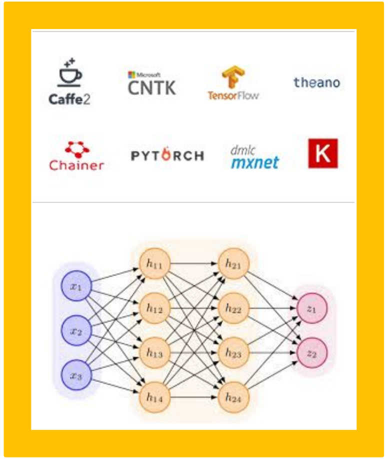


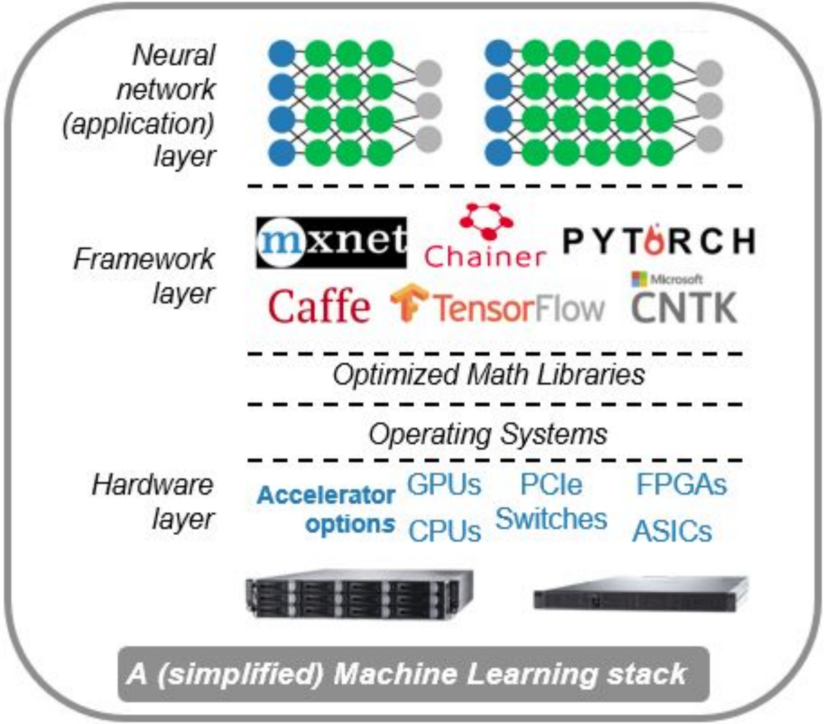
Deep Learning In Practice

Girish Varma

Deep Learning Stack

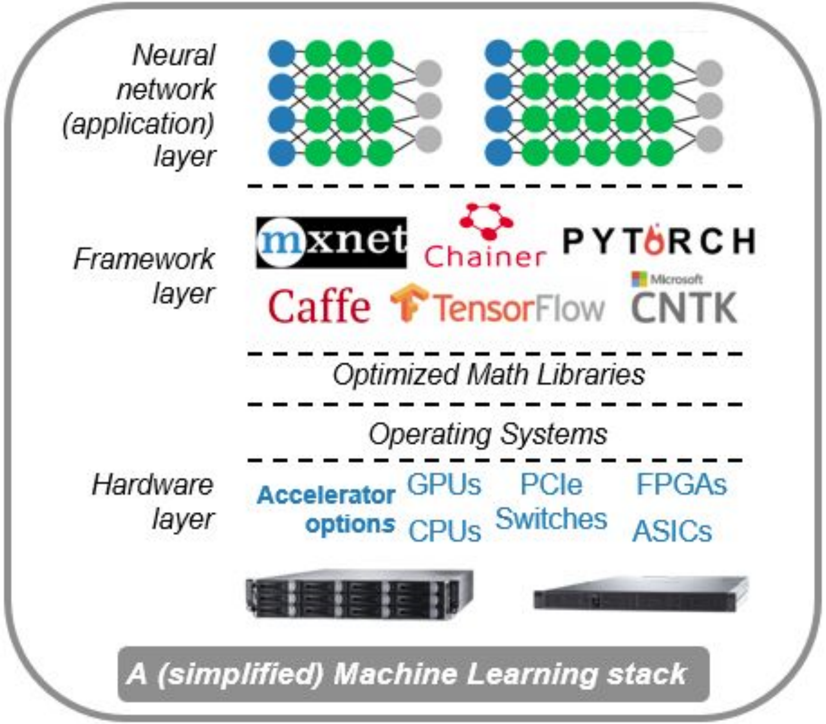
Deep learning programs can involve:

- matrix multiplications,
- computing derivatives
- loading data from network/hard disk
- plotting and visualization of loss functions



Deep Learning Stack

- For fast execution, programs needs to be parallelized and efficient assembly code needs to be generated.
- Needs to run in various platforms like cpu, gpu, mobiles, clusters.
- There is a stack of libraries that takes care of these, so that we can focus on designing neural networks.



