



Deep Learning Toolkit (*PyTorch & Timm*)

Rowel Atienza, PhD

University of the Philippines

github.com/roatienza

2023



PyTorch

<https://pytorch.org/>

<https://github.com/yunjey/pytorch-tutorial>

Why PyTorch?

Easy to build, train, validate and debug models

Available implementation and pre-trained weights of state-of-the-art (SOTA) models

Huge community of users

Production-ready

PyTorch now under Linux Foundation

PyTorch is now 2.0 – better and faster model training and inference

Install and Test

```
pip install torch torchvision torchaudio
```

Activate python3

```
>>> import torch
```

```
>>> print(torch.__version__)
```

```
2.0.1
```

Introducing PyTorch for Deep Learning

`torch.Tensor`
Model Inference

Tensor

<https://pytorch.org/docs/stable/tensors.html>

Tensor – PyTorch Data Structure

Numpy data structure: ndarray

```
>>> a = np.ones((1,2))  
>>> type(a)  
<class 'numpy.ndarray'>
```

PyTorch data structure: Tensor

```
>>> b = torch.ones((1,2))  
>>> type(b)  
<class 'torch.Tensor'>
```

Numpy ndarray vs PyTorch Tensor

Data Structure	CPU	AI Accelerator		
		GPU	TPU	IPU
Numpy ndarray	✓	✗	✗	✗
PyTorch Tensor	✓	✓	✓	✓

Tensor operations/attributes

Initialize:

```
a = torch.tensor(
    [[2., 2.],
     [4., 4.]])
x = torch.tensor(
    [[1.], [2.]])
```

Size/shape:


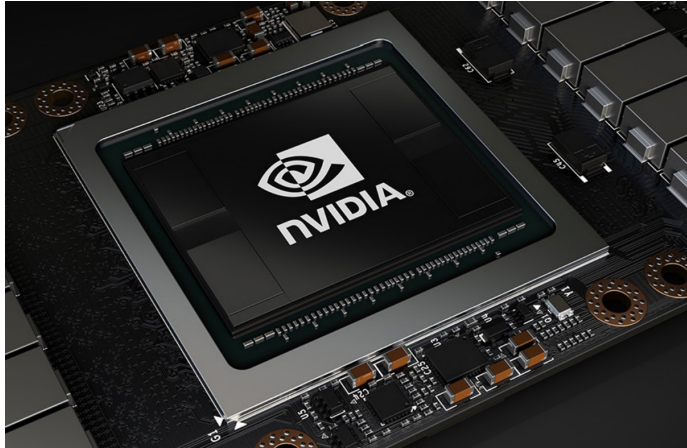
```
>>> a.size()
torch.Size([2, 2])
>>> a.shape
torch.Size([2, 2])
>>> a.dtype
torch.float32
```

Multiply:

```
>>> a @ x
tensor([[ 6.],
        [12.]])
>>> torch.matmul(a, x)
tensor([[ 6.],
        [12.]])
>>> einsum(
    'i j, j k -> i k',
    a, x)
tensor([[ 6.],
        [12.]])
```

Available Devices for PyTorch

```
device = "cuda" if torch.cuda.is_available() else "cpu"  
print(f"Using {device} device")
```

Laptop	GPU server
Using cpu device	Using cuda device
	

Tensor in GPU

```
a = torch.tensor([[2., 2.], [4., 4.]])  
a.device
```

Laptop	GPU server
<code>device(type='cpu')</code>	<code>device(type='cpu')</code>

```
a = a.to(device)
```

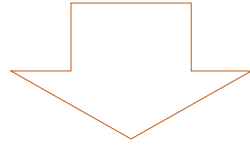
Laptop	GPU server
<code>device(type='cpu')</code>	<code>device(type='cuda', index=0)</code>

Tensor in GPU and Back to CPU

Laptop	GPU server
<pre>>>> a tensor([[2., 2.], [4., 4.]])</pre>	<pre>>>> a tensor([[2., 2.], [4., 4.]], device='cuda:0')</pre>

Laptop – Back to CPU (No Change)	GPU server – Back to CPU
<pre>>>> a = a.cpu() >>> a tensor([[2., 2.], [4., 4.]])</pre>	<pre>>>> a = a.cpu() >>> a tensor([[2., 2.], [4., 4.]])</pre>

