

Please follow our [blog](#) to see more information about new features, tips and tricks, and featured notebooks such as [Analyzing a Bank Failure with Colab](#).

2024-11-11

- Users can now import Gemini API keys from AI Studio into their user secrets, all in Colab ([tweet](#)).
- Increased limit to 1000 characters for requests to Gemini in Chat and Generate windows.
- Improved saving notebook to GitHub flow.
- Updated Gemini spark icon to be colorful.
- [uv](#) is pre-installed on the PATH for faster package installs.
- Fixed bugs
 - Dropdown text for GitHub repository not visible [#4901](#).
 - Pre-installed California housing dataset README not correct [#4862](#).
 - Backend execution error for scheduled notebook [#4850](#).
 - Drive File Stream issues [#3441](#).
 - Linking to the signup page does not preserve the authuser parameter.
 - Error messages in Gemini chat are not polished.
 - Clicking in Gemini chat feedback causes jitters the UI.
 - Hovering over a table of contents entry would show the menu icons for all entries.
 - Surveys display over open dialogs.
 - Playground mode banner not shown on mobile.

Python package upgrades

- accelerate 0.34.2 -> 1.1.1
- arviz 0.19.0 -> 0.20.0
- bigframes 1.18.0 -> 1.25.0
- bigquery-magics 0.2.0 -> 0.4.0
- bokeh 3.4.3 -> 3.6.1
- blosc 2.0.0 -> 2.7.1
- cloudpickle 2.2.1 -> 3.1.0
- cudf-cu12 24.4.1 -> 24.10.1
- dask 2024.8.0 -> 24.10.0
- debugpy 1.6.6 -> 1.8.0
- earthengine-api 1.0.0 -> 1.2.0
- folium 0.17.0 -> 0.18.0
- gscfs 2024.6.1 -> 2024.10.0
- geemap 0.34.3 -> 0.35.1
- holidays 0.57 -> 0.60
- huggingface-hub 0.24.7 -> 0.26.2
- kagglehub 0.3.0 -> 0.3.3
- lightgbm 4.4.0 -> 4.5.0
- lxml 4.9.4 -> 5.3.0
- matplotlib 3.7.1 -> 3.8.0
- mizani 0.11.4 -> 0.13.0
- networkx 3.3 -> 3.4.2
- nltk 3.8.1 -> 3.9.1
- pandas 2.1.4 -> 2.2.2
- pillow 10.4.0 -> 11.0.0
- plotnine 0.13.6 -> 0.14.1
- polars 1.6.0 -> 1.9.0
- protobuf 3.20.3 -> 4.25.5
- pyarrow 14.0.2 -> 17.0.0
- pydrive2 1.20.0 -> 1.21.1
- pymc 5.16.2 -> 5.18.0
- torch 2.4.1 -> 2.5.0
- torchaudio 2.4.1 -> 2.5.0
- torchvision 0.19.1 -> 0.20.0
- transformers 4.44.2 -> 4.46.2
- xarray 2024.9.0 -> 2024.10.0

Python package inclusions

- diffusers 0.31.0
- gitpython 3.1.43
- langchain 0.3.7
- openai 1.54.3
- pygit2 1.16.0
- pyspark 3.5.3
- sentence-transformers 3.2.1
- timm 1.0.11
- wandb 0.18.6

Library and driver upgrades

- drivers upgraded from 89.0.2 to 98.0.0

Start by importing necessary packages

You will begin by importing necessary libraries for this notebook. Run the cell below to do so.

PyTorch and Intro to Training

```
!pip install thop
import math
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
import thop
import matplotlib.pyplot as plt
from tqdm import tqdm
import time
```



Collecting thop

```
Downloading thop-0.1.1.post2209072238-py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: Jinja2 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: mpmath<1.4, >=1.1.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages
Downloading thop-0.1.1.post2209072238-py3-none-any.whl (15 kB)
Installing collected packages: thop
Successfully installed thop-0.1.1.post2209072238
```

Checking the torch version and CUDA access

Let's start off by checking the current torch version, and whether you have CUDA availability.

```
print("torch is using version:", torch.__version__, "with CUDA=", torch.cuda.is_available())
```



```
torch is using version: 2.5.1+cu121 with CUDA= True
```

By default, you will see CUDA = False, meaning that the Colab session does not have access to a GPU. To remedy this, click the Runtime menu on top and select "Change Runtime Type", then select "T4 GPU".

Re-run the import cell above, and the CUDA version / check. It should show now CUDA = True

Sometimes in Colab you get a message that your Session has crashed, if that happens you need to go to the Runtime menu on top and select "Restart session".

You won't be using the GPU just yet, but this prepares the instance for when you will.

Please note that the GPU is a scarce resource which may not be available at all time.

Additionally, there are also usage limits that you may run into (although not likely for this assignment). When that happens you need to try again later/next day/different time of the day. Another reason to start the assignment early!

A Brief Introduction to PyTorch

PyTorch, or torch, is a machine learning framework developed by Facebook AI Research, which competes with TensorFlow, JAX, Caffe and others.

Roughly speaking, these frameworks can be split into dynamic and static definition frameworks.

Static Network Definition: The architecture and computation flow are defined simultaneously. The order and manner in which data flows through the layers are fixed upon definition. These frameworks also tend to declare parameter shapes implicitly via the compute graph. This is typical of TensorFlow and JAX.

Dynamic Network Definition: The architecture (layers/modules) is defined independently of the computation flow, often during the object's initialization. This allows for dynamic computation graphs where the flow of data can change during runtime based on conditions. Since the network exists independent of the compute graph, the parameter shapes must be declared explicitly. PyTorch follows this approach.

All ML frameworks support automatic differentiation, which is necessary to train a model (i.e. perform back propagation).

Let's consider a typical pytorch module. Such modules will inherit from the `torch.nn.Module` class, which provides many built in functions such as a wrapper for `__call__`, operations to move the module between devices (e.g. `cuda()`, `cpu()`), data-type conversion (e.g. `half()`, `float()`), and parameter and child management (e.g. `state_dict()`, `parameters()`).

```
# inherit from torch.nn.Module
class MyModule(nn.Module):
    # constructor called upon creation
    def __init__(self):
        # the module has to initialize the parent first, which is what sets up the wrapper
        super().__init__()

        # we can add sub-modules and parameters by assigning them to self
        self.my_param = nn.Parameter(torch.zeros(4,8)) # this is how you define a raw parameter
        self.my_sub_module = nn.Linear(8,12) # this is how you define a linear layer

        # we can also add lists of modules, for example, the sequential layer
        self.net = nn.Sequential( # this layer type takes in a collection of modules and returns a single output
            nn.Linear(4,4),
            nn.Linear(4,8),
            nn.Linear(8,12)
        )

        # the above when calling self.net(x), will execute each module in the order they are defined
        # it would be equivalent to x = self.net[2](self.net[1](self.net[0](x)))

        # you can also create a list that doesn't execute
        self.net_list = nn.ModuleList([
            nn.Linear(7,7),
            nn.Linear(7,9),
            nn.Linear(9,14)
        ])

        # sometimes you will also see constant variables added to the module post init
        foo = torch.Tensor([4])
        self.register_buffer('foo', foo) # buffers allow .to(device, type) to apply

# let's define a forward function, which gets executed when calling the module, as follows
def forward(self, x):

    # if x is of shape Bx4
    h1 = x @ self.my_param # tensor-tensor multiplication uses the @ symbol
    # then h1 is now shape Bx8, because my_param is 4x8... 2x4 * 4x8 = 2x8

    h1 = self.my_sub_module(h1) # you execute a sub-module by calling it
    # now, h1 is of shape Bx12, because my_sub_module was a 8x12 matrix

    h2 = self.net(x)
    # similarly, h2 is of shape Bx12, because that's the output of the sequence
    # Bx4 -(4x4)-> Bx4 -(4x8)-> Bx8 -(8x12)-> Bx12

    # since h1 and h2 are the same shape, they can be added together element-wise
    return h1 + h2
```

Then you can instantiate the module and perform a forward pass by calling it.

2024-09-23

- Improved code snippet search
- Updated Marketplace image and public local runtime container
- Improved the look-and-feel of interactive form dropdowns and checkboxes
- Fixed bugs
 - activating the skip link caused the notebook to scroll out of view
 - toggleing a checkbox too much caused the page to crash
 - lightning fast drags could cause orphaned tabs
 - custom widgets snippet would show for local runtimes

Python package upgrades

- accelerate 0.32.1 -> 0.34.2
- arviz 0.18.0 -> 0.19
- autograd 1.6.2 -> 1.7.0
- bigframes 1.14.0 -> 1.18.0
- dask 2024.7.1 -> 2024.8.0
- distributed 2024.7.1 -> 2024.8.0
- duckdb 0.10.3 -> 1.1.0
- earthengine-api 0.1.416 -> 1.0.0
- flax 0.8.4 -> 0.8.5
- gdown 5.1.0 -> 5.2.0
- geemap 0.33.1 -> 0.34.3
- geopandas 0.14.4 -> 1.0.1
- google-cloud-aiplatform 1.59.0 -> 1.67.1
- google-cloud-bigquery-storage 2.25.0 -> 2.26.0
- holidays 0.54 -> 0.57
- huggingface-hub 0.23.5 -> 0.24.7
- ibis-framework 8.0.0 -> 9.2.0
- jax 0.4.26 -> 0.4.33
- jaxlib 0.4.26 -> 0.4.33
- kagglehub 0.2.9 -> 0.3.0
- lightgbm 4.4.0 -> 4.5.0
- matplotlib-venn 0.11.10 -> 1.1.1
- mizani 0.9.3 -> 0.11.4
- Pillow 9.4.0 -> 10.4.0
- plotly 5.15.0 -> 5.24.1
- plotnine 0.12.4 -> 0.13.6
- polars 0.20.2 -> 1.6.0
- progressbar2 4.2.0 -> 4.5.0
- PyDrive2 1.6.3 -> 1.20.0
- pymc 5.10.4 -> 5.16.2
- pytensor 2.18.6 -> 2.25.4
- scikit-image 0.23.2 -> 0.24.0
- scikit-learn 1.3.2 -> 1.5.2
- torch 2.3.1 -> 2.4.1
- torchaudio 2.3.1 -> 2.4.1
- torchvision 0.18.1 -> 0.19.1
- transformers 4.42.4 -> 4.44.2
- urllib3 2.0.7 -> 2.2.3
- xarray 2024.6.0 -> 2024.9.0

Python package inclusions

- bigquery-magics 0.2.0

2024-08-20

- TPU memory usage and utilization can now be checked with `!tpu-info`
- Gemini Chat responses are now grounded in relevant sources
- Added a new "Create Gemini API key" link in the user secrets panel
- Added a new "Gemini: Creating a prompt" snippet and touched up the existing "Gemini: Connecting to Gemini" snippet
- Added the ability to specify custom placeholder text for various interactive form params (see [examples](#))
- Keyboard navigation a11y improvements to comments UI
- Various minor rendering improvements to interactive forms UI
- A11y improvements for the run button and header
- Updated tooltip styling
- A11y improvements for the file browser's disk usage bar
- On mobile, tooltips now trigger on long press

```
# create the module
module = MyModule()

# you can print the module to get a high-level summary of it
print("=== printing the module ===")
print(module)
print()
# notice that the sub-module name is in parenthesis, and so are the list indicies

# let's view the shape of one of the weight tensors
print("my_sub_module weight tensor shape=", module.my_sub_module.weight.shape)
# the above works because nn.Linear has a member called .weight and .bias
# to view the shape of my_param, you would use module.my_param
# and to view the shape of the 2nd elment in net_list, you would use module.net_list

# you can iterate through all of the parameters via the state dict
print()
print("=== Listing parameters from the state_dict ===")
for key,value in module.state_dict().items():
    print(f"{key}: {value.shape}")
```

```
=== printing the module ===
MyModule(
  (my_sub_module): Linear(in_features=8, out_features=12, bias=True)
  (net): Sequential(
    (0): Linear(in_features=4, out_features=4, bias=True)
    (1): Linear(in_features=4, out_features=8, bias=True)
    (2): Linear(in_features=8, out_features=12, bias=True)
  )
  (net_list): ModuleList(
    (0): Linear(in_features=7, out_features=7, bias=True)
    (1): Linear(in_features=7, out_features=9, bias=True)
    (2): Linear(in_features=9, out_features=14, bias=True)
  )
)

my_sub_module weight tensor shape= torch.Size([12, 8])

=== Listing parameters from the state_dict ===
my_param: torch.Size([4, 8])
foo: torch.Size([1])
my_sub_module.weight: torch.Size([12, 8])
my_sub_module.bias: torch.Size([12])
net.0.weight: torch.Size([4, 4])
net.0.bias: torch.Size([4])
net.1.weight: torch.Size([8, 4])
net.1.bias: torch.Size([8])
net.2.weight: torch.Size([12, 8])
net.2.bias: torch.Size([12])
net_list.0.weight: torch.Size([7, 7])
net_list.0.bias: torch.Size([7])
net_list.1.weight: torch.Size([9, 7])
net_list.1.bias: torch.Size([9])
net_list.2.weight: torch.Size([14, 9])
net_list.2.bias: torch.Size([14])
```

```
# you can perform a forward pass by first creating a tensor to send through
x = torch.zeros(2,4)
# then you call the module (this invokes MyModule.forward() )
y = module(x)

# then you can print the result and shape
print(x, x.shape)
print(y, y.shape)

tensor([[0., 0., 0., 0.],
        [0., 0., 0., 0.]]) torch.Size([2, 4])
tensor([[[-0.0946, -0.2721, -0.0908, -0.9849, -0.1707, -0.0207, -0.1869, -0.0110,
          0.2822, 0.0379, -0.1346, -0.2241],
        [-0.0946, -0.2721, -0.0908, -0.9849, -0.1707, -0.0207, -0.1869, -0.0110,
          0.2822, 0.0379, -0.1346, -0.2241]], grad_fn=<AddBackward0>]) torch.Si
```

