Machine Learning Methods for Atmosphere, Ocean, and Climate Science

Lecture 2: Implementing ANNs in PyTorch

Mathematical modeling of Climate, Ocean, and Atmosphere processes
International Centre for Theoretical Sciences, TIFR, Bengaluru, India

Aman Gupta

Stanford University

DATAWAVE

Lecture 1

- Parametric estimation
- Introduction to deep neural networks
- The training algorithm

Lecture 2

- The PyTorch library
- Implementing Artificial Neural Nets in PyTorch

PyTorch

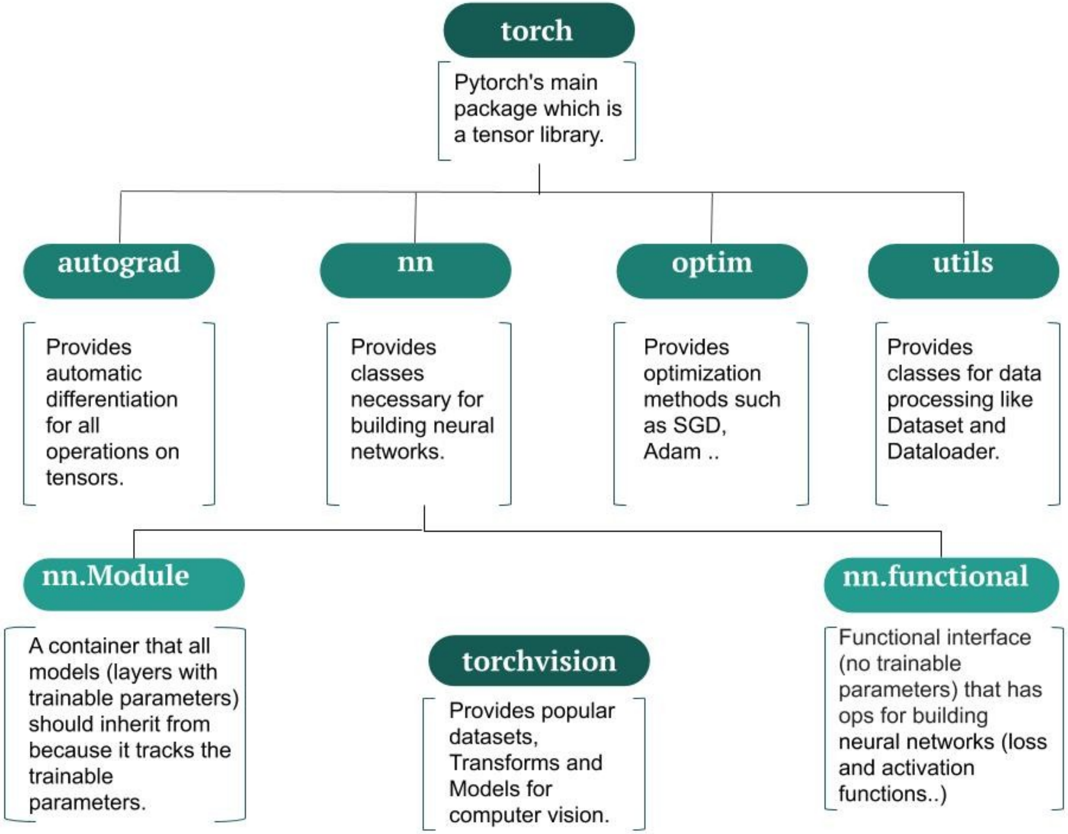
