Announcements

• HW4 requires installing software.
• Poster session December 7
Hyperparameter Optimization

Machine Learning – CSE546
Kevin Jamieson
University of Washington

November 28, 2017
Training set

Eval set
$N_{out} = 10$

$N_{hid} = 784$

Training set

Eval set

hyperparameters

learning rate $\eta \in [10^{-3}, 10^{-1}]$

$\ell_2$-penalty $\lambda \in [10^{-6}, 10^{-1}]$

# hidden nodes $N_{hid} \in [10^1, 10^3]$
$N_{out} = 10$
$N_{hid} = 784$
$N_{in} = 10$

Hyperparameters:

$\ell_2$-penalty $\lambda \in [10^{-6}, 10^{-1}]$

learning rate $\eta \in [10^{-3}, 10^{-1}]$

$\#$ hidden nodes $N_{hid} \in [10^1, 10^3]$
Hyperparameters
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$

Eval-loss
0.0577

Eval-loss

N_{out} = 10
N_{hid} = 784

Training set

Eval set

$\hat{f}$

hyperparameters
learning rate $\eta \in [10^{-3}, 10^{-1}]$
\ell_2$-penalty $\lambda \in [10^{-6}, 10^{-1}]$
\# hidden nodes $N_{hid} \in [10^1, 10^3]$
Hyperparameters

\[
\begin{align*}
N_{out} &= 10 \\
N_{hid} &= 784 \\
N_{in} &= 10
\end{align*}
\]

Eval-loss

\[
\begin{align*}
\text{Hyperparameters} &\quad \text{Eval-loss} \\
(10^{-1.6}, 10^{-2.4}, 10^{1.7}) &\quad 0.0577 \\
(10^{-1.0}, 10^{-1.2}, 10^{2.6}) &\quad 0.182
\end{align*}
\]

Learning rate \( \eta \in [10^{-3}, 10^{-1}] \)

\( \ell_2 \)-penalty \( \lambda \in [10^{-6}, 10^{-1}] \)

\# hidden nodes \( N_{hid} \in [10^1, 10^3] \)
Hyperparameters

- Training set
- Eval set

\[ N_{\text{out}} = 10 \]
\[ N_{\text{hid}} \]
\[ N_{\text{in}} = 784 \]

Hyperparameters

- Learning rate \( \eta \in [10^{-3}, 10^{-1}] \)
- \( \ell_2 \)-penalty \( \lambda \in [10^{-6}, 10^{-1}] \)
- \# hidden nodes \( N_{\text{hid}} \in [10^{1}, 10^{3}] \)

Eval-loss

| \( \text{Eval-loss} \) | 0.0577 | 0.182 | 0.0436 |
--- | --- | --- | --- |
| #hidden nodes | 3 | 10 | 10 |
| \( N_{\text{hid}} \) | 10^{1} | 10^{3} | 10^{3} |
Hyperparameters

- \( N_{in} = 784 \)
- \( N_{out} = 10 \)
- \( N_{hid} \)

Training set

<table>
<thead>
<tr>
<th>( \eta )</th>
<th>Eval-loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 10^{-1.6} )</td>
<td>0.0577</td>
</tr>
<tr>
<td>( 10^{-1.0} )</td>
<td>0.182</td>
</tr>
<tr>
<td>( 10^{-1.2} )</td>
<td>0.0436</td>
</tr>
<tr>
<td>( 10^{-2.4} )</td>
<td>0.0919</td>
</tr>
<tr>
<td>( 10^{-2.6} )</td>
<td>0.0575</td>
</tr>
<tr>
<td>( 10^{-2.7} )</td>
<td>0.0765</td>
</tr>
<tr>
<td>( 10^{-1.8} )</td>
<td>0.1196</td>
</tr>
<tr>
<td>( 10^{-1.4} )</td>
<td>0.0834</td>
</tr>
<tr>
<td>( 10^{-1.9} )</td>
<td>0.0242</td>
</tr>
<tr>
<td>( 10^{-1.8} )</td>
<td>0.029</td>
</tr>
</tbody>
</table>

