PYTHON AS DEEP LEARNING PLATFORM

Deep Learning Libraries in Python

Installing PyTorch on Your Computer

Deep Learning Libraries in Python

Deep Learning Libraries for Python

PyTorch

Developed by Facebook (Meta)

- Provides modules easy to combine
- Easy to edit network
- Many pre-trained models
- Seamless integration into Python/Numpy framework

TensorFlow

Developed by Google

- Provides Tensorboard for visualization
- Supports multiple languages (C++, Java, R)
- Slightly less intuitive to use than PyTorch
- Great community support
- Tensorflow Lite can run models on mobile devices

mxnet

Developed by Apache

- Supported by Amazon Web Service
- Supports many languages
- Fast and flexible for running DL algorithms
- Features advanced GPU support
- Popular among industrial projects

Deep Learning Libraries for Python

PyTorch

Developed by Facebook (Meta)

- Provides modules easy to combine
- Easy to edit network
- Many pre-trained models
- Seamless integration into Python/Numpy framework

Framework for this class

TensorFlow

Developed by Google

- Provides Tensorboard for visualization
- Supports multiple languages (C++, Java, R)
- Slightly less intuitive to use than PyTorch
- Great community support
- Tensorflow Lite can run models on mobile devices

mxnet

Developed by Apache

- Supported by Amazon Web Service
- Supports many languages
- Fast and flexible for running DL algorithms
- Features advanced GPU support
- Popular among industrial projects

Installing PyTorch on Your Computer

Installing PyTorch on Your Computer

Windows - **Anaconda Prompt**

Mac and Linux - **Terminal**

If your computer has one of [these GPUs] (CUDA-Enabled GeForce and TITAN Products):

Anaconda Prompt (anaconda3)

```
(base) C:\Users\Jimin>conda install pytorch torchvision torchaudio cudatoolkit=11.3 -c pytorch
```

else (uses CPU instead):

Anaconda Prompt (anaconda3)

```
(base) C:\Users\Jimin>conda install pytorch torchvision torchaudio cpuonly -c pytorch
```

Note: Make sure you have the latest graphics driver installed

Note: For non-local options, see slides 43-45

Verify PyTorch Installation

1.

```python
import torch
```

```python
x = torch.rand(5, 3)
print(x)
```

```
tensor([[0.9278, 0.0797, 0.3936],
        [0.5075, 0.4611, 0.5649],
        [0.3202, 0.1863, 0.5423],
        [0.3864, 0.6935, 0.5528],
        [0.5596, 0.8721, 0.6153]])
```

Import PyTorch

Generate randomly initialized torch tensor

2.

```python
torch.cuda.is_available()
```

```
True
```

Check if your GPU driver, CUDA is enabled and accessible by PyTorch

NEURAL NETWORK WORKFLOW IN PYTORCH

Example Task: Simple Linear Regression

Neural Net Workflow

Prepare Data

Define Model

Select Hyperparameters

Identify Tracked Values

Train Model

Visualization and Evaluation

Neural Net Workflow

Prepare Data

Define Model

Select Hyperparameters

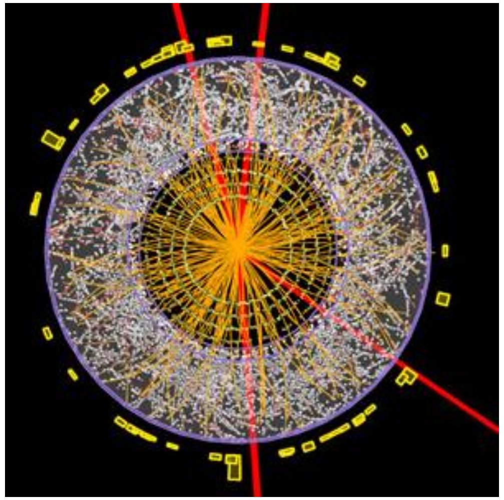
Identify Tracked Values

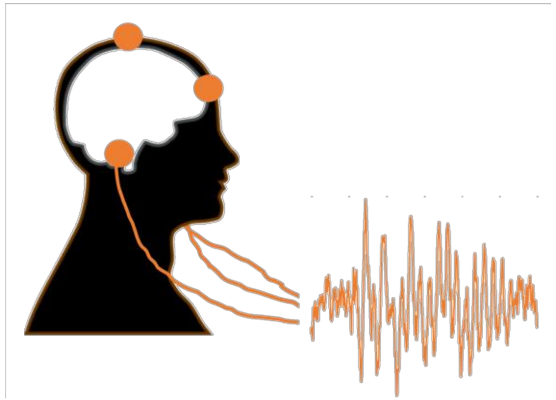
Train Model

Visualization and Evaluation

Prepare Data

Raw data

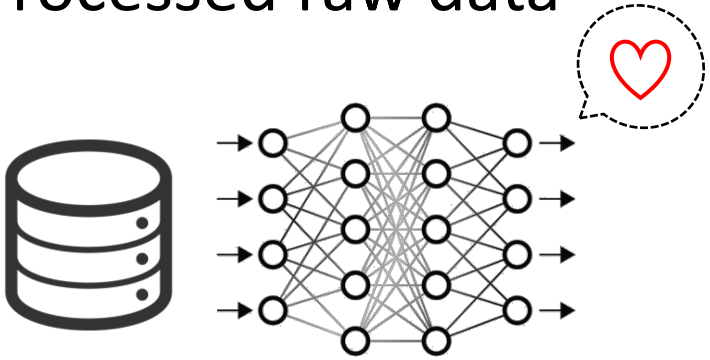




Example 1)
Particle feature data from ATLAS detector @ LHC

Example 2)
Neural recordings from the brain

Processed raw data

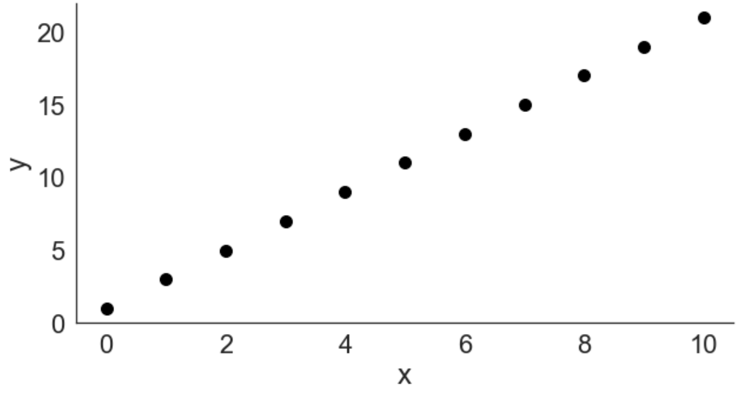


- Remove outliers
- Normalization
- Split train, validation, test sets
- etc

For a successful neural net model, dataset should be Large, Clean and Diverse
-Andrej Karpathy (Director of Tesla Autopilot AI)

