Deep Learning in Practice

Lecture 10 — CS 577 Deep Learning

Instructor: Yutong Wang

Computer Science
Illinois Institute of Technology

October 23, 2024

The gap between `ag.Tensor` and PyTorch

- `ag.Tensor` assume everything has gradients
- many layers/tools are not yet supported:
    - advanced optimizers
    - data wrangling tools like mini-batching
    - normalization (e.g., batch norm)
    - dropout
    - weight decay
    - model compression
    - ...
- and many other shortcoming

Grads

```python
class Tensor: # Tensor with grads
    def __init__(self,
                 value,
                 op="",
                 _backward= lambda : None,
                 inputs=[],
                 label=""):

        if type(value) in [float ,int]:
            value = np.array(value)
        self.value = value

        self.grad = np.zeros_like(value) #  <- always need this??
```

PyTorch tensors has

```python
torch.Tensor.requires_grad_ # true if needs to be updated
```

The gap between `ag.Tensor` and PyTorch

- `ag.Tensor` assume everything has gradients
- many layers/tools are not yet supported:
    - **advanced optimizers**
    - data wrangling tools like mini-batching
    - normalization (e.g., batch norm)
    - dropout
    - weight decay
    - model compression
    - ...
- and many other shortcoming

Stochastic gradient descent (SGD)

Let $\eta_t > 0$ be learning rates, $t = 1, 2, \ldots$
Let $m \geq 1$ be an integer

- Initialize $\boldsymbol{\theta}$
- While not converged ($t =$ iteration counter):
  - Select m samples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ and matching labels $\{y^{(1)}, \ldots, y^{(m)}\}$
  - Compute gradient $\mathbf{g} \leftarrow \nabla_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}, \boldsymbol{\theta}), y^{(i)})$
  - Compute update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_t \mathbf{g}$

Stochastic gradient descent (SGD)

Let $\eta_t > 0$ be learning rates, $t = 1, 2, \dots$
Let $m \geq 1$ be an integer

- Initialize $\boldsymbol{\theta}^{(0)}$
- While not converged ($t =$ iteration counter):
  - Select m samples $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ and matching labels $\{y^{(1)}, \dots, y^{(m)}\}$
  - Compute gradient $\mathbf{g}^{(t)} \leftarrow \nabla_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}, \boldsymbol{\theta}^{(t-1)}), y^{(i)})$
  - Compute update $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} - \eta_t \mathbf{g}^{(t)}$

“Unrolling” SGD

$\boldsymbol{\theta}^{(1)} \leftarrow \boldsymbol{\theta}^{(0)} - \eta_1 \mathbf{g}^{(1)}$

“Unrolling” SGD

$$\boldsymbol{\theta}^{(2)} \leftarrow \boldsymbol{\theta}^{(1)} - \eta_2 \mathbf{g}^{(2)} = \boldsymbol{\theta}^{(0)} - \eta_1 \mathbf{g}^{(1)} - \eta_2 \mathbf{g}^{(2)}$$

“Unrolling” gradient descent

$$\boldsymbol{\theta}^{(3)} \leftarrow \boldsymbol{\theta}^{(2)} - \eta_1 \mathbf{g}^{(3)} = \boldsymbol{\theta}^{(0)} - \eta_1 \mathbf{g}^{(1)} - \eta_2 \mathbf{g}^{(2)} - \eta_3 \mathbf{g}^{(3)}$$

“Unrolling” gradient descent

$$\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} - \eta_t \mathbf{g}^{(t)} = \boldsymbol{\theta}^{(0)} - \eta_1 \mathbf{g}^{(1)} - \eta_2 \mathbf{g}^{(2)} - \eta_3 \mathbf{g}^{(3)} \cdots - \eta_t \mathbf{g}^{(t)}$$

SGD with Momentum

Let $\eta_t > 0$ and $t = 1, 2, \dots$ be the iteration counter. Let $\gamma \in [0, 1)$ be the momentum parameter.

