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 & Wiesel (1960s): visual cortex 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

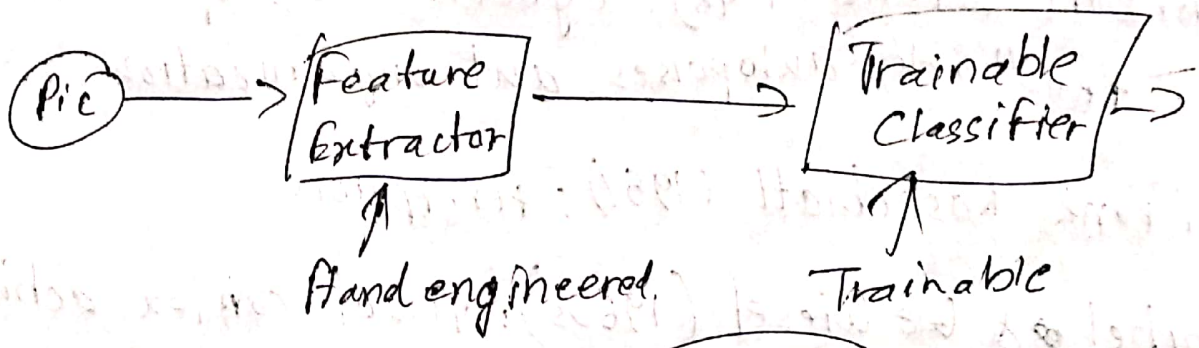
Supervised Learning goes back to the Perceptron and Adaline

The McCulloch-Pitts Binary Neuron

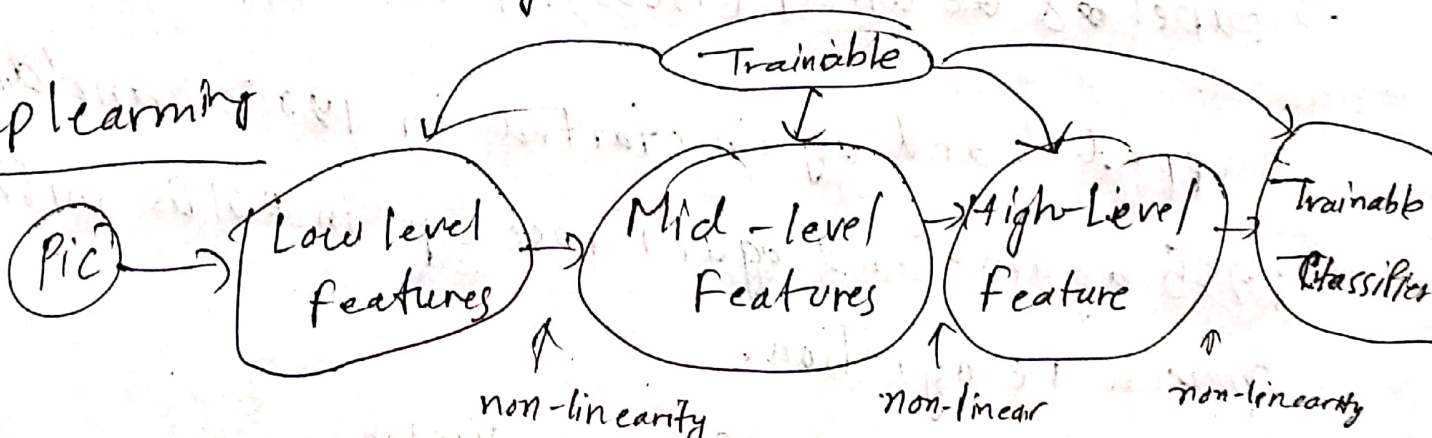
→ Perceptron: Weights are motorized potentiometers

→ Adaline: Weights are electrochemical memristors

Traditional Machine Learning



Deep learning



MLP (Multi-layer Perceptron)

$$ReLU(x) = \max(x, 0)$$

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:

$\frac{dC}{dWn}$ ⟶

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

- Hubel and Wiesel's Model of the Architecture of the Visual Cort...
  - 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 [R Hadsell 2009]
Semantic Segmentation with ConvNets [Farabet 2012]

1986-1996] ⇒ 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]

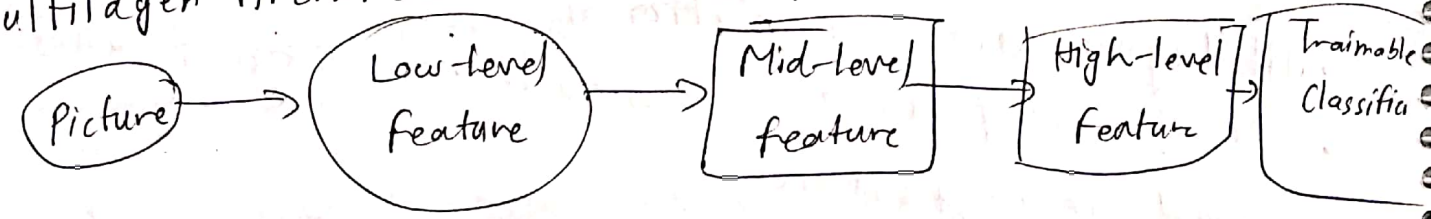
VGG [2013]

