An Introduction to PyTorch

Presented by N’yoma Diamond

Setup

Install Python: https://www.python.org/downloads/

Download files (https://github.com/nyoma-diamond/PyTorch-Demo)

```
> git clone https://github.com/nyoma-diamond/PyTorch-Demo.git
```

Install libraries

```
> pip install jupyterlab numpy matplotlib scikit-learn
> pip install torch torchvision
```

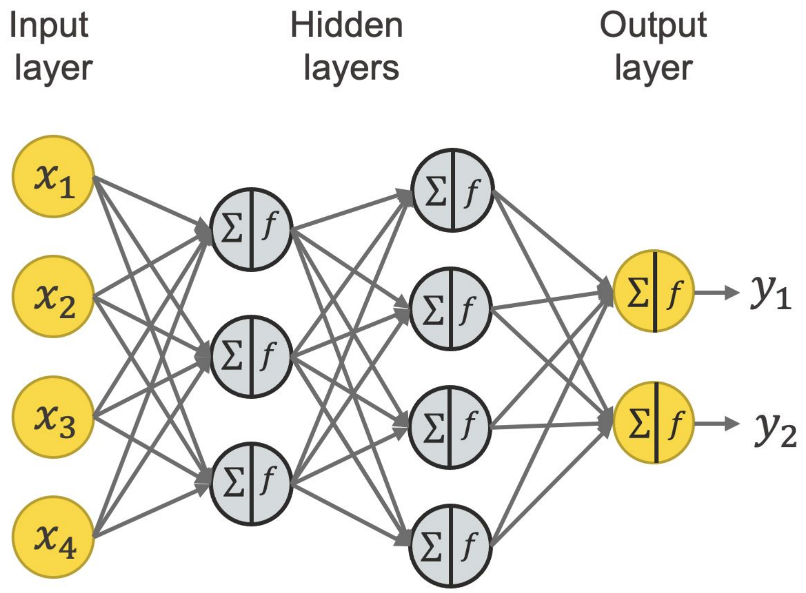
See https://pytorch.org/get-started/locally/ if you have CUDA/ROCm