- On mobile, release notes updates will no longer display automatically
- Python package upgrades
 - astropy 5.3.4 -> 6.1.2
 - bigframes 1.11.1 -> 1.14.0
 - bokeh 3.3.4 -> 3.4.3
 - dask 2023.8.1 -> 2024.7.1
 - earthengine-api 0.1.412 -> 0.1.416
 - geopandas 0.13.2 -> 0.14.4
 - kagglehub 0.2.8 -> 0.2.9
 - keras 2.15.0 -> 3.4.1
 - lightgbm 4.1.0 -> 4.4.0
 - malloy 2023.1067 -> 2024.1067
 - numba 0.58.1 -> 0.60.0
 - numpy 1.25.2 -> 1.26.4
 - opencv-python 4.8.0.76 -> 4.10.0.84
 - pandas 2.0.3 -> 2.1.4
 - pandas-gbq 0.19.2 -> 0.23.1
 - panel 1.3.8 -> 1.4.5
 - requests 2.31.0 -> 2.32.3
 - scikit-learn 1.2.2 -> 1.3.2
 - scipy 1.11.4 -> 1.13.1
 - tensorboard 2.15.2 -> 2.17.0
 - tensorflow 2.15.0 -> 2.17.0
 - tf-keras 2.15.1 -> 2.17.0
 - xarray 2023.7.0 -> 2024.6.0
 - xgboost 2.0.3 -> 2.1.1
- Python package inclusions
 - einops 0.8.0

2024-07-22

- You can now embed Google sheets directly into Colab to streamline interactions with data with InteractiveSheet.
Example:
from google.colab import sheets
sh = sheets.InteractiveSheet()
df = sh.as_df()
- Fixed multiple rendering bugs in cell editors with wide text content (i.e. text is no longer hidden or clipped)
- Fixed multiple accessibility issues in Colab's comments feature (e.g. proper keyboard focus management, added accessibility landmarks, etc)
- Fixed bug where AI code generation would fail for extremely long broken code snippets
- Fixed multiple scrollbar bugs in the user secrets panel
- Added the ability for workspace admin to purchase Colab Pro and Pro+ Subscriptions for users
- Fixed bug where user secrets couldn't be moved to a tab
- Fixed several focus management accessibility issues in tabs, the table of contents, the left toolbar, and the run button
- Fixed bug where overflowing cells may be omitted when pasting from Google Sheets
- Fixed bug where the generate code button did not activate on touch
- Python package upgrades
 - bigframes 1.9.0 -> 1.11.1
 - cvxpy 1.3.4 -> 1.5.2
 - earthengine-api 0.1.408 -> 0.1.412
 - google-api-core 2.11.1 -> 2.19.1
 - google-api-python-client 2.84.0 -> 2.137.0
 - google-cloud-aiplatform 1.56.0 -> 1.59.0
 - google-cloud-bigquery 3.21.0 -> 3.25.0
 - google-cloud-core 2.3.3 -> 2.4.1
 - google-cloud-datastore 2.15.2 -> 2.19.0
 - google-cloud-firestore 2.11.1 -> 2.16.1
 - google-cloud-functions 1.13.3 -> 1.16.4
 - google-generativeai 0.5.4 -> 0.7.2
 - kagglehub 0.2.5 -> 0.2.8
 - pip 23.1.2 -> 24.1.2
 - setuptools 67.7.2 -> 71.0.4
 - sympy 1.12.1 -> 1.13.1
 - torch 2.3.0 -> 2.3.1
 - transformers 4.41.2 -> 4.42.4
- Python package inclusions

Please check the cell below to notice the following:

1. ☒ above was created with the shape 2x4, and in the forward pass, it gets manipulated into a 2x12 tensor. This last dimension is explicit, while the first is called the batch dimension, and only exists on data (a.k.a. activations). The output shape can be seen in the print statement from y.shape

2. You can view the shape of a tensor by using `.shape`, this is a very helpful trick for debugging tensor shape errors
3. In the output, there's a `grad_fn` component, this is the hook created by the forward trace to be used in back-propagation via automatic differentiation. The function name is `AddBackward`, because the last operation performed was `h1+h2`.

We might not always want to trace the compute graph though, such as during inference. In such cases, you can use the `torch.no_grad()` context manager.

```
# you can perform a forward pass by first creating a tensor to send through
x = torch.zeros(2,4)
# then you call the module (this invokes MyModule.forward() )
with torch.no_grad():
    y = module(x)

# then you can print the result and shape
print(y, y.shape)
# notice how the grad_fn is no longer part of the output tensor, that's because not_
tensor([[ -0.0946, -0.2721, -0.0908, -0.9849, -0.1707, -0.0207, -0.1869, -0.0110,
          0.2822,  0.0379, -0.1346, -0.2241],
        [-0.0946, -0.2721, -0.0908, -0.9849, -0.1707, -0.0207, -0.1869, -0.0110,
          0.2822,  0.0379, -0.1346, -0.2241]]) torch.Size([2, 12])
```

Aside from passing a tensor through a model with the `no_grad()` context, you can also detach a tensor from the compute graph by calling `.detach()`. This will effectively make a copy of the original tensor, which allows it to be converted to numpy and visualized with matplotlib.

Note: Tensors with a `grad_fn` property cannot be plotted and must first be detached.

✓ Multi-Layer-Perceptron (MLP) Prediction of MNIST

With some basics out of the way, let's create a MLP for training MNIST. You can start by defining a simple torch model.

```
# Define the MLP model
class MLP(nn.Module):
    # define the constructor for the network
    def __init__(self):
        super().__init__()
        # the input projection layer - projects into d=128
        self.fc1 = nn.Linear(28*28, 128)
        # the first hidden layer - compresses into d=64
        self.fc2 = nn.Linear(128, 64)
        # the final output layer - splits into 10 classes (digits 0-9)
        self.fc3 = nn.Linear(64, 10)

    # define the forward pass compute graph
    def forward(self, x):
        # x is of shape BxHxW

        # we first need to unroll the 2D image using view
        # we set the first dim to be -1 meaning "everything else", the reason beir
        # we want to maintain different tensors for each training sample in the batch
        x = x.view(-1, 28*28)
        # x is of shape Bx784

        # project-in and apply a non-linearity (ReLU activation function)
        x = torch.relu(self.fc1(x))
        # x is of shape Bx128

        # middle-layer and apply a non-linearity (ReLU activation function)
        x = torch.relu(self.fc2(x))
        # x is of shape Bx64

        # project out into the 10 classes
        x = self.fc3(x)
        # x is of shape Bx10
        return x
```

- accelerate 0.32.1

2024-06-18

- Inline AI completions are now available to users on the free-of-charge tier
- Reduced latency for LSP and terminal connections
- Improved quality of inline completions
- Visual improvements to switch controls across Colab
- Various bug fixes, performance and a11y improvements to the user secrets panel
- Improved tooltip UX behavior
- Improved behavior when copying data from Google Sheets and pasting in Colab
- Scroll to cell fixes for single tabbed view and jump to cell command
- Improved tab header behavior
- A11y improvements for notebook-focused cells
- Python package upgrades
 - torch 2.2.1 -> 2.3.0
 - torchaudio 2.2.1 -> 2.3.0
 - torchvision 0.17.1 -> 0.18.0
 - torchtext 0.17.1 -> 0.18.0
 - google-cloud-aiplatform 1.51.0 -> 1.56.0
 - bigframes 1.5.0 -> 1.8.0
 - regex 2023.12.25 -> 2024.5.15

2024-05-13

- Code actions are now supported to automatically improve and refactor code. Code actions can be triggered by the keyboard shortcut "Ctrl/⌘ + ."
- Python package upgrades
 - bigframes 1.0.0 -> 1.5.0
 - google-cloud-aiplatform 1.47.0 -> 1.51.0
 - jax[tpu] 0.4.23 -> 0.4.26
- Python package inclusions
 - cudf 24.4.1

2024-04-15

- TPU v2 runtime is now available
- L4 runtime is now available for paid users
- New distributed fine-tuning Gemma tutorial on TPUs ([GitHub](#))
- Symbol rename is now supported with keyboard shortcut F2
- Fixed bug causing inability to re-upload deleted files
- Fixed breaking bug in colabtools `%upload_files_async`
- Added syntax highlighting to `%%writefile` cells
- Cuda dependencies that come with Torch are cached for faster downloads for packages that require Torch and its dependencies ([GitHub issue](#))
- Python package upgrades
 - bigframes 0.24.0 -> 1.0.0
 - duckdb 0.9.2 -> 0.10.1
 - google-cloud-aiplatform 1.43.0 -> 1.47.0
 - jax 0.4.23 -> 0.4.26

2024-03-13

- Fixed bug that sometimes caused UserSecrets to move / disappear
- Improved messaging for mounting drive in an unsupported environment ([GitHub issue](#))
- Python package upgrades
 - torch 2.1.0 -> 2.2.1
 - torchaudio 2.1.0 -> 2.2.1
 - torchvision 0.16.0 -> 0.17.1
 - torchtext 0.16.0 -> 0.17.1
 - PyMC 5.7.2 -> 5.10.4
 - BigFrames 0.21.0 -> 0.24.0
 - google-cloud-aiplatform 1.42.1 -> 1.43.0
 - tornado 6.3.2 -> 6.3.3

Before you can begin training, you have to do a little boiler-plate to load the dataset. From the previous assignment, you saw how a hosted dataset can be loaded with TensorFlow. With pytorch it's a little more complicated, as you need to manually condition the input data.

```
# define a transformation for the input images. This uses torchvision.transforms, ar
transform = transforms.Compose([
    transforms.ToTensor(), # first convert to a torch tensor
    transforms.Normalize((0.1307,), (0.3081,)) # then normalize the input
])

# let's download the train and test datasets, applying the above transform - this wi
train_dataset = datasets.MNIST('./data', train=True, download=True, transform=transi
test_dataset = datasets.MNIST('./data', train=False, transform=transform)

# we need to set the mini-batch (commonly referred to as "batch"), for now we can us
batch_size = 64

# then we need to create a dataloader for the train dataset, and we will also creat
# additionally, we will set the batch size in the dataloader
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shu
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuf

# the torch dataloaders allow us to access the __getitem__ method, which returns a
# additionally, the dataloader will pre-colate the training samples into the given t
```

Downloading <http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz>
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading <https://oss-ci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz>
Downloading <https://oss-ci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz>
100%|██████████| 9.91M/9.91M [00:00<00:00, 14.5MB/s]
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading <http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz>
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading <https://oss-ci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz>
Downloading <https://oss-ci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz>
100%|██████████| 28.9k/28.9k [00:00<00:00, 436kB/s]
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading <http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz>
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading <https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz>
Downloading <https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz>
100%|██████████| 1.65M/1.65M [00:00<00:00, 4.07MB/s]
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading <http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz>
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading <https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz>
Downloading <https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz>
100%|██████████| 4.54k/4.54k [00:00<00:00, 11.3MB/s]Extracting ./data/MNIST/raw,

Inspect the first element of the test_loader, and verify both the tensor shapes and data types. You can check the data-type with `.dtype`

Question 1

Edit the cell below to print out the first element shapes, dtype, and identify which is the training sample and which is the training label.

```
# Get the first item from the test_loader
first_item = next(iter(test_loader))

# Extract the training sample (image data) and training label
data, label = first_item
```