- Initialize $\boldsymbol{\theta}^{(0)}$ and $\boxed{\mathbf{v}^{(0)} = 0}$ (momentum vector)
- While not converged (t = iteration counter):
  - Select m samples $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ and matching labels $\{y^{(1)}, \dots, y^{(m)}\}$
  - Compute gradient $\mathbf{g}^{(t)} \leftarrow \nabla_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}, \boldsymbol{\theta}^{(t-1)}), y^{(i)})$
  - Update velocity $\boxed{\mathbf{v}^{(t)} \leftarrow \gamma \mathbf{v}^{(t-1)} + \mathbf{g}^{(t)}}$
  - Update parameters $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} - \eta_t \mathbf{v}^{(t)}$

[Sut+13] “On the importance of initialization and momentum in deep learning”

“Unrolling” Velocity in SGD with Momentum

SGD with momentum: just the key components

- Initialize $\boldsymbol{\theta}^{(0)}$ and $\boxed{\mathbf{v}^{(0)} = 0}$ (momentum vector)
- Update velocity $\boxed{\mathbf{v}^{(t)} \leftarrow \gamma \mathbf{v}^{(t-1)} + \mathbf{g}^{(t)}}$
- Update parameters $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} - \eta_t \mathbf{v}^{(t)}$

$\mathbf{v}^{(1)} \leftarrow \mathbf{g}^{(1)}$

“Unrolling” Velocity in SGD with Momentum

SGD with momentum: just the key components

- Initialize $\boldsymbol{\theta}^{(0)}$ and $\boxed{\mathbf{v}^{(0)} = 0}$ (momentum vector)
- Update velocity $\boxed{\mathbf{v}^{(t)} \leftarrow \gamma \mathbf{v}^{(t-1)} + \mathbf{g}^{(t)}}$
- Update parameters $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} - \eta_t \mathbf{v}^{(t)}$

$\mathbf{v}^{(1)} \leftarrow \mathbf{g}^{(1)}$
$\mathbf{v}^{(2)} \leftarrow \gamma \mathbf{v}^{(1)} + \mathbf{g}^{(2)} = \gamma \mathbf{g}^{(1)} + \mathbf{g}^{(2)}$

“Unrolling” Velocity in SGD with Momentum

SGD with momentum: just the key components

- Initialize $\boldsymbol{\theta}^{(0)}$ and $\boxed{\mathbf{v}^{(0)} = 0}$ (momentum vector)
- Update velocity $\boxed{\mathbf{v}^{(t)} \leftarrow \gamma \mathbf{v}^{(t-1)} + \mathbf{g}^{(t)}}$
- Update parameters $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} - \eta_t \mathbf{v}^{(t)}$

$$
\begin{aligned}
&\mathbf{v}^{(1)} \leftarrow \mathbf{g}^{(1)} \\
&\mathbf{v}^{(2)} \leftarrow \gamma \mathbf{v}^{(1)} + \mathbf{g}^{(2)} = \gamma \mathbf{g}^{(1)} + \mathbf{g}^{(2)} \\
&\mathbf{v}^{(3)} \leftarrow \gamma \mathbf{v}^{(2)} + \mathbf{g}^{(3)} = \gamma^2 \mathbf{g}^{(1)} + \gamma \mathbf{g}^{(2)} + \mathbf{g}^{(3)}
\end{aligned}
$$

"Unrolling" Velocity in SGD with Momentum

SGD with momentum: just the key components

- Initialize $\boldsymbol{\theta}^{(0)}$ and $\boxed{\mathbf{v}^{(0)} = 0}$ (momentum vector)
- Update velocity $\boxed{\mathbf{v}^{(t)} \leftarrow \gamma \mathbf{v}^{(t-1)} + \mathbf{g}^{(t)}}$
- Update parameters $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} - \eta_t \mathbf{v}^{(t)}$

$$
\begin{aligned}
&\mathbf{v}^{(1)} \leftarrow \mathbf{g}^{(1)} \\
&\mathbf{v}^{(2)} \leftarrow \gamma \mathbf{v}^{(1)} + \mathbf{g}^{(2)} = \gamma \mathbf{g}^{(1)} + \mathbf{g}^{(2)} \\
&\mathbf{v}^{(3)} \leftarrow \gamma \mathbf{v}^{(2)} + \mathbf{g}^{(3)} = \gamma^2 \mathbf{g}^{(1)} + \gamma \mathbf{g}^{(2)} + \mathbf{g}^{(3)} \\
&\mathbf{v}^{(t)} \leftarrow \gamma \mathbf{v}^{(t-1)} + \mathbf{g}^{(t)} = \gamma^{t-1} \mathbf{g}^{(1)} + \gamma^{t-2} \mathbf{g}^{(2)} + \cdots + \mathbf{g}^{(t)}
\end{aligned}
$$