Lecture 3

- Implementing Convolutional Neural Networks in PyTorch

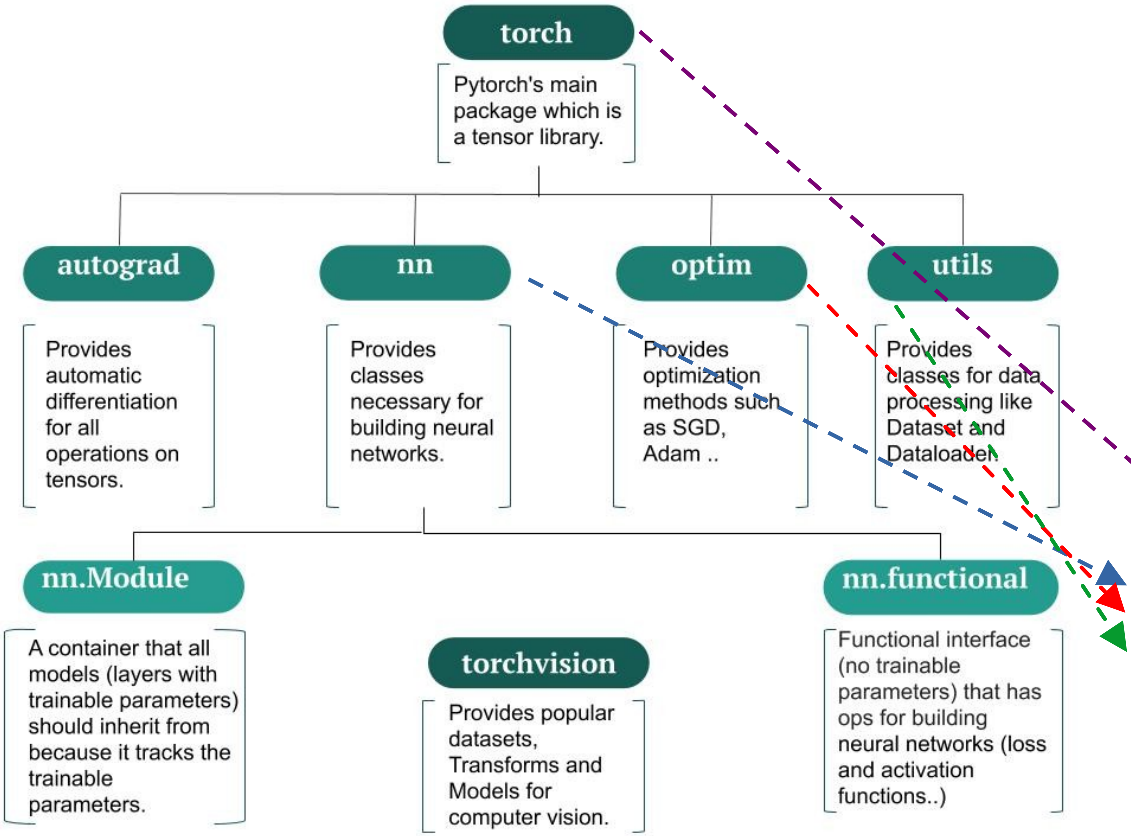
Lecture 4

- Applications of ML in climate science

PyTorch has various modules to aid coding



PyTorch has various modules to aid coding



PyTorch Build	Stable (2.0.1)	Preview (Nightly)		
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python	C++ / Java		
Compute Platform	CUDA 11.7	CUDA 11.8	ROCm 5.4.2	CPU
Run this Command:	conda install pytorch torchvision torchaudio pytorch-cuda=11.7 -c pytorch -c nvidia			