2024-02-21

- Try out Gemma on [Colab!](#)
- Allow unicode in form text inputs
- Display documentation and link to source when displaying functions
- Display image-like ndarrays as images
- Improved UX around quick charts and execution error suggestions
- Released Marketplace image for the month of February ([GitHub issue](#))
- Python package upgrades
 - bigframes 0.19.2 -> 0.21.0
 - regex 2023.6.3 -> 2023.12.25
 - spacy 3.6.1 -> 3.7.4
 - beautifulsoup4 4.11.2 -> 4.12.3
 - tensorflow-probability 0.22.0 -> 0.23.0
 - google-cloud-language 2.9.1 -> 2.13.1
 - google-cloud-aiplatform 1.39.0 -> 1.42.1
 - transformers 4.35.2 -> 4.37.2
 - pyarrow 10.0.1 -> 14.0.2

2024-01-29

- New [Kaggle Notebooks <- Colab updates!](#) Now you can:
 - Import directly from Colab without having to download/re-upload
 - Upload via link, by pasting Google Drive or Colab URLs
 - Export & run Kaggle Notebooks on Colab with 1 click
- Try these notebooks that talk to Gemini:
 - [Gemini and Stable Diffusion](#)
 - [Learning with Gemini and ChatGPT](#)
 - [Talk to Gemini with Google's Speech to Text API](#)
 - [Sell lemonade with Gemini and Sheets](#)
 - [Generate images with Gemini and Vertex](#)
- Python package upgrades
 - google-cloud-aiplatform 1.38.1 -> 1.39.0
 - bigframes 0.18.0 -> 0.19.2
 - polars 0.17.3 -> 0.20.2
 - gdown 4.6.6 -> 4.7.3 ([GitHub issue](#))
 - tensorflow-hub 0.15.0 -> 0.16.0
 - flax 0.7.5 -> 0.8.0
- Python package inclusions
 - sentencepiece 0.1.99

2024-01-08

- Avoid nested scrollbars for large outputs by using `google.colab.output.no_vertical_scroll()` [Example notebook](#)
- Fix [bug](#) where downloading models from Hugging Face could freeze
- Python package upgrades
 - huggingface-hub 0.19.4 -> 0.20.2
 - bigframes 0.17.0 -> 0.18.0

2023-12-18

- Expanded access to AI coding has arrived in Colab across 175 locales for all tiers of Colab users
- Improvements to display of ML-based inline completions (for eligible Pro/Pro+ users)
- Started a series of [notebooks](#) highlighting Gemini API capabilities
- Enable `⌘/Ctrl+L` to select the full line in an editor
- Fixed [bug](#) where we weren't correctly formatting output from multiple execution results
- Python package upgrades
 - CUDA 11.8 to CUDA 12.2
 - tensorflow 2.14.0 -> 2.15.0
 - tensorboard 2.14.0 -> 2.15.0
 - keras 2.14.0 -> 2.15.0
 - Nvidia drivers 525.105.17 -> 535.104.05
 - tensorflow-gcs-config 2.14.0 -> 2.15.0
 - bigframes 0.13.0 -> 0.17.0
 - geemap 0.28.2 -> 0.29.6


```
# Print out the shapes and data types
print("Data (Training Sample) Shape:", data.shape) # Shape of the batch of images
print("Data (Training Sample) Dtype:", data.dtype) # Data type of the images

print("Label (Training Label) Shape:", label.shape) # Shape of the batch of labels
print("Label (Training Label) Dtype:", label.dtype) # Data type of the labels

# Identify the training sample and label
print("\nThe 'Data' tensor contains the training samples (images).")
print("The 'Label' tensor contains the corresponding labels (digits 0-9).")
```

```
↳ Data (Training Sample) Shape: torch.Size([64, 1, 28, 28])
Data (Training Sample) Dtype: torch.float32
Label (Training Label) Shape: torch.Size([64])
Label (Training Label) Dtype: torch.int64

The 'Data' tensor contains the training samples (images).
The 'Label' tensor contains the corresponding labels (digits 0-9).
```

Now that we have the dataset loaded, we can instantiate the MLP model, the loss (or criterion function), and the optimizer for training.

```
# create the model
model = MLP()

# you can print the model as well, but notice how the activation functions are missing
# and not declared in the constructor
print(model)

# you can also count the model parameters
param_count = sum([p.numel() for p in model.parameters()])
print(f"Model has {param_count:,} trainable parameters")

# for a criterion (loss) function, you will use Cross-Entropy Loss. This is the most
# and is also used by tokenized transformer models it takes in an un-normalized probability
# N classes (in our case, 10 classes with MNIST). This distribution is then compared
# For MNIST, the prediction might be [-0.0056, -0.2044, 1.1726, 0.0859, 1.8443, -0.
# Cross-entropy can be thought of as finding the difference between the predicted distribution
criterion = nn.CrossEntropyLoss()

# then you can instantiate the optimizer. You will use Stochastic Gradient Descent (SGD)
# factor of 0.5. the first input to the optimizer is the list of model parameters, with lr=0.01, momentum=0.5
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)

↳ MLP(
  (fc1): Linear(in_features=784, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=10, bias=True)
)
Model has 109,386 trainable parameters
```

Finally, you can define a training, and test loop

```
# create an array to log the loss and accuracy
train_losses = []
train_steps = []
test_steps = []
test_losses = []
test_accuracy = []
current_step = 0 # Start with global step 0
current_epoch = 0 # Start with epoch 0
```

```
# declare the train function
def cpu_train(epoch, train_losses, steps, current_step):

    # set the model in training mode - this doesn't do anything for us right now, but
    # batch norm and dropout
    model.train()

    # Create tqdm progress bar to help keep track of the training progress
    pbar = tqdm(enumerate(train_loader), total=len(train_loader))

    # loop over the dataset. Recall what comes out of the data loader, and then by using
    # iterator list which we will call batch_idx
    for batch_idx, (data, target) in pbar:
```

- pyarrow 9.0.0 -> 10.0.1
- google-generativeai 0.2.2 -> 0.3.1
- jax 0.4.20 -> 0.4.23
- jaxlib 0.4.20 -> 0.4.23

- Python package inclusions
 - kagglehub 0.1.4
 - google-cloud-aiplatform 1.38.1

2023-11-27

- Removed warning when calling await to make it render as code
- Added "Run selection" to the cell context menu
- Added highlighting for the %%python cell magic
- Launched AI coding features for Pro/Pro+ users in more locales
- Python package upgrades
 - bigframes 0.12.0 -> 0.13.0
- Python package inclusions
 - transformers 4.35.2
 - google-generativeai 0.2.2

2023-11-08

- Launched Secrets, for safe storage of private keys on Colab ([tweet](#))
- Fixed issue where TensorBoard would not load ([#3990](#))
- Python package upgrades
 - lightgbm 4.0.0 -> 4.1.0
 - bigframes 0.10.0 -> 0.12.0
 - bokeh 3.2.2 -> 3.3.0
 - duckdb 0.8.1 -> 0.9.1
 - numba 0.56.4 -> 0.58.1
 - tweepy 4.13.0 -> 4.14.0
 - jax 0.4.16 -> 0.4.20
 - jaxlib 0.4.16 -> 0.4.20

2023-10-23

- Updated the **Open notebook** dialog for better usability and support for smaller screen sizes
- Added smart paste support for data from Google Sheets for R notebooks
- Enabled showing release notes in a tab
- Launched AI coding features for Pro/Pro+ users in Australia AU Canada CA India IN and Japan JP ([tweet](#))
- Python package upgrades
 - earthengine-api 0.1.357 -> 0.1.375
 - flax 0.7.2 -> 0.7.4
 - geemap 0.27.4 -> 0.28.2
 - jax 0.4.14 -> 0.4.16
 - jaxlib 0.4.14 -> 0.4.16
 - keras 2.13.1 -> 2.14.0
 - tensorboard 2.13.0 -> 2.14.1
 - tensorflow 2.13.0 -> 2.14.0
 - tensorflow-gcs-config 2.13.0 -> 2.14.0
 - tensorflow-hub 0.14.0 -> 0.15.0
 - tensorflow-probability 0.20.1 -> 0.22.0
 - torch 2.0.1 -> 2.1.0
 - torchaudio 2.0.2 -> 2.1.0
 - torchtext 0.15.2 -> 0.16.0
 - torchvision 0.15.2 -> 0.16.0
 - xgboost 1.7.6 -> 2.0.0
- Python package inclusions
 - bigframes 0.10.0
 - malloy 2023.1056

2023-09-22

- Added the ability to scope an AI generated suggestion to a specific Pandas dataframe ([tweet](#))
- Added Colab link previews to Docs ([tweet](#))
- Added smart paste support for data from Google Sheets
- Increased font size of dropdowns in interactive forms
- Improved rendering of the notebook when printing
- Python package upgrades
 - tensorflow 2.12.0 -> 2.13.0

```

# during training, the first step is to zero all of the gradients through t
# this resets the state so that we can begin back propogation with the updat
optimizer.zero_grad()

# then you can apply a forward pass, which includes evaluating the loss (cri
output = model(data)
loss = criterion(output, target)

# given that you want to minimize the loss, you need to call .backward() on
loss.backward()

# the backward step will automatically differentiate the model and apply a g
# so then all you have to do is call optimizer.step() to apply the gradient
optimizer.step()

# increment the step count
current_step += 1

# you should add some output to the progress bar so that you know which epoch
if batch_idx % 100 == 0:

    # append the last loss value
    train_losses.append(loss.item())
    steps.append(current_step)

    desc = (f'Train Epoch: {epoch} [{batch_idx * len(data)} / {len(train_loader)}]
            f' ({100. * batch_idx / len(train_loader):.0f}%) \t Loss: {loss:.4f} \t'
            pbar.set_description(desc)

return current_step

# declare a test function, this will help you evaluate the model progress on a data
# doing so prevents cross-contamination and misleading results due to overfitting
def cpu_test(test_losses, test_accuracy, steps, current_step):

    # put the model into eval mode, this again does not currently do anything for y
    # and dropout
    model.eval()
    test_loss = 0
    correct = 0

    # Create tqdm progress bar
    pbar = tqdm(test_loader, total=len(test_loader), desc="Testing...")

    # since you are not training the model, and do not need back-propagation, you ca
    with torch.no_grad():
        # iterate over the test set
        for data, target in pbar:
            # like with training, run a forward pass through the model and evaluate
            output = model(data)
            test_loss += criterion(output, target).item() # you are using .item() to

            # you can also check the accuracy by sampling the output - you can use g
            # in general, you would want to normalize the logits first (the un-norma
            # however, argmax is taking the maximum value, which will be the same in
            # so we can skip a step and take argmax directly
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader)

    # append the final test loss
    test_losses.append(test_loss)
    test_accuracy.append(correct/len(test_loader.dataset))
    steps.append(current_step)

    print(f'\nTest set: Average loss: {test_loss:.4f}, Accuracy: {correct}/{len(test
          f' ({100. * correct / len(test_loader.dataset):.0f}%) \n')

# train for 10 epochs
for epoch in range(0, 10):
    current_step = cpu_train(current_epoch, train_losses, train_steps, current_step)
    cpu_test(test_losses, test_accuracy, test_steps, current_step)
    current_epoch += 1

```

```

Train Epoch: 0 [57600/60000 (96%)] Loss: 0.143317: 100%|██████████| 938/938
Testing...: 100%|██████████| 157/157 [00:02<00:00, 75.81it/s]

```

- tensorboard 2.12.3 -> 2.13.0
- keras 2.12.0 -> 2.13.1
- tensorflow-gcs-config 2.12.0 -> 2.13.
- scipy 1.10.1 -> 1.11.2
- cython 0.29.6 -> 3.0.2
- Python package inclusions
 - geemap 0.26.0

2023-08-18

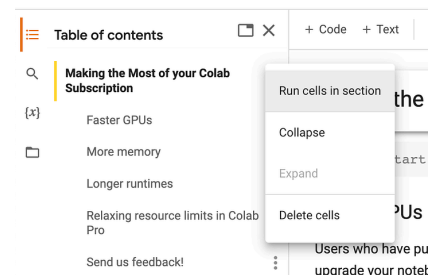
- Added "Change runtime type" to the menu in the connection button
- Improved auto-reconnection to an already running notebook ([#3764](#))
- Increased the specs of our highmem machines for Pro users
- Fixed add-apt-repository command on Ubuntu 22.04 runtime ([#3867](#))
- Python package upgrades
 - bokeh 2.4.3 -> 3.2.2
 - cmake 3.25.2 -> 3.27.2
 - cryptography 3.4.8 -> 41.0.3
 - dask 2022.12.1 -> 2023.8.0
 - distributed 2022.12.1 -> 2023.8.0
 - earthengine-api 0.1.358 -> 0.1.364
 - flax 0.7.0 -> 0.7.2
 - ipython-sql 0.4.0 -> 0.5.0
 - jax 0.4.13 -> 0.4.14
 - jaxlib 0.4.13 -> 0.4.14
 - lightgbm 3.3.5 -> 4.0.0
 - mkl 2019.0 -> 2023.2.0
 - notebook 6.4.8 -> 6.5.5
 - numpy 1.22.4 -> 1.23.5
 - opencv-python 4.7.0.72 -> 4.8.0.76
 - pillow 8.4.0 -> 9.4.0
 - plotly 5.13.1 -> 5.15.0
 - prettytable 0.7.2 -> 3.8.0
 - pytensor 2.10.1 -> 2.14.2
 - spacy 3.5.4 -> 3.6.1
 - statsmodels 0.13.5 -> 0.14.0
 - xarray 2022.12.0 -> 2023.7.0
- Python package inclusions
 - PyDrive2 1.6.3

