Deep Learning
(CNNs)

Deep Learning Readings:
Murphy 28
Bishop --
HTF --
Mitchell --
Reminders

• Homework 5 (Part II): Peer Review
  – Release: Wed, Mar. 29
  – Due: Wed, Apr. 05 at 11:59pm

• Peer Tutoring

• Homework 7: Deep Learning
  – Release: Wed, Apr. 05
  – Watch for multiple due dates!!

Expectation: You should spend at most 1 hour on your reviews
BACKPROPAGATION
1. Given training data:
\[
\{x_i, y_i\}^{N}_{i=1}
\]

2. Choose each of these:
- Decision function
  \[
  \hat{y} = f_{\theta}(x_i)
  \]
- Loss function
  \[
  \ell(\hat{y}, y_i) \in \mathbb{R}
  \]

3. Define goal:
\[
\theta^* = \arg\min_{\theta} \sum_{i=1}^{N} \ell(f_{\theta}(x_i), y_i)
\]

4. Train with SGD:
(take small steps opposite the gradient)
\[
\theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_{\theta}(x_i), y_i)
\]
Training

Whiteboard

– Example: Backpropagation for Calculus Quiz #1

Calculus Quiz #1:
Suppose \( x = 2 \) and \( z = 3 \), what are \( \frac{dy}{dx} \) and \( \frac{dy}{dz} \) for the function below?

\[
y = \exp(xz) + \frac{xz}{\log(x)} + \frac{\sin(\log(x))}{\exp(xz)}
\]
Automatic Differentiation – Reverse Mode (aka. Backpropagation)

Forward Computation
1. Write an **algorithm** for evaluating the function \( y = f(x) \). The algorithm defines a **directed acyclic graph**, where each variable is a node (i.e. the “computation graph”)
2. Visit each node in **topological order**.
   For variable \( u_i \) with inputs \( v_1, \ldots, v_N \)
   a. Compute \( u_i = g_i(v_1, \ldots, v_N) \)
   b. Store the result at the node

Backward Computation
1. **Initialize** all partial derivatives \( \frac{dy}{du_j} \) to 0 and \( \frac{dy}{dy} = 1 \).
2. Visit each node in **reverse topological order**.
   For variable \( u_i = g_i(v_1, \ldots, v_N) \)
   a. We already know \( \frac{dy}{du_i} \)
   b. Increment \( \frac{dy}{dv_j} \) by \( (\frac{dy}{du_i})(\frac{du_i}{dv_j}) \)
      (Choice of algorithm ensures computing \( \frac{du_i}{dv_j} \) is easy)

**Return** partial derivatives \( \frac{dy}{du_i} \) for all variables
Simple Example: The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

Forward

\[ J = \cos(u) \]

\[ u = u_1 + u_2 \]

\[ u_1 = \sin(t) \]

\[ u_2 = 3t \]

\[ t = x^2 \]
Simple Example: The goal is to compute \( J = \cos(\sin(x^2) + 3x^2) \) on the forward pass and the derivative \( \frac{dJ}{dx} \) on the backward pass.

Forward

\[
J = \cos(u) \\
u = u_1 + u_2 \\
u_1 = \sin(t) \\
u_2 = 3t \\
t = x^2
\]

Backward

\[
\frac{dJ}{du} = -\sin(u) \\
\frac{dJ}{du_1} + = \frac{dJ}{du} \frac{du}{du_1}, \quad \frac{du}{du_1} = 1 \\
\frac{dJ}{du} + = \frac{dJ}{du} \frac{du}{du_2}, \quad \frac{du}{du_2} = 1 \\
\frac{dJ}{du_1} + = \frac{dJ}{du} \frac{du}{du_1}, \quad \frac{du}{du_1} = 1 \\
\frac{dJ}{dt} + = \frac{dJ}{du} \frac{du}{dt}, \quad \frac{du}{dt} = \cos(t) \\
\frac{dJ}{dt} + = \frac{dJ}{du} \frac{du}{dt}, \quad \frac{du}{dt} = 3 \\
\frac{dJ}{dt} + = \frac{dJ}{du} \frac{du}{dt}, \quad \frac{du}{dt} = 3 \\
\frac{dJ}{dx} + = \frac{dJ}{dt} \frac{dt}{dx}, \quad \frac{dt}{dx} = 2x
\]
Case 1: Logistic Regression

**Forward**

\[ J = y^* \log y + (1 - y^*) \log (1 - y) \]

\[ y = \frac{1}{1 + \exp(-a)} \]

\[ a = \sum_{j=0}^{D} \theta_j x_j \]

