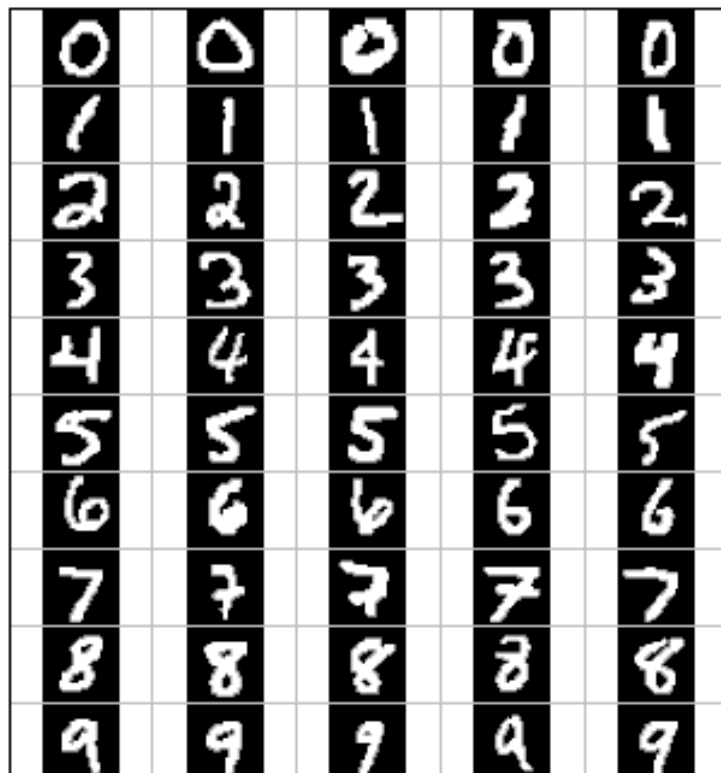




Project 3: Deep Learning



Which digit?

 This page is in Spanish  Would you like to translate it?

Which language?

Introduction

This project will be an introduction to deep learning with **PyTorch**. In this project, you will design five models to solve the following four problems: Digit Classification, Non-Linear Regression, Automatic Gradient Computation, Language Identification.

The code for this project includes the following files and data, available on the

<http://learn.tsinghua.edu.cn/>

Fills you will edit:

`models.py`: The location where you will write your models to solve each problem;

Files you should be NOT edit:

`data`: Datasets for digit classification and language identification

`backend.py`: Backend code for various tasks.

`autograder.py`: Project autograder

What to submit:

You will fill in portions of `models.py` (only) during the assignment, and submit them.

Evaluation:

Your code will be autograded for technical correctness. Please **DO NOT** change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder.

Academic Dishonesty:

We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; please don't let us down. Instead, contact the course staff if you are having trouble.

Installation

For this project, you will need to install the following three libraries:

`numpy`: provides support for large multi-dimensional arrays - installation instructions.

`pytorch`: an open source machine learning framework that accelerates the path from research prototyping to production deployment. In this project, you should use **pytorch version ≥ 1.0** .

`matplotlib`: a 2D plotting library.

For this project, you should be familiar with the following two libraries:

`numpy`: <https://numpy.org/doc/stable/user/quickstart.html>

`pytorch`: <https://pytorch.org/docs/stable/index.html>

A simple pytorch tutorial can be found in `PyTorch.ipynb`. Please read and run the notebook carefully. You may find it useful for the following tasks.

How to install pytorch

For Windows/Linux User: Run the code with

```
conda install pytorch torchvision cpuonly -c pytorch
```

For Mac User: Run the code with

```
conda install pytorch torchvision -c pytorch
```

Then the latest version of PyTorch is installed. Run the following command to confirm that the above packages are successfully installed:

```
python autograder.py --check-dependencies
```

How to run the ipython notebook

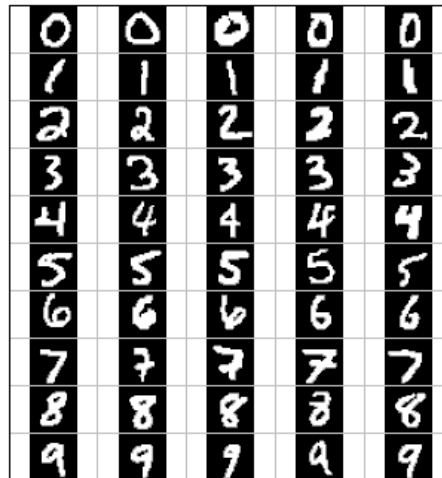
IPython Notebook is an interactive computational environment, in which you can combine code execution, rich text, mathematics, plots and rich media.

