Convolutional Network

Convolutions in 1D:

→ Impulse response

→ Multiple Kronecker Delta

→ Gaussian Averaging

Linear Local Agg.

$$f(x_i) = a x_{i-1} + b x_i + c x_{i+1}$$

Activation Functions

$y = \tanh(x)$

$y = \text{sigmoid}(x)$

$y = \text{ReLU}(x)$ → Very Fast → Gradient → Piecewise linear interpolation

Max vs Avg Pooling:

Max pooling is more sensitive to noise

Why MLE won't work?

→ Large dimensions

→ No. of parameters goes to huge range

→ Not easy to learn invariance.

Let's assume input is $W_1 \times H_1 \times C$

Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

This will produce an output of $W_2 \times H_2 \times K$

where:

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

K = (powers of 2, e.g. 32, 64, 1

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatev
- F = 1, S = 1, P = 0

CS = SC

Correlation vs Convolution

↘ Circulant Matrix

↘ Shift Invariant

Inputs & Outputs

INPUT: $B \times C_{in} \times H_{in} \times W_{in}$

↓ Samples ↘ Multi-channel processing

CONVOLUTION LAYER: $C_{out} \times C_{in} \times K_H \times K_W$

↓ Bias Parameters

OUTPUT: $B \times C_{out} \times H_{out} \times W_{out}$

C_{out} filters of shape $C_{in} \times K_H \times K_W$

Loss Functions:

→ $L = \|y - p\|^2$ → L_2 loss

→ Surface is convex

→ MLE:

↳ log likelihood: $\log(p(Y|x,\theta))$

$$L = \log p(Y|x,\theta) = \sum_i \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(-\frac{(y_i - x_i w)^2}{2\sigma^2}\right)}\right)$$

$$\max_w L = \max_w \sum_i \log c - \sum_i (y_i - x_i w)^2$$

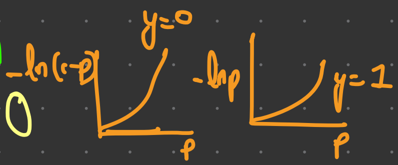
$$= \min_w \sum_i \|y_i - x_i w\|^2$$

↳ Same as MSE

→ Residual connections solve problem of vanishing gradients → Adds info

→ Parameter initialisation

→ Weights shouldn't be 0



→ 1×1 Convolution

INP: $B \times C_{in} \times H \times W$

CON: $C_{out} \times C_{in} \times 1 \times 1$

Converts C_{in} is converted to C_{out}

↳ MLP in spatial dimensions

→ Normalisation

$$x_j = \frac{x_i - \mu_i}{\sigma_j}$$

→ Batch norm:

① Normalise → Scale, Shift

② Moving Average

→ Spatial Pooling

↳ Provides invariance

Binary Cross Entropy:

→ $H = -p \log p$ ⇒ entropy

→ $\mathcal{L}_{BCE} = -[y \ln p + (1-y)\ln(1-p)]$

↳ $\{0,1\}$ $p \in [0,1]$

When extended to K-classes:

$$\mathcal{L}_{CE} = -\sum_k y_k \log(p_k)$$

① CE maximises log-likelihood

② maximise log-likelihood == minimise -ve log-likelihood

K classes: SOFTMAX

$$\text{sigmoid}(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{1+e^z}$$

$z \in \mathbb{R}^K$, $p = \text{softmax}(z)$

$$p_i = \frac{e^{z_i}}{\sum_j e^{z_j}}, \quad p_i \in [0,1], \quad \sum_i p_i = 1$$

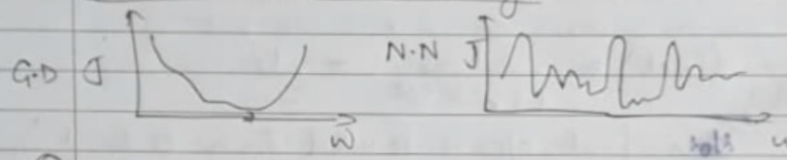
Examples:

① AlexNet ② VGG-Net

③ GoogLeNet ④ ResNet

6.8 of DHS book **Lecture - 15 : tips for Backprop**

* We need it to work effectively
* Due to complex fn in neural net. Chance of getting stuck in local minima high



① **Activation function**

Desirable Properties

to update weights we require $f'(net)$ ∴ we need this condn

a) continuity of $f(\cdot)$, $f'(\cdot)$

some neurons shouldn't explode ← b) $f(\cdot)$ should saturate (max & min)