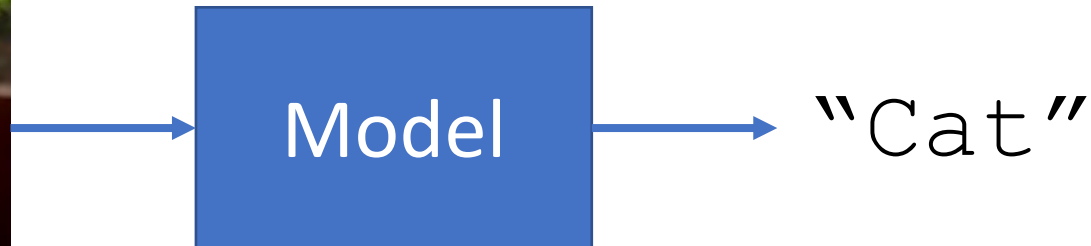
Numpy to PyTorch to Numpy



Data Structure	Device	Code
<code>np.ndarray</code>	CPU	<code>a = np.array([[1., 2.], [2., 4.]])</code>

Model Inference

Model Inference



Input Data → Trained Model → Output Prediction

Input

Can be any type of data

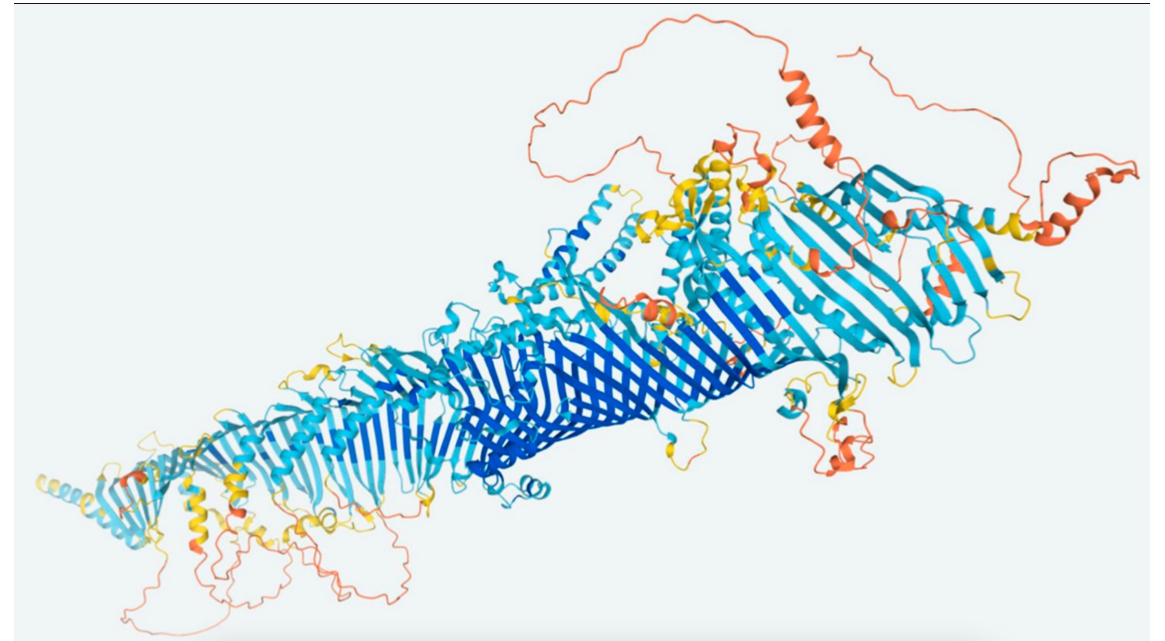
Vision: image, video

Waveforms: speech, music

3D: point cloud

Text: character, word, phoneme

Other forms: radar, multi-spectral,
protein structure, etc



Protein structure of a fruitfly
[Science.org 2021]

