Ch 10.3: Convolutional Neural Nets

Lecture 22 - CMSE 381

Michigan State University
::
Dept of Computational Mathematics, Science & Engineering

Monday, April 15, 2024

Announcements

Last time:

- Multilayer
- pyTorch

This lecture:

- CNNs

Announcements:

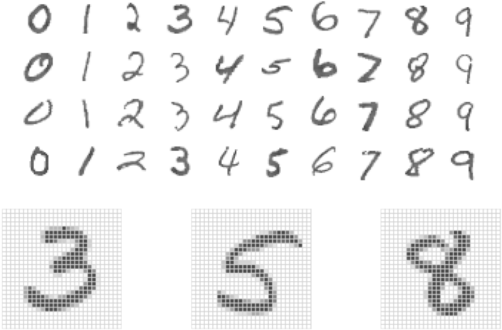
- last and this week's in-class assignments both due this week
- Grades conversion

Percent	Convert
≥ 90%	4.0
≥ 85%	3.5
≥ 80%	3
≥ 75%	2.5
≥ 70%	2
≥ 65%	1.5
≥ 60%	1
< 60%	0

Section 1

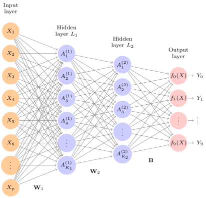
Last time: Neural Nets

MNIST



- Goal: Build a model to classify images into their correct digit class
- Each image has $p = 28 \cdot 28 = 784$ pixels
- Each pixel is grayscale value in [0,255]
- Data converted into column order
- Output represented by one-hot vector $Y = (Y_0, Y_1, \cdots, Y_9)$
- 60K training images, 10K test images

Neural network architecture for MNIST



$$
\begin{aligned}
A_k^{(1)} &= h_k^{(1)}(X) \\
&= g(w_{k0}^{(1)} + \textstyle\sum_{j=1}^{p} w_{kj}^{(1)} X_j)
\end{aligned}
$$

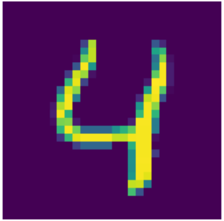
$$
\begin{aligned}
A_\ell^{(2)} &= h_\ell^{(2)}(X) \\
&= g(w_{\ell 0}^{(2)} + \textstyle\sum_{k=1}^{K_1} w_{\ell k}^{(2)} A_k^{(1)})
\end{aligned}
$$

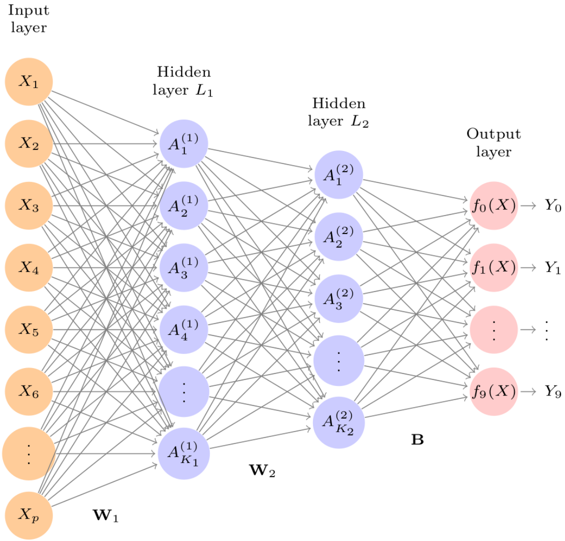
$$
\begin{aligned}
Z_m &= \beta_{m0} + \textstyle\sum_{\ell=1}^{K_2} \beta_{m\ell} h_\ell^{(2)}(X) \\
&= \beta_{m0} + \textstyle\sum_{\ell=1}^{K_2} \beta_{m\ell} A_\ell^{(2)},
\end{aligned}
$$

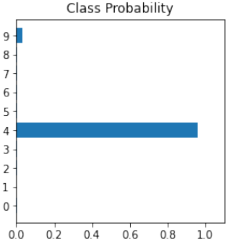
$$f_m(X) = \Pr(Y = m|X) = \frac{e^{Z_m}}{\sum_{\ell=0}^{9} e^{Z_\ell}},$$

- Two hidden layers.
- Softmax for classification output
- In lab, we used L_1 has 128 units; L_2 has 64
- 10 output variables due to class labeling
- Result is we are training approx 110K weights

