Architectural Elements of Neural Networks

Neural Networks in Context of Classification

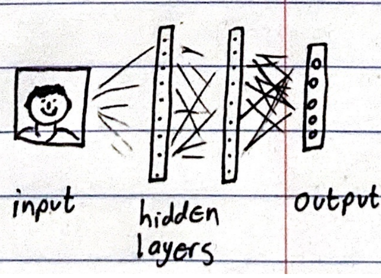
Problem: Given an image of an object, determine what category that object is (Eg cat, dog)

A neural network is a function that maps an image to a probability distribution over a known set of classes/categories. The prediction for a given image is category w/ highest probability.

A neural network is built from components:

Linear	Nonlinear
Fully connected	Activation functions
Convolution	Batch Normalization
Linear upsampling/downsampling	Max pooling
	Softmax

Visually:



Parameters are learned from data.

Logistic Function and Softmax

$$\text{logistic}(z) = \frac{e^z}{e^z+1} = \frac{1}{1+e^{-z}} \quad \text{for } z \in \mathbb{R}$$

Softmax : $\mathbb{R}^N \to \mathbb{R}^N$

$$\{z_i\}_{i=1\cdots N} \mapsto \left\{ \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \right\}_{j=1\cdots N}$$

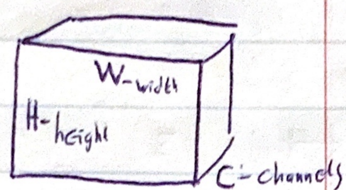
Output of Softmax is a prob. dist.
(could call it soft arg max)

Note: If a net computes logits,
Optimization is not expected to converge.
eg linearly separable binary classification
w/ logistic regression

Application: Even after 100% training accuracy
is achieved, optimization continues
(and test error does too)

Channels and Batching for images

An image is a ~~m~~ 3-dim matrix "tensor"



Grayscale - $C = 1$
RGB - $C = 3$
At internal layers of a NN C could take other values

A minibatch of images is a collection of N images
a 4-dim tensor

Pytorch [N, C, H, W]
TensorFlow [N, H, W, C]

Convolutional Layers

Mathematically: $S = X * W$

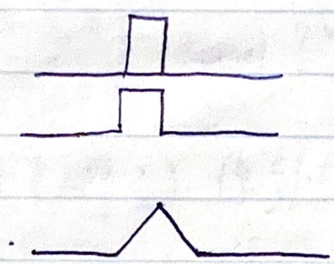
$$S(t) = \int X(a)\,W(t-a)\,da \qquad \text{"convolution"}$$

inner product of X with shifted and flipped W.

eg $W =$

$X =$

$X * W =$



ML terminology:
- X – input
- W – kernel, filter
- S – feature map, activation map

Discrete 1-d convolution:

$$S(i) = X * W(i) = \sum_m X(m)\,W(i-m)$$

Discrete 2-d convolution

$$S(i,j) = X * W(i,j) = \sum_m \sum_n X(m,n)\,W(i-m, j-n)$$

$$= \sum_m \sum_n X(i-m, j-n)\,W(m,n)$$

Instead of convolution (strictly speaking) ML libraries implement cross-correlation (dont flip the kernel)

$$S(i,j) = X * W(i,j) = \sum_m \sum_n X(i+m, j+n)\, W(m,n)$$

Visually: Input $\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix}$ Kernel $\begin{pmatrix} w & x \\ y & z \end{pmatrix}$

Convolution $\begin{pmatrix} \begin{pmatrix} a & b \\ e & f \end{pmatrix} \cdot \begin{pmatrix} w & x \\ y & z \end{pmatrix} & \begin{pmatrix} b & c \\ f & g \end{pmatrix} \cdot \begin{pmatrix} w & x \\ y & z \end{pmatrix} & \begin{pmatrix} c & d \\ g & h \end{pmatrix} \cdot \begin{pmatrix} w & x \\ y & z \end{pmatrix} \\ \begin{pmatrix} e & f \\ i & j \end{pmatrix} \cdot \begin{pmatrix} w & x \\ y & z \end{pmatrix} & \begin{pmatrix} f & g \\ j & k \end{pmatrix} \cdot \begin{pmatrix} w & x \\ y & z \end{pmatrix} & \begin{pmatrix} g & h \\ k & l \end{pmatrix} \cdot \begin{pmatrix} w & x \\ y & z \end{pmatrix} \end{pmatrix}$

w/ $\begin{pmatrix} a & b \\ e & f \end{pmatrix} \cdot \begin{pmatrix} w & x \\ y & z \end{pmatrix} = aw + bx + ey + fz$

