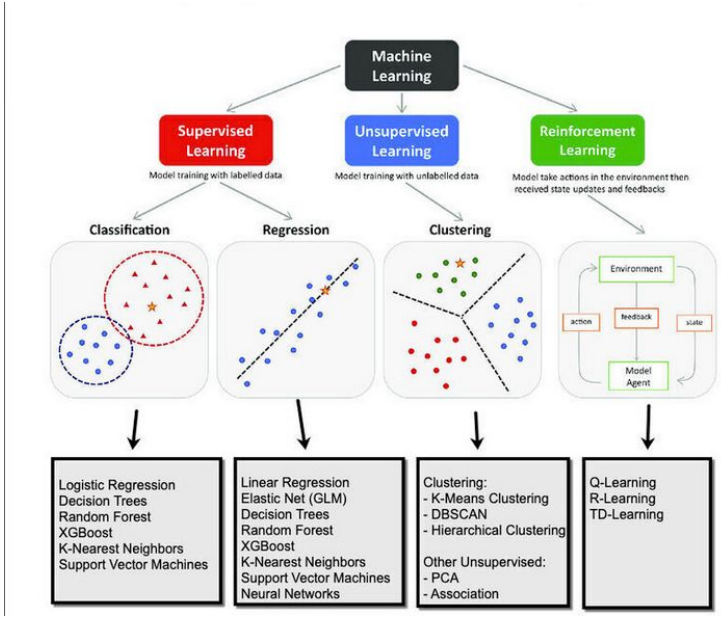
RRC Summer School Day 8

Basics of DL

Types Learning

Classified based on availability of data and the nature of data



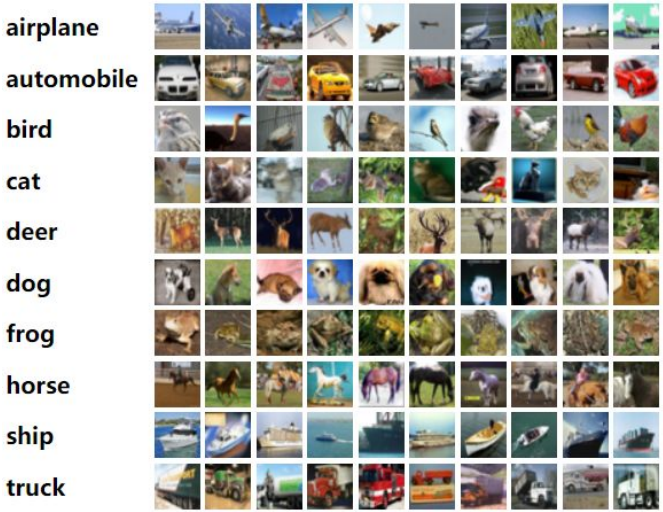
https://www.techplayon.com/

Supervised

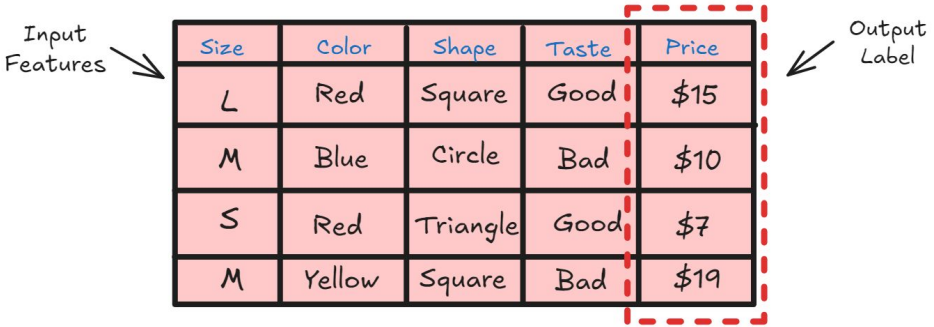
✅ Problem:

Build a model that can classify images of animals into categories like **cats, dogs, and birds**.

The model is explicitly trained with **input–output pairs**, learning a mapping from inputs (images) to known outputs (labels).



