Reminders

• Hw2 is due on Monday. Submit via ODTUClass.
• Scheduling of the midterm exam
Today

• CNN initialization
• Applications of ConvNets
  – Image classification
  – Object detection
  – Artistic style transfer
  – Visualizing ConvNet classifications
  – ConvNets for NLP
How do I initialize my CNN?
• When the number of examples is not large enough, unsupervised pre-training helps.
  – This was done in Mohamed et al. (2009) and Krizhevsky et al. (2012).
    • Add a new layer, initialize it using an unsupervised method such as Restricted Boltzman Machines or Autoencoders.
    • After adding all layers, put a supervised layer on top (e.g. softmax + cross-entropy), then continue training

• Since then, many papers on initialization. [e.g. Glorot & Bengio (2010); He et al. (2015)]
It is reasonable to assume that in the final trained network, half of the weights will be positive, and half of them will be negative.

- So, starting with all zeros might sound like a good idea.

It is reasonable to assume that in the final trained network, half of the weights will be positive, and half of them will be negative.

- So, starting with all zeros might sound like a good idea.
  - No, it’s not!! Why?

• How about initializing with small random numbers?
  
  – e.g. \( W = 0.01 \times \text{np.random.randn}(D,H) \) where \text{randn}( ) samples from the normal distribution \( N(0,1) \).
  
  – Good because it breaks the symmetry (i.e. the problem with starting with all zeros).
  
  – But it could lead to two different problems:
• How about initializing with small random numbers?
  
  – e.g. \( W = 0.01 \times \text{np.random.randn(D,H)} \) where \text{randn( )} samples from the normal distribution \( N(0,1) \).
  
  – Good because it breaks the symmetry (i.e. the problem with starting with all zeros).
  
  – But it could lead to two different problems:
    ● Gradients might vanish! (due to very small numbers)
    ● The output from a randomly initialized neuron has a variance that grows with the number of inputs. (Could lead to very large responses!) Why?
Consider a perceptron with weights $w$ and input $x$.

We are interested in $\text{Var}(W^Tx) = ?$

We start with:

$w$ are independent, random variables with small variances.
$x$ is a constant vector, so $W^Tx$ is a sum of normal vars, each scaled with a positive or a negative number.

Remember: $\text{Var}[ax] = a^2 \sigma^2$ where $\text{Var}(x) = \sigma^2$

Also $\text{Var}[x+y] = \sigma^2 + \beta^2$ where $\text{Var}[y] = \beta^2$

**Proof:** $\text{Var}[x-y] = ?$

We know $\text{Var}(x) = E[x^2] - (E[x])^2$

$E[(x-y)^2] - (E[x-y])^2 = E[(x+y)^2 - 2xy] - (E[x] - E[y])^2$


$= \text{Var}[x] + \text{Var}[y]$

So with more dimensions $\text{Var}[W^Tx]$ increases.
• Solution: calibrate the variance:
  
  - Use \( w = \text{np.random.randn}(n) / \sqrt{n} \) where \( n \) is the number of inputs to the layer.
  
  - Xavier initialization \([\text{Glorot} \text{ and Bengio (2010)}]\)
    
    • Sample weights from \( N(0, \text{Var}(W)) \) where

\[
\text{Var}(W) = \frac{2}{n_{\text{in}} + n_{\text{out}}}
\]

• Since the introduction of ReLU, initialization problem seems to have been solved:
  – In He et al. (2015),
    \[
    w = \text{np.random.randn}(n) \times \sqrt{2.0/n}
    \]
    is suggested.
  – This is the current recommendation used in practice.

• Also, with Batch Normalization (next slide) networks become less sensitive to initialization.

