

Intro to deep learning

Dr. Janoś Gabler, University of Bonn

Lecture 9: Training neural networks in Pytorch



Topics

- Git and GitHub
- Recap: Feed-forward neural networks from scratch
- Defining Models in Pytorch
- Writing Training Loops in Pytorch
- Optimization algorithms

Git and GitHub

What is Git

- Distributed version control system
 - Go back in time
 - Have multiple versions of code
- Used by almost every programmer in the world
- Code is in a git-repository (the parent folder of your project)
- Used from the terminal
- Takes a while to learn, but we only need the absolute basics

What is GitHub

- Webpage where you can upload git repositories
- Collaboration tools
 - Pull requests
 - Review features
 - Automated tests
- You need to sign up for an account

What is GitHub Classroom

- Helps me to collect your assignments
- Helps you by creating a repository for you

Create a GitHub Account and accept invitation

- Go to <https://github.com/>
- Create an account if you don't have one
- Choose a name that is easy to type, memorize and pronounce
- Accept the invitation

dl-intro


Accept the assignment — final-project

Once you accept this assignment, you will be granted access to the `final-project-janosg` repository in the [iame-uni-bonn](#) organization on GitHub.

Accept this assignment



You accepted the assignment, **final-project**. We're configuring your repository now. This may take a few minutes to complete. Refresh this page to see updates.

 Your assignment is due by **Sep 10, 2023, 23:00 CEST**

Note: You may receive an email invitation to join [iame-uni-bonn](#) on your behalf. No further action is necessary.



Join the GitHub Student Developer Pack

Verified students receive free GitHub Pro plus thousands of dollars worth of the best real-world tools and training from GitHub Education partners — for free. [Learn more](#)


[Apply](#)



You're ready to go!

You accepted the assignment, **final-project**.

Your assignment repository has been created:

 <https://github.com/iame-uni-bonn/final-project-janosg>

We've configured the repository associated with this assignment ([update](#)).

 Your assignment is due by **Sep 10, 2023, 23:00 CEST**




Join the GitHub Student Developer Pack

Verified students receive free GitHub Pro plus thousands of dollars worth of the best real-world tools and training from GitHub Education partners — for free. [Learn more](#)

Apply

 [main](#)

 [1 branch](#)

 [0 tags](#)

[Go to file](#)[Add file](#)[Code](#)[About](#)[github-classroom\[bot\]](#) Add assignment deadline url

7b29418

1 minute ago



1 commit

[README.md](#)

Add assignment deadline url

1 minute ago

README.md[Review the assignment due date](#)

final-project-janosg created by GitHub Classroom

[Readme](#)[Activity](#)[0 stars](#)[1 watching](#)[0 forks](#)

Releases

No releases published

[Create a new release](#)

Packages

No packages published

[Publish your first package](#)

© 2023 GitHub, Inc.

[Terms](#)[Privacy](#)[Security](#)[Status](#)[Docs](#)[Contact GitHub](#)[Pricing](#)[API](#)[Training](#)[Blog](#)[About](#)

Clone the repository to your computer

- Cloning means downloading the repository to your computer
- You do it from a terminal
 - Open the terminal
 - Navigate to a folder to which you want to download the repo
 - Use git-clone

Get the clone link online

The screenshot shows the GitHub interface for the repository `iame-uni-bonn / final-project-janosg`. The `Code` tab is selected, and the dropdown menu is open, displaying the `Clone` option. The clone URL is `https://github.com/iame-uni-bonn/final-pr`. The repository is private and has 1 branch and 0 tags. The `README.md` file is visible, and the `github-classroom[bot]` user is mentioned in the commit history. The right sidebar shows the `About` section, indicating the repository was created by GitHub Classroom, and the `Releases` and `Packages` sections, both showing no published items.

Search or jump to... Pull requests Issues Codespaces Marketplace Explore

iame-uni-bonn / final-project-janosg Private Edit Pins Unwatch 1 Fork 0 Star 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags

github-classroom[bot] Add assignment deadline url

README.md Add assignment deadline url

README.md

Review the assignment due date

Local Codespaces

Clone

HTTPS SSH GitHub CLI

https://github.com/iame-uni-bonn/final-pr

Use Git or checkout with SVN using the web URL

Download ZIP

About

final-project-janosg created by GitHub Classroom

Readme Activity 0 stars 1 watching 0 forks

Releases

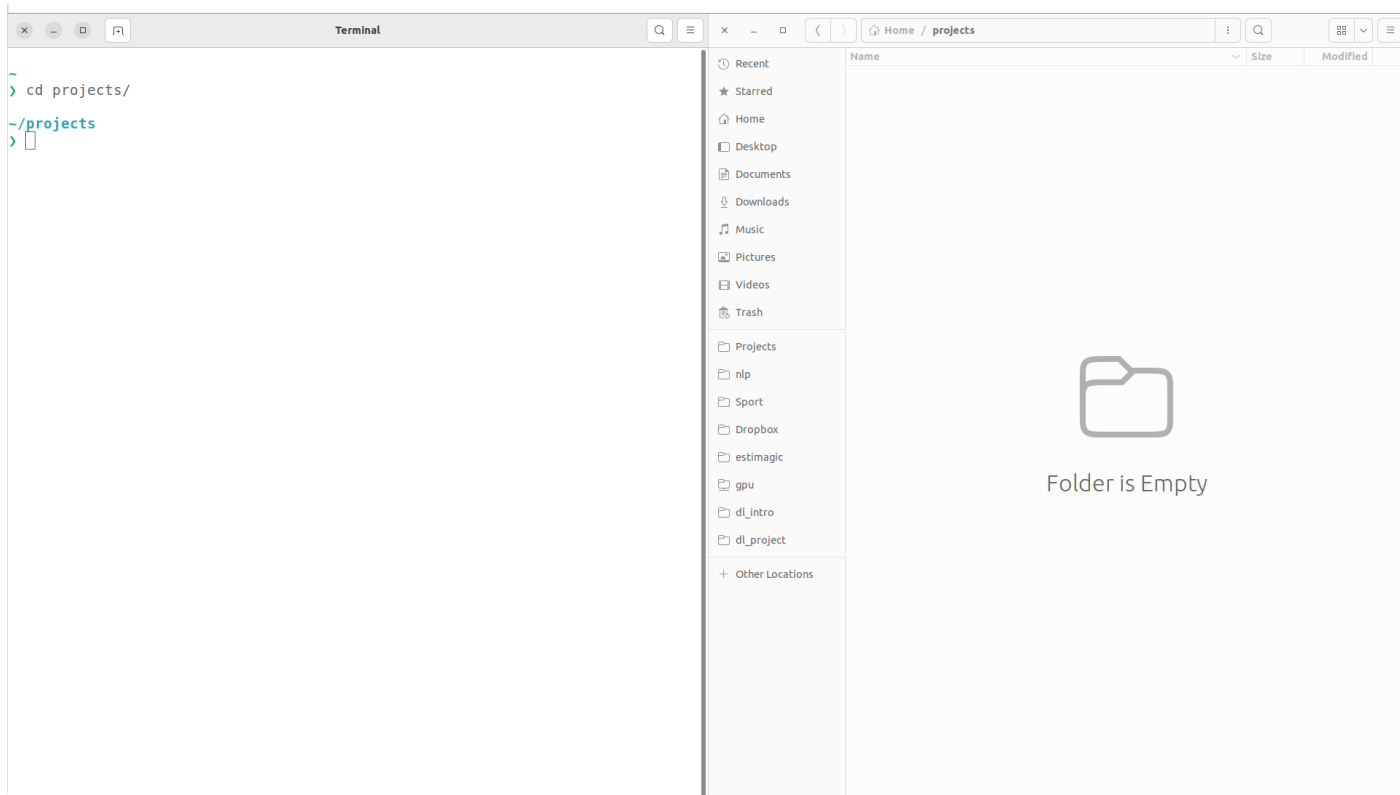
No releases published Create a new release

Packages

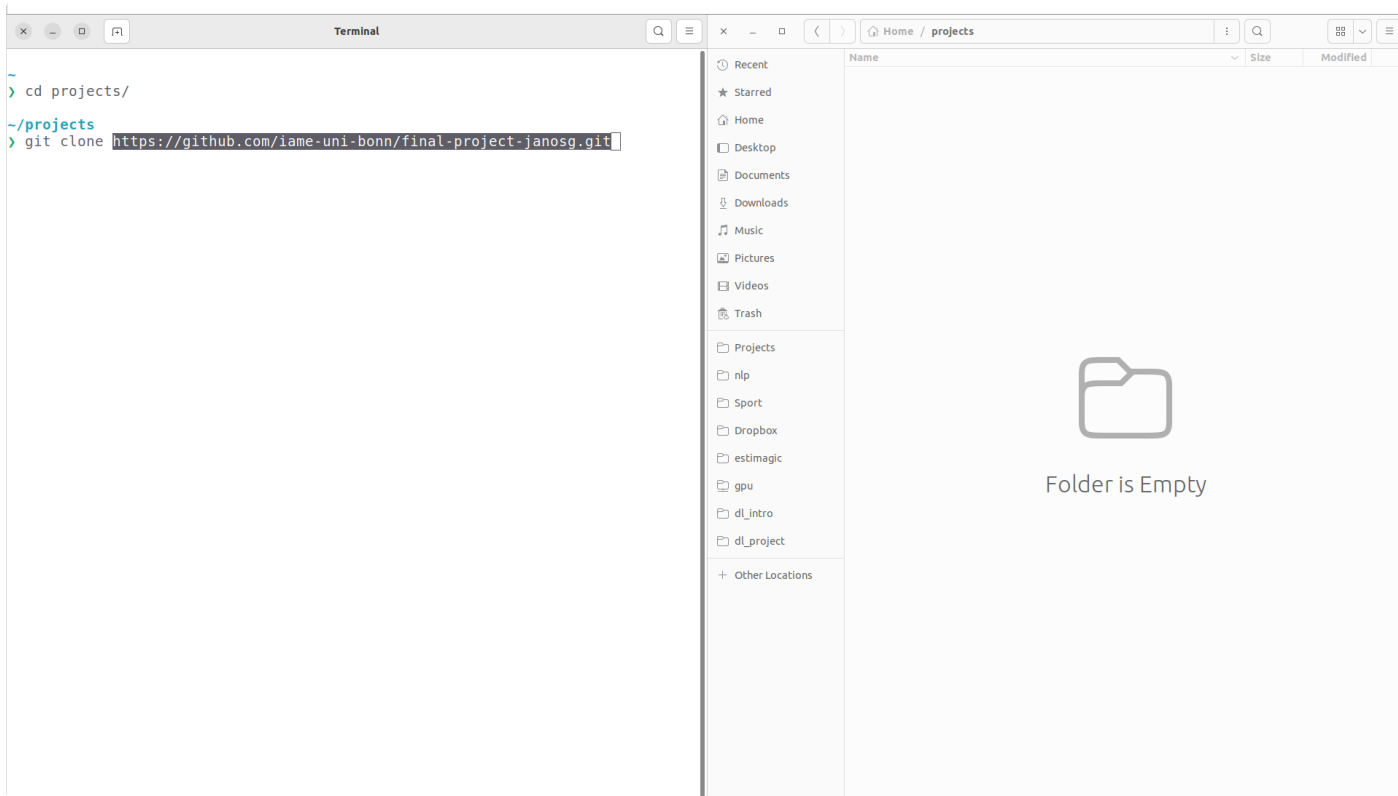
No packages published Publish your first package

