CS446/ECE 449: Machine Learning

Raymond A. Yeh

yeh17@illinois.edu

University of Illinois at Urbana Champaign, 2020

L10: Deep Learning System & Pytorch

Recap:

- Deep nets & back propagation

Goals of this lecture

- Deep / Machine learning software pipeline
- Pytorch Overview
- Pytorch Example

Reading material

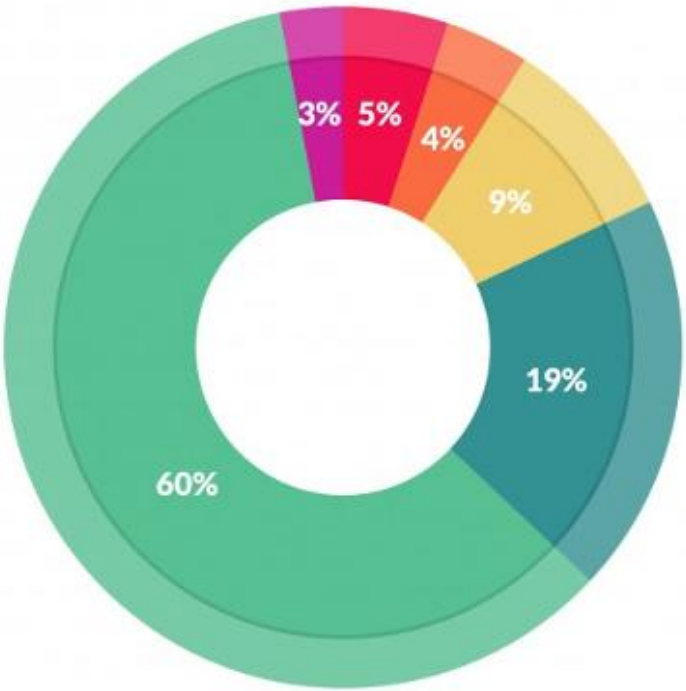
- Pytorch Tutorial (`https://pytorch.org/tutorials/`)

Machine Learning / Deep Learning Systems:

- Data $\mathcal{D}$
- Feature representation x
- Output representation y
- Model $F(\mathbf{w}, x, y)$
- Loss function L
- Optimization Algorithm

Data & Representation

- How to represent the data?
- Data processing (cleaning data, a large portion of the work.).
- Feature extraction refers to the action of converting from data to $\phi(x)$
- Cleaning data is hard!



What data scientists spend the most time doing

- *Building training sets: 3%*
- *Cleaning and organizing data: 60%*
- *Collecting data sets; 19%*
- *Mining data for patterns: 9%*
- *Refining algorithms: 4%*
- *Other: 5%*

Credit: Forbes

Toy Example Let's say we want to predict whether a student will get an A in ECE544NA. We have data from previous semester,

id	Hours studied	HW grade	Favorite animal	Final grade
1	10	90	cat	99%
2	20	100	elephant	100%
3	0	50	zebra	85%
⋮	⋮	⋮	⋮	⋮

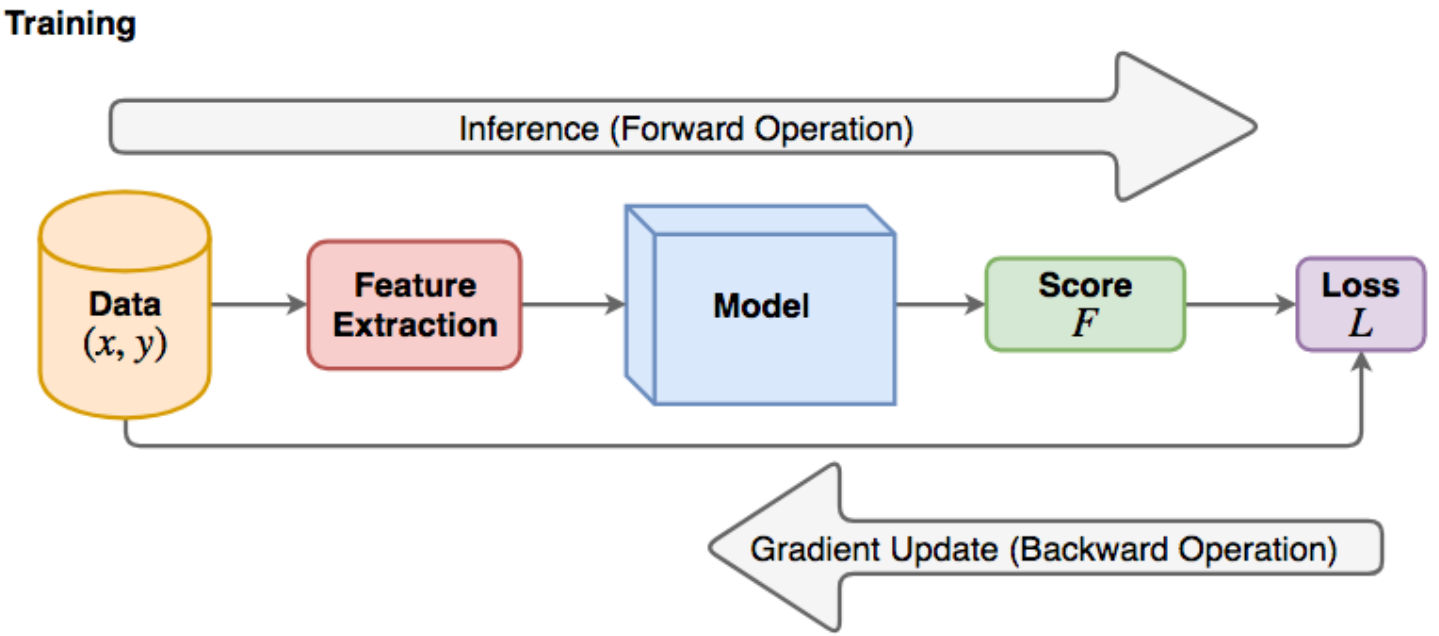
1. How to represent output y? (Regression $\in \mathbb{R}$? Classification $\in \{-1, 1\}$?)
2. How to represent input x? (Recall, $x \in \mathbb{R}^d$.)
   Do we need to use all the data?
3. Linear regression or logistic regresion?

