OPTIMIZING MODEL PARAMETERS

Now that we have a model and data it’s time to train, validate and test our model by optimizing its parameters on our data. Training a model is an iterative process; in each iteration (called an *epoch*) the model makes a guess about the output, calculates the error in its guess (*loss*), collects the derivatives of the error with respect to its parameters (as we saw in the [previous section](https://pytorch.org/tutorials/beginner/basics/autograd_tutorial.html)), and **optimizes** these parameters using gradient descent. For a more detailed walkthrough of this process, check out this video on [backpropagation from 3Blue1Brown](https://www.youtube.com/watch?v=tIeHLnjs5U8).

Prerequisite Code

We load the code from the previous sections on [Datasets & DataLoaders](https://pytorch.org/tutorials/beginner/basics/data_tutorial.html) and [Build Model](https://pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html).

**import** torch

**from** torch **import** **nn**

**from** torch.utils.data **import** **DataLoader**

**from** torchvision **import** **datasets**

**from** torchvision.transforms **import** **ToTensor,** **Lambda**

**training\_data** **=** **datasets.FashionMNIST(**

**root=**"data"**,**

**train=True,**

**download=True,**

**transform=ToTensor()**

**)**

**test\_data** **=** **datasets.FashionMNIST(**

**root=**"data"**,**

**train=False,**

**download=True,**

**transform=ToTensor()**

**)**

**train\_dataloader** **=** **DataLoader(training\_data,** **batch\_size=**64**)**

**test\_dataloader** **=** **DataLoader(test\_data,** **batch\_size=**64**)**

**class** **NeuralNetwork(nn.Module):**

**def** \_\_init\_\_**(**self**):**

super**(NeuralNetwork,** self**).**\_\_init\_\_**()**

self**.flatten** **=** **nn.Flatten()**

self**.linear\_relu\_stack** **=** **nn.Sequential(**

**nn.Linear(**28**\***28**,** 512**),**

**nn.ReLU(),**

**nn.Linear(**512**,** 512**),**

**nn.ReLU(),**

**nn.Linear(**512**,** 10**),**

**)**

**def** **forward(**self**,** **x):**

**x** **=** self**.flatten(x)**

**logits** **=** self**.linear\_relu\_stack(x)**

**return** **logits**

**model** **=** **NeuralNetwork()**

Out:

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz to data/FashionMNIST/raw/train-images-idx3-ubyte.gz

Extracting data/FashionMNIST/raw/train-images-idx3-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz to data/FashionMNIST/raw/train-labels-idx1-ubyte.gz

Extracting data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz

Extracting data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz

Extracting data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to data/FashionMNIST/raw

Hyperparameters

Hyperparameters are adjustable parameters that let you control the model optimization process. Different hyperparameter values can impact model training and convergence rates ([read more](https://pytorch.org/tutorials/beginner/hyperparameter_tuning_tutorial.html) about hyperparameter tuning)

We define the following hyperparameters for training:

* **Number of Epochs** - the number times to iterate over the dataset
* **Batch Size** - the number of data samples propagated through the network before the parameters are updated
* **Learning Rate** - how much to update models parameters at each batch/epoch. Smaller values yield slow learning speed, while large values may result in unpredictable behavior during training.

**learning\_rate** **=** 1e-3

**batch\_size** **=** 64

**epochs** **=** 5

Optimization Loop

Once we set our hyperparameters, we can then train and optimize our model with an optimization loop. Each iteration of the optimization loop is called an **epoch**.

Each epoch consists of two main parts:

* **The Train Loop** - iterate over the training dataset and try to converge to optimal parameters.
* **The Validation/Test Loop** - iterate over the test dataset to check if model performance is improving.

Let’s briefly familiarize ourselves with some of the concepts used in the training loop. Jump ahead to see the [Full Implementation](https://pytorch.org/tutorials/beginner/basics/optimization_tutorial.html#full-impl-label) of the optimization loop.

Loss Function

When presented with some training data, our untrained network is likely not to give the correct answer. **Loss function** measures the degree of dissimilarity of obtained result to the target value, and it is the loss function that we want to minimize during training. To calculate the loss we make a prediction using the inputs of our given data sample and compare it against the true data label value.

Common loss functions include [nn.MSELoss](https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html#torch.nn.MSELoss) (Mean Square Error) for regression tasks, and [nn.NLLLoss](https://pytorch.org/docs/stable/generated/torch.nn.NLLLoss.html#torch.nn.NLLLoss) (Negative Log Likelihood) for classification. [nn.CrossEntropyLoss](https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html#torch.nn.CrossEntropyLoss) combines nn.LogSoftmax and nn.NLLLoss.