Cifar 10 dataset



Size	Color	Shape	Taste	Price
L	Red	Square	Good	$15
M	Blue	Circle	Bad	$10
S	Red	Triangle	Good	$7
M	Yellow	Square	Bad	$19

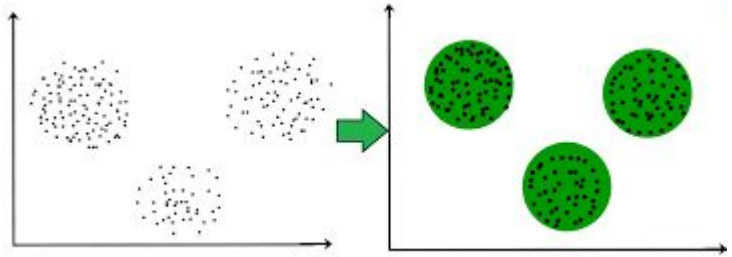
https://www.nb-data.com/p/nbd-lite-1-intro-to-supervised-learning

Unsupervised

✅ Problem:

A retail or e-commerce company wants to **segment customers** into distinct groups to personalize marketing, **without any labeled data** about customer types.

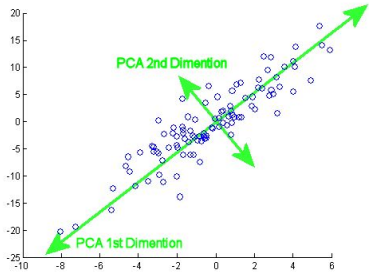
Goal is to: Discover hidden structure in data.



geeksforgeeks

Examples:

Clustering (e.g., K-means, DBSCAN)
Dimensionality reduction (e.g., PCA, autoencoders)
Generative modeling (e.g., GANs, VAEs)



Towardatascience

Semi supervised

✅ Problem:

Labeling medical images (e.g., MRI, CT scans, X-rays) requires **expert radiologists**, making it **costly and time-consuming** to build large labeled datasets. hospitals often have **thousands of unlabeled images** stored.

1,000 chest X-rays with **expert-verified labels** (e.g., pneumonia, normal, TB), 50,000 chest X-rays with **no labels**.

Can you Train a model to detect pneumonia with high accuracy using both labeled and unlabeled data?

- Train a deep convolutional neural network (e.g., ResNet) on the **1,000 labeled images**.
- Use a semi-supervised technique such as:
    - **Pseudo-labeling:** Predict labels for the 50,000 unlabeled images and retrain the model using high-confidence predictions.

Self Supervised

✅ Problem:

Training a high-performance natural language model typically requires massive labeled datasets, which are expensive to create. there's an abundance of **unlabeled text data** (e.g., books, articles, websites).

Masked Language Modeling (MLM): Randomly mask some words in a sentence. Example: <s> smoking causes [MASK] </s>

The training signal (e.g., predicting masked words or sentence order) comes **from the data itself**, **not from human-provided labels**. The model generates its own labels from the context.

Basics of Machine Learning

- The perceptron
- Design idea: How and why it works
- Multiple percptrons forming a network
- Forward Propagation
- Loss
- Gradients and backpropagation
- Optimization
- A simple network training pipeline