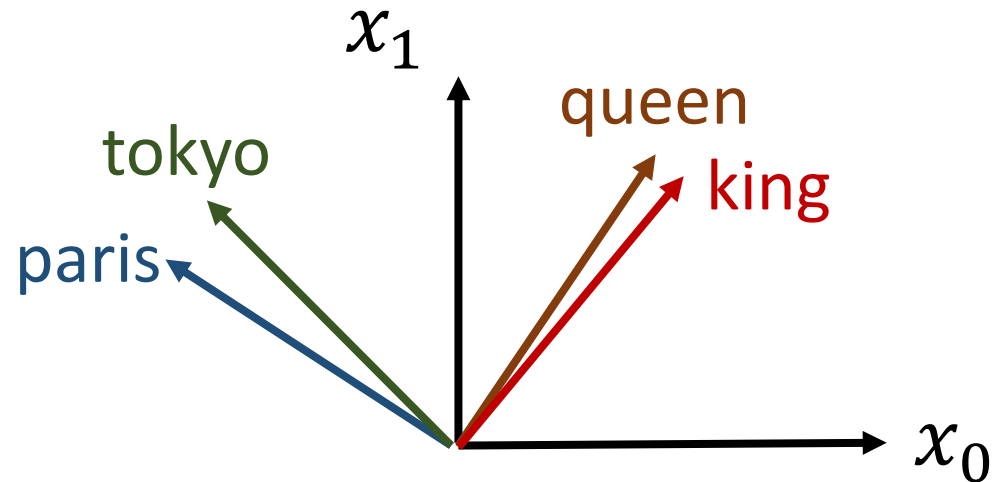
Loading Image Data

```
from PIL import Image  
img = Image.open("wonder_cat.jpg")  
  
# Visualize the data  
# in Jupyter  
display(img)
```



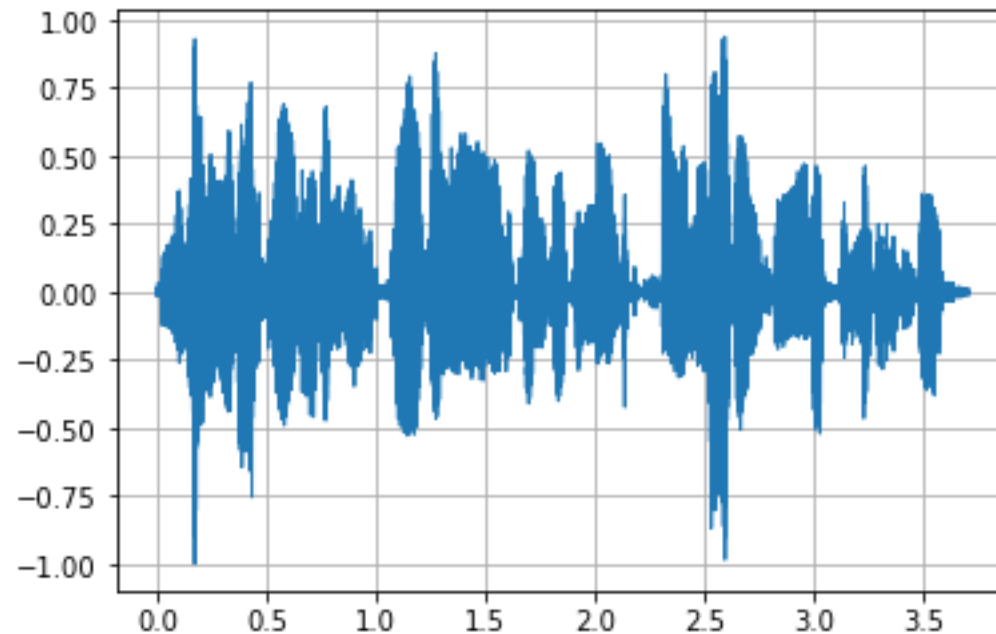
Loading Text Data

```
words = {"hello": 0, "world": 1}
embed_len = len(words)
embed_dim = 4
embed = torch.nn.Embedding(embed_len, embed_dim)
lookup = torch.tensor([words["hello"]], dtype=torch.long)
embed(lookup) # tensor([[ -0.3745,  0.1376, -0.3058,  1.0258]])
```



Loading Audio/Speech Data

```
import librosa  
wav, sample_rate = librosa.load("data/ljspeech.wav")  
plot_waveform(wav, sample_rate)  
Waveform
```



Specialized PyTorch Libraries

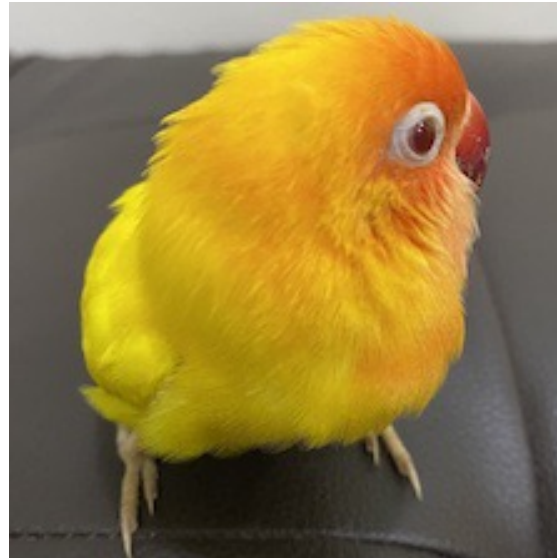
`torchvision` - package consists of popular datasets, model architectures, and common image transformations for computer vision.