© 2023 GitHub, Inc. Terms Privacy Security Status Docs Contact GitHub Pricing API Training Blog About

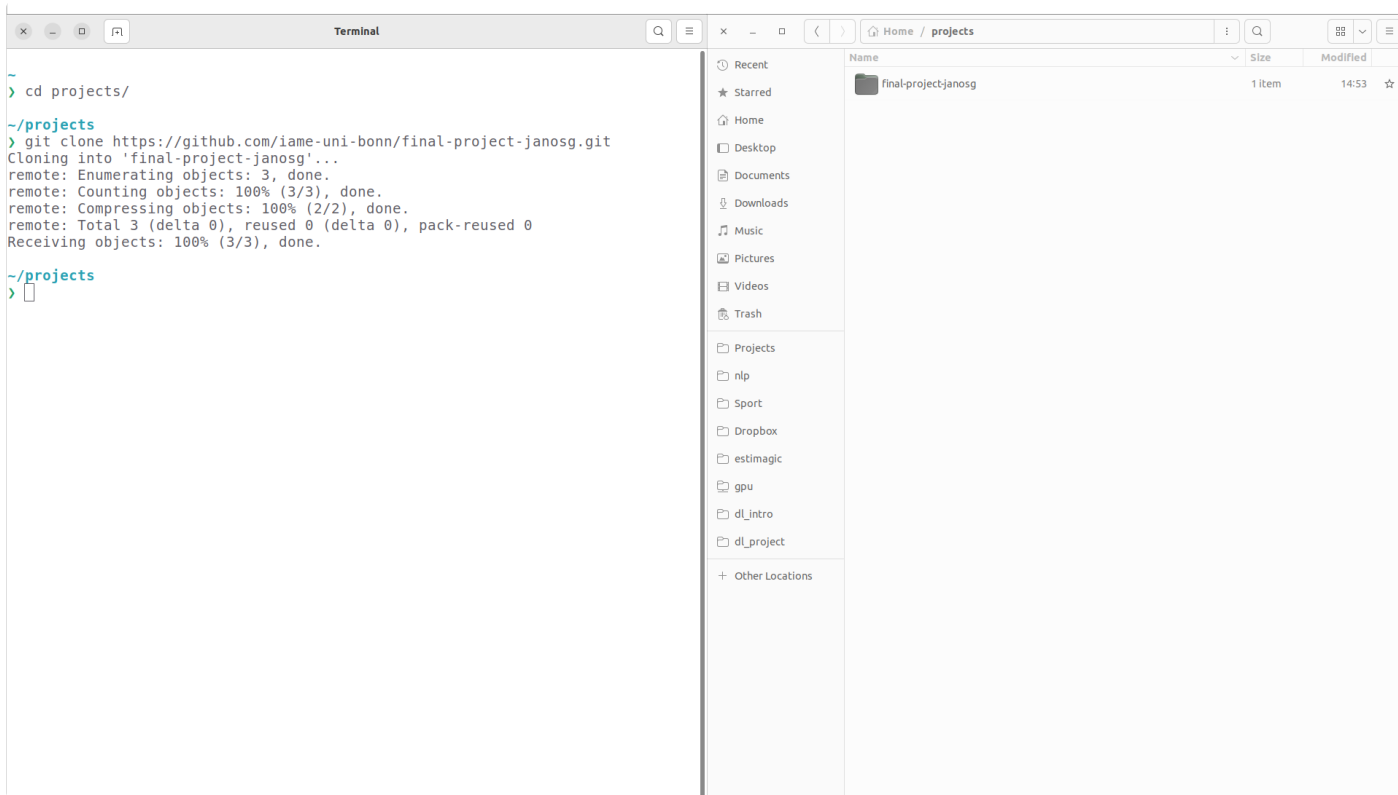
Open a terminal and navigate to folder



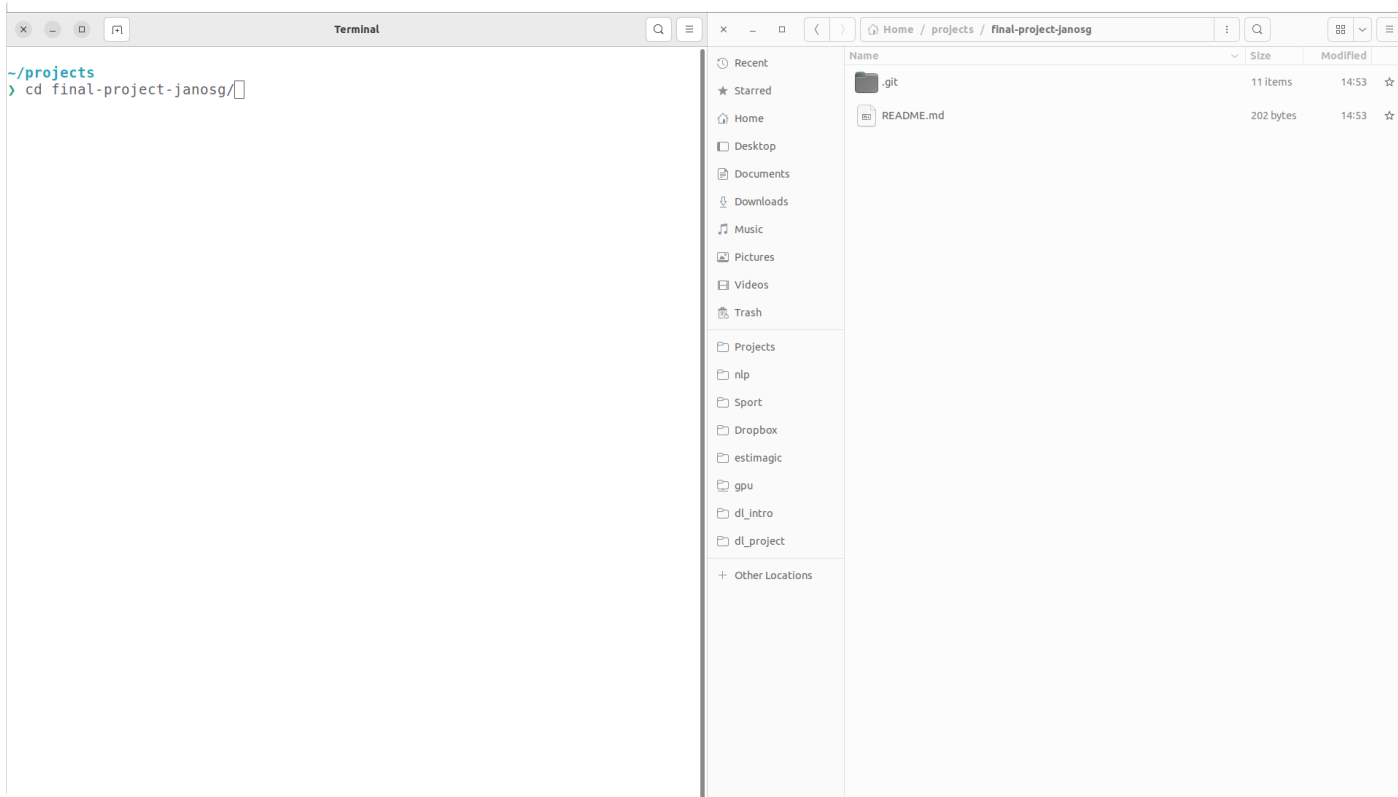
Type clone command



Hit Enter to download repo



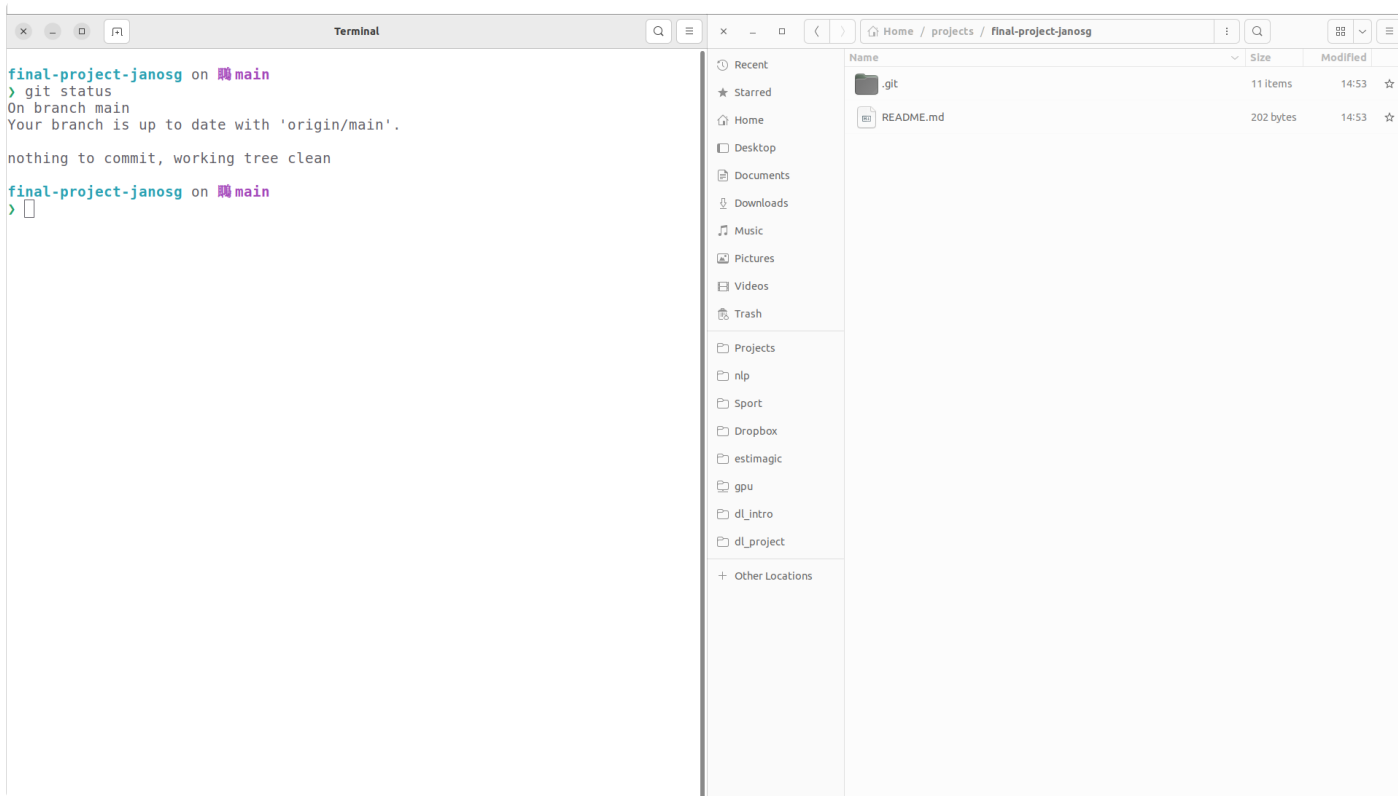
``cd`` into the repo



``git status``

- Executing ``git status`` inside a git repository gives you information
 - Which files were added?
 - Which files were modified
 - Are there un-pushed changes?
- It should be every other git command you type!

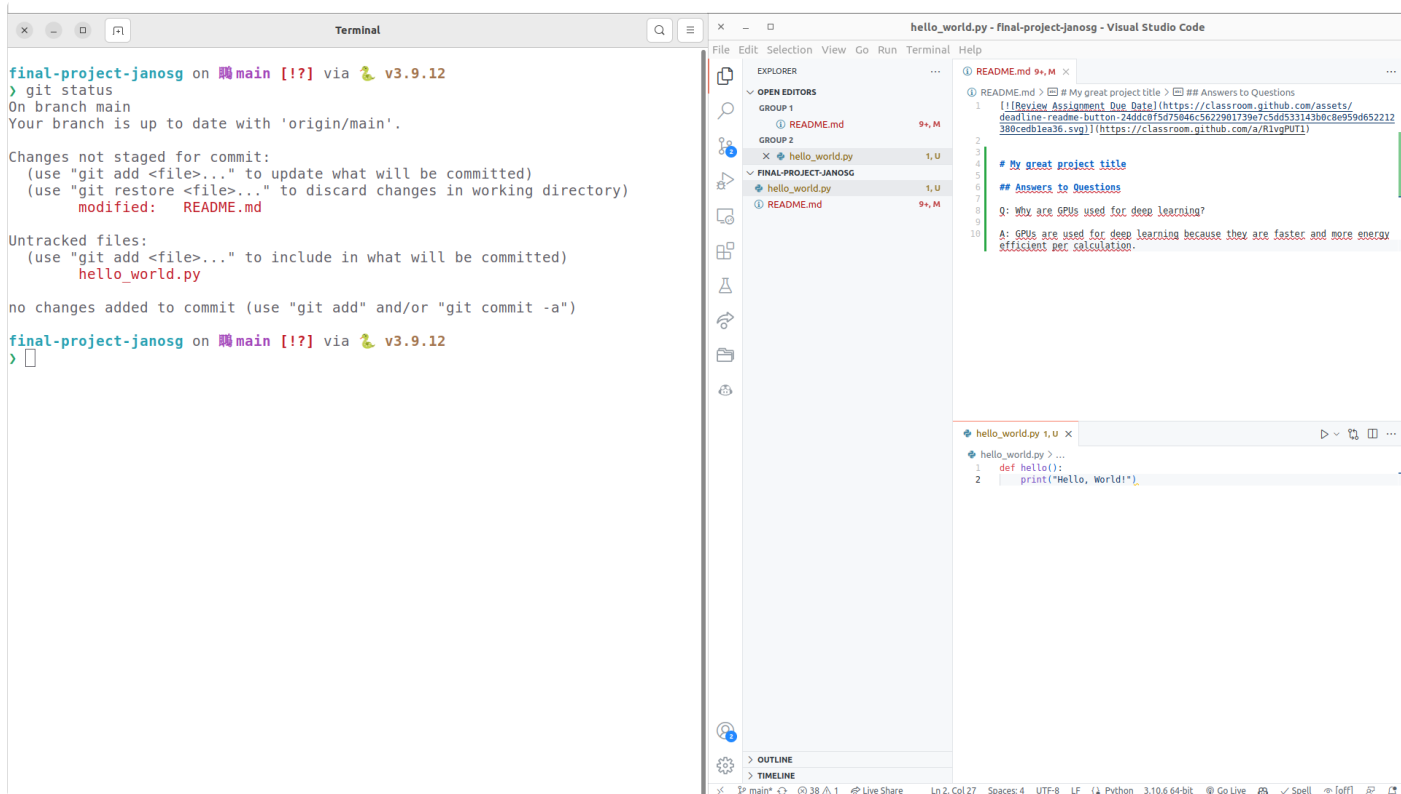
Status before changes



Make some changes

- You can now make your changes
 - Add files
 - Modify existing files
- The changes will not be synchronized to github automatically
- To share them, you need to commit and push

Status after changing and creating files



The screenshot displays a terminal window on the left and the Visual Studio Code editor on the right. The terminal shows the output of a `git status` command, indicating that the branch is up to date with 'origin/main' and listing untracked files. The VS Code interface shows the Explorer view with a file tree containing `README.md`, `hello_world.py`, and `FINAL-PROJECT-JANOSG`. The Editor view shows the content of `README.md` and `hello_world.py`.

```
final-project-janosg on 主 main [!?] via Python v3.9.12
> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        hello_world.py

no changes added to commit (use "git add" and/or "git commit -a")

final-project-janosg on 主 main [!?] via Python v3.9.12
>
```

Visual Studio Code Explorer View:

- GROUP 1
 - README.md 9+, M
- GROUP 2
 - hello_world.py 1, U
 - FINAL-PROJECT-JANOSG
 - hello_world.py 1, U
 - README.md 9+, M