**Backward**

\[ \frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1} \]

\[ \frac{dJ}{da} = \frac{dJ}{dy} \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2} \]

\[ \frac{dJ}{d\theta_j} = \frac{dJ}{da} \frac{da}{d\theta_j} = x_j \]

\[ \frac{dJ}{dx_j} = \frac{dJ}{da} \frac{da}{dx_j} = \theta_j \]
Training

Backpropagation

(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)
\[ y = \frac{1}{1 + \exp(-b)} \]
Training

Backpropagation

(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
$$J = \frac{1}{2} (y - y^*)^2$$
Case 2: Neural Network

Forward

\[ J = y^* \log y + (1 - y^*) \log(1 - y) \]
\[ y = \frac{1}{1 + \exp(-b)} \]
\[ b = \sum_{j=0}^{D} \beta_j z_j \]
\[ z_j = \frac{1}{1 + \exp(-a_j)} \]
\[ a_j = \sum_{i=0}^{M} \alpha_{ji} x_i \]

Backward

\[ \frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1} \]
\[ \frac{dJ}{db} = \frac{dy}{db} \frac{dJ}{dy} \quad \frac{db}{dy} \frac{dJ}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2} \]
\[ \frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{d\beta_j}{db} \quad \frac{d\beta_j}{db} = z_j \]
\[ \frac{dJ}{dz_j} = \frac{dJ}{db} \frac{dz_j}{db} \quad \frac{dz_j}{db} = \beta_j \]
\[ \frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{da_j}{dz_j} \quad \frac{da_j}{dz_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2} \]
\[ \frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{d\alpha_{ji}}{da_j} \quad \frac{d\alpha_{ji}}{da_j} = x_i \]
\[ \frac{dJ}{dx_i} = \frac{dJ}{da_j} \frac{da_j}{dx_i} \quad \frac{da_j}{dx_i} = \sum_{j=0}^{D} \alpha_{ji} \]
<table>
<thead>
<tr>
<th>Case 2:</th>
<th>Forward</th>
<th>Backward</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Loss</strong></td>
<td>( J = y^* \log y + (1 - y^*) \log(1 - y) )</td>
<td>( \frac{dJ}{dy} = \frac{y^<em>}{y} + \frac{(1 - y^</em>)}{y - 1} )</td>
</tr>
<tr>
<td><strong>Sigmoid</strong></td>
<td>( y = \frac{1}{1 + \exp(-b)} )</td>
<td>( \frac{dJ}{db} = \frac{dy}{db} \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2} )</td>
</tr>
<tr>
<td>( b = \sum_{j=0}^{D} \beta_j z_j )</td>
<td>( \frac{dJ}{d\beta_j} = \frac{dy}{db} \frac{db}{d\beta_j} = \frac{\exp(-b)}{(\exp(-b) + 1)^2} )</td>
<td></td>
</tr>
<tr>
<td><strong>Linear</strong></td>
<td>( z_j = \frac{1}{1 + \exp(-a_j)} )</td>
<td>( \frac{dJ}{da_j} = \frac{dz_j}{da_j} \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2} )</td>
</tr>
<tr>
<td>( a_j = \sum_{i=0}^{M} \alpha_{ji} x_i )</td>
<td>( \frac{dJ}{d\alpha_{ji}} = \frac{da_j}{da_j} \frac{da_j}{da_j} = x_i )</td>
<td></td>
</tr>
<tr>
<td><strong>Linear</strong></td>
<td>( \frac{dJ}{dx_i} = \frac{da_j}{da_j} \frac{da_j}{dx_i} = \sum_{j=0}^{D} \alpha_{ji} )</td>
<td></td>
</tr>
</tbody>
</table>
Whiteboard

– SGD for Neural Network
– Example: Backpropagation for Neural Network
Backpropagation (Auto.Diff. - Reverse Mode)

Forward Computation
1. Write an **algorithm** for evaluating the function \( y = f(x) \). The algorithm defines a **directed acyclic graph**, where each variable is a node (i.e. the “**computation graph**”)
2. Visit each node in **topological order**.
   a. Compute the corresponding variable’s value
   b. Store the result at the node

Backward Computation
1. **Initialize** all partial derivatives \( \frac{dy}{du_j} \) to 0 and \( \frac{dy}{dy} = 1 \).
2. Visit each node in **reverse topological order**.
   For variable \( u_i = g_i(v_1, \ldots, v_N) \)
   a. We already know \( dy/du_i \)
   b. Increment \( dy/dv_j \) by \( (dy/du_i)(du_i/dv_j) \)
   
   *(Choice of algorithm ensures computing \( (du_i/dv_j) \) is easy)*

**Return** partial derivatives \( dy/du_i \) for all variables
1. Given training data:

\[ \{x_i, y_i\}_{i=1}^{N} \]

2. Choose each of these:

- Decision function

\[ \hat{y} = f_\theta(x_i) \]

- Loss function

\[ \ell(\hat{y}, y_i) \in \mathbb{R} \]