Neural Networks

A Crash Course

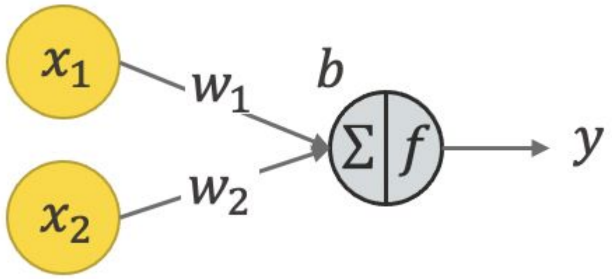
Neural Networks: A Crash Course

Networks



Neural Networks: A Crash Course

Neurons



$$y = f\left(b + \sum_{i=1}^{n_x} w_i \cdot x_i\right)$$

$x_i =$ input value i

$w_i =$ weight of x_i

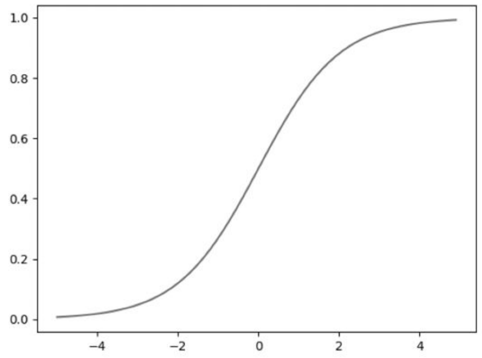
$b =$ bias

$y =$ neuron output

Neural Networks: A Crash Course

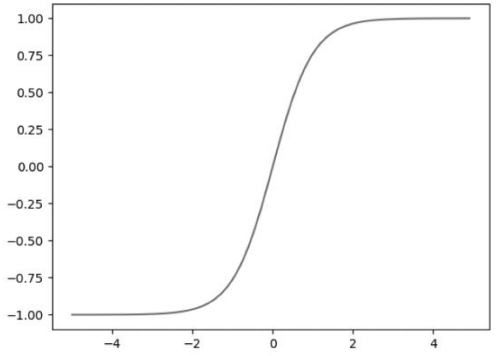
Activation Functions

Sigmoid



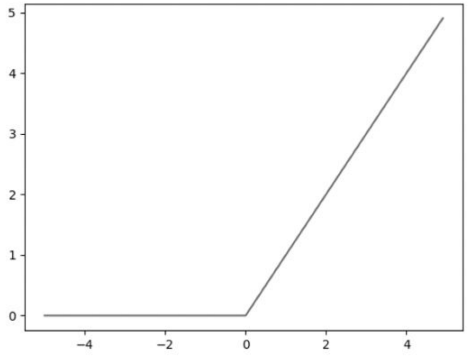
$$f(a) = \frac{1}{1 + e^{-ha}}$$

Tanh



$$f(a) = \frac{e^{2ha} - 1}{e^{2ha} + 1}$$

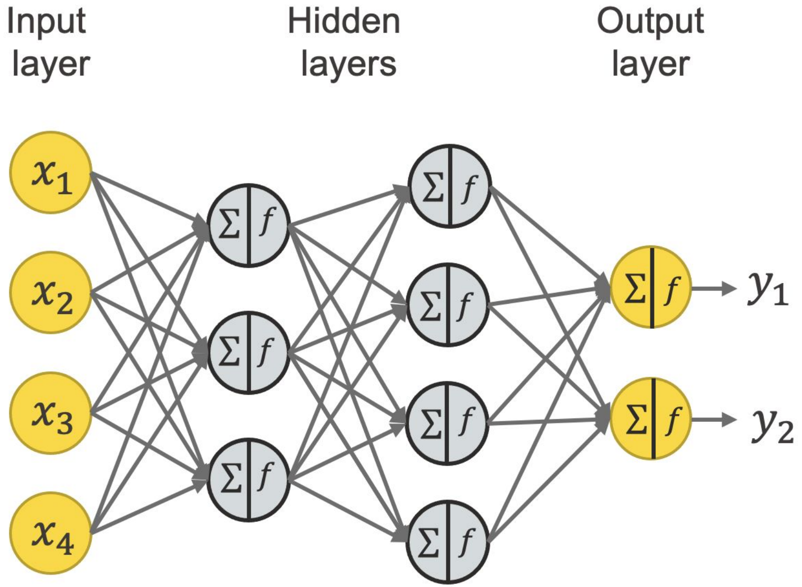
Rectified Linear Unit (ReLU)