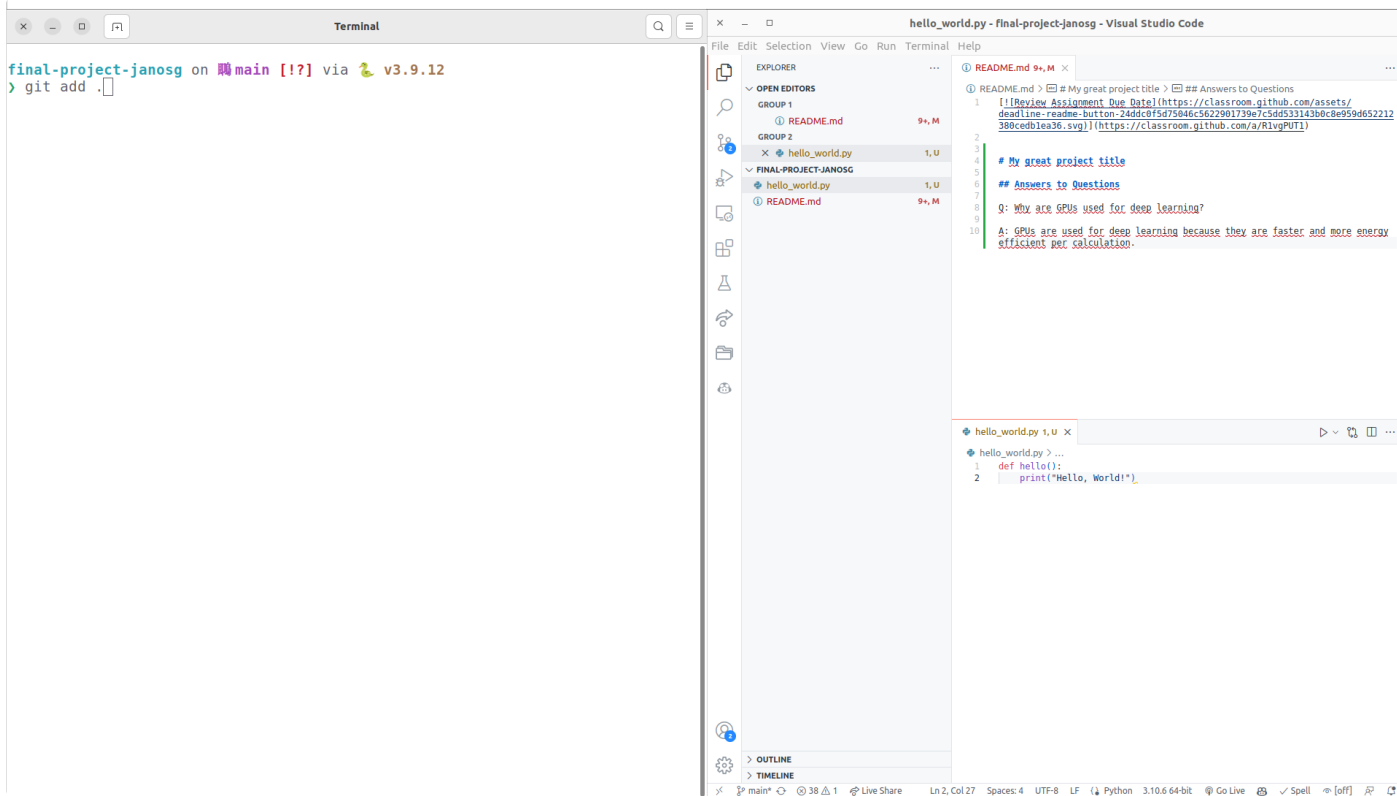
Visual Studio Code Editor View (README.md):

```
1  README.md > 1 # My great project title > 1 ## Answers to Questions
2  1  [Review Assignment Due Date](https://classroom.github.com/assets/
3  deadline-readme-button-24ddcf5d75b46c5622901739e7c5d4533143b8c8e959d652212
4  380cedblea36.svg)](https://classroom.github.com/a/RivgPUTI)
5
6  2  # My great project title
7  3  ## Answers to Questions
8  4  Q: Why are GPUs used for deep learning?
9  5  A: GPUs are used for deep learning because they are faster and more energy
10 6  efficient per calculation.
```

Visual Studio Code Editor View (hello_world.py):

```
1  def hello():
2  print("Hello, World!")
```

Use `git add .`



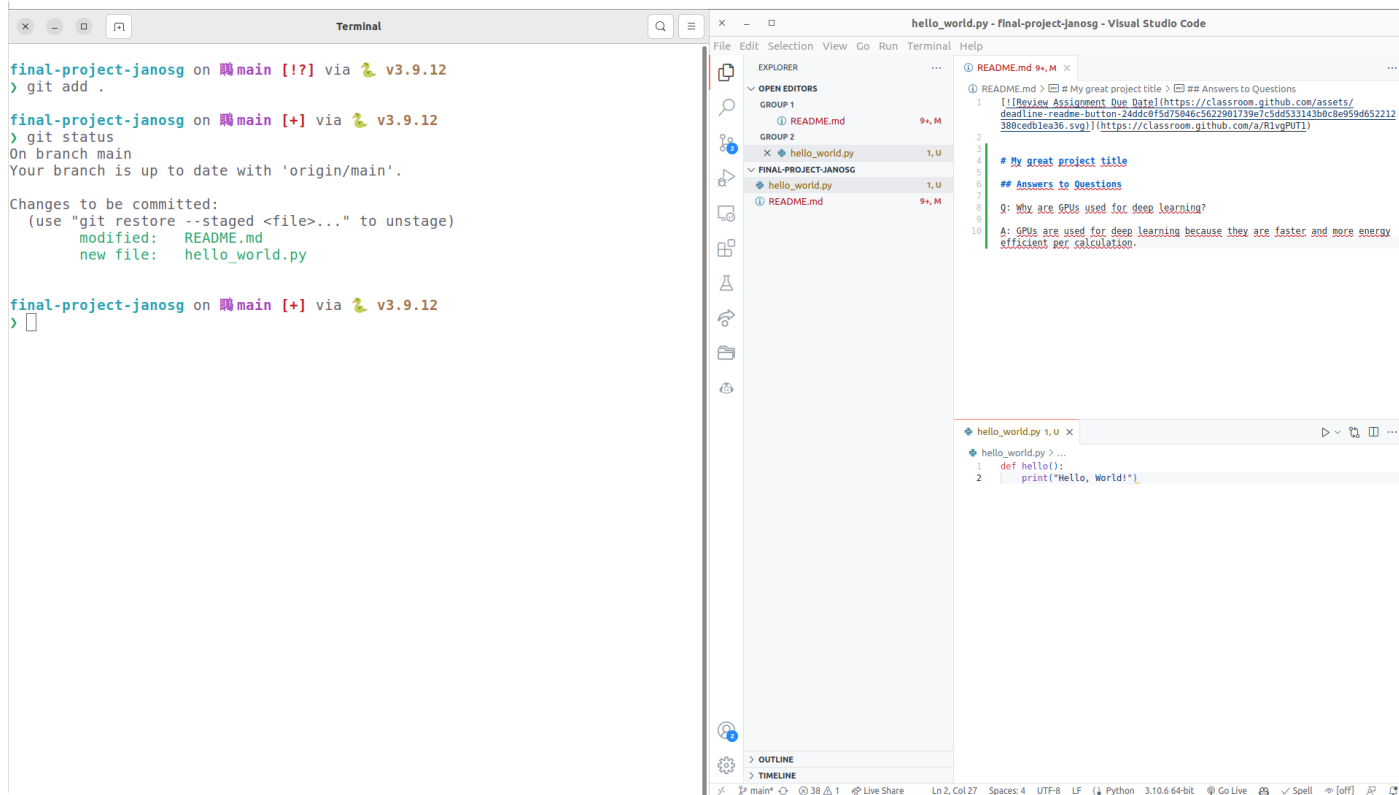
The screenshot displays the Visual Studio Code interface. On the left, a terminal window shows the command `git add .` being executed in a shell. The main editor area is split into two panes. The top pane shows the `README.md` file, which contains a project title, a link to a GitHub repository, and a Q&A section about GPUs. The bottom pane shows the `hello_world.py` file, which contains a simple Python function `hello()` that prints "Hello, World!". The Explorer sidebar on the left shows the project structure with `hello_world.py` and `README.md` files.

```
final-project-janosg on main [!?] via v3.9.12
> git add .

README.md
1 | [[Replow Assignment Due Date](https://classroom.github.com/assets/
2 | deadline-readme-button-24ddc0f5d75046c5622901739e7C5dd533143b0c8e959d652212
3 | 380cedb1ea36.svg)](https://classroom.github.com/a/R1vgPUT1)]
4 |
5 | # My great project title
6 |
7 | ## Answers to Questions
8 |
9 | Q: Why are GPUs used for deep learning?
10 | A: GPUs are used for deep learning because they are faster and more energy
    efficient per calculation.

hello_world.py 1, U
1 | def hello():
2 |     print("Hello, World!")
```

Status after `git add .`



The screenshot shows a terminal window and the Visual Studio Code interface. The terminal window displays the output of the `git add .` command, indicating that the files `README.md` and `hello_world.py` have been staged for commit. The Visual Studio Code interface shows the Explorer view with the `hello_world.py` file selected, and the Editor view showing the contents of the file.

```
final-project-janosg on main [!?] via v3.9.12
> git add .

final-project-janosg on main [+] via v3.9.12
> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
        new file:   hello_world.py

final-project-janosg on main [+] via v3.9.12
>
```

Visual Studio Code Explorer View:

- GROUP 1
 - README.md 9+, M
- GROUP 2
 - hello_world.py 1, U
- FINAL-PROJECT-JANOSG
 - hello_world.py 1, U
 - README.md 9+, M