Gradients

**Backpropagation** can compute this gradient!

And it’s a **special case of a more general algorithm** called reverse-mode automatic differentiation that can compute the gradient of any differentiable function efficiently!
Summary

1. **Neural Networks**
   - provide a way of learning features
   - are highly nonlinear prediction functions
   - (can be) a highly parallel network of logistic regression classifiers
   - discover useful hidden representations of the input

2. **Backpropagation**
   - provides an efficient way to compute gradients
   - is a special case of reverse-mode automatic differentiation
DEEP LEARNING
Deep Learning Outline

• **Background: Computer Vision**
  – Image Classification
  – ILSVRC 2010 - 2016
  – Traditional Feature Extraction Methods
  – Convolution as Feature Extraction

• **Convolutional Neural Networks (CNNs)**
  – Learning Feature Abstractions
  – Common CNN Layers:
    • Convolutional Layer
    • Max-Pooling Layer
    • Fully-connected Layer (w/tensor input)
    • Softmax Layer
    • ReLU Layer
  – Background: Subgradient
  – Architecture: LeNet
  – Architecture: AlexNet

• **Training a CNN**
  – SGD for CNNs
  – Backpropagation for CNNs
Why is everyone talking about Deep Learning?

- Because a lot of money is invested in it...
  - DeepMind: Acquired by Google for $400 million
  - DNNResearch: *Three person startup* (including Geoff Hinton) acquired by Google for unknown price tag
  - Enlitic, Ersatz, MetaMind, Nervana, Skylab: Deep Learning startups commanding *millions* of VC dollars
- Because it made the *front page* of the New York Times
Deep learning:
- Has won numerous pattern recognition competitions
- Does so with minimal feature engineering

This wasn’t always the case!
Since 1980s: Form of models hasn’t changed much, but lots of new tricks...
- More hidden units
- Better (online) optimization
- New nonlinear functions (ReLUs)
- Faster computers (CPUs and GPUs)
BACKGROUND: COMPUTER VISION
Example: Image Classification

• ImageNet LSVRC-2011 contest:
  – **Dataset**: 1.2 million labeled images, 1000 classes
  – **Task**: Given a new image, label it with the correct class
  – **Multiclass** classification problem
• Examples from http://image-net.org/
Bird
Warm-blooded egg-laying vertebrates characterized by feathers and forelimbs modified as wings
German iris, Iris kochii
Iris of northern Italy having deep blue-purple flowers; similar to but smaller than Iris germanica

halophyte (0)  succulent (39)  cultivar (0)  cultivated plant (0)  weed (54)  evergreen, evergreen plant (0)  deciduous plant (0)  vine (272)  creeper (0)  woody plant, ligneous plant (1868)  geophyte (0)  desert plant, xerophyte, xerophytic plant, xerophile, xerophilic  mesophyte, mesophytic plant (0)  aquatic plant, water plant, hydrophyte, hydrophytic plant (11)  tuberous plant (0)  bulbous plant (179)  iridaceous plant (27)  iris, flag, flour-de-lis, sword lily (19)  bearded iris (4)  Florentine iris, orris, Iris germanica florentina, Iris Germanica, Iris germanica (0)  German iris, Iris germanica (0)  Dalmatian iris, Iris pallida (0)  beardless iris (4)  bulbous iris (0)  dwarf iris, Iris cristata (0)  stinking iris, gladdon, gladdon iris, stinking gladwyn, Persian iris, Iris persica (0)  yellow iris, yellow flag, yellow water flag, Iris psueda  dwarf iris, vernal iris, Iris verna (0)  blue flag, Iris versicolor (0)
Court, courtyard
An area wholly or partly surrounded by walls or buildings; "the house was built around an inner court"
Example: Image Classification

Traditional Feature Extraction for Images:

– SIFT
– HOG
Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input is 150,528-dimensional, and the number of neurons in the network’s remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with 256 kernels of size $5 \times 5 \times 48$. The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3 \times 256$ connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 192$, and the fifth convolutional layer has 256 kernels of size $3 \times 3 \times 192$. The fully-connected layers have 4096 neurons each.

4 Reducing Overfitting

Our neural network architecture has 60 million parameters. Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting. Below, we describe the two primary ways in which we combat overfitting.

4.1 Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random $224 \times 224$ patches (and their horizontal reflections) from the $256 \times 256$ images and training our network on these extracted patches. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five $224 \times 224$ patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network’s softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components.

