Lecture 12: CNN implementation, visualization and analysis of results

Machine Learning and Imaging

BME 548L
Roarke Horstmeyer
Our very basic convolutional neural network

Forward pass: from $x_i$ and current $W$'s, find $L_{in}$

Training error

$L_{in}(y, f(W,x)) = \text{cross_entropy}(y, f(W,x))$

Output $y^*$

Model

$y^* = f(W,x)$

Training Data

Ex. $[x_1, y_1]$, Ex. $[x_K, y_K]$
Important components of a CNN

CNN Architecture
- CONV size, stride, pad, depth
- ReLU & other nonlinearities
- POOL methods
- # of layers, dimensions per layer
- Fully connected layers

Loss function & optimization
- Type of loss function
- Regularization
- Gradient descent method
- SGD batch and step size

Architecture choices
Optimization choices

Other specifics: Initialization, dropout, batch normalization, data normalization & augmentation

Knobs to turn to get things to work...
Data augmentation

• Basic idea: to simulate variation that you might actually see in real life
• It’s a form of regularization
• Not an exact science, but try it out – it’s free!
Batch normalization (BN)

• Before BN, training very deep networks was hard
  • If using sigmoid activations, large weights could result in saturation
  • Updating earlier layers’ weights causes the distribution of weights in later layers to shift – the internal covariate shift

• To address this covariate shift, BN “resets” the layer it is applied to by normalizing to 0 mean, 1 variance
  • Mean and variance are computed over the batch at the current iteration

Batch normalization update for inputs $x$:

$$x'(i) = (x(i) - E[x(i)]) / \text{STD}[x(i)]$$

• Mean subtract
• Normalize by standard deviation
Dropout

- At each train iteration, randomly delete a fraction $p$ of the nodes
- Prevents neurons from being lazy
- A form of model averaging
- (related: DropConnect – drop the connections instead of nodes)

Regularization: A common pattern

**Training**: Add some kind of randomness, $z$:

$$y = f_W(x, z)$$

**Testing**: Average out randomness (sometimes approximate)

$$y = f(x) = E_z[f(x, z)] = \int p(z)f(x, z)dz$$

**Example**: Batch Normalization

**Training**: Normalize using stats from random minibatches

**Testing**: Use fixed stats to normalize
Regularization: A common pattern

**Training:** Add some kind of randomness, $z$:

$$y = f_W(x, z)$$

**Testing:** Average out randomness (sometimes approximate)

$$y = f(x) = E_z[f(x, z)] = \int p(z)f(x, z)dz$$

**Example:** Batch Normalization

**Training:** Normalize using stats from random minibatches

**Testing:** Use fixed stats to normalize

Obvious examples: Dropout, data augmentation

Advanced examples: DropConnect, Fractional Max Pooling, Stochastic Depth

Wan et al, “Regularization of Neural Networks using DropConnect”, ICML 2013
Graham, “Fractional Max Pooling”, arXiv 2014
What you’ll typically see...

Better optimization algorithms help reduce training loss
What you’ll typically see...

![Train Loss graph]

Better optimization algorithms help reduce training loss

What you can do to help out training error:
- Optimizer choice
- Optimizer step size
What you’ll typically see…

Better optimization algorithms help reduce training loss

But we really care about error on new data - how to reduce the gap?

What you can do to help out training error:
- Optimizer choice
- Optimizer step size
What you’ll typically see…

What you can do to help out training error:
- Optimizer choice
- Optimizer step size

What you can do to help out training error:
- More regularization!
  - Dropout
- Data normalization
- Data augmentation
- A few other tricks..
Trick #1: Early stopping

Stop training the model when accuracy on the validation set decreases
Or train for a long time, but always keep track of the model snapshot that worked best on val

Slide from http://cs231n.stanford.edu/
Trick #2: Use Model Ensembles

1. Train multiple independent models
2. At test time average their results
   (Take average of predicted probability distributions, then choose argmax)

Enjoy 2% extra performance
Related concept/term: majority voting

E.g., look at same dog image from test data 9X, each w/ uniquely trained model
• Get (let’s say) [6, 3] for output classification
• so guess [1,0] = it’s a dog
• Will do better than running model once!

Related technique: Test Time Augmentation
Also relevant: Dropout-type methods
Trick #3: Transfer learning

Transfer Learning with CNNs

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

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

   - Freeze these
   - Reinitialize this and train

3. Bigger dataset
   - FC-C
   - FC-4096
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-128
   - Conv-128
   - MaxPool
   - Conv-64
   - Conv-64
   - Image

   - Train these
   - With bigger dataset, train more layers
   - Freeze these
   - Lower learning rate when fine-tuning; 1/10 of original LR is good starting point

---

Razavian et al., "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014
### Trick #3: Transfer learning

<table>
<thead>
<tr>
<th>More specific</th>
<th>More generic</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC-1000</td>
<td>Conv-512</td>
</tr>
<tr>
<td>FC-4096</td>
<td>Conv-512</td>
</tr>
<tr>
<td>MaxPool</td>
<td>Conv-512</td>
</tr>
<tr>
<td>Conv-512</td>
<td>Conv-512</td>
</tr>
<tr>
<td>MaxPool</td>
<td>Conv-256</td>
</tr>
<tr>
<td>Conv-256</td>
<td>Conv-256</td>
</tr>
<tr>
<td>MaxPool</td>
<td>Conv-128</td>
</tr>
<tr>
<td>Conv-128</td>
<td>Conv-128</td>
</tr>
<tr>
<td>MaxPool</td>
<td>Conv-64</td>
</tr>
<tr>
<td>Conv-64</td>
<td>Conv-64</td>
</tr>
<tr>
<td>Image</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td></td>
<td>You’re in trouble… Try linear classifier from different stages</td>
</tr>
<tr>
<td>quite a lot of data</td>
<td>Finetune a few layers</td>
</tr>
<tr>
<td></td>
<td>Finetune a larger number of layers</td>
</tr>
</tbody>
</table>

*Slide from http://cs231n.stanford.edu/*
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

CNN$_{\theta_c}$

Figure copyright Ross Girshick, 2015. Reproduced with permission.

Figure copyright IEEE, 2015. Reproduced for educational purposes.

Slide from http://cs231n.stanford.edu/
Trick #4: Hyperparameter optimization

For learning_rate in range(9):

For gradient_scheme in range(5):

::

Meta-learning


E. Real, ” Large-Scale Evolution of Image Classifiers,” ICML 2017
Let’s identify the following in some example code

• Structure of input/output
• Train/Validation/Test split
• Cost function
• Optimization method, steps (epochs)
• Batch size
• Data augmentation?
• Dropout?

http://deepimaging.github.io/data/basic_tensorflow_eager_example.ipynb
Visualization: a few options at different stages

During Training:
• tf.summary()
• Tensorboard
  • Plots of loss/accuracy versus iteration, etc.

After Testing:
• Sliding window
• ROC curve, Precision-Recall
• Confusion matrix
• tSNE visualization

• Beyond classification:
  • image-to-image similarity metrics
  • segmentation accuracy and overlap metrics
model = create_model()
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

log_dir="logs/fit" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

model.fit(x=x_train,
y=y_train,
          epochs=5,
          validation_data=(x_test, y_test),
          callbacks=[tensorboard_callback])

%tensorboard --logdir logs/fit
How to examine and present your results: a few options at different stages

Options to examine your test data after processing:
• ROC curve, Precision-Recall
• Confusion matrix
• Sliding window visualization
• Layer visualizations
• Saliency maps etc.
• tSNE visualization