2023-07-21

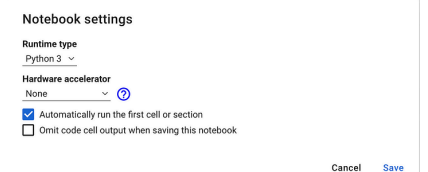
- Launched auto-plotting for dataframes, available using the chart button that shows up alongside datatables ([post](#))



- Added a menu to the table of contents to support running a section or collapsing/expanding sections ([post](#))



- Added an option to automatically run the first cell or section, available under Edit -> Notebook settings ([post](#))



- Launched Pro/Pro+ to Algeria, Argentina, Chile, Ecuador, Egypt, Ghana, Kenya, Malaysia, Nepal, Nigeria, Peru, Rwanda, Saudi Arabia, South Africa, Sri Lanka, Tunisia, and Ukraine ([tweet](#))
- Added a command, "Toggle tab moves focus" for toggling tab trapping in the editor (Tools -> Command palette, "Toggle tab moves focus")

```

Test set: Average loss: 0.2653, Accuracy: 9226/10000 (92%)

Train Epoch: 1 [57600/60000 (96%)]      Loss: 0.119472: 100%|██████████| 938/938
Testing...: 100%|██████████| 157/157 [00:02<00:00, 70.87it/s]

Test set: Average loss: 0.1943, Accuracy: 9410/10000 (94%)

Train Epoch: 2 [57600/60000 (96%)]      Loss: 0.083593: 100%|██████████| 938/938
Testing...: 100%|██████████| 157/157 [00:02<00:00, 76.03it/s]

Test set: Average loss: 0.1536, Accuracy: 9542/10000 (95%)

Train Epoch: 3 [57600/60000 (96%)]      Loss: 0.075049: 100%|██████████| 938/938
Testing...: 100%|██████████| 157/157 [00:02<00:00, 76.61it/s]

Test set: Average loss: 0.1371, Accuracy: 9591/10000 (96%)

Train Epoch: 4 [57600/60000 (96%)]      Loss: 0.106909: 100%|██████████| 938/938
Testing...: 100%|██████████| 157/157 [00:02<00:00, 61.88it/s]

Test set: Average loss: 0.1156, Accuracy: 9655/10000 (97%)

Train Epoch: 5 [57600/60000 (96%)]      Loss: 0.202814: 100%|██████████| 938/938
Testing...: 100%|██████████| 157/157 [00:02<00:00, 71.26it/s]

Test set: Average loss: 0.1114, Accuracy: 9647/10000 (96%)

Train Epoch: 6 [57600/60000 (96%)]      Loss: 0.054692: 100%|██████████| 938/938
Testing...: 100%|██████████| 157/157 [00:02<00:00, 75.24it/s]

Test set: Average loss: 0.0962, Accuracy: 9702/10000 (97%)

Train Epoch: 7 [57600/60000 (96%)]      Loss: 0.107146: 100%|██████████| 938/938
Testing...: 100%|██████████| 157/157 [00:02<00:00, 54.97it/s]

Test set: Average loss: 0.0879, Accuracy: 9722/10000 (97%)

Train Epoch: 8 [57600/60000 (96%)]      Loss: 0.017227: 100%|██████████| 938/938
Testing...: 100%|██████████| 157/157 [00:02<00:00, 75.85it/s]

Test set: Average loss: 0.0851, Accuracy: 9731/10000 (97%)

Train Epoch: 9 [57600/60000 (96%)]      Loss: 0.100554: 100%|██████████| 938/938
Testing...: 100%|██████████| 157/157 [00:02<00:00, 73.79it/s]
Test set: Average loss: 0.0801, Accuracy: 9755/10000 (98%)

```

Question 2

Using the skills you acquired in the previous assignment edit the cell below to use matplotlib to visualize the loss for training and validation for the first 10 epochs. They should be plotted on the same graph, labeled, and use a log-scale on the y-axis.

```

# visualize the losses for the first 10 epochs
import matplotlib.pyplot as plt

# Assuming train_losses and test_losses have been populated during training and test

# For the first 10 epochs, we plot the losses
plt.figure(figsize=(10, 6))

# Plot training losses
plt.plot(train_steps[:10], train_losses[:10], label='Training Loss', color='blue')

# Plot test (validation) losses
plt.plot(test_steps[:10], test_losses[:10], label='Validation Loss', color='red')

# Set the scale of the y-axis to logarithmic
plt.yscale('log')

# Add labels and title
plt.xlabel('Step')
plt.ylabel('Loss')
plt.title('Training and Validation Losses (Log Scale)')

# Add a legend
plt.legend()

# Display the plot

```

- Fixed issue where files.upload() was sometimes returning an incorrect filename ([#1550](#))
- Fixed f-string syntax highlighting bug ([#3802](#))
- Disabled ambiguous characters highlighting for commonly used LaTeX characters ([#3648](#))
- Upgraded Ubuntu from 20.04 LTS to [22.04 LTS](#)
- Updated the Colab Marketplace VM image
- Python package upgrades:
 - autograd 1.6.1 -> 1.6.2
 - driftnet 76.0 -> 77.0
 - flax 0.6.11 -> 0.7.0
 - earthengine-api 0.1.357 -> 0.1.358
 - GDAL 3.3.2 -> 3.4.3
 - google-cloud-bigquery-storage 2.20.0 -> 2.22.2
 - gsread-dataframe 3.0.8 -> 3.3.1
 - holidays 0.27.1 -> 0.29
 - jax 0.4.10 -> jax 0.4.13
 - jupyterlab-widgets 3.0.7 -> 3.0.8
 - nbformat 5.9.0 -> 5.9.1
 - opencv-python-headless 4.7.0.72 -> 4.8.0.74
 - pygame 2.4.0 -> 2.5.0
 - spacy 3.5.3 -> 3.5.4
 - SQLAlchemy 2.0.16 -> 2.0.19
 - tabulate 0.8.10 -> 0.9.0
 - tensorflow-hub 0.13.0 -> 0.14.0

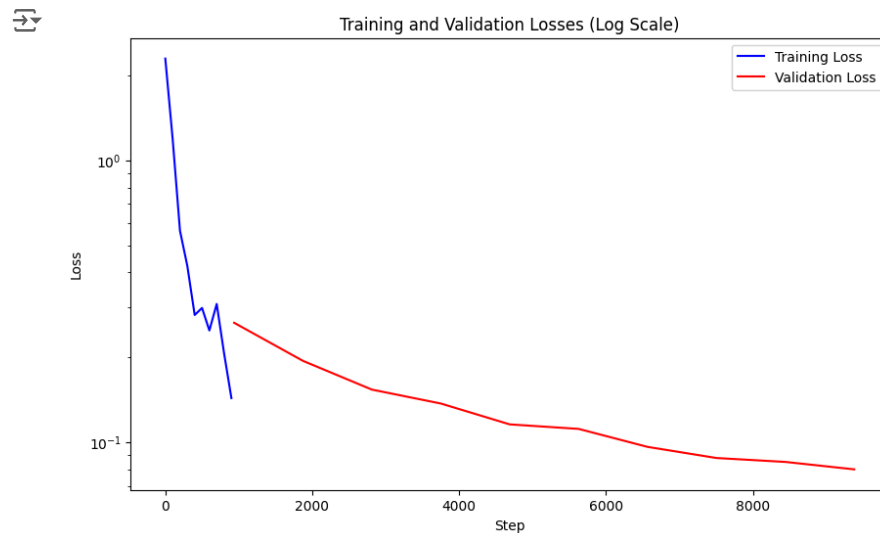
2023-06-23

- Launched AI coding features to subscribed users starting with Pro+ users in the US ([tweet](#), [post](#))
- Added the Kernel Selector in the Notebook Settings ([tweet](#))
- Fixed double space trimming issue in markdown [#3766](#)
- Fixed run button indicator not always centered [#3609](#)
- Fixed inconsistencies for automatic indentation on multi-line [#3697](#)
- Upgraded Python from 3.10.11 to 3.10.12
- Python package updates:
 - duckdb 0.7.1 -> 0.8.1
 - earthengine-api 0.1.350 -> 0.1.357
 - flax 0.6.9 -> 0.6.11
 - google-cloud-bigquery 3.9.0 -> 3.10.0
 - google-cloud-bigquery-storage 2.19.1 -> 2.20.0
 - grpcio 1.54.0 -> 1.56.0
 - holidays 0.25 -> 0.27.1
 - nbformat 5.8.0 -> 5.9.0
 - prophet 1.1.3 -> 1.1.4
 - pydata-google-auth 1.7.0 -> 1.8.0
 - spacy 3.5.2 -> 3.5.3
 - tensorboard 2.12.2 -> 2.12.3
 - xgboost 1.7.5 -> 1.7.6
- Python package inclusions:
 - gcsfs 2023.6.0
 - geopandas 0.13.2
 - google-cloud-bigquery-connection 1.12.0
 - google-cloud-functions 1.13.0
 - grpc-google-iam-v1 0.12.6
 - multidict 6.0.4
 - tensorboard-data-server 0.7.1

2023-06-02

- Released the new site [colab.google](#)
- Published Colab's Docker runtime image to us-docker.pkg.dev/colab-images/public/runtime ([tweet](#), [instructions](#))
- Launched support for Google children accounts ([tweet](#))
- Launched DagsHub integration ([tweet](#), [post](#))
- Upgraded to Monaco Editor Version 0.37.1
- Fixed various Vim keybinding bugs
- Fixed issue where the N and P letters sometimes couldn't be typed ([#3664](#))
- Fixed rendering support for compositional inputs ([#3660](#), [#3679](#))
- Fixed lag in notebooks with lots of cells ([#3676](#))
- Improved support for R by adding a Runtime type notebook setting (Edit -> Notebook settings)


```
plt.show()
```



Question 3

The model may be able to train for a bit longer. Edit the cell below to modify the previous training code to also report the time per epoch and the time for 10 epochs with testing. You can use `time.time()` to get the current time in seconds. Then run the model for another 10 epochs, printing out the execution time at the end, and replot the loss functions with the extra 10 epochs below.

```
# visualize the losses for 20 epochs
import time
import matplotlib.pyplot as plt

# Function to track time during training
for epoch in range(10, 20): # Continue from epoch 10 to 19 for a total of 20 epochs
    start_time = time.time() # Record the start time of the epoch

    # Train for one epoch
    current_step = cpu_train(epoch, train_losses, train_steps, current_step)

    # Test after training
    cpu_test(test_losses, test_accuracy, test_steps, current_step)

    # Measure time after epoch ends
    epoch_time = time.time() - start_time # Calculate time taken for the epoch
    print(f'Epoch {epoch+1} completed in {epoch_time:.2f} seconds.')

    # Increment the epoch counter
    current_epoch += 1

# Calculate the total time for 10 additional epochs (epoch 10 to 19)
total_time = time.time() - start_time
print(f'\nTotal time for 10 epochs (with testing): {total_time:.2f} seconds.')
```

```
Train Epoch: 10 [57600/60000 (96%)] Loss: 0.071250: 100%|██████████| 938/
Testing...: 100%|██████████| 157/157 [00:02<00:00, 72.58it/s]

Test set: Average loss: 0.0818, Accuracy: 9737/10000 (97%)

Epoch 11 completed in 16.75 seconds.
Train Epoch: 11 [57600/60000 (96%)] Loss: 0.049412: 100%|██████████| 938/
Testing...: 100%|██████████| 157/157 [00:02<00:00, 60.79it/s]

Test set: Average loss: 0.0761, Accuracy: 9771/10000 (98%)
```

