(An Almost) No Math Introduction to Deep Learning

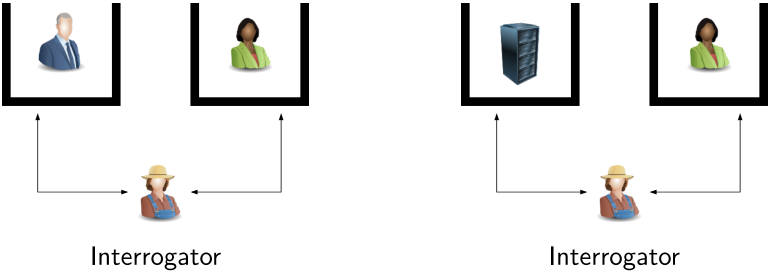
Abhijat Vatsyayan [1]

November 19, 2020

Summary

- A little bit of history
- Different flavours of AI
- Machine learning
  - Description, types
  - A little bit of math
  - An illustrative example
- If time permits
  - ImageNet challenges and Deep neural networks
  - Convolution networks, auto-encoders, transfer learning
  - Adversarial networks
  - Other architectures

Imitation game



Artificial intelligence

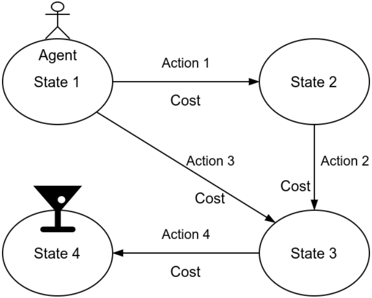
- Logic machines (50s)
- Knowledge based expert systems (80s)
- Language translation (60s) , 2000s, 2014 and later.
- Machine learning
  - Neural networks including deep learning (started in 1943)
  - Support vector machines
  - Baysian learning
- Graphs
- Genetic algorithms and genetic programs.

Expert systems

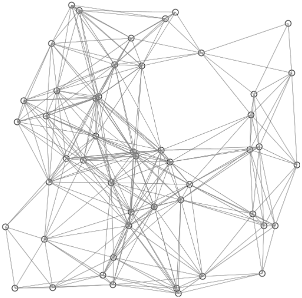
- Database of formally described "facts" or "knowledge".
- A reasoning engine for answering questions or solving problems.

Not to be confused with a true natural language processing and question-answering system.

Search


Agent
State 1
Action 1
Cost
State 2
Action 3
Cost
Action 2
Cost
State 4
Action 4
Cost
State 3

?



?

Neural Networks

1943: Warren McCulloch and Walter Pits connected neurons, computation, logic and learning.

1950: Minsky and Dean Edmonds build first neural network computer. 3000 vacuum tubes, surplus auto-pilot parts from B-24 bomber. 40 Neurons.

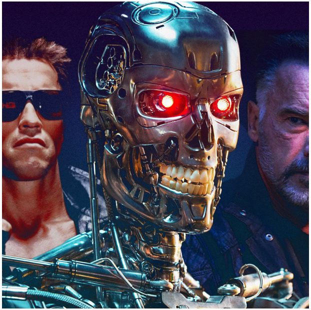
1969: Minsky and Papert publish perceptron - simple linear networks could not learn basic functions.

1980s: David Rumelhart, Jeff Hinton and Ronald Williams applied back propagation (again) for training multi-layer neural networks. Rumelhart's work also created the foundations for Recurrent Neural Networks.

1990s: LSTM networks by Hochreiter and Schmidhuber 1997. CNN for handwritten digit recognition - Yann LeCun.

2000s: LSTMs show promise in speech recognition

Different views



Agent

Tool

General

Narrow

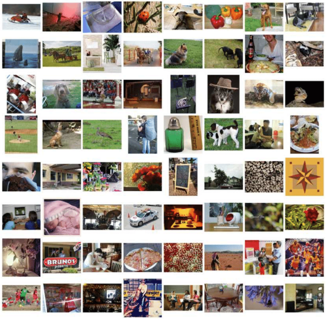
Datasets

- Modified National Institute of Standards and Technology - MNIST (60k/10k)
- Canadian Institute For Advanced Research - CIFAR-10 (50k/10k) and CIFAR-100 (2 level, 500/100)
- Pascal Visual Object Classes (VOC) - 22k images, 20 classes
- ...
- ImageNet