This is the reason why the input images in Figure 2 are $224 \times 224 \times 3$-dimensional.
CNNs for Image Recognition

Revolution of Depth

ILSVRC'15 ResNet: 3.57%
ILSVRC'14 GoogleNet: 6.7%
ILSVRC'14 VGG: 7.3%
ILSVRC'13: 8 layers, 11.7%
ILSVRC'12 AlexNet: 8 layers, 16.4%
ILSVRC'11: 25.8%
ILSVRC'10: 28.2%

ImageNet Classification top-5 error (%)


Slide from Kaiming He
CONVOLUTION
What’s a convolution?

• Basic idea:
  – Pick a 3x3 matrix F of weights
  – Slide this over an image and compute the “inner product” (similarity) of F and the corresponding field of the image, and replace the pixel in the center of the field with the output of the inner product operation

• Key point:
  – Different convolutions extract different types of low-level “features” from an image
  – All that we need to vary to generate these different features is the weights of F

---

Ex: 1 input channel, 1 output channel

<table>
<thead>
<tr>
<th>Input</th>
<th>Conv</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{11}$</td>
<td>$\alpha_{11}$</td>
<td>$y_{11}$</td>
</tr>
<tr>
<td>$x_{12}$</td>
<td>$\alpha_{12}$</td>
<td>$y_{12}$</td>
</tr>
<tr>
<td>$x_{13}$</td>
<td>$\alpha_{13}$</td>
<td>$y_{13}$</td>
</tr>
<tr>
<td>$x_{21}$</td>
<td>$\alpha_{21}$</td>
<td>$y_{21}$</td>
</tr>
<tr>
<td>$x_{22}$</td>
<td>$\alpha_{22}$</td>
<td>$y_{22}$</td>
</tr>
<tr>
<td>$x_{23}$</td>
<td>$\alpha_{23}$</td>
<td>$y_{23}$</td>
</tr>
<tr>
<td>$x_{31}$</td>
<td>$\alpha_{31}$</td>
<td>$y_{31}$</td>
</tr>
<tr>
<td>$x_{32}$</td>
<td>$\alpha_{32}$</td>
<td>$y_{32}$</td>
</tr>
<tr>
<td>$x_{33}$</td>
<td>$\alpha_{33}$</td>
<td>$y_{33}$</td>
</tr>
</tbody>
</table>

\[ y_{11} = \alpha_{11} x_{11} + \alpha_{12} x_{12} + \alpha_{21} x_{21} + \alpha_{22} x_{22} + \alpha_{31} x_{31} + \alpha_{32} x_{32} + \alpha_{33} x_{33} + \alpha_0 \]
\[ y_{12} = \alpha_{11} x_{11} + \alpha_{12} x_{12} + \alpha_{21} x_{22} + \alpha_{22} x_{22} + \alpha_{31} x_{32} + \alpha_{32} x_{32} + \alpha_{33} x_{33} + \alpha_0 \]
\[ y_{21} = \alpha_{11} x_{21} + \alpha_{12} x_{22} + \alpha_{21} x_{21} + \alpha_{22} x_{22} + \alpha_{31} x_{32} + \alpha_{32} x_{32} + \alpha_{33} x_{33} + \alpha_0 \]
\[ y_{22} = \alpha_{11} x_{22} + \alpha_{12} x_{22} + \alpha_{21} x_{22} + \alpha_{22} x_{22} + \alpha_{31} x_{33} + \alpha_{32} x_{33} + \alpha_{33} x_{33} + \alpha_0 \]

---

Slide adapted from William Cohen
A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

**Input Image**

**Convolved Image**

**Convolution**
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

![Input Image](image1)

![Convolution](image2)

![Convolved Image](image3)
A convolution matrix is used in image processing for tasks such as edge detection, blurring, sharpening, etc.
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Convolution

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Convolved Image
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

### Input Image

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

### Convolved Image

<table>
<thead>
<tr>
<th>3</th>
<th>2</th>
<th>2</th>
<th>3</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

![Input Image](image1)

![Convolution](image2)

![Convolved Image](image3)
A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.
A *convolution matrix* is used in image processing for tasks such as edge detection, blurring, sharpening, etc.
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

**Input Image**

```
0 0 0 0 0 0 0
0 1 1 1 1 0 0
0 1 0 1 0 0 0
0 1 0 1 0 0 0
0 1 1 0 0 0 0
0 1 1 0 0 0 0
0 1 0 0 0 0 0
0 0 0 0 0 0 0
```