- Improved documentation for connecting to a local runtime (Connect -> Connect to a local runtime)
- Python package updates:
 - holidays 0.23 -> 0.25
 - jax 0.4.8 -> 0.4.10
 - jaxlib 0.4.8 -> 0.4.10
 - pip 23.0.1 -> 23.1.2
 - tensorflow-probability 0.19.0 -> 0.20.1
 - torch 2.0.0 -> 2.0.1
 - torchaudio 2.0.1 -> 2.0.2
 - torchdata 0.6.0 -> 0.6.1
 - torchtext 0.15.1 -> 0.15.2
 - torchvision 0.15.1 -> 0.15.2
 - tornado 6.2 -> 6.3.1

2023-05-05

- Released GPU type selection for paid users, allowing them to choose a preferred NVidia GPU
- Upgraded R from 4.2.3 to 4.3.0
- Upgraded Python from 3.9.16 to 3.10.11
- Python package updates:
 - attrs 22.2.0 -> attrs 23.1.0
 - earthengine-api 0.1.349 -> earthengine-api 0.1.350
 - flax 0.6.8 -> 0.6.9
 - grpcio 1.53.0 -> 1.54.0
 - nbclient 0.7.3 -> 0.7.4
 - tensorflow-datasets 4.8.3 -> 4.9.2
 - termcolor 2.2.0 -> 2.3.0
 - zict 2.2.0 -> 3.0.0

2023-04-14

- Python package updates:
 - google-api-python-client 2.70.0 -> 2.84.0
 - google-auth-oauthlib 0.4.6 -> 1.0.0
 - google-cloud-bigquery 3.4.2 -> 3.9.0
 - google-cloud-datastore 2.11.1 -> 2.15.1
 - google-cloud-firestore 2.7.3 -> 2.11.0
 - google-cloud-language 2.6.1 -> 2.9.1
 - google-cloud-storage 2.7.0 -> 2.8.0
 - google-cloud-translate 3.8.4 -> 3.11.1
 - networkx 3.0 -> 3.1
 - notebook 6.3.0 -> 6.4.8
 - jax 0.4.7 -> 0.4.8
 - pandas 1.4.4 -> 1.5.3
 - spacy 3.5.1 -> 3.5.2
 - SQLAlchemy 1.4.47 -> 2.0.9
 - xgboost 1.7.4 -> 1.7.5

2023-03-31

- Improve bash ! syntax highlighting ([GitHub issue](#))
- Fix bug where VIM keybindings weren't working in the file editor
- Upgraded R from 4.2.2 to 4.2.3
- Python package updates:
 - arviz 0.12.1 -> 0.15.1
 - astropy 4.3.1 -> 5.2.2
 - dopamine-rl 1.0.5 -> 4.0.6
 - gensim 3.6.0 -> 4.3.1
 - ipykernel 5.3.4 -> 5.5.6
 - ipython 7.9.0 -> 7.34.0
 - jax 0.4.4 -> 0.4.7
 - jaxlib 0.4.4 -> 0.4.7
 - jupyter_core 5.2.0 -> 5.3.0
 - keras 2.11.0 -> 2.12.0
 - lightgbm 2.2.3 -> 3.3.5
 - matplotlib 3.5.3 -> 3.7.1
 - nltk 3.7 -> 3.8.1
 - opencv-python 4.6.0.66 -> 4.7.0.72
 - plotly 5.5.0 -> 5.13.1
 - pymc 4.1.4 -> 5.1.2
 - seaborn 0.11.2 -> 0.12.2
 - spacy 3.4.4 -> 3.5.1
 - sympy 1.7.1 -> 1.11.1
 - tensorboard 2.11.2 -> 2.12.0
 - tensorflow 2.11.0 -> 2.12.0
 - tensorflow-estimator 2.11.0 -> 2.12.0
 - tensorflow-hub 0.12.0 -> 0.13.0
 - torch 1.13.1 -> 2.0.0
 - torchaudio 0.13.1 -> 2.0.1
 - torchtext 0.14.1 -> 0.15.1
 - torchvision 0.14.1 -> 0.15.1

2023-03-10

```

Epoch 12 completed in 17.68 seconds.
Train Epoch: 12 [57600/60000 (96%)] Loss: 0.100223: 100%[████████] 938/
Testing...: 100%[████████] 157/157 [00:02<00:00, 73.87it/s]

Test set: Average loss: 0.0729, Accuracy: 9764/10000 (98%)

Epoch 13 completed in 16.67 seconds.
Train Epoch: 13 [57600/60000 (96%)] Loss: 0.013847: 100%[████████] 938/
Testing...: 100%[████████] 157/157 [00:02<00:00, 74.89it/s]

Test set: Average loss: 0.0698, Accuracy: 9779/10000 (98%)

Epoch 14 completed in 16.78 seconds.
Train Epoch: 14 [57600/60000 (96%)] Loss: 0.159503: 100%[████████] 938/
Testing...: 100%[████████] 157/157 [00:02<00:00, 74.06it/s]

Test set: Average loss: 0.0798, Accuracy: 9761/10000 (98%)

Epoch 15 completed in 17.45 seconds.
Train Epoch: 15 [57600/60000 (96%)] Loss: 0.038621: 100%[████████] 938/
Testing...: 100%[████████] 157/157 [00:02<00:00, 75.46it/s]

Test set: Average loss: 0.0681, Accuracy: 9782/10000 (98%)

Epoch 16 completed in 16.54 seconds.
Train Epoch: 16 [57600/60000 (96%)] Loss: 0.007841: 100%[████████] 938/
Testing...: 100%[████████] 157/157 [00:02<00:00, 64.42it/s]

Test set: Average loss: 0.0687, Accuracy: 9793/10000 (98%)

Epoch 17 completed in 16.98 seconds.
Train Epoch: 17 [57600/60000 (96%)] Loss: 0.036781: 100%[████████] 938/
Testing...: 100%[████████] 157/157 [00:02<00:00, 73.75it/s]

Test set: Average loss: 0.0698, Accuracy: 9787/10000 (98%)

Epoch 18 completed in 17.04 seconds.
Train Epoch: 18 [57600/60000 (96%)] Loss: 0.012288: 100%[████████] 938/
Testing...: 100%[████████] 157/157 [00:02<00:00, 72.96it/s]

Test set: Average loss: 0.0680, Accuracy: 9796/10000 (98%)

Epoch 19 completed in 16.72 seconds.
Train Epoch: 19 [57600/60000 (96%)] Loss: 0.022509: 100%[████████] 938/
Testing...: 100%[████████] 157/157 [00:02<00:00, 53.44it/s]

```

```

# Replot the losses for the first 20 epochs
plt.figure(figsize=(10, 6))

# Plot training losses
plt.plot(train_steps[:20], train_losses[:20], label='Training Loss', color='blue')

# Plot test (validation) losses
plt.plot(test_steps[:20], test_losses[:20], label='Validation Loss', color='red')

# Set the scale of the y-axis to logarithmic
plt.yscale('log')

# Add labels and title
plt.xlabel('Step')
plt.ylabel('Loss')
plt.title('Training and Validation Losses (Log Scale)')

# Add a legend
plt.legend()

# Display the plot
plt.show()

```

- Added the [Colab editor shortcuts](#) example notebook
- Fixed triggering of @-mention and email autocomplete for large comments ([GitHub issue](#))
- Added View Resources to the Runtime menu
- Made file viewer images fit the view by default, resizing to original size on click
- When in VIM mode, enable copy as well as allowing propagation to monaco-vim to escape visual mode ([GitHub issue](#))
- Upgraded CUDA 11.6.2 -> 11.8.0 and cuDNN 8.4.0.27 -> 8.7.0.84
- Upgraded Nvidia drivers 525.78.01 -> 530.30.02
- Upgraded Python 3.8.10 -> 3.9.16
- Python package updates:

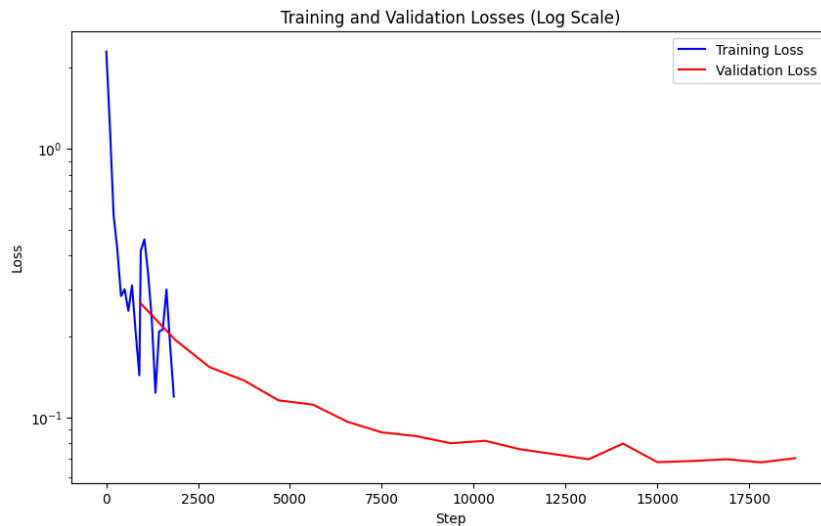
- beautifulsoup4 4.6.3 -> 4.9.3
- bokeh 2.3.3 -> 2.4.3
- debugpy 1.0.0 -> 1.6.6
- Flask 1.1.4 -> 2.2.3
- jax 0.3.25 -> 0.4.4
- jaxlib 0.3.25 -> 0.4.4
- Jinja2 2.11.3 -> 3.1.2
- matplotlib 3.2.2 -> 3.5.3
- nbconvert 5.6.1 -> 6.5.4
- pandas 1.3.5 -> 1.4.4
- pandas-datareader 0.9.0 -> 0.10.0
- pandas-profiling 1.4.1 -> 3.2.0
- Pillow 7.1.2 -> 8.4.0
- plotnine 0.8.0 -> 0.10.1
- scikit-image 0.18.3 -> 0.19.3
- scikit-learn 1.0.2 -> 1.2.2
- scipy 1.7.3 -> 1.10.1
- setuptools 57.4.0 -> 63.4.3
- sklearn-pandas 1.8.0 -> 2.2.0
- statsmodels 0.12.2 -> 0.13.5
- urllib3 1.24.3 -> 1.26.14
- Werkzeug 1.0.1 -> 2.2.3
- wrapt 1.14.1 -> 1.15.0
- xgboost 0.90 -> 1.7.4
- xlrd 1.2.0 -> 2.0.1

2023-02-17

- Show graphs of RAM and disk usage in notebook toolbar
- Copy cell links directly to the clipboard instead of showing a dialog when clicking on the link icon in the cell toolbar
- Updated the [Colab Marketplace VM image](#)
- Upgraded CUDA to 11.6.2 and cuDNN to 8.4.0.27
- Python package updates:
 - tensorflow 2.9.2 -> 2.11.0
 - tensorboard 2.9.1 -> 2.11.2
 - keras 2.9.0 -> 2.11.0
 - tensorflow-estimator 2.9.0 -> 2.11.0
 - tensorflow-probability 0.17.0 -> 0.19.0
 - tensorflow-gcs-config 2.9.0 -> 2.11.0
 - earthengine-api 0.1.339 -> 0.1.341
 - flatbuffers 1.12 -> 23.1.21
 - platformdirs 2.6.2 -> 3.0.0
 - pydata-google-auth 1.6.0 -> 1.7.0
 - python-utils 3.4.5 -> 3.5.2
 - tenacity 8.1.0 -> 8.2.1
 - tiffiffe 2023.1.23.1 -> 2023.2.3
 - notebook 5.7.16 -> 6.3.0
 - tornado 6.0.4 -> 6.2
 - aiohttp 3.8.3 -> 3.8.4
 - charset-normalizer 2.1.1 -> 3.0.1
 - fastai 2.7.0 -> 2.7.1
 - soundfile 0.11.0 -> 0.12.1
 - typing-extensions 4.4.0 -> 4.5.0
 - widgetsnbextension 3.6.1 -> 3.6.2
 - pydantic 1.10.4 -> 1.10.5
 - zipp 3.12.0 -> 3.13.0
 - numpy 1.21.6 -> 1.22.4
 - drivesfs 66.0 -> 69.0
 - gdal 3.0.4 -> 3.3.2 [GitHub issue](#)
- Added libudunits2-dev for smoother R package installs [GitHub issue](#)

2023-02-03

- Improved tooltips for pandas series to show common statistics about the series object
- Made the forms dropdown behave like an autocomplete box when it allows input
- Updated the nvidia driver from 460.32.03 to 510.47.03



