Recap: Multilayer Neural Networks

\[ z^{(0)} = x \quad \text{(input)} \]
\[ a_{j}^{(l)} = \sum_{i} W_{ij}^{(l)} z_{i}^{(l-1)} \quad l = 1, ..., L \]
\[ z_{j}^{(l)} = \sigma(a_{k}^{l}) \quad l = 1, ..., L \]

▶ Training: Parameters \( \Theta = (W^{(1)}, W^{(2)}, ..., W^{(L)}) \)
regression: (squared loss) vs classification (cross-entropy loss)

\[
\min_{\Theta} \sum_{n=1}^{N} (y_{n} - f(x_{n}))^{2} \quad \min_{\Theta} -\sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \log f_{k}(x_{n})
\]
Training Multilayer Neural Networks

- **Training:** Parameters $\Theta = (W^{(1)}, W^{(2)}, ..., W^{(L)})$
- Regression: (squared loss) vs classification (cross-entropy loss)

\[
\min_{\Theta} \sum_{n=1}^{N} (y_n - f(x_n))^2 \quad \min_{\Theta} - \sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \log f_k(x_n)
\]

- **Gradient Descent**
  \[
  \Theta_{t+1} = \Theta_{t+1} - \sum_{i=1}^{n} \frac{\partial}{\partial \Theta} R_n(\Theta)
  \]

- **Stochastic Gradient Descent**
  \[
  \Theta_{t+1} = \Theta_{t+1} - \frac{\partial}{\partial \Theta} R_{n_t}(\Theta)
  \]
  where $n_t$ is a random index

- **Non-convex optimization problem**
Computing derivatives: Backpropagation Algorithm

\[ z^{(0)} = x \quad \text{(input)} \]

\[ a_j^{(l)} = \sum_i W_{ij}^{(l)} z_i^{(l-1)} \quad l = 1, \ldots, L \]

\[ z_j^{(l)} = \sigma(a_k^l) \quad l = 1, \ldots, L \]

\[
\min_{\Theta} \sum_{n=1}^{N} \left( y_n - f(x_n) \right)^2
\]

\[
\frac{\partial R_n(\Theta)}{\partial W_{ij}^{(l)}} = \frac{\partial R_n(\Theta)}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial W_{ij}^{(l)}} = \delta_{nj}^{(l)} z_i^{(l-1)}
\]

where we defined

\[
\delta_{nj}^{(l)} \triangleq \frac{\partial R_n(\Theta)}{\partial a_j^{(l)}}
\]
Two layer networks are universal approximators

- Two layer networks (also called one hidden layer) can compose arbitrary functions to arbitrary precision.
- May require infinitely many neurons in the hidden layer.
- Deeper networks require **fewer** neurons for the same approximation error.
Overparametrization

- \((u)_+ := \max(0, u)\) (ReLU activation) denotes the positive part of a scalar
- Let \(y \in \pm 1\) denote training labels

Two layer scalar output ReLU network:

\[
    f(x) = \sum_{j=1}^{m} w^{(2)}_j (x^T w^{(1)}_j)_+ \]

- The equation \(f(x) = y\) (zero training loss) can have multiple solutions
- The numerical optimizer (gradient descent, stochastic gradient, momentum etc) may have an inductive bias
Convolutional Neural Networks (CNNs)

Gradient-based learning applied to document recognition. LeCun et al., 1998
Convolutional Neural Networks (CNNs)

Input image

C1: Convolutional Layer

No. of filters, \( n_c = 6 \)
Filter size, \( F = 5 \)
Padding, \( P = 0 \)
Stride, \( S = 1 \)

Trainable parameters
= Weight + Bias
= \( F \times F \times n_{c-1} \times n_{c-1} + n_{c-1} \)
= \( 5 \times 5 \times 1 \times 6 + 6 = 156 \)

Connections = \( 28 \times 28 \times 156 \)
= \( 122304 \)