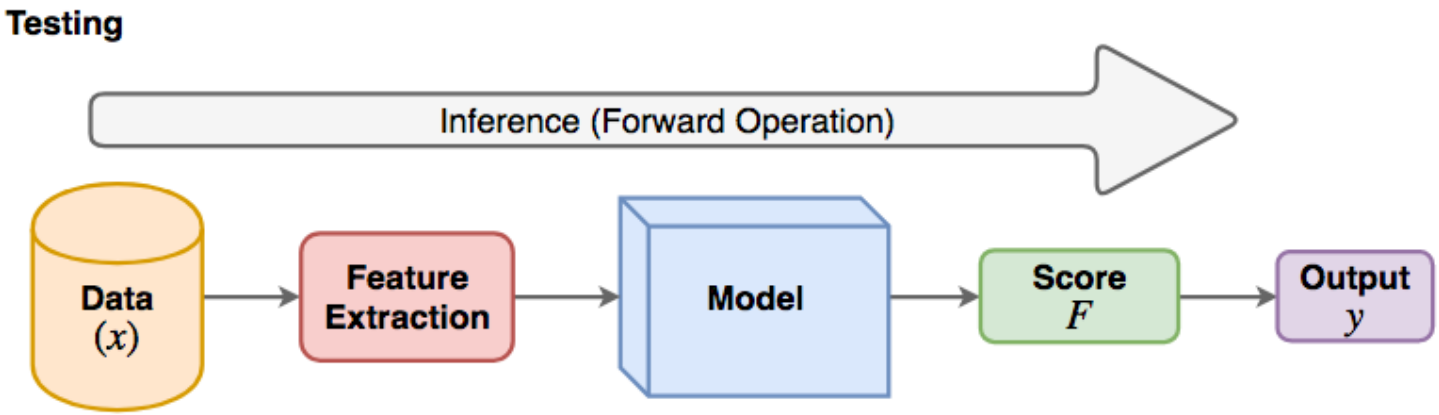
Model & Loss

- Model initialization: initialize all parameters in the model
- Forward operation: generates the score F given an input x
- Prediction operation: takes a score and maps from F to $\hat{y}$
- Loss: takes F and y and outputs a $\mathbb{R}$
- Backward operation (Optimize): updates the model parameters to minimize loss