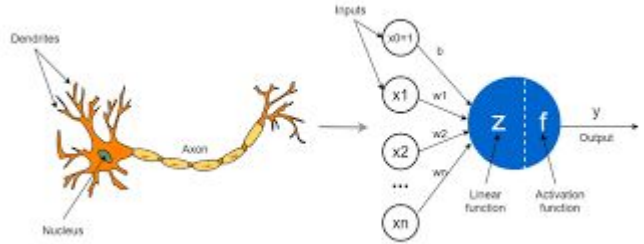
Perceptron

Quite like a biological neuron, this perceptron takes in multiple inputs,

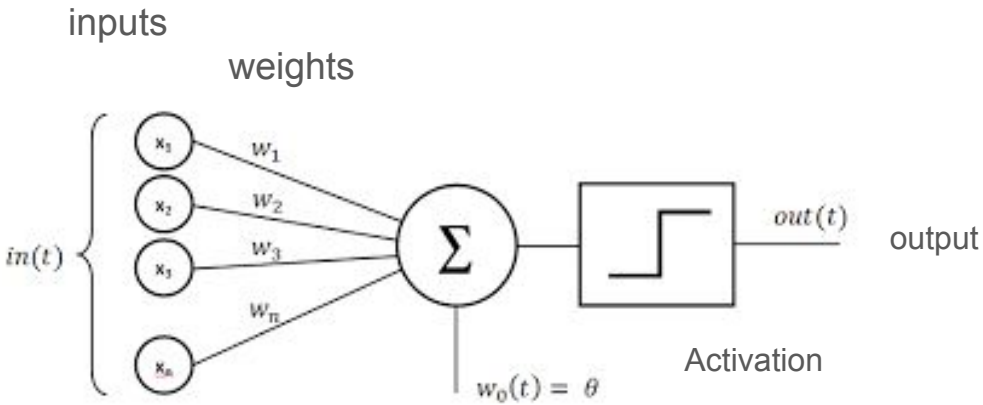
Weights them,

Gets activated

And serves as the input to the neurons after it



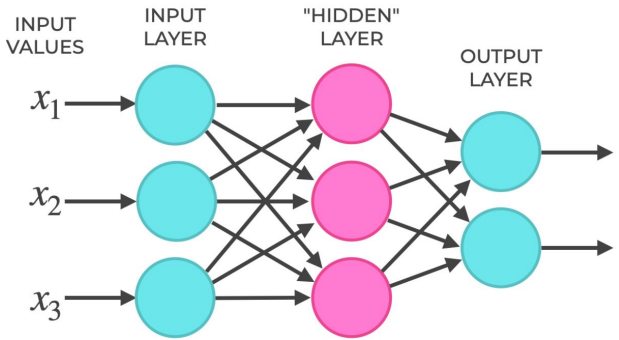
Perceptron



One bias term per neuron for all the inputs

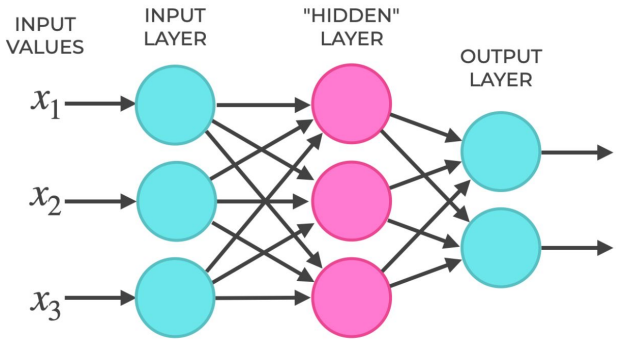
Multiple neurons forming a network “Multi layered perceptron”

- grouped into multiple layers.
- These layers allow the network to learn complex functions and patterns in data
- MLP has one or more hidden layers between them
- Hidden layers are where most of the learning and computation happen. They help the MLP capture complex pattern
- The more the number of layers, the more complex functions the network can learn to mimic



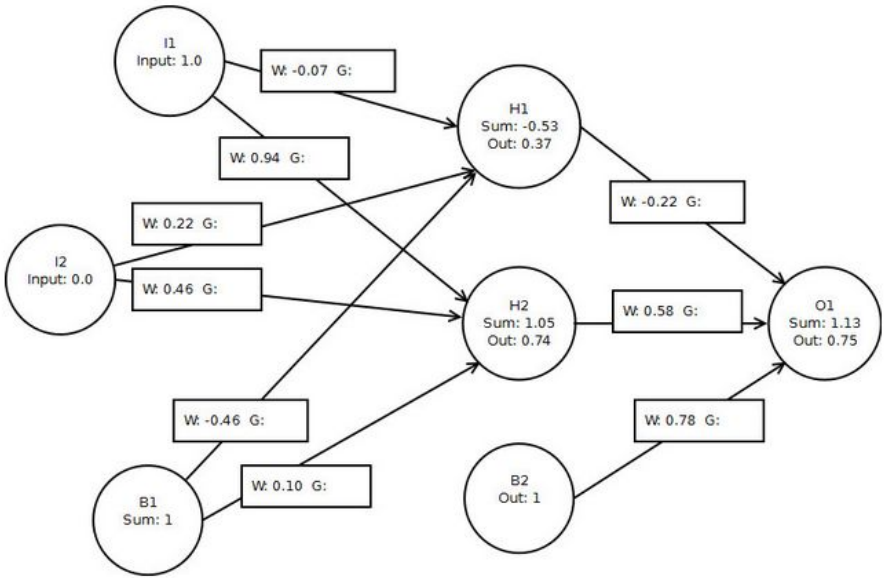
Multiple neurons forming a network “Multi layered perceptron”