ImageNet Large Scale Visual Recognition Challenge

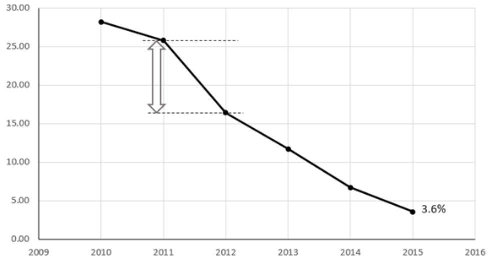
MNIST Dataset (60k, 10k)

ImageNet (14M+, 22k)



ImageNet Large Scale Visual Recognition Challenge

- Publicly available dataset - ImageNet (14M+, 22k categories)
- Annual competition
  - Image classification
  - Object detection and localization
- Increasing depth
  - 8 layer AlexNet
  - 19 layer GoogLeNet
  - 152 layer ResNet



Top 5 classification error rate

Using linear layer

Image as 2D Tensor(Matrix)

1	2	3	4	5
6	7	8	9	10
5	4	3	2	1
10	9	8	7	6

2D $\rightarrow$ 1D

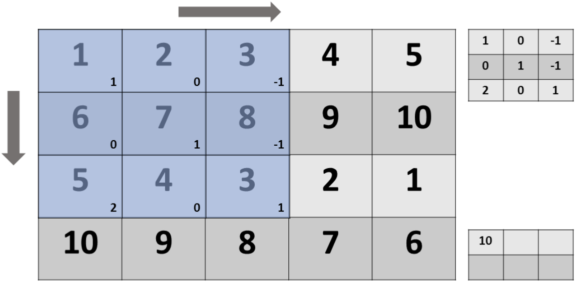
$$\mathbb{W}_{m \times 20} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ \dots \\ 8 \\ 7 \\ 6 \end{bmatrix} + \mathbb{B}_{m \times 1}$$

2D Convolution Example

1	2	3	4	5
6	7	8	9	10
5	4	3	2	1
10	9	8	7	6

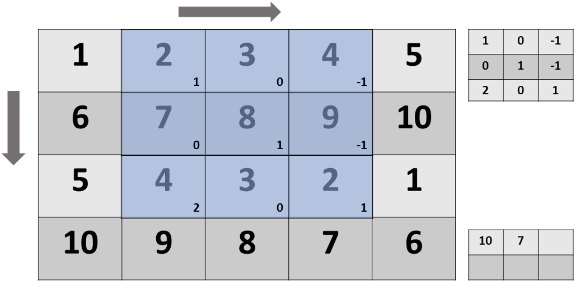
1	0	-1
0	1	-1
2	0	1

2D Convolution Example

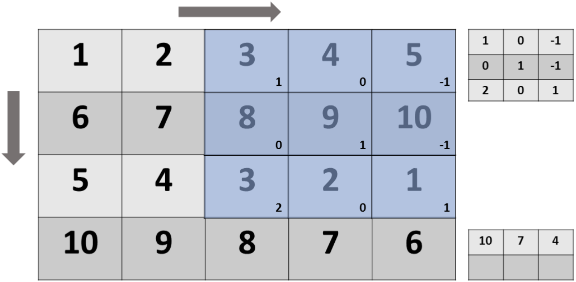


$$
(1 \times 1) + (2 \times 0) + (3 \times -1) +
$$
$$
(6 \times 0) + (7 \times 1) + (8 \times -1) +
$$
$$
(5 \times 2) + (4 \times 0) + (3 \times 1)
$$