Deep Learning Libraries

	Language	Created By	
Torch	Lua	NYU & IDIAP	2002
Theano	Python	Toronto & Montreal	2009
Caffe	C++	UC Berkeley	2012
Tensorflow	Python	Google	2015
Pytorch	Python	Facebook	2017

- Not comprehensive. Almost every company have their own implementation.
- Deployment happens in C/CPP, Python only used during training.

Tensors

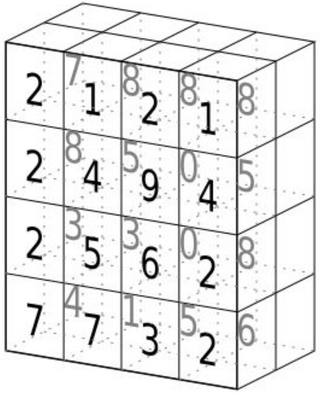
Basic objects of a Deep Learning Library

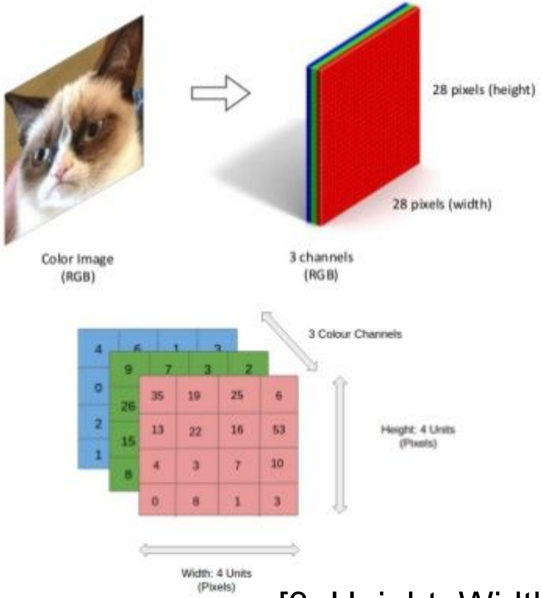
All data, intermediate outputs, learnable parameters are represented by a **tensor**.

Tensors have a **size**.

't'
'e'
'n'
's'
'o'
'r'

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3
2	3	8	4
6	2	6	4





Size [6] [6, 4] [2, 4, 4] [3, Height, Width]

Structure of a Deep Learning Program

1. Loading Data
2. Defining the Model
3. Defining Training Procedure
4. Looping over Data
5. Computing testing accuracy

Step 1: Loading Data

Machine learning datasets can be very large (few GBs to TBs).

Only a small batch (minibatch) is loaded at a time for processing.

Example : MNIST Classification

Input : x is a [28,28] shaped matrix, giving pixel values of the image

Output : y is a [10] shaped vector, giving the probabilities of being 0 to 9.

Dataset : Consist of (x,y) pairs, x is the input and y is called the label.

Divided into train, test and validation.

If the dataset gives y as a digit, convert it to probability vector by one hot encoding.

Note: y can sometimes be a number between 0-9 or a vector of dimension 10.

