Lecture 9: Convolutional Neural Nets

Mark Hasegawa-Johnson

ECE 417: Multimedia Signal Processing, Fall 2023

1. Review: Neural Network
2. Convolutional Layers
3. Backprop of Convolution is Correlation
4. Max Pooling
5. A Few Important Papers
6. Summary
7. Written Example

Outline

1. Review: Neural Network
2. Convolutional Layers
3. Backprop of Convolution is Correlation
4. Max Pooling
5. A Few Important Papers
6. Summary
7. Written Example

Review: How to train a neural network

1. Find a **training dataset** that contains n examples showing the desired output, $\mathbf{y}_i$, that the NN should compute in response to input vector $\mathbf{x}_i$:

$$\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$$

2. Randomly **initialize** the weights and biases, $\mathbf{W}_1$, $\mathbf{b}_1$, $\mathbf{W}_2$, and $\mathbf{b}_2$.
3. Perform **forward propagation**: find out what the neural net computes as $\mathbf{g}(\mathbf{x}_i)$ for each $\mathbf{x}_i$.
4. Define a **loss function** that measures how badly $\mathbf{g}(\mathbf{x})$ differs from $\mathbf{y}$.
5. Perform **back propagation** to improve $\mathbf{W}_1$, $\mathbf{b}_1$, $\mathbf{W}_2$, and $\mathbf{b}_2$.
6. Repeat steps 3-5 until convergence.

Review: Second Layer = Piece-Wise Approximation

The second layer of the network approximates $\mathbf{g}(\mathbf{x}) \approx \mathbf{y}$ using a bias term $\mathbf{b}$, plus correction vectors $\mathbf{w}_{2,:,j}$, each scaled by its activation h_j:

$$\mathbf{g}(\mathbf{x}) = \mathbf{b}_2 + \sum_j \mathbf{w}_{2,:,j} h_j$$

- Unit-step and signum nonlinearities, on the hidden layer, cause the neural net to compute a piece-wise constant approximation of the target function. Sigmoid and tanh are differentiable approximations of unit-step and signum, respectively.
- ReLU, Leaky ReLU, and PReLU activation functions cause h_j, and therefore $\mathbf{g}(\mathbf{x})$, to be a piece-wise-linear function of its inputs.

Review: First Layer = A Series of Decisions

The first layer of the network decides whether or not to “turn on” each of the h_j’s. It does this by comparing $\mathbf{x}$ to a series of linear threshold vectors:

$$h_k = \sigma\left(\mathbf{w}_{1,k,:}^T \mathbf{x} + b_k\right) \begin{cases} \approx 1 & \mathbf{w}_{1,k,:}^T \mathbf{x} + b_k > 0 \\ \approx 0 & \mathbf{w}_{1,k,:}^T \mathbf{x} + b_k < 0 \end{cases}$$

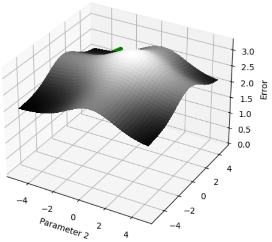
Gradient Descent: How do we improve $\mathbf{W}_l$ and $\mathbf{b}_l$?

Given some initial neural net parameter, $w_{l,k,j}$, we want to find a better value of the same parameter. We do that using gradient descent:

$$w_{l,k,j} \leftarrow w_{l,k,j} - \eta \frac{d\mathcal{L}}{dw_{l,k,j}},$$

where η is a learning rate (some small constant, e.g., $\eta = 0.001$ or so).