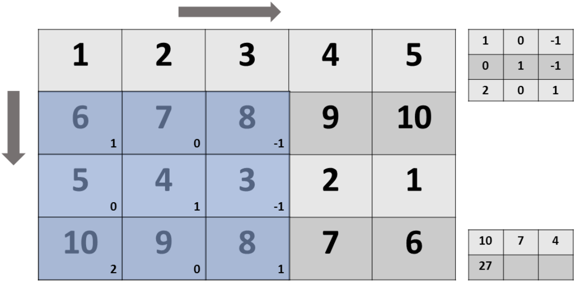
2D Convolution Example



2D Convolution Example



2D Convolution Example

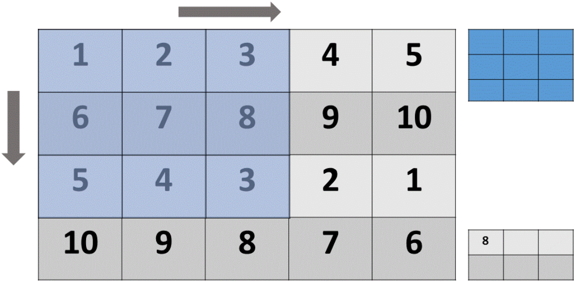


2D Convolution

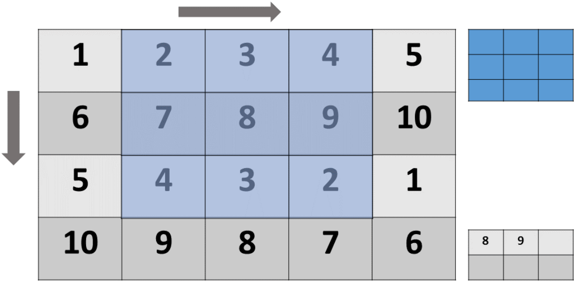
- Bias $wx + b$
- Stride
- Padding
- Layers or channels

For a 5×5 filter with bias, you need 26 parameters for gray scale images. If you have 3 channels (rgb), you need $5 \times 5 \times 3 + 1 = 76$ parameters.
Layers typically have multiple filters, each filter resulting in a single output channel. Hence, a layer with 200 5×5 filters (with bias) for 3 channel inputs will have $76 \times 200 = 15,200$ parameters.
Corresponding output will contain 200 channels.

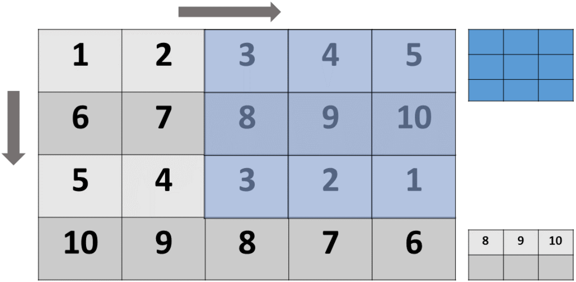
Max pooling



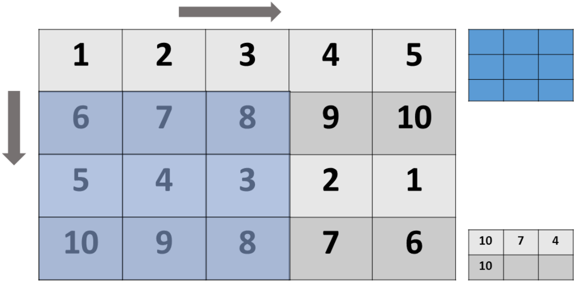
Max pooling



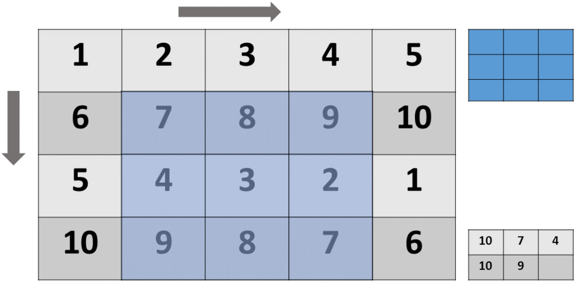
Max pooling



Max pooling



Max pooling



Max pooling

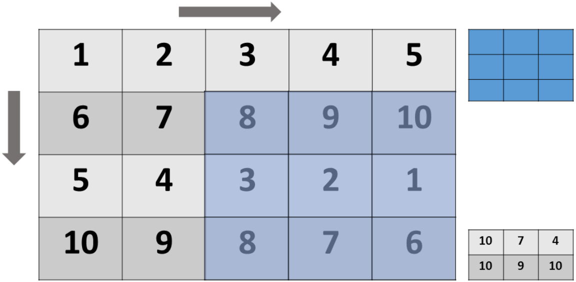


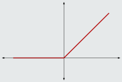
Image processing

Other functions

- RelU activation: $max(x, 0)$
- Leaky RelU

$$f(x;.1) = \begin{cases} x, & \text{if } x \geq 0 \\ .1x, & \text{otherwise} \end{cases}$$

- Dropout layer

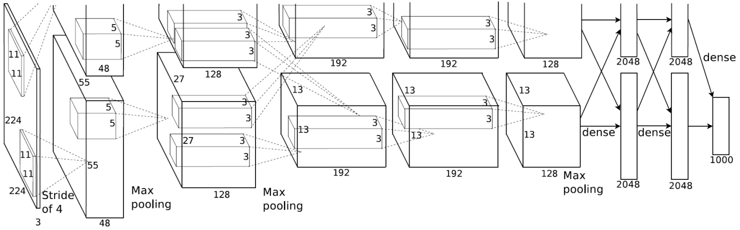


RelU

AlexNet 2012

- 8 Layers, 5 Convolutional, 3 fully connected
- Used RelU and max-pooling
- 61M parameters

