Lecture 7: Neural networks
• Discussion this week: machine learning
• Reading:
  - Szeliski 5.3
  - Goodfellow Deep Feedforward Networks
• Start thinking about project
• PS2 due today – submit to gradescope and canvas
Today

• Brief history of neural networks
• Computation in neural networks
• Multi-layer perceptrons (for PS4)
• Estimating gradients (to be continued next class).
Limitations to linear classifiers

XOR

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Limitations to linear classifiers

Wrong!

Wrong!

$x_2$

$x_1$

$0$  $1$

$0$  $0$  $1$

$1$  $1$  $0$

XOR
Limitations to linear classifiers

Wrong!

Wrong!

Wrong!

\[ \begin{array}{c|c|}
   x_1 & 0 & 1 \\
   \hline
   0 & 0 & 1 \\
   1 & 1 & 0 \\
\end{array} \]

XOR
Goal: Non-linear decision boundary

\[ \begin{array}{c|c|c}
 x_2 & 0 & 1 \\
 \hline
 0 & 0 & 1 \\
 1 & 1 & 0 \\
\end{array} \]

XOR
Perceptron

• In 1957 Frank Rosenblatt invented the perceptron
• Computers at the time were too slow to run the perceptron, so Rosenblatt built a special purpose machine with adjustable resistors
• New York Times Reported: “The Navy revealed the embryo of an electronic computer that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence”
Perceptrons, expanded edition

An Introduction to Computational Geometry

By Marvin Minsky and Seymour A. Papert

Overview

Perceptrons - the first systematic study of parallelism in computation - has remained a classical work on threshold automata networks for nearly two decades. It marked a historical turn in artificial intelligence, and it is required reading for anyone who wants to understand the connectionist counterrevolution that is going on today.

Artificial-intelligence research, which for a time concentrated on the programming of ton Neumann computers, is swinging back to the idea that intelligence might emerge from the activity of networks of neuronlike entities. Minsky and Papert's book was the first example of a mathematical analysis carried far enough to show the exact limitations of a class of computing machines that could seriously be considered as models of the brain. Now the new developments in mathematical tools, the recent interest of physicists in the theory of disordered matter, the new insights into and psychological models of how the brain works, and the evolution of fast computers that can simulate networks of automata have given Perceptrons new importance.

Witnessing the swing of the intellectual pendulum, Minsky and Papert have added a new chapter in which they discuss the current state of parallel computers, review developments since the appearance of the 1972 edition, and identify new research directions related to connectionism. They note a central theoretical challenge facing connectionism: the challenge to reach a deeper understanding of how "objects" or "agents" with individuality can emerge in a network. Progress in this area would link connectionism with what the authors have called "society theories of mind."

Source: Isola, Torralba, Freeman
Based on slide by: Isola, Torralba, Freeman
enthusiasm

Perceptrons, 1958

PDP book, 1986

Minsky and Papert, 1972

Source: Isola, Torralba, Freeman
LeCun convolutional neural networks

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Demos:
http://yann.lecun.com/exdb/lenet/index.html

Source: Isola, Torralba, Freeman
Fig. 13. Examples of unusual, distorted, and noisy characters correctly recognized by LeNet-5. The grey-level of the output label represents the penalty (lighter for higher penalties).
Neural networks to recognize handwritten digits and human faces? yes

Neural networks for tougher problems? not really
Machine learning circa 2000

• Neural Information Processing Systems (NeurIPS), is a top conference on machine learning.

• For the 2000 conference:
  – title words predictive of paper acceptance: “Belief Propagation” and “Gaussian”.
  – title words predictive of paper rejection: “Neural” and “Network”.

Source: Isola, Torralba, Freeman
Perceptrons, 1958
Minsky and Papert, 1972
PDP book, 1986
Neural network winter, 2000

Source: Isola, Torralba, Freeman
Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

“AlexNet”

Got all the “pieces” right, e.g.,
• Trained on ImageNet
• 8 layer architecture (for reference: today we have architectures with 100+ layers)
• Allowed for multi-GP training

Source: Isola, Torralba, Freeman
28 years
Perceptrons, 1958
Minsky and Papert, 1972
PDP book, 1986
Neural net winter, 2000
Krizhevsky, Sutskever, Hinton, 2012

Source: Isola, Torralba, Freeman
What comes next?