One step of gradient descent on a complicated error surface



Error Metrics Summarized

- Use MSE to achieve $\mathbf{g}(\mathbf{x}) \rightarrow E[\mathbf{y}|\mathbf{x}]$: appropriate for regression applications.
- For a binary classifier with a sigmoid output, BCE loss gives you the MSE result without the vanishing gradient problem.
- For a multi-class classifier with a softmax output, CE loss gives you the MSE result without the vanishing gradient problem.
- After you're done training, you can make your cell phone app more efficient by throwing away the uncertainty:
  - Replace softmax output nodes with max
  - Replace logistic output nodes with unit-step
  - Replace tanh output nodes with signum

Outline

1 Review: Neural Network

2 Convolutional Layers

3 Backprop of Convolution is Correlation

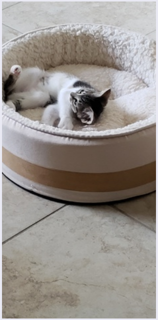
4 Max Pooling

5 A Few Important Papers

6 Summary

7 Written Example

Multimedia Inputs = Too Much Data



Does this image contain a cat?

Fully-connected solution:

$$\mathbf{g}(\mathbf{x}) = \sigma\left(\mathbf{W}_2\mathbf{a}_1 + \mathbf{b}_2\right)$$
$$\mathbf{a}_1 = \mathrm{ReLU}\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right)$$

where $\mathbf{x}$ contains all the pixels.

- Image size $2000 \times 3000 \times 3 = 18,000,000$ dimensions in $\mathbf{x}$.
- If $\mathbf{a}_1$ has 500 dimensions, then $\mathbf{W}_1$ has $500 \times 18,000,000 = 9,000,000,000$ parameters.
- ...so we should use at least $9,000,000,000$ images to train it.

Shift Invariance

The cat has moved. The fully-connected network has no way to share information between the rows of $\mathbf{W}_1$ that look at the center of the image, and the rows that look at the right-hand side.

How to achieve shift invariance: Convolution

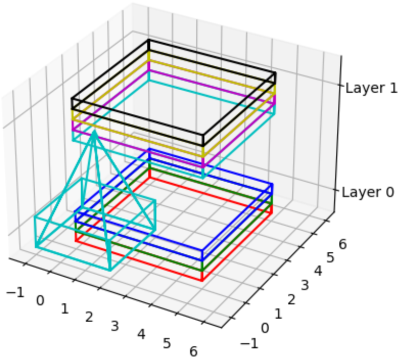
Instead of using vectors as layers, let's use images.

$$z[l,d,m,n] = \sum_c \sum_{m'} \sum_{n'} w[l,d,c,m-m',n-n']a[l-1,c,m',n']$$

where

- $z[l,c,m,n]$ and $a[l,c,m,n]$ are excitation and activation (respectively) of the $(m,n)^{\text{th}}$ pixel, in the c^{th} channel, in the l^{th} layer.
- $w[l,d,c,m-m',n-n']$ are weights connecting c^{th} input channel to d^{th} output channel, with a shift of $m-m'$ rows, $n-n'$ columns.

How to achieve shift invariance: Convolution



How to use convolutions in a classifier

- The zero$^{\text{th}}$ layer is the input image, where $c \in \{1,2,3\}$ denotes color (red, green or blue):

$$a[0,c,m,n] = x[c,m,n]$$

- Excitation and activation:

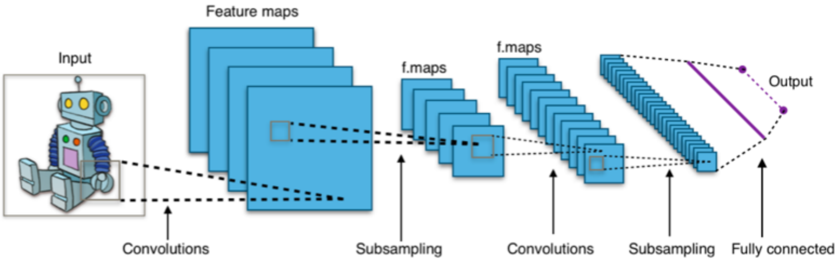
$$z[l,d,m,n] = \sum_c \sum_{m'} \sum_{n'} w[d,c,m-m',n-n']a[l-1,c,m',n']$$
$$a[l,d,m,n] = \text{ReLU}\left(z[l,d,m,n]\right)$$

- Reshape the last convolutional layer into a vector, to form the first fully-connected layer:

$$\mathbf{a}_{L+1} = [a[L,1,1,1], a[L,1,1,2], \ldots, a[L,3,M,N]]^T$$

where $M \times N$ is the image dimension.