GoogleNet [2013]

ResNet [2015]

DenseNet [2017]

Multilayer Architectures == Compositional structure of Data



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 robots

Artificial General Intelligence

Deep learning = Learning Representations/Features

Hierarchical 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 representing 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 [non-linear] manifold
- 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 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

We need non-linearity because without 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 (implicit ouput)

Loss function and average loss

- machine learning is all about optimizing functions

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 dependent on gradient specially the ones 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

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

Backprop through a non-linear functions

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

~~Perbutations~~

Perturbations:

Linear blocks $s_{k+1} = a_k z_k$

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

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

John Vidal

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

Practical tips of Backprop

softmax is just a generalization of sigmoid with multiple outputs

- Use ReLU · (tanh and others are pd falling out of favor)
- Use cross entropy for classificatin
- Use sgd on minibatched
- 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 regularizatin on the weights (Weight Decay)
  → it's best to turn it on after couple epochs
- Use dropout for regularizatin (distribute info better
- Lecun → Effient Backprop (1998)
- Neural Nets, Tricks to trade
- ~~Weight Decay~~
- Any directed acyclic graph is ok for backprop

Practicum 2: Training a neural network

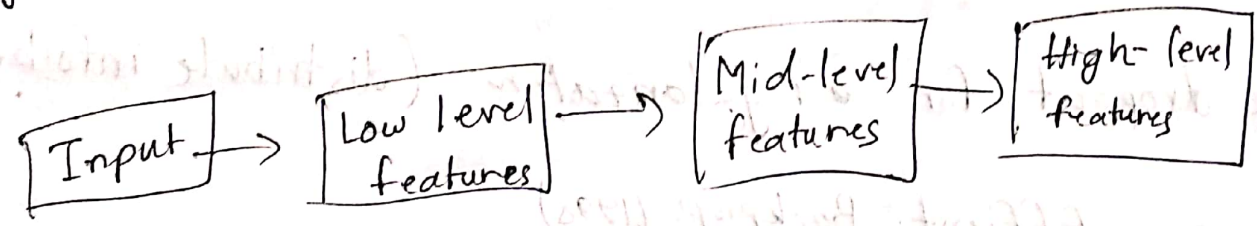
NN are mainly doing rotation and squashing

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



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

Traindata

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

$$X = \begin{bmatrix} -x^{(1)}- \\ -x^{(2)}- \\ \vdots \\ -x^{(m)}- \end{bmatrix} \quad (m \times n)$$

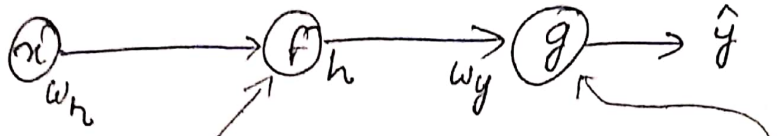
$$C = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} \quad (m \times 1)$$

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

1-hot encoding

$$Y = \begin{bmatrix} -y^{(1)}- \\ -y^{(2)}- \\ \vdots \\ -y^{(m)}- \end{bmatrix} \quad (m \times K)$$

Fully connected layer



$h = f(w_h x + b_h)$

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

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(·)

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

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

$$L(\hat{Y}, C) = \frac{1}{m} \sum_{i=1}^{m} l(\hat{y}_i, c_i)$$

$$l(\hat{y}, c) = -\log(\hat{y}[c])$$

↑ cross entropy / negative log likelihood

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

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

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

notebooks
- spiral classification
- regression

$\theta = \{w_h, b_h, w_y, b_y\}$

$J(\theta) = L(\hat{Y}(\theta), C) \in \mathbb{R}^+$

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

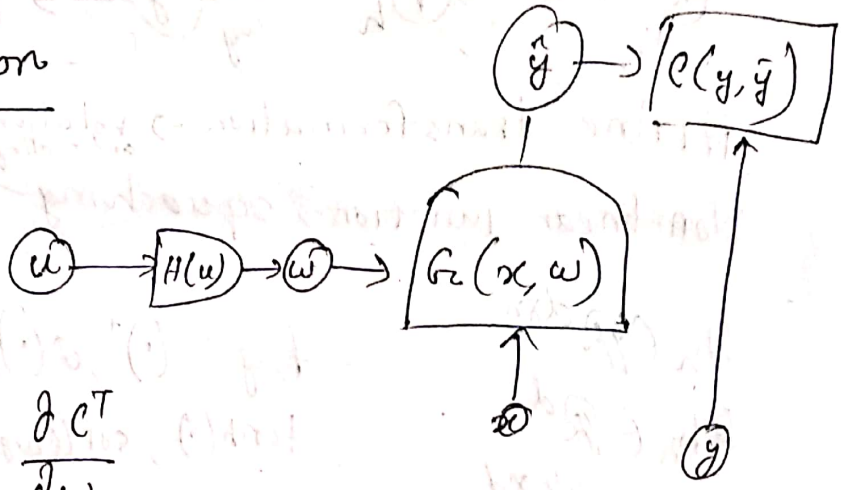
choice of activation will have effects on regression

