Lecture1 (History, Evolution and revolution of DL) 21/12/2020

Inspiration of DL: The Brain

-> McCulloch & Pitts (1943): networks of binary neuron can do logic

-> Donald Hebb (1947): Hebbian synaptic plasticity

-> Norbert Wiener (1948): cybernatics, optimal filter, feedback, autopoises, auto-organization

-> Frank Rosenblatt (1957): Perceptron

-> Hubel & ~~Gle~~ Wiesel (1960s): visual contex achitecture

stopped and again started in 1980s and died in 1995 again and again rose up in 2010s with speech recognition.

2012-2013 -> rise of Computer Vision

2016 -> rise of NLP

Supervise Learning: Training a machine by showing examples instead of programming it.

Works well for

Speech -> words

Image -> categories

Portrait -> name

Photo -> caption

Text -> topic

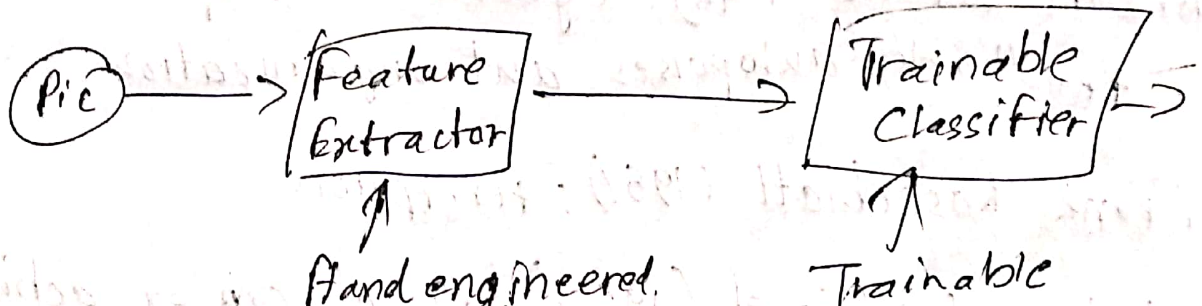
Supervised Learning goes back to the Perceptron and Adaline

The McCulloch-Pitts Binary Neuron

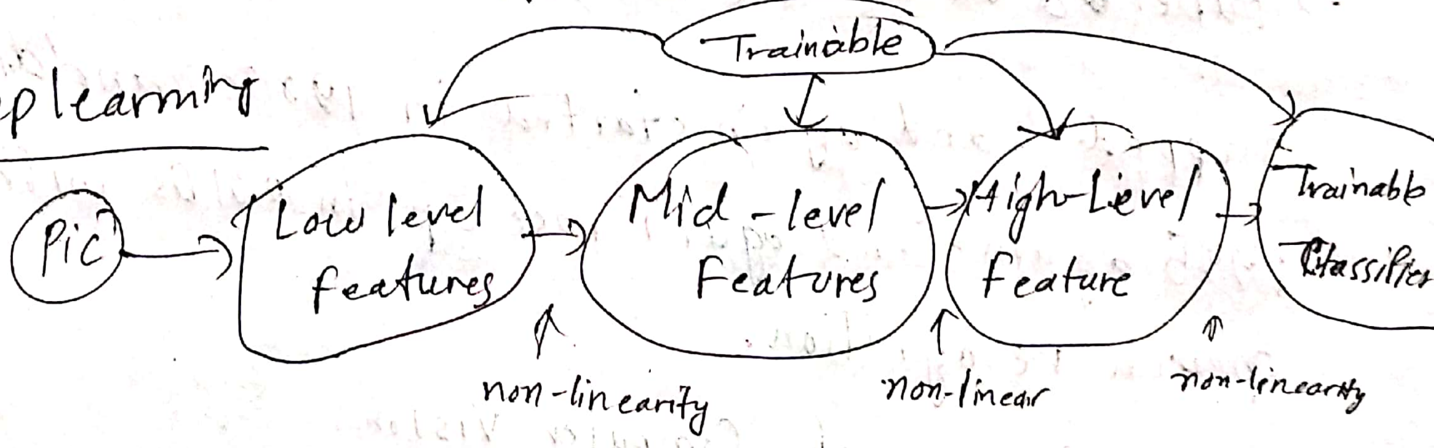
→ Perceptron: Weights are motorized potentiometers

→ Adaline: Weights are electrochemical memistors

Traditional Machine Learning



Deeplearning



MLP (Multi-layer Perceptron)

$$\text{ReLU}(x) = \max(x, 0)$$

Supervised Machine learning = function Optimization

Stochastic gradient descent

$$w_i \leftarrow w_i - \eta \frac{\partial L(w, X)}{\partial w_i}$$

Computing Gradients by Back-Propagation:

$$\frac{dC}{dW_n} \longrightarrow$$

Traditional NN has usability problems and many others dealing with images.