In 1-d, convolution corresponds to multiplication by a Toeplitz matrix

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} * \begin{pmatrix} w \\ x \end{pmatrix} = \begin{pmatrix} aw + bx \\ bw + cx \\ cw + dx \end{pmatrix}$$

$$= \begin{pmatrix} w & x & & \\ & w & x & \\ & & w & x \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$$

Toeplitz matrix $\begin{pmatrix} a & b & c & d \\ e & a & b & c \\ f & e & a & b \\ g & f & e & a \end{pmatrix}$

What about boundaries?

Choices include:

- only include windows entirely contained in input (dim decreases)
- pad w/ zeros to keep image same size as feature map
- circular
- etc

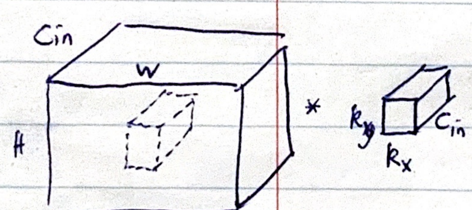
Conv2d operation in Pytorch

$[N, C_{in}, H, W] \rightarrow [N, C_{out}, H_{out}, W_{out}]$

Equal: images treated in parallel

H_{out}, W_{out}: these can differ from H, W due to padding, stride, etc

Each kernel is a 3-tensor (has



→ one channel of output

If filter has size $k_x \times k_y$, and no padding, how many parameters are in this convolution? What is output size

Ignore bias terms

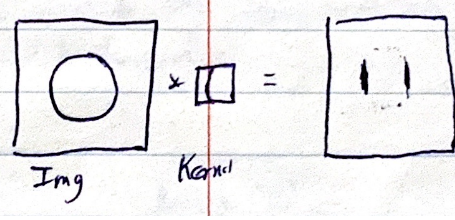
Output $N \times C_{out} \times (W - k_x + 1) \times (H - k_y + 1)$

params $C_{out} \cdot k_x \cdot k_y \cdot C_{in}$

Strength's of Convolutional Layers

- Fewer params than FC layers
  - cheaper
  - Easier to optimize
- Equivariance to translation
  - features of image in top left should be treated same as same in bottom right

Visually: Conv layers are "feature detectors"



"Edge detector"

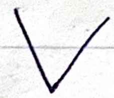
Activation functions

Rectified Linear Unit (ReLU)

$$\sigma(z) = \max(0, z)$$

Absolute value rectification

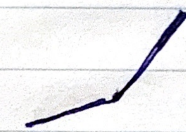
$$\sigma(z) = |z|$$



used sometimes in object recognition in images
might want invariance to image reversal

Leaky ReLU

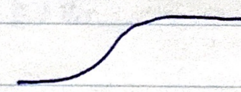
$$\sigma(z) = \alpha_i \min(0, z_i) + \max(0, z_i)$$



α_i could be set to a fixed value
could be learned

Sigmoid

$$\sigma(z) = \tanh(z) \text{ or } \mathrm{logistic}(z)$$



- Not recommended for internal/hidden layers due to saturation
- As last layer, ok as sat. can be compensated by loss function

Batch Normalization

Technique for improving speed, reliability, and perf of optimization of NN's.

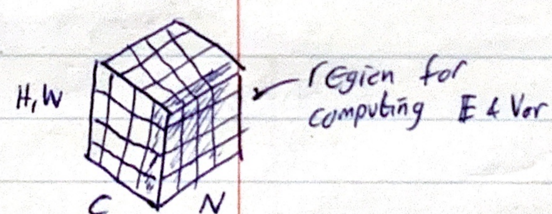
For input x

computed from input

$$y = \frac{x - \mathbb{E}[x]}{\underbrace{\sqrt{Var(x) + \epsilon}}_{\text{whitens the input}}} \cdot \gamma + \beta$$

computed from input   fixed, small   learned

Typically: channels treated separately
$\mathbb{E}$ & Var computed over (N, H, W) slices



Normalizes across images in batch and across ~~positions~~ pixels in img

Note: Equivariance of convolution layer motivates motivates computn stats across locations

Why does BN work?

Initial explanation

Internal Covariate shift

"the change in distribution of network activations due to the change in net params during training"

Idea: Modifications in an early layer of net could cause a large change downstream. BN would decouple effects more.

Explanation is disputed

Other explanation

Smoothing objective