()
alexa@ubuntu-xenial:0x00-binary_classification$ ./6-main.py
Train cost: 1.3805076999077135
Train accuracy: 64.73746545598105%
Dev cost: 1.4096194345468178
Dev accuracy: 64.49172576832152%
```



Not that great… Let’s get more data!

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `6-neuron.py`

Check your code? QA Review

7. Upgrade Train Neuron mandatory

Score: 0.00% (*Checks completed: 0.00%*)

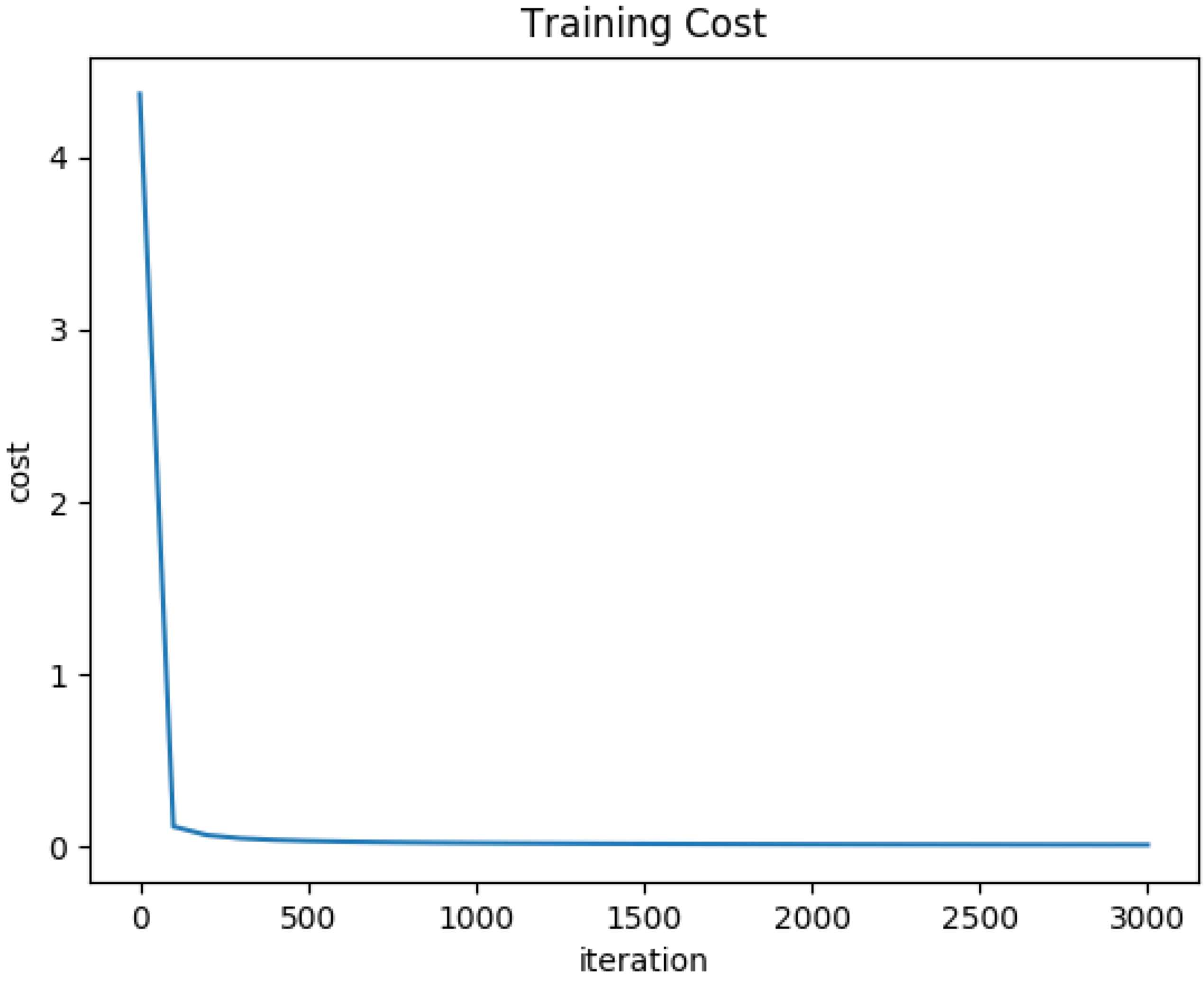
Write a class `Neuron` that defines a single neuron performing binary classification (Based on `6-neuron.py`):

☐ Done?

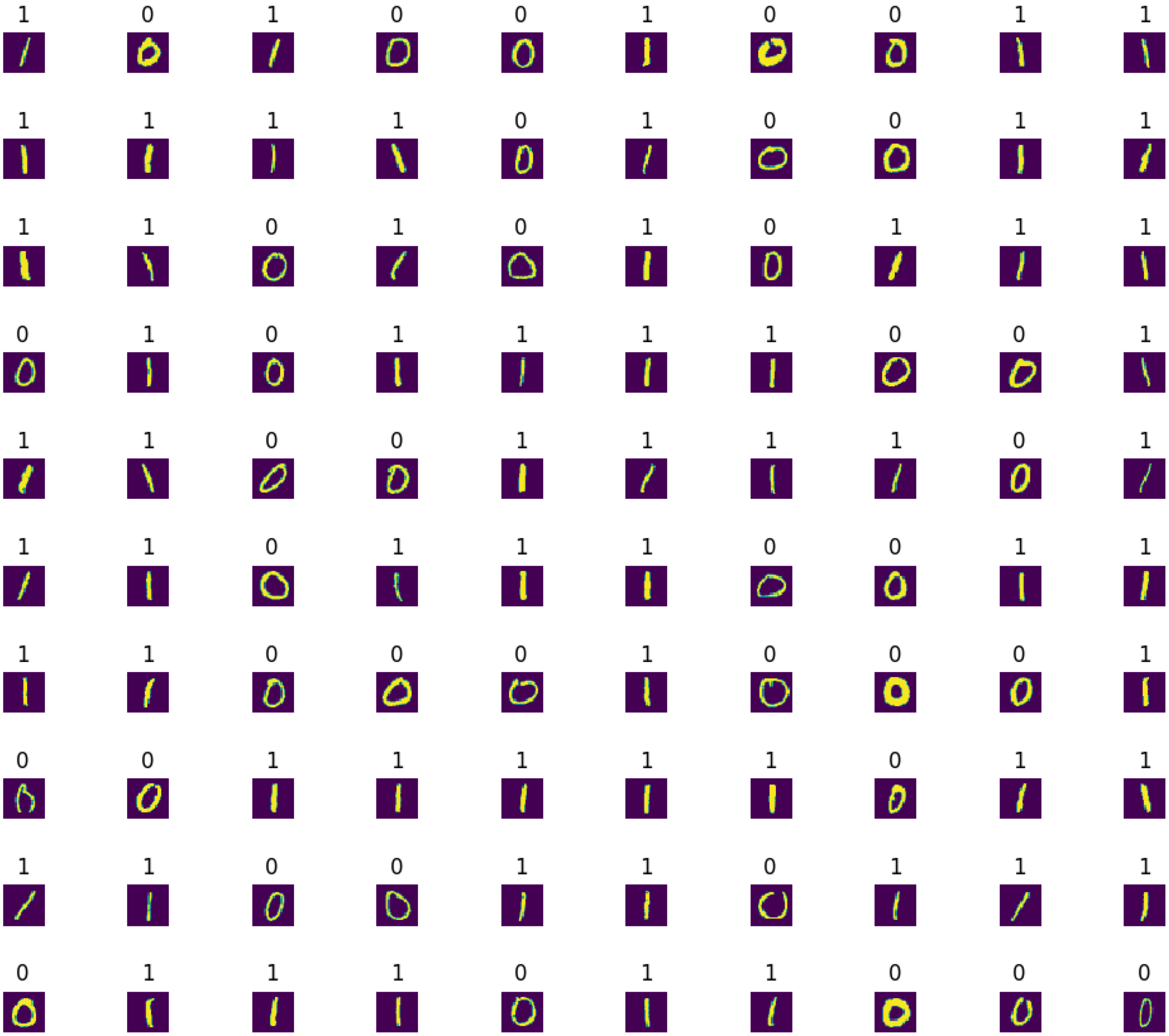
Help

- Update the public method `train` to `def train(self, X, Y, iterations=5000, alpha=0.05, verbose=True, graph=True, step=100):`
    - Trains the neuron by updating the private attributes `__W`, `__b`, and `__A`
    - `X` is a `numpy.ndarray` with shape (`nx`, `m`) that contains the input data
        - `nx` is the number of input features to the neuron
        - `m` is the number of examples
    - `Y` is a `numpy.ndarray` with shape (1, `m`) that contains the correct labels for the input data
    - `iterations` is the number of iterations to train over
        - if `iterations` is not an integer, raise a `TypeError` with the exception `iterations must be an integer`
        - if `iterations` is not positive, raise a `ValueError` with the exception `iterations must be a positive integer`
    - `alpha` is the learning rate
        - if `alpha` is not a float, raise a `TypeError` with the exception `alpha must be a float`
        - if `alpha` is not positive, raise a `ValueError` with the exception `alpha must be positive`
    - `verbose` is a boolean that defines whether or not to print information about the training. If `True`, print `Cost after {iteration} iterations: {cost}` every `step` iterations:
        - Include data from the 0th and last iteration
    - `graph` is a boolean that defines whether or not to graph information about the training once the training has completed. If `True`:
        - Plot the training data every `step` iterations as a blue line
        - Label the x-axis as `iteration`
        - Label the y-axis as `cost`
        - Title the plot `Training Cost`
        - Include data from the 0th and last iteration
    - Only if either `verbose` or `graph` are `True`:
        - if `step` is not an integer, raise a `TypeError` with the exception `step must be an integer`
        - if `step` is not positive or is greater than `iterations`, raise a `ValueError` with the exception `step must be positive and <= iterations`
    - All exceptions should be raised in the order listed above
    - The 0th iteration should represent the state of the neuron before any training has occurred
    - You are allowed to use one loop
    - You can use `import matplotlib.pyplot as plt`
    - Returns: the evaluation of the training data after `iterations` of training have occurred

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 7-main.py
#!/usr/bin/env python3

import matplotlib.pyplot as plt
import numpy as np

Neuron = __import__('7-neuron').Neuron

lib_train = np.load('../data/Binary_Train.npz')
X_train_3D, Y_train = lib_train['X'], lib_train['Y']
X_train = X_train_3D.reshape((X_train_3D.shape[0], -1)).T
lib_dev = np.load('../data/Binary_Dev.npz')
X_dev_3D, Y_dev = lib_dev['X'], lib_dev['Y']
X_dev = X_dev_3D.reshape((X_dev_3D.shape[0], -1)).T

np.random.seed(0)
neuron = Neuron(X_train.shape[0])
A, cost = neuron.train(X_train, Y_train, iterations=3000)
accuracy = np.sum(A == Y_train) / Y_train.shape[1] * 100
print("Train cost:", cost)
print("Train accuracy: {}%".format(accuracy))
A, cost = neuron.evaluate(X_dev, Y_dev)
accuracy = np.sum(A == Y_dev) / Y_dev.shape[1] * 100
print("Dev cost:", cost)
print("Dev accuracy: {}%".format(accuracy))
fig = plt.figure(figsize=(10, 10))
for i in range(100):
    fig.add_subplot(10, 10, i + 1)
    plt.imshow(X_dev_3D[i])
    plt.title(A[0, i])
    plt.axis('off')
plt.tight_layout()
plt.show()
alexa@ubuntu-xenial:0x00-binary_classification$ ./7-main.py
Cost after 0 iterations: 4.365104944262272
Cost after 100 iterations: 0.11955134491351888

...

Cost after 3000 iterations: 0.013386353289868338
```



```
Train cost: 0.013386353289868338
Train accuracy: 99.66837741808132%
Dev cost: 0.010803484515167197
Dev accuracy: 99.81087470449172%
```



Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `7-neuron.py`

QA Review

Ask a new correction

8. NeuralNetwork

mandatory

Score: 100.00% (*Checks completed: 100.00%*)

☑ Done!

Help

Write a class `NeuralNetwork` that defines a neural network with one hidden layer performing binary classification:

- class constructor: `def __init__(self, nx, nodes):`
    - `nx` is the number of input features
        - If `nx` is not an integer, raise a `TypeError` with the exception: `nx must be an integer`
        - If `nx` is less than 1, raise a `ValueError` with the exception: `nx must be a positive integer`
    - `nodes` is the number of nodes found in the hidden layer
        - If `nodes` is not an integer, raise a `TypeError` with the exception: `nodes must be an integer`
        - If `nodes` is less than 1, raise a `ValueError` with the exception: `nodes must be a positive integer`
    - All exceptions should be raised in the order listed above
- Public instance attributes:
    - `W1`: The weights vector for the hidden layer. Upon instantiation, it should be initialized using a random normal distribution.
    - `b1`: The bias for the hidden layer. Upon instantiation, it should be initialized with 0's.
    - `A1`: The activated output for the hidden layer. Upon instantiation, it should be initialized to 0.
    - `W2`: The weights vector for the output neuron. Upon instantiation, it should be initialized using a random normal distribution.
    - `b2`: The bias for the output neuron. Upon instantiation, it should be initialized to 0.
    - `A2`: The activated output for the output neuron (prediction). Upon instantiation, it should be initialized to 0.

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 8-main.py
#!/usr/bin/env python3

import numpy as np

NN = __import__('8-neural_network').NeuralNetwork

lib_train = np.load('../data/Binary_Train.npz')
X_3D, Y = lib_train['X'], lib_train['Y']
X = X_3D.reshape((X_3D.shape[0], -1)).T

np.random.seed(0)
nn = NN(X.shape[0], 3)
print(nn.W1)
print(nn.W1.shape)
print(nn.b1)
print(nn.W2)
print(nn.W2.shape)
print(nn.b2)
print(nn.A1)
print(nn.A2)
nn.A1 = 10
print(nn.A1)
alexa@ubuntu-xenial:0x00-binary_classification$ ./8-main.py
[[ 1.76405235  0.40015721  0.97873798 ...  0.52130375  0.61192719
  -1.34149673]
 [ 0.47689837  0.14844958  0.52904524 ...  0.0960042  -0.0451133
   0.07912172]
 [ 0.85053068 -0.83912419 -1.01177408 ... -0.07223876  0.31112445
  -1.07836109]]
(3, 784)
[[0.]
 [0.]
 [0.]]
[[ 1.06160017 -1.18488744 -1.80525169]]
(1, 3)
0
0
0
10
alexa@ubuntu-xenial:0x00-binary_classification$
```

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `8-neural_network.py`

Check your code?

QA Review

9. Privatize NeuralNetwork

mandatory

Score: 100.00% (*Checks completed: 100.00%*)

☑ Done!

Help

Write a class `NeuralNetwork` that defines a neural network with one hidden layer performing binary classification (based on `8-neural_network.py`):

- class constructor: `def __init__(self, nx, nodes):`
  - `nx` is the number of input features
    - If `nx` is not an integer, raise a `TypeError` with the exception: `nx must be an integer`
    - If `nx` is less than 1, raise a `ValueError` with the exception: `nx must be a positive integer`
  - `nodes` is the number of nodes found in the hidden layer
    - If `nodes` is not an integer, raise a `TypeError` with the exception: `nodes must be an integer`
    - If `nodes` is less than 1, raise a `ValueError` with the exception: `nodes must be a positive integer`
  - All exceptions should be raised in the order listed above
- **Private** instance attributes:
  - `W1` : The weights vector for the hidden layer. Upon instantiation, it should be initialized using a random normal distribution.
  - `b1` : The bias for the hidden layer. Upon instantiation, it should be initialized with 0's.
  - `A1` : The activated output for the hidden layer. Upon instantiation, it should be initialized to 0.
  - `W2` : The weights vector for the output neuron. Upon instantiation, it should be initialized using a random normal distribution.
  - `b2` : The bias for the output neuron. Upon instantiation, it should be initialized to 0.
  - `A2` : The activated output for the output neuron (prediction). Upon instantiation, it should be initialized to 0.
  - Each private attribute should have a corresponding getter function (no setter function).

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 9-main.py
#!/usr/bin/env python3

import numpy as np

NN = __import__('9-neural_network').NeuralNetwork

lib_train = np.load('../data/Binary_Train.npz')
X_3D, Y = lib_train['X'], lib_train['Y']
X = X_3D.reshape((X_3D.shape[0], -1)).T

np.random.seed(0)
nn = NN(X.shape[0], 3)
print(nn.W1)
print(nn.b1)
print(nn.W2)
print(nn.b2)
print(nn.A1)
print(nn.A2)
nn.A1 = 10
print(nn.A1)
alexa@ubuntu-xenial:0x00-binary_classification$ ./9-main.py
[[ 1.76405235  0.40015721  0.97873798 ...  0.52130375  0.61192719
  -1.34149673]
 [ 0.47689837  0.14844958  0.52904524 ...  0.0960042  -0.0451133
   0.07912172]
 [ 0.85053068 -0.83912419 -1.01177408 ... -0.07223876  0.31112445
  -1.07836109]]
[[0.]
 [0.]
 [0.]]
[[ 1.06160017 -1.18488744 -1.80525169]]
0
0
0
Traceback (most recent call last):
  File "./9-main.py", line 19, in <module>
    nn.A1 = 10
AttributeError: can't set attribute
alexa@ubuntu-xenial:0x00-binary_classification$
```

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `9-neural_network.py`

Check your code? QA Review

10. NeuralNetwork Forward Propagation

mandatory

Score: 100.00% (*Checks completed: 100.00%*)

☑ Done!

Help

Write a class `NeuralNetwork` that defines a neural network with one hidden layer performing binary classification (based on `9-neural_network.py`):

- Add the public method `def forward_prop(self, X):`
    - Calculates the forward propagation of the neural network
    - `X` is a `numpy.ndarray` with shape (`nx`, `m`) that contains the input data
        - `nx` is the number of input features to the neuron
        - `m` is the number of examples
    - Updates the private attributes `__A1` and `__A2`
    - The neurons should use a sigmoid activation function
    - Returns the private attributes `__A1` and `__A2`, respectively

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 10-main.py
#!/usr/bin/env python3

import numpy as np

NN = __import__('10-neural_network').NeuralNetwork

lib_train = np.load('../data/Binary_Train.npz')
X_3D, Y = lib_train['X'], lib_train['Y']
X = X_3D.reshape((X_3D.shape[0], -1)).T

np.random.seed(0)
nn = NN(X.shape[0], 3)
nn._NeuralNetwork__b1 = np.ones((3, 1))
nn._NeuralNetwork__b2 = 1
A1, A2 = nn.forward_prop(X)
if A1 is nn.A1:
        print(A1)
if A2 is nn.A2:
        print(A2)
alexa@ubuntu-xenial:0x00-binary_classification$ ./10-main.py
[[5.34775247e-10 7.24627778e-04 4.52416436e-07 ... 8.75691930e-05
  1.13141966e-06 6.55799932e-01]
 [9.99652394e-01 9.99999995e-01 6.77919152e-01 ... 1.00000000e+00
  9.99662771e-01 9.99990554e-01]
 [5.57969669e-01 2.51645047e-02 4.04250047e-04 ... 1.57024117e-01
  9.97325173e-01 7.41310459e-02]]
[[0.23294587 0.44286405 0.54884691 ... 0.38502756 0.12079644 0.593269  ]]
alexa@ubuntu-xenial:0x00-binary_classification$
```

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`

- File: `10-neural_network.py`

Check your code? QA Review

11. NeuralNetwork Cost mandatory

Score: 100.00% (*Checks completed: 100.00%*)

☑ Done!

Help

Write a class `NeuralNetwork` that defines a neural network with one hidden layer performing binary classification (based on `10-neural_network.py`):

- Add the public method `def cost(self, Y, A):`
  - Calculates the cost of the model using logistic regression
  - `Y` is a `numpy.ndarray` with shape (1, `m`) that contains the correct labels for the input data
  - `A` is a `numpy.ndarray` with shape (1, `m`) containing the activated output of the neuron for each example
  - To avoid division by zero errors, please use `1.0000001 - A` instead of `1 - A`
  - Returns the cost

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 11-main.py
#!/usr/bin/env python3

import numpy as np

NN = __import__('11-neural_network').NeuralNetwork

lib_train = np.load('../data/Binary_Train.npz')
X_3D, Y = lib_train['X'], lib_train['Y']
X = X_3D.reshape((X_3D.shape[0], -1)).T

np.random.seed(0)
nn = NN(X.shape[0], 3)
_, A = nn.forward_prop(X)
cost = nn.cost(Y, A)
print(cost)
alexa@ubuntu-xenial:0x00-binary_classification$ ./11-main.py
0.7917984405648548
alexa@ubuntu-xenial:0x00-binary_classification$
```

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `11-neural_network.py`

Check your code? QA Review

12. Evaluate NeuralNetwork mandatory

☑ Done!