You can run IPython Notebook as follows:

- Run `ipython notebook --port 8888` (you may use other port) in your terminal;
- Open a browser and visit website `http://localhost:8888`.
- Input token/password given in your terminal.

Task 1: Digit Classification (4 points)

For this task, you will train a convolutional neural network to classify handwritten digits from the MNIST dataset.



Each digit is of size 28×28 pixels, the values of which are stored in a $1 \times 28 \times 28$ tensors of floating-point numbers. The output of the network is a 10-dimensional real-valued vector, and each element is the score corresponding the particular class (0-9)

In this task, you should **design a 5-layer convolutional neural network, the SGD optimizer with momentum, and the cross entropy loss**. The network structure should be as follows:

- (1) 3x3 convolution layer (**without bias**) with filter size 16, stride 1, padding 0;
- BatchNorm layer with filter size 16;
- ReLU activation.
- (2) 3x3 convolution layer (**without bias**) with filter size 32, stride 1, padding 0;
- BatchNorm layer with filter size 32;
- ReLU activation.
- 2x2 max pool with stride 2.
- (3) 3x3 convolution layer (**without bias**) with filter size 64, stride 1, padding 0;
- BatchNorm layer with filter size 64;
- ReLU activation.
- 2x2 max pool with stride 2.
- Flatten the feature map (The dimension should be computed by yourself).
- (4) Fully connected layer **with bias**, producing 500 outputs;
- ReLU activation.
- Fully connected layer **with bias**, producing 10 outputs.

Implement the `DigitClassificationModel` class in `models.py`, where higher scores indicate a higher probability of a digit belonging to a particular class (0-9).

To test your implementation, run the autograder: `python autograder.py -q q1`. Note that the training process may take as long as 10 minutes. You can find the visualization results (images and the predictions) in `figures/q1.png`.

Grading:

If you successfully build the required model, you will get 1 point for that.

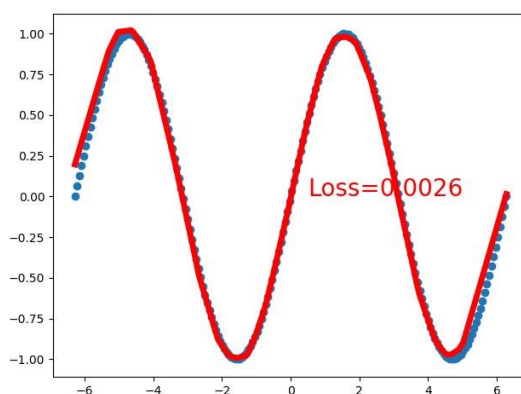
To receive full points for this task, your model should achieve the accuracy of at least 98% on the test set after training for one epoch. For reference, our staff implementation consistently achieves an accuracy of 98.5% on the test set after training for one epoch.

Question 1 (Answer it in your Report)

What is the role of the BatchNorm layer? Just give ONE aspect you think is important. To answer this question, you may remove all BatchNorm layers and retrain the model, then different classification results will get.

Task 2: Non-Linear Regression (4 points)

For this task, you will train a neural network to approximate $\sin(x)$ over $[-2\pi, 2\pi]$.



You will need to implement the `RegressionModel` class in `models.py`.

To test your implementation, run the autograder: `python autograder.py -q q2`. You can find the visualization results in `figures/q2.png`.

Grading:

If you successfully build a runnable model, you will get 1 point.

You will receive full points if your implementation gets a MSE error of 0.02 or better, averaged across all examples in the dataset. Note that the training process may take a few minutes.

Task 3: Automatic Gradient Computation (3 points)

For this task, you will use “Fast Gradient Sign Method (FGSM)” to generate adversarial examples of a given handwritten digit from the MNIST dataset to fool the LeNet model.

Each handwritten digit is a 28x28 pixel grayscale image. Each entry in the vector is a floating-point number between 0 and 1. You should make small perturbation of the original data which will result in wrong predictions.