Lecture 3: Convolution Neural Networks

Parameters transformation

$w = H(u)$

- parameter vector can be the output of a function

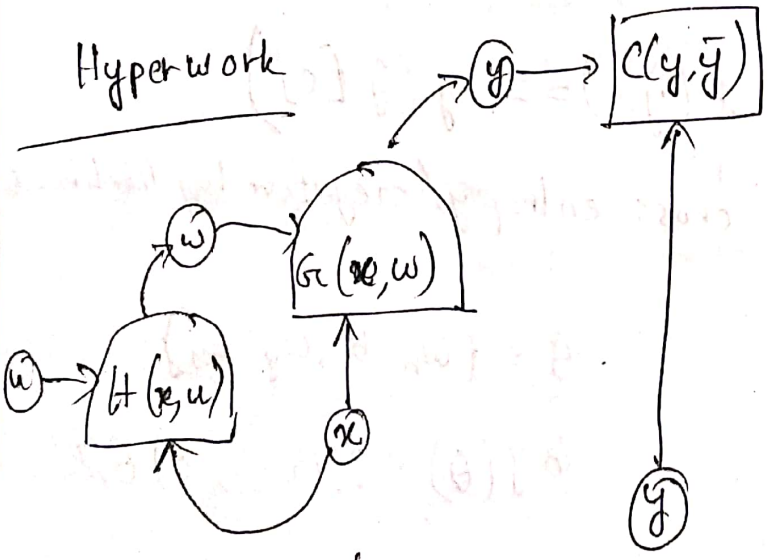


$$u \leftarrow u - \eta \frac{\partial H^T}{\partial u} \frac{\partial C^T}{\partial w}$$

$$w \leftarrow w - \eta \frac{\partial H}{\partial u} \frac{\partial H^T}{\partial u} \frac{\partial C^T}{\partial w}$$

$[N_w \times N_u] \quad [N_u \times N_w] \quad [N_w \times 1]$

Hyperwork



- when the parameter vector in the output of another network $H(x, u)$
- The weights of network $G(x, w)$ are dynamically configured by network $H(x, u)$

Shared weights

These can be necessary for motif detection anywhere on an input

Conv Basics

Stride, pad

$$y_i = \sum_j \omega_j x_{i-j} \quad \text{(conv)}$$

$$y_i = \sum_j \omega_j x_{i+j} \quad \text{(cross correlation)}$$

Deep learning = Learning Hierarchical Representations

Idea of convnets comes from biology.

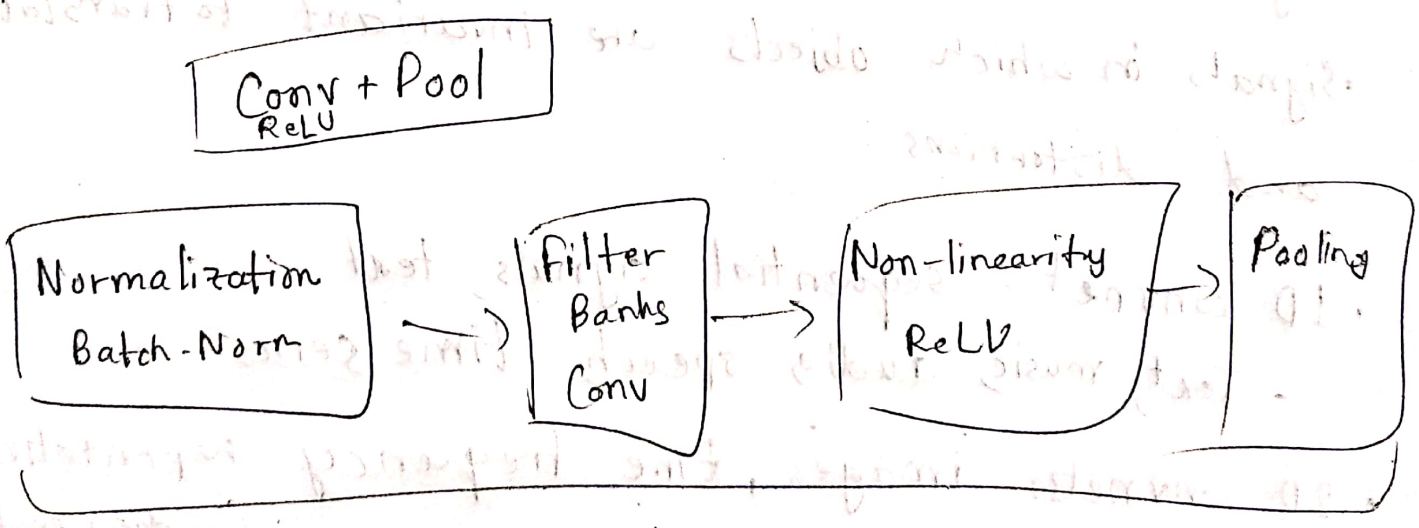
(How does the brain interprets images?)

