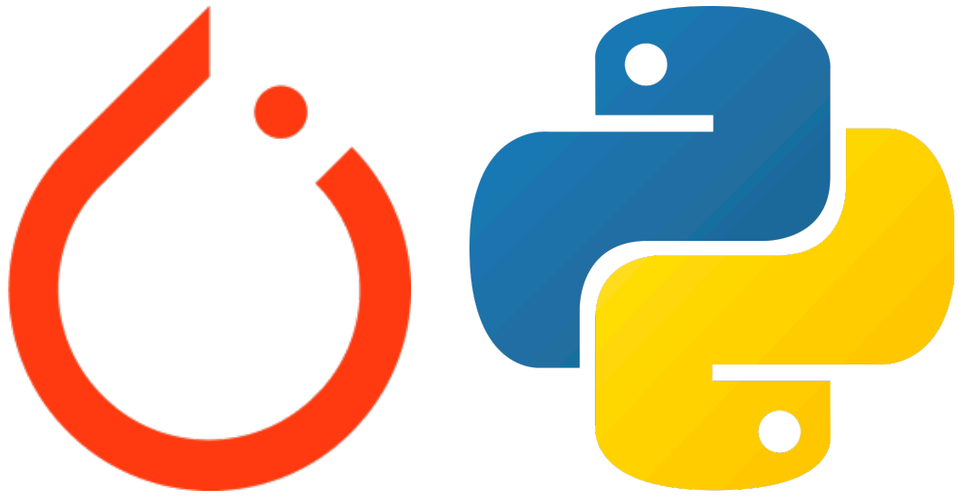


PyTorch: A deep learning package for Python

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

Table of contents

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

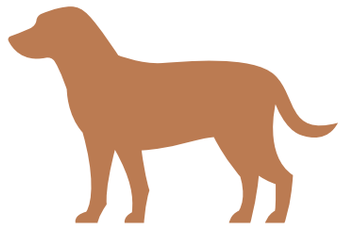
Try: `import this`

Python: An object-oriented language

- Basic OO concepts in Python:
  - Classes and objects
  - Attributes and methods
  - `self`: Equivalent of `this` in Java

Python: Classes and objects

- Like other languages, Python has primitive data structures (integers, strings, lists, etc.), but sometimes that's not sufficient to model something more complex
- Classes are a blueprint or structure of how something should be defined but has no real content
  - Example: We define a class `Dog` which has a name and an age



- Objects are instances of a class with actual values
  - Example: We create the `Mimi` object which is an instance of `Dog` and she is 2 years old

Python: Attributes and methods, the `self`

- Attributes: The `__init__()` method is called automatically to initialize an object's initial attributes. This method must have at least 1 argument and the `self` variable, which refers to the object itself

```
Class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

- Methods: Describe the behavior of an instance of a class

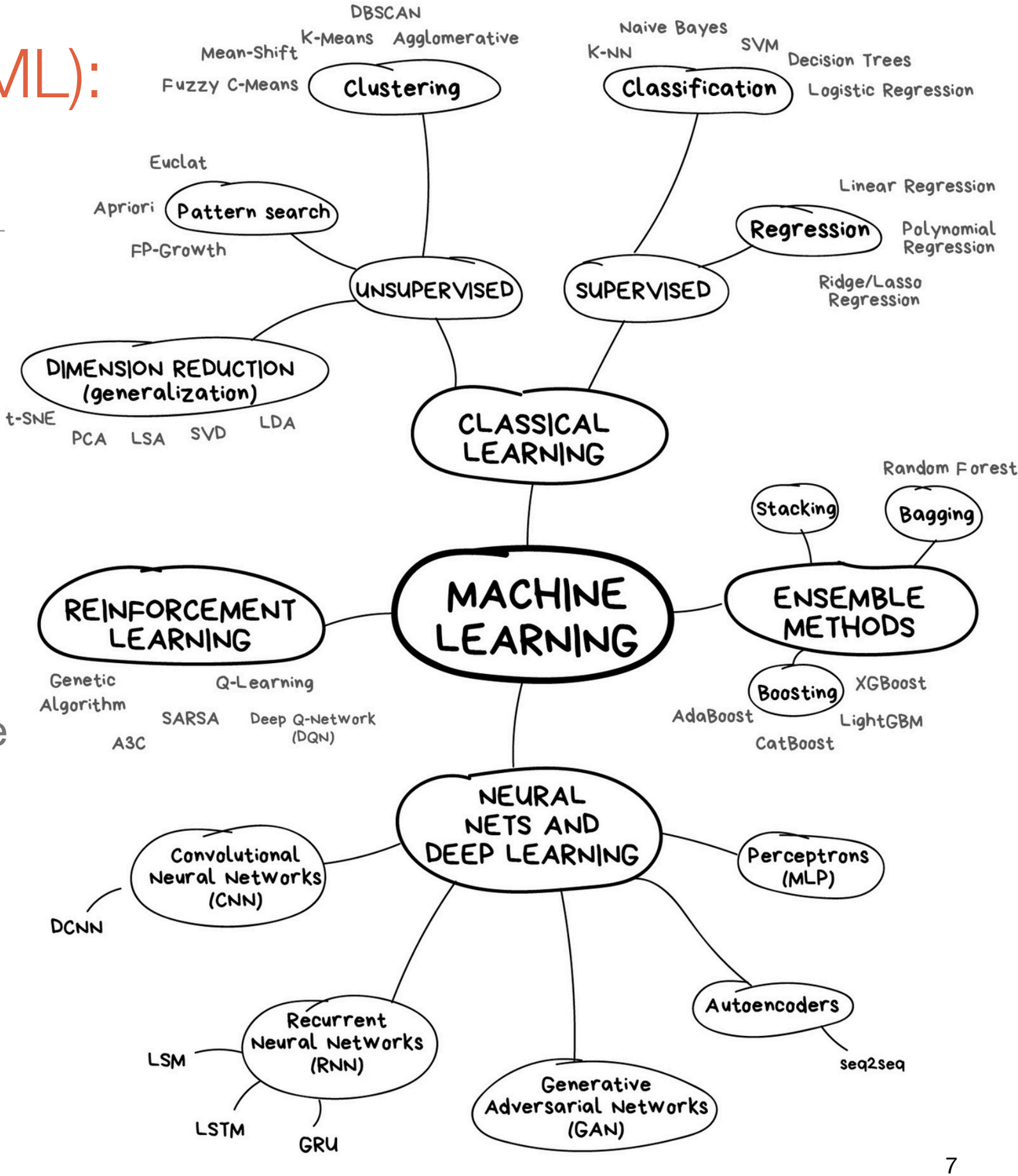
```
Class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def woof(self):
        print("Woof!")
```

- Instantiate an object of `Dog` and call `woof()`:

```
Mimi = Dog("Mimi", 2)
Mini.woof()
```

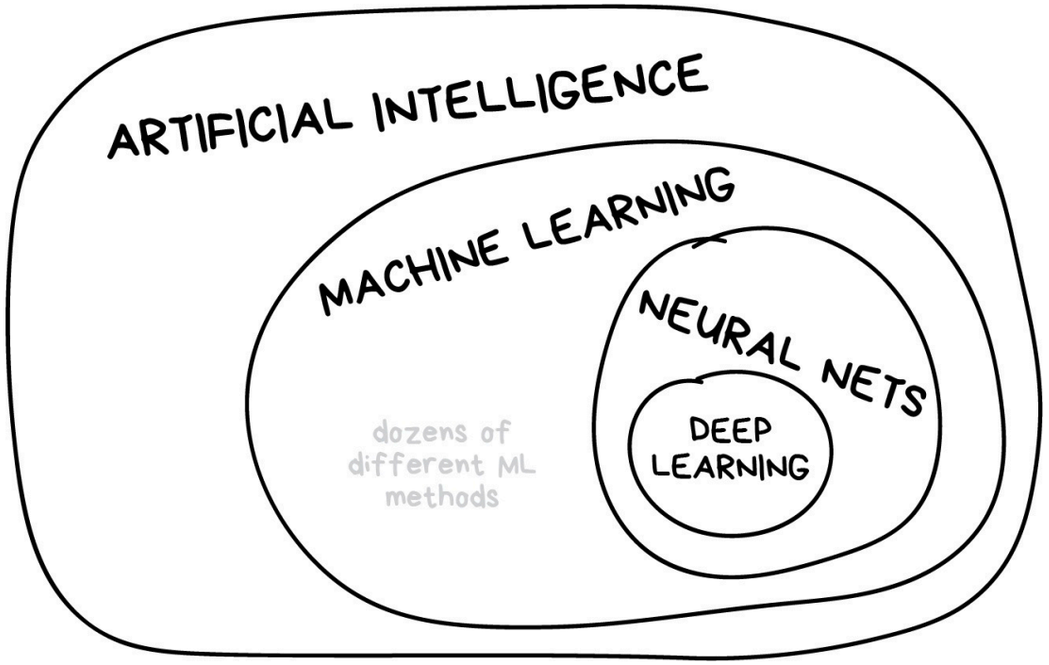
Machine learning (ML): A brief intro

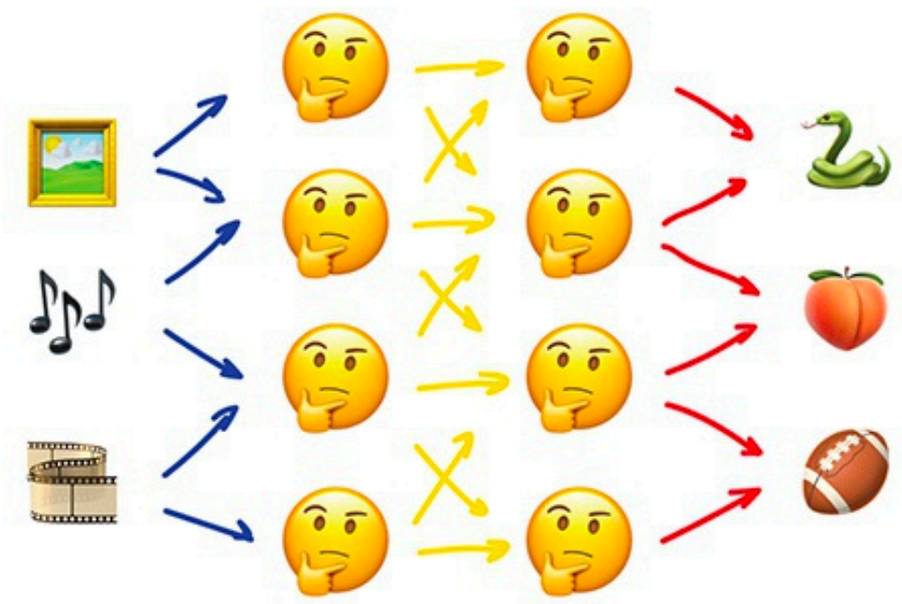
- A branch of artificial intelligence (AI) based on the idea that systems can learn from experience and improve the efficiency of their own programs to make decisions with little human intervention [4]
- Map of the ML world in the diagram [5]:
- Main types of ML today: Classical ML, emsembles, reinforcement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



Machine learning: Neural networks & deep learning

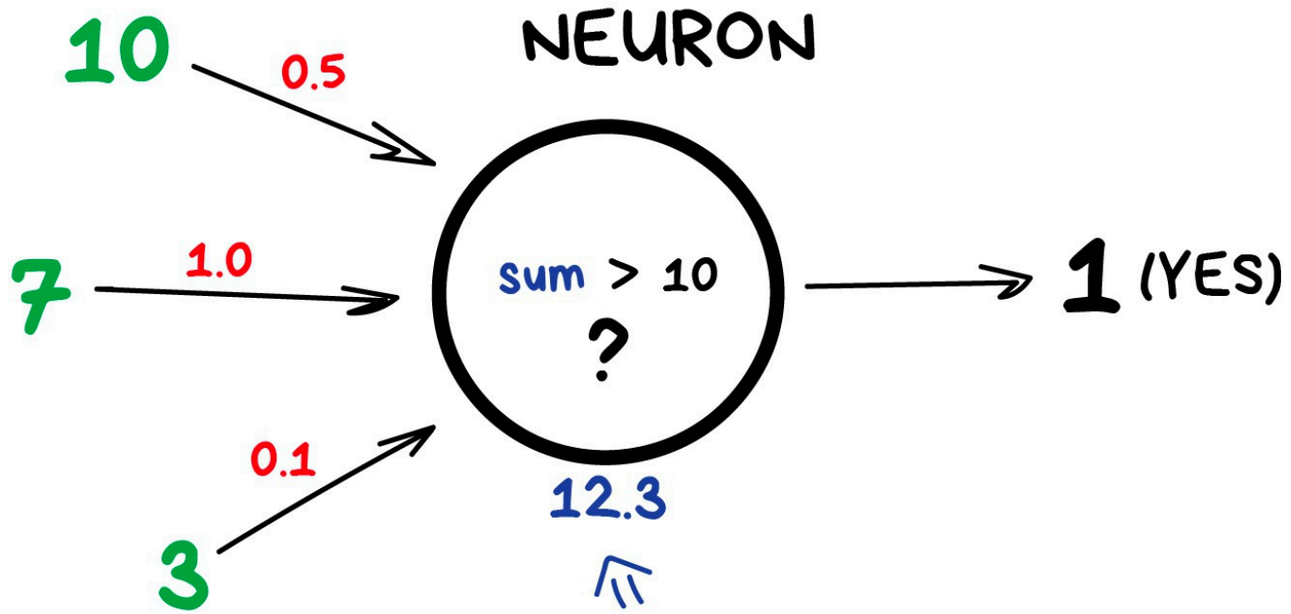
- Neural networks (NNs) are one of the ML types [6]
- A NN is a collection of neurons and connections between them [6]
- Deep learning is a method of building, training, and using NNs [7,8].
- Typical applications: Replacing classical ML algorithms, computer vision, speech processing, machine translation… [7,8]





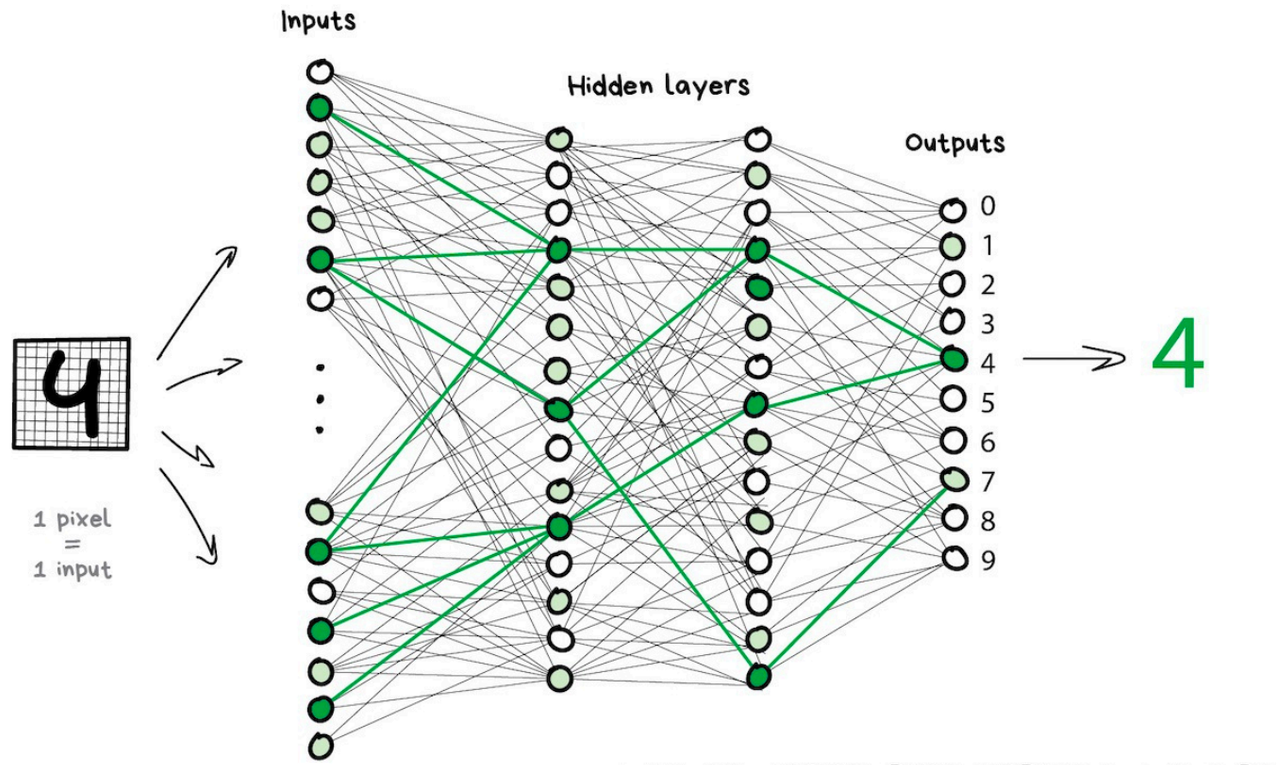
Machine learning: Neural networks & deep learning

- A neuron is a function with inputs and an output.
  - Ex: This neuron sums all the numbers from the inputs and check if the sum is greater than N. If so, give 1 as the output, otherwise 0.
- Connections are channels between neurons. Each connection has a parameter: the weight. The weights represent strength for a signal.
  - Weights are adjusted during training - essentially how the network learns.
- Neurons are linked by layers and are connected to the neurons of the next and previous layers.
