Typical goal of machine learning

**Input**
- Images/video
- Audio
- Text

**Output**
- Label: “Motorcycle”
- Suggest tags
- Image search
- Speech recognition
- Music classification
- Speaker identification
- Web search
- Anti-spam
- Machine translation
Typical goal of machine learning

**Input**
- images/video
- audio
- text

**Output**
- Label: “Motorcycle”
- Suggest tags
- Image search
  - ... 
- Speech recognition
- Music classification
- Speaker identification
  - ... 
- Web search
- Anti-spam
- Machine translation
  - ...

Feature engineering: most time consuming!
Our goal in object classification
Why is this hard?

You see this:

But the camera sees this:

<table>
<thead>
<tr>
<th>194</th>
<th>210</th>
<th>201</th>
<th>212</th>
<th>199</th>
<th>213</th>
<th>215</th>
<th>195</th>
<th>178</th>
<th>158</th>
<th>182</th>
<th>209</th>
</tr>
</thead>
<tbody>
<tr>
<td>180</td>
<td>189</td>
<td>190</td>
<td>221</td>
<td>209</td>
<td>205</td>
<td>191</td>
<td>167</td>
<td>147</td>
<td>115</td>
<td>129</td>
<td>163</td>
</tr>
<tr>
<td>114</td>
<td>126</td>
<td>140</td>
<td>188</td>
<td>176</td>
<td>165</td>
<td>152</td>
<td>140</td>
<td>170</td>
<td>106</td>
<td>78</td>
<td>88</td>
</tr>
<tr>
<td>87</td>
<td>103</td>
<td>115</td>
<td>154</td>
<td>143</td>
<td>142</td>
<td>149</td>
<td>153</td>
<td>173</td>
<td>101</td>
<td>57</td>
<td>57</td>
</tr>
<tr>
<td>102</td>
<td>112</td>
<td>106</td>
<td>131</td>
<td>122</td>
<td>138</td>
<td>152</td>
<td>147</td>
<td>128</td>
<td>84</td>
<td>58</td>
<td>66</td>
</tr>
<tr>
<td>94</td>
<td>95</td>
<td>79</td>
<td>104</td>
<td>105</td>
<td>124</td>
<td>129</td>
<td>113</td>
<td>107</td>
<td>87</td>
<td>69</td>
<td>67</td>
</tr>
<tr>
<td>68</td>
<td>71</td>
<td>69</td>
<td>98</td>
<td>89</td>
<td>92</td>
<td>98</td>
<td>95</td>
<td>89</td>
<td>88</td>
<td>76</td>
<td>67</td>
</tr>
<tr>
<td>41</td>
<td>56</td>
<td>68</td>
<td>99</td>
<td>63</td>
<td>45</td>
<td>60</td>
<td>82</td>
<td>58</td>
<td>76</td>
<td>75</td>
<td>65</td>
</tr>
<tr>
<td>20</td>
<td>43</td>
<td>69</td>
<td>75</td>
<td>56</td>
<td>41</td>
<td>51</td>
<td>73</td>
<td>55</td>
<td>70</td>
<td>63</td>
<td>44</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>57</td>
<td>69</td>
<td>75</td>
<td>75</td>
<td>73</td>
<td>74</td>
<td>53</td>
<td>68</td>
<td>59</td>
<td>37</td>
</tr>
<tr>
<td>72</td>
<td>59</td>
<td>53</td>
<td>66</td>
<td>84</td>
<td>92</td>
<td>84</td>
<td>74</td>
<td>57</td>
<td>72</td>
<td>63</td>
<td>42</td>
</tr>
<tr>
<td>67</td>
<td>61</td>
<td>58</td>
<td>65</td>
<td>75</td>
<td>78</td>
<td>76</td>
<td>73</td>
<td>59</td>
<td>75</td>
<td>69</td>
<td>50</td>
</tr>
</tbody>
</table>
Pixel-based representation

Input

Raw image

Pixel 1

Pixel 2

Motorbikes
Non-Motorbikes

Learning algorithm
Pixel-based representation

Input

Raw image

Motorbikes

“Non”-Motorbikes

Learning algorithm

pixel 1

pixel 2
Pixel-based representation

Input

Raw image

Motorbikes
“Non”-Motorbikes

Learning algorithm
What we want

Input

Raw image

Feature representation

E.g., Does it have Handlebars? Wheels?

Motorbikes
"Non"-Motorbikes

Learning algorithm

pixel 2

pixel 1

Features

Wheels

Handlebars
Some feature representations

SIFT

Spin image

HoG

RIFT

Textons

GLOH
Some feature representations

Coming up with features is often difficult, time-consuming, and requires expert knowledge.
Feature Engineering?