AlexNet 2012



Alex Krizhevsky, Sutskever, Ilya and Hinton, Geoffrey E., ”ImageNet Classification with Deep Convolutional Neural Networks”, 2012

Typical convolution net, pytorch

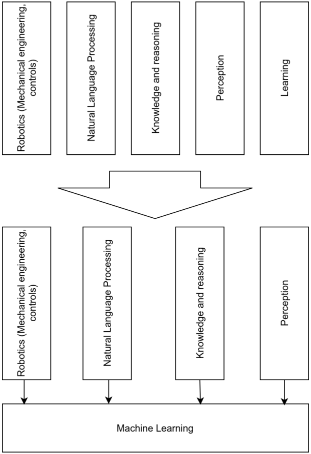
Listing 1: Typical (simple) CNN in pytorch

```
class ConvNet(nn.Module):
  def __init__(self):
    super(ConvNet, self).__init__()
    self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
    self.lrelu1 = nn.LeakyReLU(.1)
    self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
    self.lrelu2 = nn.LeakyReLU(.1)
    self.maxpool1 = nn.MaxPool2d(kernel_size=3, padding=1)
    self.dropout1 = nn.Dropout(p=.25)
    self.conv3 = nn.Conv2d(in_channels=32, out_channels=32, padding=1, kernel_size=3)
    self.lrelu2 = nn.LeakyReLU(0.1)
    self.conv4 = nn.Conv2d(in_channels=32, out_channels=64, padding=1, kernel_size=3)
    self.lrelu3 = nn.LeakyReLU(0.1)
    self.maxpool2 = nn.MaxPool2d(kernel_size=3, padding=1)
    self.dropout2 = nn.Dropout(p=.25)

    self.conv_layers = [self.conv1, self.lrelu1, self.conv2, self.lrelu2, self.maxpool1,
    self.conv3, self.lrelu2, self.conv4, self.lrelu3, self.maxpool2, self.dropout2]

    self.fc1 = nn.Linear(in_features=64*4*4, out_features=256)
    self.lrelu4 = nn.LeakyReLU(.1)
    self.dropout3 = nn.Dropout(.25)
    self.fc2 = nn.Linear(in_features=256, out_features=10)
    self.softmax = nn.Softmax(dim=1)

```

Why do you think this picture is funny?

Credit: *http://karpathy.github.io/2012/10/22/state-of-computer-vision/*

Late 2000s - machine learning dominates



- Image classification, localization and segmentation
- Neural machine translation, question answering, summary.
- Game playing, helicopter flying (stunts)
- Planning, self driving cars
- Text, audio and video processing, generation
- ...

Machine learning I

Models

- Build a model of the world
- Infer/predict using the model.

Machine learning

- Supervised learning
- Unsupervised learning
- Reinforcement learning

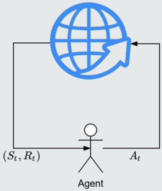
Machine learning II

Unsupervised learning

- Training a model to find patterns in a dataset, typically an unlabeled dataset.
- Learning how to extract *interesting* features.
- Learning data distribution for generating data.

Reinforcement learning

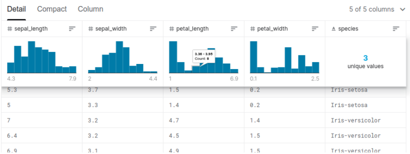
A family of algorithms that learn an optimal policy, whose goal is to maximize return when interacting with an environment.



Supervised learning I

Detail Compact Column

# sepal_length	# sepal_width	# petal_length	# petal_width	species
5.7	2.9	4.2	1.3	Iris-versicolor
6.2	2.9	4.3	1.3	Iris-versicolor
5.1	2.5	3	1.1	Iris-versicolor
5.7	2.8	4.1	1.3	Iris-versicolor
6.3	3.3	6	2.5	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
7.1	3	5.9	2.1	Iris-virginica
6.3	2.9	5.6	1.8	Iris-virginica
6.5	3	5.8	2.2	Iris-virginica