32 x 32 x 1

28 x 28 x 6
Convolutional Neural Networks (CNNs)

S2: Pooling Layer

- No. of filters, $n_f = 6$
- Filter size, $F = 2$
- Padding, $P = 0$
- Stride, $S = 2$

Trainable parameters

= (coefficient + bias) x filters
= $(1 + 1) \times 6$
= $12$

Connections

= $14 \times 14 \times 30$
= $5880$

28 x 28 x 6

14 x 14 x 6
## 2D Convolution

<table>
<thead>
<tr>
<th>Operation</th>
<th>Filter</th>
<th>Convolved Image</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Identity</strong></td>
<td>$\begin{bmatrix} 0 &amp; 0 &amp; 0 \ 0 &amp; 1 &amp; 0 \ 0 &amp; 0 &amp; 0 \end{bmatrix}$</td>
<td><img src="image1.png" alt="Image" /></td>
</tr>
<tr>
<td><strong>Edge detection</strong></td>
<td>$\begin{bmatrix} 1 &amp; 0 &amp; -1 \ 0 &amp; 0 &amp; 0 \ -1 &amp; 0 &amp; 1 \end{bmatrix}$</td>
<td><img src="image2.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>$\begin{bmatrix} 0 &amp; 1 &amp; 0 \ 1 &amp; -4 &amp; 1 \ 0 &amp; 1 &amp; 0 \end{bmatrix}$</td>
<td><img src="image3.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>$\begin{bmatrix} -1 &amp; -1 &amp; -1 \ -1 &amp; 8 &amp; -1 \ -1 &amp; -1 &amp; -1 \end{bmatrix}$</td>
<td><img src="image4.png" alt="Image" /></td>
</tr>
<tr>
<td><strong>Sharpen</strong></td>
<td>$\begin{bmatrix} 0 &amp; -1 &amp; 0 \ -1 &amp; 5 &amp; -1 \ 0 &amp; -1 &amp; 0 \end{bmatrix}$</td>
<td><img src="image5.png" alt="Image" /></td>
</tr>
<tr>
<td><strong>Box blur</strong></td>
<td>$\frac{1}{9} \begin{bmatrix} 1 &amp; 1 &amp; 1 \ 1 &amp; 1 &amp; 1 \ 1 &amp; 1 &amp; 1 \end{bmatrix}$</td>
<td><img src="image6.png" alt="Image" /></td>
</tr>
<tr>
<td>(normalized)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Gaussian blur</strong></td>
<td>$\frac{1}{16} \begin{bmatrix} 1 &amp; 2 &amp; 1 \ 2 &amp; 4 &amp; 2 \ 1 &amp; 2 &amp; 1 \end{bmatrix}$</td>
<td><img src="image7.png" alt="Image" /></td>
</tr>
<tr>
<td>(approximation)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Nonlinearity: Rectified Linear Unit

slide credit: R. Fergus
MNIST digit classification using LeNet

Imagenet Dataset

ImageNet Challenge

- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.
Revolution of Depth

ImageNet Challenge

ILSVRC’16 winner: Error rate 2.991%

Super-human precision

* Human-level performance: 5.1%
Alexnet
Alexnet Filter Weights

slide credit: H Scholte et al.
Which input images activate a particular neuron?

- Find an image that maximizes the activation of a single neuron (D. Wei et al.)
Adversarial Examples

Panda!

Gibbon class gradient

Gibbon!

Panda

Adversarial example
Signals and Dense Neural Nets

- Human Activity Recognition

![Graphs showing data from different axes for human activity recognition.](image-url)
In order to feed the data into our neural network it must be reshaped in such a way that each person has multiple two dimensional records which hold 80 time slices for each of the three accelerometer readings (x, y, z axis). One record is associated to one label (e.g. Walking). Those records are fed into the neural network during training.

Input layer is a vector with 240 elements (flattened representation of 80 time slices for 3 accelerometer readings each).