- Python package updates:
 - absl-py 1.3.0 -> 1.4.0
 - bleach 5.0.1 -> 6.0.0
 - cachetools 5.2.1 -> 5.3.0
 - cmdstanpy 1.0.8 -> 1.1.0
 - dnspython 2.2.1 -> 2.3.0
 - fsspec 2022.11.0 -> 2023.1.0
 - google-cloud-bigquery-storage 2.17.0 -> 2.18.1
 - holidays 0.18 -> 0.19
 - jupyter-core 5.1.3 -> 5.2.0
 - packaging 21.3 -> 23.0
 - prometheus-client 0.15.0 -> 0.16.0
 - pyct 0.4.8 -> 0.5.0
 - pydata-google-auth 1.5.0 -> 1.6.0
 - python-slugify 7.0.0 -> 8.0.0
 - sqlalchemy 1.4.46 -> 2.0.0
 - tensorflow-io-gcs-filesystem 0.29.0 -> 0.30.0
 - tifffile 2022.10.10 -> 2023.1.23.1
 - zipp 3.11.0 -> 3.12.0
 - Pinned sqlalchemy to version 1.4.46

2023-01-12

- Added support for @-mention and email autocomplete in comments
- Improved errors when GitHub notebooks can't be loaded
- Increased color contrast for colors used for syntax highlighting in the code editor
- Added terminal access for custom GCE VM runtimes
- Upgraded Ubuntu from 18.04 LTS to 20.04 LTS ([GitHub issue](#))
- Python package updates:
 - GDAL 2.2.2 -> 2.2.3.
 - NumPy from 1.21.5 to 1.21.6.
 - attrs 22.1.0 -> 22.2.0
 - chardet 3.0.4 -> 4.0.0
 - cloudpickle 1.6.0 -> 2.2.0
 - filelock 3.8.2 -> 3.9.0
 - google-api-core 2.8.2 -> 2.11.0
 - google-api-python-client 1.12.11 -> 2.70.0
 - google-auth-http2 0.0.3 -> 0.1.0
 - google-cloud-bigquery 3.3.5 -> 3.4.1
 - google-cloud-datastore 2.9.0 -> 2.11.0
 - google-cloud-firestore 2.7.2 -> 2.7.3
 - google-cloud-storage 2.5.0 -> 2.7.0
 - holidays 0.17.2 -> holidays 0.18
 - importlib-metadata 5.2.0 -> 6.0.0
 - networkx 2.8.8 -> 3.0
 - opencv-python-headless 4.6.0.66 -> 4.7.0.68
 - pip 21.1.3 -> 22.04
 - pip-tools 6.2.0 -> 6.6.2
 - prettytable 3.5.0 -> 3.6.0
 - requests 2.23.0 -> 2.25.1
 - termcolor 2.1.1 -> 2.2.0
 - torch 1.13.0 -> 1.13.1
 - torchaudio 0.13.0 -> 0.13.1
 - torchtext 0.14.0 -> 0.14.1
 - torchvision 0.14.0 -> 0.14.1

Question 4

Make an observation from the above plot. Do the test and train loss curves indicate that the model should train longer to improve accuracy? Or does it indicate that 20 epochs is too long? Edit the cell below to answer these questions.

The training loss is still decreasing and the test loss is flat or slightly decreasing, the model can continue training for a few more epochs, especially if the test loss hasn't reached its minimum yet.

✓ Moving to the GPU

Now that you have a model trained on the CPU, let's finally utilize the T4 GPU that we requested for this instance.

Using a GPU with torch is relatively simple, but has a few gotchas. Torch abstracts away most of the CUDA runtime API, but has a few hold-over concepts such as moving data between devices. Additionally, since the GPU is treated as a device separate from the CPU, you cannot combine CPU and GPU based tensors in the same operation. Doing so will result in a device mismatch error. If this occurs, check where the tensors are located (you can always print `.device` on a tensor), and make sure they have been properly moved to the correct device.

You will start by creating a new model, optimizer, and criterion (not really necessary in this case since you already did this above but it's better for clarity and completeness). However, one change that you'll make is moving the model to the GPU first. This can be done by calling `.cuda()` in general, or `.to("cuda")` to be more explicit. In general specific GPU devices can be targetted such as `.to("cuda:0")` for the first GPU (index 0), etc., but since there is only one GPU in Colab this is not necessary in this case.

```
# create the model
model = MLP()

# move the model to the GPU
model.cuda()

# for a criterion (loss) function, we will use Cross-Entropy Loss. This is the most
# it takes in an un-normalized probability distribution (i.e. without softmax) over
# which is < N. For MNIST, the prediction might be [-0.0056, -0.2044, 1.1726, 0.08]
# Cross-entropy can be thought of as finding the difference between what the predict
```

2022-12-06

- Made fallback runtime version available until mid-December ([GitHub issue](#))
- Upgraded to Python 3.8 ([GitHub issue](#))
- Python package updates:
 - jax from 0.3.23 to 0.3.25, jaxlib from 0.3.22 to 0.3.25
 - pyarrow from 6.0.1 to 9.0.0
 - torch from 1.12.1 to 1.13.0
 - torchaudio from 0.12.1 to 0.13.0
 - torchvision from 0.13.1 to 0.14.0
 - torchtext from 0.13.1 to 0.14.0
 - xldr from 1.1.0 to 1.2.0
 - DriveFS from 62.0.1 to 66.0.3
- Made styling of markdown tables in outputs match markdown tables in text cells
- Improved formatting for empty interactive table rows
- Fixed syntax highlighting for variables with names that contain Python keywords ([GitHub issue](#))

```

criterion = nn.CrossEntropyLoss()

# then you can instantiate the optimizer. You will use Stochastic Gradient Descent (SGD)
# the first input to the optimizer is the list of model parameters, which is obtained from the model
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)

# create a new array to log the loss and accuracy
train_losses = []
train_steps = []
test_steps = []
test_losses = []
test_accuracy = []
current_step = 0 # Start with global step 0
current_epoch = 0 # Start with epoch 0

data = data.cuda()
target = target.cuda()

import time
import torch

# Function to track time during training
for epoch in range(10, 20): # Continue from epoch 10 to 19 for a total of 20 epochs
    start_time = time.time() # Record the start time of the epoch

    # Train for one epoch
    model.train() # Set model to training mode
    pbar = tqdm(enumerate(train_loader), total=len(train_loader))

    for batch_idx, (data, target) in pbar:
        # Move data and target to GPU
        data = data.cuda()
        target = target.cuda()

        # Zero the gradients before backpropagation
        optimizer.zero_grad()

        # Forward pass
        output = model(data)

        # Compute loss
        loss = criterion(output, target)

        # Backpropagate
        loss.backward()

        # Apply gradients
        optimizer.step()

        # Update step counter
        current_step += 1

    # Update progress bar
    if batch_idx % 100 == 0:
        train_losses.append(loss.item())
        train_steps.append(current_step)
        pbar.set_description(f'Train Epoch: {epoch} [{batch_idx * len(data)} / {len(train_loader)}]')

    # Measure time after epoch ends
    epoch_time = time.time() - start_time # Calculate time taken for the epoch
    print(f'Epoch {epoch+1} completed in {epoch_time:.2f} seconds.')

    # Test after training
    model.eval() # Set model to evaluation mode
    test_loss = 0
    correct = 0

    with torch.no_grad():
        pbar = tqdm(test_loader, total=len(test_loader), desc="Testing...")

        for data, target in pbar:
            # Move data and target to GPU
            data = data.cuda()
            target = target.cuda()

            # Forward pass
            output = model(data)

```

2022-11-11

- Added more dark editor themes for Monaco (when in dark mode, "Editor colorization" appears as an option in the Editor tab of the Tools → Settings dialog)
- Fixed bug where collapsed forms were deleted on mobile [GitHub issue](#)
- Python package updates:
 - ipython from 3.4.0 to 3.5.5 ([GitHub issue](#))
 - notebook from 5.5.0 to 5.7.16
 - tornado from 5.1.1 to 6.0.4
 - tensorflow-probability from 0.16.0 to 0.17.0
 - pandas-gbq from 0.13.3 to 0.17.9
 - protobuf from 3.17.3 to 3.19.6
 - google-api-core[grpc] from 1.31.5 to 2.8.2
 - google-cloud-bigquery from 1.21.0 to 3.3.5
 - google-cloud-core from 1.0.1 to 2.3.2
 - google-cloud-datastore from 1.8.0 to 2.9.0
 - google-cloud-firestore from 1.7.0 to 2.7.2
 - google-cloud-language from 1.2.0 to 2.6.1
 - google-cloud-storage from 1.18.0 to 2.5.0
 - google-cloud-translate from 1.5.0 to 3.8.4

2022-10-21

- Launched a single-click way to get from BigQuery to Colab to further explore query results ([announcement](#))
- Launched [Pro, Pro+, and Pay As You Go](#) to 19 additional countries: Austria, Belgium, Bulgaria, Croatia, Cyprus, Czechia, Denmark, Estonia, Finland, Greece, Hungary, Latvia, Lithuania, Norway, Portugal, Romania, Slovakia, Slovenia, and Sweden ([tweet](#))
- Updated jax from 0.3.17 to 0.3.23, jaxlib from 0.3.15 to 0.3.22, TensorFlow from 2.8.2 to 2.9.2, CUDA from 11.1 to 11.2, and cuDNN from 8.0 to 8.1 ([backend-info](#))
- Added a read-only option to [drive.mount](#)
- Fixed bug where Xarray was not working ([GitHub issue](#))
- Modified Markdown parsing to ignore block quote symbol within MathJax ([GitHub issue](#))

2022-09-30

- Launched [Pay As You Go](#), allowing premium GPU access without requiring a subscription
- Added vim and tcllib to our runtime image
- Fixed bug where open files were closed on kernel disconnect ([GitHub issue](#))
- Fixed bug where the play button/execution indicator was not clickable when scrolled into the cell output ([GitHub issue](#))
- Updated the styling for form titles so that they avoid obscuring the code editor
- Created a GitHub repo, [backend-info](#), with the latest apt-list.txt and pip-freeze.txt files for the Colab runtime ([GitHub issue](#))
- Added [files.upload_file\(filename\)](#) to upload a file from the browser to the runtime with a specified filename

2022-09-16

- Upgraded pymc from 3.11.0 to 4.1.4, jax from 0.3.14 to 0.3.17, jaxlib from 0.3.14 to 0.3.15, fsspec from 2022.8.1 to 2022.8.2
- Modified our save flow to avoid persisting Drive filenames as titles in notebook JSON
- Updated our [Terms of Service](#)
- Modified the Jump to Cell command to locate the cursor at the end of the command palette input (Jump to cell in Tools → Command palette in a notebook with section headings)
- Updated the styling of the Drive notebook comment UI
- Added support for terminating your runtime from code: python from google.colab import runtime runtime.unassign()
- Added regex filter support to the Recent Notebooks dialog
- Inline google.colab.files.upload JS to fix files.upload() not working ([GitHub issue](#))

2022-08-26

```

test_loss += criterion(output, target).item()

# Compute accuracy
pred = output.argmax(dim=1, keepdim=True)
correct += pred.eq(target.view_as(pred)).sum().item()

test_loss /= len(test_loader)
test_losses.append(test_loss)
test_accuracy.append(correct / len(test_loader.dataset))
test_steps.append(current_step)

print(f'\nTest set: Average loss: {test_loss:.4f}, Accuracy: {correct}/{len(test
    f' ({100. * correct / len(test_loader.dataset):.0f}%)')

# Calculate the total time for the last 10 epochs (epoch 10 to 19)
total_time = time.time() - start_time
print(f'\nTotal time for 10 epochs (with testing): {total_time:.2f} seconds.')

