Deep Learning
Advanced Deep Learning
Recall: A Neural Network
Neural Network Model

Independent variables

Weights

Hidden Layer

Weights

Dependent variable

Prediction

“Probability of being Alive”
Decision Boundary

- 2 hidden layer
  - Combinations of convex regions

Example from to Eric Postma via Jason Eisner
Backprop in ANN

(A) Input
Given \( x_i \), \( \forall i \)

(B) Hidden (linear)
\[ a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \forall j \]

(C) Hidden (sigmoid)
\[ z_j = \frac{1}{1+\exp(-a_j)}, \forall j \]

(D) Output (linear)
\[ b = \sum_{j=0}^{D} \beta_j z_j \]

(E) Output (sigmoid)
\[ y = \frac{1}{1+\exp(-b)} \]

(F) Loss
\[ J = \frac{1}{2} (y - y^*)^2 \]
Deep Neural Networks
Why Deep Learning

- Features that are engineered by hand are difficult to maintain, error prone, and not necessarily the best for the task
- Can we learn feature directly from data?
Why Now?

- ANNs are not new (The Perceptron is almost 65 y.o.)
Single Layer NN

\[ z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j w_{j,2}^{(1)} = w_{0,2}^{(1)} + x_1 w_{1,2}^{(1)} + x_2 w_{2,2}^{(1)} + x_m w_{m,2}^{(1)} \]
Deep NN

\[ z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(z_{k-1,j}) w_{j,i}^{(k)} \]
Invariants
The General Idea

- Deep feed-forward networks are provably universal. However:
  - We can make them arbitrarily complex.
  - The number of parameters can be huge.
  - Very difficult to optimize.
  - Very difficult to achieve generalization.

- We can take advantage of structural similarities in images:
  - Self-similarity
  - Translation invariance
  - Deformation invariance
  - Hierarchy and compositionality
Self-Similarity

Data tends to be self-similar across the domain:
Translation Invariant

Translations do not change the image content.
Deformation Invariance
Invariance to partiality and isometric deformations
Hierarchy and compositionality
The Discrete Convolution in 2D Domains

- On 2D domains (e.g. RGB images $f : \mathbb{R}^2 \to \mathbb{R}^3$), for each channel:
  \[ (f \ast g)[m, n] = \sum_{k} \sum_{\ell} f[k, \ell] g[m - k, n - \ell] \]

- We can interpret this as a sort of moving windows $f$ over the vector $g$. 
Similarly in 3D

In general, for functions $f : \mathbb{R}^k \rightarrow \mathbb{R}^m$ defined on Euclidean domains, convolution is well-defined up to appropriate boundary conditions.
Boundary Conditions and Stride

- **No padding**: The convolution kernel is directly applied within the boundaries of the underlying function (an image in this example).

- The result of the convolution is a smaller image.
Boundary Conditions and Stride

- **Full padding**: The domain is enlarged and padded with zeros. The convolution kernel is applied within the (now larger) boundaries.

- The result of the convolution is a larger image.
Boundary Conditions and Stride

- **Arbitrary zero-padding, with stride**: The domain is enlarged and padded with zeros, but not enough to capture the boundary pixels. Further, each discrete step skips one pixel.

- The result is the same as no stride followed by downsampling.
ConvNet Architecture
Typical ConvNet Structure
The Convolutional Layers: Filters

- Filter “weights” are learned.
The Convolutional Layers: Filters
The Convolutional Layers: Filters
Multiple Channels

<table>
<thead>
<tr>
<th>Input Channel #1 (Red)</th>
<th>Input Channel #2 (Green)</th>
<th>Input Channel #3 (Blue)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0 ...</td>
<td>0 0 0 0 0 0 0 0 ...</td>
<td>0 0 0 0 0 0 0 0 ...</td>
</tr>
<tr>
<td>0 156 155 156 158 158 ...</td>
<td>0 167 166 167 169 169 ...</td>
<td>0 163 162 163 165 165 ...</td>
</tr>
<tr>
<td>0 153 154 157 159 159 ...</td>
<td>0 164 165 168 170 170 ...</td>
<td>0 160 161 164 166 166 ...</td>
</tr>
<tr>
<td>0 149 151 155 158 159 ...</td>
<td>0 160 162 166 169 170 ...</td>
<td>0 156 158 162 165 166 ...</td>
</tr>
<tr>
<td>0 146 146 149 153 158 ...</td>
<td>0 156 156 159 163 168 ...</td>
<td>0 155 155 158 162 167 ...</td>
</tr>
<tr>
<td>0 145 148 143 148 158 ...</td>
<td>0 155 159 153 158 168 ...</td>
<td>0 154 152 152 157 167 ...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Kernel Channel #1

\[-1 \quad -1 \quad 1\]
\[0 \quad 1 \quad -1\]
\[0 \quad 1 \quad 1\]

308 +

Kernel Channel #2

\[1 \quad 0 \quad 0\]
\[1 \quad -1 \quad -1\]
\[1 \quad 0 \quad -1\]

-498 +

Kernel Channel #3

\[0 \quad 1 \quad 1\]
\[0 \quad 1 \quad 0\]
\[1 \quad -1 \quad 1\]

164 + 1 = -25

Output

Bias = 1
Multiple Filters at the Same Time

- There is a high chance that you may need to extract a lot of different features from an image, for which you will use multiple filters.
- The example considers a 6 x 6 x 3 image, we are using 2 filters (for vertical edge and horizontal edge) of dimension 3 x 3 x 3. The resultant images of 4 x 4 each are stacked together to get a output volume of 4 x 4 x 2.
The Convolutional Layers: Pooling

Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. Left: In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. Right: The most common downsampling operation is max, giving rise to max pooling, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).
Which Pooling?