**Convolution**

```
3 2 2 3
```

**Convolved Image**

```
``
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

![Input Image](image)

![Convolution](image)

![Convolved Image](image)
A convolution matrix is used in image processing for tasks such as edge detection, blurring, sharpening, etc.
Background: Image Processing

A convolution matrix is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

Convolution

Convolved Image
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

![Input Image](image)

![Convolution](image)

![Convolved Image](image)
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

### Input Image

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Convolved Image

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Identity Convolution

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

**Input Image**

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Blurring Convolution**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>.1</td>
<td>.1</td>
<td>.1</td>
</tr>
<tr>
<td>.1</td>
<td>.2</td>
<td>.1</td>
</tr>
<tr>
<td>.1</td>
<td>.1</td>
<td>.1</td>
</tr>
</tbody>
</table>

**Convolved Image**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>.4</td>
<td>.5</td>
<td>.5</td>
<td>.5</td>
<td>.4</td>
</tr>
<tr>
<td>.4</td>
<td>.2</td>
<td>.3</td>
<td>.6</td>
<td>.3</td>
</tr>
<tr>
<td>.5</td>
<td>.4</td>
<td>.4</td>
<td>.2</td>
<td>.1</td>
</tr>
<tr>
<td>.5</td>
<td>.6</td>
<td>.2</td>
<td>.1</td>
<td>0</td>
</tr>
<tr>
<td>.4</td>
<td>.3</td>
<td>.1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
What’s a convolution?

http://matlabtricks.com/post-5/3x3-convolution-kernels-with-online-demo
What’s a convolution?

http://matlabtricks.com/post-5/3x3-convolution-kernels-with-online-demo
What’s a convolution?

http://matlabtricks.com/post-5/3x3-convolution-kernels-with-online-demo
What’s a convolution?

http://matlabtricks.com/post-5/3x3-convolution-kernels-with-online-demo
What’s a convolution?

http://matlabtricks.com/post-5/3x3-convolution-kernels-with-online-demo
What’s a convolution?

http://matlabtricks.com/post-5/3x3-convolution-kernels-with-online-demo
What’s a convolution?

• Basic idea:
  – Pick a 3x3 matrix F of weights
  – Slide this over an image and compute the “inner product” (similarity) of F and the corresponding field of the image, and replace the pixel in the center of the field with the output of the inner product operation

• Key point:
  – Different convolutions extract different types of low-level “features” from an image
  – All that we need to vary to generate these different features is the weights of F

---

Ex: 1 input channel, 1 output channel

<table>
<thead>
<tr>
<th>Input</th>
<th>Conv</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_{11})</td>
<td>(x_{12})</td>
<td>(x_{13})</td>
</tr>
<tr>
<td>(x_{21})</td>
<td>(x_{22})</td>
<td>(x_{23})</td>
</tr>
<tr>
<td>(x_{31})</td>
<td>(x_{32})</td>
<td>(x_{33})</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
  y_{11} &= \alpha_{11}x_{11} + \alpha_{12}x_{12} + \alpha_{21}x_{21} + \alpha_{22}x_{22} + \alpha_{32}x_{32} + \alpha_0 \\
  y_{12} &= \alpha_{11}x_{12} + \alpha_{12}x_{13} + \alpha_{21}x_{22} + \alpha_{22}x_{23} + \alpha_{32}x_{33} + \alpha_0 \\
  y_{21} &= \alpha_{11}x_{21} + \alpha_{12}x_{22} + \alpha_{21}x_{31} + \alpha_{22}x_{32} + \alpha_{32}x_{33} + \alpha_0 \\
  y_{22} &= \alpha_{11}x_{22} + \alpha_{12}x_{23} + \alpha_{21}x_{32} + \alpha_{22}x_{33} + \alpha_{32}x_{33} + \alpha_0 \\
\end{align*}
\]

Slide adapted from William Cohen
Suppose we use a convolution with stride 2
Only 9 patches visited in input, so only 9 pixels in output
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output

![Image showing input and convolved images with a convolution setup and output](image-url)
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output

Input Image

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Convolution

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Convolved Image

<table>
<thead>
<tr>
<th>3</th>
<th>3</th>
</tr>
</thead>
</table>
Downsampling

• Suppose we use a convolution with stride 2
• Only 9 patches visited in input, so only 9 pixels in output

![Input Image](image)

![Convolved Image](image)
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output

![Input Image](image1)

![Convolved Image](image2)
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output
CONVOLUTIONAL NEURAL NETS
Deep Learning Outline

• **Background: Computer Vision**
  – Image Classification
  – ILSVRC 2010 - 2016
  – Traditional Feature Extraction Methods
  – Convolution as Feature Extraction

• **Convolutional Neural Networks (CNNs)**
  – Learning Feature Abstractions
  – Common CNN Layers:
    • Convolutional Layer
    • Max-Pooling Layer
    • Fully-connected Layer (w/tensor input)
    • Softmax Layer
    • ReLU Layer
  – Background: Subgradient
  – Architecture: LeNet
  – Architecture: AlexNet

• **Training a CNN**
  – SGD for CNNs
  – Backpropagation for CNNs
Convolutional Neural Network (CNN)

- Typical layers include:
  - Convolutional layer
  - Max-pooling layer
  - Fully-connected (Linear) layer
  - ReLU layer (or some other nonlinear activation function)
  - Softmax

- These can be arranged into arbitrarily deep topologies

**Architecture #1: LeNet-5**

![Architecture of LeNet-5](image)

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.
Convolutional Layer

**CNN key idea:** Treat convolution matrix as parameters and learn them!

![Input Image](image1)

![Learned Convolution Matrix](image2)

![Convoluted Image](image3)
Downsampling by Averaging

- Downsampling by averaging **used to be** a common approach
- This is a special case of convolution where the weights are fixed to a uniform distribution
- The example below uses a stride of 2

**Input Image**