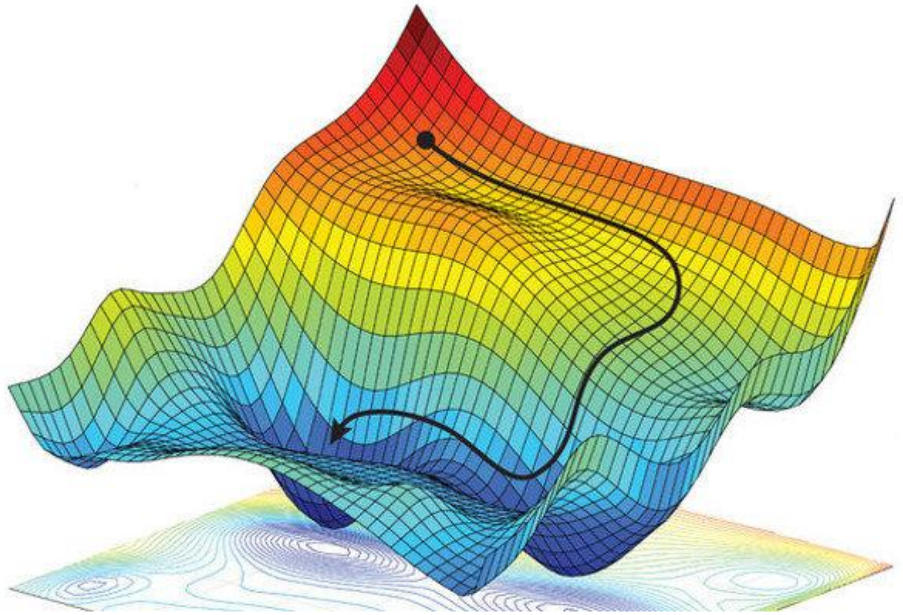


$$f(a) = max\{0, ha\}$$

Neural Networks: A Crash Course

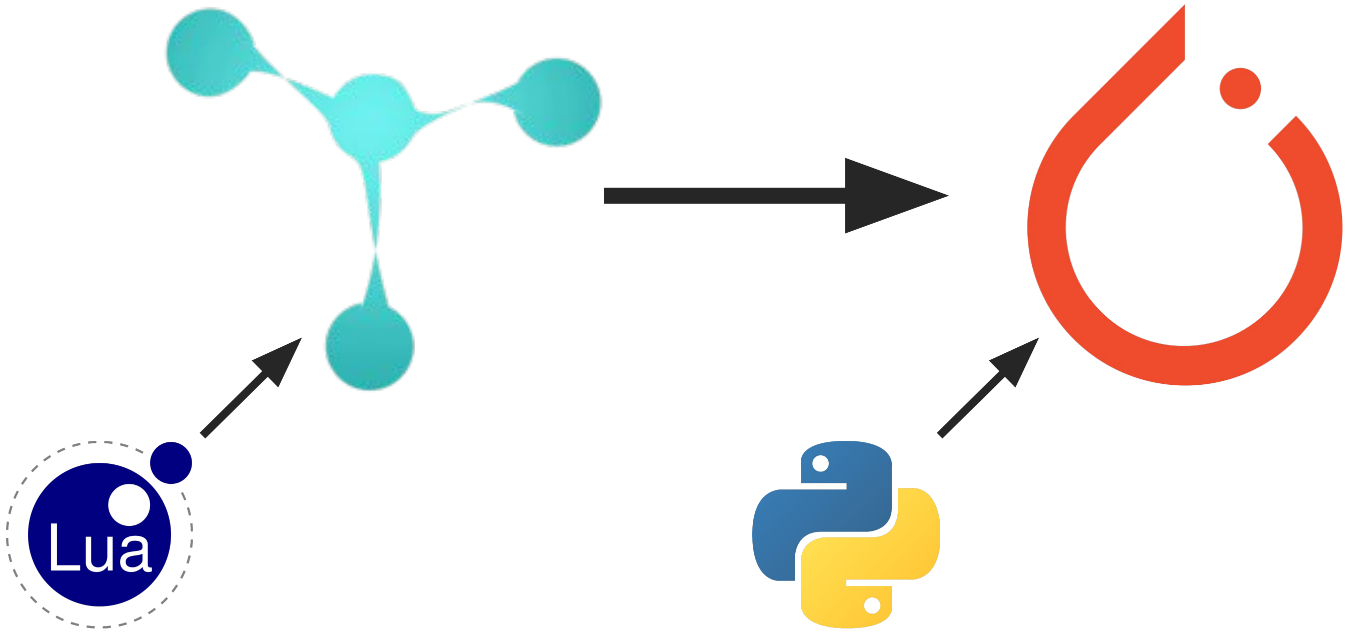
Optimization (Training)





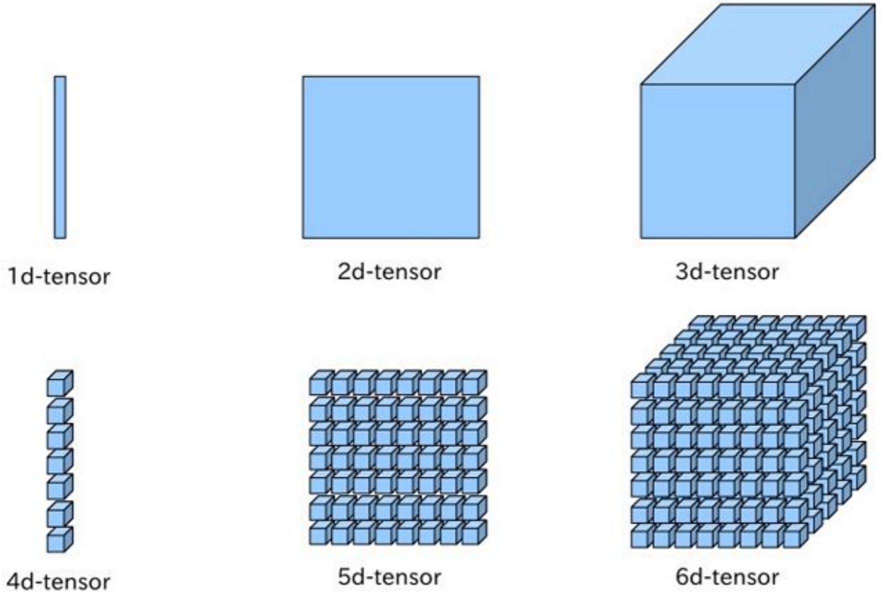
Optimization (Training)

PyTorch: What is it?



PyTorch: What is it?

Tensors



Let’s Write Some Code!