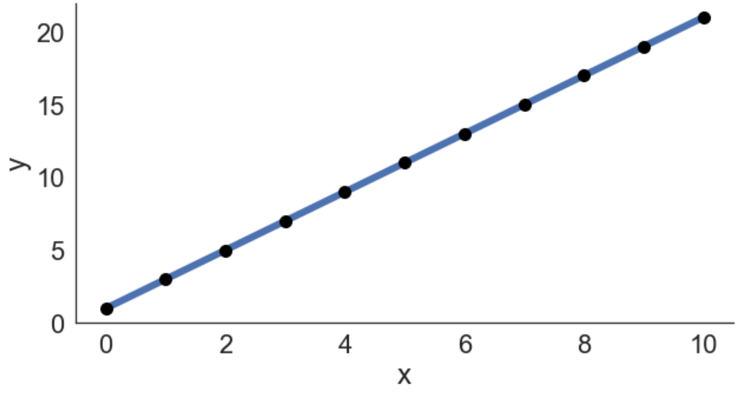
Example Task: Linear Regression

Dataset



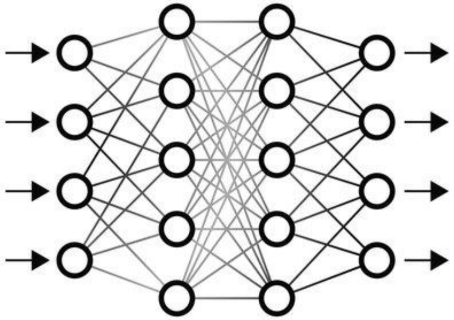
x = input
y = output

Rule to be learned



$y = 2x + 1$

5



11

Input

Neural net model

Output

2(5) + 1

Prepare Data (example task)

```python
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
import torch
```

Import necessary libraries

```python
x_train = np.arange(11, dtype = np.float32)
x_train = x_train[:, np.newaxis]

y_train = (2 * x_train) + 1
```

Generate training data for x and y

```python
print(x_train)
```

```
[[ 0.]
 [ 1.]
 [ 2.]
 [ 3.]
 [ 4.]
 [ 5.]
 [ 6.]
 [ 7.]
 [ 8.]
 [ 9.]
 [10.]]
```

x

Inputs (features)

```python
print(y_train)
```

```
[[ 1.]
 [ 3.]
 [ 5.]
 [ 7.]
 [ 9.]
 [11.]
 [13.]
 [15.]
 [17.]
 [19.]
 [21.]]
```

y

Output targets

Print the training data (x_train, y_train)

Neural Net Workflow Steps

Prepare Data

Define Model

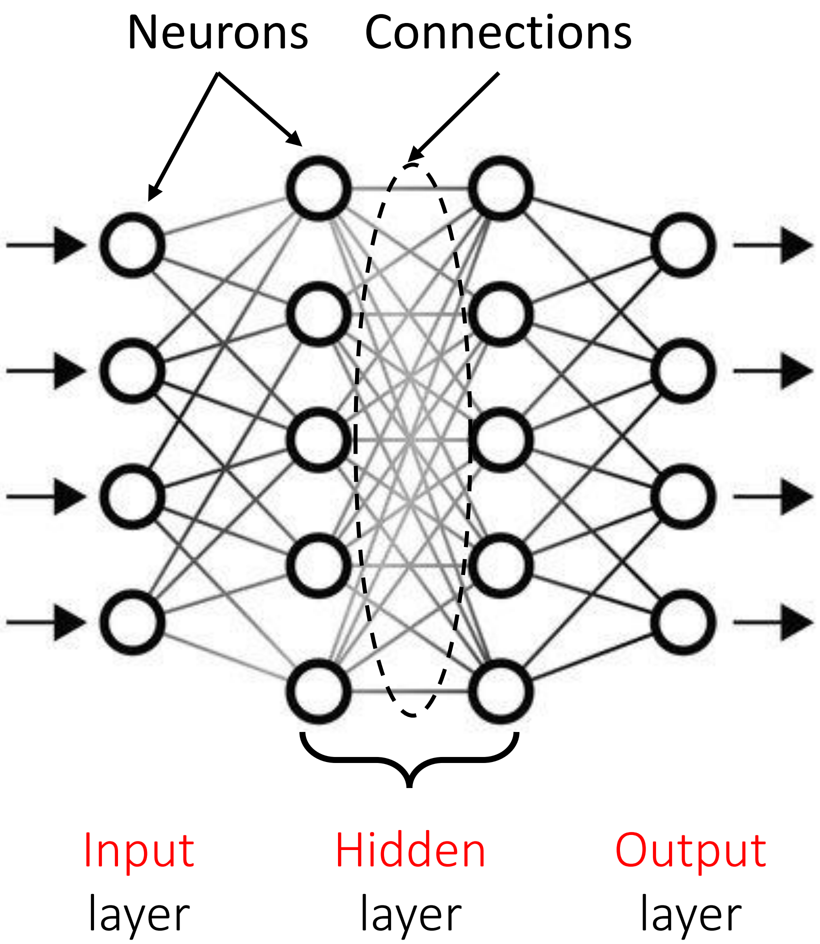
Select Hyperparameters

Identify Tracked Values

Train Model

Visualization and Evaluation

Define Model



Dimensions (number of neurons) of Input and output layers

Number of hidden layers

Number of neurons for each hidden layer

Other network features

Define Model (example task)

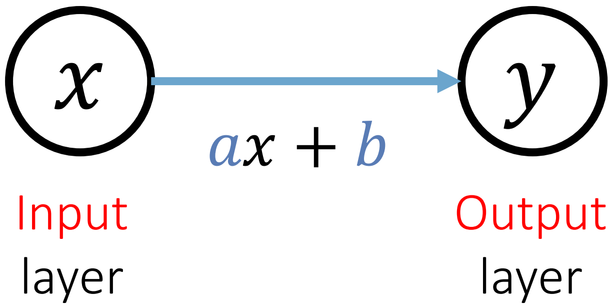
```python
class linearRegression(torch.nn.Module):
    def __init__(self, input_dim, output_dim):
        super(linearRegression, self).__init__()
        self.linear = torch.nn.Linear(input_dim, output_dim)
    def forward(self, x):
        out = self.linear(x)
        return out
```