The FGSM method for generating adversarial examples x' is shown as follows

$$x' = x + \varepsilon \cdot \text{sgn}(\nabla_x L(\theta, x, y))$$

where ε is the perturbation scale, $L(\theta, x, y)$ is the *cross entropy loss* for training the classifiers. The main difficulty for this task is computing the gradients regarding to the input data. You should use PyTorch to accomplish the automatic gradient computation.

You will need to implement the `DigitAttackModel` class in `models.py`.

To test your implementation, run the autograder: `python autograder.py -q q3`. You can find the visualization results (adversarial examples and the predictions) in `figures/q3.png`.

Grading:

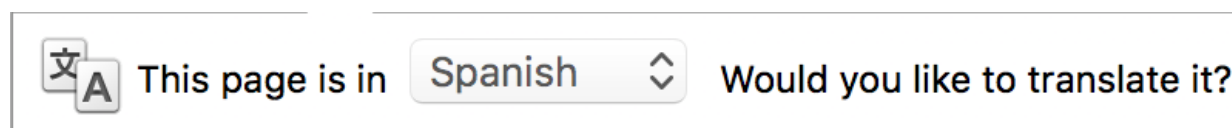
To receive points for this task, the adversarial examples with your model generates should make an accuracy less than 50% when $\varepsilon = 0.2$.

Question 2 (Answer it in your Report)

Visualize the adversarial examples in `figures/q3.png`. Describe what the adversarial examples look like. Give ONE possible reason for the cause of the adversarial example underlying the neural network.

Task 4: Language Identification (4 points + 1 bonus)

Language identification is the task of figuring out, given a piece of text, what language the text is written in. For example, your browser might be able to detect if you've visited a page in a foreign language and offer to translate it for you. Here is an example from Chrome (which uses a neural network to implement this feature):



In this project, you should build a neural network model that identifies language for one word at a time. Our dataset consists of words in five languages, such as the table below:

Word	Language
discussed	English
eternidad	Spanish
itseänne	Finnish
paleis	Dutch
mieszkać	Polish

Different words consist of different numbers of letters, so our model needs to have an architecture that can handle variable-length inputs. Instead of a single input xx (like in the previous tasks), we'll have a separate input for each character in the word: x_0, x_1, \dots, x_{L-1} where L is the length of the word. We'll start by applying a network $f_{initial}$ that is just like the feed-forward networks in the previous problems. It accepts its input x_0 and computes an output vector h_1 of dimensionality d :

$$h_1 = f_{initial}(x_0)$$

Next, we'll combine the output of the previous step with the next letter in the word, generating a vector summary of the first two letters of the word. To do this, we'll apply a sub-network that accepts a letter and outputs a hidden state, but now also depends on the previous hidden state h_1 . We denote this sub-network as f .

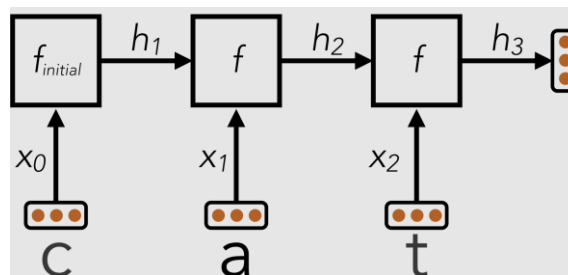
$$h_2 = f(h_1, x_1)$$

This pattern continues for all letters in the input word, where the hidden state at each step summarizes all the letters the network has processed thus far:

$$h_3 = f(h_2, x_2)$$

...

The technique described above is called a Recurrent Neural Network (RNN). A schematic diagram of the RNN is shown below:



Here, an RNN is used to encode the word “cat” into a fixed-size vector h_3 .

After the RNN has processed the full length of the input, it has encoded the arbitrary-length input word into a fixed-size vector h_L , where L is the length of the word. This vector summary of the input word can now be fed through additional output layers to generate classification scores for the word’s language identity.

You will need to implement the `LanguageIDModelModel` class in `models.py`. You may use RNN or GRU or LSTM to build you model.

To test your implementation, run the autograder: `python autograder.py -q q4`. Note that the training process may take as long as 10 minutes.

Grading:

If you successfully build a runnable model, you will get 1 point. Other points for achieving the accuracy such that

Points	Test Accuracy
1	$\geq 60\%$
2	$\geq 75\%$
3	$\geq 80\%$

Extra credit for assignments with the best 10% classification accuracy.

Congratulations! You've finished with Projects 4.