Deep Neural Network

Ying Wu
Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208

http://www.eecs.northwestern.edu/~yingwu
Outline

Convolutional Neural Network

Recurrent Neural Network
Motivation

- Traditional model of pattern recognition: Fixed hand-crafted feature (kernel) + trainable classifier
- Can both the feature and classifier be learned?
- Feature extractor is equivalent to a kernel, e.g. edge filter, low level feature
- Feature map is response map of convolution operation on image with specified convolution kernel.
- Can this convolution kernel be learned?
- How to get high level feature?
**Architecture Overview**

- **Convolution layer:** Each convolution kernel is a feature extractor. Translation invariance.
- **Pooling layer:** Larger receptive field (see more of input). Invariance to small transformation.
- **Fully connected layer:** Classifier using features from previous layers.

*Figure: CNN structure '98 [2]*
Convolution Layer

- Denote the k-th feature map at a given layer as $h^k$, whose filters are determined by weights $W^k$ and bias $b_k$, $x$ is feature of previous layer.

$$h^k_{ij} = (W^k \ast x)_{ij} + b_k \quad (1)$$

where $\ast$ denotes convolution operation

- If there are $M$ input feature maps $x^m$, $m = 1 \ldots M$

$$h^k_{ij} = \sum_{m=1}^{M} (W^{km} \ast x^m)_{ij} + b_k \quad (2)$$

- Activation value. Use non linear activation function to bring in non-linearity, e.g., for each element of $h$, perform $\tanh(h)$, Sigmoid function: $\frac{1}{1+\exp(-h)}$, Rectified Linear Unit (ReLU): $\max(0, h)$,
Convolution Layer forward propagation

Figure: Convolution forward propagation
Backward propagation

- w.l.g., consider 1-D case for convolution, 1 input feature map and 1 output feature map. The response is calculated in the following:

\[ h = x \ast W, \quad h_n = W^T x_{n:n+|W|-1} \] (3)

where \( h_n \) is the response of \( h \) at location \( n \), \( |W| \) is the length of weight \( W \).

- Consider the derivative of \( h \) w.r.t \( x \) and \( W \):

\[
\frac{\partial h_{n-i+1}}{\partial x_n} = W_i, \quad \frac{\partial h_n}{\partial W_i} = x_{n+i-1}, \quad 1 \leq i \leq |W| \] (4)
Consider sensitivity $\delta^{(x)}$ of objective $J$ w.r.t. input feature $x$.

$$
\delta^{(x)}_n = \frac{\partial J}{\partial x_n} = \frac{\partial J}{\partial h} \frac{\partial h}{\partial x_n} = \left| W \right| \sum_{i=1}^{\left| W \right|} \frac{\partial J}{\partial h_{n-i+1}} \frac{\partial h_{n-i+1}}{\partial x_n} = \sum_{i=1}^{\left| W \right|} \delta^{(h)}_{n-i+1} W_i
$$

(5)

$$
= (\delta^{(h)} \ast flip(W))[n]
$$

where $\delta^{(h)} = \frac{\partial J}{\partial h}$ is sensitivity of $J$ w.r.t. output response $h$ (input feature for the next layer), $\delta^{(x)}_n$ denote value of $\delta^{(x)}$ at location $n$, $W_i$ is weight $W$ at location $i$.

$$
\delta^{(x)} = \delta^{(h)} \ast flip(W)
$$

(6)
Back propagation

Consider the partial derivative of objective $J$ over weight $W_i$.

$$
\frac{\partial J}{\partial W_i} = \frac{\partial J}{\partial h_n} \cdot \frac{\partial h_n}{\partial W_i}
$$

$$
= |x| - |W| + 1 \sum_{n=1} \frac{\partial J}{\partial h_n} \cdot \frac{\partial h_n}{\partial W_i}
$$

$$
= |x| - |W| + 1 \sum_{n=1} \delta^{(h)} x_{n+i-1}
$$

$$
= (\delta^{(h)} * x)[i]
$$

where $|x|$ is the size of $x$

So we have

$$
\frac{\partial J}{\partial W} = \delta^{(h)} * x = x * \delta^{(h)}
$$
Pooling Layer

- Pooling layer is commonly inserted between successive convolution layers.
- It progressively reduce the spatial size of the representation, hence reduces the amount of parameters and computation in the network.
- Common types of pooling includes: average pooling, max pooling, $L^P$ pooling. Recent CNN networks use max pooling.
Forward propatation