```
1 1 1 1 1 0
1 0 0 1 0 0
1 0 1 0 0 0
1 1 0 0 0 0
1 0 0 0 0 0
0 0 0 0 0 0
```

**Convolution**

```
1/4 1/4
1/4 1/4
```

**Convolved Image**

```
3/4 3/4 1/4
3/4 1/4 0
1/4 0 0
```
Max-Pooling

- Max-pooling is another (common) form of downsampling
- Instead of averaging, we take the max value within the same range as the equivalently-sized convolution
- The example below uses a stride of 2

\[
y_{ij} = \max(x_{ij}, x_{i,j+1}, x_{i+1,j}, x_{i+1,j+1})
\]
Multi-Class Output

Output

Hidden Layer

Input
Multi-Class Output

Softmax Layer:

\[ y_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)} \]

Output

\[ y_1 \quad \ldots \quad y_k \quad \ldots \quad y_M \]

Hidden Layer

\[ z_1 \quad z_2 \quad \ldots \quad z_D \]

Input

\[ x_1 \quad x_2 \quad x_3 \quad \ldots \quad x_M \]

(A) Input

Given \( x_i, \ \forall i \)

(B) Hidden (linear)

\[ a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j \]

(C) Hidden (nonlinear)

\[ z_j = \sigma(a_j), \ \forall j \]

(D) Output (linear)

\[ b_k = \sum_{j=0}^{D} \beta_{kj} z_j \ \forall k \]

(E) Output (softmax)

\[ y_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)} \]

(F) Loss

\[ J = \sum_{k=1}^{K} y_k^* \log(y_k) \]
Training a CNN

*Whiteboard*

- SGD for CNNs
- Backpropagation for CNNs
Common CNN Layers

Whiteboard

- ReLU Layer
- Background: Subgradient
- Fully-connected Layer (w/tensor input)
- Softmax Layer
- Convolutional Layer
- Max-Pooling Layer
Convolutional Layer

Ex. 1 input channel, 1 output channel

\[
\begin{array}{c|c|c|c}
\text{Input} & \text{Conv} & \text{Output} \\
\hline
x_{11} & x_{12} & x_{13} \\
x_{21} & x_{22} & x_{23} \\
x_{31} & x_{32} & x_{33} \\
\hline
\end{array}
\begin{array}{c|c|c|c}
\alpha_1 & \alpha_{12} & \alpha_2 \\
\alpha_{11} & \alpha_{12} & \alpha_3 \\
\alpha_{11} & \alpha_{12} & \alpha_2 \\
\hline
\end{array}
\begin{array}{c|c|c}
y_{11} & y_{12} \\
y_{21} & y_{22} \\
\hline
\end{array}
\]

\[
\begin{align*}
y_{11} &= \alpha_{11} x_{11} + \alpha_{12} x_{12} + \alpha_{21} x_{21} + \alpha_{22} x_{22} + \alpha_0 \\
y_{12} &= \alpha_{11} x_{11} + \alpha_{12} x_{12} + \alpha_{21} x_{22} + \alpha_{22} x_{23} + \alpha_0 \\
y_{21} &= \alpha_{11} x_{21} + \alpha_{12} x_{22} + \alpha_{21} x_{31} + \alpha_{22} x_{32} + \alpha_0 \\
y_{22} &= \alpha_{11} x_{22} + \alpha_{12} x_{23} + \alpha_{21} x_{22} + \alpha_{22} x_{33} + \alpha_0
\end{align*}
\]

Ex. 1 input channel, 2 output channels

\[
\begin{array}{c|c|c|c}
\text{Input} & \text{Conv#1} & \text{Output#1} & \text{Conv#2} & \text{Output#2} \\
\hline
x_{11} & x_{12} & x_{13} \\
x_{21} & x_{22} & x_{23} \\
x_{31} & x_{32} & x_{33} \\
\hline
\alpha_1 & \alpha_{12} & \alpha_2 \\
\alpha_{11} & \alpha_{12} & \alpha_3 \\
\alpha_{11} & \alpha_{12} & \alpha_2 \\
\hline
\end{array}
\begin{array}{c|c|c|c}
(\text{1}) & (\text{1}) & (\text{1}) & (\text{1}) \\
(\text{1}) & (\text{1}) & (\text{1}) & (\text{1}) \\
(\text{1}) & (\text{1}) & (\text{1}) & (\text{1}) \\
\hline
(\text{1}) & (\text{1}) & (\text{1}) & (\text{1}) \\
(\text{1}) & (\text{1}) & (\text{1}) & (\text{1}) \\
(\text{1}) & (\text{1}) & (\text{1}) & (\text{1}) \\
\hline
(\text{1}) & (\text{1}) & (\text{1}) & (\text{1}) \\
(\text{1}) & (\text{1}) & (\text{1}) & (\text{1}) \\