Define a model class

Initialize the model with a linear layer with input/output dimension

Define a feed forward function describing the information flow within the network

Neural Network Diagram



Training goal

$a \cong 2$ (weight)

$b \cong 1$ (bias)

Input layer dimension = 1

Output layer dimension = 1

No hidden layers

Neural Net Workflow

Prepare Data

Define Model

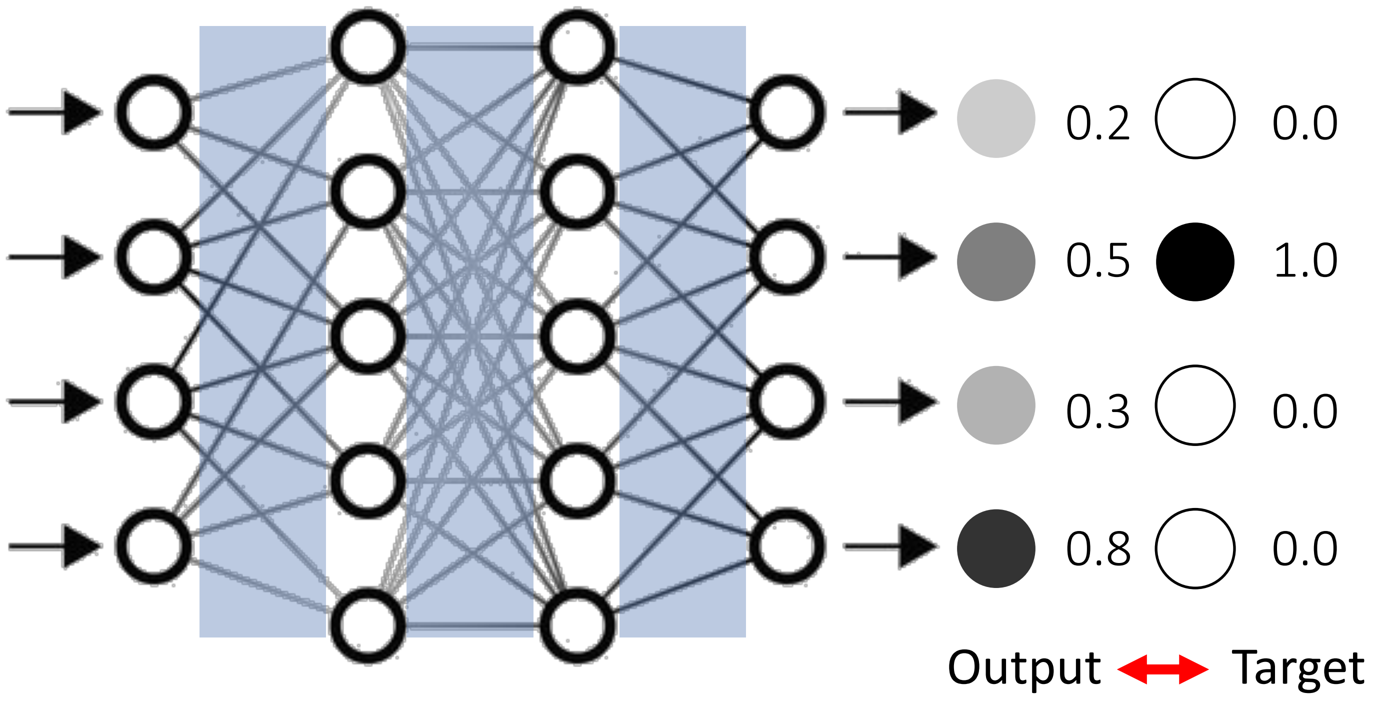
Select Hyperparameters

Identify Tracked Values

Train Model

Visualization and Evaluation

Select Hyperparameters



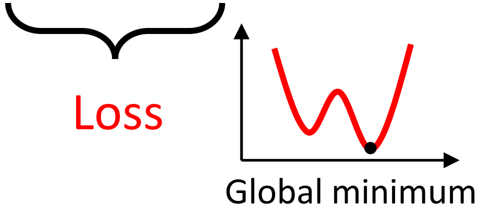
Optimizer

Loss function

Learning rate

Epochs (Number of iterations)

Other hyperparameters



Optimizer ***minimizes*** the loss throughout epochs by changing the connection weights/biases at the pace of learning rate

Select Hyperparameters (example task)

```
model = linearRegression(input_dim = 1, output_dim = 1)

learning_rate = 0.01
epochs = 100

loss_func = torch.nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr = learning_rate)

if torch.cuda.is_available():
    model.cuda()
```

Define the model with input/output layer dimensions

Define learning rate, epochs (# of iterations)

Define loss function (MSE) and optimizer (Gradient Descent)