- Without losing of generality, we discuss 1-D case, with kernel window size $m$.
- Then response $h$ at location $n$ is the value of subsampled $x$ using subsample function $g$ at location $n$.

$$ h_n = \text{subsample}(x, g)[n] = g(x_{(n-1)m+1:nm}) $$ (9)

where $x_{(n-1)m+1:nm}$ is the value of $x$ at location $(n - 1)m + 1$ to $nm$

- Average pooling, $g(x) = \frac{\sum_{k=1}^{m} x_k}{m}$
- Max pooling, $g(x) = \max(x)$
- $L^P$ pooling, $g(x) = \| x \|_p = \left( \sum_{k=1}^{m} |x_k|^p \right)^{\frac{1}{p}}$
Back propagation

- Average pooling,

\[
\frac{\partial g}{\partial x} = \frac{1}{m}
\]  

(10)

- Max pooling,

\[
\frac{\partial g}{\partial x_i} = \begin{cases} 
1, & \text{if } x_i = \max(x), \\
0, & \text{otherwise}
\end{cases}
\]  

(11)

- \(L^p\) Pooling,

\[
\frac{\partial g}{\partial x_i} = \left(\sum_{k=1}^{m} |x_k|^p\right)^{1/p-1} |x_i|^{p-1}
\]  

(12)
Consider sensitivity $\delta^{(x)}$. $\text{upsample}(\delta^{(h)}, g')[n]$ denotes $n$-th element of $\text{upsample}(\delta^{(h)}, g')$

$$
\delta^{(x)}_{(n-1)m+1:nm} = \text{upsample}(\delta^{(h)}, g')[n] = \delta^{(h)}_n \frac{\partial g}{\partial x_{(n-1)m+1:nm}} = \frac{\partial J}{\partial h_n} \frac{\partial g}{\partial x_{(n-1)m+1:nm}}
$$

(13)

where $\delta^{(h)}$ is sensitivity w.r.t output response $h$ (input feature for the next layer).
Other layers

- **Dropout layer**: randomly discard some activation values, in order to avoid overfitting.
- **Normalization layer**: normalize activation value in a kernel window.
- **Fully connected layer**: a special case of convolution layer, where kernel window size is the same spatial size of input feature map.
What does CNN learn?

- Lower convolution layers: (a) learns edges.
- Middle convolution layers: (b) basic shapes.
- Higher convolution layers: (c) semantic shapes.

Figure: Visualization of features learned on ImageNet from [3]
Outline

Convolutional Neural Network

Recurrent Neural Network
Motivation

- CNN is used to model spatial signal.
- How to model temporal sequence?
- Feed-back connection, so that the activations can flow around in a loop.
- Recurrent Neural Network
Structure Overview

- Fully recurrent network. MLP with the previous set of hidden unit activations feeding back into the network along with the inputs.
- Delay unit. Time $t$ is discrete. Activations are updated at each time step.

Figure: Structure of RNN
State space model

- Inputs and outputs are vectors $x, z$.
- Three connection weight matrices are $W_{xh}, W_{hh}, W_{hz}$.
- Hidden and output unit activation functions are $g$.

\[
\begin{align*}
  h_t & = g(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \\
  z_t & = g(W_{hz}h_t + b_z)
\end{align*}
\]
Stability, Controllability and Observability

- **Stability.** The boundedness over time of the network outputs.
- **Controllability.** If an initial state is steerable to any desired state within a finite number of time steps.
- **Observability.** Observable if the state of the network can be determined from a finite set of input/output measurements.
Unfolding Over Time

Figure: Unfold over time
Backpropagation through time (BPTT)

- With a training sequence from $t_0$ to $t_1$, total cost function is simply the sum over time of the standard error function $E_{sse/ce}(t)$ is standard error function at each time step.

$$E_{total}(t_0, t_1) = \sum_{t=t_0}^{t_1} E_{sse/ce}(t)$$  \hspace{1cm} (16)

- Gradient over weights is the sum over time of gradient of standard error function at each time step

$$\Delta W_i = -\eta \frac{\partial E_{total}(t_0, t_1)}{\partial W_i} = -\eta \sum_{t=t_0}^{t_1} \frac{\partial E_{sse/ce}(t)}{\partial W_i}$$  \hspace{1cm} (17)

where $W_i$ is the weight value of $W$ at location $i$
Backpropagation through time (BPTT)