  - A network with multiple layers with connections between them is a multilayer perception (MLP), the simplest architecture.



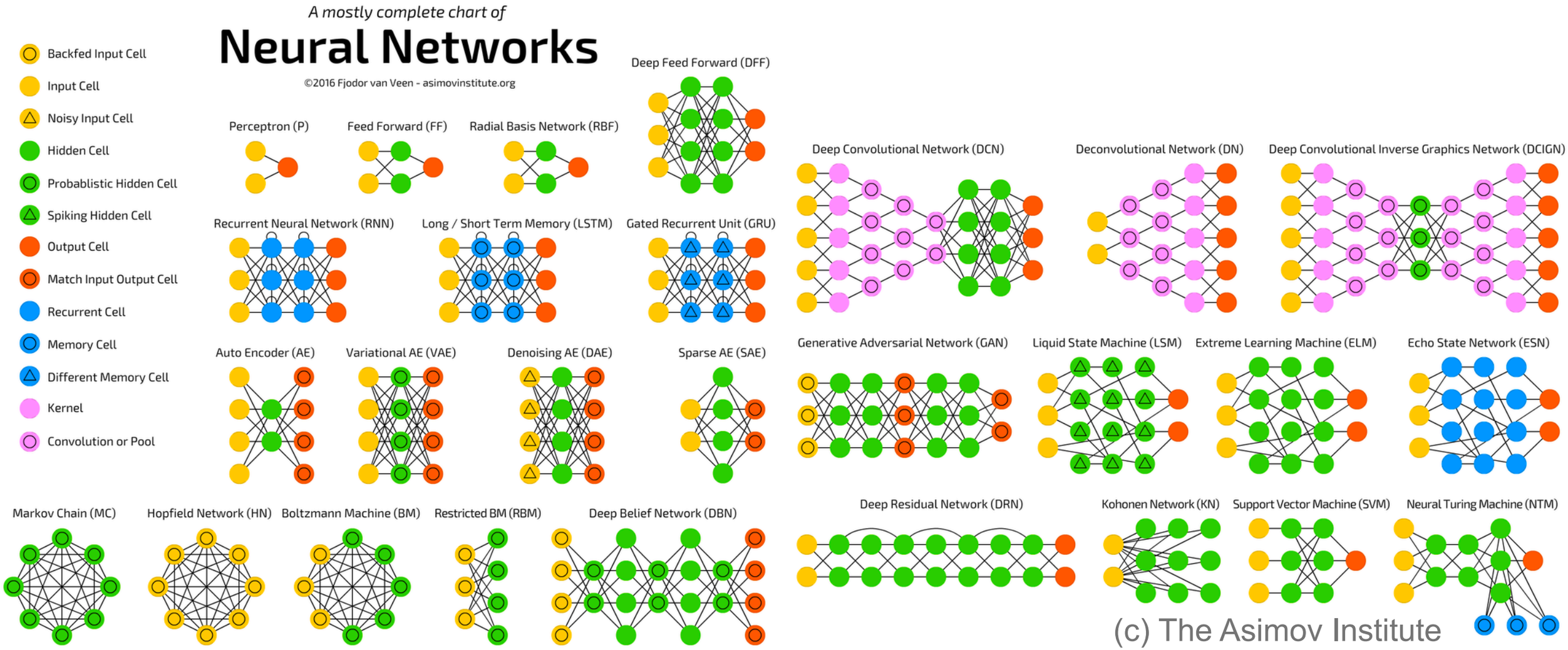
Machine learning: Neural networks & deep learning

- Example neural network classification task: The input is an image of a handwritten digit 4, black pixels activate the associated neurons, then they activate the next layers, until it lights up the neuron in the output layer for the digit.
- Essentially, the NN's job is to adapt its weights during training so that when it sees this input, its output would emit 4.
- The weights are randomly initialized. At first the NN will likely emit incorrect outputs. We'd use backpropagation to traverse the NN backward from outputs to inputs and tell every neuron that the final result is not good - give more and less attention to this and that connection…



(c) Vasily Zubarev

Machine learning: The neural network zoo

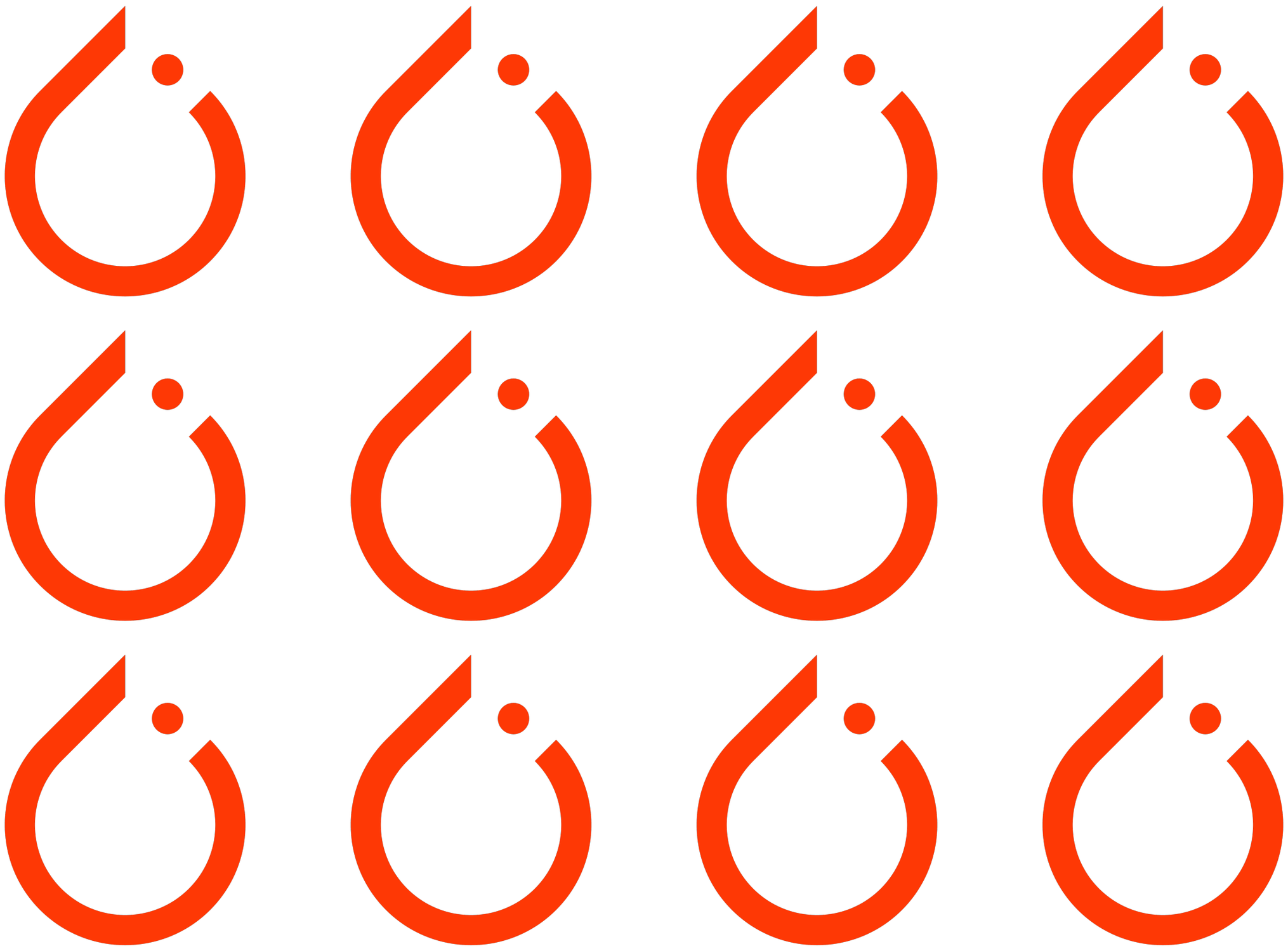


Our later code example will be a CNN (convolutional NN) for image classification

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**

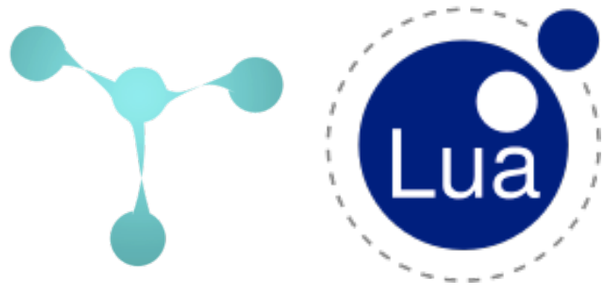
Deep dive into PyTorch



PyTorch: A brief history

Torch: A brief history

- PyTorch initial release: October 2016 [12]
- Before PyTorch there was Torch
  - Torch initial release: October 2002 [9]



  - Original authors: Ronan Collobert, Clement Farabet, Koray Kavukcuoglu [11]
  - Open-source machine learning library and scientific computing framework in Lua with an underlying C implementation [9]