Code for MNIST

AIML

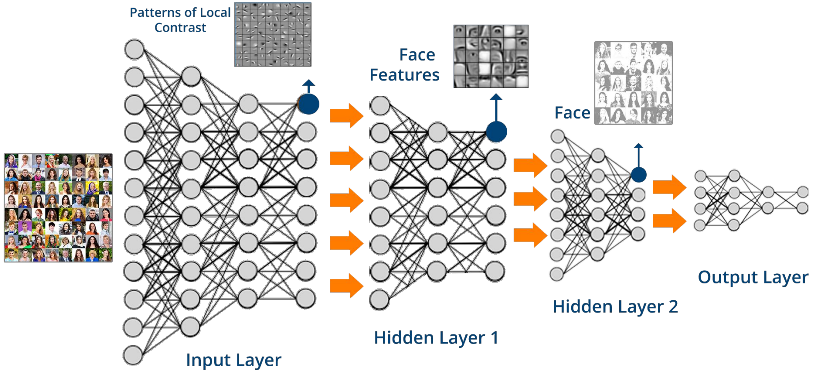
```
mnist_train = datasets.MNIST(
      root='./mnist/',
      train=True,
      transform=transforms.ToTensor(),
      download=True
)

train_loader = DataLoader(dataset=mnist_train,
                          batch_size=100,
                          shuffle=True)

for mini_batch in train_loader:
      images, labels = mini_batch
      for j in range(batch_size):
            print images[j].size(), labels[j]
```

Prints: torch.Size([1, 28, 28]) 3

1. Download and extract MNIST in to local machine.
2. Shuffle the dataset such that the labels are very well mixed and create a loader that loads 100 images at a time.
3. Two loops
   a. Outer loops loads 100 images each
   b. Inner loop iterates over the 100 images and their labels.

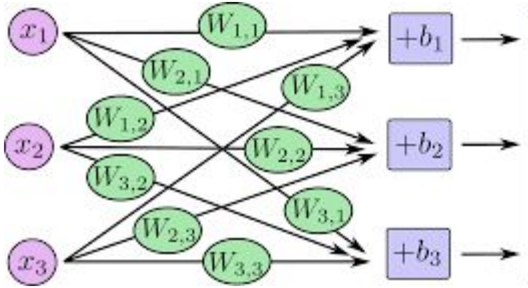
Step 2: Define the Model

- We need to specify the architecture of the deep learning model.
- Typically consists of a sequence of layers.
- Layers could be Linear(Fully Connected), ReLU Activation, CNN, RNN etc.
- Each layer can have hyperparameters.
- Depth of the model = number of layers (can be 30-100). Results in better models.



Model Architecture : Linear Model

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc = nn.Linear(28*28, 10)

    def forward(self, x):
        x = x.view(-1,28*28)
        x = self.fc(x)
        return x

network = Net()
logits = network(image)
predictions = softmax(logits)
```

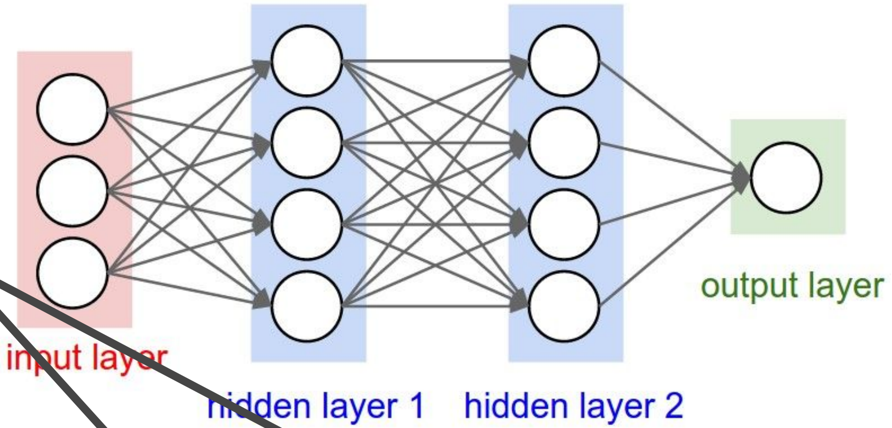


Convert input from a tensor with size [28, 28] to [784]

Obtain prediction probabilities for 10 classes, by applying softmax function.

Model Architecture : MLP Model

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 120)
        self.fc2 = nn.Linear(120, 80)
        self.fc3 = nn.Linear(80, 10)

    def forward(self, x):
        x = x.view(-1,28*28)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```



120
Hidden units

80
Hidden units

Activation in between linear layers

Model Architecture : CNN Model

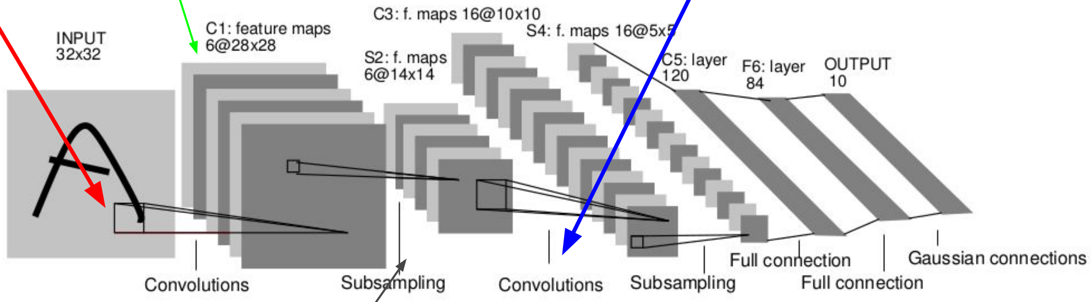
AIML