How to use convolutions in a classifier



"Typical CNN," by Aphex34 2015, CC-SA 4.0, `https://commons.wikimedia.org/wiki/File:Typical_cnn.png`

Outline

1. Review: Neural Network
2. Convolutional Layers
3. Backprop of Convolution is Correlation
4. Max Pooling
5. A Few Important Papers
6. Summary
7. Written Example

How to back-prop through a convolutional neural net

You already know how to back-prop through fully-connected layers. Now let's back-prop through convolution:

$$\frac{\partial \mathcal{L}}{\partial a[l-1, c, m', n']} = \sum_m \sum_n \sum_d \frac{\partial \mathcal{L}}{\partial z[l, d, m, n]} \frac{\partial z[l, d, m, n]}{\partial a[l-1, c, m', n']}$$

We need to find two things:

1. What is $\frac{\partial \mathcal{L}}{\partial z[l,d,m,n]}$? Answer: We can assume it's already known, because we have already back-propagated as far as layer l.
2. What is $\frac{\partial z[l,d,m,n]}{\partial a[l-1,c,m',n']}$? Answer: That is the new thing that we need, in orer to back-propagate to layer $l-1$.

How to back-prop through convolution

Here is the formula for convolution:

$$z[l,d,m,n] = \sum_{c}\sum_{m'}\sum_{n'} w[l,d,c,m-m',n-n']a[l-1,c,m',n']$$

If we differentiate the left side w.r.t. the right side, we get:

$$\frac{\partial z[l,d,m,n]}{\partial a[l-1,c,m',n']} = w[l,d,c,m-m',n-n']$$

Plugging into the formula on the previous slide, we get:

$$\frac{\partial \mathcal{L}}{\partial a[l-1,c,m',n']} = \sum_{m}\sum_{n}\sum_{d} w[l,d,c,m-m',n-n']\frac{d\mathcal{L}}{dz[l,d,m,n]}$$

Convolution forward, Correlation backward

In signal processing, we defined $a[n] * w[n]$ to mean $\sum w[n']a[n-n']$. Let's use the same symbol to refer to this multi-channel 2D convolution:

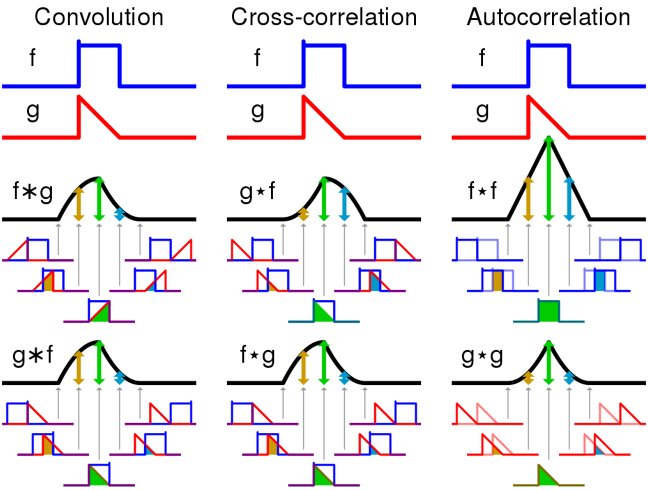
$$
\begin{aligned}
z[l,d,m,n] &= \sum_c \sum_{m'} \sum_{n'} w[l,d,c,m-m',n-n']a[l-1,c,m',n'] \\
&\equiv w[l,m,n,c,d] * h[l-1,c,m,n]
\end{aligned}
$$

Back-propagation looks kind of similar, but notice that now, instead of $\sum_{n'} w[n-n']a[n']$, we have $\sum_n w[n-n']a[n]$:

$$
\frac{\partial \mathcal{L}}{\partial a[l-1,c,m',n']} = \sum_m \sum_n \sum_c w[l,d,c,m-m',n-n'] \frac{\partial \mathcal{L}}{\partial z[l,d,m,n]}
$$

In other words, we are summing over the variable on which $w[n]$ has **not been flipped**. What is that?