- Step 1: Input Layer — Receiving Data
- Step 2: Weights and Biases — Adjusting the Importance of Inputs
- Step 3: Hidden Layers — Processing with Nonlinear Functions
- Step 4: Forward Propagation — Moving Data Through the Layers
- Step 5: Output Layer — Producing the Final Prediction
- Step 6: Training the Network — Adjusting Weights and Biases
- Step 7: Using the Trained Model for Predictions



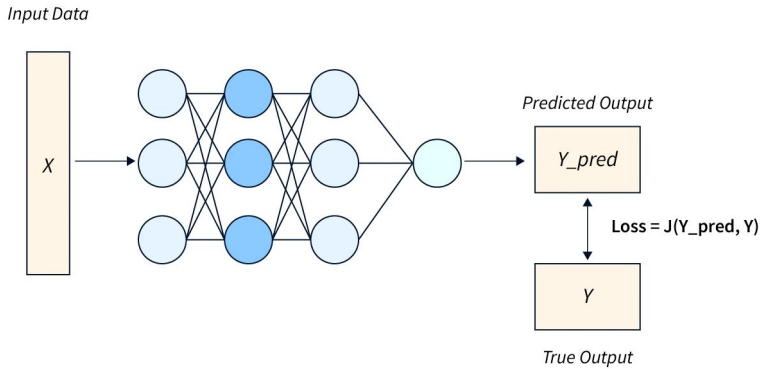
Forward Propogation

-



Loss Computation

- a function that measures how well a machine learning model's predicted outputs match the true output labels
- The goal of the “training” process is to optimize the model by minimizing the loss, which means that the model makes fewer mistakes on the training data
- evaluate the model's performance and guide the optimization process by indicating how the model's predictions differ from the true output labels.

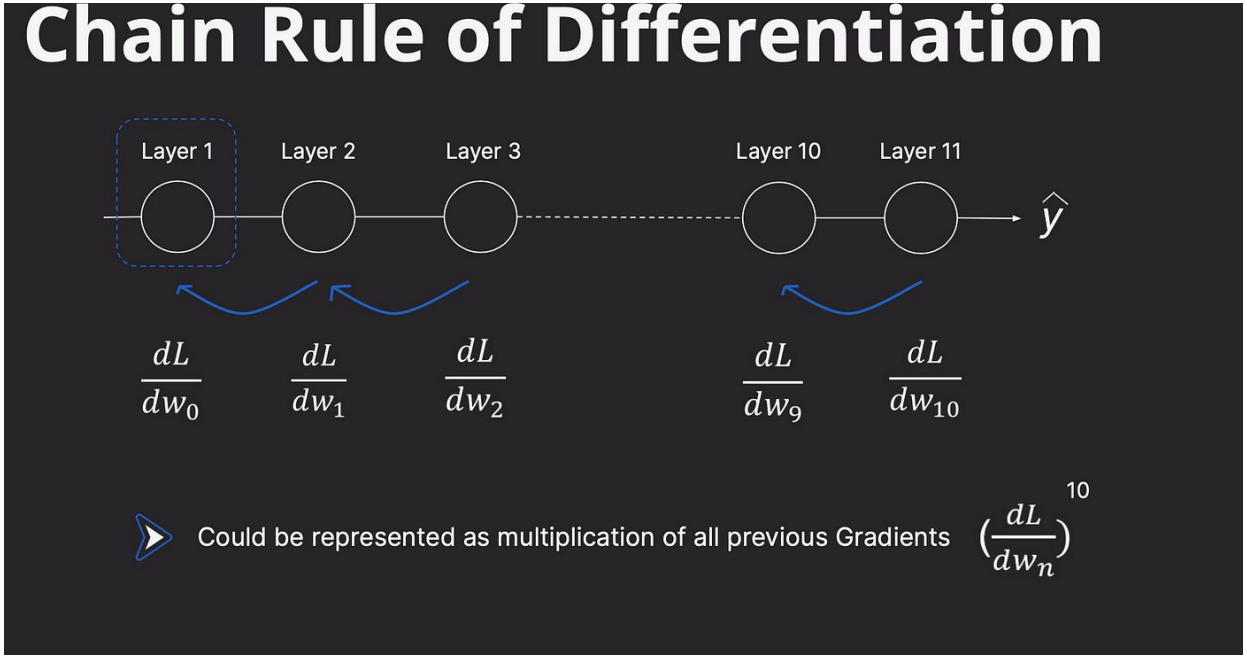


Examples:

- Mean squared error (MSE) loss (basically an L2 distance)
- L1 loss
- Binary Cross entropy loss for binary classification
- Multiclass cross entropy loss for multiclass classification
- KL Divergence for difference between two probability distributions, etc

Gradient calculation

After calculating Forward prediction(propagation) and calculating loss, calculate each partial derivatives by applying chain rule of derivative, then modify parameters with these gradient.