```python
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1   = nn.Linear(16*5*5, 120)
        self.fc2   = nn.Linear(120, 84)
        self.fc3   = nn.Linear(84, 10)

    def forward(self, x):
        out = F.relu(self.conv1(x))
        out = F.max_pool2d(out, 2)
        out = F.relu(self.conv2(out))
        out = F.max_pool2d(out, 2)
        out = out.view(out.size(0), -1)
        out = F.relu(self.fc1(out))
        out = F.relu(self.fc2(out))
        out = self.fc3(out)
        return out
```

6 channels

5x5 windows

Conv2: 6-> 16 channels, 5x5 window size



Maxpool reduces dimension. Has no parameters

Step 3: Specify Loss & Gradient Update Algo

AIML

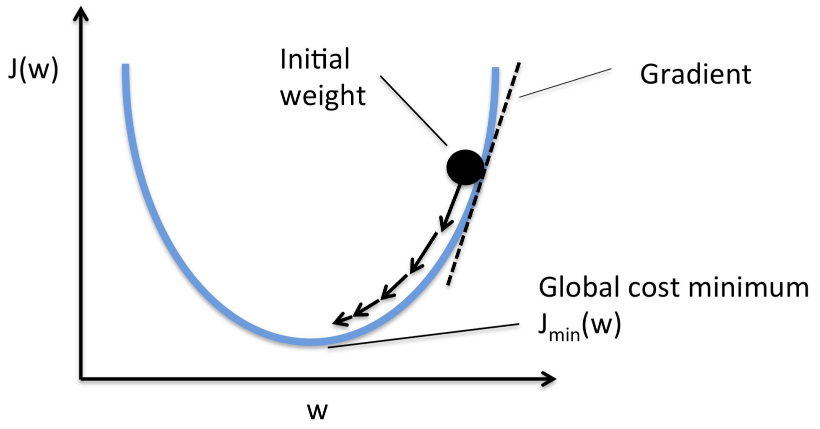
The difference between predictions and correct value.
Different loss function for different tasks.

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

Updating the weight using the gradients can be done using learning rate.

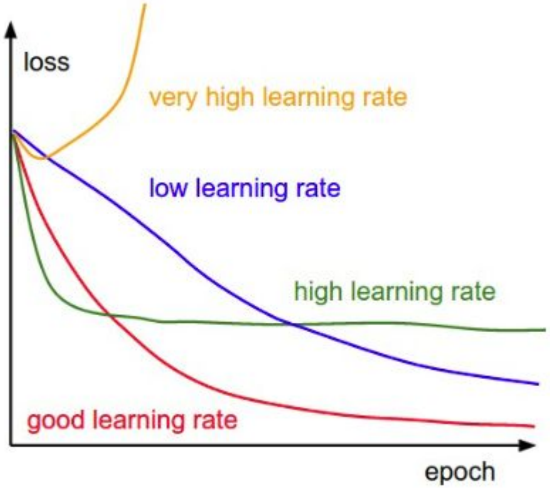
There are multiple gradient update algorithms.

```
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```



Step 4: Training Loop

- Repeat
  a. Take a small random subset of the dataset that will fit in memory (minibatch)
  b. Forward Pass: pass the subset through the model and obtain predictions
  c. Compute the mean loss function for the subset
  d. Backward Pass: compute the gradients of the parameters, last layer to the first,update the gradients using **learning rate**
  e. Plot loss



Step 5: Computing accuracy on Testing data

Similar to Step 4, in looping over the test data.

However we do not do the backward pass.

We just compute the accuracy of the model.

Demo

GPUs and Deep Learning

In deep learning programs, the same operation needs to be done for different data.

For example:

Every image in a batch has to be processed by the DNN.

Every neuron operation is the same, if we consider the weights also as inputs.

GPUs and Deep Learning

GPUs have 1000s of small processors that run **same instructions** on **different data.**