`torchaudio` - library for audio and signal processing with PyTorch. It provides I/O, signal and data processing functions, datasets, model implementations and application components.

Other libraries – `torchtext`, `torchrec`, `torchmultimodal`, `torchrl`

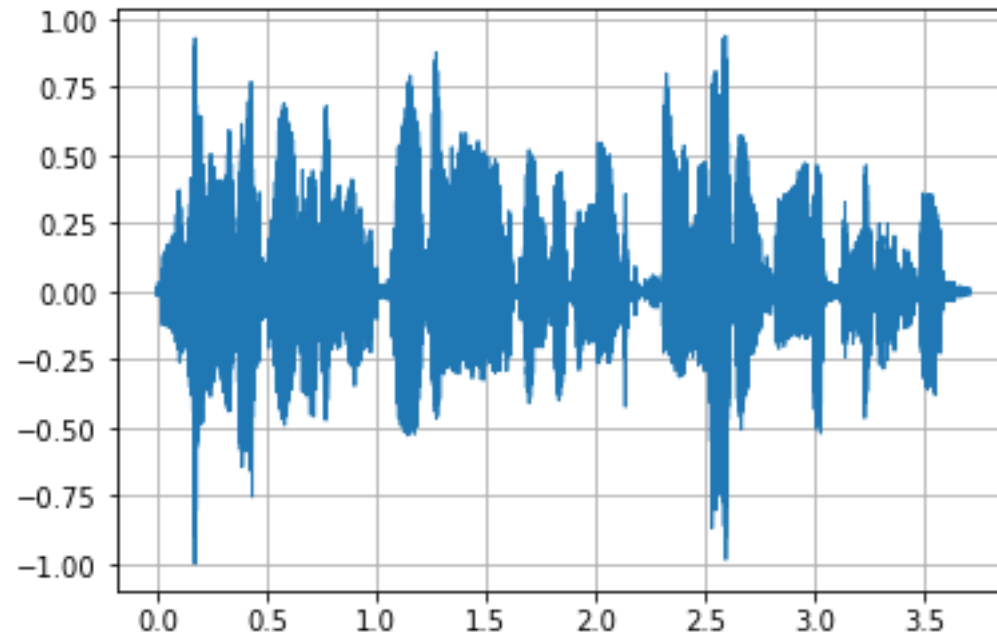
TorchVision

```
import torchvision  
img = torchvision.io.read_image("data/birdie2.jpg")  
img = torchvision.transforms.ToPILImage()(img)  
display(img)
```



TorchAudio

```
import torchaudio  
wav, sample_rate = torchaudio.load ("data/ljspeech.wav")  
plot_waveform(wav, sample_rate)  
Waveform
```



Loading Pre-trained Model from torchvision

```
resnet = torchvision.models.resnet18(pretrained=True)
```

Other pretrained models available:

AlexNet, SqueezeNet, VGG, EfficientNet, MobileNet, RegNet, ViT, ConvNeXt, etc.

See: <https://pytorch.org/vision/master/models.html>

Input Data Preparation for Model Ingestion

Simple transform:

```
from PIL import Image
import torchvision.transforms as transforms

img = Image.open("wonder_cat.jpg")
img = transforms.ToTensor()(img)
```



img

Input Data Preparation for Model Ingestion

Better:

```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],  
                                std=[0.229, 0.224, 0.225])  
  
transform = transforms.Compose([  
    transforms.Resize(256),  
    transforms.CenterCrop(224),  
    transforms.ToTensor(),  
    normalize,])  
  
# PIL image undergoes transforms.  
img = transform(img)
```



img

Output: Model Prediction

Model must be in evaluation model: `resnet.eval()`

Ensure that there is a batch dim. If none, add:

```
img = rearrange(img, 'c h w -> 1 c h w')
```

Do the inference in no gradient tracking context:

```
with torch.no_grad():  
    pred = resnet(img)
```

Finally, get the index of the maximum probability:

```
pred = torch.argmax(pred, dim=1)
```

What is `argmax()` of `pred` ?