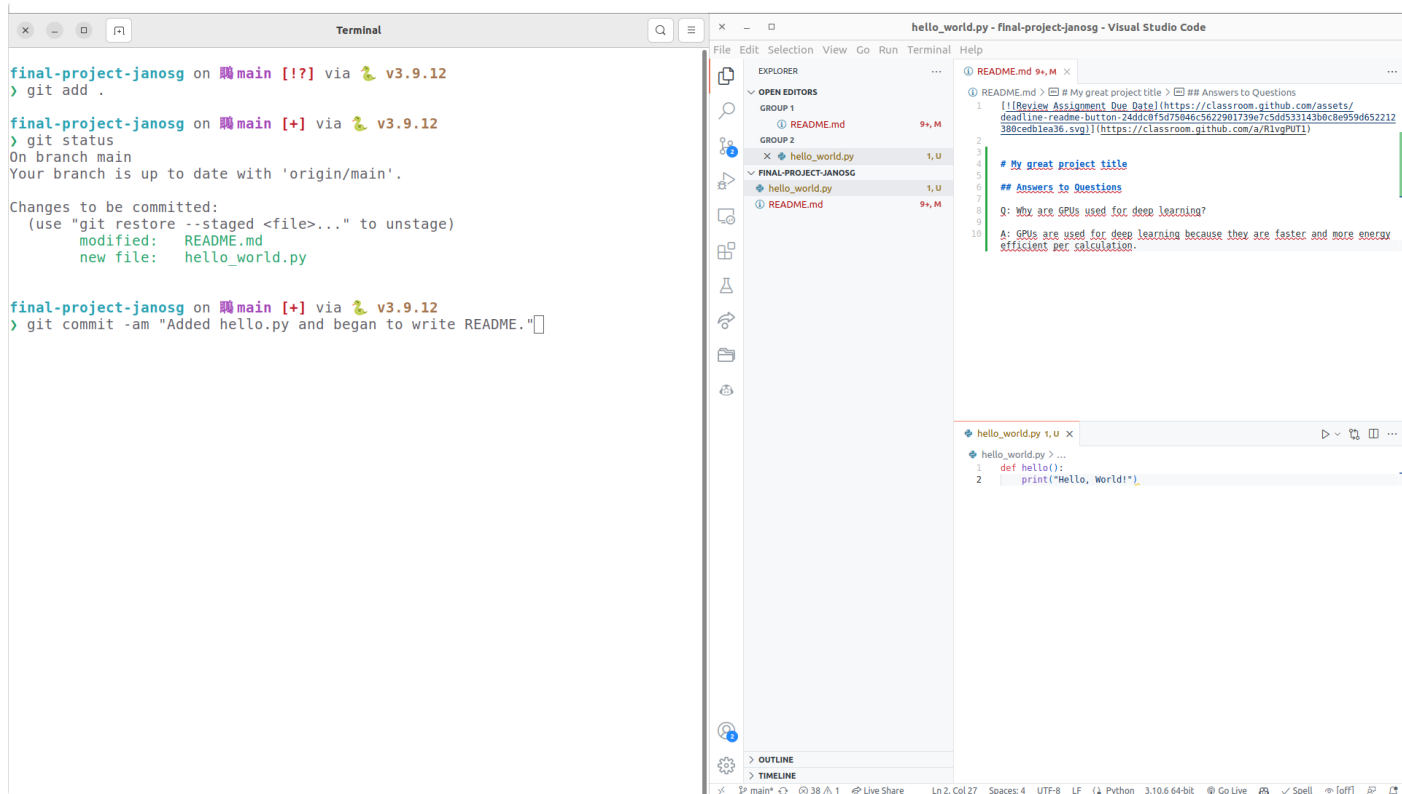
Visual Studio Code Editor View (README.md):

```
1  README.md > # My great project title > ## Answers to Questions
2  [!Re]view Assignment Due Date!(https://classroom.github.com/assets/
3  deadline-readme-button-24ddc0f5d75046c5622901739e7C5dd533143b0c8e959d652212
4  380cedb1ea36.svg)(https://classroom.github.com/a/R1vgPUT1)
5
6  # My great project title
7  ## Answers to Questions
8  Q: Why are GPUs used for deep learning?
9  A: GPUs are used for deep learning because they are faster and more energy
10 efficient per calculation.
```

Visual Studio Code Editor View (hello_world.py):

```
1  def hello():
2      print("Hello, World!")
```

Type commit command with message



The screenshot displays a terminal window and the Visual Studio Code editor interface. The terminal window on the left shows the execution of git commands: `git add .`, `git status`, and `git commit -am "Added hello.py and began to write README."`. The Visual Studio Code editor on the right shows the Explorer view with files `hello_world.py` and `README.md`. The editor also shows the content of `README.md` and `hello_world.py`.

```
final-project-janosg on main [!?] via v3.9.12
> git add .

final-project-janosg on main [+] via v3.9.12
> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
        new file:   hello_world.py

final-project-janosg on main [+] via v3.9.12
> git commit -am "Added hello.py and began to write README."
```

Visual Studio Code Explorer View:

- GROUP 1
 - README.md 9%, M
- GROUP 2
 - hello_world.py 1, U
- FINAL-PROJECT-JANOSG
 - hello_world.py 1, U
 - README.md 9%, M

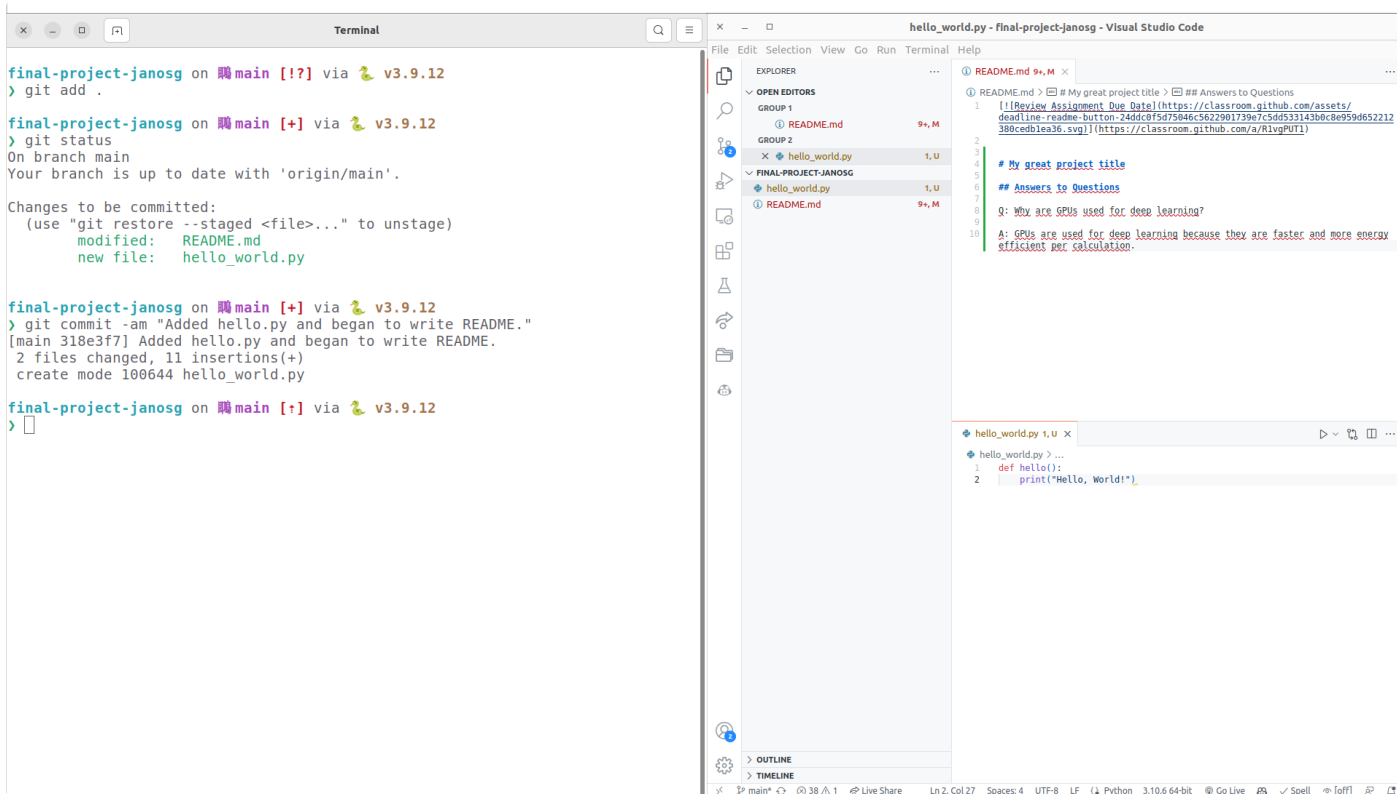
Visual Studio Code Editor View (README.md):

```
1 [[Review Assignment Due Date](https://classroom.github.com/assets/
2 deadline-readme-button-24ddc0f5d75b46c5622901739e7c5dd533143b0c8e959d652212
3 380ced1ea36.svg)](https://classroom.github.com/a/RivgPUt1)]
4
5 # My great project title
6
7 ## Answers to Questions
8
9 Q: Why are GPUs used for deep learning?
10 A: GPUs are used for deep learning because they are faster and more energy
    efficient per calculation.
```

Visual Studio Code Editor View (hello_world.py):

```
1 def hello():
2     print("Hello, World!")
```


Execute commit



The image shows a terminal window and a Visual Studio Code editor window. The terminal window displays the output of a series of git commands. The Visual Studio Code editor shows the Explorer view with a file tree containing 'README.md' and 'hello_world.py'. The editor also shows the content of 'README.md' and 'hello_world.py'.

```
final-project-janosg on main [!?] via v3.9.12
> git add .

final-project-janosg on main [+] via v3.9.12
> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
        new file:   hello_world.py

final-project-janosg on main [+] via v3.9.12
> git commit -am "Added hello.py and began to write README."
[main 318e3f7] Added hello.py and began to write README.
 2 files changed, 11 insertions(+)
 create mode 100644 hello_world.py

final-project-janosg on main [!] via v3.9.12
>
```

The Visual Studio Code editor window shows the Explorer view with a file tree containing 'README.md' and 'hello_world.py'. The editor also shows the content of 'README.md' and 'hello_world.py'.

README.md

```
1 [[Review Assignment Due Date](https://classroom.github.com/assets/
2 deadline-readme-button-24ddc0f5d75046c5622901739e7c5dd533143b0c8e959de652212
3 380cedb1ea36.svg)](https://classroom.github.com/a/R1vgPUT1)]
4
5 # My great project title
6
7 ## Answers to Questions
8
9 Q: Why are GPUs used for deep learning?
10 A: GPUs are used for deep learning because they are faster and more energy
    efficient per calculation.
```

hello_world.py

```
1 def hello():
2     print("Hello, World!")
```

Commit does not publish changes

The image illustrates a common misconception in Git: that committing changes automatically publishes them to a remote repository. The left side shows a terminal window where a user has successfully committed two files, `README.md` and `hello_world.py`, to the `main` branch. The right side shows the corresponding GitHub repository page for `final-project-janosg`. The repository page indicates that the commit was created 14 minutes ago, but the `README.md` file is not yet visible in the repository's file list, and the `Releases` and `Packages` sections show no published items, demonstrating that a local commit does not automatically push changes to the remote.

```
final-project-janosg on 主 main [!?] via 🐪 v3.9.12
> git add .

final-project-janosg on 主 main [+] via 🐪 v3.9.12
> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
        new file:   hello_world.py

final-project-janosg on 主 main [+] via 🐪 v3.9.12
> git commit -am "Added hello.py and began to write README."
[main 318e3f7] Added hello.py and began to write README.
2 files changed, 11 insertions(+)
create mode 100644 hello_world.py

final-project-janosg on 主 main [!:] via 🐪 v3.9.12
>
```

The GitHub repository page for `final-project-janosg` shows the following details:

- Repository: `final-project-janosg` (Private)
- Branches: `main`
- Files: `README.md` (Add assignment deadline url, 14 minutes ago)
- Activity: `final-project-janosg created by GitHub Classroom` (14 minutes ago)
- Releases: No releases published. [Create a new release](#)
- Packages: No packages published. [Publish your first package](#)

Footer: © 2023 GitHub, Inc.

