Training, Transfer Learning, & Generative Models

By Abe Davis
With some slides from
Jin Sun, Noah Snavely, Philipp Isola
Announcements

• Project 5 (Convolutional Neural Networks) released today
  • Due Wednesday, April 29
• Take-home final exam planned May 11-14
This Lecture (and maybe part of the next one)

• Visualizing Deep Classification
• A Review of Overfitting
• Regularization in Deep Learning
• How to Train Deep Nets
• Transfer Learning
• Generative Models
• Transpose Convolution
Visualizing Linear Classification

Classification Problem: Separate Red & Blue

Linear Solution

Visualizing Classification With a Neural Network

Example Network

Classification Results for Every Point in Original Space

Classification Results for Every Point in Transformed Feature Space

Demo

https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html
What Makes Training Deep Nets Hard?

• It's easy to get high training accuracy:
  • Use a huge, fully connected network with tons of layers
  • Let it memorize your training data

• It's hard to get high test accuracy

This would be an example of overfitting
Related Question: Why Convolutional Layers?

• A fully connected layer can generally represent the same functions as a convolutional one
  • Think of the convolutional layer as a version of the FC layer with constraints on parameters

• What is the advantage of CNNs?
A Review of Overfitting
Overfitting: More Parameters, More Problems

- Non-Deep Example: consider the function \( x^2 + x \)
- Let’s take some noisy samples of the function...
Overfitting: More Parameters, More Problems

• Now lets fit a polynomial to our samples of the form \( P_N(x) = \sum_{k=0}^{N} x^k p_k \)
Overfitting: More Parameters, More Problems

• A Model with more parameters can represent more functions

• E.g.,: if \( P_N(x) = \sum_{k=0}^{N} x^k p_k \) then \( P_2 \subset P_{15} \)

• More parameters will often **reduce training error** but **increase testing error**. This is **overfitting**.

• When overfitting happens, models do not generalize well.
Deep Learning: More Parameters, More Problems?

• More parameters let us represent a larger space of functions

• The larger that space is, the harder our optimization becomes

• This means we need:
  • More data
  • More compute resources
  • Etc.
Deep Learning: More Parameters, More Problems?

A convolutional layer looks for components of a function that are spatially-invariant.
How to Avoid Overfitting: Regularization

• In general:
  • More parameters means higher risk of overfitting
  • More constraints/conditions on parameters can help

• If a model is overfitting, we can
  • Collect more data to train on
  • Regularize: add some additional information or assumptions to better constrain learning

• Regularization can be done through:
  • the design of architecture
  • the choice of loss function
  • the preparation of data
  • ...

Regularization: Architecture Choice

- “Bigger” architectures (typically, those with more parameters) tend to be more at risk of overfitting.
Regularization: Dropout

- At training time, randomly “drop” (zero out) some fraction of the connections in your network

- This will prevent your network from relying too heavily on any specific connections

- Encourages redundancy/consensus across various paths through the network

Regularization: In the Loss Function

\[ L = L_{\text{data}} + L_{\text{reg}} \]

\[ L_{\text{reg}} = \lambda \frac{1}{2} \|W\|_2^2 \]

[Andrej Karpathy http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html]
Regularization: In Data Preparation

Preprocess the data so that learning is better conditioned:

\[
X = \text{np.mean}(axis=0, \text{keepdims}=\text{True})
\]

\[
X /= \text{np.std}(axis=0, \text{keepdims}=\text{True})
\]

Figure: Andrej Karpathy
Regularization: In Data Preparation

For ConvNets, typically only the mean is subtracted.

A per-channel mean also works (one value per R,G,B).

Figure: Alex Krizhevsky
Augment the data — extract random crops from the input, with slightly jittered offsets. Without this, typical ConvNets (e.g. [Krizhevsky 2012]) overfit the data.

E.g. 224x224 patches extracted from 256x256 images

Randomly reflect horizontally

Perform the augmentation live during training

Figure: Alex Krizhevsky
Putting It All Together: How To Train Deep Nets

Roughly speaking:

Gather labeled data

Find a ConvNet architecture

Minimize the loss
Training a Convolutional Neural Network

• Split and preprocess your data
• Choose your network architecture
• Initialize the weights
• Find a learning rate and regularization strength/strategy
• Minimize the loss and monitor the progress
• Fiddle with things until they work
(1) Data Pre-Processing

Examples:

• Normalizing and centering Data
• Data Augmentation
  • Random Cropping
  • Mirror Flips
(2) Choose your architecture

https://playground.tensorflow.org/
(we will come back to this later)
(2) Choose your architecture

- **AlexNet**
  - [Krizhevsky et al. NIPS 2012]

- **GoogLeNet**
  - [Szegedy et al. CVPR 2015]

- **VGG Net**
  - [Simonyan & Zisserman, ICLR 2015]

- **ResNet**
  - [He et al. CVPR 2016]