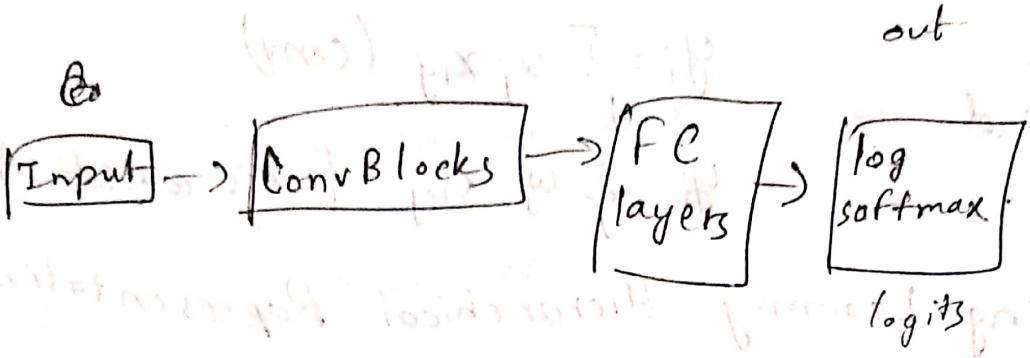
- Neuron firing

Hubel and Wiesel 1962

- simple cells detect local features
- complex cell 'pool' the output of simple

First 'Real' Convnets at Bell Labs [Lecun 1989] NIPS





LeNet (Character recog)

Multiple Character recognition [Matan 1992]

SDNN

features to get deefect something

Feature Binding and What convnets are good for?

- Signals that comes to you in the form of ndarray
- Signals that have strong local correlations
- Signals where features can appear anywhere
- Signals in which objects are invariant to translations and distortions

- 1D convnet: sequential signals, text
  - Text, music, audio, speech, time series
- 2D convnets: images, time frequency repentation (audio speech)
  - Object detection, localization, recognition
- 3D convnets: video, volumetric images, tomography images
  - Video recognition/understanding
  - Bio image analysis
  - Hyperspectral image analysis

Practicum 3 Natural signals properties and CNNs

Foundation of DL

CNN (exploiting stationarity, locality and compositionality of natural data)

$x = [x_1\ x_2\ x_3 \dots\dots x_t \dots]^T$ [Time series]

x_t are waveform heights

$x = [x_{11}\ x_{12}\ x \dots\dots x_{1n}\ x_{21}\ x_{22} \dots]$ [Image]

x_{ij} are pixel values

$x = [x_1\ x_2\ x_3 \dots\dots x_n]^T$ [Text]

x_t are one hot vectors

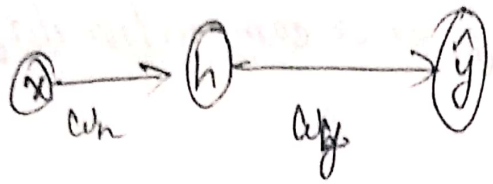
3 properties of natural data

Stationarity → we observe the same pattern over and over in our data

Locality → the patterns are localized

Compositionality → perceiving things with composing

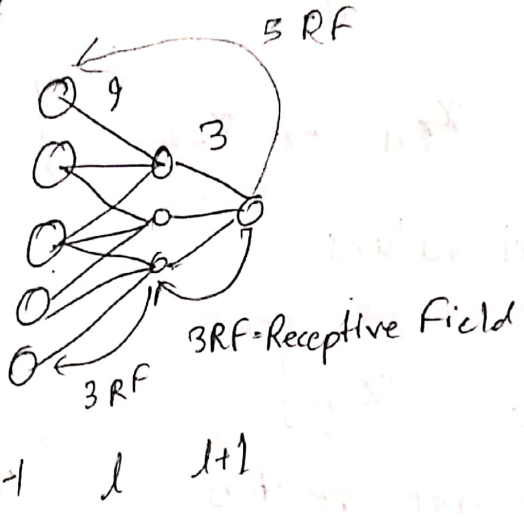
FC(Fully connected) Layers



$$h = f(W_h x + b_h)$$
$$\hat{y} = g(W_{\hat{y}} h + b_h)$$

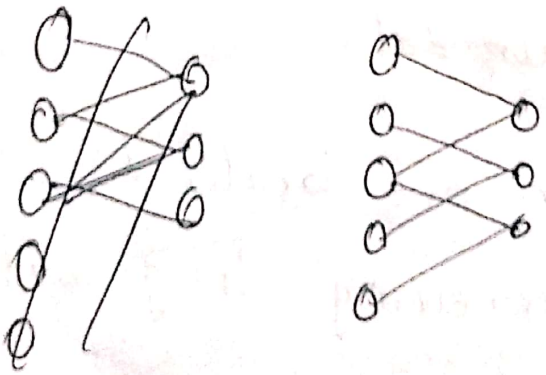
$$f, g = \theta(\cdot)^+, \sigma(\cdot)$$