If using GPU, transfer the model to GPU memory

Neural Net Workflow

Prepare Data

Define Model

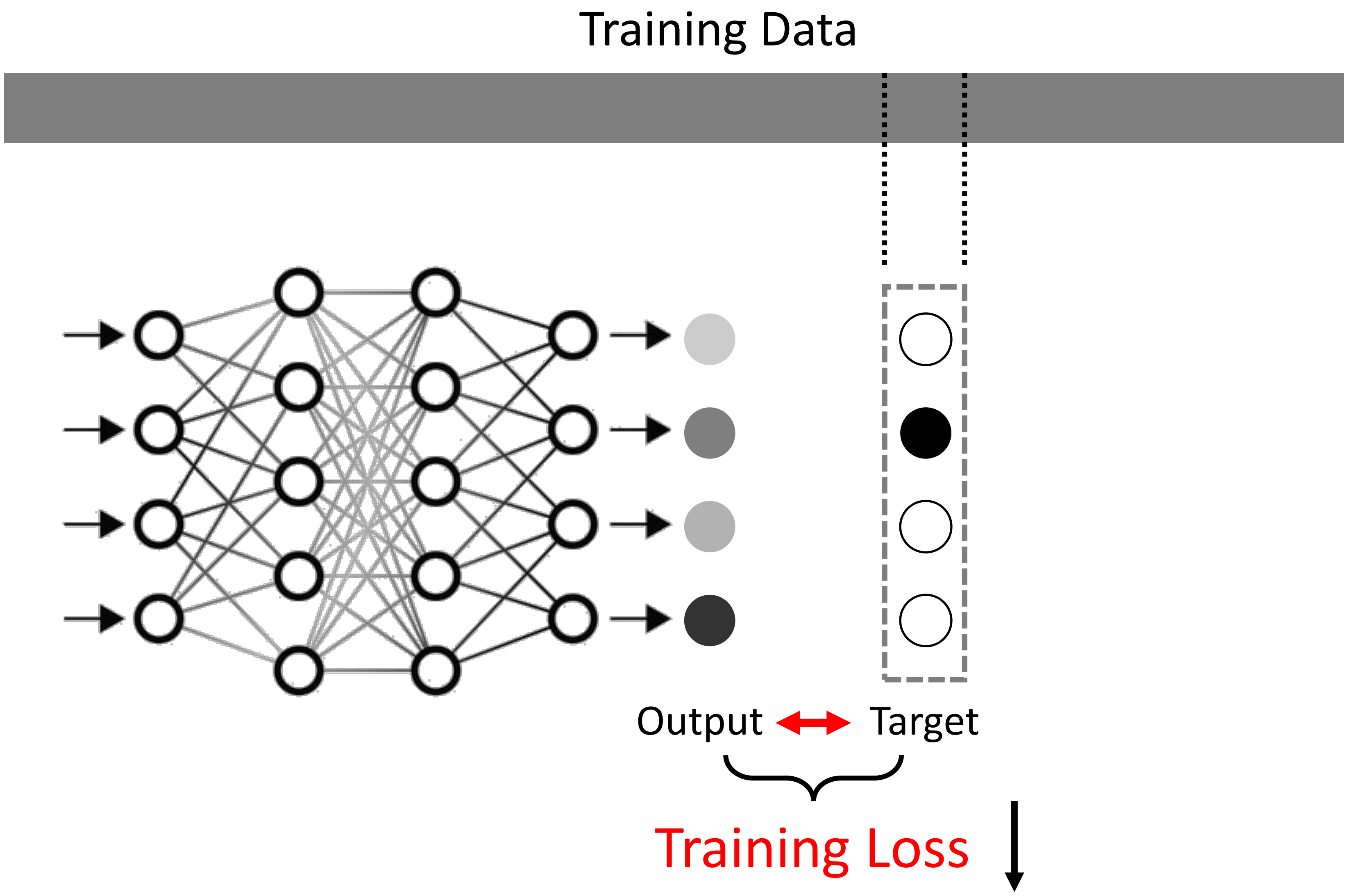
Select Hyperparameters

Identify Tracked Values

Train Model

Visualization and Evaluation

Identify Tracked Values



Identify Tracked Values (example task)

```
train_loss_list = []
```

Create an **empty list** or **arrays** to contain training loss

Neural Net Workflow

Prepare Data

Define Model

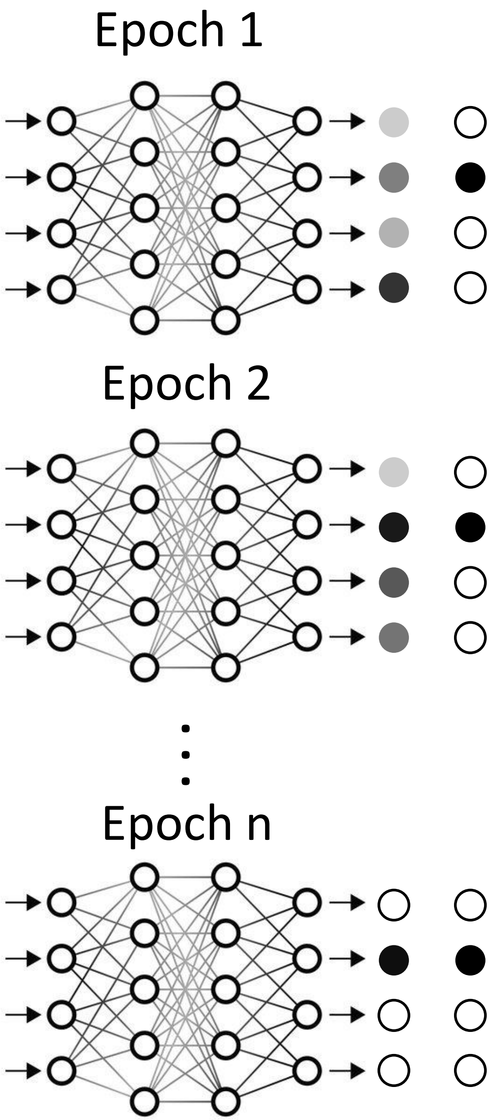
Select Hyperparameters

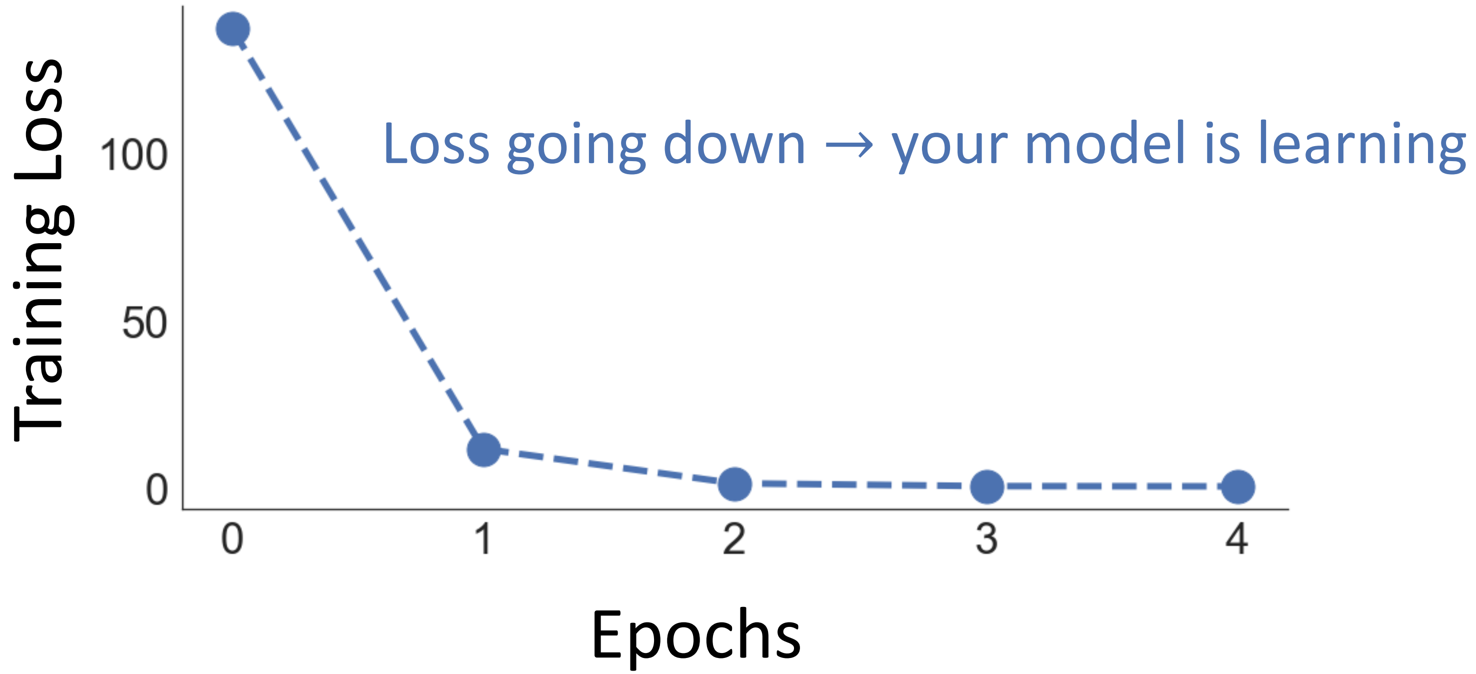
Identify Tracked Values

Train Model

Visualization and Evaluation

Train the Model





1 Epoch : Model goes through a full training dataset

Train the Model (example task)

```
if torch.cuda.is_available():
    inputs = torch.from_numpy(x_train).cuda()
    targets = torch.from_numpy(y_train).cuda()
else:
    inputs = torch.from_numpy(x_train)
    targets = torch.from_numpy(y_train)

for epoch in range(epochs):

    optimizer.zero_grad()