"All models are wrong, but some are useful" — *George Box*

Train and Test Stage

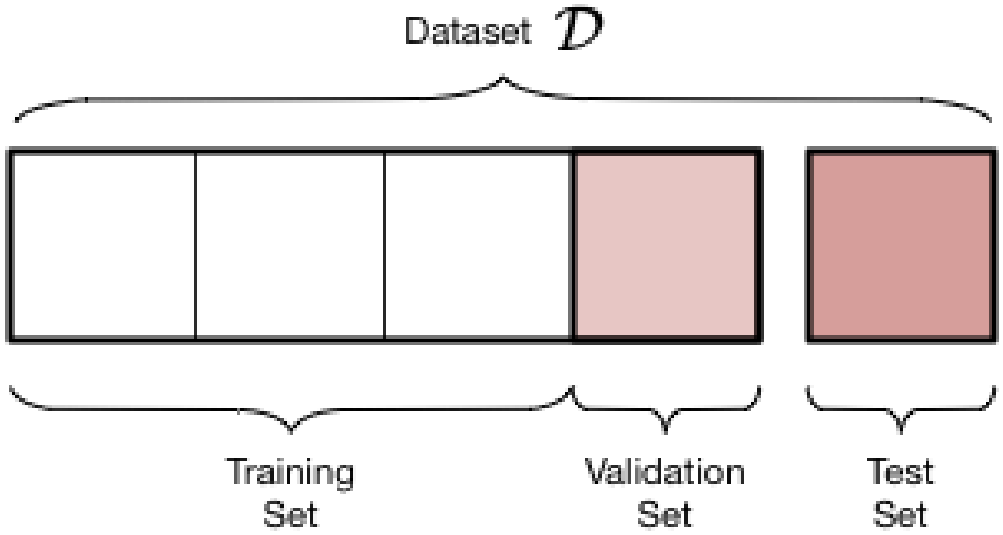




Machine learning project structure

```
data
   processed       <- The data sets for modeling.
   raw             <- Raw data
checkpoints  <- Save and logs experiments
models       <- Model implementations
dataset      <- Dataset implementations
experiment   <- Train and test scripts
main.py
```

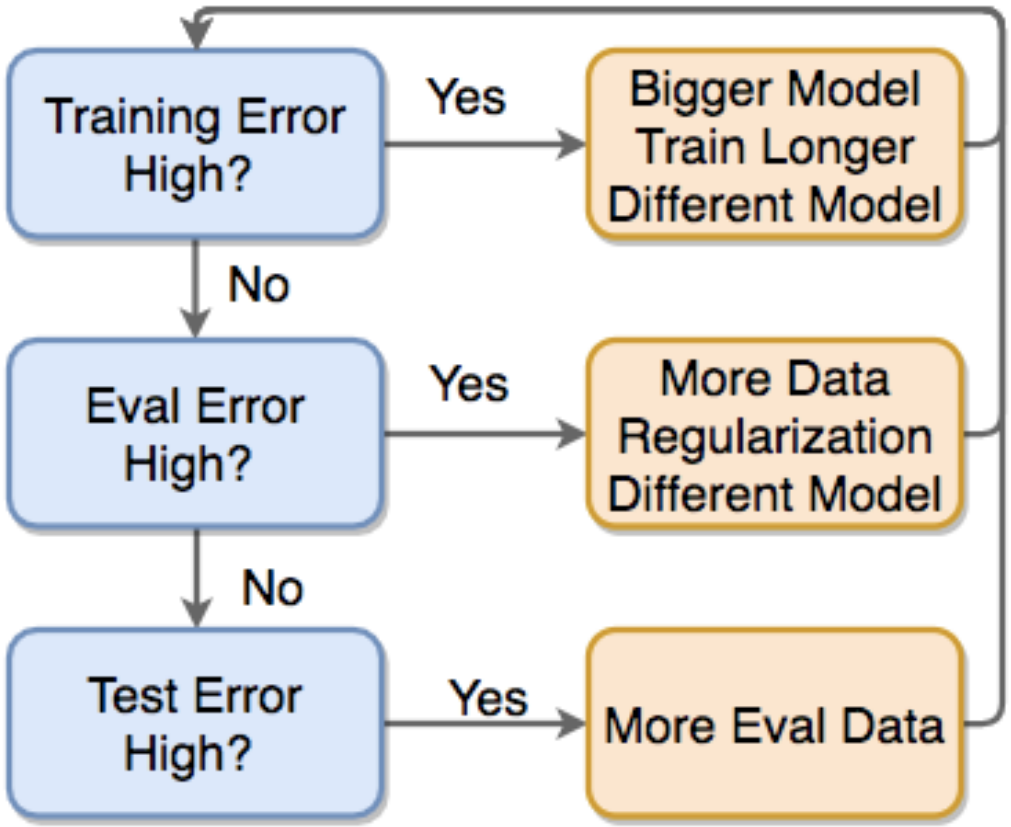
Three phases of experiments.



- Training: Update model parameters based on training data
- Evaluation: Measure performance on the validation data to choose the hyperparameters
- Testing: Measure performance on the testing data

Credit: am207.github.io

A standard machine learning workflow.



Credit: Andrew Ng

Why is machine learning hard?

- Data is noisy
- Many possibilities for data representation
- Hard to debug (many things could go wrong!)

Credit: xkcd.com

Pytorch

- 

- Facebook's deep learning framework.