```

```

Train Epoch: 10 [57600/60000 (96%)]    Loss: 0.057679: 100%|██████████| 938/
Epoch 11 completed in 15.73 seconds.
Testing...: 100%|██████████| 157/157 [00:02<00:00, 75.83it/s]

Test set: Average loss: 0.0786, Accuracy: 9750/10000 (98%)

Train Epoch: 11 [57600/60000 (96%)]    Loss: 0.053418: 100%|██████████| 938/
Epoch 12 completed in 13.98 seconds.
Testing...: 100%|██████████| 157/157 [00:02<00:00, 75.08it/s]

Test set: Average loss: 0.0779, Accuracy: 9761/10000 (98%)

Train Epoch: 12 [57600/60000 (96%)]    Loss: 0.035589: 100%|██████████| 938/
Epoch 13 completed in 14.14 seconds.
Testing...: 100%|██████████| 157/157 [00:02<00:00, 55.18it/s]

Test set: Average loss: 0.0752, Accuracy: 9767/10000 (98%)

Train Epoch: 13 [57600/60000 (96%)]    Loss: 0.079151: 100%|██████████| 938/
Epoch 14 completed in 14.17 seconds.
Testing...: 100%|██████████| 157/157 [00:02<00:00, 76.81it/s]

Test set: Average loss: 0.0734, Accuracy: 9767/10000 (98%)

Train Epoch: 14 [57600/60000 (96%)]    Loss: 0.016529: 100%|██████████| 938/
Epoch 15 completed in 14.16 seconds.
Testing...: 100%|██████████| 157/157 [00:02<00:00, 73.96it/s]

Test set: Average loss: 0.0726, Accuracy: 9769/10000 (98%)

Train Epoch: 15 [57600/60000 (96%)]    Loss: 0.021297: 100%|██████████| 938/
Epoch 16 completed in 14.10 seconds.
Testing...: 100%|██████████| 157/157 [00:02<00:00, 56.17it/s]

Test set: Average loss: 0.0708, Accuracy: 9780/10000 (98%)

Train Epoch: 16 [57600/60000 (96%)]    Loss: 0.010828: 100%|██████████| 938/
Epoch 17 completed in 14.15 seconds.
Testing...: 100%|██████████| 157/157 [00:02<00:00, 77.47it/s]

Test set: Average loss: 0.0698, Accuracy: 9775/10000 (98%)

Train Epoch: 17 [57600/60000 (96%)]    Loss: 0.022888: 100%|██████████| 938/
Epoch 18 completed in 13.81 seconds.
Testing...: 100%|██████████| 157/157 [00:02<00:00, 75.84it/s]

Test set: Average loss: 0.0702, Accuracy: 9777/10000 (98%)

Train Epoch: 18 [57600/60000 (96%)]    Loss: 0.051657: 100%|██████████| 938/
Epoch 19 completed in 16.42 seconds.
Testing...: 100%|██████████| 157/157 [00:03<00:00, 50.94it/s]

Test set: Average loss: 0.0703, Accuracy: 9773/10000 (98%)

Train Epoch: 19 [57600/60000 (96%)]    Loss: 0.029705: 100%|██████████| 938/
Epoch 20 completed in 16.43 seconds.
Testing...: 100%|██████████| 157/157 [00:02<00:00, 57.26it/s]

```

Now, copy your previous training code with the timing parameters below. It needs to be slightly modified to move everything to the GPU.

Before the line `output = model(data)`, add:

- Upgraded PyYAML from 3.13 to 6.0 ([GitHub issue](#)), drivefs from 61.0.3 to 62.0.1
- Upgraded TensorFlow from 2.8.2 to 2.9.1 and ipywidgets from 7.7.1 to 8.0.1 but rolled both back due to a number of user reports ([GitHub issue](#), [GitHub issue](#))
- Stop persisting inferred titles in notebook JSON ([GitHub issue](#))
- Fix bug in background execution which affected some Pro+ users ([GitHub issue](#))
- Fix bug where Download as .py incorrectly handled text cells ending in a double quote
- Fix bug for Pro and Pro+ users where we weren't honoring the preference (Tools → Settings) to use a temporary scratch notebook as the default landing page
- Provide undo/redo for scratch cells
- When writing ipynb files, serialize empty multiline strings as `[]` for better consistency with JupyterLab

2022-08-11

- Upgraded ipython from 5.5.0 to 7.9.0, fbprophet 0.7 to prophet 1.1, tensorflow-datasets from 4.0.1 to 4.6.0, drivefs from 60.0.2 to 61.0.3, pytorch from 1.12.0 to 1.12.1, numba from 0.51 to 0.56, and lxml from 4.2.0 to 4.9.1
- Loosened our requests version requirement ([GitHub issue](#))
- Removed support for TensorFlow 1
- Added Help → Report Drive abuse for Drive notebooks
- Fixed indentation for Python lines ending in `[`
- Modified styling of tables in Markdown to left-align them rather than centering them
- Fixed special character replacement when copying interactive tables as Markdown
- Fixed ansi 8-bit color parsing ([GitHub issue](#))
- Configured logging to preempt transitive imports and other loading from implicitly configuring the root logger
- Modified forms to use a value of `None` instead of causing a parse error when clearing raw and numeric-typed form fields

2022-07-22

- Update scipy from 1.4.1 to 1.7.3, drivefs from 59.0.3 to 60.0.2, pytorch from 1.11 to 1.12, jax & jaxlib from 0.3.8 to 0.3.14, opencv-python from 4.1.2.30 to 4.6.0.66, spaCy from 3.3.1 to 3.4.0, and dlib from 19.18.0 to 19.24.0
- Fix Open in tab doc link which was rendering incorrectly ([GitHub issue](#))
- Add a preference for the default tab orientation to the Site section of the settings menu under Tools → Settings
- Show a warning for USE_AUTH_EPHEM usage when running `authenticate_user` on a TPU runtime ([code](#))

2022-07-01

- Add a preference for code font to the settings menu under Tools → Settings
- Update drivefs from 58.0.3 to 59.0.3 and spacy from 2.2.4 to 3.3.1
- Allow [display_data](#) and [execute_result](#) text outputs to wrap, matching behavior of JupyterLab (does not affect stream outputs/print statements).
- Improve LSP handling of some magics, esp. `%%writefile` ([GitHub issue](#)).
- Add a [FAQ entry](#) about the mount Drive button behavior and include link buttons for each FAQ entry.
- Fix bug where the notebook was sometimes hidden behind other tabs on load when in single pane view.
- Fix issue with inconsistent scrolling when an editor is in multi-select mode.
- Fix bug where clicking on a link in a form would navigate away from the notebook
- Show a confirmation dialog before performing Replace all from the Find and replace pane.

2022-06-10


```
data = data.cuda()
target = target.cuda()
```

Note that this is needed in both the train and test functions.

Question 5

Please edit the cell below to show the new GPU train and test functions.

```
# the new GPU training functions
import time
from tqdm import tqdm

def gpu_train(epoch, train_losses, steps, current_step, model, train_loader, optimizer):
    # Set the model in training mode
    model.train()

    # Create tqdm progress bar
    pbar = tqdm(enumerate(train_loader), total=len(train_loader), desc=f"Training Epoch {epoch}")

    for batch_idx, (data, target) in pbar:
        # Move data and target to GPU
        data = data.cuda()
        target = target.cuda()

        # Zero the gradients before the backpropagation
        optimizer.zero_grad()

        # Forward pass
        output = model(data)

        # Compute loss
        loss = criterion(output, target)

        # Backpropagation
        loss.backward()

        # Apply gradients
        optimizer.step()

        # Increment global step count
        current_step += 1

    # Update progress bar and log training loss
    if batch_idx % 100 == 0:
        train_losses.append(loss.item())
        steps.append(current_step)
        pbar.set_description(f"Train Epoch: {epoch} [{batch_idx * len(data)} / {len(train_loader)}] \t loss: {loss.item():.4f} \t steps: {current_step}")

    return current_step
```

```
# new GPU training for 10 epochs
def gpu_test(test_losses, test_accuracy, steps, current_step, model, test_loader, criterion):
    # Set the model in evaluation mode
    model.eval()
    test_loss = 0
    correct = 0

    # Create tqdm progress bar for testing
    pbar = tqdm(test_loader, total=len(test_loader), desc="Testing...")

    # No need for gradients during testing
    with torch.no_grad():
        for data, target in pbar:
            # Move data and target to GPU
            data = data.cuda()
            target = target.cuda()

            # Forward pass
            output = model(data)
            test_loss += criterion(output, target).item()

            # Compute accuracy by getting the index of the max output
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()
```

- Update drivefs from 57.0.5 to 58.0.3 and tensorflow from 2.8.0 to 2.8.2
- Support more than 100 repos in the GitHub repo selector shown in the open dialog and the clone to GitHub dialog
- Show full notebook names on hover in the open dialog
- Improve the color contrast for links, buttons, and the ipywidgets.Accordion widget in dark mode

2022-05-20

- Support URL params for linking to some common pref settings: [force_theme=dark](#), [force_corgi_mode=1](#), [force_font_size=14](#). Params forced by URL are not persisted unless saved using Tools → Settings.
- Add a class markdown-google-sans to allow Markdown to render in Google Sans
- Update monaco-vim from 0.1.19 to 0.3.4
- Update drivefs from 55.0.3 to 57.0.5, jax from 0.3.4 to 0.3.8, and jaxlib from 0.3.2 to 0.3.7

2022-04-29

- Added 🌶️ mode (under Miscellaneous in Tools → Settings)
- Added "Disconnect and delete runtime" option to the menu next to the Connect button
- Improved rendering of filter options in an interactive table
- Added git-lfs to the base image
- Updated torch from 1.10.0 to 1.11.0, jupyter-core from 4.9.2 to 4.10.0, and cmake from 3.12.0 to 3.22.3
- Added more details to our [FAQ](#) about unsupported uses (using proxies, downloading torrents, etc.)
- Fixed [issue](#) with apt-get dependencies

2022-04-15

- Add an option in the file browser to show hidden files.
- Upgrade gdown from 4.2.0 to 4.4.0, google-api-core[grpc] from 1.26.0 to 1.31.5, and pytz from 2018.4 to 2022.1

2022-03-25

- Launched [Pro/Pro+](#) to 12 additional countries: Australia, Bangladesh, Colombia, Hong Kong, Indonesia, Mexico, New Zealand, Pakistan, Philippines, Singapore, Taiwan, and Vietnam
- Added [google.colab.auth.authenticate_service_account](#) to support using [Service Account keys](#)
- Update jax from 0.3.1 to 0.3.4 & jaxlib from 0.3.0 to 0.3.2
- Fixed an issue with Twitter previews of notebooks shared as GitHub Gists

2022-03-10

- Launched [Pro/Pro+](#) to 10 new countries: Ireland, Israel, Italy, Morocco, the Netherlands, Poland, Spain, Switzerland, Turkey, and the United Arab Emirates
- Launched support for [scheduling notebooks for Pro+ users](#)
- Fixed bug in interactive datatables where filtering by number did not work
- Finished removing the python2 kernelspec

2022-02-25

- Made various accessibility improvements to the header
- Fix bug with [forms run:auto](#) where a form field change would trigger multiple runs
- Minor updates to the [bigquery example notebook](#) and snippet
- Include background execution setting in the sessions dialog for Pro+ users
- Update tensorflow-probability from 0.15 to 0.16
- Update jax from 0.2.25 to 0.3.1 & jaxlib from 0.1.71 to 0.3.0

2022-02-11

- Improve keyboard navigation for the open dialog

```
# Average test loss
test_loss /= len(test_loader)
test_losses.append(test_loss)

# Calculate accuracy
accuracy = 100. * correct / len(test_loader.dataset)
test_accuracy.append(accuracy)

steps.append(current_step)

print(f'\nTest set: Average loss: {test_loss:.4f}, Accuracy: {correct}/{len(test_loader.dataset)} ({accuracy:.0f}%)')
```

Question 6

Is training faster now that it is on a GPU? Is the speedup what you would expect? Why or why not? Edit the cell below to answer.

Yes, Without GPU 20 epochs completed in 17.49 secs where as with GPU 20 epochs completed in 14.18 secs

✓ Another Model Type: CNN

Until now you have trained a simple MLP for MNIST classification, however, MLPs are not a particularly good for images.

Firstly, using a MLP will require that all images have the same size and shape, since they are unrolled in the input.

Secondly, in general images can make use of translation invariance (a type of data symmetry), but this cannot but leveraged with a MLP.

For these reasons, a convolutional network is more appropriate, as it will pass kernels over the 2D image, removing the requirement for a fixed image size and leveraging the translation invariance of the 2D images.