Convolution versus Correlation



https://upload.wikimedia.org/wikipedia/commons/thumb/2/21/Comparison_convolution_correlation.svg/1024px-Comparison_convolution_correlation.svg.png

Convolution versus Correlation

- Convolution is when we flip one of the two signals, shift, multiply, then add:

$$a[m] * w[m] = \sum_{m'} w[m-m']a[m']$$

- Correlation is when we only shift, multiply, and add:

$$a[m'] \bigstar w[m'] = \sum_{m} w[m-m']a[m]$$

The Back-Prop of Convolution is Correlation

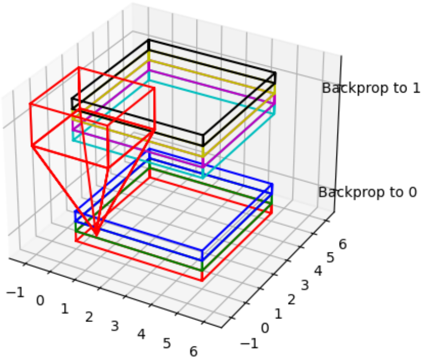
$$\frac{\partial \mathcal{L}}{\partial a[l-1,c,m',n']} = \sum_m \sum_n \sum_c w[l,d,c,m-m',n-n'] \frac{\partial \mathcal{L}}{\partial z[l,d,m,n]}$$
$$= w[l,d,c,m',n'] \bigstar \frac{d\mathcal{L}}{\partial z[l,d,m',n']}$$

The Back-Prop of Convolution is Correlation

$$z[l, d, m, n] = w[l, m, n, c, d] * h[l-1, c, m, n]$$

$$\frac{\partial \mathcal{L}}{\partial a[l-1, c, m', n']} = w[l, d, c, m', n'] \bigstar \frac{d\mathcal{L}}{\partial z[l, d, m', n']}$$

Back-prop through a convolutional layer



Similarities between convolutional and fully-connected back-prop

- In a fully-connected layer, forward-prop means multiplying a matrix by a column vector on the right. Back-prop means multiplying the same matrix by a row vector from the left:

$$\mathbf{z}_l = \mathbf{W}_l \mathbf{a}_{l-1}$$
$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}_{l-1}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}_l} \mathbf{W}_l$$

- In a convolutional layer, forward-prop is a convolution, Back-prop is a correlation:

$$z[l,d,m,n] = w[l,m,n,c,d] * h[l-1,c,m,n]$$
$$\frac{d\mathcal{L}}{dh[l-1,c,m,n]} = w[l,d,c,m',n'] \bigstar \frac{d\mathcal{L}}{dz[l,d,m',n']}$$

Convolutional layers: Weight gradient

Finally, we need to combine back-prop and forward-prop in order to find the weight gradient:

$$\frac{d\mathcal{L}}{dw[l,d,c,m',n']} = \sum_m \sum_n \frac{d\mathcal{L}}{dz[l,d,m,n]} \frac{\partial z[l,d,m,n]}{\partial w[l,d,c,m',n']}$$