Feature Learning?
Representation Learning

• Is there some way to extract **meaningful features** from data in a *supervised* or *unsupervised* manner?
Representation Learning

• Is there some way to extract meaningful features from data in a supervised or unsupervised manner?

• Biologically inspired systems
  – to make the computer more robust, intelligent, and learn, ...
Representation Learning

• Is there some way to extract meaningful features from data in a supervised or unsupervised manner?

• Biologically inspired systems
  – to make the computer more robust, intelligent, and learn, ...
  – Model our systems after the brain!
Representation Learning

• Is there some way to extract **meaningful features** from data in a *supervised* or *unsupervised* manner?

• Biologically inspired systems
  – to make the computer more robust, intelligent, and learn, ...
  – Model our systems after *the brain*!
    • Brain interprets imprecise information from the senses at an incredibly rapid rate!
    • It discerns a whisper in a noisy room, a face in a dimly lit alley, and hidden agenda in a political statement.
A common logic to seeing cats and cosmos

There may be a universal logic to how physicists, computers and brains tease out important features from among other irrelevant bits of data.
Recap: Types of Learning

• **Supervised (inductive) learning**
  – Training data includes desired outputs

• **Unsupervised learning**
  – Training data does not include desired outputs

• **Semi-supervised learning**
  – Training data includes a few desired outputs

• **Reinforcement learning**
  – Rewards from sequence of actions
Neurons in the Brain

- Brain is composed of neurons.
- A neuron receives input from other neurons (generally thousands) from its synapses.
- Inputs are approximately summed.
- When the input exceeds a threshold the neuron sends an electrical spike that travels from the body, down the axon, to the next neuron(s).
Background in Neural Nets (NN)

• Neural Nets can be:
  – Biological Models
  – Artificial Models

• Desire to produce artificial systems capable of sophisticated computations similar to human brain!
Brain Analogy of NN: Brain is a remarkable Computer

• The brain is composed of a **mass of interconnected neurons**
  – each neuron is connected to many other neurons
• Neurons transmit signals to each other
• Whether a signal is transmitted is an **all-or-nothing** event (the electrical potential in the cell body of the neuron is **thresholded**)
• Whether a signal is sent, depends on the **strength of the bond** (synapse) between two neurons
Brain Analogy of NN: Brain is a remarkable Computer

- The brain is composed of a mass of interconnected neurons
  - each neuron is connected to many other neurons
- Neurons transmit signals to each other
- Whether a signal is transmitted is an all-or-nothing event (the electrical potential in the cell body of the neuron is thresholded)
- Whether a signal is sent, depends on the strength of the bond (synapse) between two neurons

- 200 billion neurons, 32 trillion synapses
- Element size: $10^{-6}$ m
- Energy use: 25W
- Processing speed: 100 Hz
- Parallel, Distributed
- Fault Tolerant
- Learns: Yes
- Intelligent/Conscious: Usually

- 1 billion bytes RAM but trillions of bytes on disk
- Element size: $10^{-9}$ m
- Energy watt: 30-90W (CPU)
- Processing speed: $10^9$ Hz
- Serial, Centralized
- Generally not Fault Tolerant
- Learns: Some
- Intelligent/Conscious: Generally No
Use of NNs

• High-Frequency Trading (HFT)
• Credit Applications
• Data Center Management
• Robotics
• Medical Monitoring
• Computer Vision
• Data Analysis
• Predictive Systems
• Artificial Systems
• ......
Computational Implementation of the Neural Activation Function
Computational Implementation of the Neural Activation Function
Computational Implementation of the Neural Activation Function

Dendrites

input

Soma (cell body)

Axon

output
Computational Implementation of the Neural Activation Function

Dendrites

Soma (cell body)

Axon

Simple activation function (sum of individual inputs):

\[ n_j = \sum_{i} x_i w_{ij} \]

Activation value (sigmoid / logistic):

\[ y_j = \frac{1}{1 + e^{-n_j}} \]
Why Sigmoid?

- It is a standard function in NN.
- It transforms the net input value into an activation value $y$, which is then sent on to other units.
Background on Logit

Assumes the following functional form for $P(Y | X)$:

$$P(Y = 1 | X) = \frac{1}{1 + \exp(-(w_0 + \sum_i w_i X_i))}$$

Logistic function applied to a linear function of the data

Logistic function (or Sigmoid):

$$\frac{1}{1 + \exp(-z)}$$
Background on Logit

Logistic Regression is a Linear Classifier!