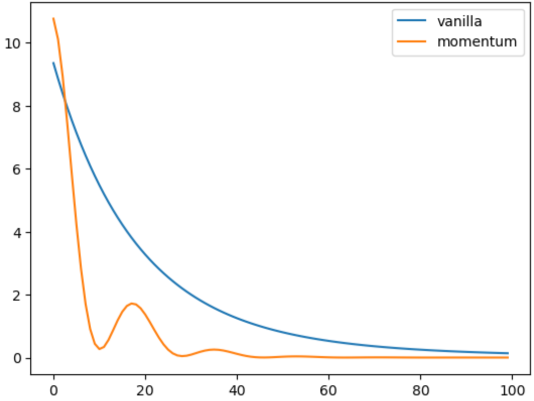
SGD with Momentum

Let $\eta_t > 0$ and $t = 1, 2, \ldots$ be the iteration counter. Let $\gamma \in [0, 1)$ be the momentum parameter.

- Initialize $\boldsymbol{\theta}^{(0)}$ and $\boxed{\mathbf{v}^{(0)} = 0}$ (momentum)
- While not converged (t = iteration counter):
  - Select m samples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ and matching labels $\{y^{(1)}, \ldots, y^{(m)}\}$
  - Compute gradient $\mathbf{g}^{(t)} \leftarrow \nabla_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}, \boldsymbol{\theta}^{(t-1)}), y^{(i)})$
  - Update velocity $\boxed{\mathbf{v}^{(t)} \leftarrow \gamma \mathbf{v}^{(t-1)} + \mathbf{g}^{(t)}}$
  - Update parameters $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} - \eta_t \mathbf{v}^{(t)}$

In class exercise

`lec10-in-class-ex-optim.ipynb`



Adam Optimizer

Let $\eta_t > 0$, $\beta_1, \beta_2 \in [0, 1)$, and $t = 1, 2, \ldots$ be the iteration counter.

- Initialize $\boldsymbol{\theta}^{(0)}$, $\boxed{\mathbf{v}^{(0)} = 0}$ (first moment), $\boxed{\mathbf{s}^{(0)} = 0}$ (second moment)
- While not converged (t = iteration counter):
  - Select m samples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ and matching labels $\{y^{(1)}, \ldots, y^{(m)}\}$
  - Compute gradient $\mathbf{g}^{(t)} \leftarrow \nabla_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}, \boldsymbol{\theta}^{(t-1)}), y^{(i)})$
  - Update first moment: $\boxed{\mathbf{v}^{(t)} \leftarrow \beta_1 \mathbf{v}^{(t-1)} + (1 - \beta_1)\mathbf{g}^{(t)}}$ $\boxed{\hat{\mathbf{v}}^{(t)} \leftarrow \dfrac{\mathbf{v}^{(t)}}{1 - \beta_1^t}}$
  - Update second moment: $\boxed{\mathbf{s}^{(t)} \leftarrow \beta_2 \mathbf{s}^{(t-1)} + (1 - \beta_2)(\mathbf{g}^{(t)})^2}$ $\boxed{\hat{\mathbf{s}}^{(t)} \leftarrow \dfrac{\mathbf{s}^{(t)}}{1 - \beta_2^t}}$
  - Update parameters: $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} - \eta_t \hat{\mathbf{v}}^{(t)} / (\sqrt{\hat{\mathbf{s}}^{(t)}} + \epsilon)$