Locality ⇒ sparsity



- reduced amount of composition

Stationary ⇒ parameters sharing



- faster convergence
- better generalization
- constrained to input size
- kernel independence ⇒ high parallelisation

Kernels - 1D data

Padding 1D data

Standard spatial CNN

- Multiple layers
  - Convolutions
  - Non-linearity (ReLU and Leaky)
  - Pooling (Geoffrey Hinton don't like this) He is working on it ~~to make this more~~ so that we don't need to throw away information
  - Batch normalisation
  - Residual bypass connection

If we have the 3 properties we can use CNN

Or else it's not appropriate

Practicum 4: Listening to convolution [1D convs]

Linear Algebra Review

~~$\underline{h} = f(\underline{z})$~~

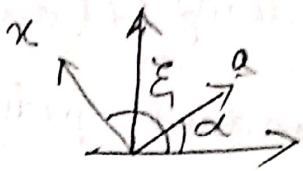
$\underline{h} = f(\underline{z})$

$$\underset{\in \mathbb{R}^m}{\underline{z}} = \underset{m \times n}{A} \underset{\in \mathbb{R}^n}{\underline{x}} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} =$$

$$= \begin{bmatrix} - & a^{(1)} & - \\ - & a^{(2)} & - \\ & \vdots & \\ - & a^{(m)} & - \end{bmatrix} \; | \; x$$

$$= \begin{bmatrix} a^{(1)} x \\ a^{(2)} x \\ \vdots \\ a^{(m)} x \end{bmatrix}$$

$$= \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} = z$$



$a^T x$ when $n=2$

$a^T x = a_1 x_1 + a_2 x_2$

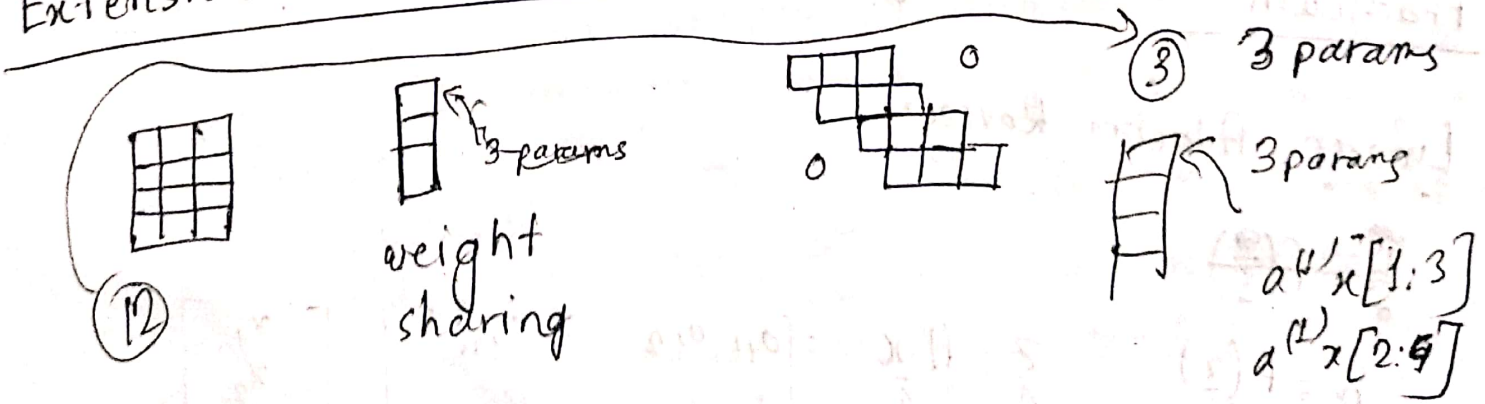
$= \|a\| \cos\alpha \; \|x\| \cos\xi + \|a\| \sin\alpha \; \|x\| \sin\xi$

$= \|a\| \|x\| (\cos\alpha \cos\xi + \sin\alpha \sin\xi)$

$= \|a\| \|x\| \cos(\xi - \alpha)$

$$z = a x_1 + a x_2 + \cdots + a x_n$$

Extension to convolutions:



Convolution is a matrix multiplication with a lot of zeros

Listening to convs

$\mathcal{X} = \{x^{(i)} \in \mathbb{R}^n \mid x^{(i)} \text{ is a data sample}\}_{i=1}^m$ input samples