    outputs = model(inputs)

    loss = loss_func(outputs, targets)

    train_loss_list.append(loss.item())

    loss.backward()

    optimizer.step()

    print('epoch {}, loss {}'.format(epoch, loss.item()))
```

Convert inputs and targets into [PyTorch tensors]
(GPU)

(CPU)

This ensures learning from each epoch is separate

Forward pass the inputs through the network to produce outputs

Compute the loss and append to tracking list

Compute how much changes to be made to weights/biases

Update the weights/biases

Print the epoch # and loss value

Neural Net Workflow

Prepare Data

Define Model

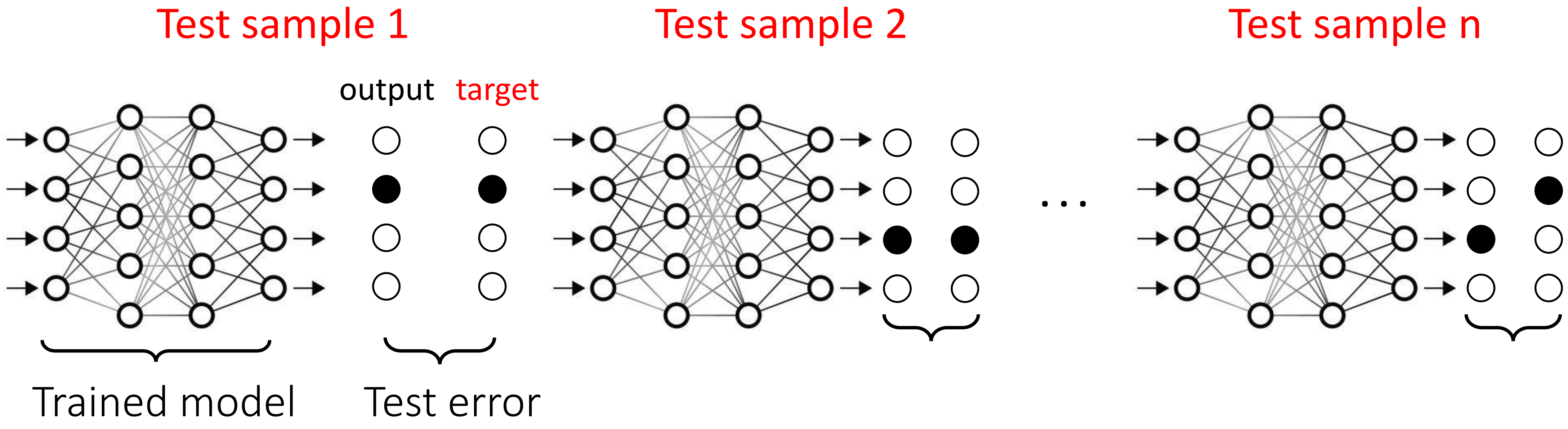
Select Hyperparameters

Identify Tracked Values

Train Model

Visualization and Evaluation

Visualization and Evaluation



Commonly used evaluation metrics:
Mean squared error (MSE), Classification accuracy, etc

Typically, your trained model is tested on an unknown dataset outside of training data
(More on this in Lab 3)