Batch normalization
Basically, normalizes the input (i.e. activations of the previous layer) at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.
Batch Normalization

- Applying this technique usually amounts to inserting the BatchNorm layer immediately after fully connected or convolutional layers before non-linearities.
- It has become a very common practice to use BatchNorm.
- Networks that use BatchNorm are significantly more robust to bad initialization.
- BatchNorm also regularizes the model: “in a batch-normalized network we found that Dropout can be either removed or reduced in strength” [Ioffe and Szegedy (2015)]
Fine-tuning: another initialization method

• Typical scenario:
  – You want to train a network on your dataset $D$.
  – Grab an already trained network $N$. $N$ was trained on a different dataset than $D$ (but still in the same domain, e.g. natural images).
  – Use $N$ as your initial network and continue training (with very small learning rates).

• This is done a lot in computer vision.
  – e.g. Faster RCNN object detector starts its training from a network that was trained on ImageNET.
Applications of ConvNets
Image classification
Image Classification

• Most popular benchmark/challenge is ILSVRC
  – ImageNET dataset: 1.2 million images, 1000 categories
  – Since 2010
  – The task: given an image, make 5 predictions for the dominant object in the image. If one of them is correct, then it is counted as a success.
Image Classification

• Most popular benchmark/challenge is ILSVRC
  – ImageNET dataset: 1.2 million images, 1000 categories
  – Since 2010
  – The task: given an image, make 5 predictions for the dominant object in the image. If one of them is correct, then it is counted as a success.

• Newer datasets since then
  – e.g. Google's Open Image Dataset (Sep 30, 2016)
    • 9 million images, 6000 categories [link]
The second success story

Top-5 error rate over time

- 2012: AlexNet  16.5%
- 2013: ZF  11.7%
- 2014: VGG  7.3%
  2014: GoogLeNet  6.7%
- 2015: ResNet  3.6%
- Today (Aug 2016)  3.1%
  GoogLeNet-v4

Human error rate: 5.1%  [http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/]
AlexNet [Krizhevsky et al. NIPS 2012]

5 convolutional layers

3 full-connected layers

Each convolutional layer consists of:
convolution + ReLU + normalization + max-pooling
Top-5 error rate over time

- 2012: AlexNet 16.5% [Krizhevsky et al. (2012)]
- 2013: ZF 11.7% [Zeiler & Fergus (2014)]
- 2014: VGG 7.3%
  2014: GoogLeNet 6.7%
- 2015: ResNet 3.6%
- Today (Aug 2016) GoogLeNet-v4 3.1%

“It was an improvement on AlexNet by tweaking the architecture hyperparameters, in particular by expanding the size of the middle convolutional layers and making the stride and filter size on the first layer smaller.”

[Source: http://cs231n.github.io/convolutional-networks/#case]
Top-5 error rate over time

- **2012**: AlexNet 16.5% [Krizhevsky et al. (2012)]
- **2013**: ZF 11.7% [Zeiler & Fergus (2014)]
- **2014**: VGG 7.3% [Simonyan & Zisserman (2014)]
- **2014**: GoogLeNet 6.7%
  - “Main contribution: depth is critical. They used 16 layers.”
- **2015**: ResNet 3.6%
- **Today (Aug 2016)**
  - **GoogLeNet-v4** 3.1%
  - Extremely homogeneous architecture: only 3x3 convolutions and 2x2 pooling.
  - But, very expensive to evaluate and requires more memory.

[Source: http://cs231n.github.io/convolutional-networks/#case]
VGG or VGGnet

VGG stands for Visual Geometry Group (at Oxford)

Is the network above a VGG16 or a VGG19?
Top-5 error rate over time

- 2012: AlexNet 16.5% [Krizhevsky et al. (2012)]
- 2013: ZF 11.7% [Zeiler & Fergus (2014)]
- 2014: VGG 7.3% [Simonyan & Zisserman (2014)]
- 2014: GoogLeNet 6.7% [Szegedy et al. (2015)]
- 2015: ResNet
- Today (Aug 2016) GoogLeNet-v4

"Main contribution: Inception Module that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M. Increased # layers to 22).

Uses Average Pooling instead of Fully Connected layers at the top of the ConvNet, eliminating a large amount of parameters that do not seem to matter much.