$\mathcal{X} = \{x^{(i)} : \underset{\text{domain}}{\Omega} \to \underset{\text{channels}}{\mathbb{R}^c},\ \omega \to x^{(i)}(\omega)\}_{i=1}^m$

$\Omega = \{1, 2, \ldots, T/\Delta t\} \subset \mathbb{N}$ $\quad c \in \{1, 2, 5, 1, \ldots$
mono dolby 5.1

$\Omega = \{1, \ldots h\} \times \{1, \ldots w\} \in \mathbb{N}^2$ $\quad c = \{1, 3, 20, \ldots\}$
grey RGB hyperspectral

$\Omega = \mathbb{R}^4 \times \mathbb{R}^4$, $c = 1$ (Hamiltonian)
space-time four-momentum

Week 5

Lecture 4: Optimization

Aaron DeFazio (Facebook Research Scientist)

- Gradient deseent & SGD
- Momentum
- Diagonal rescaling: RMS Prop & Adam
- Normalization layers
- Solving optimization problems without optimizing
- Application: MRI reconstruction

Gradient descent:

minimize $f(\omega)$ over ω

$$\omega_{k+1} = \omega_k - \gamma_k \nabla f(\omega_k)$$

(γ_k: stepsize)

- follow direction of the negative gradient

"worst optimization algo"

- The negative gradient direction is the steepest direction locally

SGD:

$$w_{k+1} = w_k - \gamma_k \nabla f_i(w_k)$$

SGD: often the best method available

$$E\left|\nabla f_i(w_k)\right| = \nabla f(w_k)$$

SGD is GD with noise

Advantages of SGD:

- There is redundant info across instances the noise is lower when there is higher redundancy
- At early stages of optimization, the noise is small compared to the information in the gradient. so a SGD step is virtually as good as a GD step
- The noise can prevent the optimizing converging to bad local minima, a phenomena called annealing
- Stochastic gradients are drastically cheaper to compute (proportional to your dataset size), so you can often take thousands of SGD steps for the cost of one GD steps.

Mini-batching

$$\omega_{k+1} = \omega_k - \frac{\gamma}{k|B_i|} \sum_{j \in B_i} \nabla f_j(\omega_k)$$

- Done for ~~parallelizi~~ parrallelization in GPU

Theoricitically is not right

Empirically gives a good result

For Batch optimization, LBFGS & Conjugate Giradients are for better than vanilla GD.

Momentum:

$$\omega_{k+1} = \omega_k - \gamma_k \nabla f_i(\omega_k) + \beta(\omega_k - \omega_{k-1}) \quad \left[\text{SGD + Momentum}\right]$$

$$P_{k+1} = \hat{\beta}_k P_k + \nabla f_i(\omega_k)$$

$$\omega_{k+1} = \omega_k - \gamma_k P_{k+1}$$

- The next step becomes a combination of the previous steps direction and the new negative gradient

optimization process resembles a heavy ball rolling down a hill.

Larger momentum = slower reaction to change in the landshape

$\beta = 0.9$ or 0.99 almost always works well.

- step size need to be decreased when the momentum parameter is increased to maintain convergence

why momentum works?

Explanation one
Acceleration

Nesterov's momentum

$$v_{k+1} = \beta_k v_k + \nabla f_i(w_k)$$
$$w_{k+1} = w_k - \gamma_k (\nabla f_i(w_k) + \beta_k v_{k+1})$$

when using carefully chosen constants, probably "accelerates" convergence on problems with simple structure

torch.optim.SGD(..., nesterov=True)

Explanation two: Noise Smoothing

$$\bar{w}_k = \frac{1}{k} \sum_{k=1}^{k} w_k$$

- smoothens the noise from SGD

Adaptive methods

The magnitude of the gradients often varies highly between layers due to, so a global learning rate may not work well.

- Instead of using the same learning rate for every weight in our network, maintain an estimate of a better rate seperately for each weight.
- most methods either adapt to the variance of the weights, or to the local curvature of the problem

RMSProp (Root Mean Square Propagation)

$$V_{t+1} = \alpha V_t + (1-\alpha) f_i(\omega_t)^2$$

$$\omega_{t+1} = \omega_t - \gamma \frac{\Delta f_i(\omega_t)}{\sqrt{V_{t+1}} + \epsilon}$$

- If the expected value of gradient is small, this is similar to dividing by the standard deviation

Adam (Adaptive moment estimation)

RMSProp with a kind of momentum

momentum (exponentially moving average)

$$m_{t+1} = \beta v_t + (1-\beta)\nabla f_i(\omega_t)$$

same as RMSProp

$$v_{t+1} = \alpha v_t + (1-\alpha)\nabla f_i(\omega_t)^2$$

$$\omega_{t+1} = \omega_t - r\frac{m_t}{\sqrt{v_{t+1}} + \epsilon}$$

using m instead of gradient

Recommended SGD + Momentum Adam

- The exponentially-moving-average method of updating momentum is equivalent to the standard form under rescaling

