PyTorch

12 Jun

Gene
Aom

✨ Motivation of Machine Learning

What is Machine Learning?

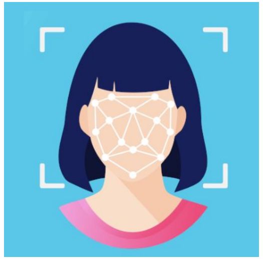
Machine learning is how we teach the machine to **process the raw data** and get some **useful information or prediction** from it.

Why do we need to train the model?

At first, the model makes a prediction based on guesses. To improve the accuracy, we need to train the model massively so that it can learn patterns from the data and **minimize prediction errors** over time.

Application of ML

Robot, vision, transformer, graphic, chatbot



🔥 Introduction to PyTorch

What is PyTorch?

- PyTorch is an **open-source machine learning framework** developed by **Meta (Facebook AI)**.
- It provides tools to **build, train, and deploy deep learning models**.
- Known for its **flexibility, ease of use**, and **dynamic computation**.

Why Use PyTorch?

- 🧠 **Deep Learning**: Build neural networks for tasks like image classification, natural language processing, and reinforcement learning.
- 🧪 **Research-Friendly**: Designed with researchers in mind; easy to experiment and prototype.
- ⚡ **Dynamic**: Allows for more intuitive debugging and flexible model design.
- 🚀 **Production Ready**: Supports deployment to mobile, servers, and edge devices.

🧪 PyTorch Quickstart Tutorial — Full Overview

This notebook gives a hands-on walkthrough of a full PyTorch deep learning workflow:

full-workflow

1. Working with Data
2. Creating Models
3. Training the Model
4. Testing the Model
5. Saving & Loading Models

📊 SECTION 1: Tensors in PyTorch

what is tensor

What are Tensors?

- Core data structure in PyTorch
- Similar to NumPy arrays but support GPUs and autograd

Creating Tensors:

- From data: `torch.tensor([[1, 2], [3, 4]])`
- From NumPy: `torch.from_numpy(np_array)`
- With defaults: `torch.ones()`, `torch.rand()`, `torch.zeros()`

Tensor Attributes: tensor.shape, tensor.dtype, tensor.device

Operations:

- Indexing: `tensor[0]`, `tensor[:, 0]`
- Concatenation: `torch.cat([...], dim=1)`
- Arithmetic: `tensor + 5`, `tensor @ tensor.T`
- In-place ops: `tensor.add_(1)`

Bridge with NumPy:

- Tensors and NumPy arrays share memory
- Changes in one reflect in the other

📦 SECTION 2: Dataset & DataLoader

how to prepare data

🔍 Why They Matter

- In machine learning, you often deal with **large datasets**.
- PyTorch provides two essential tools to **manage and feed data efficiently** into your model:

🧱 `Dataset` — What is it?

- Think of it as a **data container**:
  - Holds **features (images, text, etc.)** and **labels**.
  - Behaves like a list: you can index to get one sample.
- Can be:
  - **Built-in** datasets (like MNIST, CIFAR10).
  - **Custom** datasets (for your own files or formats).

🚚 `DataLoader` — What does it do?

- Think of it as a **data delivery service**:
  - Feeds data from the `Dataset` in **mini-batches**.
  - Can **shuffle** the data to prevent model overfitting.
  - Can load data **in parallel (multiprocessing)** to speed things up.

🧠 Why Use Both?

- `Dataset`: Cleanly **stores and prepares** your data.
- `DataLoader`: Efficiently **feeds it to your model** during training.

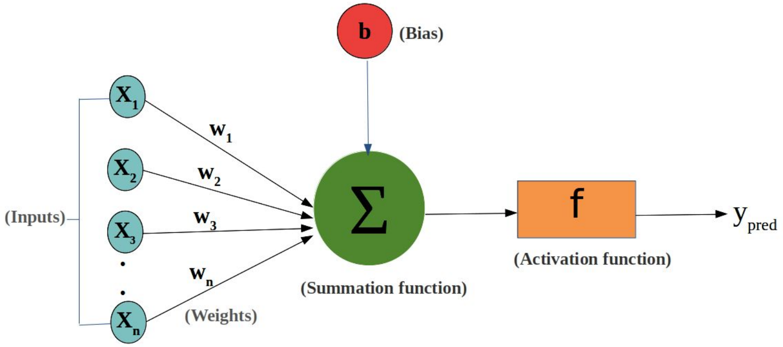
Together, they help you build **modular**, **scalable**, and **maintainable** training pipelines.