Git push

```
final-project-janosg on main [!?] via v3.9.12
> git add .

final-project-janosg on main [+] via v3.9.12
> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
        new file:   hello_world.py

final-project-janosg on main [+] via v3.9.12
> git commit -am "Added hello.py and began to write README."
[main 318e3f7] Added hello.py and began to write README.
 2 files changed, 11 insertions(+)
 create mode 100644 hello_world.py

final-project-janosg on main [!] via v3.9.12
> git push
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 633 bytes | 633.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/iame-uni-bonn/final-project-janosg.git
 7b29418..318e3f7  main -> main

final-project-janosg on main via v3.9.12
> 
```

The screenshot shows a web browser displaying the GitHub repository page for 'final-project-janosg' by 'iame-uni-bonn'. The repository is private. The page shows the repository's main branch, 'main', with a commit history table. The commit history table shows two commits: 'janosg Added hello.py and began to write README.' (now) and 'janosg Added hello.py and began to write README.' (now). The repository has 0 stars, 1 watching, and 0 forks. The repository is created by 'iame-uni-bonn' and is part of the 'final-project-janosg' repository. The repository has a README file and a 'hello_world.py' file. The repository has a 'My great project title' and 'Answers to Questions' section. The repository has a 'Review the assignment due date' button. The repository has a 'Terms Privacy Security Status Docs Contact GitHub Pricing API Training Blog About' footer.

Search or jump to...

final-project-janosg / final-project-janosg Private

Edit Pins Unwatch 1 Fork Star 0

Code Issues Pull requests Actions Projects Wiki Security Insights

main Go to file Add file Code About

janosg Added hello.py and began to write README. now 2

README.md Added hello.py and began to write README. now

hello_world.py Added hello.py and began to write README. now

README.md

Review the assignment due date

My great project title

Answers to Questions

Q: Why are GPUs used for deep learning?

A: GPUs are used for deep learning because they are faster and more energy efficient per calculation.

final-project-janosg created by GitHub Classroom

Readme Activity 0 stars 1 watching 0 forks

Releases No releases published Create a new release

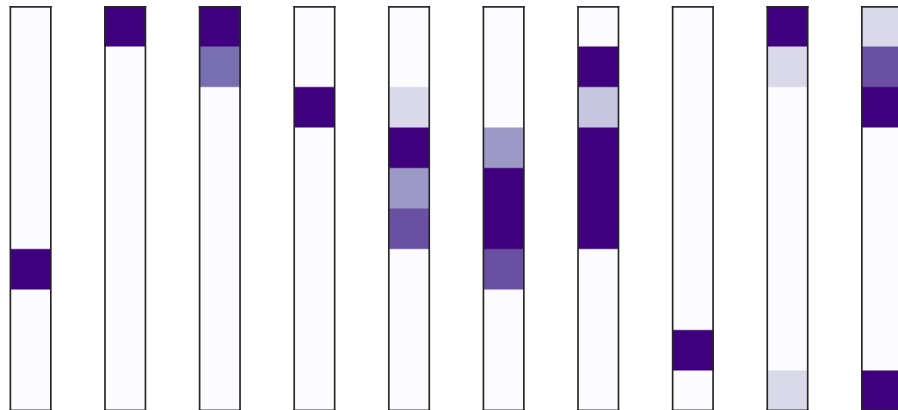
Packages No packages published Publish your first package

Terms Privacy Security Status Docs Contact GitHub Pricing API Training Blog About

© 2023 GitHub, Inc.

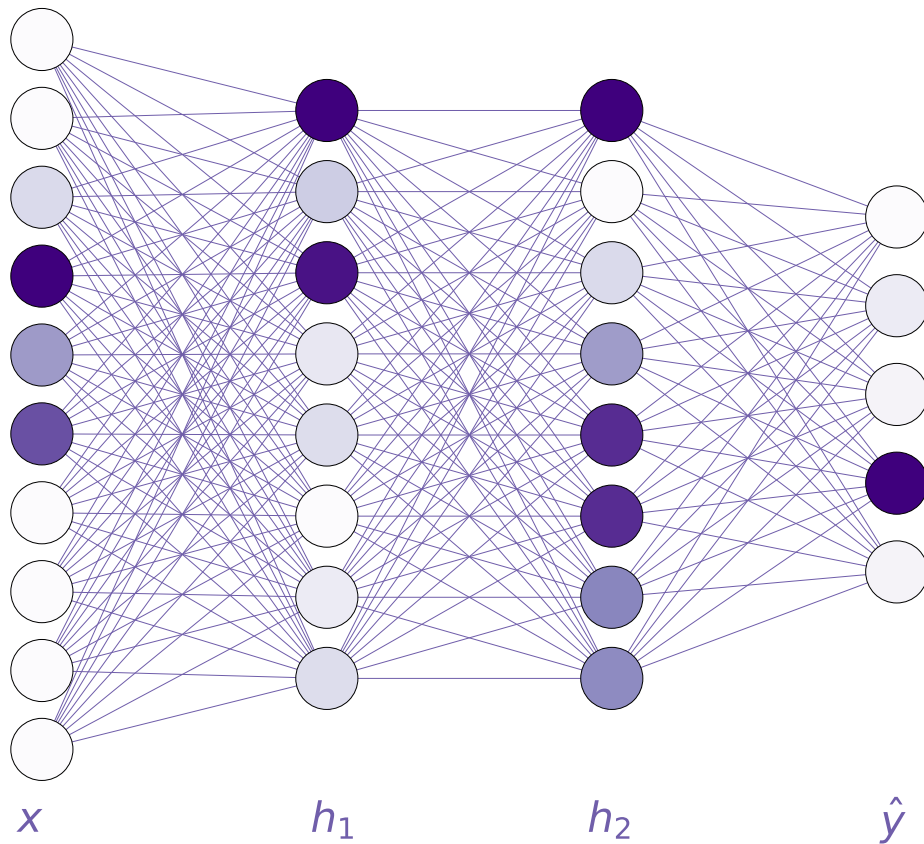
Recap: Feed-forward neural networks

Line length recognition



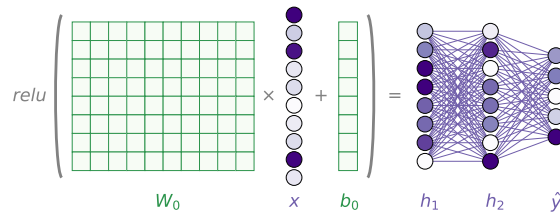
- Images of 10 x 1 Pixels with lines of length 1 to 5
- Task: Estimate the length of the line
- Can see correct result from flattened image

The Multi Layer Perceptron

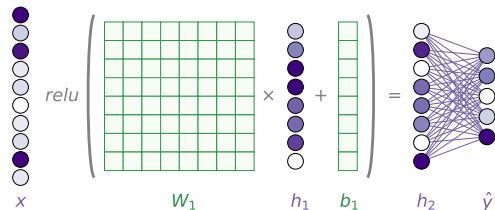


Written out matrix multiplications

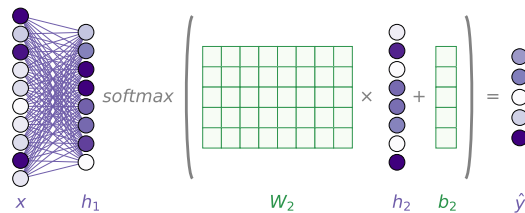
$$h_1 = \text{relu}(W_0 x + b_0)$$



$$h_2 = \text{relu}(W_1 h_1 + b_1)$$



$$\hat{y} = \text{softmax}(W_2 h_2 + b_2)$$



Why do we need nonlinearities

- Without nonlinearities, our model would be:

$$W_2(W_1(W_0x + b_0) + b_1) + b_2$$

- This could be simplified to:

$$Wx + b$$

- Thus we would end up with one linear model!

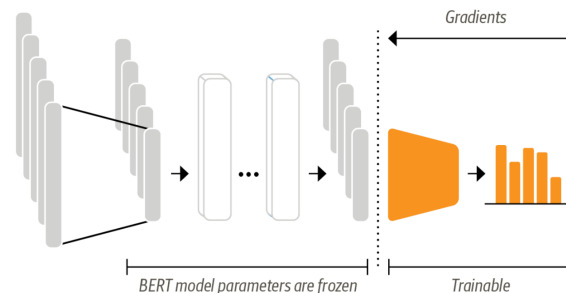
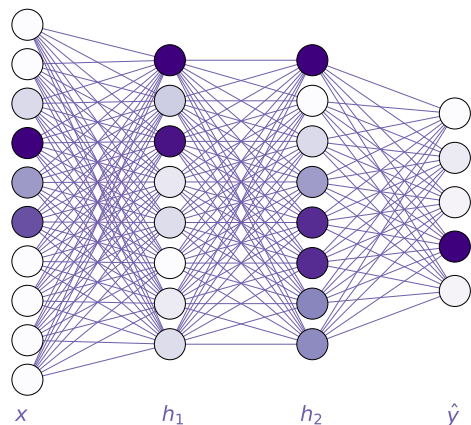
Trainable parameters

- Weights have shapes:
 - $n^{hidden} \times n^{in}, n^{hidden} \times n^{hidden}, n^{out} \times n^{hidden}$
- Biases have shapes:
 - $n^{hidden}, n^{hidden}, n^{out}$
- 205 for line lengths example
- 13 002 for digit recognition
- ~60 Million in the model we fine-tuned
- 175 Billion in GPT-3

Why are neural networks so powerful?