→ Full version of Adam has bias-correction, which just keeps the moving averages unbiased during early iterations

Disadvantages of Adam

i) doesn't converge at all on some simple example problems

ii) Gives worse generalization on some computer vision problems

iii) Requires more memory than SGD

iv) Need to tune both momentum params

Normalization layers

learnable scalable factor — input

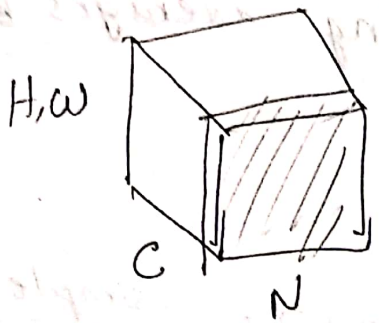
$$y = \frac{\alpha}{\sigma}(x - \mu) + b$$

learnable bias term

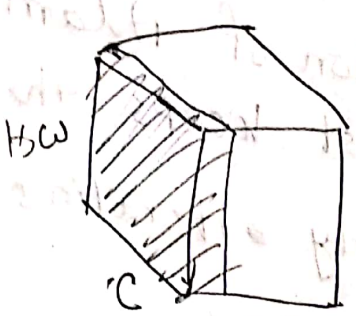
Most norm layers whitens activations

Batch Norm

In practice, group/batch norm work well for computer vision problems and instance/layer norm are heavily used for language problems



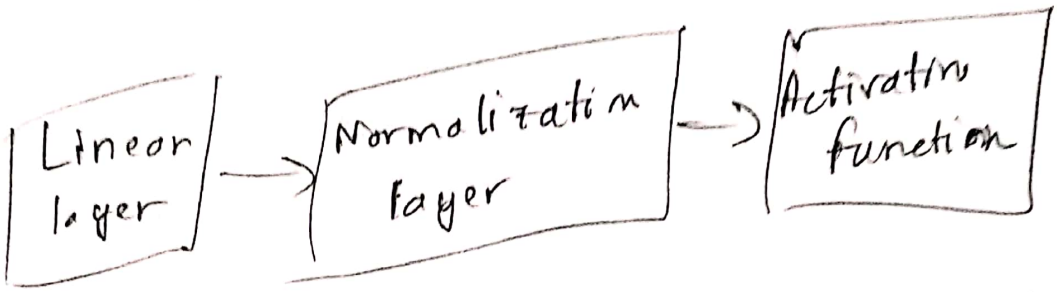
Batch Norm



Layer Norm

- Networks with normalization layers are easier to optimize, allowing for the use of larger lr
- extra l "noise" results in better generalizatio
- Reduce sensitivity to weight initialization

Group Norm is Recommended since it's more stable theoritically simple and asually works better



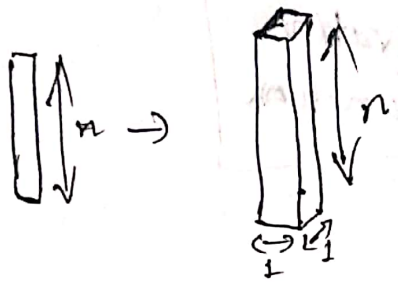
The Death of Optimization:

MRI reconstruction

MRI machines ~~acbie. eq~~ acquire data one row or column at time of the fourier domain representation. The inverse fourier transform is applied at the end to get the anatomical image.

Instead of doing optimization while processing a big neural network can be used to automate this. which gives a much ~~mi big~~ better and fine-grained result

Practicum 5: 1D multi-channel convolution and autograd



$x : \Omega \to \mathbb{R}^c$

$\{0, 1, 2, \ldots\} \subseteq \mathbb{N}$

stereophonic signal → 2 channels

- Mostly done in ~~cuda~~ code

a. backprop() in pytorch accumulates all the gradients backwards wrt params

With Great flexibility Comes Great Responsibility

with torch.no_grad()

purge [no tracing]

Not possible with tensorflow

Week 6

Lecture 5

CNN application, RNN and attention

No maximum suppression → ~~one~~ highest score is the result

⇒ Word-level training with weak supervision [Matan 1992]

⇒ Multiple Character Recognition [Matan et. al 1992]

Sliding Window ConvNet + Weighted finite state Machine

Face Detection [Vailiant et. al 93, 94]

- ConvNet applied to large images
- Heatmaps at multiple scales
- Non-maximum suppression for candidates
- 6 second on sparcstation for 256x256

Semantic Segmentation

[ConvNet for Long Range Adaptive Robot Vision (DARPA LAGR Program 2005-2008]

~~Fine~~ To find out the regions where the robot can drive (Stereo Labels)

Robot adapted the last layers as it moved on. N-1 layers were frozen and 1 or last layer was being trained constantly