Assumes the following functional form for \( P(Y|X) \):

\[
P(Y = 1|X) = \frac{1}{1 + \exp(-(w_0 + \sum_i w_i X_i))}
\]

Decision boundary:

\[
0 \geq w_0 + \sum_i w_i X_i
\]

(Linear Decision Boundary)
Perceptron

- is the basic processing element. It has inputs that may come from the environment or may be the outputs of other perceptrons.
Perceptron

- is the basic processing element. It has inputs that may come from the environment or may be the outputs of other perceptrons.

![Perceptron Diagram]

\[ y = w_0 x_0 + w_1 x_1 + \ldots + w_d x_d \]

\[ x_0 = +1 \]
Perceptron

\[ y = \sum_{j=1}^{d} w_j x_j + w_0 \]

is the intercept value to make the model more general

Connection weights

Bias unit

Output unit

Input units
**Perceptron**

\[ y = \sum_{j=1}^{d} w_j x_j + w_0 \]

is the intercept value to make the model more general

\[ w = [w_0, w_1, \ldots, w_d]^T \]

\[ x = [x_0, x_1, \ldots, x_d]^T \]

**Connection weights**

**Bias unit**

**Output unit**

**Input units**
Perceptron-Nonlinear Statistical Models

- A NN is a two-stage regression or classification model
Perceptron-Nonlinear Statistical Models

- A NN is a two-stage **regression** or **classification** model

Typically $K=1$ for regression
And there is only one output
Unit at the top

For classification, $K>1$
Perceptron-Nonlinear Statistical Models

- A NN is a two-stage **regression** or **classification** model

Typically $K=1$ for regression
And there is only one output
Unit at the top

For classification, $K>1$
**Perceptron-Nonlinear Statistical Models**

- A NN is a two-stage **regression** or **classification** model

Typically $K=1$ for regression
And there is only one output Unit at the top

For classification, $K>1$
Perceptron-Nonlinear Statistical Models

- A NN is a two-stage **regression** or **classification** model

Each target
Class is coded as 0 -1

Hidden layers
(derived features)

Typically \( K = 1 \) for regression
And there is only one output
Unit at the top

For classification, \( K > 1 \)

\[
Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), \quad m = 1, \ldots, M
\]
Perceptron-Nonlinear Statistical Models

• A NN is a two-stage **regression** or **classification** model

Typically $K=1$ for regression
And there is only one output
Unit at the top

For classification, $K>1$

\[
Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), \quad m = 1, \ldots, M
\]

\[
T_k = \beta_{0k} + \beta_k^T Z, \quad k = 1, \ldots, K
\]
Perceptron-Nonlinear Statistical Models

• A NN is a two-stage regression or classification model

Typically $K=1$ for regression
And there is only one output
Unit at the top

For classification, $K>1$

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), \ m = 1, \ldots, M$$
$$T_k = \beta_{0k} + \beta_k^T Z, \ k = 1, \ldots, K$$
$$f_k(X) = g_k(T), \ k = 1, \ldots, K$$
Perceptron-Nonlinear Statistical Models

• A NN is a two-stage regression or classification model

For regression, output function is
\[ g_k(T) = T_k \]

For classification, it is softmax func:
\[ g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}} \]

\[ Z_m = \sigma(\alpha_{0_m} + \alpha_{m}^T X), \ m = 1, \ldots, M \]
\[ T_k = \beta_{0_k} + \beta_{k}^T Z, \ k = 1, \ldots, K \]
\[ f_k(X) = g_k(T), \ k = 1, \ldots, K \]
\[ \sigma(v) = 1/(1 + e^{-v}) \]
Fitting NNs

- The NN model has unknown parameters, often called weights, and we seek values for them that make the model fit the training data well.

\[
\{\alpha_{0m}, \alpha_m; m = 1, 2, \ldots, M\} \quad M(p + 1) \text{ weights},
\]
\[
\{\beta_{0k}, \beta_k; k = 1, 2, \ldots, K\} \quad K(M + 1) \text{ weights}.
\]
Fitting NNs

• The NN model has unknown parameters, often called weights, and we seek values for them that make the model fit the training data well.

\[
\{a_0, a_m; \ m = 1, 2, \ldots, M\} \ M(p + 1) \text{ weights,}
\]

\[
\{\beta_0, \beta_k; \ k = 1, 2, \ldots, K\} \ K(M + 1) \text{ weights.}
\]

• For regression, we use sum-of-squared errors as our measure of fit:

\[
R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2.
\]
Fitting NNs

- The NN model has unknown parameters, often called weights, and we seek values for them that make the model fit the training data well.

\[
\{\alpha_{0m}, \alpha_m; \ m = 1, 2, \ldots, M\} \quad M(p + 1) \text{ weights,}
\]
\[
\{\beta_{0k}, \beta_k; \ k = 1, 2, \ldots, K\} \quad K(M + 1) \text{ weights.}
\]

- For regression, we use sum-of-squared errors as our measure of fit:

\[
R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2.
\]

- For classification, we use either squared error or cross-entropy:

\[
R(\theta) = - \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log f_k(x_i),
\]

and the corresponding classifier is

\[
G(x) = \arg\max_k f_k(x).
\]
Fitting NN: Back-propagation

• Typically we don’t want the global minimizer of $R(.)$, as this likely to be an overfit solution.