  - As of 2018, Torch is no longer in active development [10]

PyTorch: A brief history
Torch: A brief history

- At the time, there were many new algorithms but it was difficult for scientists to compare them with usual tools to solve a particular task (e.g. speech recognition)
- Torch's objective: To ease the comparison between algorithms and simplify the process of extending them or adding new ones with its objet-oriented design [9].

PyTorch: A brief history
Torch: Main concepts

- Developed using OO paradigm, implemented in C++
- A modular strategy was chosen through the definition of the following broad classes [9]:
  - `DataSet`: This class handles data. Its subclasses provide ways to handle different types of data (e.g. static, dynamic, etc.)
  - `Machine`: This class represents any algorithm (e.g. neural network, support vector machine, hidden Markov model) that, given an input and parameters, returns an output.
  - `Trainer`: This class selects an optimal set of parameters of a machine according to a given criterion and a given `DataSet`, and test it using another (or same) `DataSet`.
  - `Measurer`: Objects of this class prints various measures of interest (e.g. classification error, mean-squared error, log-likelihood, etc.)

PyTorch: A brief history
Torch: General idea

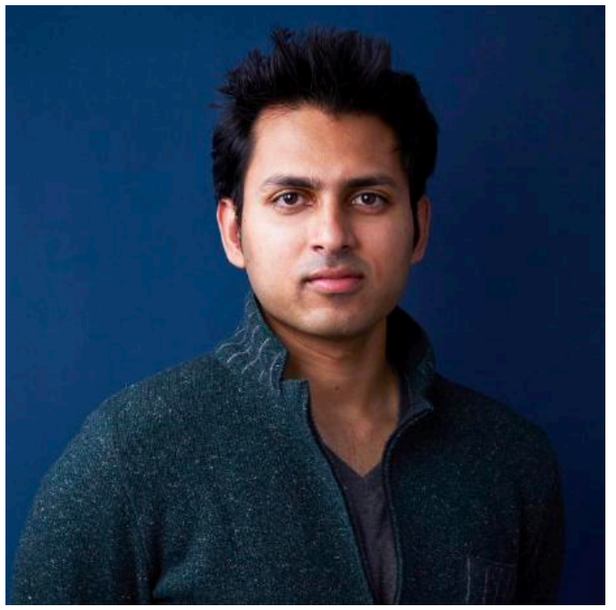
- `DataSet` produces 1 or several training examples.
- `Trainer` gives them to `Machine` which computes an output, which is used by `Trainer` to tune the parameters of `Machine`.