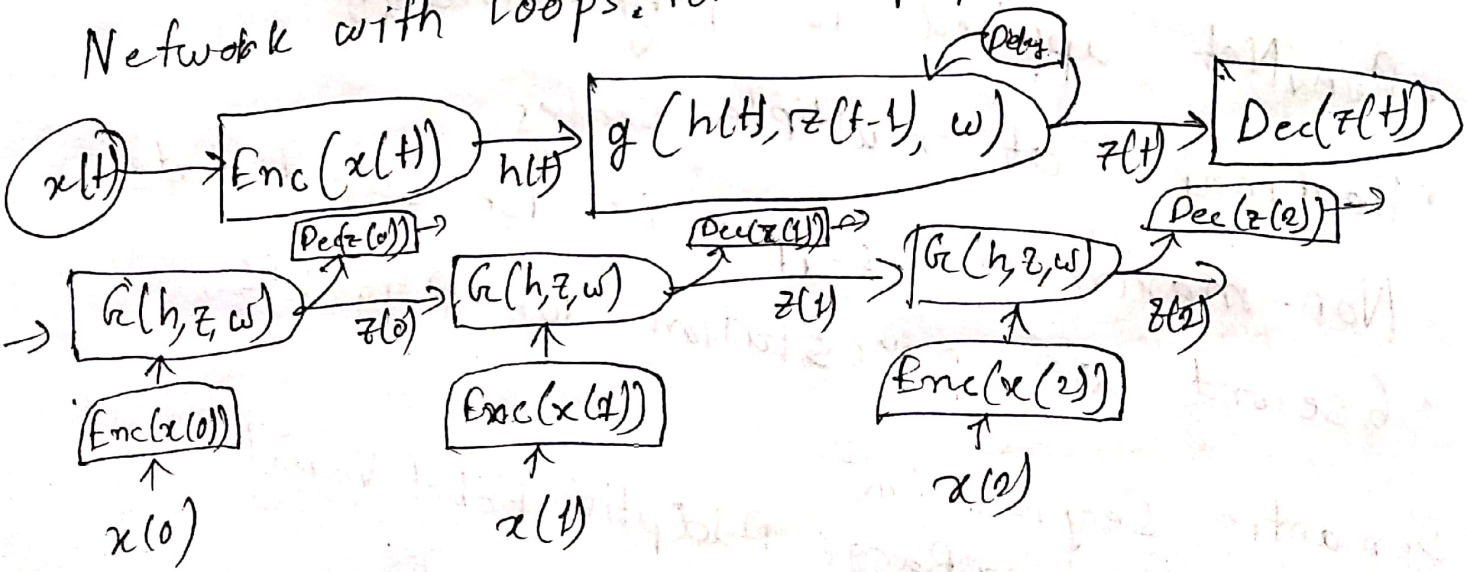
lagr@net-scale.com

Scene Parsing / Labeling: ~~Fabet~~ ~~Fab~~ [Fabebet et al ICML 2013, PAMI-2013]

Multiscale Convnet architecture

RNN Architecture and Attention:

Network with loops. for backprop, unroll the loop



BPTT → Backprop Through Time

- Vanishing and exploding ~~grad~~ gradient problem

RNN tricks [Pascanu ICML 2013, Bengio ICASSP2013]

- Clipping gradients [avoids exploding gd gradients]
- Leaky integration (propagate long term dependencies)
- Momentum (cheap 2nd order)
- Initialization (start * in right ballpark avoids exploding/vanishing)
- Sparse Gradients (symmetry breaking)
- Gradient propagation regularizer (avoid gd vanishing gradient)
- LSTM & self-loops (avoid vanishing gradient)

Attention module

$$s_i = \sum_j \omega_j x_{ij} \qquad \omega_j = \frac{e^{z_j}}{\sum_k e^k}$$

GRUs, LSTMs, Seq2Seq

↓

[Hochreiter & Schimidhuber 97]

Seq2Seq

Sequence encoder → sequence decoder

Applications

→ Translation [Sutskever NIPS 2014]

→ Multilayer LSTM

Translation [Bahdanau, Cho, Bengio]

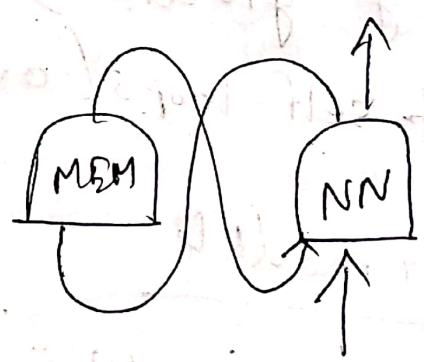
Memory Networks (~~Story~~ Storing Memory)

Key and values

$$a_i = k_i^T x$$

$$c = \text{softmax}(a)$$

$$s = \sum_i c_i v_i$$



Neural Turing Machine

→ Differentiable Computer