The gap between `ag.Tensor` and PyTorch

- `ag.Tensor` assume everything has gradients
- many layers/tools are not yet supported:
    - advanced optimizers
    - **data wrangling tools like mini-batching**
    - normalization (e.g., batch norm)
    - dropout
    - weight decay
    - model compression
    - ...
- and many other shortcoming

Data minibatching

From `lenet5_mnist.ipynb`

```
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])

mnist_train = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)

mnist_train_loader = DataLoader(mnist_train, batch_size=64, shuffle=True)
```

Stochastic gradient descent (SGD)

Let $\eta_t > 0$ be learning rates, $t = 1, 2, \ldots$
Let $m \geq 1$ be an integer

- Initialize $\boldsymbol{\theta}$
- While not converged (t = iteration counter):
    - Select m samples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ and matching labels $\{y^{(1)}, \ldots, y^{(m)}\}$
    - Compute gradient $\mathbf{g} \leftarrow \nabla_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}, \boldsymbol{\theta}), y^{(i)})$
    - Compute update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_t \mathbf{g}$

Stochastic gradient descent (SGD)

Let $\eta_t > 0$ be learning rates, $t = 1, 2, \ldots$
Let $m \geq 1$ be an integer

- Initialize $\boldsymbol{\theta}^{(0)}$
- While not converged ($t =$ iteration counter):
  - Select m samples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ and matching labels $\{y^{(1)}, \ldots, y^{(m)}\}$
  - Compute gradient $\mathbf{g}^{(t)} \leftarrow \nabla_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}, \boldsymbol{\theta}^{(t-1)}), y^{(i)})$
  - Compute update $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} - \eta_t \mathbf{g}^{(t)}$

Stochastic gradient descent (SGD)

Let $\eta_t > 0$ be learning rates, $t = 1, 2, \ldots$
Let $m \geq 1$ be an integer

- Initialize $\boldsymbol{\theta}^{(0)}$
- While not converged ($t =$ iteration counter):
  - Select m samples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ and matching labels $\{y^{(1)}, \ldots, y^{(m)}\}$
  - Compute gradient $\mathbf{g}^{(t)} \leftarrow \nabla_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}, \boldsymbol{\theta}^{(t-1)}), y^{(i)})$
  - Compute update $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} - \eta_t \mathbf{g}^{(t)}$

Stochastic gradient descent (SGD)

Let $\eta_t > 0$ be learning rates, $t = 1, 2, \ldots$
Let $m \geq 1$ be an integer

- Initialize $\boldsymbol{\theta}^{(0)}$
- While not converged (t = iteration counter):
  - Select m samples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ and matching labels $\{y^{(1)}, \ldots, y^{(m)}\}$
  - Compute gradient $\mathbf{g}^{(t)} \leftarrow \nabla_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}, \boldsymbol{\theta}^{(t-1)}), y^{(i)})$
  - Compute update $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} - \eta_t \mathbf{g}^{(t)}$

```
mnist_train_loader = DataLoader(mnist_train, batch_size=64, shuffle=True
    )
```

Test data

```python
mnist_test = torchvision.datasets.MNIST(root='./data', train=False,
    download=True, transform=transform)

mnist_test_loader = DataLoader(mnist_test, batch_size=64, shuffle=False)
```

Don't look at the test data...

...when you are

- deciding on an architecture to use
- tuning hyperparameters
- training the data

- Because...

Test data

```
mnist_test = torchvision.datasets.MNIST(root='./data', train=False,
    download=True, transform=transform)

mnist_test_loader = DataLoader(mnist_test, batch_size=64, shuffle=False)
```

Don't look at the test data...

...when you are

- deciding on an architecture to use
- tuning hyperparameters
- training the data

- Because...
- ...You might trick yourself (and others) into making tweaks that are helping

Test data

```
mnist_test = torchvision.datasets.MNIST(root='./data', train=False,
    download=True, transform=transform)

mnist_test_loader = DataLoader(mnist_test, batch_size=64, shuffle=False)
```

Don't look at the test data...