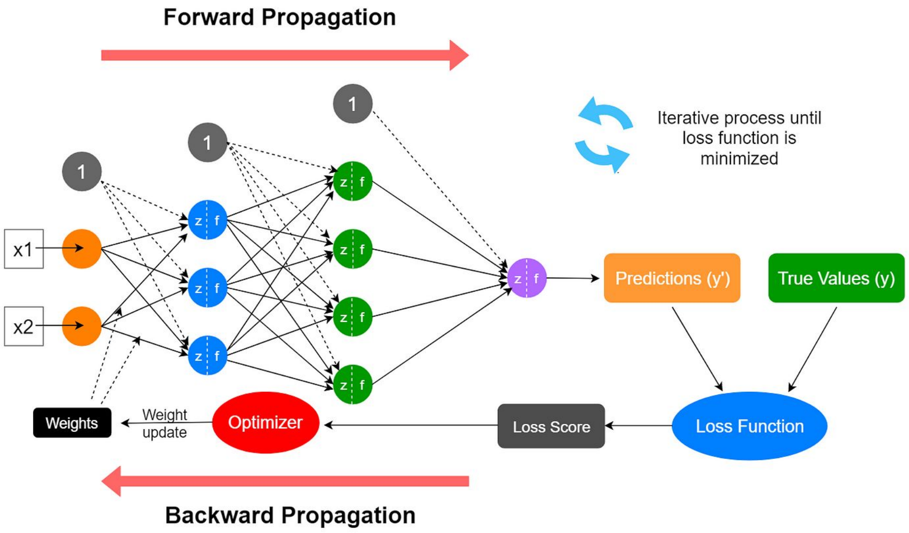
🧠 SECTION 3: Build the Neural Network

how to construct a network

🔧 What is a Neural Network? Same as last workshop!!!

- A **neural network** is a collection of **layers** (or modules) that process input data to generate outputs.
- Think of it as a **function made of multiple transformations**.





