PyTorch: A deep learning package for Python

CSCI 5448
Graduate Presentation
15 April 2019
By: Dieu My Nguyen

Python: A (very) brief history

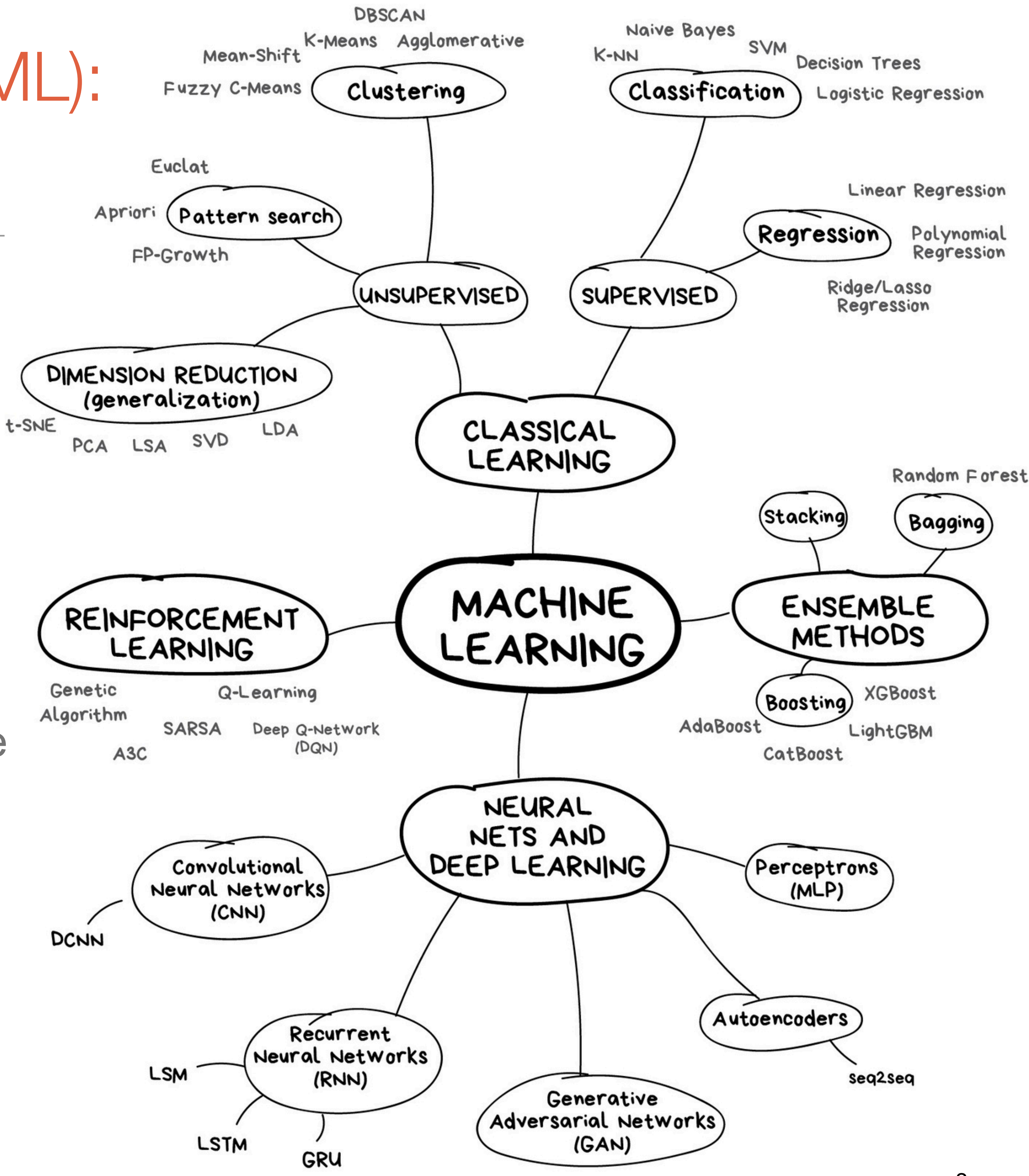
- An interpreted, high-level, general-pose language [1]
- Created by Guido van Rossum [1]
- First release: 1991 [2]
- Current version as of April 2019: 3.7.3
- Design philosophy emphasizes code readability, follows the Zen of Python [3]
- Supports multiple programming paradigms: object-oriented, imperative, functional, procedural [1,2]