`pred`

Index	Unnormalized Probabilities
0	1.7247
1	2.2064
...	
284	7.4005
285	11.4601
286	6.6287
...	
999	3.2967

`argmax()`



285

Human Readable Labels

For ImageNet1k, each index corresponds to a text label:

```
{0: 'tench, Tinca tinca',  
 1: 'goldfish, Carassius auratus',  
 2: 'great white shark, white shark',  
 3: 'tiger shark, Galeocerdo cuvieri',  
 ...  
 998: 'ear, spike, capitulum',  
 999: 'toilet tissue, toilet paper'}
```

Human Readable Label

For example, `pred` has a value of `285`. This value corresponds to:

...

283: 'Persian cat',

284: 'Siamese cat, Siamese',

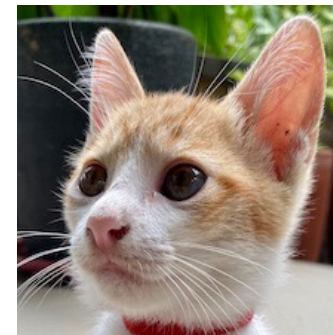
285: 'Egyptian cat', 

286: 'cougar, puma, catamount, mountain lion, painter, panther, Felis concolor',

287: 'lynx, catamount',

288: 'leopard, Panthera pardus',

...



TIMM: pyTorch IMage Models

<https://rwightman.github.io/pytorch-image-models/>

Just announced : Timm is now part of huggingface

<https://github.com/huggingface/pytorch-image-models>

Why timm?

From the doc:

`timm` is a deep-learning library created by Ross Wightman and is a collection of SOTA computer vision models, layers, utilities, optimizers, schedulers, data-loaders, augmentations and also training/validating scripts with ability to reproduce ImageNet training results.

In short:

`timm` extends PyTorch by implementing many deep learning SOTA models, optimization, regularization and other useful algorithms.

Install and Use

Install:

```
pip install timm
```

Use it like torchvision:

```
if use_timm:
```

```
    resnet = timm.create_model('resnet18', pretrained=True)
```

```
else:
```