Let's define a simple CNN below.

```
# Define the CNN model
class CNN(nn.Module):
    # define the constructor for the network
    def __init__(self):
        super().__init__()
        # instead of declaring the layers independently, let's use the nn.Sequential
        # these blocks will be executed in list order

        # you will break up the model into two parts:
        # 1) the convolutional network
        # 2) the prediction head (a small MLP)

        # the convolutional network
        self.net = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1), # the input projection
            nn.ReLU(), # activation
            nn.Conv2d(32, 64, kernel_size=3, stride=2, padding=1), # an inner layer -
            nn.ReLU(), # activation
            nn.Conv2d(64, 128, kernel_size=3, stride=2, padding=1), # an inner layer -
            nn.ReLU(), # activation
            nn.AdaptiveMaxPool2d(1), # a pooling layer w

        )

        # the prediction head
        self.head = nn.Sequential(
            nn.Linear(128, 64), # input projection, the output from the pool layer
            nn.ReLU(), # activation
            nn.Linear(64, 10) # class projection to one of the 10 classes (digits)
        )

    # define the forward pass compute graph
    def forward(self, x):
```

- Fix issue where nvidia-smi stopped reporting resource utilization for some users who were modifying the version of nvidia used
- Update tensorflow from 2.7 to 2.8, keras from 2.7 to 2.8, numpy from 1.19.5 to 1.21.5, tables from 3.4.4 to 3.7.0

2022-02-04

- Improve UX for opening content alongside your notebook, such as files opened from the file browser. This includes a multi-pane view and drag-drop support
- Better Twitter previews when sharing example Colab notebooks and notebooks opened from GitHub Gists
- Update pandas from 1.1.5 to 1.3.5
- Update openpyxl from 2.5.9 to 3.0.0 and pyarrow from 3.0.0 to 6.0.0
- Link to the release notes from the Help menu

2022-01-28

- Add a copy button to [data tables](#)
- Python LSP support for better completions and code diagnostics. This can be configured in the Editor Settings (Tools → Settings)
- Update [gspread examples](#) in our documentation
- Update gdown from 3.6 to 4.2

2022-01-21

- New documentation for the [google.colab package](#)
- Show GPU RAM in the resource usage tab
- Improved security for mounting Google Drive which disallows mounting Drive from accounts other than the one currently executing the notebook

2022-01-14

- Add a preference (Tools → Settings) to use a temporary scratch notebook as the default landing page
- Fix bug where / and : weren't working in VIM mode
- Update gspread from 3.0 to 3.4
- Update the [Colab Marketplace VM image](#)

```
# pass the input through the convolution network
x = self.net(x)

# reshape the output from Bx128x1x1 to Bx128
x = x.view(x.size(0), -1)

# pass the pooled vector into the prediction head
x = self.head(x)

# the output here is Bx10
return x
```

```
# create the model
model = CNN()

# print the model and the parameter count
print(model)
param_count = sum([p.numel() for p in model.parameters()])
print(f"Model has {param_count:,} trainable parameters")

# the loss function
criterion = nn.CrossEntropyLoss()

# then you can instantiate the optimizer. You will use Stochastic Gradient Descent (SGD)
# momentum factor of 0.5
# the first input to the optimizer is the list of model parameters, which is obtained from model.parameters()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)
```

```
↗ CNN(
  (net): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (3): ReLU()
    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (5): ReLU()
    (6): AdaptiveMaxPool2d(output_size=1)
  )
  (head): Sequential(
    (0): Linear(in_features=128, out_features=64, bias=True)
    (1): ReLU()
    (2): Linear(in_features=64, out_features=10, bias=True)
  )
)
Model has 101,578 trainable parameters
```

Question 7

Notice that this model now has fewer parameters than the MLP. Let's see how it trains.

Using the previous code to train on the CPU with timing, edit the cell below to execute 2 epochs of training.

```
# create a new array to log the loss and accuracy
train_losses = []
train_steps = []
test_steps = []
test_losses = []
test_accuracy = []
current_step = 0 # Start with global step 0
current_epoch = 0 # Start with epoch 0
```

```
# train for 2 epochs on the CPU
import time

# create a new array to log the loss and accuracy
train_losses = []
train_steps = []
test_steps = []
test_losses = []
test_accuracy = []
current_step = 0 # Start with global step 0
current_epoch = 0 # Start with epoch 0

# Train for 2 epochs with timing
for epoch in range(0, 2):
    start_time = time.time() # Start timing the epoch

    current_step = cpu_train(current_epoch, train_losses, train_steps, current_step,
```

```

cpu_test(test_losses, test_accuracy, test_steps, current_step)

epoch_time = time.time() - start_time # Calculate the time taken for this epoch
print(f"Epoch {epoch + 1} took {epoch_time:.2f} seconds.")

current_epoch += 1

```

Train Epoch: 0 [57600/60000 (96%)] Loss: 0.493148: 100% [██████████] 938/938
 Testing...: 100% [██████████] 157/157 [00:06<00:00, 25.21it/s]

Test set: Average loss: 0.5423, Accuracy: 8207/10000 (82%)

Epoch 1 took 83.23 seconds.

Train Epoch: 1 [57600/60000 (96%)] Loss: 0.208951: 100% [██████████] 938/938
 Testing...: 100% [██████████] 157/157 [00:06<00:00, 22.49it/s]
 Test set: Average loss: 0.2933, Accuracy: 9084/10000 (91%)

Epoch 2 took 82.56 seconds.

Question 8

Now, let's move the model to the GPU and try training for 2 epochs there.

```

# create the model
model = CNN()

model.cuda()

# print the model and the parameter count
print(model)
param_count = sum([p.numel() for p in model.parameters()])
print(f"Model has {param_count:,} trainable parameters")

# the loss function
criterion = nn.CrossEntropyLoss()

# then you can instantiate the optimizer. You will use Stochastic Gradient Descent (SGD)
# the first input to the optimizer is the list of model parameters, which is obtained from model.parameters()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)

```

```

CNN(
  (net): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (3): ReLU()
    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (5): ReLU()
    (6): AdaptiveMaxPool2d(output_size=1)
  )
  (head): Sequential(
    (0): Linear(in_features=128, out_features=64, bias=True)
    (1): ReLU()
    (2): Linear(in_features=64, out_features=10, bias=True)
  )
)
Model has 101,578 trainable parameters

```

```

# create a new array to log the loss and accuracy
train_losses = []
train_steps = []
test_steps = []
test_losses = []
test_accuracy = []
current_step = 0 # Start with global step 0
current_epoch = 0 # Start with epoch 0

```

```

# train for 2 epochs on the GPU
def gpu_train(epoch, train_losses, steps, current_step):
    model.train()
    pbar = tqdm(enumerate(train_loader), total=len(train_loader))

    for batch_idx, (data, target) in pbar:
        data, target = data.cuda(), target.cuda() # Move data and target to GPU

        optimizer.zero_grad()

```

```

    output = model(data)
    loss = criterion(output, target)

    loss.backward()
    optimizer.step()

    current_step += 1

    if batch_idx % 100 == 0:
        train_losses.append(loss.item())
        steps.append(current_step)

        desc = (f'Train Epoch: {epoch} [{batch_idx * len(data)} / {len(train_loader)}]
                f' ({100. * batch_idx / len(train_loader):.0f}%) \t Loss: {loss:.4f} \t '
                f' {100. * current_step / len(train_loader.dataset):.0f}%)')
        pbar.set_description(desc)

    return current_step

def gpu_test(test_losses, test_accuracy, steps, current_step):
    model.eval() # Set the model to evaluation mode
    test_loss = 0
    correct = 0

    # Create tqdm progress bar
    pbar = tqdm(test_loader, total=len(test_loader), desc="Testing...")

    # Use no_grad() as we do not need gradients for evaluation
    with torch.no_grad():
        for data, target in pbar:
            # Move data and target to the GPU
            data, target = data.cuda(), target.cuda()

            # Forward pass through the model
            output = model(data)
            test_loss += criterion(output, target).item() # Add the loss for this batch

            # Calculate the number of correct predictions
            pred = output.argmax(dim=1, keepdim=True) # Get the index of the max logit
            correct += pred.eq(target.view_as(pred)).sum().item() # Compare predicted vs. target

    test_loss /= len(test_loader) # Calculate the average test loss

    # Log the test loss and accuracy
    test_losses.append(test_loss)
    test_accuracy.append(correct / len(test_loader.dataset)) # Accuracy as a fraction
    steps.append(current_step)

    print(f'\nTest set: Average loss: {test_loss:.4f}, Accuracy: {correct}/{len(test_loader.dataset)} ({100. * correct / len(test_loader.dataset):.0f}%) \n')

# Train for 2 epochs with timing on GPU
for epoch in range(0, 2):
    start_time = time.time() # Start timing the epoch

    current_step = gpu_train(current_epoch, train_losses, train_steps, current_step)
    gpu_test(test_losses, test_accuracy, test_steps, current_step) # Use GPU version of test

    epoch_time = time.time() - start_time # Calculate the time taken for this epoch
    print(f"Epoch {epoch + 1} took {epoch_time:.2f} seconds.")

    current_epoch += 1

```

```

Train Epoch: 0 [57600/60000 (96%)]      Loss: 0.412111: 100%|██████████| 938/938
Testing....: 100%|██████████| 157/157 [00:02<00:00, 68.91it/s]

```

```
Test set: Average loss: 0.2596, Accuracy: 9204/10000 (92%)
```

```
Epoch 1 took 17.35 seconds.
```

```

Train Epoch: 1 [57600/60000 (96%)]      Loss: 0.155251: 100%|██████████| 938/938
Testing....: 100%|██████████| 157/157 [00:02<00:00, 59.29it/s]

```

```
Test set: Average loss: 0.1737, Accuracy: 9457/10000 (95%)
```

```
Epoch 2 took 18.20 seconds.
```

Question 9

How do the CPU and GPU versions compare for the CNN? Is one faster than the other? Why do you think this is, and how does it differ from the MLP? Edit the cell below to answer.

For CPU version Epoch 1 took 83.23 seconds and Epoch 2 took 82.56 seconds.

For GPU version Epoch 1 took 17.35 seconds and Epoch 2 took 18.20 seconds.

The GPU version is faster for several reasons:

1. Parallelization: GPU: GPUs are designed to perform many operations simultaneously. They excel at handling parallel tasks, especially for operations like matrix multiplications, convolutions, and other tensor-based operations, which are common in deep learning. The large number of cores in GPUs allows them to execute many computations at once, significantly speeding up tasks such as training and inference in deep neural networks. CPU: The CPU has fewer cores (typically 4 to 16), which makes it more suited for single-threaded tasks, and while it can execute parallel operations, it does so with far fewer resources compared to a GPU. This limits its ability to efficiently handle large-scale operations like those required in deep learning.
2. CNN vs MLP (Multi-Layer Perceptron): MLP: A Multi-Layer Perceptron is a fully connected network where each neuron in a layer is connected to every neuron in the next layer. This creates a large number of parameters, and each input feature is multiplied by all weights in the layer, making the computation intensive, especially for large datasets like images. While the computations themselves can be parallelized, the dense connections in an MLP can result in fewer advantages from GPU acceleration compared to a convolutional approach. CNN: A Convolutional Neural Network (CNN) is specifically designed for image data and utilizes convolutional layers that apply filters (kernels) across the input image. These convolutional operations are highly parallelizable because: The same filter is applied repeatedly across different parts of the image, allowing for efficient computation of feature maps. Convolutions are localized, meaning fewer parameters need to be learned per filter, which makes the process less computationally expensive than an MLP. CNNs often have pooling layers that downsample the feature maps, reducing the size of the data and computation required in later layers.

Key Differences in Operation: MLP requires flattening the entire image into a vector before feeding it into the network. This means the input image size must be fixed, and every pixel is connected to every node in the network. For large images, this can result in a huge number of weights and computations.

CNN works directly with the 2D structure of the image, applying filters across small regions (local receptive fields) to capture spatial hierarchies in the data. The convolution operation, combined with pooling, reduces the computational complexity by reducing the number of connections and focusing on local features first before moving on to higher-level features.

Why CNNs Are Faster: CNNs exploit the spatial structure of images by applying filters locally, which reduces the number of parameters compared to MLPs. CNNs benefit greatly from GPUs because convolutions and matrix multiplications can be performed in parallel, taking full advantage of the GPU's parallel processing capabilities. The use of pooling layers in CNNs reduces the data size, meaning fewer computations are needed as the network progresses.

In summary, the GPU is faster because it can handle parallel operations more efficiently, and CNNs are faster than MLPs on image data due to their local, sparse connections and the ability to exploit spatial hierarchies, making them more computationally efficient for tasks like image classification.


As a final comparison, you can profile the FLOPs (floating-point operations) executed by each model. You will use the `thop.profile` function for this and consider an MNIST batch size of 1.

```
# the input shape of a MNIST sample with batch_size = 1
input = torch.randn(1, 1, 28, 28)
```

```
# create a copy of the models on the CPU
mlp_model = MLP()
cnn_model = CNN()

# profile the MLP
flops, params = thop.profile(mlp_model, inputs=(input, ), verbose=False)
print(f"MLP has {params:}, params and uses {flops:}, FLOPs")

# profile the CNN
flops, params = thop.profile(cnn_model, inputs=(input, ), verbose=False)
print(f"CNN has {params:}, params and uses {flops:}, FLOPs")
```

 MLP has 109,386.0 params and uses 109,184.0 FLOPs
CNN has 101,578.0 params and uses 7,459,968.0 FLOPs

Start coding or [generate](#) with AI.

Question 10

Are these results what you would have expected? Do they explain the performance difference between running on the CPU and GPU? Why or why not? Edit the cell below to answer.

Even though the CNN has fewer parameters than the MLP, it has a significantly higher number of FLOPs. This is because convolutions require computing the dot product of filters with patches of the input image. These operations are computationally expensive due to the number of times the kernel is applied across the entire image.

Additionally, CNNs often perform more advanced operations like pooling, activation functions, and downsampling, all of which contribute to the high FLOPs count. However, the CNN has fewer parameters because it shares weights across the entire image, making it more efficient in terms of parameterization.