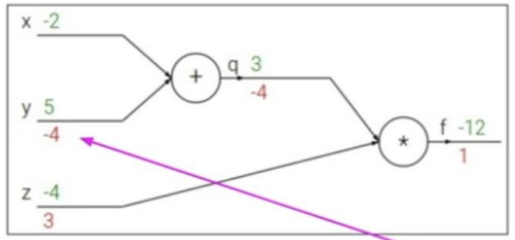
Gradient calculation

Simple example: $f(x, y, z) = (x + y)z$

$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$

$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



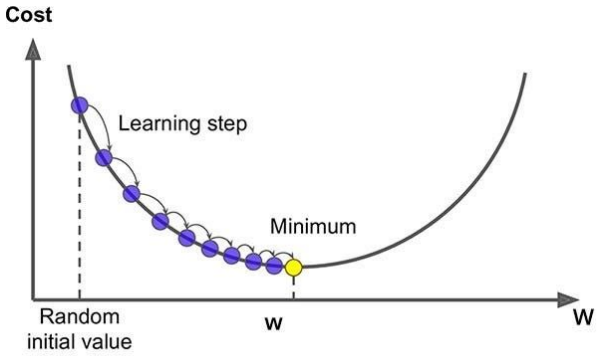
Chain rule:

$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$

Optimization

Updating weights based on the gradients computed. Do this for all weights and all biases for each neuron, at each training step. The following is Gradient Descent as an example.

Other optimizers are AdaM, RMSProp (and their modifications, etc)



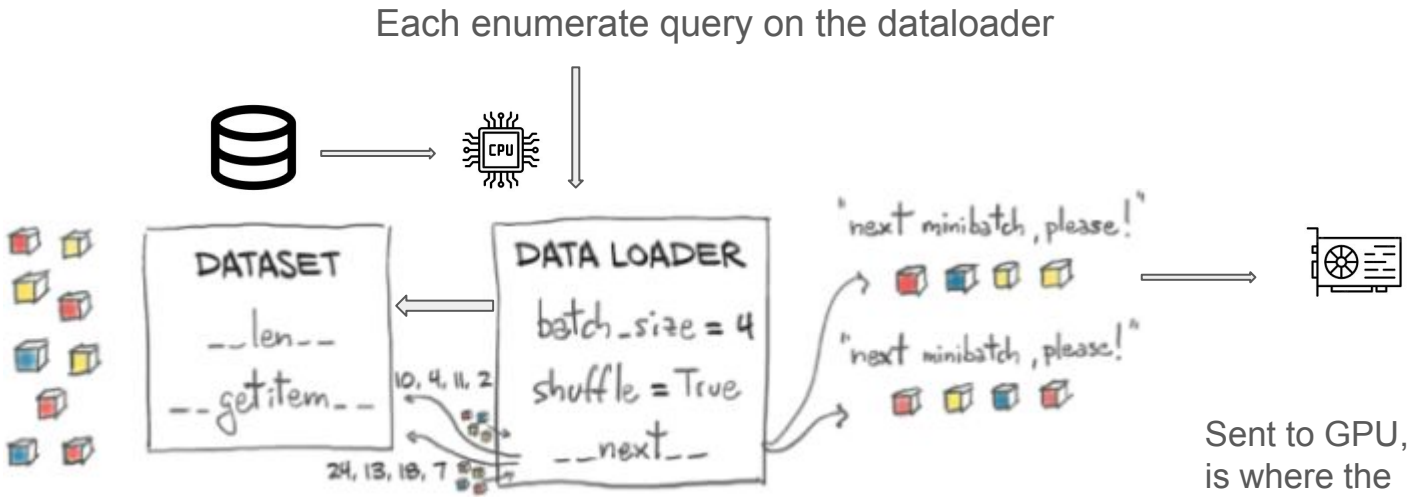
$$\underbrace{\mathrm{w}^{(t+1)}}_{\text{position of next iteration}} = \underbrace{\mathrm{w}}_{\text{position of previous step}} - \underbrace{\alpha \nabla f_i(\mathrm{w}^{(t)})}_{\text{step}}$$