Again, here's the formula for convolution:

$$z[l,d,m,n] = \sum_c \sum_{m'} \sum_{n'} w[l,d,c,m',n'] a[l-1,c,m-m',n-n']$$

If we differentiate the left side w.r.t. the right side, we get:

$$\frac{\partial z[l,d,m,n]}{\partial w[l,d,c,m',n']} = a[l-1,c,m-m',n-n']$$

Convolutional layers: Weight gradient

$$\frac{\partial\mathcal{L}}{\partial w[l,d,c,m',n']} = \sum_m\sum_n \frac{d\mathcal{L}}{dz[l,d,m,n]}\frac{\partial z[l,d,m,n]}{\partial w[l,d,c,m',n']}$$

$$\frac{\partial z[l,d,m,n]}{\partial w[l,d,c,m',n']} = a[l-1,c,m-m',n-n']$$

Putting those together, we discover that the weight gradient is a correlation:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial w[l,d,c,m',n']} &= \sum_m\sum_n \frac{\partial\mathcal{L}}{\partial z[l,d,m,n]}a[l-1,c,m-m',n-n'] \\ &= \frac{\partial\mathcal{L}}{\partial z[l,d,m',n']}\bigstar a[l-1,c,m',n']\end{aligned}$$

Steps in training a CNN

1. Forward-prop is convolution:

$$z[l,d,m,n] = w[l,d,c,m,n] * a[l-1,c,m,n]$$

2. Back-prop is correlation:

$$\frac{\partial \mathcal{L}}{\partial a[l-1,c,m,n]} = w[l,d,c,m,n] \bigstar \frac{\partial \mathcal{L}}{\partial z[l,d,m,n]}$$

3. Weight gradient is correlation:

$$\frac{\partial \mathcal{L}}{\partial w[l,d,c,m,n]} = \frac{\partial \mathcal{L}}{\partial z[l,d,m,n]} \bigstar a[l-1,c,m,n]$$

Outline

1. Review: Neural Network
2. Convolutional Layers
3. Backprop of Convolution is Correlation
4. Max Pooling
5. A Few Important Papers
6. Summary
7. Written Example

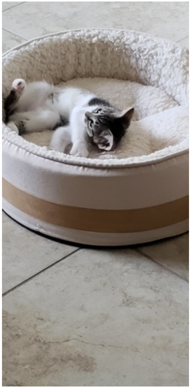
Features and PWL Functions

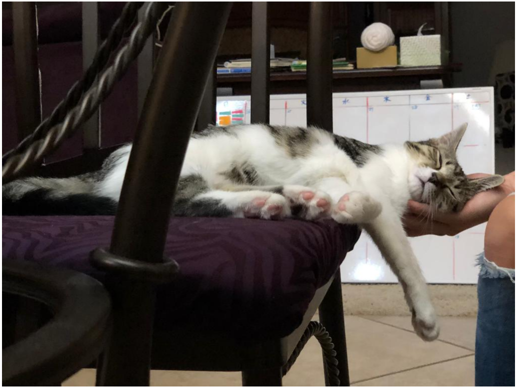
Remember the PWL model of a ReLU neural net:

1. The hidden layer activations are positive if some feature is detected in the input, and zero otherwise.
2. The rows of the output layer are vectors, scaled by the hidden layer activations, in order to approximate some desired piece-wise-linear (PWL) output function.
3. What happens next is different for regression and classification:
    1. Regression: The PWL output function is the desired output.
    2. Classification: The PWL function is squashed down to the $[0, 1]$ range using a sigmoid.

Features and PWL Functions

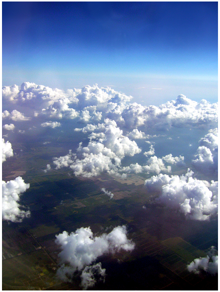
In image processing, often we don't care where in the image the "feature" occurs:





Features and PWL Functions

Sometimes we care **roughly** where the feature occurs, but not exactly. Blue at the bottom is sea, blue at the top is sky:



"Paracas National Reserve," World Wide Gifts, 2011, CC-SA 2.0,

`https://commons.wikimedia.org/wiki/File:Paracas_National_Reserve,_Ica,_Peru-3April2011.jpg`.