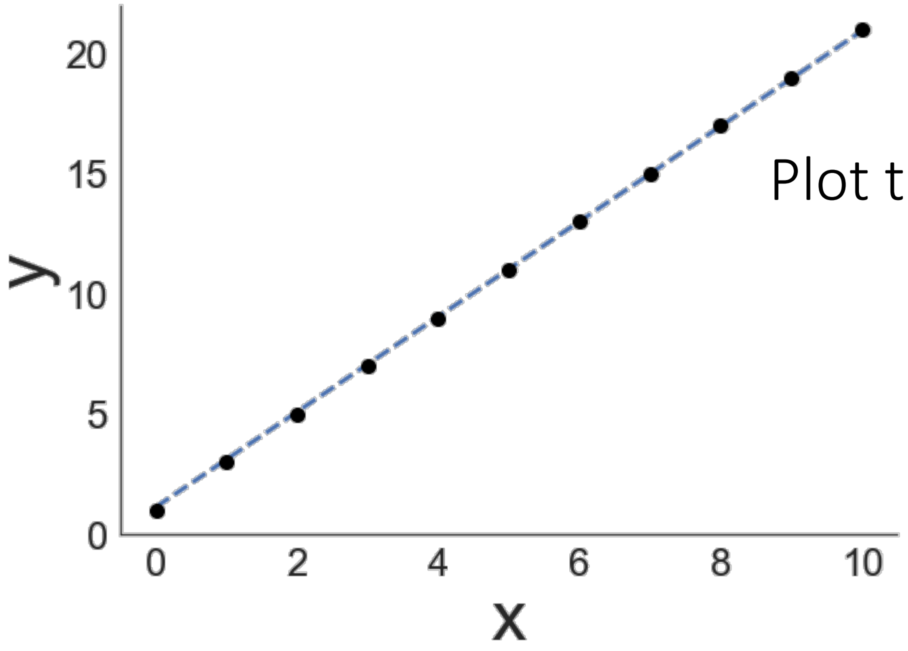
Visualization and Evaluation (example task)

```python
with torch.no_grad():

    if torch.cuda.is_available():

        predicted = model(torch.from_numpy(x_train).cuda()).cpu().numpy()

    else:

        predicted = model(torch.from_numpy(x_train)).numpy()

    print(predicted)
    print("a: " + str(model.linear.weight.cpu().numpy()), "b: " + str(model.linear.bias.cpu().numpy()))
```

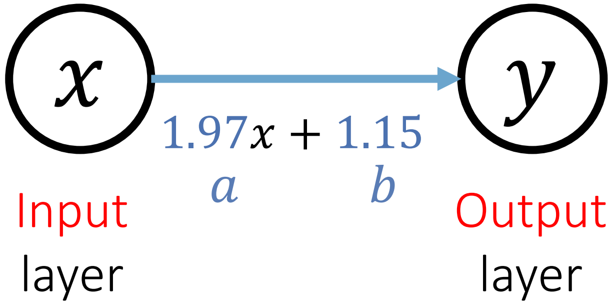
Feed the trained model with the original x_train to produce y predictions

```
[[ 1.1490046]
 [ 3.12752  ]
 [ 5.1060357]
 [ 7.0845513]
 [ 9.063067 ]
 [11.041583 ]
 [13.020099 ]
 [14.998614 ]
 [16.977129 ]
 [18.955645 ]
 [20.93416  ]]
a: [[1.9782206]] b: [1.1490046]
```

Predicted y



Plot the **targets** vs prediction



PYTHON CONCEPTS FOR PYTORCH

Python Classes

PyTorch Tensors

Python Classes

Python Classes

Class: Pokemon

Name: Pikachu
Type: Electric
Health: 70

Attack()
Dodge()
Evolve()

Attributes

Methods

Class: Pokemon

Name: Espeon
Type: Psychic
Health: 90

Attack()
Dodge()
Evolve()

Class: collection of objects (**pokemon**) of the same type (**pokemon class**)

Python Classes

Creating a class “Pokemon”

Initialize the Pokemon object with provided Name, Type and Health parameters

Add functions for each method

```python
class Pokemon():
    def __init__(self, Name, Type, Health):
        self.Name = Name
        self.Type = Type
        self.Health = Health

    def whats_your_name(self):
        print("My name is " + self.Name + "!")

    def attack(self):
        print("Electric attack! Zap!!")

    def dodge(self):
        print("Pikachu Dodge!")

    def evolve(self):
        print("Evolving to Raichu!!")
```

Create a Pokemon object named “pk1”

```python
pk1 = Pokemon(Name = "Pikachu", Type = "Electric", Health = 70)
```

Name can be inferred using .Name directive

```python
pk1.Name
```

```
'Pikachu'
```

Calling each function within the class prints the intended statements

```python
pk1.whats_your_name()
```

```
My name is Pikachu!
```

```python
pk1.attack()
```

```
Electric attack! Zap!!
```

Python Classes: super() function

Parent class

```python
class linearRegression(torch.nn.Module):
    def __init__(self, inputSize, outputSize):
        super(linearRegression, self).__init__()
        self.linear = torch.nn.Linear(inputSize, outputSize)

    def forward(self, x):
        out = self.linear(x)
        return out
```

Initializing the **parent class (torch.nn.Module)** allows us to use **attributes/methods** from nn.Module - e.g. nn.Linear()