Perceptrons, 1958

Minsky and Papert, 1972

PDP book, 1986

AI winter, 2000

Krizhevsky, Sutskever, Hinton, 2012

Source: Isola, Torralba, Freeman
Inspiration: Neurons

Image source: Khan academy
Inspiration: Hierarchical Representations

Best to treat as *inspiration*. The neural nets we’ll talk about aren’t very biologically plausible.
Goal: automatically learn a function that maps data from the input space to a feature space, i.e., "feature learning", rather than use hand-crafted features.
Computation in a neural net

Let's say we have some 1D input that we want to convert to some new feature space:

**Linear layer**

\[ y_j = \sum_i w_{ij} x_i \]

Adapted from: Isola, Torralba, Freeman
Computation in a neural net

Let's say we have some 1D input that we want to convert to some new feature space:

**Linear layer**

\[ y_j = \sum_i w_{ij} x_i + b_j \]

Adapted from: Isola, Torralba, Freeman
Computation in a neural net – Matrix Multiplication

\[ y_j = \sum_i w_{ij} x_i + b_j \quad \text{Vector of all input units} \]

\[ \sum_i w_{ij} x_i = x \cdot w_j = x^T w_j \quad \text{Vector of weights} \]

\[ y_j = x^T w_j + b_j \]

\[ \begin{bmatrix} x_1 & x_2 & \ldots & x_n \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} + b_j = \begin{bmatrix} x_1 & x_2 & \ldots & x_n & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \\ b_j \end{bmatrix} \]
Example: Linear Regression

\[ f_{w,b}(x) = x^T w + b \]
Computation in a neural net – Full Layer

\[ y = Wx + b \]

\[
\begin{bmatrix}
W_{11} & \cdots & W_{1n} \\
\vdots & \ddots & \vdots \\
W_{j1} & \cdots & W_{jn}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_n
\end{bmatrix}
+ 
\begin{bmatrix}
b_1 \\
b_2 \\
\vdots \\
b_j
\end{bmatrix} = 
\begin{bmatrix}
y_1 \\
y_2 \\
\vdots \\
y_j
\end{bmatrix}
\]

parameters of the model: \( \theta = \{W, b\} \)

Adapted from: Isola, Torralba, Freeman
Computation in a neural net – Full Layer

Linear layer

\[
\begin{array}{c}
\text{Input representation} \\
\hline
\bullet \quad \bullet \\
\bullet \quad \bullet \\
\bullet \quad \bullet \\
\bullet \quad \bullet \\
\bullet \quad \bullet \\
\end{array}
\quad \begin{array}{c}
\text{Output representation} \\
\hline
\bullet \quad \bullet \\
\bullet \quad \bullet \\
\bullet \quad \bullet \\
\bullet \quad \bullet \\
\bullet \quad \bullet \\
\end{array}
\]

\[w_j \]

\[x \]

\[b_j \]

\[y \]

Full layer

\[
y = Wx + b
\]

\[
\begin{bmatrix}
w_{11} & \cdots & w_{jn} & b_1 \\
\vdots & \ddots & \vdots & \vdots \\
w_{j1} & \cdots & w_{jn} & b_j \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_n \\
1
\end{bmatrix}
= \begin{bmatrix}
y_1 \\
y_2 \\
\vdots \\
y_j
\end{bmatrix}
\]

Can again simplify notation by appending a 1 to \(x\)

Adapted from: Isola, Torralba, Freeman
We can now transform our input representation vector into some output representation vector using a bunch of linear combinations of the input:

\[ \mathbf{x} \rightarrow \mathbf{y} \rightarrow \mathbf{z} \]

We can repeat this as many times as we want!
What is the problem with this idea?

Can be expressed as single linear layer!

\[
\left( \prod_{i} W_i \right) x = \hat{W} x
\]

Limited power: can’t solve XOR :(
Solution: simple nonlinearity

Linear layer

Input representation: $x$, $1$, $w_j$, $b_j$

Output representation: $y$, $g(y)$

$g(y) = \begin{cases} 
1, & \text{if } y > 0 \\
0, & \text{otherwise}
\end{cases}$

Adapted from: Isola, Torralba, Freeman
Example: linear classification with a perceptron

\[ y = \mathbf{x}^T \mathbf{w} + b \]

Source: Isola, Torralba, Freeman
Example: linear classification with a perceptron

$$y = \mathbf{x}^T \mathbf{w} + b$$

$$g(y) = \begin{cases} 
1, & \text{if } y > 0 \\
0, & \text{otherwise}
\end{cases}$$

Source: Isola, Torralba, Freeman
Example: linear classification with a perceptron

\[ y = \mathbf{x}^T \mathbf{w} + b \]

\[ g(y) = \begin{cases} 
1, & \text{if } y > 0 \\
0, & \text{otherwise}
\end{cases} \]

“when \( y \) is greater than 0, set all pixel values to 1 (green), otherwise, set all pixel values to 0 (red)”