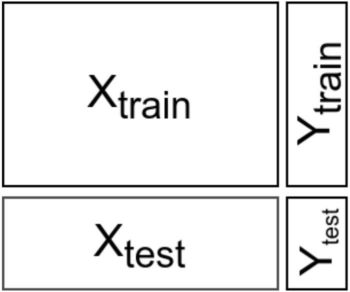


# sepal_length	# sepal_width	# petal_length	# petal_width	species
4.3 – 7.9	2 – 4.4	1 – 6.9	0.1 – 2.5	3 unique values
5.3	3.7	1.5	0.2	Iris-setosa
5	3.3	1.4	0.2	Iris-setosa
7	3.2	4.7	1.4	Iris-versicolor
6.4	3.2	4.5	1.5	Iris-versicolor
6.9	3.1	4.9	1.5	Iris-versicolor

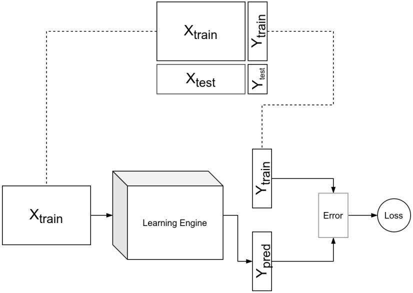
Supervised learning II

Source: *https://www.kaggle.com/arshid/iris-flower-dataset?select=IRIS.csv*

150 rows, 5 attributes (columns), 4 numerical and 1 categorical.

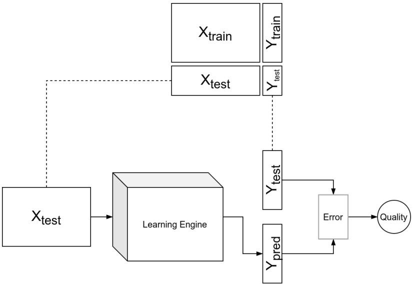


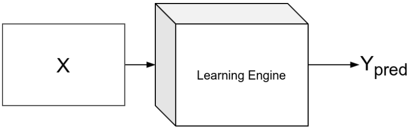
Data



Training

Supervised learning III





Testing

Predicting

Fitting a function to data

Description (supervised learning)

Given a set of data $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, can we find a function $y = f(x)$ that "fits" this data?

Questions

- What is this function $f(x)$?
- What does "fit" mean?
- How do we know this works?
- What kinds of problems can we solve?

More about $f(x)$

Class of functions

Starting with a function $f(x; \theta_1, \theta_2, \ldots, \theta_n)$ where x is the input to the function and θs are its parameters, we need to find the set of θs that best "fits" the give data $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$

Class of linear functions

Consider $f(x) = ax + b$. If we can say, with some confidence, that our data is linearly related, we need to find $\theta_1 = a$, $\theta_2 = b$ that *fits* the given data. We can also write it as $f(x; a, b) = ax + b$. Preferred,

$$f(x; \theta_1, \theta_2) = \theta_1 x + \theta_2 \tag{1}$$

Fit

Mean squared Euclidean distance as one possible measure of fit

Let L_i be the squared Euclidean distance between the predicted value, $\hat{y}_i = f(x_i)$ and the actual, y_i. Then,

$$L_i = z_i^2 \tag{2}$$

$$z_i = y_i - f(x_i) \tag{3}$$

$$= y_i - \theta_1 x_i - \theta_2 \tag{4}$$

Minimizing L_i with respect to the parameters θ_1 and θ_2,

$$\frac{\partial L_i}{\partial \theta_1} = \frac{\partial z_i^2}{\partial z_i} \frac{\partial z_i}{\partial \theta_1} \tag{5}$$

$$\frac{\partial L_i}{\partial \theta_2} = \frac{\partial z_i^2}{\partial z_i} \frac{\partial z_i}{\partial \theta_2} \tag{6}$$

For n data points, mean loss is

$$L = \frac{1}{n}\sum_{i=1}^{i=n} L_i = \frac{1}{n}\sum_{i=1}^{i=n}(y_i - \theta_1 x_i - \theta_2)^2$$

In practice

- Choose a small (64 or 128) random subset of training data.
- Compute predicted values, then loss.
- Compute gradients of loss W.R.T. parameters then update parameters.

An **epoch** refers to a single iteration over all training data.

Two different spaces

- Space spanned by x and y. Optimization tries to find the surface (model) in this space that best fits the data.
- Space spanned by θs. We minimize the loss function in this space.

References I

V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," 2018.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.

I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations*, 2015.

A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *ICLR Workshop*, 2017.

I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

References II

I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 3104–3112, Curran Associates, Inc., 2014.

S. J. Russell and P. Norvig, *Artificial Intelligence: a modern approach*. Pearson, 3 ed., 2009.