- Hubel and Wiesel's Model of the Architecture of the Visual Cortex
  - simple cells detect local features [1962]
  - complex cells "pool" the outputs of simple cells within a space
  - → They found out how things work in Brains for recognition

[Fukushima 1982]
[LeCun 1989, 1998]

ConvNets can recognize multiple objects

- All layers are convolutional
- Networks performs simultaneous segmentation and recognition

Face and Pedestrian Detection with ConvNets (1993-2015)

Training a Robot to Drive itself in Nature [Hadsell 2009]

Semantic Segmentation with ConvNets [Farabet 2012]

1986-1996] ⇒ ~~A~~ Special NN chips

Deep learning Revolution

Speech Recognition 2010

Image Recognition 2013

NLP 2015

Deep ConvNets for Object Recognition (on GPU)

AlexNet [2013] OverFeat [2013]

VGG [2013]

GoogleNet [2013]

ResNet [2015]

DenseNet [2017]

Multilayer Architectures == Compositional structure of Data

Picture → Low-level Feature → Mid-level feature → High-level Feature → Trainable Classifier

MSRA_2015 Mask R-CNN 2017 Mask R-CNN 2017

® Mask-RCNN on COCO dataset

→ Individual objects are segmented

3D ConvNet for Medical Image Analysis

Deep learning enables

→ Safer cars, autonomous cars

→ Better medical image analysis

→ Personalized medicine

→ Adequate language translation

→ Useful but stupid chatbots

→ Info search, retrieval, filtering

→ other

It can't yet

Machines with common sense

Intelligent personal assistants

Smart chatbots

Household robots

Agile and dexterous robs

Artificial General Intelligence

Deep learning : Learning Representations/Features

Hierarchial representation

→ Hierarchy of representations with increasing level of abstraction

→ Each stage is a kind of trainable feature transform

SVM is nothing but two layer neural nets and the first layer is trained in an unsupervised way

→ Deep machines are more efficient for repesenting certain classes of functions.

Why would DL be more efficient? [Bengio, Lecun 2007]

→ less params needed to get good result

What are Good Features?

Discovering the Hidden Structure in High Dimensional Data: The Manifold Hypothesis

The Manifold Hypothesis

- Natural data lives in a low-dimensional manifold [non-linear]
- Because variables in natural data are mutually dependent

Disentangling factors of variation

- The ideal disentangling Feature Extractor
- PCA can find the representation if they are linear

Practicum 1 [Yann LeCun, Alfredo Canziani
Mark Goldstein]

Classification, linear algebra and visualization

We have to find the point of classification ~~to~~ and bring it to the main focus

Matrix multiplication: (what does it do?) [Linear transformation]

- Rotation
- Stretching/Scaling (Zooming / shearing)
- Reflection
- Translation
- Affine transformation

- Neural net does this by trying to reach convergence

~~B.~~ We need non-linearity because without ~~non-linear~~ non-linearity we can't get

Space Stretching

We aim to divide the space so that we can get linear seperable translation

Week 2: Lecture: SGD and Backprop

Parameterized Model

$\bar{y} = G(x, w)$ [Parameterized Deterministic Function]

x: params, w: weights

Computing function G may involve complicated algo.

Variables (tensor, scalar, continuous, discrete, ...)

Deterministic function

Scalar valued function (impicit ouput)

L Loss function and average loss

- machine learning is all about optimizing functions

~~Sto~~ Gradient descent

$$w \leftarrow w - \eta \frac{\partial L(S, w)}{\partial w}$$

- SGD exploits the redundancy in the samples
- we use minibatch for parallelization in practice

There are optimization algorithms that are not dependendal on gradient specially the ones ~~t~~ where we don't know the grads or can't get the grads. Those are called zeroth order method.

- In RL, the output function is not differentiable most of the time. which makes it really inefficient.
- Reward is the negative of cost. in RL. RL practioners generally use a secondary cost function which is differentiable to make RL efficient.