(\text{1}) & (\text{1}) & (\text{1}) & (\text{1}) \\
\hline
\end{array}
\begin{array}{c|c|c}
y_{11} & y_{12} \\
y_{21} & y_{22} \\
\hline
\end{array}
\begin{array}{c|c|c}
y_{11} & y_{12} \\
y_{21} & y_{22} \\
\hline
\end{array}
\]

\[
\begin{align*}
y_{11}^{(1)} &= \alpha_{11}^{(1)} x_{11} + \alpha_{12}^{(1)} x_{12} + \alpha_{21}^{(1)} x_{21} + \alpha_{22}^{(1)} x_{22} + \alpha_0^{(1)} \\
y_{11}^{(2)} &= \alpha_{11}^{(2)} x_{11} + \alpha_{12}^{(2)} x_{12} + \alpha_{21}^{(2)} x_{21} + \alpha_{22}^{(2)} x_{22} + \alpha_0^{(2)} \\
y_{12}^{(1)} &= \alpha_{11}^{(1)} x_{11} + \alpha_{12}^{(1)} x_{12} + \alpha_{21}^{(1)} x_{22} + \alpha_{22}^{(1)} x_{23} + \alpha_0^{(1)} \\
y_{12}^{(2)} &= \alpha_{11}^{(2)} x_{11} + \alpha_{12}^{(2)} x_{12} + \alpha_{21}^{(2)} x_{22} + \alpha_{22}^{(2)} x_{23} + \alpha_0^{(2)} \\
y_{21}^{(1)} &= \alpha_{11}^{(1)} x_{21} + \alpha_{12}^{(1)} x_{22} + \alpha_{21}^{(1)} x_{31} + \alpha_{22}^{(1)} x_{32} + \alpha_0^{(1)} \\
y_{21}^{(2)} &= \alpha_{11}^{(2)} x_{21} + \alpha_{12}^{(2)} x_{22} + \alpha_{21}^{(2)} x_{31} + \alpha_{22}^{(2)} x_{32} + \alpha_0^{(2)} \\
y_{22}^{(1)} &= \alpha_{11}^{(1)} x_{22} + \alpha_{12}^{(1)} x_{23} + \alpha_{21}^{(1)} x_{22} + \alpha_{22}^{(1)} x_{33} + \alpha_0^{(1)} \\
y_{22}^{(2)} &= \alpha_{11}^{(2)} x_{22} + \alpha_{12}^{(2)} x_{23} + \alpha_{21}^{(2)} x_{22} + \alpha_{22}^{(2)} x_{33} + \alpha_0^{(2)}
\end{align*}
\]
Convolutional Layer

Ex: \( C^I \) input channels, \( C^O \) output channels

\[ H^o, H^p = \left[ (h^i + 2p - k)/s + 1 \right] \]
\[ W^o = \left[ (w^i + 2p - k)/s + 1 \right] \]

where \( p \) = # pixels of padding on input \( k \) = size of conv. matrix \( s \) = stride length

Forward:
\[ y^{(k)}_{ij} = y^{(k)}_0 + \sum_{c=1}^{C^I} \sum_{q=1}^{K} \sum_{r=1}^{K} x^{(c)}_r X^{(c)}_{mn} \text{ where } m = s(i-1) + q, n = s(j-1) + r \]

Backward:
\[ \frac{dJ}{dw^{(k)}_0} = \sum_i \sum_j \frac{dJ}{dy^{(k)}_{ij}} \frac{dy^{(k)}_{ij}}{dalpha^{(k)}_0} \]
\[ \frac{dJ}{dx^{(k)}_{ij}} = \sum_k \frac{dJ}{dy^{(k)}_{ij}} \frac{dy^{(k)}_{ij}}{dalpha^{(k)}} \frac{dalpha^{(k)}}{dx^{(k)}_{ij}} \]
\[ \frac{dJ}{dx^{(c)}_{mn}} = \sum_i \sum_j \frac{dJ}{dy^{(c)}_{ij}} \frac{dy^{(c)}_{ij}}{dalpha^{(c)}} \frac{dalpha^{(c)}}{dx^{(c)}_{mn}} \]

Just save calculus
Max-Pooling Layer

Ex: 1 input channel, 1 output channel, stride of 1

\[
\begin{array}{c}
\text{Input} \\
X_{11} & X_{12} & X_{13} \\
X_{21} & X_{22} & X_{23} \\
X_{31} & X_{32} & X_{33}
\end{array}
\quad \rightarrow 
\begin{array}{c}
\text{Pool Size} \\
\end{array}
\quad \rightarrow 
\begin{array}{c}
\text{Output} \\
Y_{11} & Y_{12} \\
Y_{21} & Y_{22} \\
Y_{31} & Y_{32}
\end{array}
\]

\[
\begin{align*}
Y_{11} &= \max \left( X_{11}, \ X_{12}, \ X_{21}, \ X_{22} \right) \\
Y_{12} &= \max \left( X_{12}, \ X_{13}, \ X_{22}, \ X_{23} \right) \\
Y_{21} &= \max \left( X_{21}, \ X_{22}, \ X_{31}, \ X_{32} \right) \\
Y_{22} &= \max \left( X_{22}, \ X_{23}, \ X_{32}, \ X_{33} \right)
\end{align*}
\]
Max-Pooling Layer

Forward:

\[ Y^{(k)}_{ij} = \max_{q \in \{1, \ldots, k^3\}, r_1 \in \{2, \ldots, k\}} X^{(k)}_{mn} \text{ where } m = s(i-1) + q, \]
\[ n = s(j-1) + r \]

Backward:

\[ \frac{dJ}{dx^{(k)}_{mn}} = \sum_{i} \sum_{j} \frac{dJ}{dy^{(k)}_{ij}} \frac{dy^{(k)}_{ij}}{dx^{(k)}_{mn}} \]