...when you are

- deciding on an architecture to use
- tuning hyperparameters
- training the data

- Because...
- ...You might trick yourself (and others) into making tweaks that are helping
- ...In practice you don't have the test data ahead of time

Real world example

- Goal: you're helping the US Postal Service to improve mail delivery (it's the early 1990s)

Real world example

- Goal: you're helping the US Postal Service to improve mail delivery (it's the early 1990s)

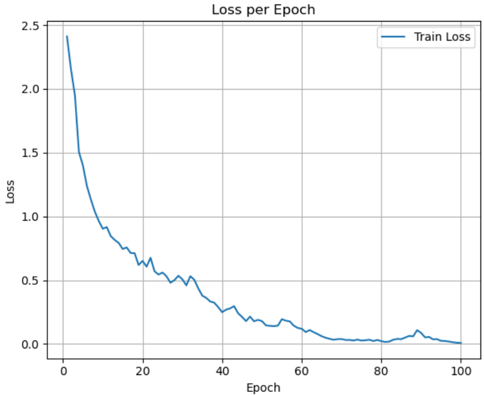
- Collect *training data* and hire people to label handwritten digits

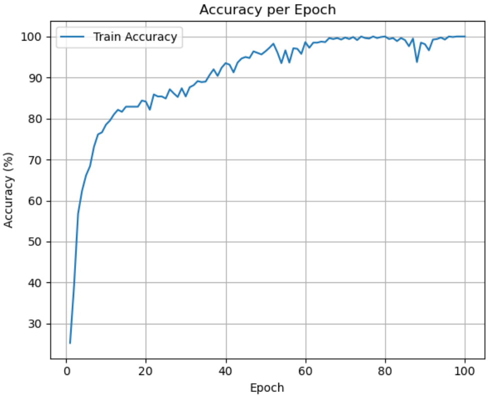
Real world example

- Goal: you're helping the US Postal Service to improve mail delivery (it's the early 1990s)

- Collect *training data* and hire people to label handwritten digits
- Deploy in the wild (on test data that the model has never seen before)