Source: Isola, Torralba, Freeman
Example: linear classification with a perceptron

\[ g(y) \]

\[ y = x^T w + b \]

\[ g(y) = \begin{cases} 1, & \text{if } y > 0 \\ 0, & \text{otherwise} \end{cases} \]

"when y is greater than 0, set all pixel values to 1 (green), otherwise, set all pixel values to 0 (red)"

Source: Isola, Torralba, Freeman
Computation in a neural net - nonlinearity

Linear layer

Input representation

\( x \)

\( w_j \)

\( b_j \)

Output representation

\( y \)

\( g(y) \)

Can’t use with gradient descent, \( \frac{\partial}{\partial y} g = 0 \)

\( g(y) = \begin{cases} 
1, & \text{if } y > 0 \\
0, & \text{otherwise}
\end{cases} \)
Computation in a neural net - nonlinearity

**Linear layer**

Input representation

\[
x \quad w_j \quad b_j
\]

Output representation

\[
y \quad g(y)
\]

**Sigmoid**

\[
g(y) = \sigma(y) = \frac{1}{1 + e^{-y}}
\]

Adapted from: Isola, Torralba, Freeman
Computation in a neural net - nonlinearity

• Bounded between $[0,1]$

• Saturation for large +/- inputs

• Gradients go to zero

• Better in practice to use: $\tanh(y)$
  
  $= 2g(y) - 1$

Sigmoid

$$g(y) = \sigma(y) = \frac{1}{1 + e^{-y}}$$
Computation in a neural net — nonlinearity

- Unbounded output (on positive side)
- Efficient to implement: \[ \frac{\partial g}{\partial y} = \begin{cases} 0, & \text{if } y < 0 \\ 1, & \text{if } y \geq 0 \end{cases} \]
- Also seems to help convergence (6x speedup vs. tanh in [Krizhevsky et al. 2012])
- Drawback: if strongly in negative region, unit is dead forever (no gradient).
- Default choice: widely used in current models!

**Rectified linear unit (ReLU)**

\[ g(y) = \max(0, y) \]

Source: Isola, Torralba, Freeman
Computation in a neural net — nonlinearity

- where $\alpha$ is small (e.g., 0.02)
- Efficient to implement: $\frac{\partial g}{\partial y} = \begin{cases} -\alpha, & \text{if } y < 0 \\ 1, & \text{if } y \geq 0 \end{cases}$
- Has non-zero gradients everywhere (unlike ReLU)

Leaky ReLU

$$g(y) = \begin{cases} \max(0, y), & \text{if } y \geq 0 \\ \alpha \min(0, y), & \text{if } y < 0 \end{cases}$$

Source: Isola, Torralba, Freeman
Stacking layers

Input representation

Intermediate representation

Output representation

\[ \mathbf{h} = \text{“hidden units”} \]

Adapted from: Isola, Torralba, Freeman
Connectivity patterns

**Fully connected layer**

**Locally connected layer**
*(Sparse W)*

Source: Isola, Torralba, Freeman
Stacking layers

$$h = g(W^1 x + b^1) \quad y = g(W^2 h + b^2)$$

ReLU

$$\theta = \{W^1, \ldots, W^L, b^1, \ldots, b^L\}$$

Source: Isola, Torralba, Freeman
Stacking layers

\[ h = g(W^1 x + b^1) \]
\[ y = g(W^2 h + b^2) \]

ReLU

\[ \theta = \{W^1, \ldots, W^L, b^1, \ldots, b^L\} \]

Source: Isola, Torralba, Freeman
Stacking layers

Source: Isola, Torralba, Freeman
Stacking layers

\[ h = g(W^1 x + b^1) \quad y = g(W^2 h + b^2) \]

ReLU

\[ \theta = \{W^1, \ldots, W^L, b^1, \ldots, b^L\} \]

Source: Isola, Torralba, Freeman
Stacking layers

Input representation

Intermediate representation

Output representation

\[
\begin{align*}
\mathbf{h} &= g(\mathbf{W}^1 \mathbf{x} + \mathbf{b}^1) \\
\mathbf{y} &= g(\mathbf{W}^2 \mathbf{h} + \mathbf{b}^2)
\end{align*}
\]

ReLU

\[
\theta = \{\mathbf{W}^1, \ldots, \mathbf{W}^L, \mathbf{b}^1, \ldots, \mathbf{b}^L\}
\]

Source: Isola, Torralba, Freeman
Stacking layers

\[ h = g(W^1 x + b^1) \quad y = g(W^2 h + b^2) \]

ReLU \[ \theta = \{W^1, \ldots, W^L, b^1, \ldots, b^L\} \]

Source: Isola, Torralba, Freeman
Stacking layers - What’s actually happening?

