CENG 783

Deep Learning

Week – 7
Convolutional Neural Networks (cont.)

Sinan Kalkan
Today

• CNN operations
  • Convolution
  • Pooling
  • Non-linearity
  • Normalization

• CNN Architectures

• Training a CNN

• Practical issues
More on connectivity

Small RF & Stacking
- E.g., 3 CONV layers of 3x3 RFs
- Pros:
  - Same extent for these example figures
  - Non-linearity added with 2nd and 3rd layers \( \Rightarrow \) More expressive! More representational capacity!
  - Less parameters: \( 3 \times [(3 \times 3 \times C) \times C] = 27CxC \)
- Cons?
  
So, we prefer a stack of small filter sizes against big ones

Large RF & Single Layer
- 7x7 RFs of single CONV layer
- Cons:
  - Linear operation in one layer
  - More parameters: \( (7x7xC)xC = 49CxC \)
- Pros?
Convolution layer: How to train?

• How to backpropagate?
  • Calculate error for each neuron, add up gradients but update only one set of weights per slice

• More on this later...
Implementation Details: Numpy example

- Suppose input is volume is $X$ of shape $(11,11,4)$
- Depth slice at depth $d$ (i.e., channel $d$): $X[:,:,d]$  
- Depth column at position $(x,y)$: $X[x,y,:]$ 
- $F$: 5, $P$:0 (no padding), $S=2$
  - Output volume $(V)$ width, height $=(11-5+0)/2+1 = 4$
- Example computation for some neurons in first channel:
  
  $$V[0,0,0] = \text{np.sum}(X[:,5,:,:] * W0) + b0$$
  $$V[1,0,0] = \text{np.sum}(X[2:7,:,:] * W0) + b0$$
  $$V[2,0,0] = \text{np.sum}(X[4:9,:,:] * W0) + b0$$
  $$V[3,0,0] = \text{np.sum}(X[6:11,:,:] * W0) + b0$$
- Note that this is just along one dimension

http://cs231n.github.io/convolutional-networks/
• A second activation map (channel):

\[
\begin{align*}
V[0,0,1] &= \text{np.sum}(X[:,5,:,] \times W1) + b1 \\
V[1,0,1] &= \text{np.sum}(X[2:7,:,] \times W1) + b1 \\
V[2,0,1] &= \text{np.sum}(X[4:9,:,] \times W1) + b1 \\
V[3,0,1] &= \text{np.sum}(X[6:11,:,] \times W1) + b1 \\
V[0,1,1] &= \text{np.sum}(X[:,5,2:7,:] \times W1) + b1 \\
V[2,3,1] &= \text{np.sum}(X[4:9,6:11,:] \times W1) + b1
\end{align*}
\]

• To utilize fast processing of matrix multiplications, we will convert each $w \times h \times d$ volume (input & weights) into a single vector using `im2col` function.
  • This leads to redundancy since the receptive fields overlap
  • We can then have just a single dot product: \texttt{np.dot(W\_row, X\_col)}
  • The result needs to be shaped back

http://cs231n.github.io/convolutional-networks/
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters $K$,
  - their spatial extent $F$,
  - the stride $S$,
  - the amount of zero padding $P$.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P) / S + 1$
  - $H_2 = (H_1 - F + 2P) / S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and $K$ biases.
- In the output volume, the $d$-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the $d$-th filter over the input volume with a stride of $S$, and then offset by $d$-th bias.
Unshared convolution

• In some cases, sharing the weights do not make sense
  • When?

• Different parts of the input might require different types of features

• In such a case, we just have a network with local connectivity

• E.g., a face.
  • Features are not repeated across the space.
Dilated Convolution

Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) $F_1$ is produced from $F_0$ by a 1-dilated convolution; each element in $F_1$ has a receptive field of $3 \times 3$. (b) $F_2$ is produced from $F_1$ by a 2-dilated convolution; each element in $F_2$ has a receptive field of $7 \times 7$. (c) $F_3$ is produced from $F_2$ by a 4-dilated convolution; each element in $F_3$ has a receptive field of $15 \times 15$. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.
Deconvolution

In a sense, upsampling with factor $f$ is convolution with a *fractional* input stride of $1/f$. So long as $f$ is integral, a natural way to upsample is therefore *backwards convolution* (sometimes called *deconvolution*) with an *output* stride of $f$. Such an operation is trivial to implement, since it simply reverses the forward and backward passes of convolution.