Very common modern choice
(3) Initialize Your Weights

Set the weights to small random numbers:

\[ W = \text{np.random.randn}(D, H) \times 0.001 \]

(matrix of small random numbers drawn from a Gaussian distribution)

Set the bias to zero (or small nonzero):

\[ b = \text{np.zeros}(H) \]

(if you use ReLU activations, folks tend to initialize bias to small positive number)
(3) Start with a Small Portion of the Data

```python
model = init_two_layer_model(32*32*3, 50, 10)  # input size, hidden size, number of classes
trainer = ClassifierTrainer()
X_tiny = X_train[:20]  # take 20 examples
y_tiny = y_train[:20]
best_model, stats = trainer.train(X_tiny, y_tiny, X_tiny, y_tiny,
model, two_layer_net,
num_epochs=200, reg=0.0,
update='sgd', learning_rate_decay=1,
sample_batches = False,
learning_rate=1e-3, verbose=True)
```

The above code:
- take the first 20 examples from CIFAR-10
- turn off regularization (reg = 0.0)
- use simple vanilla ‘sgd’
(3) Start with a Small Portion of the Data

```python
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
X_tiny = X_train[:20] # take 20 examples
y_tiny = y_train[:20]
best_model, stats = trainer.train(X_tiny, y_tiny, X_tiny, y_tiny,
                                 model, two_layer_net,
                                 num_epochs=200, reg=0.0,
                                 update='sgd', learning_rate_decay=1,
                                 sample_batches = False,
                                 learning_rate=1e-3, verbose=True)
```

Details:

' sgld ': vanilla gradient descent (no momentum etc)

learning_rate_decay = 1: constant learning rate

sample_batches = False (full gradient descent, no batches)

epochs = 200: number of passes through the data

*Slide: Andrej Karpathy*
(3) Start with a Small Portion of the Data

100% accuracy on the training set (good)
(4) Find a learning rate

- Too high won’t converge
- Too low will converge slowly
Aside: Some Training Vocabulary

• An *Epoch* is one complete pass through your training data

• An *iteration* of SGD happens on a batch of examples.

• The *Batch Size* is the number of examples in a single training batch.

• The number of iterations per epoch depends on the total number of examples divided by the batch size.
How do we change the learning rate over time?

Various choices:

- Step down by a factor of 0.1 every 50,000 mini-batches (used by SuperVision [Krizhevsky 2012])
- Decrease by a factor of 0.97 every epoch (used by GoogLeNet [Szegedy 2014])
- Scale by $\sqrt{1-t/max_t}$ (used by BVLC to re-implement GoogLeNet)
- Scale by $1/t$
- Scale by $\exp(-t)$
Summary of things to fiddle

- Network architecture
- Learning rate, decay schedule, update type
- Regularization (L2, L1, maxnorm, dropout, …)
- Loss function (softmax, SVM, …)
- Weight initialization
Summary of things to fiddle

- Network architecture
- Learning rate, decay schedule, update type (+batch size)
- Regularization (L2, L1, maxnorm, dropout, …)
- Loss function (softmax, SVM, …)
- Weight initialization
Questions?
Demo

https://playground.tensorflow.org/
(we will come back to this later)
Transfer Learning

“You need a lot of data if you want to train/use CNNs”
Transfer Learning

“You need a lot of data if you want to train/use CNNs”
Transfer Learning with CNNs

1. Train on Imagenet

- FC-1000
- FC-4096
- FC-4096
- MaxPool
- Conv-512
- Conv-512
- MaxPool
- Conv-512
- Conv-512
- MaxPool
- Conv-256
- Conv-256
- MaxPool
- Conv-128
- Conv-128
- MaxPool
- Conv-64
- Conv-64
- Image
Transfer Learning with CNNs

1. Train on Imagenet
   - FC-1000
   - FC-4096
   - FC-4096
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-266
   - Conv-266
   - MaxPool
   - Conv-128
   - Conv-128
   - MaxPool
   - Conv-64
   - Conv-64
   - Image

2. Small Dataset (C classes)
   - FC-C
   - FC-4096
   - FC-4096
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-266
   - Conv-266
   - MaxPool
   - Conv-128
   - Conv-128
   - MaxPool
   - Conv-64
   - Conv-64
   - Image

   *Reinitialize this and train*
   *Freeze these*
Transfer Learning with CNNs

1. Train on Imagenet
   - FC-1000
   - FC-4096
   - FC-4096
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-64
   - Conv-64
   - Image

2. Small Dataset (C classes)
   - FC-C
   - FC-4096
   - FC-4096
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-64
   - Conv-64
   - Image

   Reinitialize this and train

3. Bigger dataset
   - FC-C
   - FC-4096
   - FC-4096
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-64
   - Conv-64
   - Image