- During this process, 1 or more `Measurer`(s) monitor the performance of the system.

PyTorch: A brief history
Torch: Examples and comparison

- Examples of commonly used machines [9]:
  - Gradient machines (functions that can be trained by gradient descent in back-propagation)
  - Support vector machines
  - Distributions (Gaussian mixture models, hidden Markov models, ensembles)
- Compared to other tools [9]:
  - Most machine learning algorithms in Tech were already standalone softwares
  - Torch provides a unified platform for all these algorithms in the same efficient programming environment, so users can compare various solutions on the same tasks, using the same measures of quality
  - Torch showed similar or faster results in performance and time.

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: Benefits and considerations

- Well suited for research purposes: Relatively easy process of developing and experimenting with new deep learning architectures [12].
- It is a thin framework and stays out of the way and allow us to focus on the neural networks rather than the framework, and we are actually building NNs from scratch [14].
- More intuitive source code, tighter correspondence to mathematical expressions describing the network [14].
- Due to it dynamic nature, debugging is easier [14].
  - With TensorFlow, debugging requires going through 2 abstractions, the Python code and the actual graph [15].
- However, the define-compile-run paradigm gives more room to optimize the underlying computation [15].

PyTorch: Philosophy

- PyTorch's development is guided by these ideas [14]:
  - Stay out of the way
  - Cater to the impatient
  - Promote linear code-flow
  - Full interop with the Python ecosystem
  - Be as fast as anything else
- These ideas make PyTorch great for focusing on the neural networks themselves and for a reliable framework in the long term.

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 major element: Tensors for deep learning

- A `torch.tensor` is a multi-dimensional matrix containing elements of a single data type. It is an instance of the `torch.Tensor` class [16].
  - A mathematical generalization of specific concepts like number/scalar (0 dimensional tensor), array/vector (1 dimensional tensor), and 2d-array/matrix (2 dimensional tensor).
    - The n tells us the number of indices required to access a specific element in the array.
- Similar to NumPy's ndarrays, but can be used on a GPU to accelerate computing
- 100+ tensor operations: transposing, indexing, slicing, mathematical operations, linear algebra, random numbers, etc.
- Central to neural networks because they are the data structure we use to build and train the networks

PyTorch major element: Tensors for deep learning

- Every `torch.Tensor` has these attributes [16]:
  - `torch.dtype`: specifies the uniform data type contained within the tensor.
  - `torch.device`: specifies the device (CPU or GPU) where the tensor's data is allocated.
  - `torch.layout`: specifies how the tensor is stored in memory.

PyTorch major element: Example tensors