More on Python classes: http://introtopython.org/classes.html

PyTorch Tensors

PyTorch Tensors

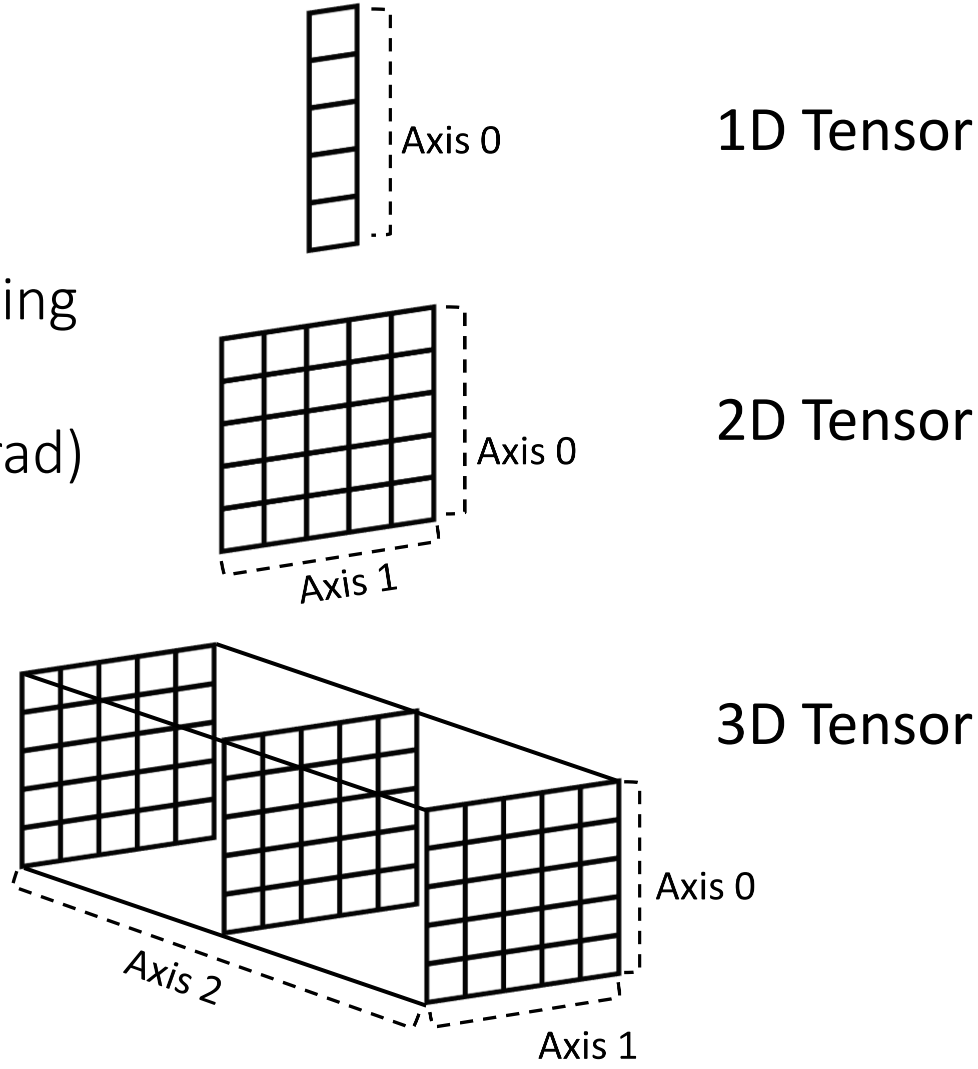
Main data structure for PyTorch

Like NumPy arrays, but optimized for machine learning

- Can be processed by both CPU and GPU
- Optimized for automatic differentiation (auto-grad)

Three main attributes

- shape – dimensions of array
- datatype – form of each entry (float, int, etc)
- device – CPU or cuda (GPU)



PyTorch Tensors vs NumPy Arrays

Creating a NumPy array

```
array1 = np.array([1,2,3,4])
print(array1, type(array1))
```

```
[1 2 3 4] <class 'numpy.ndarray'>
```

Creating a torch tensor

```
tensor1 = torch.tensor([1,2,3,4])
print(tensor1, type(tensor1))
```

```
tensor([1, 2, 3, 4]) <class 'torch.Tensor'>
```

NumPy Array -> PyTorch Tensor

```
array1_torch = torch.from_numpy(array1)
print(array1_torch, type(array1_torch))
```

```
tensor([1, 2, 3, 4], dtype=torch.int32) <class 'torch.Tensor'>
```

PyTorch Tensor -> NumPy Array

```
tensor1_numpy = tensor1.numpy()
print(tensor1_numpy, type(tensor1_numpy))
```

```
[1 2 3 4] <class 'numpy.ndarray'>
```

More on Numpy arrays vs torch tensors:
https://rickwierenga.com/blog/machine%20learning/numpy-vs-pytorch-linalg.html

Handling Torch Tensors

Moving tensors to CPU .cpu()

```python
tensor1_cpu = tensor1.cpu()
print(tensor1_cpu.device)
```

```
cpu
```

Moving tensors to GPU .gpu()

```python
tensor1_gpu = tensor1.cuda()
print(tensor1_gpu.device)
```

```
cuda:0
```

Disabling gradient calculation torch.no_grad()

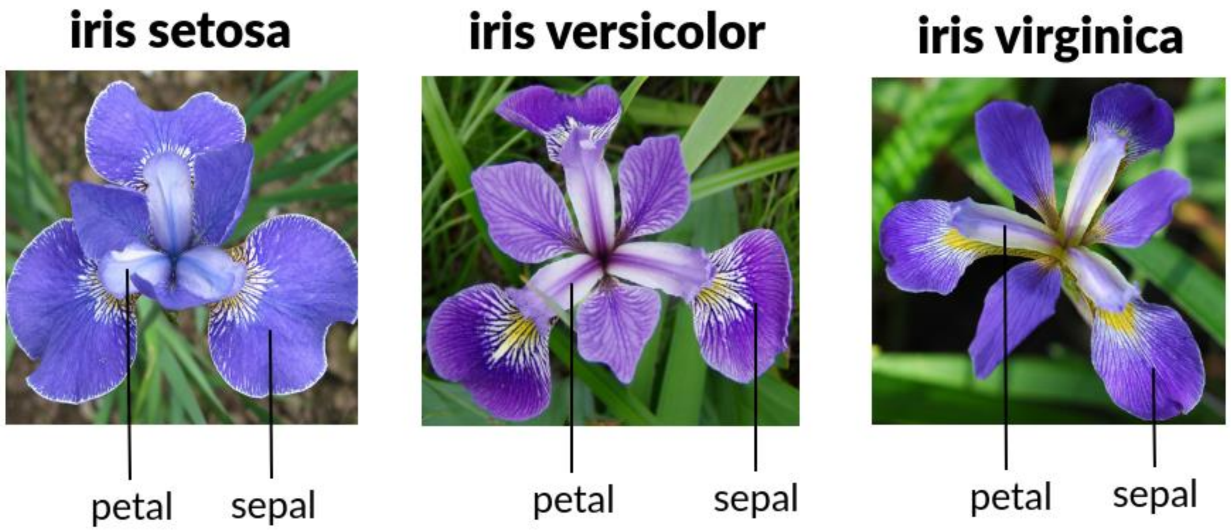
```python
with torch.no_grad():

    predicted = model(torch.from_numpy(x_train)).numpy()
```

(useful when you just want to inspect the values inside tensors)

LAB 2 ASSIGNMENT:

Iris Classification using Regression

Iris Dataset

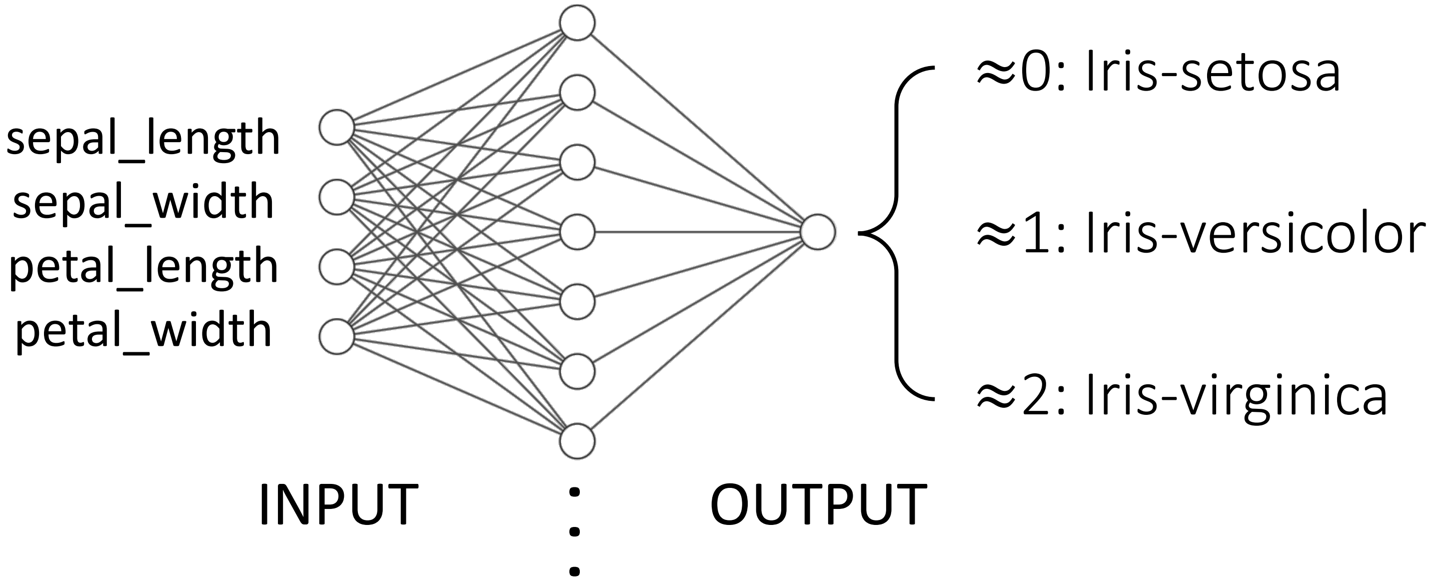


	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

Features Labels (Targets)

- 150 samples
- 4 features/sample
  - Sepal length (cm)
  - Sepal width (cm)
  - Petal length (cm)
  - Petal width (cm)
- 3 labels
  - Iris setosa (0)
  - Iris versicolor (1)
  - Iris virginica (2)

Exercise 1: Iris Classification using Regression



In this exercise, you will train a neural network with a single hidden layer consisting of linear neurons to perform regression on iris datasets.

Your goal is to **achieve a training accuracy of >90% under 50 epochs**.

You are free to experiment with different data normalization methods, size of the hidden layer, learning rate and epochs.

You can round the output value to an integer (e.g. 0.34 -> 0, 1.78 -> 2) to compute the model accuracy.

Demonstrate the performance of your model via plotting the training loss and printing out the training accuracy.

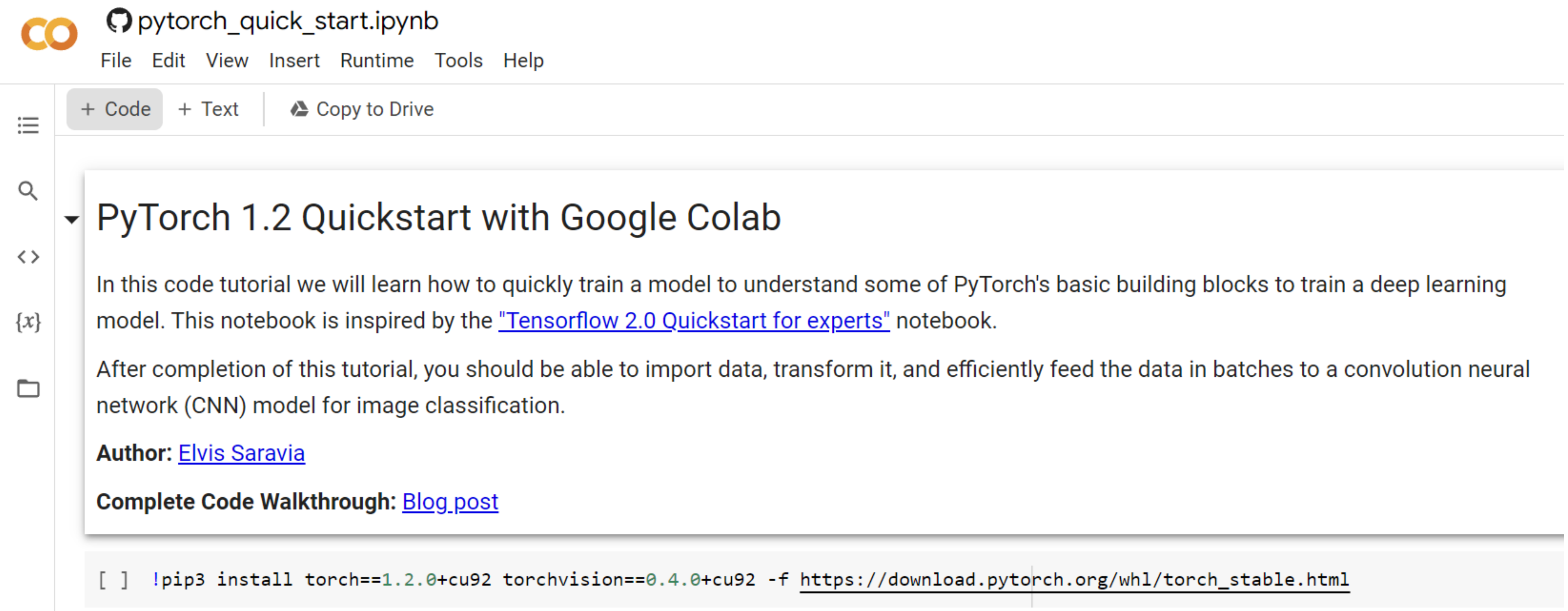
SUPPLEMENTARY:

ADDITIONAL PLATFORMS

Google Colab

Google Cloud

Google Colab



Quickstart Colab Notebook

https://colab.research.google.com/github/omarsar/pytorch_notebooks/blob/master/pytorch_quick_start.ipynb

Google Cloud

Suite of cloud computing services offered by Google

- Offers AI Platform for deploying DL models
- Support Jupyter Notebook instances
- Provide instances with DL libraries
- Fully customizable hardware spec with state-of-the-art components
- Monthly charge for the service (Free with Google cloud credits)

Setup tutorial → https://cloud.google.com/deep-learning-vm/docs/pytorch_start_instance