• Instead, some regularization is needed:
  – This is achieved directly through a penalty term, or indirectly by early stopping.
Fitting NN: Back-propagation

• Typically we don’t want the global minimizer of \( R(.) \), as this likely to be an overfit solution.

• Instead, some regularization is needed:
  — This is achieved directly through a penalty term, or indirectly by early stopping.

• The generic approach for minimizing \( R(.) \) is by gradient-descent, called back-propagation.
Fitting NN: Back-propagation

• Typically we don’t want the global minimizer of $R(.)$, as this likely to be an overfit solution.
• Instead, some regularization is needed:
  – This is achieved directly through a penalty term, or indirectly by early stopping.
• The generic approach for minimizing $R(.)$ is by gradient-descent, called back-propagation.
• Due to compositional form of the model, the gradient can be easily derived using the chain rule for differentiation. This can be computed by a forward and backward sweep over the network, keeping track only of quantities local to each unit.
Back-propagation

\[ z_{m_i} = \sigma(\alpha_{0m} + \alpha_m^T x_i), \quad z_i = (z_{1i}, z_{2i}, \ldots, z_{Mi}) \]

\[
R(\theta) \equiv \sum_{i=1}^{N} R_i = \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2,
\]
Back-propagation

\[ z_{m_i} = \sigma(\alpha_{0m} + \alpha^T_{m} x_i), \quad z_i = (z_{1i}, z_{2i}, \ldots, z_{Mi}) \]

\[
R(\theta) \equiv \sum_{i=1}^{N} R_i = \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2,
\]

Taking the derivative of \( R \) w.r.t. weight parameters

\[
\frac{\partial R_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i))g'_k(\beta_k^T z_i)z_{mi},
\]

\[
\frac{\partial R_i}{\partial \alpha_{ml}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i))g'_k(\beta_k^T z_i)\beta_{km}\sigma'(\alpha^T_{m} x_i)x_{il}.
\]
Back-propagation

• Given these derivatives, a gradient descent update at the (r+1)st iteration has the form

\[
\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}},
\]

\[
\alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}},
\]
Back-propagation

• Given these derivatives, a gradient descent update at the \( (r+1) \)st iteration has the form

\[
\begin{align*}
\beta_{km}^{(r+1)} &= \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}, \\
\alpha_{ml}^{(r+1)} &= \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}}.
\end{align*}
\]
Back-propagation

- Given these derivatives, a gradient descent update at the (r+1)st iteration has the form:

\[ \beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}, \]

\[ \alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}}, \]

Learning rate

\[ \frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi}, \]

\[ \frac{\partial R_i}{\partial \alpha_{ml}} = s_{mi} x_{il}. \]

Errors from the current model at the output units

Errors from the current model at the hidden units
Back-propagation

• Given these derivatives, a gradient descent update at the \((r+1)\)st iteration has the form

\[
\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}},
\]

\[
\alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}},
\]

Backpropagation equations

\[
\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki}z_{mi},
\]

\[
\frac{\partial R_i}{\partial \alpha_{ml}} = s_{mi}x_{il}.
\]

Batch learning
Back-propagation-Delta Rule

1. **Forward pass**: calculate \( f(.) \) and obtain error rates,
2. **Backward pass**: calculate \( s \) from error rates, and update gradient equations.
Back-propagation-Delta Rule

1. **Forward pass:** calculate \( f(\cdot) \) and obtain error rates,
2. **Backward pass:** calculate \( s \) from error rates, and update gradient equations.

**If the weights are near zero:** operative part of sigmoid is roughly linear, hence NN collapses into an approximately linear model.

Often, starting values are chosen to be random values near zero; hence the model starts nearly linear and becomes nonlinear as the weights increase.
What is wrong with back-propagation?

- It requires labeled training data.
  - Almost all data is unlabeled.
- The learning time does not scale well
  - It is very slow in networks with multiple hidden layers.
- It can get stuck in poor local optima.
Avoid Overfitting

- NNs usually have too many weights, and will overfit the data at the global minimum of R.
Avoid Overfitting

• NNs usually have too many weights, and will overfit the data at the global minimum of R.