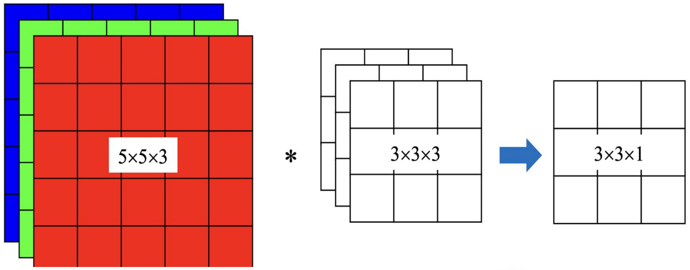
🧠 SECTION 3: Build the Neural Network

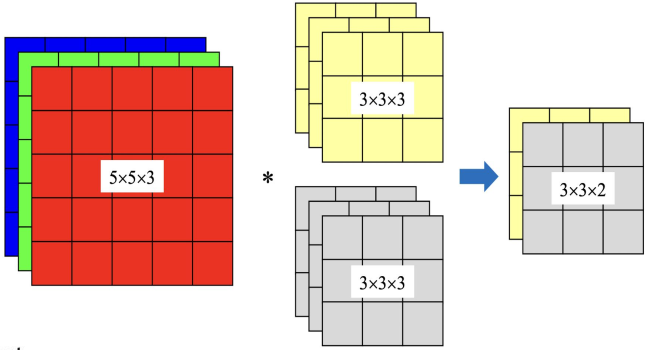
how to construct a network

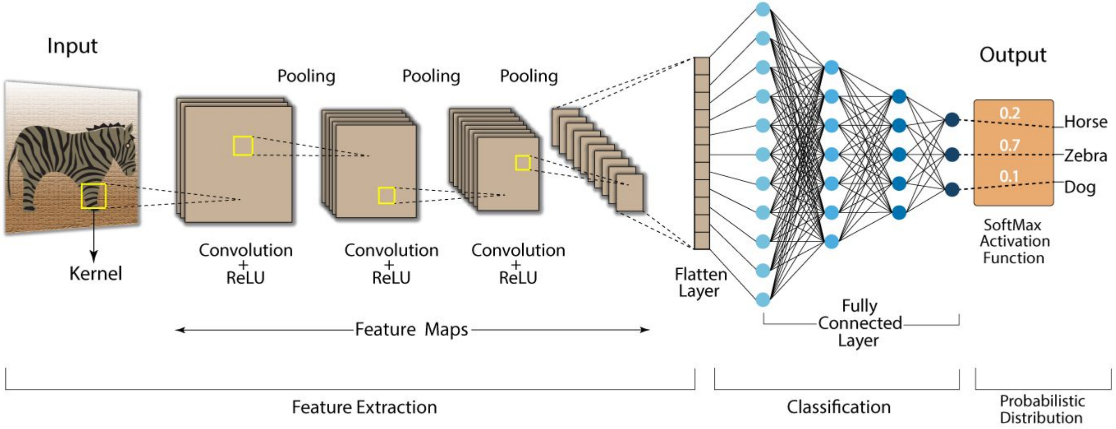
🔧 What is a Neural Network? Same as last workshop!!!

- A **neural network** is a collection of **layers** (or modules) that process input data to generate outputs.
- Think of it as a **function made of multiple transformations**.

CNN







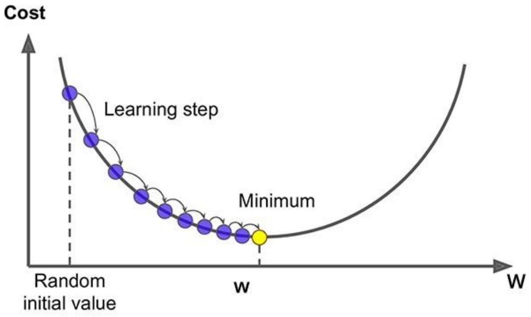
CNN Architecture
Vgg16
AlexNet
ResNet…

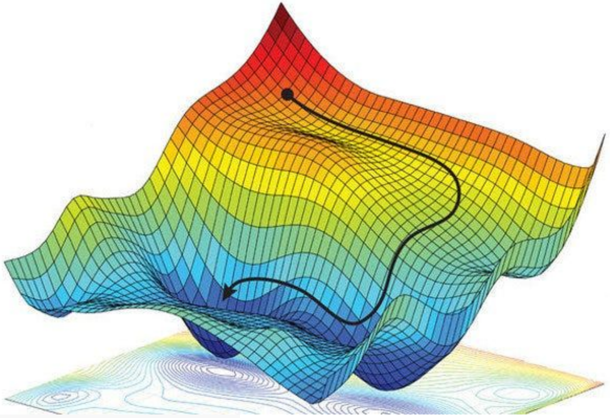
Apart From CNN: RNN, GAN, Transformer, ..

📉 SECTION 4: Loss and Gradient

gradient

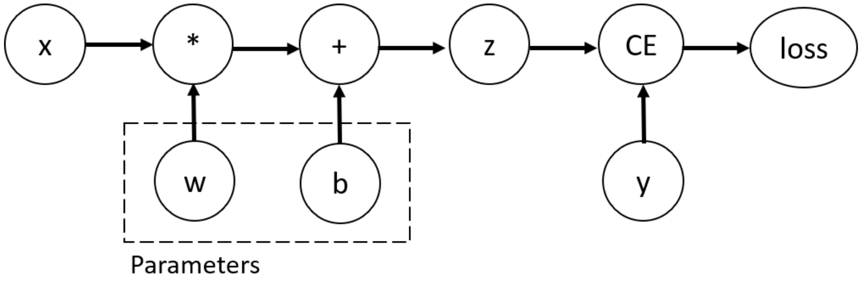
Recall, what is gradient???





```
import torch

x = torch.ones(5)  # input tensor
y = torch.zeros(3)  # expected output
w = torch.randn(5, 3, requires_grad=True)
b = torch.randn(3, requires_grad=True)
z = torch.matmul(x, w)+b
loss = torch.nn.functional.binary_cross_entropy_with_logits(z, y)
```

```
loss.backward()
```



w1 = w0 +lr*gradient

✅ SECTION 5: Optimizer

Optimize Parameter

What is an Optimizer?