We pass our model’s output logits to nn.CrossEntropyLoss, which will normalize the logits and compute the prediction error.

*# Initialize the loss function*

**loss\_fn** **=** **nn.CrossEntropyLoss()**

Optimizer

Optimization is the process of adjusting model parameters to reduce model error in each training step. **Optimization algorithms** define how this process is performed (in this example we use Stochastic Gradient Descent). All optimization logic is encapsulated in the optimizer object. Here, we use the SGD optimizer; additionally, there are many [different optimizers](https://pytorch.org/docs/stable/optim.html) available in PyTorch such as ADAM and RMSProp, that work better for different kinds of models and data.

We initialize the optimizer by registering the model’s parameters that need to be trained, and passing in the learning rate hyperparameter.

**optimizer** **=** **torch.optim.SGD(model.parameters(),** **lr=learning\_rate)**

Inside the training loop, optimization happens in three steps:

* Call optimizer.zero\_grad() to reset the gradients of model parameters. Gradients by default add up; to prevent double-counting, we explicitly zero them at each iteration.
* Backpropagate the prediction loss with a call to loss.backwards(). PyTorch deposits the gradients of the loss w.r.t. each parameter.
* Once we have our gradients, we call optimizer.step() to adjust the parameters by the gradients collected in the backward pass.

Full Implementation

We define train\_loop that loops over our optimization code, and test\_loop that evaluates the model’s performance against our test data.

**def** **train\_loop(dataloader,** **model,** **loss\_fn,** **optimizer):**

**size** **=** len**(dataloader.dataset)**

**for** **batch,** **(X,** **y)** **in** enumerate**(dataloader):**

*# Compute prediction and loss*

**pred** **=** **model(X)**

**loss** **=** **loss\_fn(pred,** **y)**

*# Backpropagation*

**optimizer.zero\_grad()**

**loss.backward()**

**optimizer.step()**

**if** **batch** **%** 100 **==** 0**:**

**loss,** **current** **=** **loss.item(),** **batch** **\*** len**(X)**

print**(**f"loss: {**loss**:>7f} [{**current**:>5d}/{**size**:>5d}]"**)**

**def** **test\_loop(dataloader,** **model,** **loss\_fn):**

**size** **=** len**(dataloader.dataset)**

**num\_batches** **=** len**(dataloader)**

**test\_loss,** **correct** **=** 0**,** 0

**with** **torch.no\_grad():**

**for** **X,** **y** **in** **dataloader:**

**pred** **=** **model(X)**

**test\_loss** **+=** **loss\_fn(pred,** **y).item()**

**correct** **+=** **(pred.argmax(**1**)** **==** **y).type(torch.float).sum().item()**

**test\_loss** **/=** **num\_batches**

**correct** **/=** **size**

print**(**f"Test Error: \n Accuracy: {**(**100**\*correct)**:>0.1f}%, Avg loss: {**test\_loss**:>8f} \n"**)**

We initialize the loss function and optimizer, and pass it to train\_loop and test\_loop. Feel free to increase the number of epochs to track the model’s improving performance.

**loss\_fn** **=** **nn.CrossEntropyLoss()**

**optimizer** **=** **torch.optim.SGD(model.parameters(),** **lr=learning\_rate)**

**epochs** **=** 10

**for** **t** **in** range**(epochs):**

print**(**f"Epoch {**t+**1}\n-------------------------------"**)**

**train\_loop(train\_dataloader,** **model,** **loss\_fn,** **optimizer)**

**test\_loop(test\_dataloader,** **model,** **loss\_fn)**

print**(**"Done!"**)**

Out:

Epoch 1

-------------------------------

loss: 2.299019 [ 0/60000]

loss: 2.293434 [ 6400/60000]

loss: 2.272197 [12800/60000]

loss: 2.269407 [19200/60000]

loss: 2.254936 [25600/60000]

loss: 2.212348 [32000/60000]

loss: 2.230545 [38400/60000]

loss: 2.186401 [44800/60000]

loss: 2.180039 [51200/60000]

loss: 2.157638 [57600/60000]

Test Error:

Accuracy: 38.0%, Avg loss: 2.150000

Epoch 2

-------------------------------

loss: 2.155288 [ 0/60000]

loss: 2.152143 [ 6400/60000]

loss: 2.090481 [12800/60000]

loss: 2.113899 [19200/60000]

loss: 2.069696 [25600/60000]

loss: 1.991187 [32000/60000]

loss: 2.032744 [38400/60000]

loss: 1.946663 [44800/60000]

loss: 1.942007 [51200/60000]

loss: 1.876344 [57600/60000]

Test Error:

Accuracy: 52.6%, Avg loss: 1.877733

Epoch 3

-------------------------------

loss: 1.910148 [ 0/60000]

loss: 1.882627 [ 6400/60000]

loss: 1.764288 [12800/60000]

loss: 1.808971 [19200/60000]

loss: 1.698971 [25600/60000]

loss: 1.641954 [32000/60000]

loss: 1.672373 [38400/60000]

loss: 1.572031 [44800/60000]

loss: 1.581498 [51200/60000]

loss: 1.482749 [57600/60000]

Test Error:

Accuracy: 59.2%, Avg loss: 1.505292

Epoch 4

-------------------------------

loss: 1.575709 [ 0/60000]

loss: 1.541534 [ 6400/60000]

loss: 1.392061 [12800/60000]

loss: 1.466007 [19200/60000]

loss: 1.346880 [25600/60000]

loss: 1.335655 [32000/60000]

loss: 1.361143 [38400/60000]

loss: 1.284683 [44800/60000]

loss: 1.302845 [51200/60000]

loss: 1.214065 [57600/60000]

Test Error:

Accuracy: 62.6%, Avg loss: 1.242792

Epoch 5

-------------------------------

loss: 1.321774 [ 0/60000]

loss: 1.304524 [ 6400/60000]

loss: 1.139267 [12800/60000]

loss: 1.248103 [19200/60000]

loss: 1.125497 [25600/60000]

loss: 1.138984 [32000/60000]

loss: 1.173565 [38400/60000]

loss: 1.109160 [44800/60000]

loss: 1.132148 [51200/60000]

loss: 1.057497 [57600/60000]

Test Error:

Accuracy: 64.4%, Avg loss: 1.081785

Epoch 6

-------------------------------

loss: 1.153331 [ 0/60000]

loss: 1.155597 [ 6400/60000]

loss: 0.974328 [12800/60000]

loss: 1.111334 [19200/60000]

loss: 0.989498 [25600/60000]

loss: 1.008430 [32000/60000]

loss: 1.056605 [38400/60000]

loss: 0.997322 [44800/60000]

loss: 1.021251 [51200/60000]

loss: 0.958663 [57600/60000]

Test Error:

Accuracy: 65.9%, Avg loss: 0.977729

Epoch 7

-------------------------------

loss: 1.035878 [ 0/60000]

loss: 1.058628 [ 6400/60000]

loss: 0.861519 [12800/60000]

loss: 1.019441 [19200/60000]

loss: 0.903302 [25600/60000]

loss: 0.916979 [32000/60000]

loss: 0.979479 [38400/60000]

loss: 0.924112 [44800/60000]

loss: 0.944378 [51200/60000]

loss: 0.892103 [57600/60000]

Test Error:

Accuracy: 67.1%, Avg loss: 0.906723

Epoch 8

-------------------------------

loss: 0.949358 [ 0/60000]

loss: 0.990939 [ 6400/60000]

loss: 0.780747 [12800/60000]

loss: 0.954489 [19200/60000]

loss: 0.846069 [25600/60000]

loss: 0.850167 [32000/60000]

loss: 0.924916 [38400/60000]

loss: 0.874231 [44800/60000]

loss: 0.888764 [51200/60000]

loss: 0.843961 [57600/60000]

Test Error:

Accuracy: 68.2%, Avg loss: 0.855560

Epoch 9

-------------------------------

loss: 0.882698 [ 0/60000]

loss: 0.940110 [ 6400/60000]

loss: 0.720155 [12800/60000]

loss: 0.906579 [19200/60000]

loss: 0.805538 [25600/60000]

loss: 0.800145 [32000/60000]

loss: 0.883490 [38400/60000]

loss: 0.838615 [44800/60000]

loss: 0.846827 [51200/60000]

loss: 0.807034 [57600/60000]

Test Error:

Accuracy: 69.5%, Avg loss: 0.816762

Epoch 10

-------------------------------

loss: 0.829010 [ 0/60000]

loss: 0.899572 [ 6400/60000]

loss: 0.672739 [12800/60000]

loss: 0.869905 [19200/60000]

loss: 0.774595 [25600/60000]

loss: 0.761919 [32000/60000]

loss: 0.849775 [38400/60000]

loss: 0.812002 [44800/60000]

loss: 0.814165 [51200/60000]

loss: 0.777352 [57600/60000]

Test Error:

Accuracy: 70.8%, Avg loss: 0.785891

Done!

Further Reading

* [Loss Functions](https://pytorch.org/docs/stable/nn.html#loss-functions)
* [torch.optim](https://pytorch.org/docs/stable/optim.html)
* [Warmstart Training a Model](https://pytorch.org/tutorials/recipes/recipes/warmstarting_model_using_parameters_from_a_different_model.html)

**Total running time of the script:** ( 1 minutes 47.198 seconds)