Low level features: e.g., edge, texture, …

higher level features: e.g., shape

even higher level features: e.g., “paw”, “fur”

Source: Isola, Torralba, Freeman
Deep nets

\[ f(x) = f_L( ... f_3(f_2(f_1(x))) ) \]

Source: Isola, Torralba, Freeman
Deep nets - Intuition

“has horizontal edge”
“has vertical edge”

Source: Isola, Torralba, Freeman
Deep nets - Intuition

“has rounded edge”

Source: Isola, Torralba, Freeman
Deep nets - Intuition

“has white fur”
“has paw”
etc

How do we make a classification?

“dog”

Source: Isola, Torralba, Freeman
Deep nets - Intuition

“has white fur”
“has paw”
etc

Classify

Recall:

Source: Isola, Torralba, Freeman
Computation has a simple form

• Composition of linear functions with nonlinearities in between
• E.g. matrix multiplications with ReLU, $\max(0, x)$ afterwards
• Do a matrix multiplication, set all negative values to 0, repeat

But where do we get the weights from?
How do we learn the parameters?

\[ \theta^* = \arg\min_{\theta} \sum_{i=1}^{N} \mathcal{L}(f_{\theta}(x_i), y_i) \]

Source: Isola, Torralba, Freeman
Learning parameters

Squared loss with single-variable network:

\[ L = \frac{1}{2} (y - f(x))^2 \]

\[ L = \frac{1}{2} (y - \sigma(wx + b))^2 \]

Want: derivatives \( \frac{\partial L}{\partial w}, \frac{\partial L}{\partial b} \)

Example source: Roger Grosse
Computing derivatives with the chain rule

Given: $L = \frac{1}{2} (y - \sigma(wx + b))^2$

Writing out the layers explicitly:
- $z = wx + b$
- $t = \sigma(z)$
- $L = \frac{1}{2} (y - t)^2$

Example source: Roger Grosse
Computing derivatives with the chain rule

\[
\frac{\partial L}{\partial w} = \frac{\partial}{\partial w} \left[ \frac{1}{2} (y - \sigma(wx + b))^2 \right] \\
= (y - \sigma(wx + b)) \frac{\partial}{\partial w} (y - \sigma(wx + b)) \\
= (y - \sigma(wx + b)) \sigma'(wx + b) \frac{\partial}{\partial w} (wx + b) \\
= (y - \sigma(wx + b)) \sigma'(wx + b)x
\]

\[
\frac{\partial L}{\partial b} = \frac{\partial}{\partial b} \left[ \frac{1}{2} (y - \sigma(wx + b))^2 \right] \\
= (y - \sigma(wx + b)) \frac{\partial}{\partial b} (y - \sigma(wx + b)) \\
= (y - \sigma(wx + b)) \sigma'(wx + b) \frac{\partial}{\partial b} (wx + b) \\
= (y - \sigma(wx + b)) \sigma'(wx + b)
\]

Note: For each of these derivatives, you’ll have to compute many things multiple times!
Limitations to this approach

• Inefficient! Lots of redundant computation

• We’ll also need to extend this to multivariable functions

• Next lecture: backpropagation
1 layer? Linear decision surface.

2+ layers? In theory, can represent any function! (if it was infinitely wide with infinite data)
  - Simple proof by M. Nielsen

But issue is efficiency: very wide two layers vs narrow deep model? In practice, more layers helps.
Backup Slides
Softmax outputs a probability distribution over all predicted classes:

\[
\begin{bmatrix}
  3 \\
  1.75 \\
  -2 \\
  0.5
\end{bmatrix}
\xrightarrow{\text{SoftMax}}
\begin{bmatrix}
  0.725 \\
  0.21 \\
  0.005 \\
  0.06
\end{bmatrix}
\]

Sigmoid – not a probability distribution:

\[
\begin{bmatrix}
  3 \\
  1.75 \\
  -2 \\
  0.5
\end{bmatrix}
\xrightarrow{\text{Sigmoid}}
\begin{bmatrix}
  0.95 \\
  0.85 \\
  0.12 \\
  0.62
\end{bmatrix}
\]
Example: perceptron

\[ y = \sigma(W^{(1)}x) \]
Example: multilayer perceptron (MLP)

\[ y = \sigma(W^{(2)} \max(0, W^{(1)}x)) \]

Example source: http://playground.tensorflow.org
Example: multilayer perceptron (MLP)

Example source: http://playground.tensorflow.org
Example: multilayer perceptron (MLP)

What does this unit do?

Example source: http://playground.tensorflow.org