- Networks get their power from training!
- Done with some form of gradient descent
- Last week, we did it from scratch
- This week, we will look at simpler ways

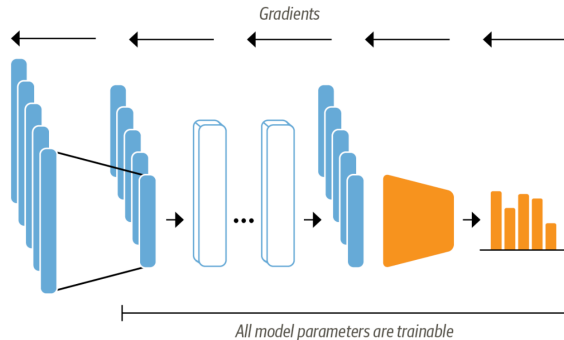
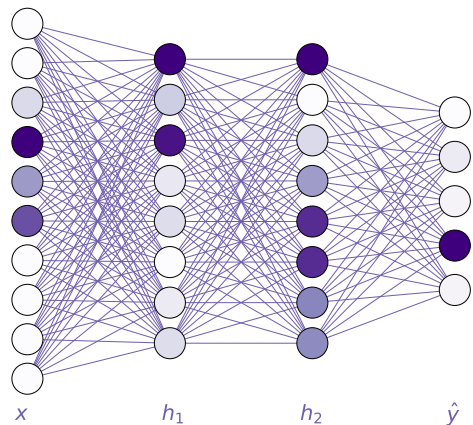
How does it relate to feature extraction



Source: Natural Language Processing with transformers, Fig 2-4

- Parameters are not initialized randomly but pre-trained
- Only W_2 and b_2 are specialized to classification task

How does it relate to fine-tuning

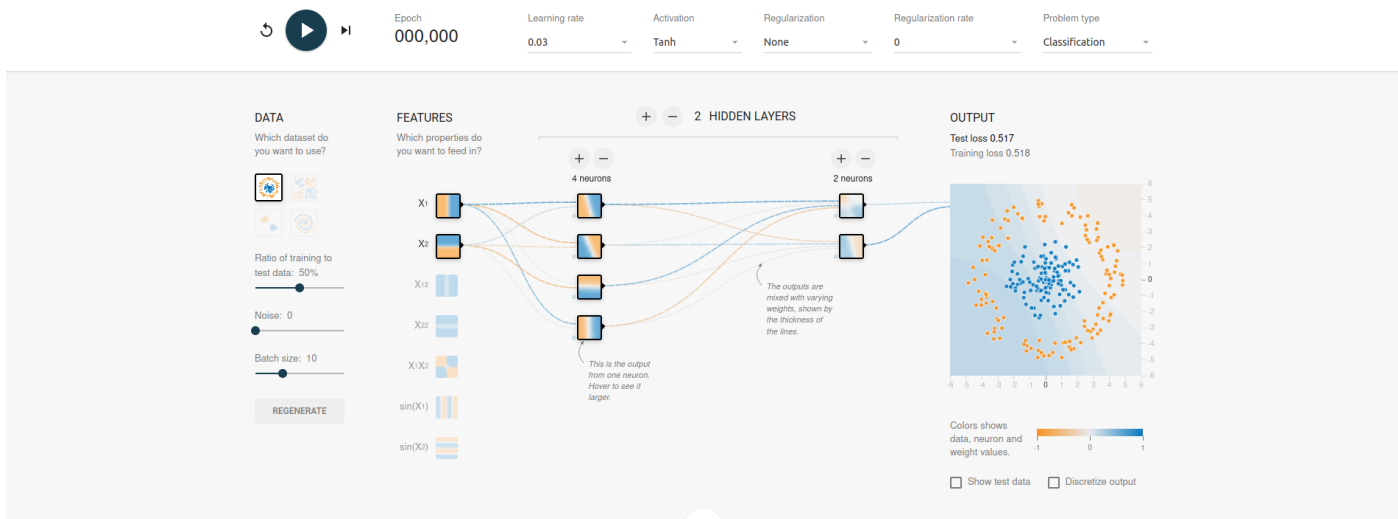


Source: Natural Language Processing with transformers, Fig 2-6

- Parameters are not initialized randomly but pre-trained
- All parameters are specialized to classification task

Tensorflow Playground

Tinker With a **Neural Network** Right Here in Your Browser.
Don't Worry, You Can't Break It. We Promise.



Use pytorch
properly

What did we do last week?

- Implement entire training process from scratch
 - The model
 - The optimizer
 - The loss function
 - The training loop
- Why did we do this?
 - Need to know the mechanics of a neural network
 - It will be easier to understand the built-in pytorch functions after you did the same steps from scratch

What was annoying last week?

- Had to initialize weights and biases with correct shape
- Several operations on each parameter tensor
 - Set `requires_grad` to `True`
 - (Put on GPU)
 - Update with gradients and `zero_` gradients
- Had to know the mechanics of a lot of functions
 - `relu`, `softmax`, `nll_loss`, gradient descent
- Had to think about numerical stability
- Our code was slow

Goal for this lecture: Simpler training

```
# training hyperparameters
n_epochs = 3
batch_size = 64
learning_rate = 0.01

# initialization
model = NeuralNetwork(n_in, n_hidden, n_out).to(device)
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
train_dataloader = DataLoader(
    training_data, batch_size=batch_size, shuffle=True, drop_last=True,
)
test_dataloader = DataLoader(
    test_data, batch_size=batch_size, drop_last=True,
)

# training loop
for t in range(n_epochs):
    print(f"Epoch {t+1}\n-----")
    train_loop(train_dataloader, model, loss_func, optimizer)
    test_loop(test_dataloader, model)
print("Done!")
```

Approach

- Good news
 - After today you can put pytorch on your CV
 - Some of the snippets you write can be re-used in other projects
- Bad news
 - You will have to learn quite a few new concepts
- Approach
 - I show you the final training loop at the beginning
 - You practice all the building blocks in exercises
 - (You don't have to write the final loop yourself today)

Steps

1. Defining a model
2. Define DataLoaders (should be called batch loaders)
3. Loss functions
4. Optimizers
5. Inner training loop
6. Inner test loop

Defining models

Purpose of the model

The pytorch model replaces two steps we did manually

- Initializing the parameters (and defining their shapes)
- Defining the calculations done in the model

Models are classes

- Defining a model = defining a class
- Class specifies:
 - Number, type and shape of layers (similar to defining the parameter matrices)
 - Forward method (similar to our model function)
- Class inherits other methods
 - Initialize all parameters
 - Put all parameters on a device
 - ...

Classes in Python

```
class Circle:
    def __init__(self, x, y, radius):
        self.x = x
        self.y = y
        self.radius = radius

    def area(self):
        return np.pi * self.radius ** 2

    def diameter(self):
        return 2 * self.radius

    def __repr__(self):
        return f"Circle at x={self.x}, y={self.y} with rad
```

```
circle = Circle(0, 0, 1)
circle
```

```
Circle at x=0, y=0 with radius 1
```

```
circle.area()
```

```
3.141592653589793
```

- Defined with `class` keyword
- Class = bundle of methods and data
- Methods are like functions but often have no arguments beyond the class attributes
- `__dunder__` methods are special
- Class != instance
 - Class: Blueprint
 - Instance: Has concrete values for attributes

The `__init__` method

```
class Circle:
    def __init__(self, x, y, radius):
        print("I was called")
        self.x = x
        self.y = y
        self.radius = radius
```

```
circle = Circle(0, 0, 1)
```

```
print("I was called")
```

- `__init__` sets up the model instance
- arguments of `__init__` become arguments of class
- Can execute arbitrary code here
- First argument is always `self`
- See `self` as a flexible data container where you can store and retrieve stuff and to which you have access in all methods

Methods

```
class Circle:
    def __init__(self, x, y, radius):
        self.x = x
        self.y = y
        self.radius = radius

    def area(self):
        return np.pi * self.radius ** 2
```

```
circle = Circle(0, 0, 1)
circle.area()
```

```
3.141592653589793
```

- Methods are defined like functions
- First argument is always ``self``
- Can have additional arguments
- When calling the method, ``self`` is passed automatically
- Inside methods you can do everything you can do inside functions

Anatomy of a Pytorch Model

```
class NeuralNetwork(nn.Module):  
    def __init__(self, n_in, n_hidden, n_out):  
        super().__init__()  
        ...  
  
    def forward(self, x):  
        ...  
        return logits
```

- Pytorch models are subclasses of `nn.Module`
- Have two mandatory methods:
 - `__init__` which calls the init method of it's superclass and defines shapes
 - `__forward__` Which does what our model function did

Writing the `__init__` method

```
class NeuralNetwork(nn.Module):
    def __init__(self, n_in, n_hidden, n_out):
        super().__init__()
        self.flatten = nn.Flatten()
        self.all_layers = nn.Sequential(
            nn.Linear(n_in, n_hidden),
            nn.ReLU(),
            nn.Linear(n_hidden, n_out),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.all_layers(x)
        return logits
```

- Similar model as last week but only one hidden layer
- Always call `super().__init__()`
- Assign functions we need later to `self`
 - `flatten`: Go from (28x28) to (784)
 - `all_layers`: Model calculations
- Use built-in pytorch functions
 - `nn.Sequential` chains functions
 - `Linear` and `ReLU`
- Each layer knows how many parameters it needs and registers them with the class

Writing the forward function

```
class NeuralNetwork(nn.Module):
    def __init__(self, n_in, n_hidden, n_out):
        super().__init__()
        self.flatten = nn.Flatten()
        self.all_layers = nn.Sequential(
            nn.Linear(n_in, n_hidden),
            nn.ReLU(),
            nn.Linear(n_hidden, n_out),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.all_layers(x)
        return logits
```

- Forward does the actual calculation
- Mainly calls functions that were assigned to self before
- Should return logits, i.e. not take softmax yet

Using the model

```
n_in = 28 * 28
n_hidden = 16
n_out = 10
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = NeuralNetwork(n_in, n_hidden, n_out).to(device)
model
```

```
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (all_layers): Sequential(
    (0): Linear(in_features=784, out_features=16, bias=True)
    (1): ReLU()
    (2): Linear(in_features=16, out_features=10, bias=True)
  )
)
```

- The model class worked for general shapes, the instance has specific ones
- Model has a printable and human readable text summary
- Can put all tensors belonging to the model on the GPU with one method

Looking at model parameters

```
list(model.parameters())
```

```
[Parameter containing:
  tensor([[ 0.0057, ..., -0.0129],
          ...,
          [ 0.0142, ..., -0.0051]],
        requires_grad=True),
 Parameter containing:
  tensor([-0.0002, ..., -0.0232],
        requires_grad=True),
 ...,
]
```

- Each layer knew which parameters it needs
- Registered them automatically with the model
- They get initialized randomly behind the scenes
- Optimizers will work on this parameter list
- You don't ever have to look at them

What else is in `torch.nn`?