• **Weight decay:** is analogous to ridge regression used for linear models for avoiding overfit.
Avoid Overfitting

• NNs usually have too many weights, and will overfit the data at the global minimum of R.

• **Weight decay:** is analogous to ridge regression used for linear models for avoiding overfit.

• A new functional is to be minimized:

\[ R(\theta) + \lambda J(\theta) \]
Avoid Overfitting

• NNs usually have too many weights, and will overfit the data at the global minimum of R.

• **Weight decay:** is analogous to ridge regression used for linear models for avoiding overfit.

• A new functional is to be minimized:

\[ R(\theta) + \lambda J(\theta) \]

where

\[ J(\theta) = \sum_{km} \beta^2_{km} + \sum_{ml} \alpha^2_{ml} \]
Avoid Overfitting

• NNs usually have too many weights, and will overfit the data at the global minimum of $R$.

• **Weight decay:** is analogous to ridge regression used for linear models for avoiding overfit.

• A new functional is to be minimized:

$$R(\theta) + \lambda J(\theta)$$

where

$$J(\theta) = \sum_{km} \beta_{km}^2 + \sum_{ml} \alpha_{ml}^2$$

where $\lambda$ is tuning parameter $>0$, estimated by cross validation.
Perceptron Training Algorithm

For $i = 1, \ldots, K$
For $j = 0, \ldots, d$
    $w_{ij} \leftarrow \text{rand}(-0.01, 0.01)$
Repeat
    For all $(x^t, r^t) \in \mathcal{X}$ in random order
    For $i = 1, \ldots, K$
        $o_i \leftarrow 0$
    For $j = 0, \ldots, d$
        $o_i \leftarrow o_i + w_{ij}x_j^t$
    For $i = 1, \ldots, K$
        $y_i \leftarrow \exp(o_i) / \sum_k \exp(o_k)$
    For $i = 1, \ldots, K$
        For $j = 0, \ldots, d$
            $w_{ij} \leftarrow w_{ij} + \eta(r_i^t - y_i)x_j^t$
Until convergence

- $x, r \rightarrow \text{input, output}$
- $y = \text{sigmoid}(w^Tx)$
Ex: Learning Boolean Functions

\[ y = s(x_1 + x_2 - 1.5) \]

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Question?

- What is the Boolean function that the following perceptron calculates?

\[ y = s(-x + 0.5) \]
Question?

• What is the Boolean function that the following perceptron calculates?

\[ y = s(-x + 0.5) \]

NOT of x
Question?

• What is the Boolean function that the following perceptron calculates?

\[ y = s(x_1 + x_2 - 0.5) \]
Question?

• What is the Boolean function that the following perceptron calculates?

\[ y = s(x_1 + x_2 - 0.5) \]

x1 OR x2
Key Ideas in Neural Nets

1. Learn features from data
Key Ideas in Neural Nets

1. Learn features from data
2. Use differentiable functions that produce features efficiently
Key Ideas in Neural Nets

1. Learn features from data
2. Use differentiable functions that produce features efficiently
3. End-to-End learning: no distinction between feature extractor and classifier
Key Ideas in Neural Nets

1. Learn features from data
2. Use differentiable functions that produce features efficiently
3. End-to-End learning: no distinction between feature extractor and classifier
When Considering NNs?

• Input is
  – high-dimensional
  – discrete or real-valued
    • e.g., raw sensor inputs
  – noisy
• Long training times
• Form of target function is unknown
• Human readability is unimportant
• Especially good for complex recognition problems
  – Speech recognition
  – Image classification
  – Financial prediction
ZIP-CODE example

- Character recognition task: classification of handwritten numerals.
- It has remained a benchmark problem in the field for many years.

Each image is 16x16 8-bit grayscale representation of handwritten digit.
Net-1: No hidden layer, equivalent to multinomial logistic regression.

Net-2: One hidden layer, 12 hidden units fully connected.

Net-3: Two hidden layers locally connected.

Net-4: Two hidden layers, locally connected with weight sharing.

Net-5: Two hidden layers, locally connected, two levels of weight sharing.
5 networks used in the ZIP-CODE example
Neural Network trained to drive a car!

Weights to output units from the hidden unit

Weights of each pixel for one hidden unit
References and Slice Credits

• Rob Fergus, Deep Learning Tutorial CVPR 2012
• A. Torralba, MIT
• Aarti Singh & Barnabas Poczos (CMU)
• Hastie, Tibshirani, Friedman, The Elements of Statistical Learning.