Help

Score: 100.00% (*Checks completed: 100.00%*)

Write a class `NeuralNetwork` that defines a neural network with one hidden layer performing binary classification (based on `11-neural_network.py`):

- Add the public method `def evaluate(self, X, Y):`
  - Evaluates the neural network's predictions
  - `X` is a `numpy.ndarray` with shape (`nx` , `m`) that contains the input data
    - `nx` is the number of input features to the neuron
    - `m` is the number of examples
  - `Y` is a `numpy.ndarray` with shape (1, `m`) that contains the correct labels for the input data
  - Returns the neuron's prediction and the cost of the network, respectively
    - The prediction should be a `numpy.ndarray` with shape (1, `m`) containing the predicted labels for each example
    - The label values should be 1 if the output of the network is >= 0.5 and 0 otherwise

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 12-main.py
#!/usr/bin/env python3

import numpy as np

NN = __import__('12-neural_network').NeuralNetwork

lib_train = np.load('../data/Binary_Train.npz')
X_3D, Y = lib_train['X'], lib_train['Y']
X = X_3D.reshape((X_3D.shape[0], -1)).T

np.random.seed(0)
nn = NN(X.shape[0], 3)
A, cost = nn.evaluate(X, Y)
print(A)
print(cost)
alexa@ubuntu-xenial:0x00-binary_classification$ ./12-main.py
[[0 0 0 ... 0 0 0]]
0.7917984405648548
alexa@ubuntu-xenial:0x00-binary_classification$
```

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `12-neural_network.py`

Check your code? QA Review

13. NeuralNetwork Gradient Descent mandatory

☑ Done!

Help

Score: 100.00% (*Checks completed: 100.00%*)

Write a class `NeuralNetwork` that defines a neural network with one hidden layer performing binary classification (based on `12-neural_network.py`):

- Add the public method `def gradient_descent(self, X, Y, A1, A2, alpha=0.05):`
  - Calculates one pass of gradient descent on the neural network
  - `X` is a `numpy.ndarray` with shape (`nx`, `m`) that contains the input data
    - `nx` is the number of input features to the neuron
    - `m` is the number of examples
  - `Y` is a `numpy.ndarray` with shape (1, `m`) that contains the correct labels for the input data
  - `A1` is the output of the hidden layer
  - `A2` is the predicted output
  - `alpha` is the learning rate
  - Updates the private attributes `__W1`, `__b1`, `__W2`, and `__b2`

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 13-main.py
#!/usr/bin/env python3

import numpy as np

NN = __import__('13-neural_network').NeuralNetwork

lib_train = np.load('../data/Binary_Train.npz')
X_3D, Y = lib_train['X'], lib_train['Y']
X = X_3D.reshape((X_3D.shape[0], -1)).T

np.random.seed(0)
nn = NN(X.shape[0], 3)
A1, A2 = nn.forward_prop(X)
nn.gradient_descent(X, Y, A1, A2, 0.5)
print(nn.W1)
print(nn.b1)
print(nn.W2)
print(nn.b2)
alexa@ubuntu-xenial:0x00-binary_classification$ ./13-main.py
[[ 1.76405235  0.40015721  0.97873798 ...  0.52130375  0.61192719
  -1.34149673]
 [ 0.47689837  0.14844958  0.52904524 ...  0.0960042  -0.0451133
   0.07912172]
 [ 0.85053068 -0.83912419 -1.01177408 ... -0.07223876  0.31112445
  -1.07836109]]
[[ 0.003193  ]
 [-0.01080922]
 [-0.01045412]]
[[ 1.06583858 -1.06149724 -1.79864091]]
[[0.15552509]]
alexa@ubuntu-xenial:0x00-binary_classification$
```

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `13-neural_network.py`

Check your code? QA Review

14. Train NeuralNetwork mandatory

Done!

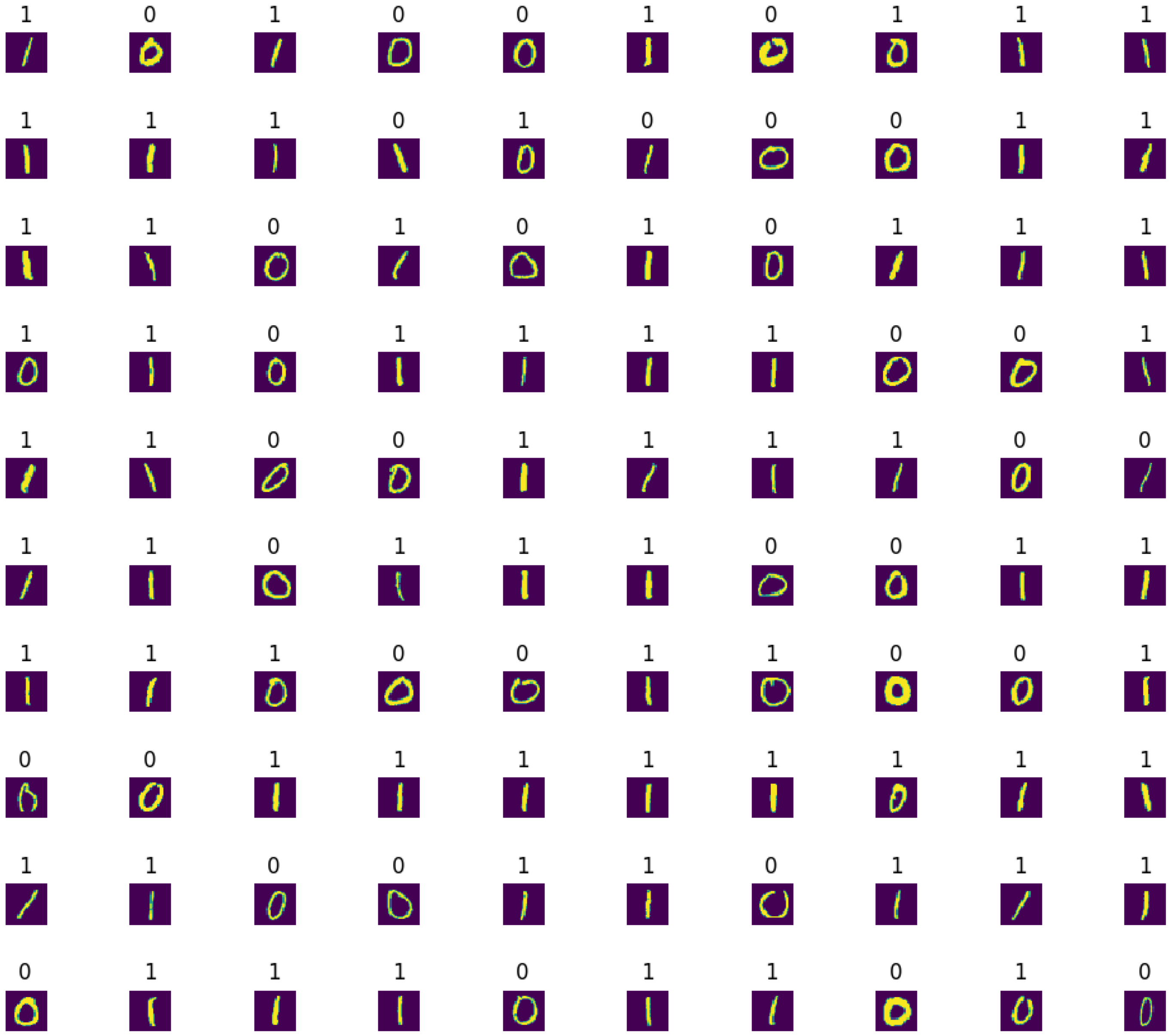
Help

Score: 100.00% (*Checks completed: 100.00%*)

Write a class `NeuralNetwork` that defines a neural network with one hidden layer performing binary classification (based on `13-neural_network.py`):

- Add the public method `def train(self, X, Y, iterations=5000, alpha=0.05):`
    - Trains the neural network
    - `X` is a `numpy.ndarray` with shape (`nx` , `m`) that contains the input data
        - `nx` is the number of input features to the neuron
        - `m` is the number of examples
    - `Y` is a `numpy.ndarray` with shape (1, `m`) that contains the correct labels for the input data
    - `iterations` is the number of iterations to train over
        - if `iterations` is not an integer, raise a `TypeError` with the exception `iterations must be an integer`
        - if `iterations` is not positive, raise a `ValueError` with the exception `iterations must be a positive integer`
    - `alpha` is the learning rate
        - if `alpha` is not a float, raise a `TypeError` with the exception `alpha must be a float`
        - if `alpha` is not positive, raise a `ValueError` with the exception `alpha must be positive`
    - All exceptions should be raised in the order listed above
    - Updates the private attributes `__W1` , `__b1` , `__A1` , `__W2` , `__b2` , and `__A2`
    - You are allowed to use one loop
    - Returns the evaluation of the training data after `iterations` of training have occurred