Traditional NN

Stacked linear and non-linear functional blocks

$$s[i] = \sum_{j \in UP(i)} w[i,j]\, z[j] \qquad z[i] = f(s[i])$$

Backprop through a non-linear functions

Chainrule $g(h(s))' = g'(h(s)) \cdot h'(s)$

~~Perbutations:~~

Perturbations:

Linear blocks $s_{k+1} = \omega_k z_k$

Nonlinear blocks $z_k = h(s_k)$

PyTorch definition: (object oriented version)

nn. Linear automatically adds a bias factor

Jacobian Matrix

Partial derivatives of ith output wrt jth input

$$\left(\frac{\partial z_g}{\partial z_f}\right)_{ij} = \frac{(\partial z_g)_i}{(\partial z_f)_j}$$

Backprop through a multi-stage graph

Two Jacobian matrices

One with respect to $z[k]$ → input

One with respect to $w[k]$ → weight

John Vidal

Log softmax $y_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$ [vector of probabilities] $\log y_i$

$\log y_i = x_i - \log \sum_j e^{x_j}$

Practical tips of Backprop

softmax is just a generilization of ~~sof~~ sigmoid with multiple outputs

- Use ReLU · (tanh and others are ~~pd~~ falling out of favor)
- Use cross entropy for classification
- Use sgd on minibatches
- Shuffle the training samples
- Normalize the input variables (zero mean, unit variance)
- Schedule to decrease the learning rate
- Use a bit of L1 ~~and~~ or L2 regularization on the weights (Weight Decay)
  - → it's best to turn ~~on~~ it on after couple epochs
- Use dropout for regularization (distribute info better
- Lecun → Effient Backprop (1998)
- Neural Nets, Tricks for trade

~~Weight Decay~~

- Any directed acyclic graph is ok for backprop

Practicum 2: Training a neural network

NN are mainly doing rotation and squashing

ANN - supervised learning

$$x_c(t) = t\begin{pmatrix} \sin\left(\frac{2\pi}{C}(2t+c-1)\right) + N(0,\sigma^2) \\ \cos\left(\frac{2\pi}{C}(2t+c-1)\right) + N(0,\sigma^2) \end{pmatrix}$$

$0 \leq t \leq 1, \quad c = 1, \ldots, C$

we try to adapt the decision boundaries by rotating and scaling

$\boxed{\text{Input}} \rightarrow \boxed{\text{Low level features}} \rightarrow \boxed{\text{Mid-level features}} \rightarrow \boxed{\text{High-level features}}$

If we have multiple output then it is called multi-headed network

Traindata

$x^{(1)} \in \mathbb{R}^n$

$$X = \overbrace{\begin{bmatrix} -x^{(1)}- \\ -x^{(2)}- \\ \vdots \\ -x^{(m)}- \end{bmatrix}}^{n} \updownarrow m \qquad C = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} \updownarrow m \rightarrow$$

1-hot encoding

$(1\,0\,0)$

$$Y = \overbrace{\begin{bmatrix} -y^{(1)}- \\ -y^{(2)}- \\ \vdots \\ -y^{(m)}- \end{bmatrix}}^{k} \updownarrow m$$

$c_i \in \{1, 2, \ldots K\}$

Fully connected layer

$$x \xrightarrow{W_h} f \xrightarrow{h \quad W_y} g \longrightarrow \hat{y}$$

$h = f(W_h x + b_h)$

$\hat{y} = g(W_y h + b_y)$

Affine transformation → rotation and scaling

Non-linear function → squashing

$W_h \in \mathbb{R}^{d \times n}$

$b_h \in \mathbb{R}^{d}$

$W_y \in \mathbb{R}^{k \times d}$

$b_y \in \mathbb{R}^{k}$

$f, g = (\cdot)^+, \sigma(\cdot)$

$\tanh(\cdot)$, soft(arg)max$(\cdot)$

- Backward pass computes the grads of loss w.r.t. our learnable params

$$\text{soft(arg)max}(l)[c] = \frac{\exp(l[c])}{\sum_{k=1}^{K} \exp(l[k])} \in (0,1)$$

$$\mathcal{L}(\hat{Y}, c) \doteq \frac{1}{m} \sum_{i=1}^{m} l(\hat{y}_i, c_i) \qquad l(\hat{y}, c) \doteq -\log(\hat{y}[c])$$

cross entropy / negative log likelihood

$x, c = 1 \Rightarrow y = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$

$\hat{y}(x) = \begin{pmatrix} \sim 1 \\ \sim 0 \\ \sim 0 \end{pmatrix} \Rightarrow l\left(\begin{pmatrix} \sim 1 \\ \sim 0 \\ \sim 0 \end{pmatrix}, 1\right) \to 0^+$

$\hat{y}(x) = \begin{pmatrix} \sim 0 \\ \sim 1 \\ \sim 0 \end{pmatrix} \Rightarrow l\left(\begin{pmatrix} \sim 0 \\ \sim 1 \\ \sim 0 \end{pmatrix}, 1\right) \to +\infty$

$\Theta \doteq \{W_h, b_h, W_y, b_y\}$

$J(\Theta) \doteq \mathcal{L}(\hat{Y}(\Theta), c) \in \mathbb{R}^+$

$-\frac{dJ(\vartheta)}{d\vartheta}(\vartheta_0)$

notebooks
- spiral classification
- ~~mse~~ regression

choice of activation will have effects on regression