- The unfold network is quite complex.
- Updates are made in an online fashion, which requires storage of the history of the inputs and past network states. For consideration of computation, truncation is needed at a certain number of time steps.
- If a network is stable, further back in time, smaller the contribution to the weight updates.
- The error brought by a time step too far back in time can be discarded.
Simple Recurrent Network

- Truncate unfolded network to just one time step. (known as Elman network).
- Each set of weights only appears once. Can apply standard back propagation algorithm rather than full BPTT.

![Diagram of Simple Recurrent Network]

**Figure**: Simple unfolding.
NARX model

- Non-linear Auto-Regressive with eXogeneous inputs (NARX) model.
- Single input, single output, a delay line on the inputs, outputs fed back to the input by another delay line.

Figure: NARX model
Long Short-term Memory (LSTM)

- Simple RNN is limited in learning the long-term representation of video sequences.
- Because of the exponential decay in retaining the context information of video frames.
- LSTM is proposed to learn long-range dependency between output label and the input frame.
- LSTM is composed of a sequence of memory cells to preserve long-term temporal information.
Long Short-term Memory (LSTM)

Figure: LSTM [1]
Long Short-term Memory

- In addition to hidden unit $h_t \in \mathbb{R}^N$, LSTM includes an input gate $i_t \in \mathbb{R}^N$, forget gate $f_t \in \mathbb{R}^N$, output gate $o_t \in \mathbb{R}^N$, input modulation gate $g_t \in \mathbb{R}^N$, and memory cell $c_t \in \mathbb{R}^N$. Function $\sigma$ and $\phi$ introduces nonlinearity.

\[
\begin{align*}
    i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \quad (18) \\
    f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \quad (19) \\
    o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \quad (20) \\
    g_t &= \phi(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (21) \\
    c_t &= f_t \odot c_{t-1} + i_t \odot g_t \quad (22) \\
    h_t &= o_t \odot \phi(c_t) \quad (23)
\end{align*}
\]
Long Short-term Memory

- Memory cell $c_t$ is a summation the previous memory cell $c_{t-1}$ and $g_t$, modulation of current input $i_t$ and previous hidden state $h_{t-1}$. $c_{t-1}$ is modulated by value of forget gate $f_t$, $g_t$ is modulated by input gate $i_t$.

- Current hidden state $h_t$ is value of function $\phi(c_t)$ of current memory cell modulated by output gate $o_t$.

- $f_t$, $g_t$ and $o_t$ are decided by input $x_t$ and previous hidden state $h_{t-1}$.

- Input gate $i_t$ controls how much influence input $x_t$, $h_{t-1}$ has on current memory cell $c_t$.

- Input gate $f_t$ controls how much influence previous memory cell $c_{t-1}$ has on current memory cell $c_t$.

- Output gate $o_t$ controls how much influence current cell $c_t$ has on hidden state cell $h_t$. 
Long-term Recurrent Convolutional Networks

- LRCN processes variable-length visual input with a CNN. The output of CNN is fed into a stack of LSTMs. LSTM finally produces a variable-length prediction. CNN is used to capture visual feature from each image. Learned visual information is fed into a stack of LSTM memory cells, which store and pass long range temporal information. The final output is the output of LSTM, which considers both temporal information and visual information.

Figure: Structure of LRCN [1]
Applications of LRCN

- Consider three vision problems
  - Activity recognition. Sequential input, fixed output. Input is video of arbitrary length $T$. The prediction output is a single label of activity, e.g., running, jumping, which are drawn from a fixed vocabulary.
  - Image description. Fixed input, sequential output. Input is a single image. The output prediction is a much larger space consisting of sentences of any length.
  - Video description. Sequential input, sequential output. Input is a sequence of video with varying length $T$. Output is a sentence with varying length $T'$. The length of input video frames has no constraint on the length of the natural-language description.
Applications of LRCN

Figure: Applications of LRCN [1]
Long-term recurrent convolutional networks for visual recognition and description.


Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner.
Gradient-based learning applied to document recognition.


M. D. Zeiler and R. Fergus.
Visualizing and understanding convolutional networks.