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 14-main.py
#!/usr/bin/env python3

import matplotlib.pyplot as plt
import numpy as np

NN = __import__('14-neural_network').NeuralNetwork

lib_train = np.load('../data/Binary_Train.npz')
X_train_3D, Y_train = lib_train['X'], lib_train['Y']
X_train = X_train_3D.reshape((X_train_3D.shape[0], -1)).T
lib_dev = np.load('../data/Binary_Dev.npz')
X_dev_3D, Y_dev = lib_dev['X'], lib_dev['Y']
X_dev = X_dev_3D.reshape((X_dev_3D.shape[0], -1)).T

np.random.seed(0)
nn = NN(X_train.shape[0], 3)
A, cost = nn.train(X_train, Y_train, iterations=100)
accuracy = np.sum(A == Y_train) / Y_train.shape[1] * 100
print("Train cost:", cost)
print("Train accuracy: {}%".format(accuracy))
A, cost = nn.evaluate(X_dev, Y_dev)
accuracy = np.sum(A == Y_dev) / Y_dev.shape[1] * 100
print("Dev cost:", cost)
print("Dev accuracy: {}%".format(accuracy))
fig = plt.figure(figsize=(10, 10))
for i in range(100):
    fig.add_subplot(10, 10, i + 1)
    plt.imshow(X_dev_3D[i])
    plt.title(A[0, i])
    plt.axis('off')
plt.tight_layout()
plt.show()
alexa@ubuntu-xenial:0x00-binary_classification$ ./14-main.py
Train cost: 0.4680930945144984
Train accuracy: 84.69009080142123%
Dev cost: 0.45985938789496067
Dev accuracy: 86.52482269503547%
alexa@ubuntu-xenial:0x00-binary_classification$
```



Pretty good… but there are still some incorrect labels. We need more data to see why…

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `14-neural_network.py`

Check your code? QA Review

15. Upgrade Train NeuralNetwork mandatory

Score: 60.71% (*Checks completed: 50.00%*)

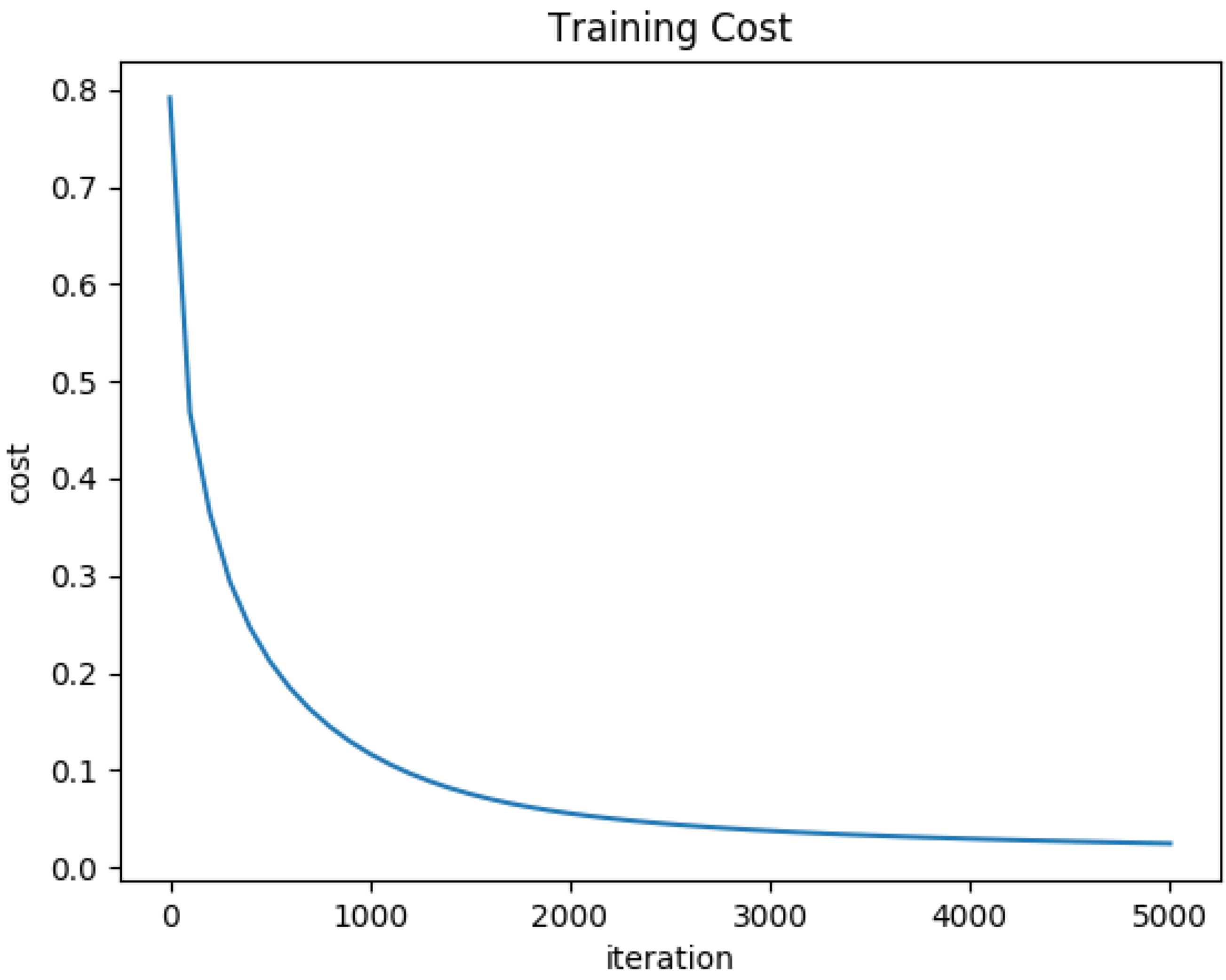
□ Done?

Help

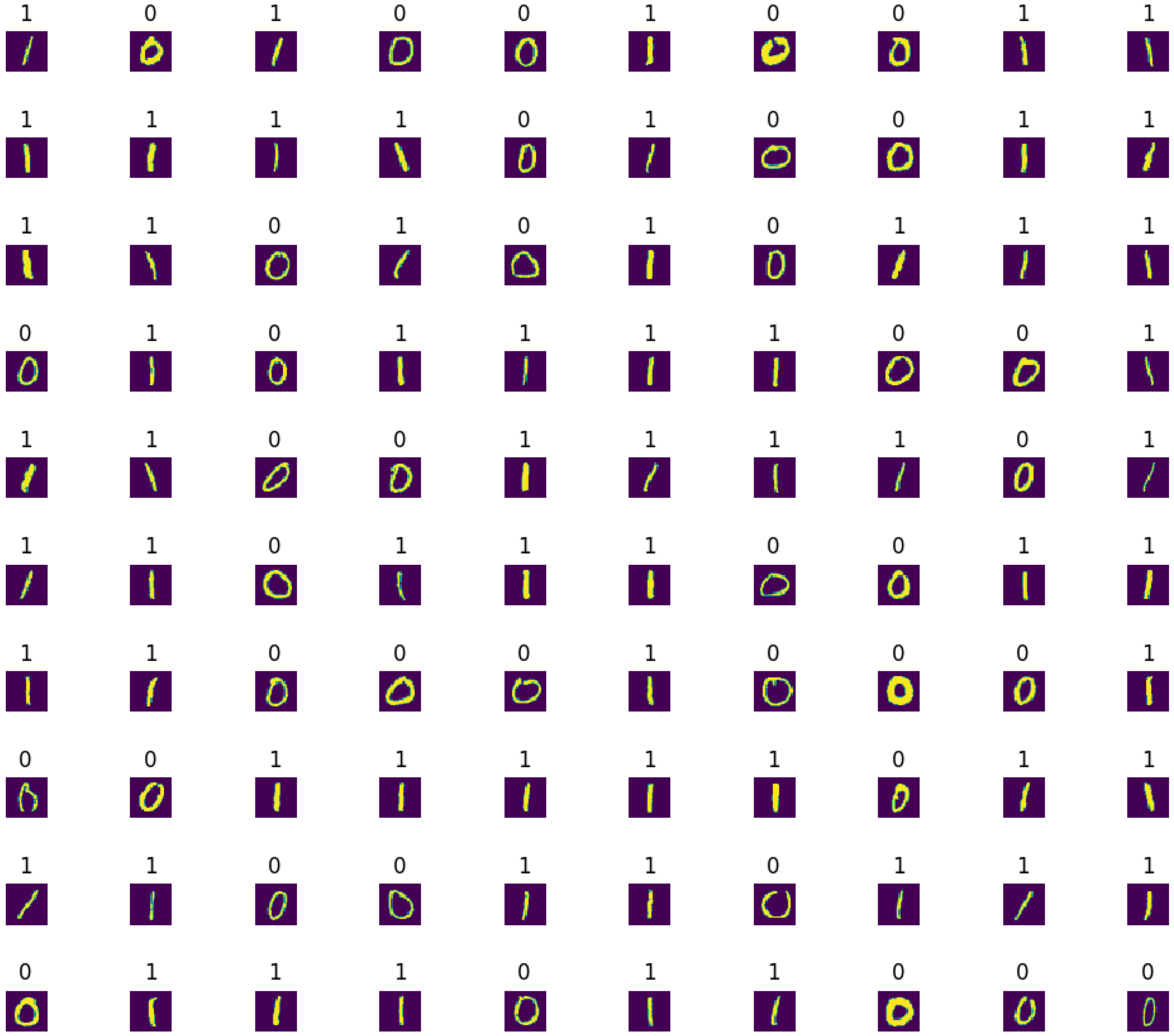
Write a class `NeuralNetwork` that defines a neural network with one hidden layer performing binary classification (based on `14-neural_network.py`):

- Update the public method `train` to `def train(self, X, Y, iterations=5000, alpha=0.05, verbose=True, graph=True, step=100)`:
    - Trains the neural network
    - `X` is a `numpy.ndarray` with shape (`nx`, `m`) that contains the input data
        - `nx` is the number of input features to the neuron
        - `m` is the number of examples
    - `Y` is a `numpy.ndarray` with shape (1, `m`) that contains the correct labels for the input data
    - `iterations` is the number of iterations to train over
        - if `iterations` is not an integer, raise a `TypeError` with the exception `iterations must be an integer`
        - if `iterations` is not positive, raise a `ValueError` with the exception `iterations must be a positive integer`
    - `alpha` is the learning rate
        - if `alpha` is not a float, raise a `TypeError` with the exception `alpha must be a float`
        - if `alpha` is not positive, raise a `ValueError` with the exception `alpha must be positive`
    - Updates the private attributes `__W1`, `__b1`, `__A1`, `__W2`, `__b2`, and `__A2`
    - `verbose` is a boolean that defines whether or not to print information about the training. If `True`, print `Cost after {iteration} iterations: {cost}` every `step` iterations:
        - Include data from the 0th and last iteration
    - `graph` is a boolean that defines whether or not to graph information about the training once the training has completed. If `True`:
        - Plot the training data every `step` iterations as a blue line
        - Label the x-axis as `iteration`
        - Label the y-axis as `cost`
        - Title the plot `Training Cost`
        - Include data from the 0th and last iteration
    - Only if either `verbose` or `graph` are `True`:
        - if `step` is not an integer, raise a `TypeError` with the exception `step must be an integer`
        - if `step` is not positive and less than or equal to `iterations`, raise a `ValueError` with the exception `step must be positive and <= iterations`
    - All exceptions should be raised in the order listed above
    - The 0th iteration should represent the state of the neuron before any training has occurred
    - You are allowed to use one loop
    - Returns the evaluation of the training data after `iterations` of training have occurred

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 15-main.py
#!/usr/bin/env python3

import matplotlib.pyplot as plt
import numpy as np

NN = __import__('15-neural_network').NeuralNetwork

lib_train = np.load('../data/Binary_Train.npz')
X_train_3D, Y_train = lib_train['X'], lib_train['Y']
X_train = X_train_3D.reshape((X_train_3D.shape[0], -1)).T
lib_dev = np.load('../data/Binary_Dev.npz')
X_dev_3D, Y_dev = lib_dev['X'], lib_dev['Y']
X_dev = X_dev_3D.reshape((X_dev_3D.shape[0], -1)).T

np.random.seed(0)
nn = NN(X_train.shape[0], 3)
A, cost = nn.train(X_train, Y_train)
accuracy = np.sum(A == Y_train) / Y_train.shape[1] * 100
print("Train cost:", cost)
print("Train accuracy: {}%".format(accuracy))
A, cost = nn.evaluate(X_dev, Y_dev)
accuracy = np.sum(A == Y_dev) / Y_dev.shape[1] * 100
print("Dev cost:", cost)
print("Dev accuracy: {}%".format(accuracy))
fig = plt.figure(figsize=(10, 10))
for i in range(100):
    fig.add_subplot(10, 10, i + 1)
    plt.imshow(X_dev_3D[i])
    plt.title(A[0, i])
    plt.axis('off')
plt.tight_layout()
plt.show()
alexa@ubuntu-xenial:0x00-binary_classification$ ./15-main.py
Cost after 0 iterations: 0.7917984405648547
Cost after 100 iterations: 0.4680930945144984

...

Cost after 5000 iterations: 0.024369225667283875
```



```
Train cost: 0.024369225667283875
Train accuracy: 99.3999210422424%
Dev cost: 0.020330639788072768
Dev accuracy: 99.57446808510639%
```



Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `15-neural_network.py`

QA Review   Ask a new correction

16. DeepNeuralNetwork

mandatory

Score: 100.00% (*Checks completed: 100.00%*)

☑ Done!

Help

Write a class `DeepNeuralNetwork` that defines a deep neural network performing binary classification:

- class constructor: `def __init__(self, nx, layers):`
    - `nx` is the number of input features
        - If `nx` is not an integer, raise a `TypeError` with the exception: `nx must be an integer`
        - If `nx` is less than 1, raise a `ValueError` with the exception: `nx must be a positive integer`
    - `layers` is a list representing the number of nodes in each layer of the network
        - If `layers` is not a list, raise a `TypeError` with the exception: `layers must be a list of positive integers`
        - The first value in `layers` represents the number of nodes in the first layer, …
        - If the elements in `layers` are not all positive integers, raise a TypeError with the exception `layers must be a list of positive integers`
    - All exceptions should be raised in the order listed above
    - Sets the public instance attributes:
        - `L`: The number of layers in the neural network.
        - `cache`: A dictionary to hold all intermediary values of the network. Upon instantiation, it should be set to an empty dictionary.
        - `weights`: A dictionary to hold all weights and biased of the network. Upon instantiation:
            - The weights of the network should be initialized using the `He et al.` method and saved in the `weights` dictionary using the key `W{l}` where `{l}` is the hidden layer the weight belongs to
            - The biases of the network should be initialized to 0's and saved in the `weights` dictionary using the key `b{l}` where `{l}` is the hidden layer the bias belongs to
    - You are allowed to use one loop

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 16-main.py
#!/usr/bin/env python3

import numpy as np

Deep = __import__('16-deep_neural_network').DeepNeuralNetwork

lib_train = np.load('../data/Binary_Train.npz')
X_3D, Y = lib_train['X'], lib_train['Y']
X = X_3D.reshape((X_3D.shape[0], -1)).T

np.random.seed(0)
deep = Deep(X.shape[0], [5, 3, 1])
print(deep.cache)
print(deep.weights)
print(deep.L)
deep.L = 10
print(deep.L)
alexa@ubuntu-xenial:0x00-binary_classification$ ./16-main.py
{}
{'b3': array([[0.]]), 'W2': array([[ 0.4609219 ,  0.56004008, -1.2250799 , -0.09454199,
0.57799141],
       [-0.16310703,  0.06882082, -0.94578088, -0.30359994,  1.15661914],
       [-0.49841799, -0.9111359 ,  0.09453424,  0.49877298,  0.75503205]]), 'W3': array
([[-0.42271877,  0.18165055,  0.4444639 ]]), 'b2': array([[0.],
       [0.],
       [0.]]), 'W1': array([[ 0.0890981 ,  0.02021099,  0.04943373, ...,  0.02632982,
         0.03090699, -0.06775582],
       [ 0.02408701,  0.00749784,  0.02672082, ...,  0.00484894,
        -0.00227857,  0.00399625],
       [ 0.04295829, -0.04238217, -0.05110231, ..., -0.00364861,
         0.01571416, -0.05446546],
       [ 0.05361891, -0.05984585, -0.09117898, ..., -0.03094292,
        -0.01925805, -0.06308145],
       [-0.01667953, -0.04216413,  0.06239623, ..., -0.02024521,
        -0.05159656, -0.02373981]]), 'b1': array([[0.],
       [0.],
       [0.],
       [0.],
       [0.]])}
3
10
alexa@ubuntu-xenial:0x00-binary_classification$
```

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `16-deep_neural_network.py`

Check your code? QA Review

17. Privatize DeepNeuralNetwork mandatory

Score: 100.00% (*Checks completed: 100.00%*)

☑ Done!

Help

Write a class `DeepNeuralNetwork` that defines a deep neural network performing binary classification (based on `16-deep_neural_network.py`):

- class constructor: `def __init__(self, nx, layers):`
    - `nx` is the number of input features
        - If `nx` is not an integer, raise a `TypeError` with the exception: `nx must be an integer`
        - If `nx` is less than 1, raise a `ValueError` with the exception: `nx must be a positive integer`
    - `layers` is a list representing the number of nodes in each layer of the network
        - If `layers` is not a list, raise a `TypeError` with the exception: `layers must be a list of positive integers`
        - The first value in `layers` represents the number of nodes in the first layer, …
        - If the elements in `layers` are not all positive integers, raise a TypeError with the exception `layers must be a list of positive integers`
    - All exceptions should be raised in the order listed above
    - Sets the **private** instance attributes:
        - `__L` : The number of layers in the neural network.
        - `__cache` : A dictionary to hold all intermediary values of the network. Upon instantiation, it should be set to an empty dictionary.
        - `__weights` : A dictionary to hold all weights and biased of the network. Upon instantiation:
            - The weights of the network should be initialized using the `He et al.` method and saved in the `__weights` dictionary using the key `W{l}` where `{l}` is the hidden layer the weight belongs to
            - The biases of the network should be initialized to `0` 's and saved in the `__weights` dictionary using the key `b{l}` where `{l}` is the hidden layer the bias belongs to
        - Each private attribute should have a corresponding getter function (no setter function).
    - You are allowed to use one loop

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 17-main.py
#!/usr/bin/env python3

import numpy as np

Deep = __import__('17-deep_neural_network').DeepNeuralNetwork

lib_train = np.load('../data/Binary_Train.npz')
X_3D, Y = lib_train['X'], lib_train['Y']
X = X_3D.reshape((X_3D.shape[0], -1)).T

np.random.seed(0)
deep = Deep(X.shape[0], [5, 3, 1])
print(deep.cache)
print(deep.weights)
print(deep.L)
deep.L = 10
print(deep.L)
alexa@ubuntu-xenial:0x00-binary_classification$ ./17-main.py
{}
{'b1': array([[0.],
       [0.],
       [0.],
       [0.],
       [0.]]), 'b2': array([[0.],
       [0.],
       [0.]]), 'W2': array([[ 0.4609219 ,  0.56004008, -1.2250799 , -0.09454199,  0.577
99141],
       [-0.16310703,  0.06882082, -0.94578088, -0.30359994,  1.15661914],
       [-0.49841799, -0.9111359 ,  0.09453424,  0.49877298,  0.75503205]]), 'W1': array
([[ 0.0890981 ,  0.02021099,  0.04943373, ...,  0.02632982,
         0.03090699, -0.06775582],
       [ 0.02408701,  0.00749784,  0.02672082, ...,  0.00484894,
        -0.00227857,  0.00399625],
       [ 0.04295829, -0.04238217, -0.05110231, ..., -0.00364861,
         0.01571416, -0.05446546],
       [ 0.05361891, -0.05984585, -0.09117898, ..., -0.03094292,
        -0.01925805, -0.06308145],
       [-0.01667953, -0.04216413,  0.06239623, ..., -0.02024521,
        -0.05159656, -0.02373981]]), 'b3': array([[0.]]), 'W3': array([[-0.42271877,
0.18165055,  0.4444639 ]])}
3
Traceback (most recent call last):
  File "./17-main.py", line 16, in <module>
    deep.L = 10
AttributeError: can't set attribute
alexa@ubuntu-xenial:0x00-binary_classification$
```