Eval set

Hyperparameters

- Learning rate \( \eta \in [10^{-3}, 10^{-1}] \)
- \( \ell_2 \)-penalty \( \lambda \in [10^{-6}, 10^{-1}] \)
- Number of hidden nodes \( N_{hid} \in [10^1, 10^3] \)
Hyperparameters

(10^{-1.6}, 10^{-2.4}, 10^{1.7})
(10^{-1.0}, 10^{-1.2}, 10^{2.6})
(10^{-1.2}, 10^{-5.7}, 10^{1.4})
(10^{-2.4}, 10^{-2.0}, 10^{2.9})
(10^{-2.6}, 10^{-2.9}, 10^{1.9})
(10^{-2.7}, 10^{-2.5}, 10^{2.4})
(10^{-1.8}, 10^{-1.4}, 10^{2.6})
(10^{-1.4}, 10^{-2.1}, 10^{1.5})
(10^{-1.9}, 10^{-5.8}, 10^{2.1})
(10^{-1.8}, 10^{-5.6}, 10^{1.7})

Eval-loss

0.0577
0.182
0.0436
0.0919
0.0575
0.0765
0.1196
0.0834
0.0242
0.029

Training set

Eval set

$N_{out} = 10$
$N_{hid}$
$N_{in} = 784$

hyperparameters

learning rate $\eta \in [10^{-3}, 10^{-1}]$
$\ell_2$-penalty $\lambda \in [10^{-6}, 10^{-1}]$

# hidden nodes $N_{hid} \in [10^1, 10^3]$
$N_{out} = 10$

$N_{hid}$

$N_{in} = 784$

**Hyperparameters**

$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$
$(10^{-1.0}, 10^{-1.2}, 10^{2.6})$
$(10^{-1.2}, 10^{-5.7}, 10^{1.4})$
$(10^{-2.4}, 10^{-2.0}, 10^{2.9})$
$(10^{-2.6}, 10^{-2.9}, 10^{1.9})$
$(10^{-2.7}, 10^{-2.5}, 10^{2.4})$
$(10^{-1.8}, 10^{-1.4}, 10^{2.6})$
$(10^{-1.4}, 10^{-2.1}, 10^{1.5})$
$(10^{-1.9}, 10^{-5.8}, 10^{2.1})$

**Eval-loss**

0.0577
0.182
0.0436
0.0919
0.0575
0.0765
0.1196
0.0834
0.0242
0.029

How do we choose hyperparameters to train and evaluate?
How do we choose hyperparameters to train and evaluate?

Grid search:

Hyperparameters on 2d uniform grid
How do we choose hyperparameters to train and evaluate?

Grid search:
Hyperparameters on 2d uniform grid

Random search:
Hyperparameters randomly chosen
How do we choose hyperparameters to train and evaluate?

Grid search:
Hyperparameters on 2d uniform grid

Random search:
Hyperparameters randomly chosen

Bayesian Optimization:
Hyperparameters adaptively chosen
Bayesian Optimization:

How does it work?

Hyperparameters *adaptively* chosen

<table>
<thead>
<tr>
<th></th>
<th>GRID</th>
<th>RANDOM</th>
<th>SPEARMINT</th>
<th>AUTOWEKA</th>
<th>HYPEROPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Australian</td>
<td><img src="chart1.png" alt="" /></td>
<td><img src="chart2.png" alt="" /></td>
<td><img src="chart3.png" alt="" /></td>
<td><img src="chart4.png" alt="" /></td>
<td><img src="chart5.png" alt="" /></td>
</tr>
<tr>
<td>Breast</td>
<td><img src="chart1.png" alt="" /></td>
<td><img src="chart2.png" alt="" /></td>
<td><img src="chart3.png" alt="" /></td>
<td><img src="chart4.png" alt="" /></td>
<td><img src="chart5.png" alt="" /></td>
</tr>
<tr>
<td>Diabetes</td>
<td><img src="chart1.png" alt="" /></td>
<td><img src="chart2.png" alt="" /></td>
<td><img src="chart3.png" alt="" /></td>
<td><img src="chart4.png" alt="" /></td>
<td><img src="chart5.png" alt="" /></td>
</tr>
<tr>
<td>Fourclass</td>
<td><img src="chart1.png" alt="" /></td>
<td><img src="chart2.png" alt="" /></td>
<td><img src="chart3.png" alt="" /></td>
<td><img src="chart4.png" alt="" /></td>
<td><img src="chart5.png" alt="" /></td>
</tr>
<tr>
<td>Splice</td>
<td><img src="chart1.png" alt="" /></td>
<td><img src="chart2.png" alt="" /></td>
<td><img src="chart3.png" alt="" /></td>
<td><img src="chart4.png" alt="" /></td>
<td><img src="chart5.png" alt="" /></td>
</tr>
</tbody>
</table>