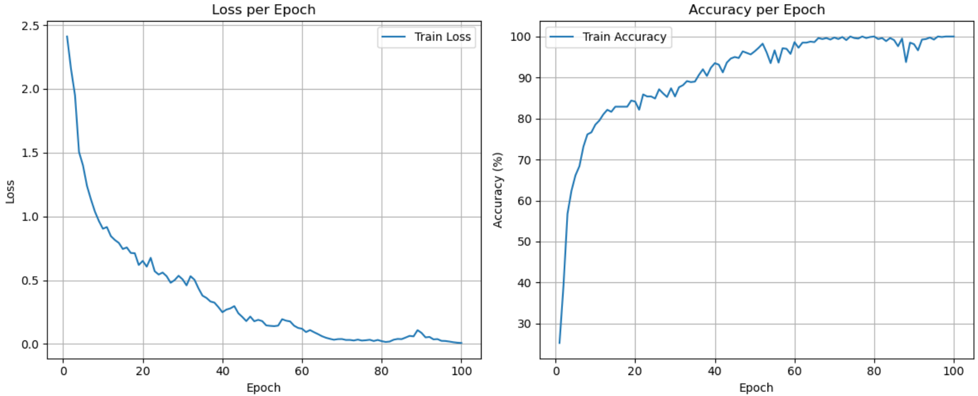
Overfitting





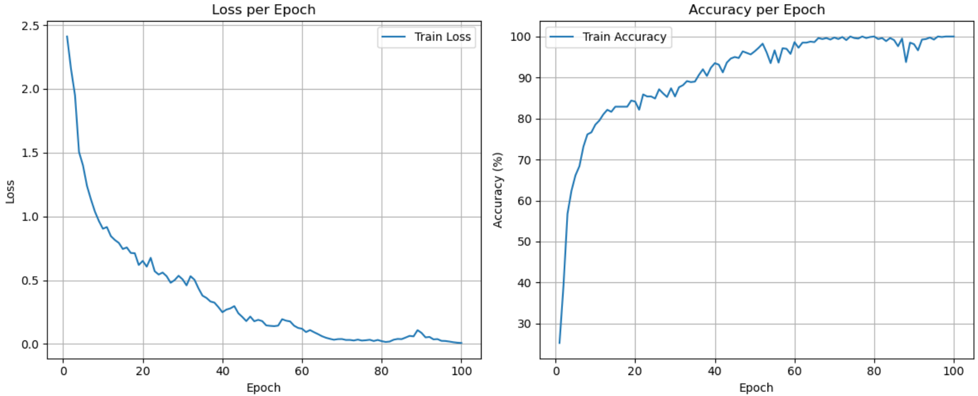
-

Overfitting



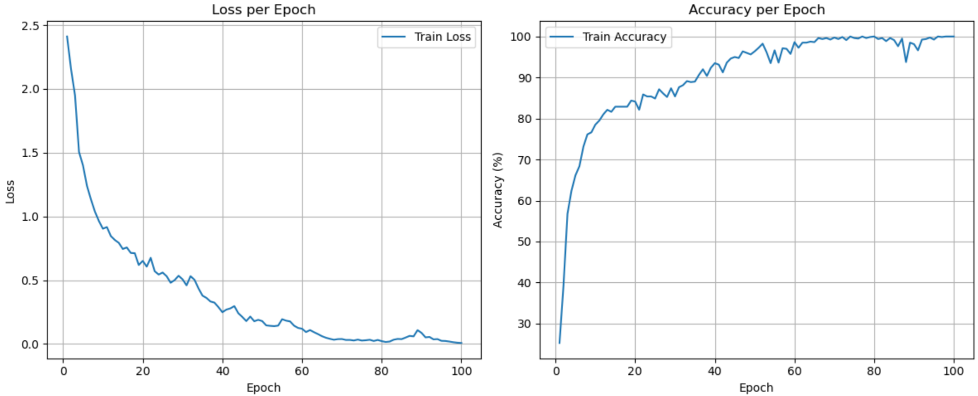
-
- `Test accuracy: 81.1% accuracy`

Overfitting



-
- `Test accuracy: 81.1% accuracy`
- Disclaimer: I injected noise into the training data (but not the test data) to prove a point. MNIST is very easy. Even LeNet5 can reach 98% test accuracy.

Overfitting

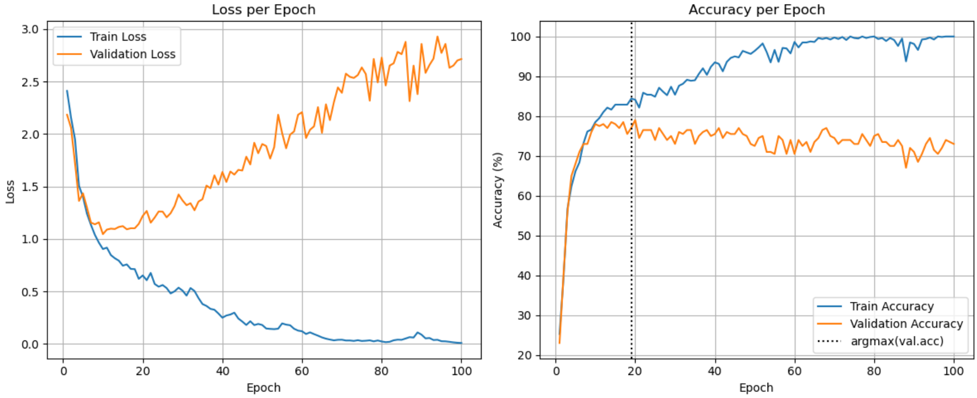


-
- `Test accuracy: 81.1% accuracy`
- Disclaimer: I injected noise into the training data (but not the test data) to prove a point. MNIST is very easy. Even LeNet5 can reach 98% test accuracy.
- Solution: split the training data further into train and validation sets.