   Train these
   - With bigger dataset, train more layers
   - Freeze these

   Lower learning rate when finetuning; 1/10 of original LR is good starting point
<table>
<thead>
<tr>
<th>More specific</th>
<th>very similar dataset</th>
<th>very different dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>More generic</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>very little data</th>
<th>?</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>quite a lot of data</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>More specific</td>
<td>More generic</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td><strong>FC-1000</strong></td>
<td><strong>MaxPool</strong></td>
<td></td>
</tr>
<tr>
<td><strong>FC-4096</strong></td>
<td><strong>Conv-512</strong></td>
<td></td>
</tr>
<tr>
<td><strong>FC-4096</strong></td>
<td><strong>MaxPool</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MaxPool</strong></td>
<td><strong>Conv-512</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Conv-512</strong></td>
<td><strong>MaxPool</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Conv-512</strong></td>
<td><strong>Conv-256</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MaxPool</strong></td>
<td><strong>Conv-256</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MaxPool</strong></td>
<td><strong>Conv-128</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Conv-128</strong></td>
<td><strong>MaxPool</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MaxPool</strong></td>
<td><strong>Conv-64</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Conv-64</strong></td>
<td><strong>Conv-64</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Image</strong></td>
<td><strong>Conv-64</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>very similar dataset</th>
<th>very different dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>very little data</strong></td>
<td>Use Linear Classifier on top layer</td>
<td>?</td>
</tr>
<tr>
<td><strong>quite a lot of data</strong></td>
<td>Finetune a few layers</td>
<td>?</td>
</tr>
<tr>
<td>More specific</td>
<td>More generic</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>very little data</th>
<th>very similar dataset</th>
<th>very different dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Linear Classifier on top layer</td>
<td>You’re in trouble... Try linear classifier from different stages</td>
<td></td>
</tr>
</tbody>
</table>

| quite a lot of data | Finetune a few layers | Finetune a larger number of layers |
Transfer learning with CNNs is pervasive...
(it’s the norm, not an exception)

Object Detection
(Fast R-CNN)

Image Captioning: CNN + RNN

Figure copyright Ross Girshick, 2015. Reproduced with permission.

Figure copyright IEEE, 2015. Reproduced for educational purposes.
Transfer learning with CNNs is pervasive…
(it’s the norm, not an exception)

Object Detection
(Fast R-CNN)

CNN pretrained
on ImageNet

Image Captioning: CNN + RNN

Copyright 2015 by Ross Girshick. Reproduced with permission.

Copyright 2015 by IEEE. Reproduced for educational purposes.
Transfer learning with CNNs is pervasive…
(it’s the norm, not an exception)

Object Detection
(Fast R-CNN)

CNN pretrained on ImageNet

Image Captioning: CNN + RNN

Word vectors pretrained with word2vec
Some Takeaways

Have some dataset of interest but it has < ~1M images?

1. Find a very large dataset that has similar data, train a big ConvNet there
2. Transfer learn to your dataset

Deep learning frameworks provide a “Model Zoo” of pretrained models so you don’t need to train your own

TensorFlow: https://github.com/tensorflow/models
PyTorch: https://github.com/pytorch/vision
Questions?
Autoencoders: Unsupervised Dimensionality Reduction

- Learn a transformation into some compressed space (encoder)
- Learn a transformation from compressed space back to original content (decoder)
- Loss function can be difference between input and decoded output

- Does not require labels!
Autoencoders: Unsupervised Dimensionality Reduction

• Good way to learn useful features from large amounts of unlabeled data
  • E.g., for transfer learning

• We can do this with CNNs, but we need some way to expand feature dimensionality...

• For this we will use Transpose Convolution
Regular Convolution

- **Stride**: The step size used when computing the convolution
- **Padding**: What is assumed about pixels “outside” of image bounds

Animations from: [https://github.com/vdumoulin/conv_arithmetic](https://github.com/vdumoulin/conv_arithmetic)
Regular Convolution

- **Stride**: The step size used when computing the convolution
- **Padding**: What is assumed about pixels “outside” of image bounds

Animations from: https://github.com/vdumoulin/conv_arithmetic
Regular Convolution

- **Stride**: The step size used when computing the convolution
- **Padding**: What is assumed about pixels “outside” of image bounds

Kernel size: 3x3
Padding: “same” (1)
Stride: 0

Animations from: [https://github.com/vdumoulin/conv_arithmetic](https://github.com/vdumoulin/conv_arithmetic)
Regular Convolution

- **Stride**: The step size used when computing the convolution
- **Padding**: What is assumed about pixels “outside” of image bounds

- Stride is applied to the output and padding is applied to the input

Animations from: https://github.com/vdumoulin/conv_arithmetic
Transpose Convolution: Upscaling Our Data

- Stride applied to input
- Padding applied to output (think of it as removing boundary pixels)

 Animations from: [https://github.com/vdumoulin/conv_arithmetic](https://github.com/vdumoulin/conv_arithmetic)
Generative Models
Abe Davis
Some slides from Jin Sun, Phillip Isola