(c) Guido van Rossum's GitHub

Try: `import this`

Machine learning (ML): A brief intro

- A branch of artificial intelligence (AI) based on the idea that systems can learn from experience and improve the efficiency of their own programs to make decisions with little human intervention [4]
- Map of the ML world in the diagram [5]:
- Main types of ML today: Classical ML, emsembles, reinforcement learning, **neural networks and deep learning [4]**



(c) Vasily Zubarev

Machine learning: Packages for Python

- ❖ Essentials for data processing and math:
  - ❖ NumPy, SciPy, Pandas
- ❖ Visualization:
  - ❖ Matplotlib, Seaborn, Plotly, and more
- ❖ Machine learning:
  - ❖ SciKit-Learn: Concise and consistent inferface to common classical ML algorithms
  - ❖ TensorFlow: Library of data flow graphs computations, multi-layered nodes enable quick training of NNs on large datasets, define-compile-run paradigm
  - ❖ Keras: High-level, minimalist interface for building NNs, uses TensorFlow as backend
  - ❖ **PyTorch: A deep learning framework and scientific computing package with GPU support**

PyTorch: A brief history

- Initial release: October 2016 [12]
- Current version as of April 2019: 1.0.1 [13]
- Original authors: Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan [13]
- Soumith Chintala is credited with bootstrapping the project - Lua Torch was aging, so a Python version came to be [14].
- Soumith Chintala worked (and still does) work at Facebook at the time of creating PyTorch, so often we hear that PyTorch is created and maintained by Facebook. However, many other companies also have a vested interest [14].

(c) Soumith Chintala's GitHub

PyTorch: What is it?

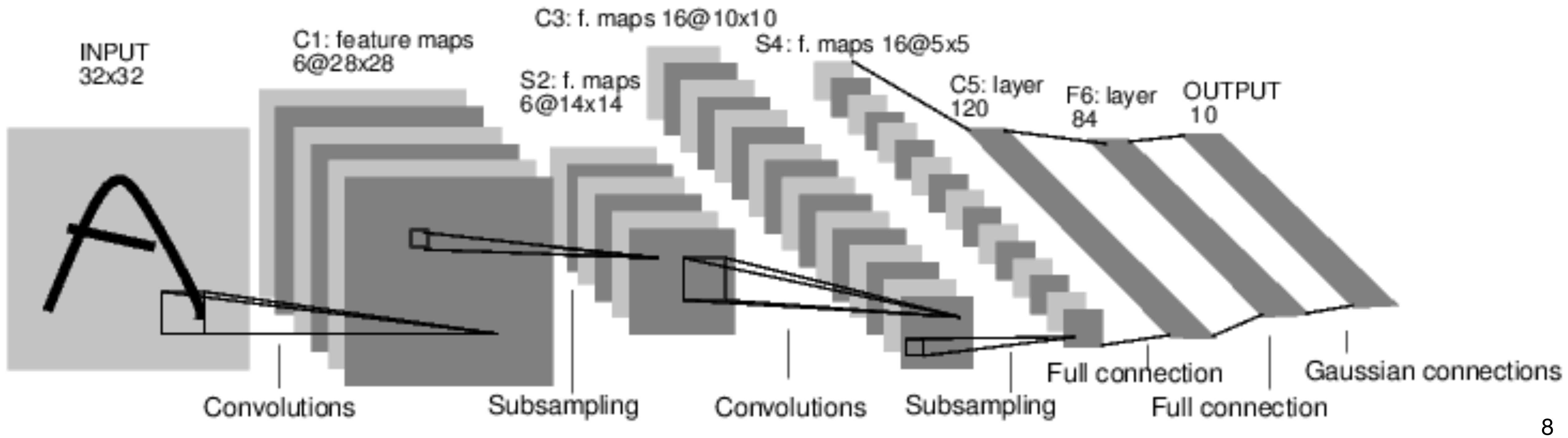
- A deep learning framework with speed and flexibility [13]
- A scientific computing package [12]
    - Like NumPy but can leverage the power of GPUs
- TensorFlow (and Theano) provides constructs and users can express a computational graph representing a mathematical expression, which is processed/compiled to compute the expression of gradient. The graph is static and fully determined before actual operations occur [15].
- Compared to the define-compile-run paradigm of TensorFlow, PyTorch is a dynamic, define-by-run framework: there is no compilation step. The graph is generated on the fly as the operations are created [13].
    - Users can define mathematical expressions and directly evoke an operator to compute it