Train/valid splitting

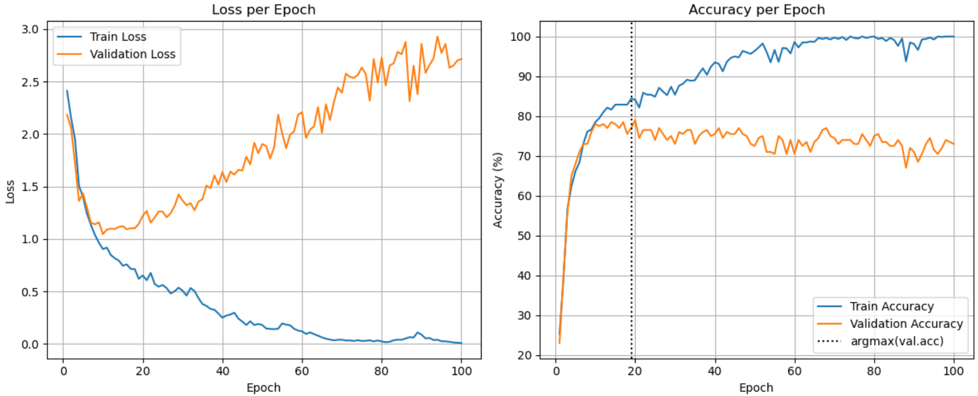
```
import torch.utils.data as data
# [...]
train_size = int(0.8 * len(train_dataset))
val_size = len(train_dataset) - train_size
train_dataset, val_dataset = data.random_split(train_dataset, [
    train_size, val_size])
```

Split train into train/validation sets



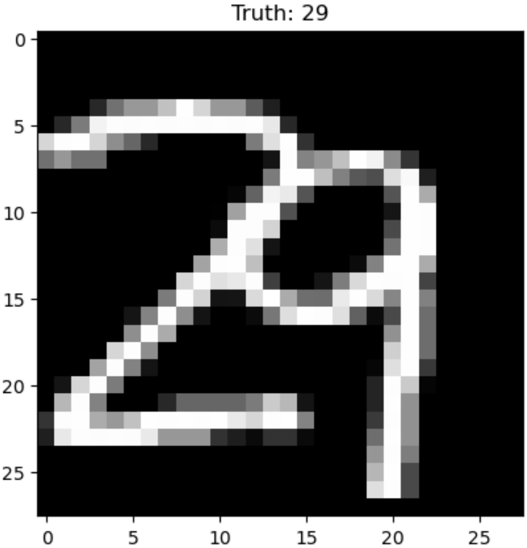
-

Split train into train/validation sets

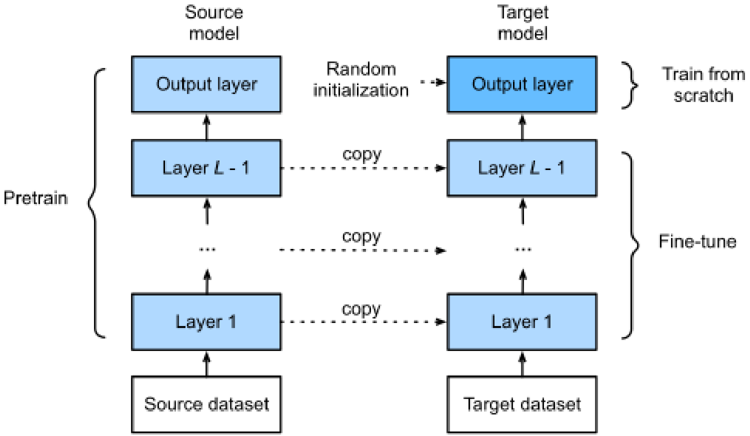


-
- `85.8% test accuracy` at epoch with the highest validation accuracy.

A data challenge for today



Transfer learning



From https://d2l.ai/ chapter on fine-tuning

Transfer learning

- Train model on one task
- Copy the weights to a new task
- Update weights based on optimization from new task

```python
class LeNet5_Custom(nn.Module):
    def __init__(self, pretrained_model):
        super(LeNet5_Custom, self).__init__()
        # Clone the layers from the pre-trained model
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv1.weight = nn.Parameter(pretrained_model.conv1.weight.clone())
        self.conv1.bias = nn.Parameter(pretrained_model.conv1.bias.clone())
```

Other ideas

- Data augmentation
- Batch normalization
- Weight decay
- Skip connections
- Dropout

References I

[Sut+13] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. "On the importance of initialization and momentum in deep learning". In: *International conference on machine learning*. PMLR. 2013, pp. 1139–1147.