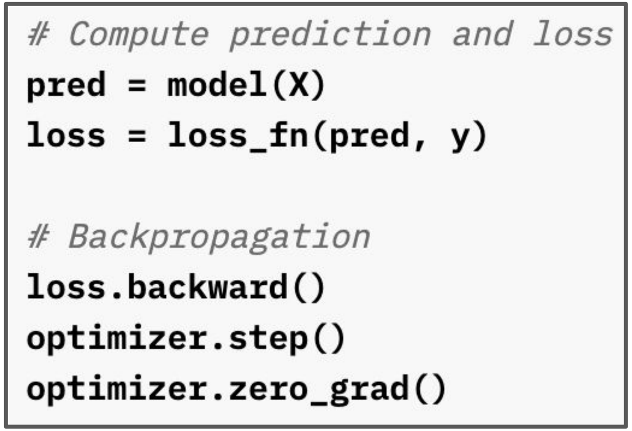
An optimizer adjusts the parameters (weights and biases) of a neural network during training to minimize the loss function, which measures prediction errors.

Common Optimizers

1. **Stochastic Gradient Descent (SGD)**: Updates parameters based on small batches of data.
2. **Momentum**: Enhances SGD by adding a fraction of the previous update.
3. **Adam**: Combines features of SGD and RMSProp, adjusting learning rates for each parameter.
4. **RMSProp**: Adapts learning rates based on recent gradients.

Key Functions

- `optimizer.step()`: Updates model parameters using calculated gradients.
- `optimizer.zero_grad()`: Clears old gradients to prevent accumulation.



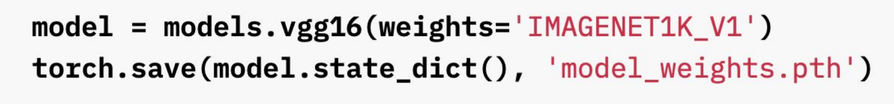
```
# Compute prediction and loss
pred = model(X)
loss = loss_fn(pred, y)

# Backpropagation
loss.backward()
optimizer.step()
optimizer.zero_grad()
```

✅ SECTION 6: Save & Load Model

How to save and load model

Saving Data

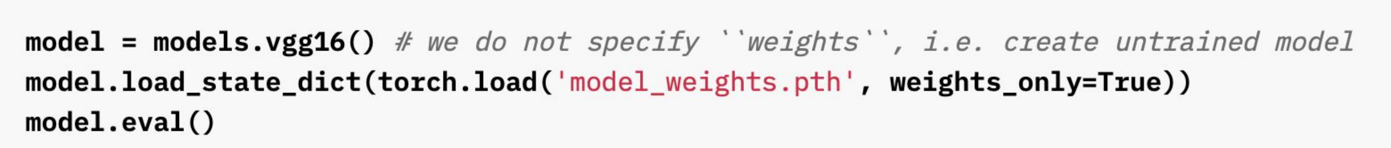
- After training the model, we can save its learned parameters (weights) for reuse in other applications.
- Weights will be stored in *.pth* file and use it during loading.



```
model = models.vgg16(weights='IMAGENET1K_V1')
torch.save(model.state_dict(), 'model_weights.pth')
```

Loading Data

- To reuse the model, we need to recreate the same model architecture, then load the weights into the new instance.
- Don't forget to call `model.eval()` to turn on the evaluation mode. This will turn off the training weights behaviors and keep the final weights fixed for real use.



```
model = models.vgg16() # we do not specify ``weights``, i.e. create untrained model
model.load_state_dict(torch.load('model_weights.pth', weights_only=True))
model.eval()
```