[Source: http://cs231n.github.io/convolutional-networks/#case]
Inception module

(a) Inception module, naïve version

Multiscale processing + wider network

But very costly!! Solution is to reduce dimension (next slide)

[Figure from Szegedy et al. (2015)]
Dimension is reduced using 1x1 convolutions.
(Explain how)
Top-5 error rate over time

- 2012: AlexNet 16.5% [Krizhevsky et al. (2012)]
- 2013: ZF 11.7% [Zeiler & Fergus (2014)]
- 2014: VGG 7.3% [Simonyan & Zisserman (2014)]
- 2014: GoogLeNet 6.7% [Szegedy et al. (2015)]
- 2015: ResNet 3.6%
- Today (Aug 2016): GoogLeNet-v4 3.1% [Szegedy et al. (2016)]

v4 of Google's Inception Network is the best right now (as of Fall 2016). Uses better crafted inception modules + residual connections.
Top-5 error rate over time

- 2012: AlexNet 16.5% [Krizhevsky et al. (2012)]
- 2013: ZF 11.7% [Zeiler & Fergus (2014)]
- 2014: VGG 7.3% [Simonyan & Zisserman (2014)]
  - 2014: GoogLeNet 6.7% [Szegedy et al. (2015)]
- 2015: ResNet 3.6% [He et al. (2015)]
- Today (Aug 2016) GoogLeNet-v4 3.1% [Szegedy et al. (2016)]
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's talk at ICCV 2015 ImageNet and COCO joint workshop
Is learning better networks as simple as stacking more layers?

Simply stacking layers?

- Plain nets: stacking 3x3 conv layers...
- 56-layer net has **higher training error** and test error than 20-layer net

Slide from Kaiming He's talk at ICCV 2015 ImageNet and COCO joint workshop
A deeper model should not have higher training error.

A solution by construction:
- original layers: copied from a learned shallower model
- extra layers: set as identity
- at least the same training error

Optimization difficulties: solvers cannot find the solution when going deeper...


Slide from Kaiming He's talk at ICCV 2015 ImageNet and COCO joint workshop
Plain network

\[
x \\ \downarrow \text{weight layer} \\ \downarrow \text{relu} \\ H(x)
\]

Residual network

\[
F(x) \\ \downarrow \text{relu} \\ \downarrow \text{weight layer} \\ \downarrow \text{weight layer} \\ \downarrow \text{relu} \\ H(x) = F(x) + x
\]

\[
F(x) = W_2 \sigma(W_1 x)
\]
CIFAR-10 experiments

- Deep ResNets can be trained without difficulties
- Deeper ResNets have **lower training error**, and also lower test error

MSRA @ ILSVRC & COCO 2015 Competitions

• **1st places in all five main tracks**
  - ImageNet Classification: “Ultra-deep” (quote Yann) **152-layer nets**
  - ImageNet Detection: **16%** better than 2nd
  - ImageNet Localization: **27%** better than 2nd
  - COCO Detection: **11%** better than 2nd
  - COCO Segmentation: **12%** better than 2nd

*improvements are relative numbers


---

Slide from Kaiming He's talk at ICCV 2015 ImageNet and COCO joint workshop
“ResNets are currently by far state of the art Convolutional Neural Network models and are the default choice for using ConvNets in practice (as of May 10, 2016).” [From http://cs231n.github.io/convolutional-networks/#case]

- ResNet codes available for different frameworks
  - Keras: https://github.com/raghakot/keras-resnet
  - Tensorflow https://github.com/ry/tensorflow-resnet
  - Torch https://github.com/KaimingHe/resnet-1k-layers

- Good blog post explaining ResNets, Inception Xception architectures.
Object detection
Object detection

Task: given an image and an object class, find its instance(s):

Desired result:

aeroplane
The ConvNet approach to object detection

Image

Object proposals or candidates
- Generic (class independent)
- Typically around 1000s (much larger than the # of object instances in the image, but much smaller than the # of total sliding windows in the image)

Each proposal

ConvNet

Estimated class of the proposal

E.g. Fast RCNN (Girshick, ICCV 2015) uses image segmentation (the Selective Search algorithm) to obtain object proposals.
Example of Selective Search [Ujlings et al. (2013)]
Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun

NIPS, June 2015; IEEE TPAMI, June 2017
Novelty

Replaces the segmentation-based, separate “object proposal generator” with a neural network called the “Region Proposal Network (RPN)”.