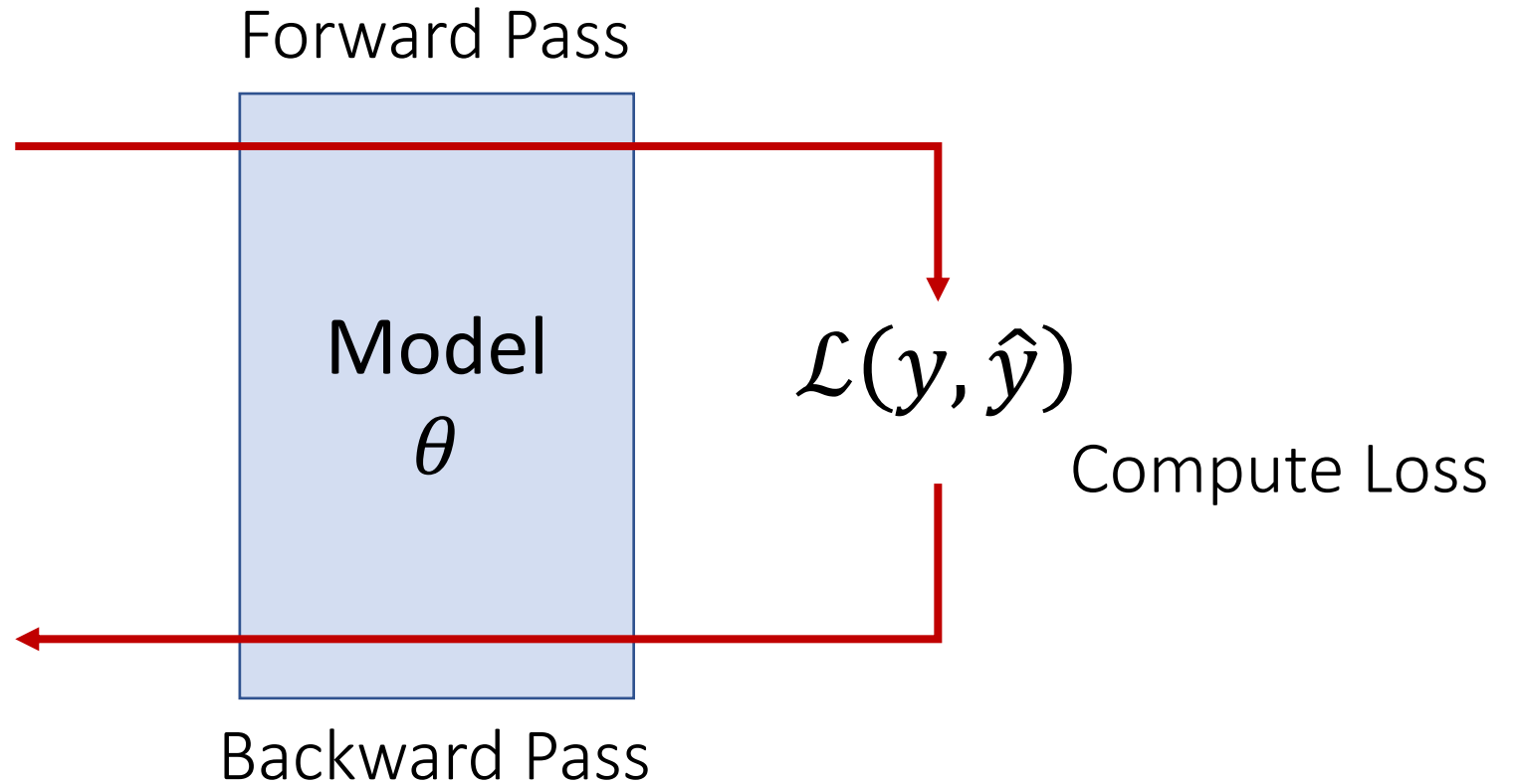
```
    resnet = torchvision.models.resnet18(pretrained=True)
```


Autograd

Forward and Backward Passes



Input Data



Forward Propagation

- In forward prop, the model makes its best guess about the correct output.
- It runs the input data through each of its layers (functions) to make this prediction.

```
prediction = model(data)
```

Backward Propagation

- In backprop, the model adjusts its parameters proportionate to the error in its guess.
- It does this by traversing backwards from the output, collecting the derivatives of the error with respect to the parameters of the functions (gradients), and optimizing the parameters using gradient descent.

Compute Loss and gradients

- Compute loss

```
loss = (prediction - labels).sum()
```

- Backpropagate the error

```
loss.backward()
```

- Autograd then calculates and stores the gradients for each model parameter.
`parameter.grad` attribute.

Instantiate an optimizer to compute gradients

- Stochastic Gradient Descent (SGD) applied on model parameters with learning rate of 0.01 and momentum of 0.9

```
optim = torch.optim.SGD(model.parameters(),  
                        lr=1e-2,  
                        momentum=0.9)
```

Zero the optimizer gradients before re-computing gradients

- Compute loss

```
loss = (prediction - labels).sum()
```

- **Zero the parameter gradients**

```
optim.zero_grad()
```

- Backpropagate the error

```
loss.backward()
```

- Autograd then calculates and stores the gradients for each model parameter.grad attribute.

Initiate gradient descent

optim.step()