https://github.com/vdumoulin/conv_arithmetic
Convolution demos

- https://github.com/vdumoulin/conv_arithmetic


Efficient convolution

• Convolution in time-domain means multiplication in frequency domain
  • For some problem sizes, converting the signals to the frequency domain, performing multiplication in the frequency domain and transforming them back is more efficient

• A kernel is called separable if it can be written as a product of two vectors:

\[
\frac{1}{4} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \ast \frac{1}{4} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}
\]

• Separable kernels can be stored with less memory if you store just the vectors
• Moreover, processing using the vectors is faster than using naïve convolution

http://cs231n.github.io/convolutional-networks/
Pooling
layer / operation
Pooling

- Apply an **operation** on the “detector” results **to combine or to summarize** the answers of a set of units.
  - Applied to each channel (depth slice) **independently**
  - The operation has to be differentiable of course.

- **Alternatives:**
  - Maximum
  - Sum
  - Average
  - Weighted average with distance from the center pixel
  - L2 norm
  - Second-order statistics?
  - ...

- Different problems may perform better with different pooling methods

- Pooling can be overlapping or non-overlapping

http://cs231n.github.io/convolutional-networks/
Pooling

- **Example**
  - Pooling layer with filters of size 2x2
  - With stride = 2
  - Discards 75% of the activations
  - Depth dimension remains unchanged

- **Max pooling with F=3, S=2 or F=2, S=2** are quite common.
  - Pooling with bigger receptive field sizes can be destructive

- Avg pooling is an obsolete choice. Max pooling is shown to work better in practice.

http://cs231n.github.io/convolutional-networks/
Pooling

- Pooling provides invariance to \textit{small} translation.

- If you pool over different convolution operators, you can gain invariance to different transformations.

Pooling can downsample

- Especially needed when to produce an output with fixed-length on varying length input.

Some theoretical work gives guidance as to which kinds of pooling one should use in various situations (Boureau et al., 2010). It is also possible to dynamically pool features together, for example, by running a clustering algorithm on the locations of interesting features (Boureau et al., 2011). This approach yields a different set of pooling regions for each image. Another approach is to learn a single pooling structure that is then applied to all images (Jia et al., 2012).

Pooling can downsample

- If you want to use the network on images of varying size, you can arrange this with pooling (with the help of convolutional layers)
- E.g., to the last classification layer, provide pooling results of four quadrants in the image.
  - Invariant on the size.
Backpropagation for pooling

• E.g., for a max operation:

• Remember the derivative of $\text{max}(x, y)$:
  • $f(x, y) = \text{max}(x, y)$
  • $\nabla f = [1(x \geq y), \ 1(y \geq x)]$

• Interpretation: only routing the gradient to the input that had the biggest value in the forward pass.

• This requires that we save the index of the max activation (sometimes also called the switches) so that gradient “routing” is handled efficiently during backpropagation.

http://cs231n.github.io/convolutional-networks/
Recent developments

“Striving for Simplicity: The All Convolutional Net proposes to discard the pooling layer in favor of architecture that only consists of repeated CONV layers. To reduce the size of the representation they suggest using larger stride in CONV layer once in a while.”

http://cs231n.github.io/convolutional-networks/
For more on pooling

- This is for binary features though.
Convolution & pooling

• Provide strong bias on the model and the solution

• They directly affect the overall performance of the system
Non-linearity
Layer
Non-linearity

- Sigmoid
- Tanh
- ReLU and its variants
  - The common choice
  - Faster
  - Easier (in backpropagation etc.)
  - Avoids saturation issues
- ...
- ...
Normalization Layer
Normalization layer

• Normalize the responses of neurons
• Necessary especially when the activations are not bounded (e.g., a ReLU unit)
• Replaced by drop-out, batch normalization, better initialization etc.
• Has little impact
• Not favored anymore
generalization. Denoting by $a^i_{x,y}$ the activity of a neuron computed by applying kernel $i$ at position $(x, y)$ and then applying the ReLU nonlinearity, the response-normalized activity $b^i_{x,y}$ is given by the expression

$$b^i_{x,y} = a^i_{x,y} / \left( k + \alpha \sum_{j=\max(0, i-n/2)}^{\min(N-1, i+n/2)} (a^j_{x,y})^2 \right)^\beta$$

where the sum runs over $n$ “adjacent” kernel maps at the same spatial position, and $N$ is the total number of kernels in the layer. The ordering of the kernel maps is of course arbitrary and determined before training begins. This sort of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels. The constants $k$, $n$, $\alpha$, and $\beta$ are hyper-parameters whose values are determined using a validation set; we used $k = 2$, $n = 5$, $\alpha = 10^{-4}$, and $\beta = 0.75$. We
Fully-Connected Layer
Fully-connected layer