Three hidden layers with 100 nodes each. Layers are fully connected. One additional layer upfront for reshaping the input into 80x3 matrix and a Softmax activation layer as the final layer.

Output layer with the six available labels. The network will provide the probability regarding each output class. Probabilities will add up to 1.
Activity Recognition Performance

- 4 dense layers, total parameters: 68,606
- Accuracy on test data: 0.76

1D Convolutional Nets

- 6 layers, total parameters: 520,486
  \[ \text{Conv}(\text{length}=10, \text{filters}=100) \rightarrow \text{Conv}(10,100) \rightarrow \text{MaxPool}(3) \rightarrow \text{Conv}(10,160) \rightarrow \text{Conv}(10,160) \rightarrow \text{GlobalAveragePooling} \]
- Accuracy on test data: 0.92
- Dense (non-convolutional) net accuracy was 0.76 with 68,606 parameters

2D Convolutional Networks for Signal Classification

- Given 1D signals $x_1[n], x_2[n], \ldots$
- train neural networks on two dimensional representations of the signals
- examples: spectrogram, wavelet transforms...
Short-time Fourier Transform

- window signal

\[ w[m] = \begin{cases} 
0 & m < 0, \ m \geq L \\
1 & 0 \leq m \leq L - 1 
\end{cases} \]

- Short Time Fourier Transform (STFT)

\[
X[n, k] = \sum_{m=0}^{L-1} x[n + m] w[m] e^{-j(2\pi/N)km}, \quad 0 \leq k \leq N - 1.
\]
Short-time Fourier Transform

- window signal
  
  e.g. $w[m] = \begin{cases} 
  0 & m < 0, m \geq L \\
  1 & 0 \leq m \leq L - 1 
  \end{cases}$

- Short Time Fourier Transform (STFT)

  \[
  X[n, k] = \sum_{m=0}^{L-1} x[n + m]w[m]e^{-j(2\pi/N)km}, \quad 0 \leq k \leq N - 1.
  \]

- Continuous Frequency STFT

  \[
  X[n, \lambda] = \sum_{m=0}^{L-1} x[n + m]w[m]e^{-j\lambda m}.
  \]
Speech Commands Dataset

```matlab
load('commandNet.mat')
```

The network is trained to recognize the following speech commands:

- "yes"
- "no"
- "up"
- "down"
- "left"
- "right"
- "on"
- "off"
- "stop"
- "go"

Load a short speech signal where a person says "stop".

```matlab
[x,fs] = audioread('stop_command.flac');
```

Listen to the command.

```matlab
sound(x,fs)
```
<table>
<thead>
<tr>
<th>Label</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>down</td>
<td>1842</td>
</tr>
<tr>
<td>go</td>
<td>1861</td>
</tr>
<tr>
<td>left</td>
<td>1839</td>
</tr>
<tr>
<td>no</td>
<td>1853</td>
</tr>
<tr>
<td>off</td>
<td>1839</td>
</tr>
<tr>
<td>on</td>
<td>1864</td>
</tr>
<tr>
<td>right</td>
<td>1852</td>
</tr>
<tr>
<td>stop</td>
<td>1885</td>
</tr>
<tr>
<td>unknown</td>
<td>6483</td>
</tr>
<tr>
<td>up</td>
<td>1843</td>
</tr>
<tr>
<td>yes</td>
<td>1860</td>
</tr>
</tbody>
</table>
Spectrograms

segmentDuration = 1;
frameDuration = 0.025;
hopDuration = 0.010;
numBands = 40;

Compute the spectrograms for the training, validation, and test sets by using the supporting function `speechSpectrograms`. The `speechSpectrograms` function uses `auditorySpectrogram` for the spectrogram calculations. To obtain data with a smoother distribution, take the logarithm of the spectrograms using a small offset `epsil`.

```
epsil = 1e-6;
XTrain = speechSpectrograms(adsTrain,segmentDuration,frameDuration,hopDuration,numBands);
XTrain = log10(XTrain + epsil);

XValidation = speechSpectrograms(adsValidation,segmentDuration,frameDuration,hopDuration,numBands);
XValidation = log10(XValidation + epsil);

XTest = speechSpectrograms(adsTest,segmentDuration,frameDuration,hopDuration,numBands);
XTest = log10(XTest + epsil);
```
Spectrograms