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `17-deep_neural_network.py`

Check your code? QA Review

18. DeepNeuralNetwork Forward Propagation

mandatory

☑ Done!

Help

Score: 100.00% (*Checks completed: 100.00%*)

Write a class `DeepNeuralNetwork` that defines a deep neural network performing binary classification (based on `17-deep_neural_network.py`):

- Add the public method `def forward_prop(self, X):`
  - Calculates the forward propagation of the neural network
  - `X` is a `numpy.ndarray` with shape (`nx` , `m`) that contains the input data
    - `nx` is the number of input features to the neuron
    - `m` is the number of examples
  - Updates the private attribute `__cache` :
    - The activated outputs of each layer should be saved in the `__cache` dictionary using the key `A{l}` where `{l}` is the hidden layer the activated output belongs to
    - `X` should be saved to the `cache` dictionary using the key `A0`
  - All neurons should use a sigmoid activation function
  - You are allowed to use one loop
  - Returns the output of the neural network and the cache, respectively

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 18-main.py
#!/usr/bin/env python3

import numpy as np

Deep = __import__('18-deep_neural_network').DeepNeuralNetwork

lib_train = np.load('../data/Binary_Train.npz')
X_3D, Y = lib_train['X'], lib_train['Y']
X = X_3D.reshape((X_3D.shape[0], -1)).T

np.random.seed(0)
deep = Deep(X.shape[0], [5, 3, 1])
deep._DeepNeuralNetwork__weights['b1'] = np.ones((5, 1))
deep._DeepNeuralNetwork__weights['b2'] = np.ones((3, 1))
deep._DeepNeuralNetwork__weights['b3'] = np.ones((1, 1))
A, cache = deep.forward_prop(X)
print(A)
print(cache)
print(cache is deep.cache)
print(A is cache['A3'])
alexa@ubuntu-xenial:0x00-binary_classification$ ./18-main.py
[[0.75603476 0.7516025  0.75526716 ... 0.75228888 0.75522853 0.75217069]]
{'A1': array([[0.4678435 , 0.64207147, 0.55271425, ..., 0.61718097, 0.56412986,
        0.72751504],
       [0.79441392, 0.87140579, 0.72851107, ..., 0.8898201 , 0.79466389,
        0.82257068],
       [0.72337339, 0.68239373, 0.63526533, ..., 0.7036234 , 0.7770501 ,
        0.69465346],
       [0.65305735, 0.69829955, 0.58646313, ..., 0.73949722, 0.52054315,
        0.73151973],
       [0.67408798, 0.69624537, 0.73084352, ..., 0.70663173, 0.76204175,
        0.72705428]]), 'A3': array([[0.75603476, 0.7516025 , 0.75526716, ..., 0.7522888
8, 0.75522853,
        0.75217069]]), 'A0': array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32), 'A2': array([[0.75067742, 0.7831
9533, 0.77755571, ..., 0.77891002, 0.75847839,
        0.78517215],
       [0.70591081, 0.71159364, 0.7362214 , ..., 0.70845465, 0.72133875,
        0.71090691],
       [0.72032379, 0.69519095, 0.72414599, ..., 0.70067751, 0.71161433,
        0.70420437]])}
True
True
alexa@ubuntu-xenial:0x00-binary_classification$
```

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `18-deep_neural_network.py`

Check your code? QA Review

19. DeepNeuralNetwork Cost

mandatory

☑ Done!

Help

Score: 100.00% (*Checks completed: 100.00%*)

Write a class `DeepNeuralNetwork` that defines a deep neural network performing binary classification (based on `18-deep_neural_network.py`):

- Add the public method `def cost(self, Y, A):`
  - Calculates the cost of the model using logistic regression
  - `Y` is a `numpy.ndarray` with shape (1, `m`) that contains the correct labels for the input data
  - `A` is a `numpy.ndarray` with shape (1, `m`) containing the activated output of the neuron for each example
  - To avoid division by zero errors, please use `1.0000001 - A` instead of `1 - A`
  - Returns the cost

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 19-main.py
#!/usr/bin/env python3

import numpy as np

Deep = __import__('19-deep_neural_network').DeepNeuralNetwork

lib_train = np.load('../data/Binary_Train.npz')
X_3D, Y = lib_train['X'], lib_train['Y']
X = X_3D.reshape((X_3D.shape[0], -1)).T

np.random.seed(0)
deep = Deep(X.shape[0], [5, 3, 1])
A, _ = deep.forward_prop(X)
cost = deep.cost(Y, A)
print(cost)
alexa@ubuntu-xenial:0x00-binary_classification$ ./19-main.py
0.6958649419170609
alexa@ubuntu-xenial:0x00-binary_classification$
```

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `19-deep_neural_network.py`

Check your code? QA Review

20. Evaluate DeepNeuralNetwork mandatory

☑ Done!

Help

Score: 100.00% (*Checks completed: 100.00%*)

Write a class `DeepNeuralNetwork` that defines a deep neural network performing binary classification (based on `19-deep_neural_network.py`):

- Add the public method `def evaluate(self, X, Y):`
  - Evaluates the neural network's predictions
  - `X` is a `numpy.ndarray` with shape (`nx` , `m`) that contains the input data
    - `nx` is the number of input features to the neuron
    - `m` is the number of examples
  - `Y` is a `numpy.ndarray` with shape (1, `m`) that contains the correct labels for the input data
  - Returns the neuron's prediction and the cost of the network, respectively
    - The prediction should be a `numpy.ndarray` with shape (1, `m`) containing the predicted labels for each example
    - The label values should be 1 if the output of the network is >= 0.5 and 0 otherwise

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 20-main.py
#!/usr/bin/env python3

import numpy as np

Deep = __import__('20-deep_neural_network').DeepNeuralNetwork

lib_train = np.load('../data/Binary_Train.npz')
X_3D, Y = lib_train['X'], lib_train['Y']
X = X_3D.reshape((X_3D.shape[0], -1)).T

np.random.seed(0)
deep = Deep(X.shape[0], [5, 3, 1])
A, cost = deep.evaluate(X, Y)
print(A)
print(cost)
alexa@ubuntu-xenial:0x00-binary_classification$ ./20-main.py
[[1 1 1 ... 1 1 1]]
0.6958649419170609
alexa@ubuntu-xenial:0x00-binary_classification$
```

Repo:

- GitHub repository: `holbertonschool-machine_learning`

- Directory: `supervised_learning/0x00-binary_classification`
- File: `20-deep_neural_network.py`

Check your code? QA Review

21. DeepNeuralNetwork Gradient Descent

mandatory

Score: 0.00% (*Checks completed: 0.00%*)

☐ Done?

Help

Write a class `DeepNeuralNetwork` that defines a deep neural network performing binary classification (based on `20-deep_neural_network.py`):

- Add the public method `def gradient_descent(self, Y, cache, alpha=0.05):`
  - Calculates one pass of gradient descent on the neural network
  - `Y` is a `numpy.ndarray` with shape (1, `m`) that contains the correct labels for the input data
  - `cache` is a dictionary containing all the intermediary values of the network
  - `alpha` is the learning rate
  - Updates the private attribute `__weights`
  - You are allowed to use one loop

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 21-main.py
#!/usr/bin/env python3

import numpy as np

Deep = __import__('21-deep_neural_network').DeepNeuralNetwork

lib_train = np.load('../data/Binary_Train.npz')
X_3D, Y = lib_train['X'], lib_train['Y']
X = X_3D.reshape((X_3D.shape[0], -1)).T

np.random.seed(0)
deep = Deep(X.shape[0], [5, 3, 1])
A, cache = deep.forward_prop(X)
deep.gradient_descent(Y, cache, 0.5)
print(deep.weights)
alexa@ubuntu-xenial:0x00-binary_classification$ ./21-main.py
{'b3': array([[0.00659936]]), 'b2': array([[-0.00055419],
       [ 0.00032369],
       [ 0.0007201 ]]), 'W2': array([[ 0.4586347 ,  0.55968571, -1.22435332, -0.0951687
4,  0.57668454],
       [-0.16209305,  0.06902405, -0.9460547 , -0.30329296,  1.15722071],
       [-0.49595566, -0.91068385,  0.09382566,  0.49948968,  0.75647764]]), 'b1': array
([[-1.01835520e-03],
       [-1.22929756e-04],
       [ 9.25521878e-05],
       [ 1.07730873e-04],
       [ 2.29014796e-04]]), 'W3': array([[-0.41262664,  0.18889024,  0.44717929]]), 'W
1': array([[ 0.0890981 ,  0.02021099,  0.04943373, ...,  0.02632982,
         0.03090699, -0.06775582],
       [ 0.02408701,  0.00749784,  0.02672082, ...,  0.00484894,
        -0.00227857,  0.00399625],
       [ 0.04295829, -0.04238217, -0.05110231, ..., -0.00364861,
         0.01571416, -0.05446546],
       [ 0.05361891, -0.05984585, -0.09117898, ..., -0.03094292,
        -0.01925805, -0.06308145],
       [-0.01667953, -0.04216413,  0.06239623, ..., -0.02024521,
        -0.05159656, -0.02373981]])}
alexa@ubuntu-xenial:0x00-binary_classification$
```