- At the top of the network for mapping the feature responses to output labels
- Full connectivity
- Can be many layers
- Various activation functions can be used
FC & Conv Layer Conversion

• CONV to FC conversion:
  • weight matrix would be a large matrix that is mostly zero except for at certain blocks (due to local connectivity) where the weights in many of the blocks are equal (due to parameter sharing).

• FC to CONV conversion:
  • Treat the neurons of the next layer as the different channels.
CNN Architectures
INPUT $\rightarrow$ ([CONV $\rightarrow$ RELU]$^N$ $\rightarrow$ POOL?]$^M$ $\rightarrow$ ([FC $\rightarrow$ RELU]$^K$ $\rightarrow$ FC)

where the * indicates repetition, and the POOL? indicates an optional pooling layer. Moreover, $N \geq 0$ (and usually $N \leq 3$), $M \geq 0$, $K \geq 0$ (and usually $K < 3$). For example, here are some common ConvNet architectures you may see that follow this pattern:

- INPUT $\rightarrow$ FC, implements a linear classifier. Here $N = M = K = 0$.
- INPUT $\rightarrow$ CONV $\rightarrow$ RELU $\rightarrow$ FC
- INPUT $\rightarrow$ ([CONV $\rightarrow$ RELU $\rightarrow$ POOL]$^2$ $\rightarrow$ FC $\rightarrow$ RELU $\rightarrow$ FC). Here we see that there is a single CONV layer between every POOL layer.
- INPUT $\rightarrow$ ([CONV $\rightarrow$ RELU $\rightarrow$ CONV $\rightarrow$ RELU $\rightarrow$ POOL]$^3$ $\rightarrow$ ([FC $\rightarrow$ RELU]$^2$ $\rightarrow$ FC). Here we see two CONV layers stacked before every POOL layer. This is generally a good idea for larger and deeper networks, because multiple stacked CONV layers can develop more complex features of the input volume before the destructive pooling operation.
Demo

http://scs.ryerson.ca/~aharley/vis/conv/
General rule of thumbs: The input layer

- The size of the input layer should be divisible by 2 many times
  - Hopefully a power of 2
- E.g.,
  - 32 (e.g. CIFAR-10),
  - 64,
  - 96 (e.g. STL-10), or
  - 224 (e.g. common ImageNet ConvNets),
  - 384, and 512 etc.
General rule of thumbs: The conv layer

- Small filters with stride 1
- Usually zero-padding applied to keep the input size unchanged
- In general, for a certain $F$, if you choose
  \[ P = (F - 1)/2, \]
  the input size is preserved (for $S=1$):
  \[ \frac{W - F + 2P}{S} + 1 \]
- Number of filters:
  - A convolution channel is more expensive compared to fully-connected layer.
  - We should keep this as small as possible.
General rule of thumbs: The pooling layer

- Commonly,
  - $F=2$ with $S=2$
  - Or: $F=3$ with $S=2$

- Bigger $F$ is very destructive
Taking care of downsampling

- At some point(s) in the network, we need to reduce the size

- If conv layers do not downsize, then only pooling layers take care of downsampling

- If conv layers also downsize, you need to be careful about strides etc. so that
  (i) the dimension requirements of all layers are satisfied and
  (ii) all layers tile up properly.

- S=1 seems to work well in practice

- However, for bigger input volumes, you may try bigger strides
Training a CNN

Fig: http://www.robots.ox.ac.uk/~vgg/practicals/cnn/
Feed-forward pass

\[ x_{ij}^\ell = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \omega_{ab} y_{i+a,j+b}^{\ell-1} \]

Note that this is written in terms of the weight indices.