- This is the end of 1 training step

Model

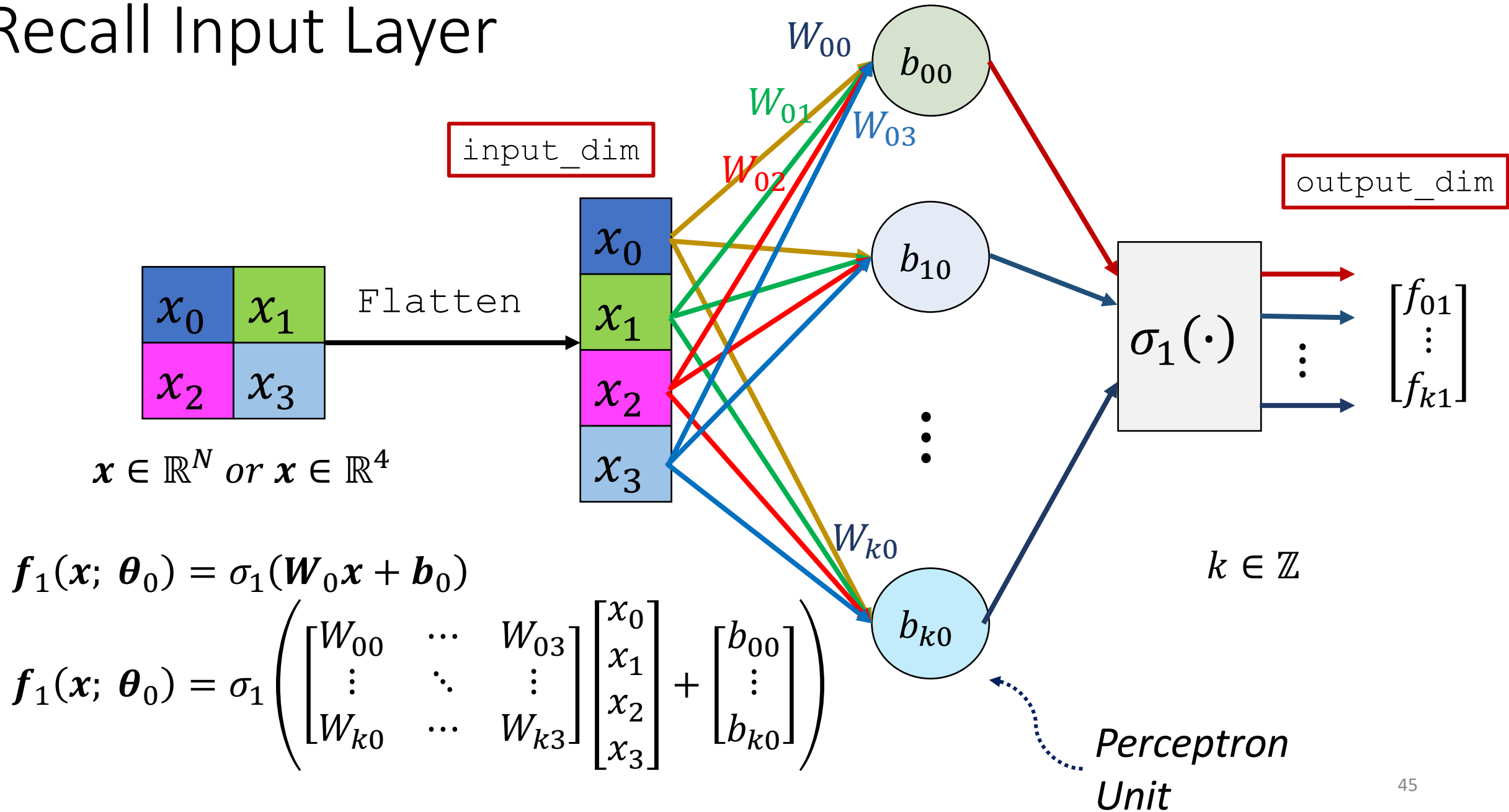
3-layer MLP for Imagenet1k Classification

```
class MLP(nn.Module):  
    def __init__(self, input_size, hidden_size, output_size):  
        super(MLP, self).__init__()  
        self.fc1 = nn.Linear(input_size, hidden_size)  
        self.fc2 = nn.Linear(hidden_size, hidden_size)  
        self.fc3 = nn.Linear(hidden_size, output_size)  
        self.relu = nn.ReLU()  
  
    def forward(self, x):  
        x = x.view(x.size(0), -1) # Flatten the input tensor  
        x = self.relu(self.fc1(x))  
        x = self.relu(self.fc2(x))  
        x = self.fc3(x)  
  
        return x
```

```
nn.Linear(input_dim, output_dim)
```