Subderivatives:

- Max() is not differentiable, but subdifferentiable.
- There are a set of derivatives and we can just choose one for SGD.

\[ y = \max(a, b) \]
\[ \Rightarrow \frac{dJ}{da} = \frac{dJ}{dy} \frac{dy}{da} \text{ where } \frac{dy}{da} = \begin{cases} 1 & \text{if } a > b \\ 0 & \text{otherwise} \end{cases} \]
Convolutional Neural Network (CNN)

- Typical layers include:
  - Convolutional layer
  - Max-pooling layer
  - Fully-connected (Linear) layer
  - ReLU layer (or some other nonlinear activation function)
  - Softmax

- These can be arranged into arbitrarily deep topologies

**Architecture #1: LeNet-5**

**Proc. of the IEEE, November 1998**

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.
Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with 256 kernels of size $5 \times 5 \times 48$.

The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3 \times 256$ connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 192$, and the fifth convolutional layer has 256 kernels of size $3 \times 3 \times 192$. The fully-connected layers have 4096 neurons each.

Reducing Overfitting

Our neural network architecture has 60 million parameters. Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting. Below, we describe the two primary ways in which we combat overfitting.

4.1 Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random $224 \times 224$ patches (and their horizontal reflections) from the $256 \times 256$ images and training our network on these extracted patches. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five $224 \times 224$ patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network's softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components.

This is the reason why the input images in Figure 2 are $224 \times 224 \times 3$-dimensional.

CNN for Image Classification (Krizhevsky, Sutskever & Hinton, 2012)
15.3% error on ImageNet LSVRC-2012 contest
CNNs for Image Recognition

Revolution of Depth

ImageNet Classification top-5 error (%)

ILSVRC'15 ResNet: 3.57
ILSVRC'14 GoogleNet: 6.7
ILSVRC'14 VGG: 7.3
ILSVRC'13: 11.7
ILSVRC'12 AlexNet: 16.4
ILSVRC'11: 25.8
ILSVRC'10: 28.2


Slide from Kaiming He
CNN VISUALIZATIONS
3D Visualization of CNN

http://scs.ryerson.ca/~aharley/vis/conv/
Convolution of a Color Image

- Color images consist of 3 floats per pixel for RGB (red, green, blue) color values.
- Convolution must also be 3-dimensional.

Figure from Fei-Fei Li & Andrej Karpathy & Justin Johnson (CS231N)
Animation of 3D Convolution

http://cs231n.github.io/convolutional-networks/

Figure from Fei-Fei Li & Andrej Karpathy & Justin Johnson (CS231N)
MNIST Digit Recognition with CNNs (in your browser)

https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

Figure from Andrej Karpathy
CNN Summary

CNNs

– Are used for all aspects of computer vision, and have won numerous pattern recognition competitions
– Able learn interpretable features at different levels of abstraction
– Typically, consist of convolution layers, pooling layers, nonlinearities, and fully connected layers

Other Resources:
– Readings on course website
– Andrej Karpathy, CS231n Notes http://cs231n.github.io/convolutional-networks/