Then, let us say that we add an immediate non-linearity after each convolution layer:

\[ y_{ij}^\ell = \sigma(x_{ij}^\ell). \]

Backpropagation

- Convolution Layer:
  - Calculate the gradient on the weights of the filter summing up over all the expressions that the weights were used in

\[
\frac{\partial E}{\partial \omega_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial x^l_{ij}} \frac{\partial x^l_{ij}}{\partial \omega_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial x^l_{ij}} y^l_{(i+a)(j+b)}
\]

\[
\frac{\partial E}{\partial x^l_{ij}} = \frac{\partial E}{\partial y^l_{ij}} \frac{\partial y^l_{ij}}{\partial x^l_{ij}} = \frac{\partial E}{\partial y^l_{ij}} \frac{\partial}{\partial x^l_{ij}} \left( \sigma(x^l_{ij}) \right) = \frac{\partial E}{\partial y^l_{ij}} \sigma'(x^l_{ij})
\]

Backpropagation

Gradient for the previous layer:

\[
\frac{\partial E}{\partial y_{ij}^{l-1}} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial x_{(i-a)(j-b)}^l} \frac{\partial x_{(i-a)(j-b)}^l}{\partial y_{ij}^{l-1}} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial x_{(i-a)(j-b)}^l} \omega_{ab}
\]

Note that we have convolution here

- However, the indices have flipped!

Backpropagation

- Max pooling and fully-connected layers are straight-forward
How to initialize the weights?

• Option 1: randomly
  • This has been shown to work nicely in the literature

• Option 2:
  • Train/obtain the “filters” elsewhere and use them as the weights
  • Unsupervised pre-training using image patches (windows)
  • Avoids full feedforward and backward pass, allows the search to start from a better position
  • You may even skip training the convolutional layers
CONVOLUTIONAL CLUSTERING FOR UNSUPERVISED LEARNING

Aysegul Dundar, Jonghoo Jin, and Eugenio Culurciello
Purdue University, West Lafayette, IN 47907, USA
{adundar, jhjin, euge}@purdue.edu

3.1 LEARNING FILTERS WITH K-MEANS

Our method for learning filters is based on the k-means algorithm. The classic k-means algorithm finds cluster centroids that minimize the distance between points in the Euclidean space. In this context, the points are randomly extracted image patches and the centroids are the filters that will be used to encode images. From this perspective, k-means algorithm learns a dictionary \( D \in \mathbb{R}^{n \times k} \) from the data vector \( w^{(i)} \in \mathbb{R}^n \) for \( i = 1, 2, \ldots, m \). The algorithm finds the dictionary as follows:

\[
s_j^{(i)} := \begin{cases} D(j)^T w^{(i)} & \text{if } j = \arg\max_l \left| D(l)^T w^{(i)} \right|, \\ 0 & \text{otherwise}, \end{cases}
\]

\[
D := \frac{WS^T + D}{\|D(j)\|_2},
\]

where \( s^{(i)} \in \mathbb{R}^k \) is the code vector associated with the input \( w^{(i)} \), and \( D(j) \) is the \( j \)th column of the dictionary \( D \). The matrices \( W \in \mathbb{R}^{n \times m} \) and \( S \in \mathbb{R}^{k \times m} \) have the columns \( w^{(i)} \) and \( s^{(i)} \), respectively. \( w^{(i)} \)'s are randomly extracted patches from input images that have the same dimension as the dictionary vectors, \( D(j) \).
Now

- More on CNNs
  - Memory constraints
  - Design choices
  - Visualizing and understanding CNNs
  - Popular CNN models
  - CNN applications
Memory & performance considerations
Memory

• Main sources of memory load:

• Activation maps:
  • Training: They need to be kept during training so that backpropagation can be performed
  • Testing: No need to keep the activations of earlier layers

• Parameters:
  • The weights, their gradients and also another copy if momentum is used

• Data:
  • The originals + their augmentations

• If all these don’t fit into memory,
  • Use batches
  • Decrease the size of your batches
Memory constraints

• Using smaller RFs with stacking means more memory since you need to store more activation maps

• In such memory-scarce cases,
  • the first layer may use bigger RFs with $S > 1$
  • information loss from the input volume may be less critical than the following layers

• E.g., AlexNet uses RFs of 11x11 and $S = 4$ for the first layer.
Trade-offs in architecture

- Between filter size and number of layers
  - Keep the layer widths fixed.
  - Deeper networks with smaller filter sizes perform better (if you keep the overall computational complexity fixed)

- Between layer width and number of layers
  - Keep the size of the filters fixed.
  - Increase the number of layers or the depth
  - Increasing depth improves performance

- Between filter size and layer width
  - Keep the number of layers fixed.
  - No significant difference