Jupyter Notebooks

Combines Python, Markdown, LaTeX. Cell-based

Not to be confused with Jupyter Notebook IDE

Working in Jupyter Lab:

In terminal:

```
> jupyter-lab
```

Working in PyCharm:

Supported natively :)

DataLoaders

Code Cell 4:

```
# DataLoaders
train_loader = DataLoader(mnist_train, batch_size=10000, shuffle=True)
test_loader = DataLoader(mnist_test, batch_size=10000, shuffle=True)
```

Example Input: Raw Data

Code Cell 5:

```
for images, labels in test_loader:
    sample_image = images[0]
    sample_label = labels[0]
    break

print('type:', type(sample_image))
print('shape:', sample_image.shape)
print('raw data:', sample_image)
```

Example Input: Visualization

Code Cell 6:

```
plt.imshow(sample_image, cmap='gray')

print('sample label:', sample_label)
plt.show()
```

Example Input: Reshaping Data

Code Cell 7:

```
print('original shape:', sample_image.shape)

print('reshaped using `view`:', sample_image.view(sample_image.size(0)*sample_image.size(1)).shape)

print('reshaped using `reshape`:', sample_image.reshape(sample_image.size(0)*sample_image.size(1)).shape)

print('reshaped using `flatten`:', sample_image.flatten().shape)
```

Model Design: Hidden Layers

Code Cell 8:

```
def __init__(self):
    super(Model, self).__init__()

    # Hidden layers
    self.hidden1 = nn.Linear(28*28, 100)
    self.hidden2 = nn.Linear(100, 100)
    self.hidden3 = nn.Linear(100, 100)

    ...
```

Model Design: Output Layer

Code Cell 8:

```
def __init__(self):
    super(Model, self).__init__()

    ...

    # Output layer
    self.out = nn.Linear(100, 10)

    ...
```

Model Design: Output Layer

Code Cell 8:

```
def __init__(self):
    super(Model, self).__init__()

    ...

    # Activation functions
    self.relu = nn.ReLU()               # Hidden layer activation
    self.softmax = nn.Softmax(dim=1) # Output layer activation
```

Model Design: Forward Function

Code Cell 8:

```
# Model operation
def forward(self, x):
    super(Model, self).__init__()

    h1 = self.relu(self.hidden1(x))    # hidden layer 1, ReLU activation
    h2 = self.relu(self.hidden2(h1))   # hidden layer 2, ReLU activation
    h3 = self.relu(self.hidden3(h2))   # hidden layer 3, ReLU activation
    return self.softmax(self.out(h3)) # output layer, Softmax activation
```

Model Initialization

Code Cell 9:

```
model = Model()
if cuda_available:
    model.cuda()

print(model)
```

Training Protocol Initialization

Code Cell 10:

```
criterion = nn.CrossEntropyLoss()                          # Loss function
optim = torch.optim.Adam(model.parameters(), lr=1e-3) # Optimizer
```

Training Function: Training vs. Testing

Code Cell 11:

```
def run_epoch(train):
    # Set model mode and desired dataloader
    if train:
        model.train()
        loader = train_loader
    else:
        model.eval()
        loader = test_loader
```

Training Function: Using DataLoaders

Code Cell 11:

```
def run_epoch(train):

    ...

    for x, y in loader:
        if cuda_available:
            x = x.cuda()
            y = y.cuda()
```

Training Function: Prediction

Code Cell 11:

```
# fit batches
for x, y in loader:

    ...

    x = x.flatten(start_dim=1, end_dim=2)
    predictions = model(x)
    loss = criterion(predictions, y)
```

Training Function: Backpropogation

Code Cell 11:

```
# fit batches
for x, y in loader:

    ...

    if train:
        optim.zero_grad() # Reset gradients
        loss.backward()   # Calculate new gradients
        optim.step()      # Update weights and biases
```

Training Variables

Code Cell 12:

```
epochs = 50 # Epochs to train for
```

TRAINING LOOP

Code Cell 13:

```
for e in range(epochs):
    print(f'Training epoch {e+1}/{epochs}...')
    run_epoch(True) # Train model

print('Done training!')
```

Performance Analysis

Code Cell 14:

```
# All truths and predictions on the test set
test_truths = []
test_preds = []

# Iterate over the test set
for x, y in test_loader:
    if cuda_available:
        x = x.cuda()
        y = y.cuda()
```

Performance Analysis

Code Cell 14:

```
# Iterate over the test set
for x, y in test_loader:

    ...

    x = x.flatten(start_dim=1, end_dim=2)
    predictions = model(x)

    # Store truths and predictions from this batch
    test_truths.extend(y.cpu().numpy())
    test_preds.extend(predictions.argmax(dim=1).cpu().numpy())
```

Performance Analysis

Code Cell 14:

```
...

# Generate and display confusion matrix
ConfusionMatrixDisplay.from_predictions(test_truths, test_preds, cmap='Reds')
plt.show()
```

References

https://www.knime.com/blog/a-friendly-introduction-to-deep-neural-networks

https://www.researchgate.net/publication/325142728_Spatial_Uncertainty_Sampling_for_End-to-End_Control

https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

https://medium.com/@anoorasfatima/10-most-common-maths-operation-with-pytorchs-tensor-70a491d8cafd

https://pytorch.org/