learning rate

observation i

Simple network training pipeline

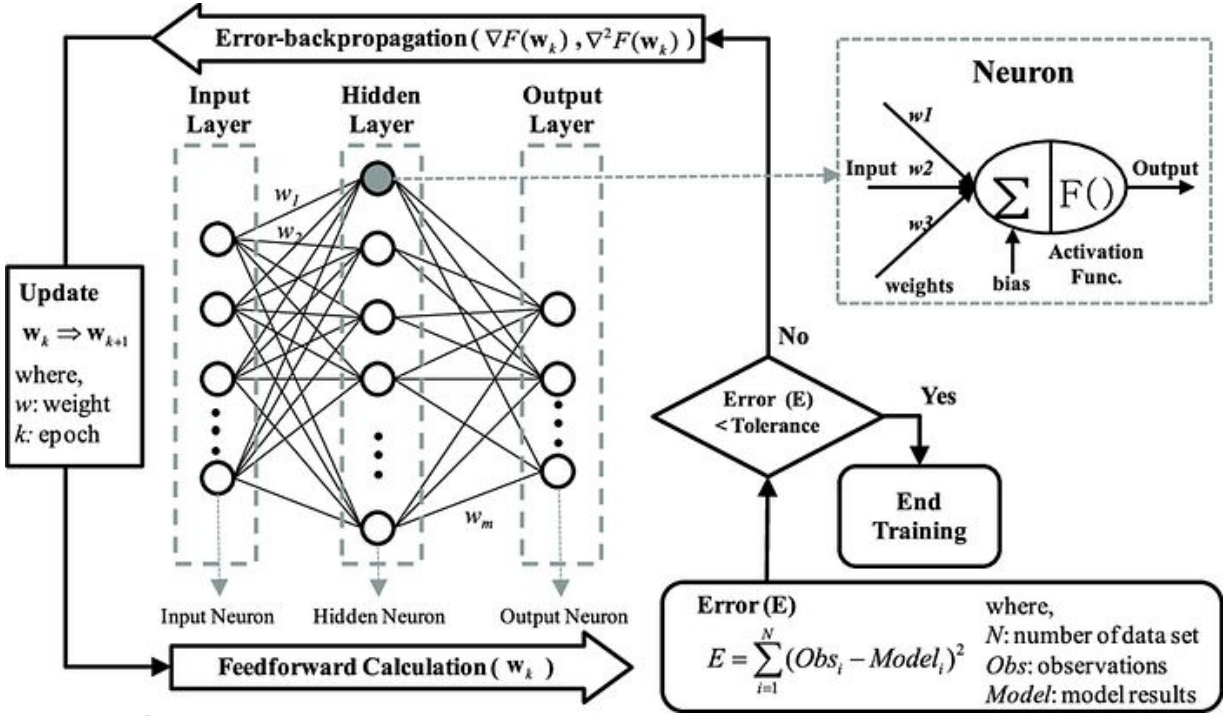
Loading the data:



Data read from the disk

Sent to GPU, that is where the model lives

Simple network training pipeline



Start here

Pseudocode

```
1. Load data
2. For i in len(dataset):
    a. Let training_example, label <- dataset[i]
    b. Input <- training_example
    c. For layer in model.layers:
        i.  input = layer(output)
    d. Let Output <- input
    e. Let loss <- loss_function(output, label)
    f. Loss.backward # calculate gradients for each learnable parameter in
       the network
    g. optimizer.step() # update each of the learnable parameters
```

Your model is trained. This was one “epoch”, i.e. entire dataset passes through the model once. Usually trained for multiple (40-50, 100-200, 400,etc epochs)

Resources

- 3b1b’s deep learning playlist
- Andrew Ng’s deeplearning course
- deepLizard youtube channel
- Textbook: deep learning by Ian Goodfellow
- Textbook: Mathematics for machine learning, by A Aldo Faisal et al