Uses a ConvNet for both

- Generating object proposals,
- And classifying them.
Faster RCNN’s architecture:

- Region Proposal Network:
  - Proposals
  - Conv layers
  - Feature maps
  - RoI pooling
  - Classifier

- Region Proposal Network:

Code is available on GitHub.

CPU only mode takes 17 seconds per image. Using an optimized linear algebra package (OpenBLAS), this time goes down to 4 seconds.

On Tesla K40 GPU, it takes 0.25 seconds per image.
Other state-of-the-art object detectors

- Single-shot detection (SSD) family
  - Focal loss for dense object detection (Lin et al. ICCV 2017 – Best paper award)
- R-FCN
- Good reference for state-of-the-art: Huang et al. CVPR 2017
Artistic Style Transfer
Gatys et al. (2016) “Image style transfer using convolutional neural networks”
Gatys et al. (2016) “Image style transfer using convolutional neural networks”
Gatys et al. (2016) “Image style transfer using convolutional neural networks”
ConvNet1 captures style

ConvNets are VGG nets

ConvNet3 generates output image by using back-propagation to minimize $L_{total}$ by changing $x$.

ConvNet2 captures content
Self-driving cars
[Bojarski et al. 2016] Source of images
Output: vehicle control

Fully-connected layer
Fully-connected layer
Fully-connected layer

Convolutional feature map
64@1x18

Convolutional feature map
64@3x20

Convolutional feature map
48@5x22

Convolutional feature map
36@14x47

Convolutional feature map
24@31x98

Normalized input planes
3@66x200

Input planes
3@66x200
Visual explanations for ConvNet classification
There are several ways to visualize what CNNs learn.

- Directly plotting the learned filters:
• Showing the layer activations, i.e. feature maps

A very good example:

3D visualization of a CNN on handwritten digit classification: 
http://scs.ryerson.ca/~aharley/vis/conv/
• Retrieving images that maximally activate a neuron

[From http://cs231n.github.io/understanding-cnn/]
• Occluding parts of the image

[From http://cs231n.github.io/understanding-cnn/]
Another way: Grad-CAM

- Gradient weighted class activation mapping [Selvaraju et al. (2016)].

- Applicable to any ConvNet (that was trained for image classification)

- Uses class-specific gradient information flowing into the final convolutional layer.

- Given an image and a class label, visually explains which part of the image is responsible for the class label.
Grad-CAM for “cat”

Grad-CAM for “dog”
How does it work?

\[ \alpha^c_k = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A^k_{i,j}} \]

- \( y \): predicted class vector
- \( A \): final conv layer (has \( k \) channels)

\[ L^c_{Grad-CAM} = ReLU \left( \sum_k \alpha^c_k A^k \right) \]

linear combination
A very educative post with amazing visualizations

Different optimization objectives show what different parts of a network are looking for.

- $n$: layer index
- $x, y$: spatial position
- $z$: channel index
- $k$: class index

- **Neuron**: $layer_n[x,y,z]$
- **Channel**: $layer_n[::,z]$
- **Layer/DeepDream**: $layer_n[::,::]^2$
- **Class Logits**: pre_softmax[k]
- **Class Probability**: softmax[k]
• Finally, ConvNets are not just for images

• They are used for speech recognition.

• Even for natural language understanding (next slide).
Each row represents a word (word2vec)

Words sharing common context have similar word2vec vectors. Also learned by a neural network.
References (for week 5 & 6)


References (for week 5 & 6)


References (for week 5 & 6)