PyTorch: Deep learning features

- Primary components/packages to build neural networks [16]:
  - `torch`: The top-level PyTorch package and tensor library.
  - `torch.nn`: A subpackage that contains modules and extensible classes for building neural networks.
  - `torch.autograd`: A subpackage that supports all the differentiable Tensor operations in PyTorch.
  - `torch.nn.functional`: A functional interface that contains typical operations used for building neural networks like loss functions, activation functions, and convolution operations.
  - `torch.optim`: A subpackage that contains standard optimization operations like SGD and Adam.
  - `torch.utils`: A subpackage that contains utility classes like data sets and data loaders that make data preprocessing easier.
  - `torchvision`: A package that provides access to popular datasets, model architectures, and image transformations for computer vision.

PyTorch: CNN for Fashion-MNIST example

Let's delve into a code example using a CNN and the Fashion-MNIST dataset to see more PyTorch functionalities at work.



PyTorch: CNN for Fashion-MNIST example

❖ Four primary steps of this project:

1. Prepare the data
2. Build the CNN model
3. Train the model
4. Analyze the model’s results

PyTorch: CNN for Fashion-MNIST example

- Prepare the data:
  - Since Fashion-MNIST comes with `torchvision`, we can quickly get an instance of it
  - And use the built-in `transforms.ToTensor()` transformation to transform our images into tensors.
  - Then, we create a `DataLoader` wrapper for the train and test sets.

```
train_set = torchvision.datasets.FashionMNIST(
            root='./data/FashionMNIST',
            train=True,
            download=True,
            transform=transforms.Compose([transforms.ToTensor()]))

test_set = torchvision.datasets.FashionMNIST(
            root='./data/FashionMNIST',
            train=False,
            download=True,
            transform=transforms.Compose([transforms.ToTensor()]))
```

```
train_loader = torch.utils.data.DataLoader(train_set,
                                           batch_size=batch_size,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(test_set,
                                          batch_size=batch_size,
                                          shuffle=True)
```

PyTorch: CNN for Fashion-MNIST example

- Build the CNN model:
  - Steps to a CNN in PyTorch:
    - Create a neural network class that extends the `torch.nn.Module` base class.
    - In the class constructor, define the network's layers as class attributes using pre-built layers from `torch.nn`.
    - Use the network's layer attributes as well as operations from the `torch.nn.functional` API to define the network's forward pass.

An example CNN class inheriting from `nn.Module`, with the network's architecture defined in the constructor and a `forward()` method to pass a batch of images from the train set through all the layers.

```
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.drop_out = nn.Dropout()
        self.fc1 = nn.Linear(7 * 7 * 64, 1000)
        self.fc2 = nn.Linear(1000, 10)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.reshape(out.size(0), -1)
        out = self.drop_out(out)
        out = self.fc1(out)
        out = self.fc2(out)
        return out
```

More details on function parameters are in the code example.

PyTorch: CNN for Fashion-MNIST example

- Train the model:
  - Training the CNN is a matter of forward and backward passes to let the network learn by itself how to build important features.

    We first define hyperparameters and choose a loss and optimizer.

```
# Hyperparameters
num_epochs = 20
num_classes = 10
batch_size = 100
learning_rate = 0.001

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(),
lr=learning_rate)
```

Then instantiate a `ConvNet` object as the model.

```
model = ConvNet()
```

PyTorch: CNN for Fashion-MNIST example

Then, we train! Looping over the epochs and over the batches of images, we simple call `model.train()`, get the model's predictions via `model(images)`, compute the loss as a cross-entropy difference between the network's predictions and the ground-truth labels via `criterion(predictions, labels)`, then use the optimizer to proceed.