Budget: 16, 81, 256, 625
~15 dimensional hyperparameter space

Test error of output hyperparameter setting from each searcher after 1 hour per dataset

Li et al 2016
~15 dimensional hyperparameter space

Test error of output hyperparameter setting from each searcher after 1 hour per dataset

Li et al 2016
Recent work attempts to speed up hyperparameter evaluation by stopping poor performing settings before they are fully trained.


### Hyperparameters

- \((10^{-1.6}, 10^{-2.4}, 10^{1.7})\)
- \((10^{-1.0}, 10^{-1.2}, 10^{2.6})\)
- \((10^{-1.2}, 10^{-5.7}, 10^{1.4})\)
- \((10^{-2.4}, 10^{-2.0}, 10^{2.9})\)
- \((10^{-2.6}, 10^{-2.9}, 10^{1.9})\)
- \((10^{-2.7}, 10^{-2.5}, 10^{2.4})\)
- \((10^{-1.8}, 10^{-1.4}, 10^{2.6})\)
- \((10^{-1.4}, 10^{-2.1}, 10^{1.5})\)
- \((10^{-1.9}, 10^{-5.8}, 10^{2.1})\)
- \((10^{-1.8}, 10^{-5.6}, 10^{1.7})\)

### Eval-loss

- 0.0577
- 0.182
- 0.0436
- 0.0919
- 0.0575
- 0.0765
- 0.1196
- 0.0834
- 0.0242
- 0.029

**How computation time was spent?**
Hyperparameter Optimization

In general, hyperparameter optimization is non-convex optimization and little is known about the underlying function (only observe validation loss)

Your time is valuable, computers are cheap: **Do not employ “grad student descent” for hyper parameter search.** Write modular code that takes parameters as input and automate this embarrassingly parallel search. Use crowd resources (see pywren)

Tools for different purposes:
- Very few evaluations: use random search (and pray) or be clever
- Few evaluations and long-running computations: see refs on last slide
- Moderate number of evaluations (but still exp(#params)) and high accuracy needed: use Bayesian Optimization
- Many evaluations possible: use random search. Why overthink it?
Contains slides from...

- LeCun & Ranzato
- Russ Salakhutdinov
- Honglak Lee
- Google images…
Convolution of images

\[(I \ast K)(i,j) = \sum_m \sum_n I(i+m,j+n)K(m,n)\]

(Note to EEs: deep learning uses the word “convolution” to mean what is usually known as “cross-correlation”, e.g., neither signal is flipped)

Image \(I\)

Filter \(K\)

Convolved Feature \(I \ast K\)

Slide credit: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
Convolution of images

$$(I \ast K)(i, j) = \sum_m \sum_n I(i + m, j + n) K(m, n).$$

(Note to EEs: deep learning uses the word “convolution” to mean what is usually known as “cross-correlation”, e.g., neither signal is flipped)

Image $I$

Filter $K$

Convolved Feature

$I \ast K$

Slide credit: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
Convolution of images

\[(I * K)(i, j) = \sum_{m} \sum_{n} I(i + m, j + n)K(m, n)\]