![Spectrogram of the word 'yes']
Spectrograms
Training and Validation

Training Label Distribution

Validation Label Distribution
Training and Validation

```plaintext
numF = 12;
layers = [
    imageInputLayer([numHops numBands])
    convolution2dLayer(3,numF,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(3,'Stride',2,'Padding','same')
    convolution2dLayer(3,2*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(3,'Stride',2,'Padding','same')
    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(3,'Stride',2,'Padding','same')
    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer
convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer
    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer([timePoolSize,1])
    dropoutLayer(dropoutProb)
    fullyConnectedLayer(numClasses)
    softmaxLayer
    weightedClassificationLayer(classWeights)];
```
Training the network

Train Network
Specify the training options. Use the Adam optimizer with a mini-batch size of 128. Train for 25 epochs and reduce the learning rate by a factor of 10 after 20 epochs.

```matlab
miniBatchSize = 128;
validationFrequency = floor(numel(YTrain)/miniBatchSize);
options = trainingOptions('adam', ...'
  'InitialLearnRate',3e-4, ...
  'MaxEpochs',25, ...
  'MiniBatchSize',miniBatchSize, ...
  'Shuffle','every-epoch', ...
  'Plots','training-progress', ...
  'Verbose',false, ...
  'ValidationData',{XValidation,YValidation}, ...
  'ValidationFrequency',validationFrequency, ...
  'LearnRateSchedule','piecewise', ...
  'LearnRateDropFactor',0.1, ...
  'LearnRateDropPeriod',20);

Train the network. If you do not have a GPU, then training the network can take time.

trainedNet = trainNetwork(augimdsTrain,layers,options);
```
## Confusion Matrix for Validation Data

<table>
<thead>
<tr>
<th>True Class</th>
<th>yes</th>
<th>no</th>
<th>up</th>
<th>down</th>
<th>left</th>
<th>right</th>
<th>on</th>
<th>off</th>
<th>stop</th>
<th>go</th>
<th>unknown</th>
<th>background</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>252</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>no</td>
<td>256</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>up</td>
<td>245</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>down</td>
<td>13</td>
<td>238</td>
<td>7</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>left</td>
<td>2</td>
<td>1</td>
<td>242</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>right</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>249</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>on</td>
<td>2</td>
<td>241</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>off</td>
<td>9</td>
<td>2</td>
<td>2</td>
<td>242</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stop</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>go</td>
<td>11</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>232</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unknown</td>
<td>2</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>4</td>
<td>15</td>
<td>771</td>
<td>7</td>
</tr>
</tbody>
</table>

**Predicted Class**

<table>
<thead>
<tr>
<th>yes</th>
<th>no</th>
<th>up</th>
<th>down</th>
<th>left</th>
<th>right</th>
<th>on</th>
<th>off</th>
<th>stop</th>
<th>go</th>
<th>unknown</th>
<th>background</th>
</tr>
</thead>
<tbody>
<tr>
<td>98.4%</td>
<td>86.8%</td>
<td>89.7%</td>
<td>94.8%</td>
<td>96.0%</td>
<td>96.1%</td>
<td>94.5%</td>
<td>92.4%</td>
<td>97.9%</td>
<td>89.2%</td>
<td>96.4%</td>
<td>96.3%</td>
</tr>
<tr>
<td>1.6%</td>
<td>13.2%</td>
<td>10.3%</td>
<td>5.2%</td>
<td>4.0%</td>
<td>3.9%</td>
<td>5.5%</td>
<td>7.6%</td>
<td>2.1%</td>
<td>10.8%</td>
<td>3.6%</td>
<td>3.7%</td>
</tr>
</tbody>
</table>