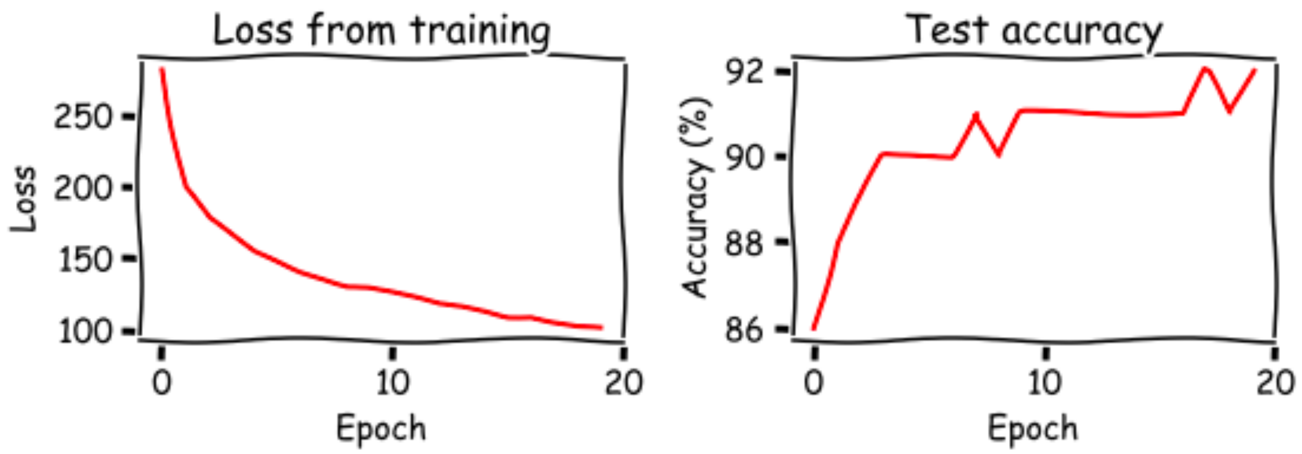
```
for epoch_i in range(num_epochs):
    model.train()
    batch_loss = 0
    for batch_i, (images, labels) in enumerate(train_loader):
        sys.stdout.write(f"\rBatch {batch_i+1}/{len(train_loader)}")
        sys.stdout.flush()

        optimizer.zero_grad()
        predictions = model(images)
        loss = criterion(predictions, labels)
        loss.backward()
        batch_loss += loss.item()
        optimizer.step()
    loss_list.append(batch_loss)
    loss_val = loss_list[-1]

    accuracy = evaluate_cnn()
    acc_list.append(accuracy)
    print(f"\nEpoch {epoch_i+1}/{num_epochs} -- Loss {loss_val:0.4f} -- Test accuracy
{accuracy:0.3f}")
```

PyTorch: CNN for Fashion-MNIST example

- Analyze the model's results:
  - While we train, we can accumulate a list of loss and test accuracy values and simple plot via `matplotlib` to see how they change over time.
  - Evaluating the network's performance is a simple task. So is saving and loading a model.



Just 20 epochs using a simple CNN architecture on my CPU gets us a max of 92% test accuracy.

```
def evaluate_cnn():
    model.eval()
    correct = 0
    total = 0
    for images, labels in test_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum()

    accuracy = 100*correct/total
    return accuracy
```

```
SAVE_MODEL = True
if SAVE_MODEL:
    save_dict = {
        "model"  : model.state_dict(),
        "optim"  : optimizer.state_dict()
    }
    torch.save(save_dict, “test.pt")

LOAD_MODEL = False
if LOAD_MODEL:
    load_dict = torch.load("test.pt")
    model_dict = load_dict['model']
    optim_dict = load_dict['optim']

    cnn.load_state_dict(model_dict)
    optimizer.load_state_dict(optim_dict)
```

References

[1] Guido van Rossum (2003) Python Language Reference Manual Release 2.3. Network Theory Limited.

[2] MF Sanner (1999) Python: a programming language for software integration and development. J Mol Graph Model.

[3] T Peters (2004) The Zen of Python.

[4] MI Jordan and TM Mitchell (2015) Machine learning: Trends, perspectives, and prospects. Science.

[5] Vasily Zubarev (2019) Machine Learning for Everyone.

[6] Martin Anthony, Peter L. Bartlett (1999) Neural Network Learning: Theoretical Foundations

[7] Yann LeCun, Yoshua Bengio & Geoffrey Hinton (2015) Deep learning. Nature.

[8] JürgenSchmidhuber (2015) Deep learning in neural networks: An overview. Neural Networks.

[9] Ronan Collobert et al (2002) Torch: A Modular Machine Learning Software Library. IDIAP.

[10] Torch GitHub repository ReadMe.

[11] Ronan Collobert et al (2011) Torch7: A Matlab-like Environment for Machine Learning.

[12] Ketkar N. (2017) Introduction to PyTorch. Deep Learning with Python.

[13] PyTorch GitHub repository.

[14] deeplizard PyTorch Lesson Series.

[15] Martín Abadi et al (2016). TensorFlow: A System for Large-Scale Machine Learning. OSDI.

[16] PyTorch Documentation.

[17] Fashion-MNIST GitHub.