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, random_split
from torch.utils.data import Dataset as TensorDataset
```

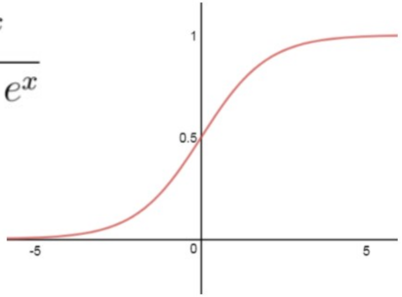
https://pytorch.org/docs/stable/nn.html

Choice of Activation Function is Problem-Dependent

Sigmoid function: Classification

torch.nn.Sigmoid()

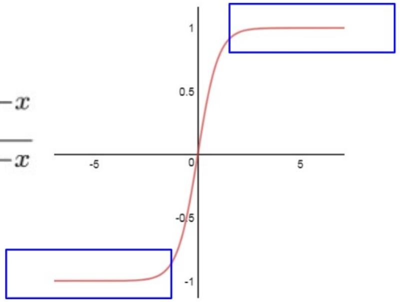
$$sigmoid(x) = \frac{e^x}{1 + e^x}$$



Tanh function: Classification

torch.nn.Tanh()

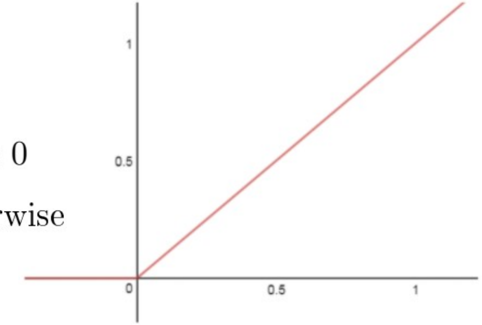
$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



Rectified Linear Unit (ReLU)

torch.nn.ReLU()

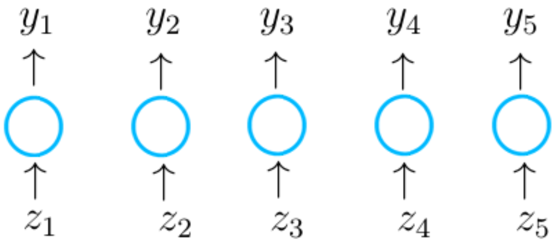
$$R(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



Softmax: Probability, LLMs

torch.nn.Softmax()

$$y_i = \frac{e^{z_i}}{\sum_{i=0}^{m} e^{z_i}}$$



Choice of Loss Function is Problem-Dependent

L1 Loss:
torch.nn.L1Loss()
Outliers

$$\text{loss}(x, y) = |x - y|$$

Mean Squared Error Loss:
torch.nn.MSELoss()
Regression

$$\text{loss}(x, y) = (x - y)^2$$

Cross Entropy Loss (Softmax loss):
torch.nn.CrossEntropyLoss()
multi-class classification

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top$$

$$l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})} \cdot 1$$

KL Divergence Loss:
torch.nn.KLDivLoss()
VAEs, classification, PDFs

$$L(y_{\text{pred}},\ y_{\text{true}}) = y_{\text{true}} \cdot (\log y_{\text{true}} - \log y_{\text{pred}})$$

Choice of Optimizers

Stochastic Gradient Descent (1847):

torch.optim.SGD()

$$w_{t+1} = w_t - \alpha g_t$$

....

$$v_{t+1} = \beta v_t + g_t$$