- **Max Pooling**:
  - Top right: 20, 30
  - Bottom left: 112, 37

- **Average Pooling**:
  - Bottom right: 13, 8
  - Top left: 79, 20
Typical Setup

Here, channels are colors.

Convolutional pooling

Convolutional pooling

Convolutional pooling

Flatten dense

Now, channels are feature maps.
Automatic Feature Extraction
Automatic Feature Extraction

CONV: Convolutional kernel layer
RELU: Activation function
POOL: Dimension reduction layer
FC: Fully connection layer
Try On Your Own

Go to: https://www.cs.ryerson.ca/~aharley/vis/conv/
CNN Explainer

https://poloclub.github.io/cnn-explainer/
Convolutional Layer and Neural Cortex
Some Examples of Architectures
LeNet-5 for MNIST
AlexNet
VGG-16
GoogleNet (Inception Modules)
ResNet

\[ \mathcal{F}(x) \]

\[ \text{weight layer} \]

\[ \text{relu} \]

\[ \text{weight layer} \]

\[ \text{identity} \]

\[ \mathcal{F}(x) + x \]

\[ \text{relu} \]
LeNet for MNIST Colab
CNN for Text
import torch.nn as nn

class CharacterLevelCNN(nn.Module):
    def __init__(self, n_classes=14, input_length=1014, input_dim=68,
                 n_conv_filters=256,
                 n_fc_neurons=1024):
        super(CharacterLevelCNN, self).__init__()
        self.conv1 = nn.Sequential(nn.Conv1d(input_dim, n_conv_filters, kernel_size=7, padding=0), nn.ReLU(), nn.MaxPool1d(3))
        self.conv2 = nn.Sequential(nn.Conv1d(n_conv_filters, n_conv_filters, kernel_size=7, padding=0), nn.ReLU(), nn.MaxPool1d(3))
        self.conv3 = nn.Sequential(nn.Conv1d(n_conv_filters, n_conv_filters, kernel_size=3, padding=0), nn.ReLU())
        self.conv4 = nn.Sequential(nn.Conv1d(n_conv_filters, n_conv_filters, kernel_size=3, padding=0), nn.ReLU())
        self.conv5 = nn.Sequential(nn.Conv1d(n_conv_filters, n_conv_filters, kernel_size=3, padding=0), nn.ReLU())
        self.conv6 = nn.Sequential(nn.Conv1d(n_conv_filters, n_conv_filters, kernel_size=3, padding=0), nn.ReLU(), nn.MaxPool1d(3))

        dimension = int((input_length - 96) / 27 * n_conv_filters)
        self.fc1 = nn.Sequential(nn.Linear(dimension, n_fc_neurons), nn.Dropout(0.5))
        self.fc2 = nn.Sequential(nn.Linear(n_fc_neurons, n_fc_neurons), nn.Dropout(0.5))
        self.fc3 = nn.Linear(n_fc_neurons, n_classes)

        if n_conv_filters == 256 and n_fc_neurons == 1024:
            self._create_weights(mean=0.0, std=0.05)
        elif n_conv_filters == 1024 and n_fc_neurons == 2048:
            self._create_weights(mean=0.0, std=0.02)

    def _create_weights(self, mean=0.0, std=0.05):
        for module in self.modules():
            if isinstance(module, nn.Conv1d) or isinstance(module, nn.Linear):
                module.weight.data.normal_(mean, std)

    def forward(self, input):
        input = input.transpose(1, 2)
        output = self.conv1(input)
        output = self.conv2(output)
        output = self.conv3(output)
        output = self.conv4(output)
        output = self.conv5(output)
        output = self.conv6(output)

        output = output.view(output.size(0), -1)
        output = self.fc1(output)
        output = self.fc2(output)
        output = self.fc3(output)

        return output
## Evaluation on **Text Classification Tasks**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Classes</th>
<th>Train samples</th>
<th>Test samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG’s News</td>
<td>4</td>
<td>120,000</td>
<td>7,600</td>
</tr>
<tr>
<td>Sogou News</td>
<td>5</td>
<td>450,000</td>
<td>60,000</td>
</tr>
<tr>
<td>DBPedia</td>
<td>14</td>
<td>560,000</td>
<td>70,000</td>
</tr>
<tr>
<td>Yelp Review Polarity</td>
<td>2</td>
<td>560,000</td>
<td>38,000</td>
</tr>
<tr>
<td>Yelp Review Full</td>
<td>5</td>
<td>650,000</td>
<td>50,000</td>
</tr>
<tr>
<td>Yahoo! Answers</td>
<td>10</td>
<td>1,400,000</td>
<td>60,000</td>
</tr>
<tr>
<td>Amazon Review Full</td>
<td>5</td>
<td>3,000,000</td>
<td>650,000</td>
</tr>
<tr>
<td>Amazon Review Polarity</td>
<td>2</td>
<td>3,600,000</td>
<td>400,000</td>
</tr>
</tbody>
</table>
AG News
Deep Learning

End of Lecture

09 - Convolutional Neural Networks