- `input_dim` = input dimensions
- `output_dim` = output dimensions

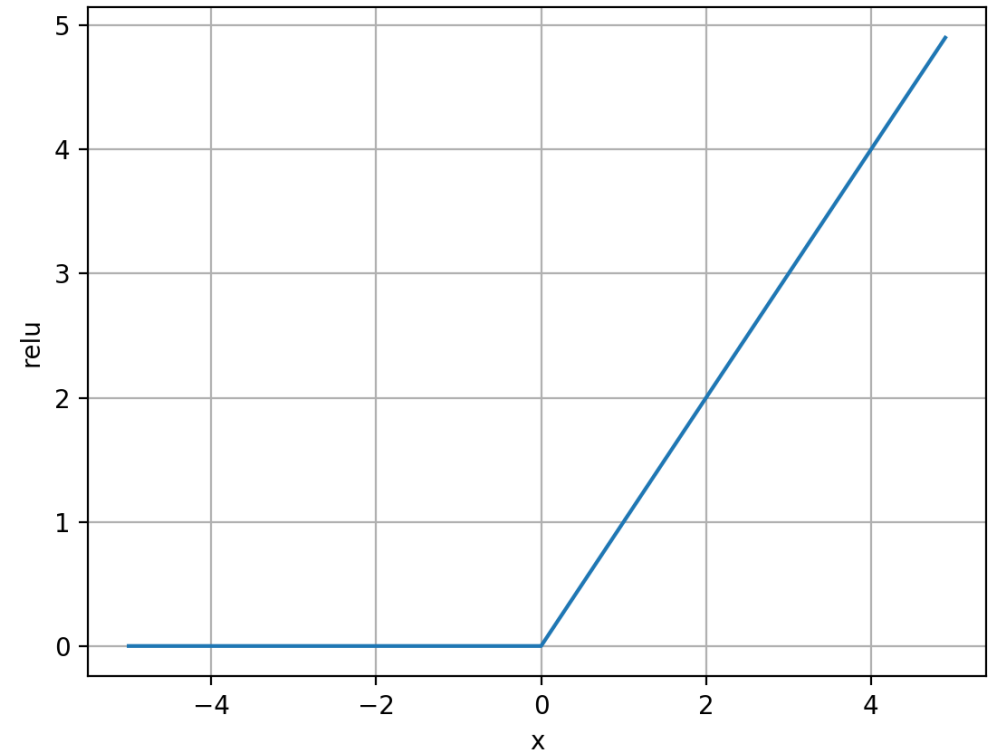
Recall Input Layer



```
nn.relu()
```

Rectified Linear Unit:

$$\sigma(x) = \text{ReLU}(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$



Where is the softmax?

Softmax:

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{k=0}^C e^{x_k}}$$

Recall Cross Entropy

For discrete distribution, Categorical Cross-Entropy is:

$$CE = H(P, Q) = -\sum_i P(x_i) \log Q(x_i)$$

If CE is used, no need for softmax

```
loss = nn.CrossEntropyLoss()
```


Model Training

Model Training

- Data Pre-processing Transform
- Dataset and Dataloader
- Model Training
 - Identify the loss function and optimizer
 - Model in train mode
 - Train for N epochs

Data Transform

```
# Define the transforms to be applied to the images
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

Dataset and Dataloader

```
# Load the ImageNet dataset
imagenet_dataset = datasets.ImageNet(root="/raid/imagenet/dataset/",
                                     split="train",
                                     transform=transform)

# Create a DataLoader to load the images in batches
dataloader = DataLoader(imagenet_dataset, batch_size=32, shuffle=True)
```

Device, Loss and Optimizer

```
import torch.optim as optim
import torch

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(mlp.parameters(), lr=0.001)
```

Model Training

```
# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(mlp.parameters(), lr=0.001)

mlp.train()
# Train the model for 10 epochs
for epoch in range(10):
    running_loss = 0.0
    for i, data in enumerate(dataloader, 0):
        # Get the inputs and labels from the dataloader
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward + backward + optimize
        outputs = mlp(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

Model Inference

Model Inference

- Data Pre-processing Transform
- Dataset and Dataloader
- Model Evaluation
 - Model in eval mode
 - Compute performance on evaluation split

Data Pre-processing Transform

```
# Define the transforms to be applied to the images
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

Dataset and Dataloader

```
# create a test dataloader
imagenet_dataset = datasets.ImageNet(root="/raid/imagenet/dataset/",
                                     split="val",
                                     transform=transform)

dataloader = DataLoader(imagenet_dataset, batch_size=32, shuffle=False)
```

Performance Evaluation

```
# evaluate the model on the validation set
correct = 0
total = 0
with torch.no_grad():
    for data in dataloader:
        # Get the inputs and labels from the dataloader
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        # Forward pass
        outputs = mlp(inputs)
        _, predicted = torch.max(outputs.data, 1)

        # Compute accuracy
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"Accuracy: {100 * correct / total:.2f}%")
```

End