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `21-deep_neural_network.py`

Check your code? Ask a new correction QA Review

22. Train DeepNeuralNetwork

mandatory

Score: 0.00% (*Checks completed: 0.00%*)

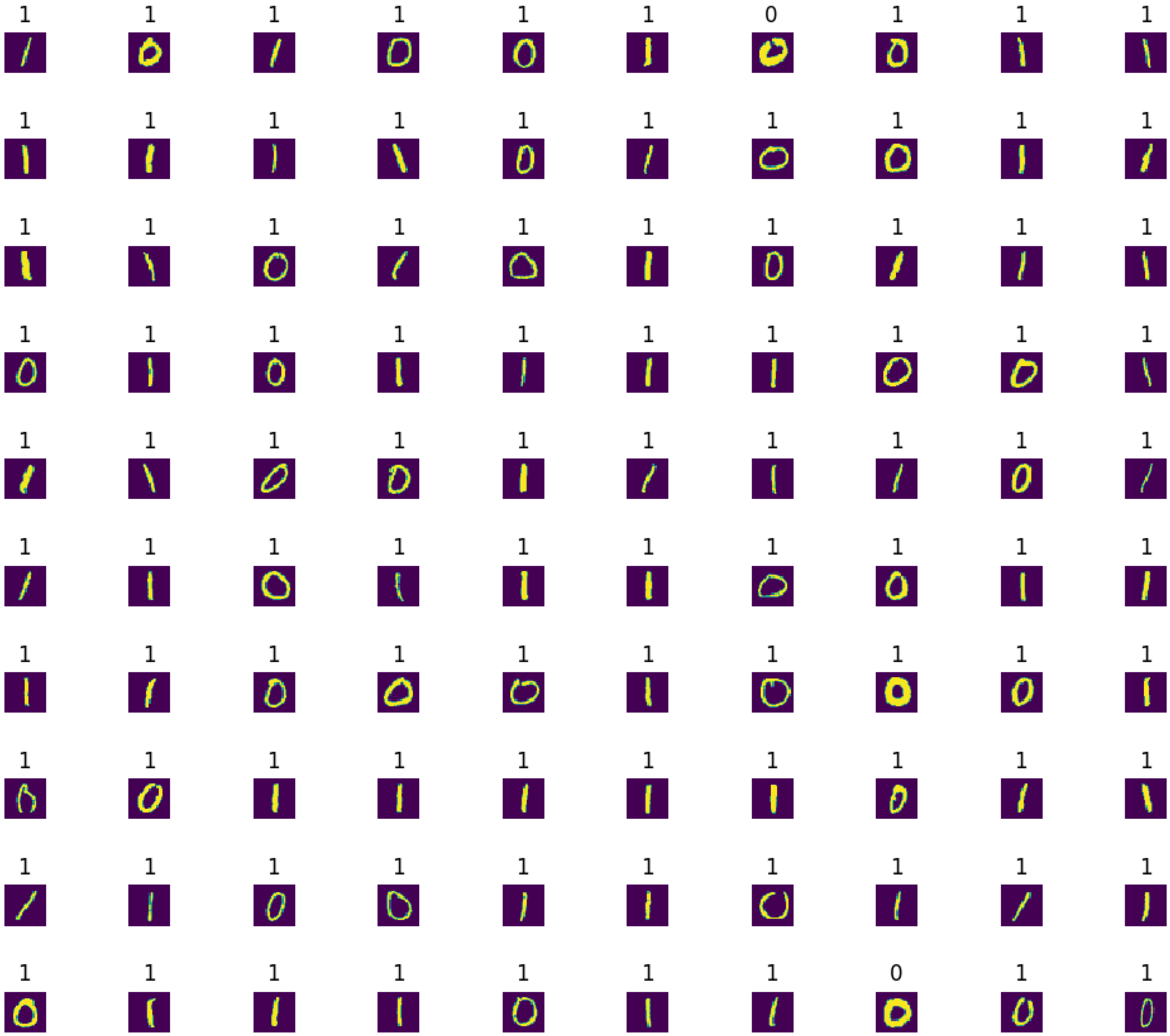
☐ Done?

Help

Write a class `DeepNeuralNetwork` that defines a deep neural network performing binary classification (based on `21-deep_neural_network.py`):

- Add the public method `def train(self, X, Y, iterations=5000, alpha=0.05):`
  - Trains the deep neural network
  - `X` is a `numpy.ndarray` with shape (`nx` , `m`) that contains the input data
    - `nx` is the number of input features to the neuron
    - `m` is the number of examples
  - `Y` is a `numpy.ndarray` with shape (1, `m`) that contains the correct labels for the input data
  - `iterations` is the number of iterations to train over
    - if `iterations` is not an integer, raise a `TypeError` with the exception `iterations must be an integer`
    - if `iterations` is not positive, raise a `ValueError` with the exception `iterations must be a positive integer`
  - `alpha` is the learning rate
    - if `alpha` is not a float, raise a TypeError with the exception `alpha must be a float`
    - if `alpha` is not positive, raise a ValueError with the exception `alpha must be positive`
  - All exceptions should be raised in the order listed above
  - Updates the private attributes `__weights` and `__cache`
  - You are allowed to use one loop
  - Returns the evaluation of the training data after `iterations` of training have occurred

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 22-main.py
#!/usr/bin/env python3

import matplotlib.pyplot as plt
import numpy as np

Deep = __import__('22-deep_neural_network').DeepNeuralNetwork

lib_train = np.load('../data/Binary_Train.npz')
X_train_3D, Y_train = lib_train['X'], lib_train['Y']
X_train = X_train_3D.reshape((X_train_3D.shape[0], -1)).T
lib_dev = np.load('../data/Binary_Dev.npz')
X_dev_3D, Y_dev = lib_dev['X'], lib_dev['Y']
X_dev = X_dev_3D.reshape((X_dev_3D.shape[0], -1)).T