```
import torch
```

Make a tensor of a specific type from a list using the `torch.tensor()` constructor.

```
torch.tensor([[0, 1], [2, 3]], dtype=torch.int32)
```

```
tensor([[0, 1],
        [2, 3]], dtype=torch.int32)
```

Access and modify contents of a tensor by using Python's normal indexing and slicing notation.

```
x = torch.tensor([[1, 2, 3], [4, 5, 6]])
print(x[1][2])
```

```
tensor(6)
```

```
x[0][1] = 8
print(x)
```

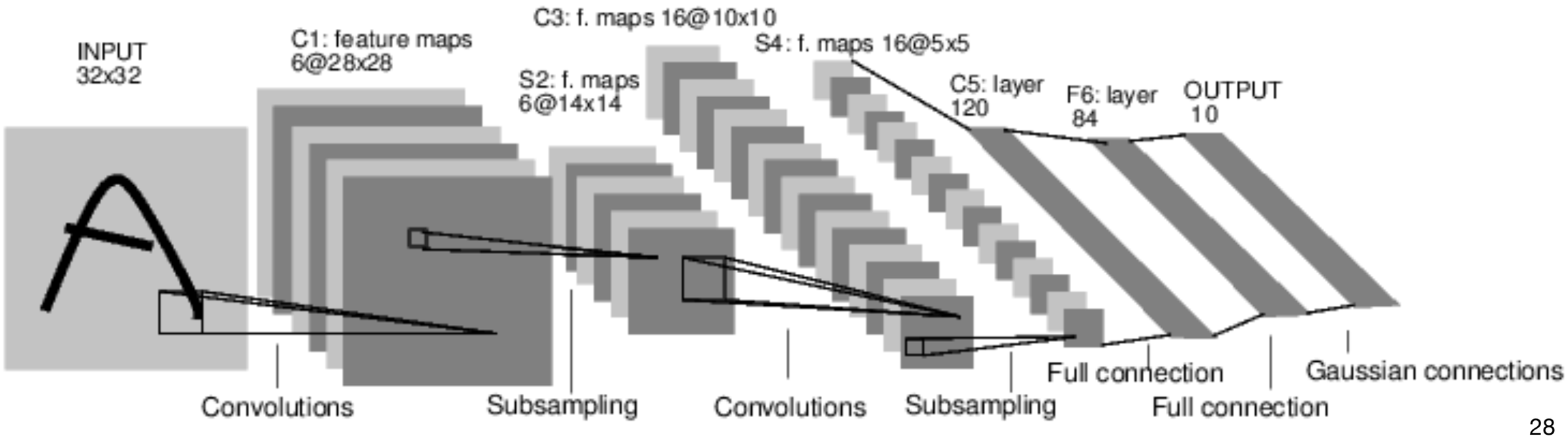
```
tensor([[1, 8, 3],
        [4, 5, 6]])
```

PyTorch: Tensor reshape, squeeze, flatten

- The neural networks we build transform input data to the correct output we are seeking. Shaping and reshaping tensors is a frequent task.
- Suppose we have a tensor `t`. In PyTorch, we can get the shape of a tensor with `t.size()` or `t.shape`, both giving us the row by column shape.
- The number of elements in the tensor can be viewed via `t.numel()`.
- We can reshape the tensor with `t.reshape()` by specifying the row x column shape we want.
- We can squeeze a tensor to remove to dimensions that have a length of 1. Or unsqueeze to add a dimension with a length of 1. The functions are simple `squeeze()` and `unsqueeze()`.
- To flatten, we can first reshape the tensor `t.reshape(1, -1)` then squeeze `t.squeeze()`.
- To combine tensors, we have the `cat()` function.
- Yes, very similar to NumPy!

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

- A bit about the dataset:
  - MNIST (Modified National Institute of Standards and Technology database) is a famous handwritten digits (70,000 images) often used as a benchmark to compare models. It has been widely used and is considered too easy given advancements in image recognition.
- Thus, Fashion-MNIST was created, by the Zalando Research group in 2017 [17].
- Fashion-MNIST has 10 classes of items: top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot [17].
- With PyTorch we can access Fashion-MNIST with the package `torchvision`.

PyTorch: CNN for Fashion-MNIST example

- Prepare the data:
  - Steps: Grab data from source, put data in tensor form, put data into object to make it more accessible
  - For these steps, PyTorch gives us 2 classes:
    - `torch.utils.data.Dataset`: An abstract class to represent a dataset. We extend Dataset to create a concrete subclass.
    - `torch.utils.data.DataLoader`: Wraps a dataset and provide access to the underlying data
- Encapsulated for us, we just need to know: All subclasses of Dataset class must override `__len__`, which provides the size of the dataset, and `__getitem__`, supporting integer indexing in range from 0 to `len(self)` exclusive.

PyTorch: CNN for Fashion-MNIST example

- Prepare the data:
  - Since Fashion-MNIST comes with `torchvision`, we can get an instance of it by specifying the following arguments:
    - `root`: The location on disk where the data is located
    - `train`: If the dataset is the training set
    - `download`: If the data should be downloaded
    - `transform`: A composition of transformations that should be performed on the dataset elements.
- And use the built-in `transforms.ToTensor()` transformation to transform our images into tensors.