"Clouds above Earth at 10,000 feet," Jessie Eastland, 2010, CC-SA 4.0,

`https://commons.wikimedia.org/wiki/File:Sky-3.jpg`.

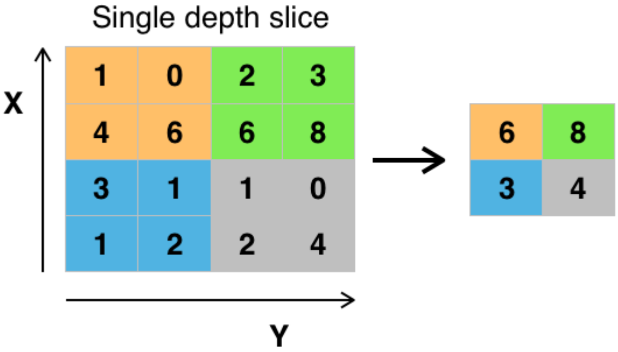
Max Pooling

- Philosophy: the activation $a[l,c,m,n]$ should be greater than zero if the corresponding feature is detected anywhere within the vicinity of pixel (m,n). In fact, let's look for the *best matching* input pixel.
- Equation:

$$a[l,c,m,n] = \max_{m'=0}^{M-1} \max_{n'=0}^{M-1} \text{ReLU}\left(z[l,c,mM+m',nM+n']\right)$$

where M is a max-pooling factor (often $M=2$, but not always).

Max Pooling



"max pooling with 2x2 filter and stride = 2," Aphex34, 2015, CC SA 4.0,

`https://commons.wikimedia.org/wiki/File:Max_pooling.png`

Back-Prop for Max Pooling

The back-prop is pretty easy to understand. The activation gradient, $\frac{\partial \mathcal{L}}{\partial a[l,c,m,n]}$, is back-propagated to just one of the excitation gradients in its pool: the one that had the maximum value.

$$\frac{\partial \mathcal{L}}{\partial z[l,c,mM+m',nM+n']} = \begin{cases} \frac{\partial \mathcal{L}}{\partial a[l,c,m,n]} & \begin{array}{l}(m',n')=(m^*,n^*) \\ a[l,c,m,n]>0\end{array} \\ 0 & \text{otherwise,} \end{cases}$$

where:

$$(m^*,n^*) = \underset{m'=0}{\overset{M-1}{\operatorname{argmax}}}\ \underset{n'=0}{\overset{M-1}{\operatorname{argmax}}}\ z[l,c,mM+m',nM+n']$$

Other types of pooling

- **Average pooling:**

$$a[l,c,m,n] = \frac{1}{M^2}\sum_{m'=0}^{M-1}\sum_{n'=0}^{M-1} \mathrm{ReLU}\left(z[l,c,mM+m',nM+n']\right)$$

Philosophy: instead of finding the pixels that best match the feature, find the average degree of match.

- **Decimation pooling:**

$$a[l,c,m,n] = \mathrm{ReLU}\left(z[l,c,mM,nM]\right)$$

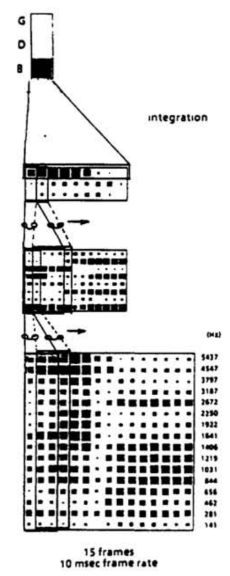
Philosophy: the convolution has already done the averaging for you, so it's OK to just throw away the other M^2-1 inputs.