TensorFlow: machine learning library for research and production

- 

- Google's deep learning framework.

Others

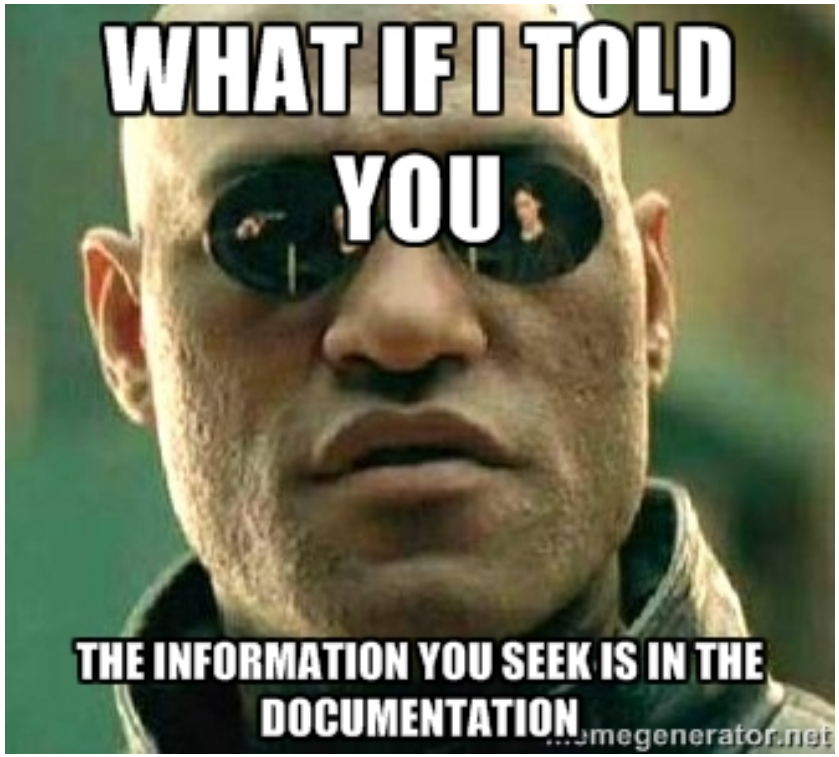
- MXNET: A flexible and efficient library for deep learning
- JAX: Autograd and XLA

Pytorch

- Installation (`https://pytorch.org/get-started/locally/`)
- Using Google Colab (CPU and GPUs for Free!).

A few usful Pytorch Packges

- `torch`
- `torch.nn`
- `torch.optim`
- `torchvision`



Data with Pytorch

- Tensor $\approx$ multidimensional array.
- First dimension is, conventionally, used for batching.

```
1 import torch
2 num_example = 3
3 feature_dim = 2
4 x = torch.randn(num_example, feature_dim)
5 print(x)
```

```
tensor([[-1.6828,  1.4117],
        [ 0.0263, -0.8230],
        [ 1.2949, -1.2613]])
```

Mathematical operations with Pytorch

- Matrix multiplication (`torch.mm`, `torch.matmul`)

```
1 import torch
2 x = torch.randn(3, 3)
3 out = torch.mm(x, x.t())
```

- Pseudo-inverse (`torch.pinverse`)
- Least squares solver (`torch.lstsq`)
- ...

Model with Pytorch

- Represent a model via a computation graph (acyclic)
- Standard building block ("modules") for deep-nets are in `torch.nn`
- Standard loss functions are also in `torch.nn`

```
1  class LinearClassifier(nn.Module):
2    # Adapted from torch.nn.Linear
3    def __init__(self, in_features=2, out_features=1):
4      super(LinearClassifier, self).__init__()
5      self.w = nn.Parameter(torch.Tensor(out_features,
6                                         in_features))
7      self.b = nn.Parameter(torch.Tensor(out_features))
8
9    def forward(self, input):
10     x = F.linear(input, self.w, self.b)
11     return torch.sigmoid(x)
12
13   def predict(self, input):
14     return F.linear(input, self.w, self.b) > 0
15   ...
```

Optimizer

- Standard gradient-based optimizers are in `torch.optim`

```
# Training
model = LinearClassifier(in_features=feature_dim)
epochs = 100
optimizer = torch.optim.SGD(model.parameters())
loss_epoch = []
acc_epoch = []
for epoch in range(epochs):
  score = model(x)
  optimizer.zero_grad()
  loss = model.loss(score, y.float().unsqueeze(-1))
  loss.backward()
  optimizer.step()
```

Computer Vision (torchvision)

- `torchvision.datasets`
- `torchvision.transforms`
- `torchvision.models`

Credit: Pytorch

Demo on Google Colab [Link]