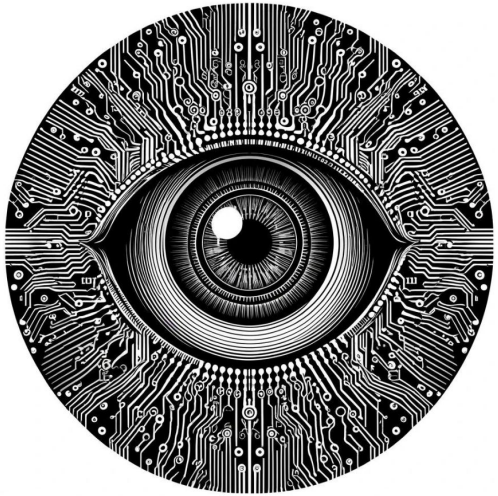
KI Service Zentrum
by Hasso-Plattner-Institut
HPI

PyTorch Datasets and Dataloaders



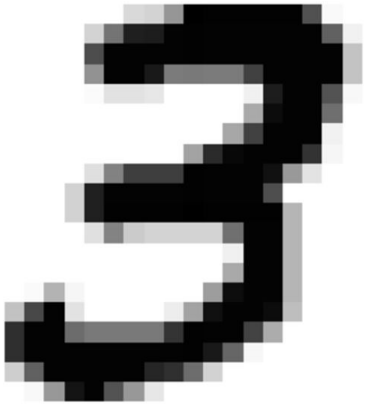
Antonio Rueda-Toicen

SPONSORED BY THE
Federal Ministry
of Education
and Research

Learning goals

- Create `Dataset` objects in PyTorch to wrap images and target values together
- Implement `DataLoader` PyTorch objects to feed data to a model
- Understand the connection between `DataLoader` and Stochastic Gradient Descent

The batch dimension is important for our models



`torch` image tensors follow the format: `[N, C, H, W]` where:

- `N` = batch size (number of images)
- `C` = channels (e.g., 1 for grayscale, 3 for RGB)
- `H` = height in pixels
- `W` = width in pixels

```
print(torch_tensor_gray.shape)

# torch.Size([1, 1, 28, 28])
```

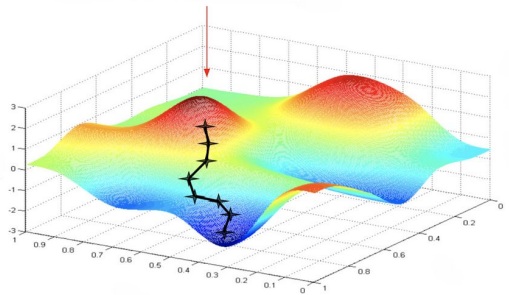
Feeding `torch` tensors to a PyTorch model

```
# Define the train loader with batch size and shuffling
batch_size = 32
train_loader = DataLoader(dataset=train_dataset,
                          batch_size=batch_size,
                          shuffle=True)

# Move model to device (GPU/CPU)
model = model.to(device)

# Training loop
for images, labels in train_loader:
    # Remember that data has to be explicitly sent to the GPU
    images, labels = images.to(device), labels.to(device)
    output = model(images)
    batch_loss = loss_function(output, labels)
```

Tip: use the nvidia-smi bash command to check the RAM available in your GPU

Shuffling the training set only

```
# Create data loaders
batch_size = 32
# Notice that we shuffle the training loader, but not the validation or test loaders.
# This practice of shuffling the training set is one of the techniques
# that have been shown to improve training.
train_loader = DataLoader(train_subset, batch_size=batch_size, shuffle=True)
valid_loader = DataLoader(valid_subset, batch_size=batch_size, shuffle=False)
test_loader  = DataLoader(test_set,     batch_size=batch_size, shuffle=False)
```



$$w_{ij} = w_{ij} - (\text{learning rate} * \frac{dL}{dw_{ij}})$$

Dataloader depends on Dataset

```
# Define the MNIST Dataset class as a subclass of Dataset
class MNISTDataset(Dataset):
    # Content of the dataset
    # gets called on dataset_instance = MNISTDataset(<params>)
    def __init__(self, dataframe, labels=True, transform=None):
        …
    def __len__(self):
        # What gets called on len(dataset_instance)
        return len(self.data)

    def __getitem__(self, idx):
        # What gets called on dataset_instance[index]
        …
        return image_tensor, label_tensor
```

Practice creating a `Dataset`

Submit Prediction

KAGGLE · GETTING STARTED PREDICTION COMPETITION · ONGOING

Digit Recognizer

Learn computer vision fundamentals with the famous MNIST data

https://www.kaggle.com/competitions/digit-recognizer

Summary

Dataset objects wrap images and labels together

- Provide a standard way to access data through `__getitem__` and `__len__` methods

Dataloaders split datasets in batches

- Dataloaders shuffle the training data and maintain sequential order for validation and test sets

Dataloaders are fundamental to implement Stochastic Gradient Descent

- Their random sampling and shuffling of training samples provide the ‘stochastic’ part of Stochastic Gradient Descent

SPONSORED BY THE
Federal Ministry of Education and Research

References

Datasets and Dataloaders

- https://pytorch.org/tutorials/beginner/basics/data_tutorial.html

MNIST on PyTorch's datasets

- https://pytorch.org/vision/0.20/generated/torchvision.datasets.MNIST.html

MNIST on Fiftyone

- https://try.fiftyone.ai/datasets/mnist/samples

SPONSORED BY THE
Federal Ministry of Education and Research