Outline

1. Review: Neural Network
2. Convolutional Layers
3. Backprop of Convolution is Correlation
4. Max Pooling
5. A Few Important Papers
6. Summary
7. Written Example

"Phone Recognition: Neural Networks vs. Hidden Markov Models," Waibel, Hanazawa, Hinton, Shikano and Lang, 1988

- 1D convolution
- average pooling
- max pooling invented by Yamaguchi et al., 1990, based on this architecture



"Backpropagation Applied to Handwritten Zip Code Recognition," LeCun, Boser, Denker & Henderson, 1989 (2D convolution, decimation pooling)

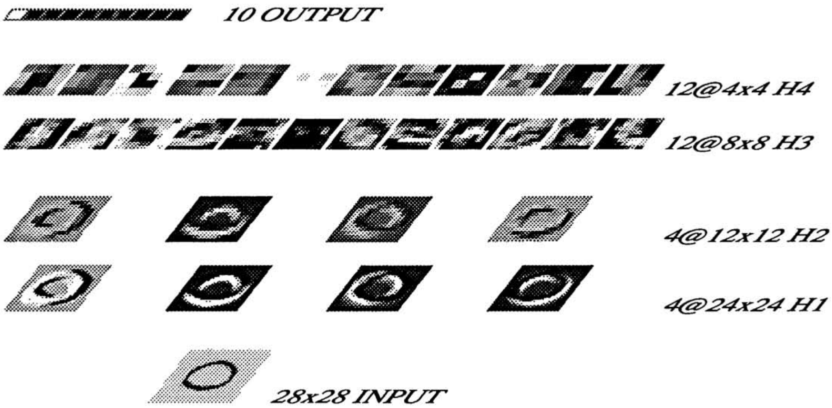
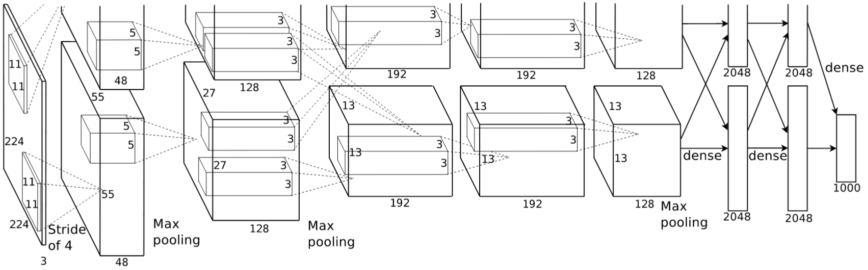


Image copyright Lecun, Boser, et al., 1990

"Imagenet Classification with Deep Convolutional Neural Networks," Krizhevsky, Sutskever & Hinton, 2012 (GPU training)



Outline

1 Review: Neural Network

2 Convolutional Layers

3 Backprop of Convolution is Correlation

4 Max Pooling

5 A Few Important Papers

6 Summary

7 Written Example

Summary

- Convolutional layers: forward-prop is a convolution, back-prop is a correlation, weight gradient is a correlation.
- Max pooling: back-prop just propagates the derivative to the pixel that was chosen by forward-prop.
- Many-layer CNNs trained on GPUs, with small convolutions in each layer, have won Imagenet every year since 2012, and are now a component in every image, speech, audio, and video processing system.

Outline

1. Review: Neural Network
2. Convolutional Layers
3. Backprop of Convolution is Correlation
4. Max Pooling
5. A Few Important Papers
6. Summary
7. Written Example

Written Example

Suppose our input image is a delta function:

$$x[n] = \delta[n]$$

Suppose we have one convolutional layer, and the weights are initialized to be Gaussian:

$$w[n] = e^{-\frac{n^2}{2}}$$

Suppose that the neural net output is

$$\mathbf{g}(\mathbf{x}) = \sigma\left(\max\left(w[n] * x[n]\right)\right),$$

where $\sigma(\cdot)$ is the logistic sigmoid, and $\max(\cdot)$ is max-pooling over the entire output of the convolution. Suppose that the target output is $y = 1$, and we are using binary cross-entropy loss. What is $d\mathcal{L}/dw[n]$, as a function of n?