↳ to keep weights & activations in check

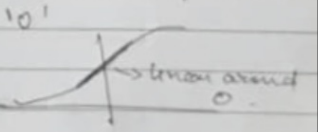
↳ Also to interpret o/p of neuron as probability

c) $f(\cdot)$ monotonic

for J (error) fn to be simple f must be simple

d) $f(\cdot)$: linear around '0'

Sigmoid satisfy all these



② **Parameters of Activation fn** (Assume sigmoid is used)

$$f(net) = \frac{1}{1 + e^{-net}} \quad \} \text{ basic sigmoid}$$

We want some extra properties like the range of linearity & which part non-linearity lies

$$f(net) = \frac{1}{1+e^{-net}} \Rightarrow \frac{1}{1+e^{-b\,net}}, \; b \} \text{ control range of linearity}$$

ideally weights should be +ve & negative ∴ o/p going from -ve to +ve is desirable

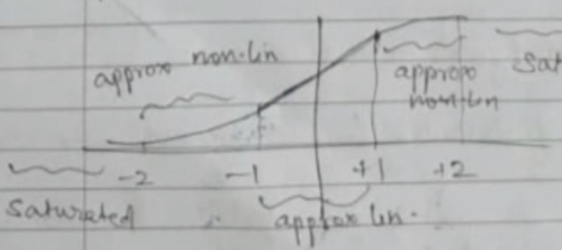
$$\therefore f(net) = \frac{1}{1+e^{-b \cdot net}} - \frac{1}{2}$$

(Note: final o/p is wanted from 0 to 1 so maybe we won't do it for o/p)

$$\boxed{f(net) = 2a\left(\frac{1}{1+e^{-b\,net}} - \frac{1}{2}\right)}$$

multiplying this so that range changes from $[-1/2, 1/2] \to [-a, a]$

$$\boxed{f(net) = \frac{2a}{1+e^{-b\,net}} - a}$$



} our goal ranges of lin & non-lin

$\therefore$ $a = 1.716$, $b = 2/3$] gives desirable props

max/min $(f''(net)) = +2/-2$
max/min $(f(net)) = +a/-a$
$f()$ ⇒ also antisymmetric ☆

$$f(-x) = -f(x)$$

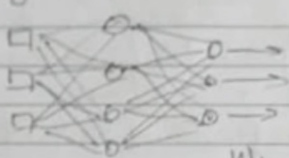
(3) input values: if too large ⇒ saturate to $-a/+a$
if too small ⇒ only non linearity becomes linear
$\therefore$ power of N.N lost. back to lin. perceptron

$\therefore$ shouldn't be too large / too small
☆ $\therefore$ Normalise : 0 mean, unit variance
. ! $\sim -2, +2$ range

(4) Weights Initialization: should be in approp range
$\sum_{j=1}^{d} w_j y_j$ } net should be in proper range

Note: don't set all weights to 0 (No learning happens, every eqn goes to 0)
So choose randomly
$\therefore$ $W = U[-\tilde{w}, \tilde{w}]$ } uniform R.V
☆ $\tilde{w} = 1/\sqrt{d}$

(5) Target Value



lets assume w_1 correspond to class 1
how is it encoded? w_1 → High w_2, w_3 → Low
→ $w_1, w_2, w_3 = a, -a, -a$. lets assume
in an attempt to make w_1 go to a
& w_2 & w_3 go to $-a$. the net value at w_1 goes to ∞, and at w_2, w_3 go to $-\infty$

$\therefore$ weights explode to push outputs to target values at saturation

☆ $\therefore$ put target value less than saturation
eg:- $a = 1.716$ then put target at 1.5 or 1
both high & low should change
(ie high = 1.5 low = -1.5)

(6) Training with Noise # training samples small
→ add noise into data as training samples
→ helps regularising
→ if test data has noise, this helps
→ increases training set size, → add to ftr vector

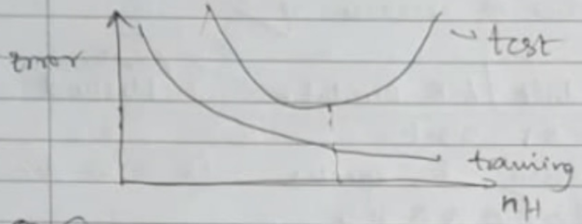
Data Augmentation

(7) Manufacture new Data
→ rotate, image process, so on.
→ given one example get variants of sample
→ add as training data

⑧ # of hidden ~~layer~~ units $\to n_H$ (Assume 3 layers)

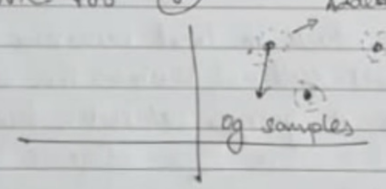
i/p layer — hidden layer — o/p layer
$d+1$ — n_H — # of classes

trial & error method.