$$w_{t+1} = w_t - \alpha v_{t+1}$$

Adagrad (2011):

torch.optim.Adagrad()

1st to have adaptive LRs

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{diag(G_t)+\epsilon}} \cdot g_t$$

RMSProp (2012):

torch.optim.RMSProp()

$$v_{t+1} = \beta v_t + (1-\beta) g_t^2$$

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_{t+1}+\epsilon}} \cdot g_t$$

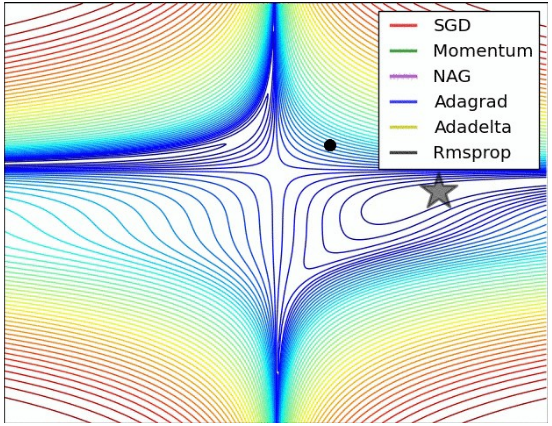
ADAM (2014):

torch.optim.Adam()

combines Adagrad, RMSProp, and momentum

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_{t+1}+\epsilon}} \cdot g_t$$

Choice of Optimizers



Stochastic Gradient Descent (1847):

torch.optim.SGD()

$$w_{t+1} = w_t - \alpha g_t$$

....

$$v_{t+1} = \beta v_t + g_t$$

$$w_{t+1} = w_t - \alpha v_{t+1}$$

Adagrad (2011):

torch.optim.Adagrad()

1st to have adaptive LRs

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{diag(G_t) + \epsilon}} \cdot g_t$$

RMSProp (2012):

torch.optim.RMSProp()

$$v_{t+1} = \beta v_t + (1-\beta) g_t^2$$

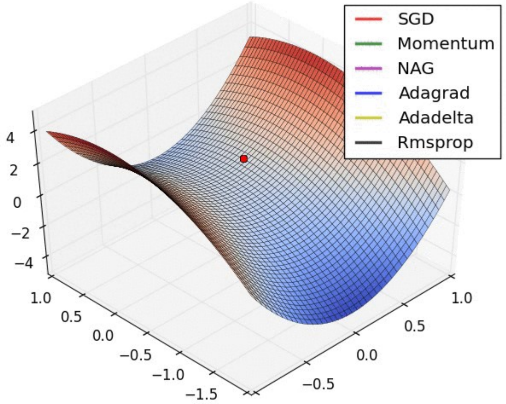
$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_{t+1}} + \epsilon} \cdot g_t$$

ADAM (2014):

torch.optim.Adam()

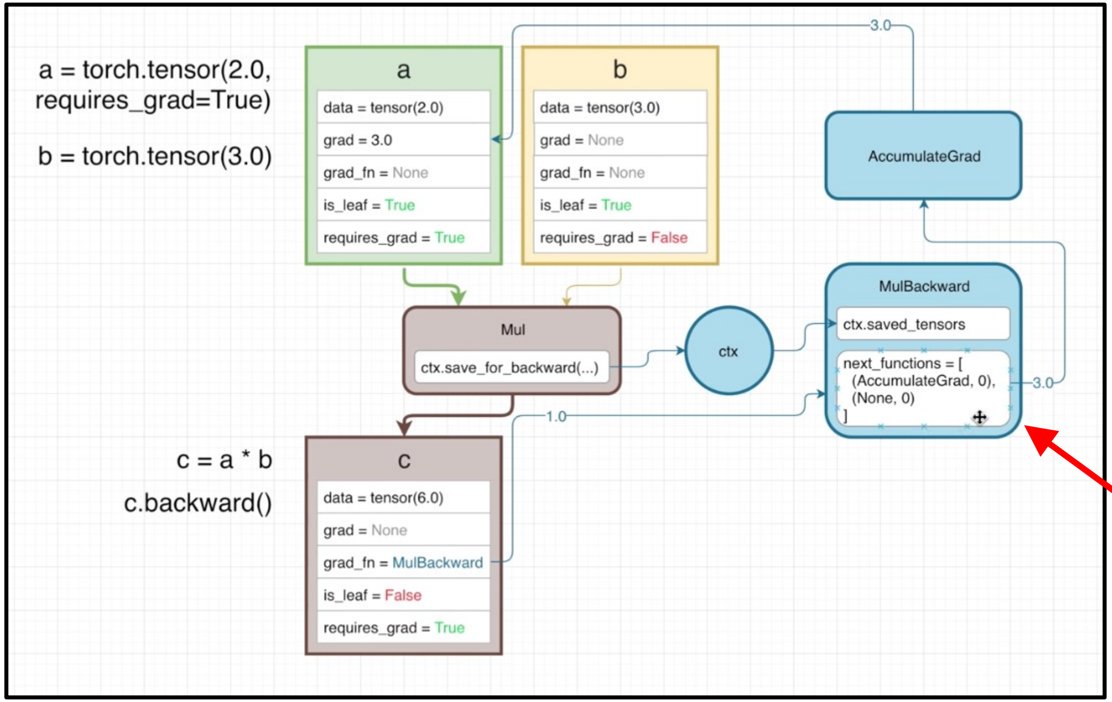
combines Adagrad, RMSProp, and momentum

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_{t+1}} + \epsilon} \cdot g_t$$



(Figures: www.datasciencecentral.com)

PyTorch Autograd



PyTorch's Autograd module maintains a graph of connections between different variables

This helps compute gradients of the loss function very efficiently, speeding up the optimization process