np.random.seed(0)
deep = Deep(X_train.shape[0], [5, 3, 1])
A, cost = deep.train(X_train, Y_train, iterations=100)
accuracy = np.sum(A == Y_train) / Y_train.shape[1] * 100
print("Train cost:", cost)
print("Train accuracy: {}%".format(accuracy))
A, cost = deep.evaluate(X_dev, Y_dev)
accuracy = np.sum(A == Y_dev) / Y_dev.shape[1] * 100
print("Dev cost:", cost)
print("Dev accuracy: {}%".format(accuracy))
fig = plt.figure(figsize=(10, 10))
for i in range(100):
    fig.add_subplot(10, 10, i + 1)
    plt.imshow(X_dev_3D[i])
    plt.title(A[0, i])
    plt.axis('off')
plt.tight_layout()
plt.show()
alexa@ubuntu-xenial:0x00-binary_classification$ ./22-main.py
Train cost: 0.6444304786060048
Train accuracy: 56.241610738255034%
Dev cost: 0.6428913158565179
Dev accuracy: 57.730496453900706%
```



Hmm… doesn’t seem like this worked very well. Could it be because of our architecture or that it wasn’t trained properly? We need to see more information…

Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `22-deep_neural_network.py`

Check your code? Ask a new correction QA Review

23. Upgrade Train DeepNeuralNetwork mandatory

Score: 0.00% (*Checks completed: 0.00%*)

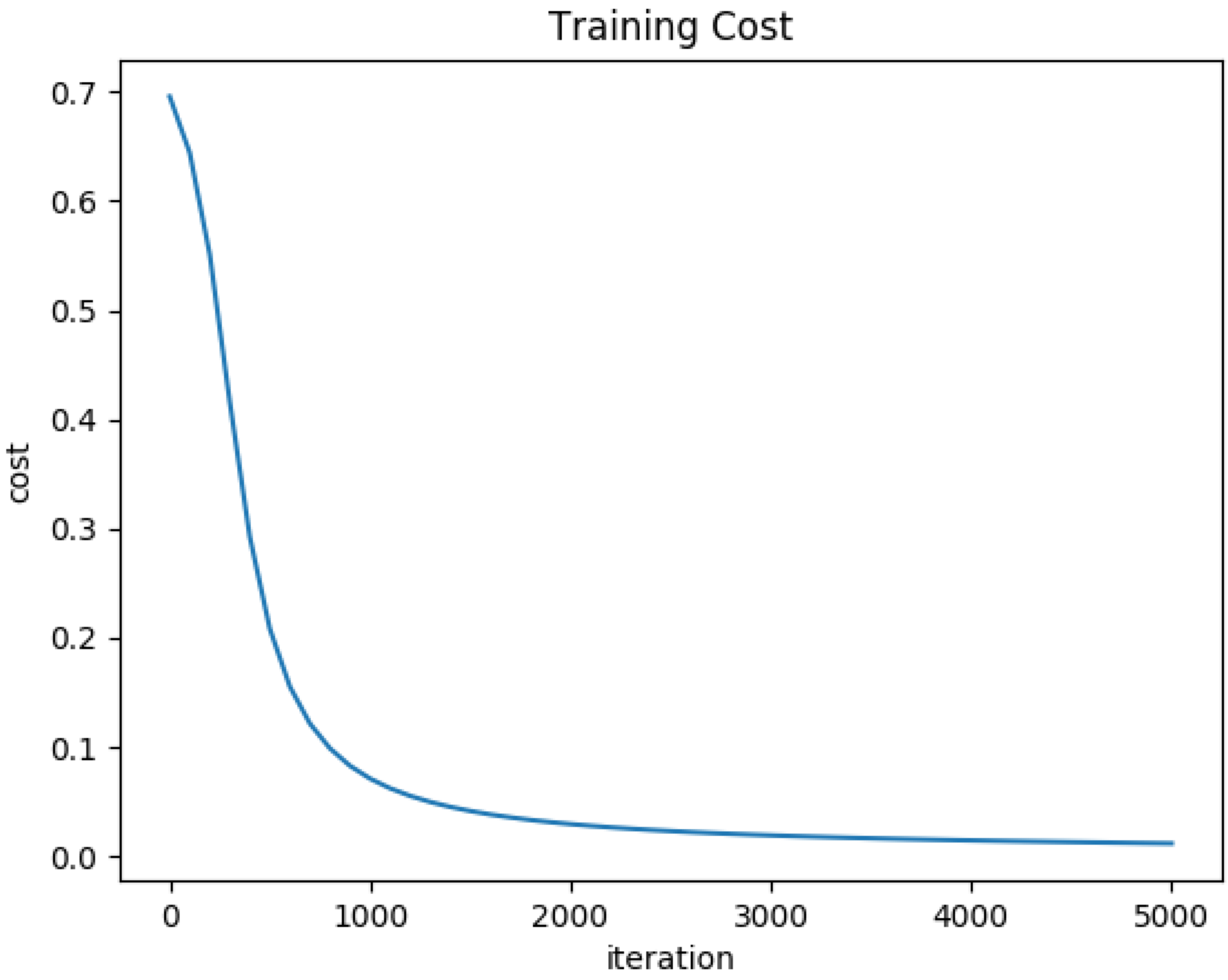
□ Done?

Help

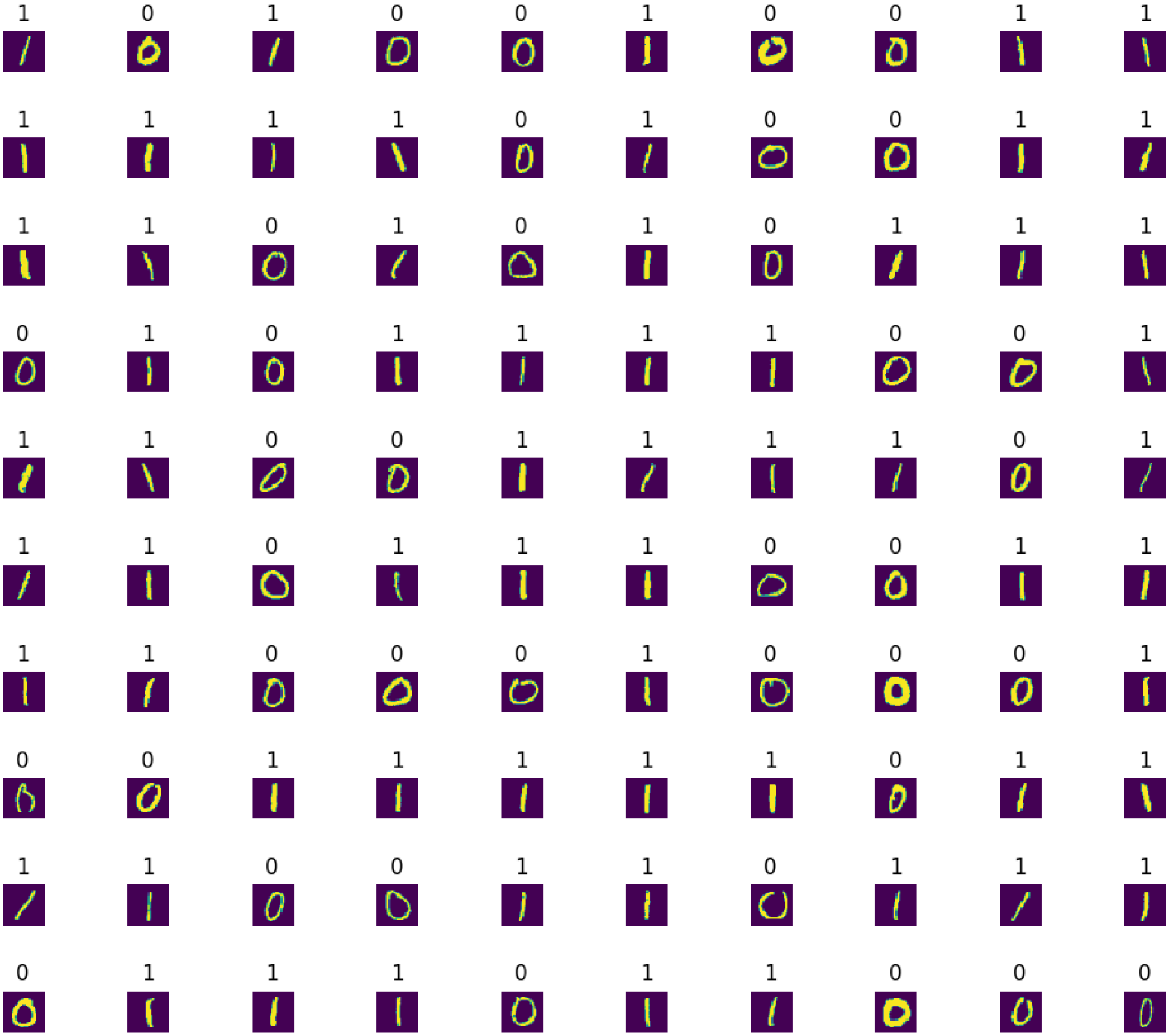
Write a class `DeepNeuralNetwork` that defines a deep neural network performing binary classification (based on `22-deep_neural_network.py`):

- Update the public method `train` to `def train(self, X, Y, iterations=5000, alpha=0.05, verbose=True, graph=True, step=100):`
    - Trains the deep neural network by updating the private attributes `__weights` and `__cache`
    - `X` is a `numpy.ndarray` with shape (`nx`, `m`) that contains the input data
        - `nx` is the number of input features to the neuron
        - `m` is the number of examples
    - `Y` is a `numpy.ndarray` with shape (1, `m`) that contains the correct labels for the input data
    - `iterations` is the number of iterations to train over
        - if `iterations` is not an integer, raise a `TypeError` with the exception `iterations must be an integer`
        - if `iterations` is not positive, raise a `ValueError` with the exception `iterations must be a positive integer`
    - `alpha` is the learning rate
        - if `alpha` is not a float, raise a `TypeError` with the exception `alpha must be a float`
        - if `alpha` is not positive, raise a `ValueError` with the exception `alpha must be positive`
    - `verbose` is a boolean that defines whether or not to print information about the training. If `True`, print `Cost after {iteration} iterations: {cost}` every `step` iterations:
        - Include data from the 0th and last iteration
    - `graph` is a boolean that defines whether or not to graph information about the training once the training has completed. If `True`:
        - Plot the training data every `step` iterations as a blue line
        - Label the x-axis as `iteration`
        - Label the y-axis as `cost`
        - Title the plot `Training Cost`
        - Include data from the 0th and last iteration
    - Only if either `verbose` or `graph` are `True`:
        - if `step` is not an integer, raise a `TypeError` with the exception `step must be an integer`
        - if `step` is not positive and less than or equal to `iterations`, raise a `ValueError` with the exception `step must be positive and <= iterations`
    - All exceptions should be raised in the order listed above
    - The 0th iteration should represent the state of the neuron before any training has occurred
    - You are allowed to use one loop
    - Returns the evaluation of the training data after `iterations` of training have occurred

```
alexa@ubuntu-xenial:0x00-binary_classification$ cat 23-main.py
#!/usr/bin/env python3

import matplotlib.pyplot as plt
import numpy as np

Deep = __import__('23-deep_neural_network').DeepNeuralNetwork

lib_train = np.load('../data/Binary_Train.npz')
X_train_3D, Y_train = lib_train['X'], lib_train['Y']
X_train = X_train_3D.reshape((X_train_3D.shape[0], -1)).T
lib_dev = np.load('../data/Binary_Dev.npz')
X_dev_3D, Y_dev = lib_dev['X'], lib_dev['Y']
X_dev = X_dev_3D.reshape((X_dev_3D.shape[0], -1)).T

np.random.seed(0)
deep = Deep(X_train.shape[0], [5, 3, 1])
A, cost = deep.train(X_train, Y_train)
accuracy = np.sum(A == Y_train) / Y_train.shape[1] * 100
print("Train cost:", cost)
print("Train accuracy: {}%".format(accuracy))
A, cost = deep.evaluate(X_dev, Y_dev)
accuracy = np.sum(A == Y_dev) / Y_dev.shape[1] * 100
print("Dev cost:", cost)
print("Dev accuracy: {}%".format(accuracy))
fig = plt.figure(figsize=(10, 10))
for i in range(100):
    fig.add_subplot(10, 10, i + 1)
    plt.imshow(X_dev_3D[i])
    plt.title(A[0, i])
    plt.axis('off')
plt.tight_layout()
plt.show()
alexa@ubuntu-xenial:0x00-binary_classification$ ./23-main.py
Cost after 0 iterations: 0.6958649419170609
Cost after 100 iterations: 0.6444304786060048

...

Cost after 5000 iterations: 0.011671820326008168
```



```
Train cost: 0.011671820326008168
Train accuracy: 99.88945913936044%
Dev cost: 0.00924955213227925
Dev accuracy: 99.95271867612293%
```



Repo:

- GitHub repository: `holbertonschool-machine_learning`
- Directory: `supervised_learning/0x00-binary_classification`
- File: `23-deep_neural_network.py`

QA Review

Ask a new correction