```
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()]))
```

PyTorch: CNN for Fashion-MNIST example

Then, we create a `DataLoader` wrapper for the train and test sets.

```
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)
```

See the distribution of labels: Lucky us, Fashion-MNIST is a balanced dataset with 6000 images for each label in the train set.

```
train_set.targets.bincount()
```

```
tensor([6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000])
```

Pass the `train_set` object to `iter()` and `next()` to grab a batch of images from the data. Then unpack batch to images and their associated labels.

```
batch = next(iter(display_loader))
images, labels = batch
```

Plot the batch using a nice function from PyTorch, adjusting to matplotlib's order.

```
grid_img = torchvision.utils.make_grid(images, nrow=10, pad_value=0, padding=2)
plt.imshow(grid_img.permute(1,2,0))
```

PyTorch: CNN for Fashion-MNIST example

- Build the CNN model:
  - We'll focus more on the usage of PyTorch to create object-oriented neural networks than on the networks themselves (plenty of resources on CNNs out there!)
  - To build NNs in PyTorch, we extend the `torch.nn.Module` class from PyTorch.
  - `torch.nn.Module` has a `forward()` method. When we pass a tensor to our network as input, the tensor flows forward though each layer transformation until the tensor reaches the output layer.
  - The goal of the overall transformation is to transform or map the input to the correct prediction output class, and during the training process, the layer weights are updated in such a way that cause the mapping to adjust to make the output closer to the correct prediction.

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.

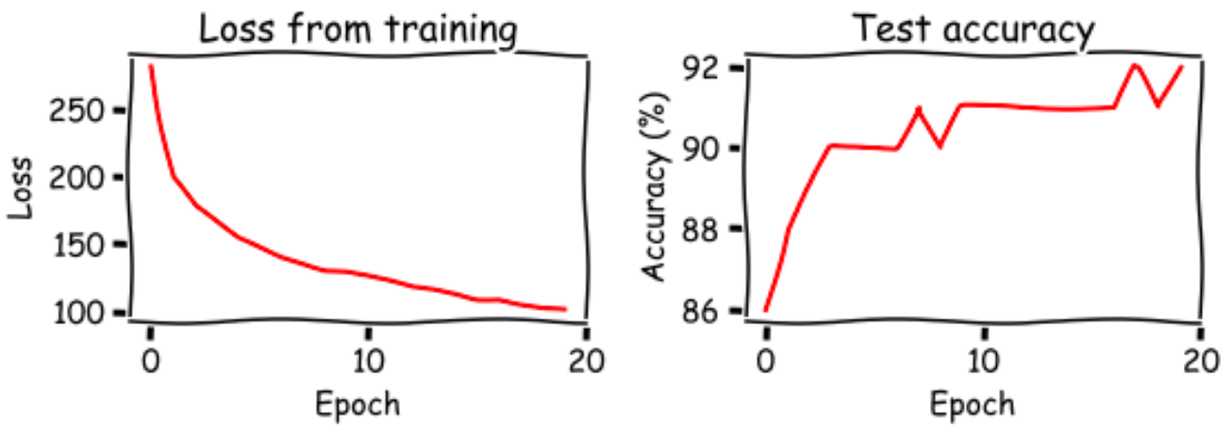
```
for epoch_i in range(num_epochs):
    model.train()
    batch_loss = 0
    for batch_i, (images, labels) in enumerate(train_loader):
        sys.stdout.write(f"\rBatch {batch_i+1}/{len(train_loader)}")
        sys.stdout.flush()

        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.