```
loss.backward()
optimizer.step()
```

Why PyTorch? Why not TensorFlow?

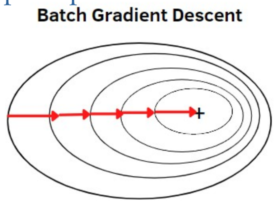
- Released in 2016, **PyTorch is easier to learn for researchers compared to Tensorflow.** Ex: ChatGPT-3, DALL-E was written in PyTorch.

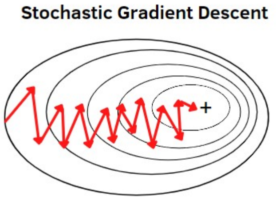
- Since newer, performs better than TensorFlow on most benchmarks.
  TensorFlow still preferred for ML code deployment in large systems.

- Problem with model translation.
  Does not have TensorBoard :-(

Using Minibatch Gradient Descent for Better Training

full dataset
per update

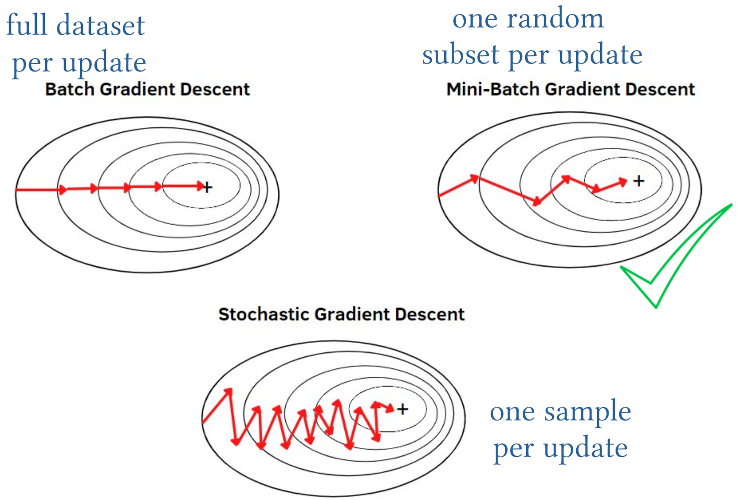




one sample
per update

(Figure: analyticsvidhya.com)

Using Minibatch Gradient Descent for Better Training

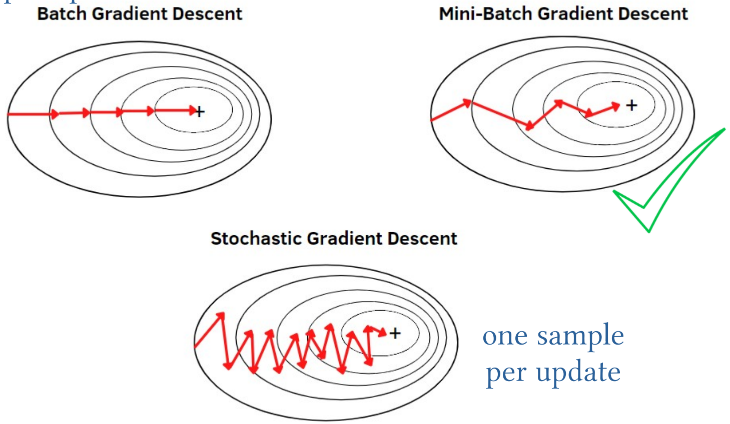


(Figure: analyticsvidhya.com)

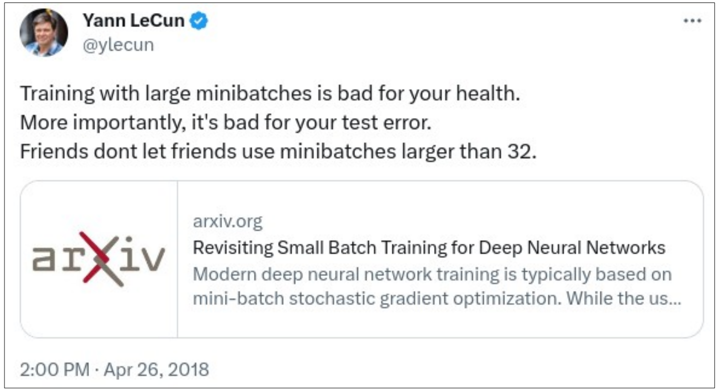
Using Minibatch Gradient Descent for Better Training

full dataset per update

one random subset per update



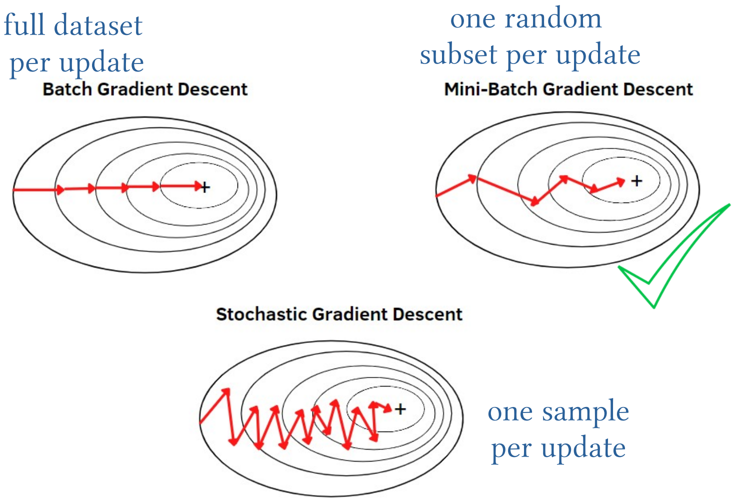
one sample per update