MNIST learning







Section 2

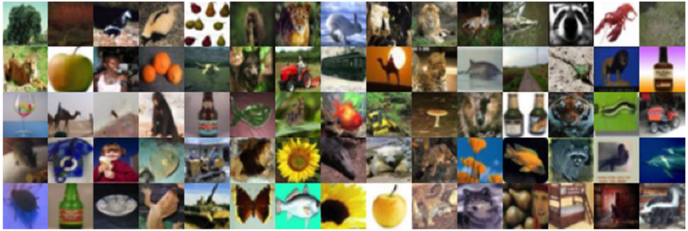
Convolutional Neural Network

When dealing with images with NN: Flattening the image

$$\begin{pmatrix} 1 & 1 & 0 \\ 4 & 2 & 1 \\ 0 & 2 & 1 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 \\ 1 \\ 0 \\ 4 \\ 2 \\ 1 \\ 0 \\ 2 \\ 1 \end{pmatrix}$$

- Just ignore the fact that we have a picture
- Loses a lot of the structure that we might want for more complicated
- Answer is CNNs, which take in image data and build a neural net that specifically tracks that structure

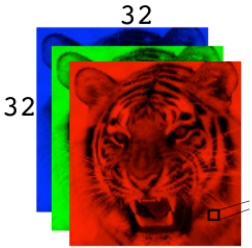
Example data set: CIFAR100 Data



- 60,000 images: 50K training, 10K test
- Labels with 20 super classes (e.g. aquatic mammals)
- 5 classes per super class (beaver, dolphin, otter, seal, whale)
- Images are 32x32

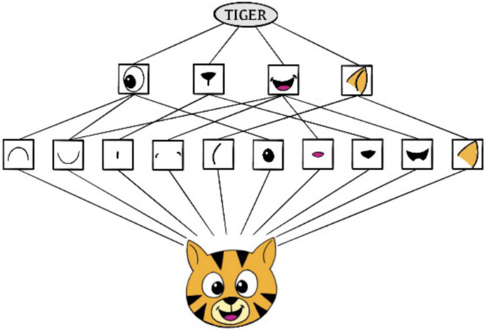
Image channel data

- 3 numbers per pixel (red, green, blue)
- Numbers are organized in a *feature map* , which is 3 dimensional array
- First two directions are spatial (32x32)
- Third is the *channel* axis representing teh three colors



CNNs

- Idea is to build a neural net that works on this data
- Identify low level features on the input image such as small edges, patches of color, etc
- Low level features combined into higher-level features (parts of ears, eyes, etc)
- Does this using special types of hidden layers: *convolution* layers and *pooling layers*



1D convolution

Definition. Let's start with 1D convolution (a 1D "image," is also known as a signal, and can be represented by a regular 1D vector in Matlab). Let's call our input vector f and our kernel g, and say that f has length n, and g has length m. The convolution $f * g$ of f and g is defined as:

$$(f * g)(i) = \sum_{j=1}^{m} g(j) \cdot f(i - j + m/2)$$

$$f = \boxed{10 \mid 50 \mid 60 \mid 10 \mid 20 \mid 40 \mid 30}$$

$$g = \boxed{1/3 \mid 1/3 \mid 1/3}$$

$$h = \boxed{20 \mid 40 \mid 40 \mid 30 \mid 20 \mid 30 \mid 23.333}$$

What is the output if using the kernel $g = [0.2, 0.6, 0.2]$ instead? What about $g = [0, 1, 0]$

Original Image:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{bmatrix}$$

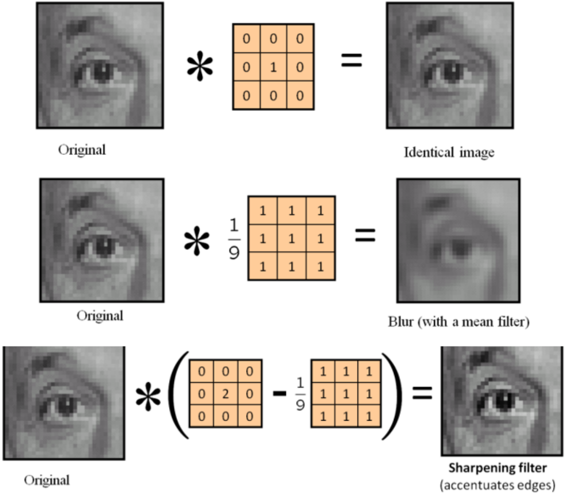
Convolution filter:

$$\begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}$$

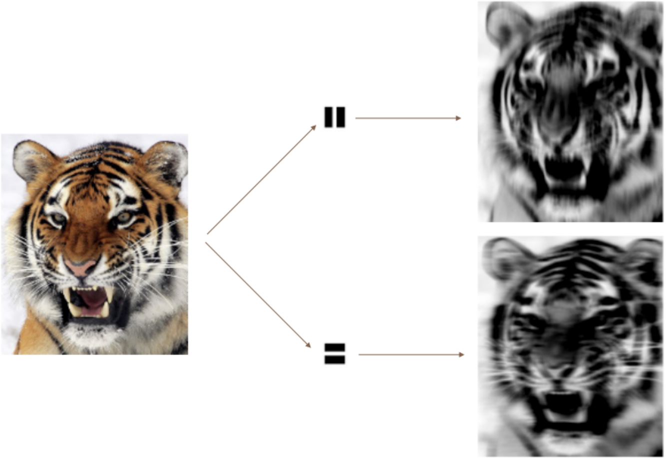
Convolved Image

$$\begin{bmatrix} a\alpha + b\beta + d\gamma + e\delta & b\alpha + c\beta + e\gamma + f\delta \\ d\alpha + e\beta + g\gamma + h\delta & e\alpha + f\beta + h\gamma + i\delta \\ g\alpha + h\beta + j\gamma + k\delta & h\alpha + i\beta + k\gamma + l\delta \end{bmatrix}$$

Convolution Filter Example

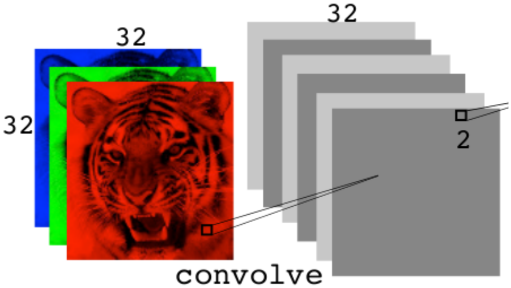


Convolution filter: Bigger example



- 192x179 image of tiger
- Convolution filters are 15x15 images with mostly zero (black) narrow strip of ones (white)
- Highlights areas of the tiger that resemble the filter

Convolution layer -3D



- Image is in color, so 3 channels
- This means the convolution filter will have three channels, one per color, each of dimension 3x3
- Results are summed to form a 2-dimensional output. Color is "forgotton" at this point
- K different filters here means K 2-d output maps, which can then be treated as channels
- Padding to make the updated image the same size

More notes on convolution

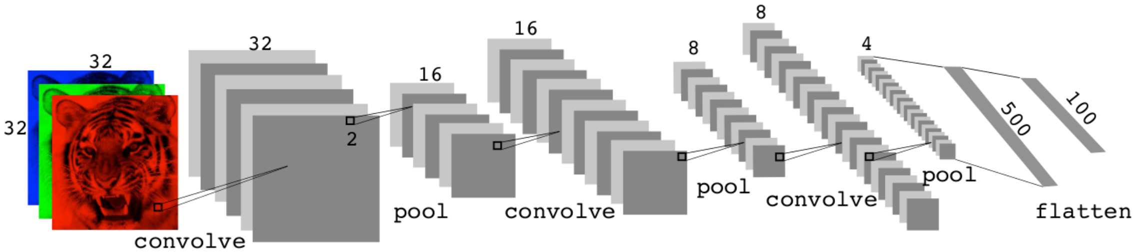
- Filters idea from image processing
- Novelty in DL is that the choice of filters are learned
- Filter weights: one hidden unit per pixel in convolved image; but more constrained than that
- Applying ReLU to convolved image. Sometimes viewed as separate layer in CNN, and then called a detector layer.
- Kernels in CNN corresponds to weights in NN.

Pooling layers

- Condense a large image into a smaller summary image
- Lots of options, but max pooling is common:
- For each 2x2 non-overlapping block of pixels, get a single new pixel with the maximum value from the block
- Provides location invariance: so long as large value in one of the four pixels, the whole block registers as a large value

$$\text{Max pool} \begin{bmatrix} 1 & 2 & 5 & 3 \\ 3 & 0 & 1 & 2 \\ 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 0 \end{bmatrix} \to \begin{bmatrix} 3 & 5 \\ 2 & 4 \end{bmatrix}$$

Putting it together to make a CNN



- Start with three channels
- get a new channel at first hidden layer for each filter. Padding to keep the same size
- Next is max pool layer,
- convolve & pool several more times
- Since size of image decreased, we increase number of filters in the next convolve layer to compensate
- Flattened, then fed into 1 or more fully-connected layers
- Last is softmax activation for the 100 classes

https://poloclub.github.io/cnn-explainer/