$n_H \uparrow \Rightarrow$ overfit

(Note: for ⑥



)

⑨ Learning rate — too small ⇒ slow
too large ⇒ jump around. No learn

$\eta_{opt} \to$ optimal η
(Newton gradient descent method can be used (approx fn in small region & take appropriate jump))

$\eta_{opt} = \left(\frac{\partial^2 J}{\partial w^2}\right)^{-1}$ } Computing is impratical ☆
⇓
faster to do G.D

$\eta_{max} = 2 \cdot \eta_{opt}$

1 step N.D ≈ 10 step G.D in time taken
also while 1 step N.D > better than 1 step G.D
10 step G.D better than 1 step N.D

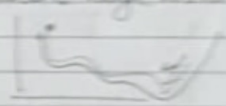
I method { ∴ start with $\eta = 0.1$
J diverges ⇒ reduce η
J is slow ⇒ increase η } ☆

η schedule / L.R schedule } Method

$\eta = 0.1$ start
☆ after every k epochs ($k \to$ we decide)
$\eta = 0.9 \times \eta$

⑩ Don't get stuck in local minimum
think of ball rolling down → has velocity which keeps it going until huge minimum

∴ Momentum



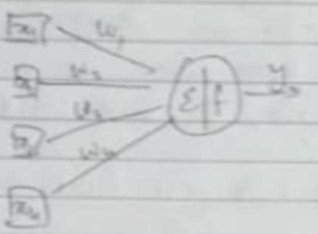
(Oscillation possible)
$\Delta w_m \Rightarrow (1-\alpha)\Delta w_m + \alpha \Delta w_{m-1}$
lin combination of prev & new direction
(don't stop even if $\Delta w_m = 0$, continue in direction of Δw_{m-1})

⑪ Weight Decay — $\varepsilon \propto 0.99, 0.95, 0.9$

$w \leftarrow w(1-\varepsilon)$ after every update
all numbers moved toward 0

→ let's say w_1 is important ~~w_1 says pu~~
G.D says push w_1 up } G.D wins & takes over
Weight decay says push it down }

→ lets say w_2 is not relevant
G.D doesn't push anywhere } W. Decay wins
weight decay push to 0 }



∴ W. Decay makes all useless weights → 0

Useful weights determined by G.D.

☆ Also complexity of fn reduces as unwanted things go into linear region ⇒ simplification

equivalent to $\boxed{J_{wd} = J_{original} + \frac{2\varepsilon}{\eta} \cdot w^t w}$

new loss fn — weight decay loss fn

adding $w^t w$ to J (J has to be minimized (known)
→ we want to reduce values inside w
ie size of w (minimizing $w^t \cdot w$)

∴ this simple update eqv. to adding complex term to J

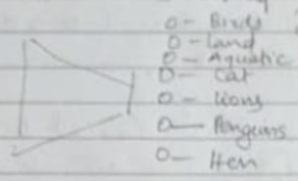
kept weights in check

$\boxed{\text{Note: } \frac{2\varepsilon}{\eta} w^t \cdot w \text{ called regularizer}}$

(12) Use of Hints

lets say we want to classify animals -
add extra nodes corresponding to animals

eg:-



} Additional outputs so N.N can learn that cat is land animal

when cat is high, so will land output

☆ Adding extra outputs to give hints to neural networks

We chop off extra o/ps at the end ☆

Unlike normal: more than ~~one~~ one output high

(13) Loss Fn / criterion Fn

a) Square Error $J(w) = \sum_k (t_k - z_k)^2$ (target, orig)

b) Minkowski Error:
$J(w) = \sum_k (|t_k - z_k|)^R$

simplified version

c) Cross Entropy
$J(w) = \sum_{k=1}^{c} t_k \ln(t_k / z_k)$

(14) # of Hidden Layers
hidden layer > 1 ⇒ Deep Network ☆