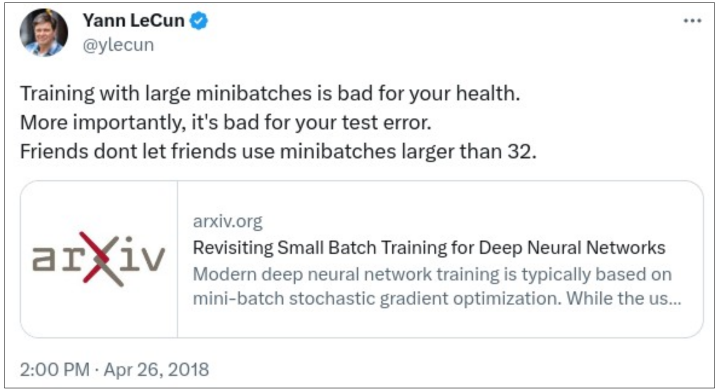


Learning rate	Batch size	Max word accuracy (%)	Training epochs
0.1	1	96.49	21
0.1	10	96.13	41
0.1	100	95.39	43
0.1	1000	84.13 +	4747 +
0.01	1	96.49	27
0.01	10	96.49	27
0.01	100	95.76	46
0.01	1000	95.20	1612
0.01	20,000	23.25 +	4865 +
0.001	1	96.49	402
0.001	100	96.68	468
0.001	1000	96.13	405
0.001	20,000	90.77	1966
0.0001	1	96.68	4589
0.0001	100	96.49	5340
0.0001	1000	96.49	5520
0.0001	20,000	96.31	8343

(Figure: analyticsvidhya.com
Table: Wilson and Martinez (2003))

Using Minibatch Gradient Descent for Better Training

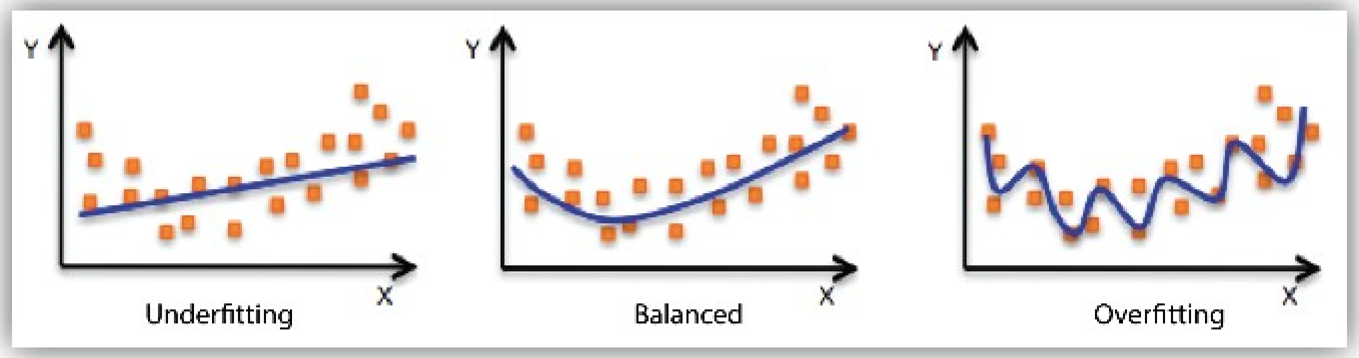


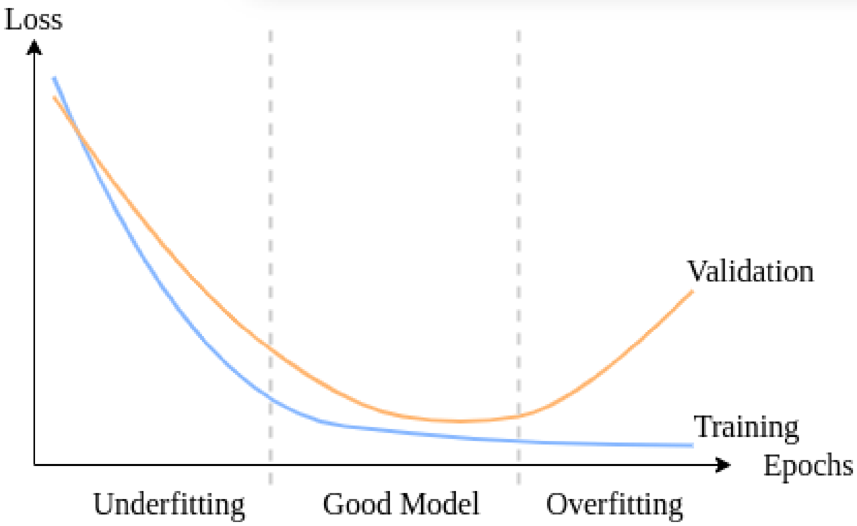


Learning rate	Batch size	Max word accuracy (%)	Training epochs
0.1	1	96.49	21
0.1	10	96.13	41
0.1	100	95.39	43
0.1	1000	84.13 +	4747 +
0.01	1	96.49	27
0.01	10	96.49	27
0.01	100	95.76	46
0.01	1000	95.20	1612
0.01	20,000	23.25 +	4865 +
0.001	1	96.49	402
0.001	100	96.68	468
0.001	1000	96.13	405
0.001	20,000	90.77	1966
0.0001	1	96.68	4589
0.0001	100	96.49	5340
0.0001	1000	96.49	5520