- Many different nonlinearities
- Many different layers
 - Convolutional layers
 - Recurrent layers
 - Transformers
- Loss functions
- See the documentation for details

Task 1

DataLoaders

Purpose of the DataLoaders

The DataLoader serves two purposes

- Make it easy to loop over batches of the data
- (Make it possible to load data in parallel)

How we did it last time

```
batch_indices = torch.randperm(len(data)).reshape(-1, batch_size)
for idxs in batch_indices:
    batch = data[idxs]
```

- Did not work if dataset size was not a multiple of batch size
- Hard to read if you don't know the trick

DataLoaders

```
batch_size = 64
train_dataloader = DataLoader(
    training_data,
    batch_size=batch_size,
    shuffle=True,
    drop_last=True,
)

for i, (X, y) in enumerate(train_dataloader):
    logits = model(X)
    ...
```

```
len(train_dataloader)
```

```
937
```

```
len(train_dataloader.dataset)
```

```
60000
```

- the dataloader is something you can iterate over
- mechanics of shuffling and batching are abstracted away
- Enable or disable shuffling
- `drop_last` is how we handle dataset length that are not multiples of batch size

Task 2

Loss functions

What we did last week

```
def nll_loss(probs, labels):  
    likelihoods = probs[torch.arange(len(probs)), labels] + 1e-50  
    loglikes = torch.log(likelihoods)  
    return -loglikes.mean()
```

- Needed to bother with the mechanics of indexing
- Very crude way of ensuring numerical stability

Pre-implemented loss function

Docs > torch.nn

0-2

Loss Functions

<code>nn.L1Loss</code>	Creates a criterion that measures the mean absolute error (MAE) between each element in the input x and target y .
<code>nn.MSELoss</code>	Creates a criterion that measures the mean squared error (squared L2 norm) between each element in the input x and target y .
<code>nn.CrossEntropyLoss</code>	This criterion computes the cross entropy loss between input logits and target.
<code>nn.CTCLoss</code>	The Connectionist Temporal Classification loss.
<code>nn.NLLLoss</code>	The negative log likelihood loss.
<code>nn.PoissonNLLLoss</code>	Negative log likelihood loss with Poisson distribution of target.
<code>nn.GaussianNLLLoss</code>	Gaussian negative log likelihood loss.
<code>nn.KLDivLoss</code>	The Kullback-Leibler divergence loss.
<code>nn.BCELoss</code>	Creates a criterion that measures the Binary Cross Entropy between the target and the input probabilities:
<code>nn.BCEWithLogitsLoss</code>	This loss combines a <i>Sigmoid</i> layer and the <i>BCELoss</i> in one single class.
<code>nn.MarginRankingLoss</code>	Creates a criterion that measures the loss given inputs x_1 , x_2 , two 1D mini-batch or 0D Tensors, and a label 1D mini-batch or 0D Tensor y (containing 1 or -1).
<code>nn.HingeEmbeddingLoss</code>	Measures the loss given an input tensor x and a labels tensor y (containing 1 or -1).

- Many loss functions are pre-implemented
- Numerically stable implementations
- We will use `CrossEntropyLoss`
- See the documentation for details

```
loss_func = nn.CrossEntropyLoss()  
loss_func(logits, y)
```

```
tensor(2.3018, grad_fn=<NllLossBackward0>)
```


Task 3

Optimizers

What we did last week

```
for i in range(3):  
    # SGD updates for each parameter  
    weights[i].data = weights[i].data - learning_rate * weights[i].grad.data  
    biases[i].data = biases[i].data - learning_rate * biases[i].grad.data  
    # Zero the gradients for the next iteration  
    weights[i].grad.data.zero_()  
    biases[i].grad.data.zero_()
```

- Had to know the gradient descent update equation
- Had to do things per-parameter
- Had to know when to use `tensor` and `tensor.data`

Optimizers in pytorch

```
learning_rate = 0.1  
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

In the training loop:

```
...  
loss.backward()  
optimizer.step()  
optimizer.zero_grad()
```

- Optimizer instance knows which parameter tensors are there
- Automatically applies relevant steps to all of them
- Many optimizers available

The optimization problem

- $\theta \in \mathcal{R}^d$ is a vector of parameters
- $Z \in \mathcal{R}^{n \times m}$ is a matrix containing a batch of data (x and y)
- $\ell(\theta, Z)$ is a scalar loss function
- $j(\theta, Z)$ is the gradient of ℓ w.r.t. θ
- Goal: $\min_{\theta} \ell(\theta)$
- In words: find parameters of the neural net that minimize the loss function

SGD

- Tuning parameters:
 - η : learning rate, typically 1e-3
- Update equation
 - $\theta_{k+1} = \theta_k - \eta \cdot j(\theta_k, Z)$
- Problems
 - Get's stuck in local optima (gradient = 0)
 - Learning rate is the same for all parameters
 - Learning rate is hart to pick

SGD + Momentum

- Tuning parameters:
 - η : learning rate, typically $1e-3$
 - γ : momentum parameter, typically 0.9
- Update equations:
 - $\nu_k = \gamma \nu_{k-1} + \eta \cdot j(\theta_k, Z)$
 - $\theta_{k+1} = \theta_k - \nu_k$
- Advantage
 - Can pass over a local flat spot
 - Less oscillation around the main direction of progress

Adam (Adaptive Moment Estimation)

- Tuning parameters
 - η : learning rate, typically $1e-3$
 - β_1 : momentum in gradients, typically 0.9
 - β_2 : momentum in squared gradients, typically 0.99
 - ϵ : clipping value, typically $1e-8$
- Update equations
 - $m_k = \frac{1}{1-\beta_1} \cdot [\beta_1 m_{k-1} + (1 - \beta_1)j(\theta_k, Z)]$
 - $v_k = \frac{1}{1-\beta_2} \cdot [\beta_2 v_{k-1} + (1 - \beta_2)j(\theta_k, Z)^2]$
 - $\theta_{k+1} = \theta_k - \frac{\eta}{\sqrt{v_k} + \epsilon} m_k$

Adam

- Adam behaves like a heavy ball with friction, which thus prefers flat minima in the error surface
- Widely used in practice (e.g. for GPT)
- Often together with a learning rate schedule, where η decays over time

Task 4

Inner training loop

Remember the goal

```
# training hyperparameters
n_epochs = 3
batch_size = 64
learning_rate = 0.01

# initialization
model = NeuralNetwork(n_in, n_hidden, n_out).to(device)
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
train_dataloader = DataLoader(
    training_data, batch_size=batch_size, shuffle=True, drop_last=True,
)
test_dataloader = DataLoader(
    test_data, batch_size=batch_size, drop_last=True,
)

# training loop
for t in range(n_epochs):
    print(f"Epoch {t+1}\n-----")
    train_loop(train_dataloader, model, loss_func, optimizer)
    test_loop(test_dataloader, model)
print("Done!")
```

The inner `train_loop`

```
def train_loop(dataloader, model, loss_fn, optimizer):  
    model.train()  
  
    # loop over the dataloader  
    # evaluate model  
    # evaluate loss  
    # backpropagate  
    # call optimizer step  
    # zero the gradients
```

- `model.train()` Puts the model in training mode
- Best practice: Do it for any model, even if it does not make a difference
- Some models have layers that behave differently during training and inference

Task 5

Remember the goal

```
# training hyperparameters
n_epochs = 3
batch_size = 64
learning_rate = 0.01

# initialization
model = NeuralNetwork(n_in, n_hidden, n_out).to(device)
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
train_dataloader = DataLoader(
    training_data, batch_size=batch_size, shuffle=True, drop_last=True,
)
test_dataloader = DataLoader(
    test_data, batch_size=batch_size, drop_last=True,
)

# training loop
for t in range(n_epochs):
    print(f"Epoch {t+1}\n-----")
    train_loop(train_dataloader, model, loss_func, optimizer)
    test_loop(test_dataloader, model)
print("Done!")
```

Inner test loop

```
def test_loop(dataloader, model):  
    model.eval()  
  
    correct = 0  
    # use torch.no_grad()  
    # loop over data  
        # calculate predictions  
        # add te number of correct predictions to corr  
    # calculate accuracy  
    # print accuracy
```

- Want to have similar monitory as last time
- Print accuracy on the test data after each epoch

Task 6

Task 7

Summary

- We solved the same problem as last week but this time used pytask properly
- Not less code, but it scales to larger model
- Using the ingredients you learned today, you can build large neural networks and train them on CPU or GPU
- There are many different optimizers
- If in doubt, use Adam