0.0001	20,000	96.31	8343

(Figure: analyticsvidhya.com
Table: Wilson and Martinez (2003))

Tracking Progress using Training and Validation Loss



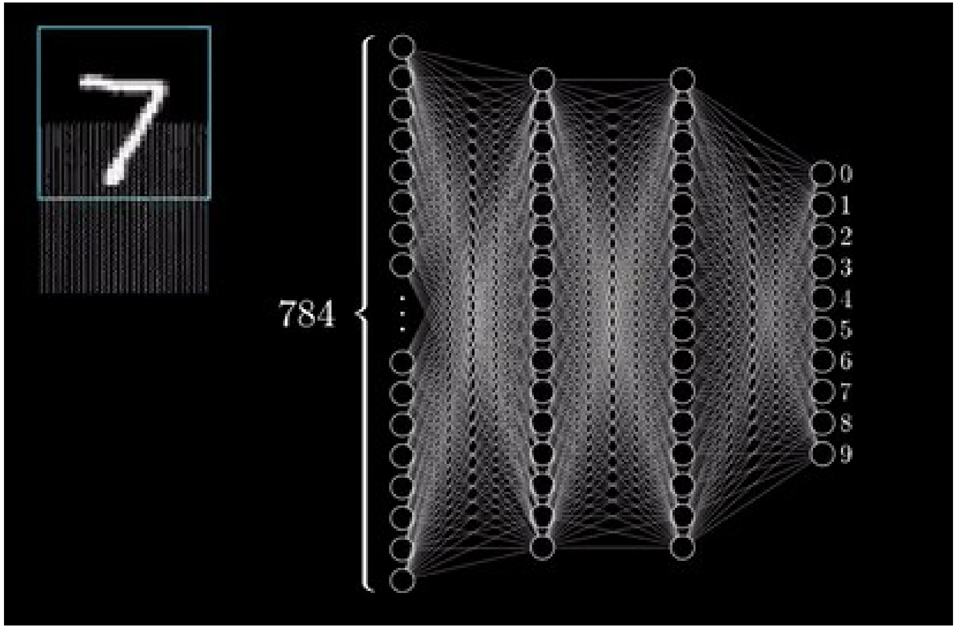


- Keeping track of validation loss, and using **regularization techniques** help prevent overfitting, enhance **generalizability,** and determine how long to train the model

- Early stopping, Dropout L2/L1 regularization etc.

Let's Code!

Jupyter Notebook URL: tiny.cc/coaps_lec2



- Train the ANN on MNIST data with Adam optimizer for learning rates: 10^{-1}, 10^{-2}, 10^{-3}
- Does SGD learn efficiently at the learning rate of 10^{-3}?
- Does adding momentum (say ~0.9) to SGD help?
- Perhaps the learning rate is too small for SGD. What if the leaning rate is increased?
- We have achieved an impressive recognition skill with Adam and a small learning rate. Does training the model indefinitely produce better and better skill?
- What if we use a large batch size, say 10000, with these best parameters and fast converging optimizers?