(Note to EEs: deep learning uses the word “convolution” to mean what is usually known as “cross-correlation”, e.g., neither signal is flipped)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Filter</th>
<th>Convolved Image</th>
</tr>
</thead>
</table>
| Edge detection   | \[
\begin{bmatrix}
0 & 1 & 0 \\
1 & -4 & 1 \\
0 & 1 & 0
\end{bmatrix}
\] | ![Convolved Image](image1.png) |
| Sharpen          | \[
\begin{bmatrix}
0 & -1 & 0 \\
-1 & 5 & -1 \\
0 & -1 & 0
\end{bmatrix}
\] | ![Convolved Image](image2.png) |
| Box blur (normalized) | \[\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}\] | ![Convolved Image](image3.png) |
| Gaussian blur (approximation) | \[\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}\] | ![Convolved Image](image4.png) |

Image I

![Image I](image5.png)
Convolution of images

Input image $X$

filters $H_k$

convolved image $H_k * X$

flatten into vector

$$\begin{bmatrix}
\text{vec}(H_1 * X) \\
\text{vec}(H_2 * X) \\
\vdots
\end{bmatrix}$$
Stacking convolved images

64 filters
Stacking convolved images

64 filters
Stacking convolved images

Apply Non-linearity to the output of each layer, Here: ReLu (rectified linear unit)

Other choices: sigmoid, arctan
Stacking convolved images

Apply Non-linearity to the output of each layer, Here: ReLu (rectified linear unit)

Other choices: sigmoid, arctan
Pooling

Pooling reduces the dimension and can be interpreted as “This filter had a high response in this general region”
Pooling Convolution layer

Convolve with 64 6x6x3 filters

MaxPool with 2x2 filters and stride 2
Full feature pipeline

Convolve with 64 6x6x3 filters

MaxPool with 2x2 filters and stride 2

Flatten into a single vector of size $14 \times 14 \times 64 = 12544$

How do we choose the filters?
- Hand design them (digital signal processing, c.f. wavelets)
- Learn them (deep learning)
Some hand-created image features

- SIFT
- Spin Image
- HoG
- RIFT
- Texton
- GLOH

Slide from Honglak Lee
Inspired by Coates and Ng (2012)

Input is CIFAR-10 dataset: 50000 examples of 32x32x3 images

1. Construct set of patches by random selection from images
2. Standardize patch set (de-mean, norm 1, whiten, etc.)
3. Run k-means on random patches
4. Convolve each image with all patches (plus an offset)
5. Push through ReLu
6. Solve least squares for multiclass classification
7. Classify with argmax
Mini case study 2/3

Inspired by Coates and Ng (2012)

Methods of standardization:
Dealing with class imbalance:
Convolution Layer

Example: 200x200 image
- Fully-connected, 400,000 hidden units = 16 billion parameters
- Locally-connected, 400,000 hidden units 10x10 fields = 40 million params
- Local connections capture local dependencies
Could be very complicated...

Learn the convolutional filters using back propagation.

Once learned, you can fix and apply the learned features to other datasets, and only learn the last fully connected layers.
Could be very complicated…

Different architectures have different effects (not well understood)
Example from Krizhevsky, Sutskever, Hinton 2012
<table>
<thead>
<tr>
<th>mite</th>
<th>container ship</th>
<th>motor scooter</th>
<th>leopard</th>
</tr>
</thead>
<tbody>
<tr>
<td>mite</td>
<td>container ship</td>
<td>motor scooter</td>
<td>leopard</td>
</tr>
<tr>
<td>black widow</td>
<td>lifeboat</td>
<td>go-kart</td>
<td>jaguar</td>
</tr>
<tr>
<td>cockroach</td>
<td>amphibian</td>
<td>moped</td>
<td>cheetah</td>
</tr>
<tr>
<td>tick</td>
<td>fireboat</td>
<td>bumper car</td>
<td>snow leopard</td>
</tr>
<tr>
<td>starfish</td>
<td>drilling platform</td>
<td>golfcart</td>
<td>Egyptian cat</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>grille</th>
<th>mushroom</th>
<th>cherry</th>
<th>Madagascar cat</th>
</tr>
</thead>
<tbody>
<tr>
<td>convertible</td>
<td>agaric</td>
<td>dalmatian</td>
<td>squirrel monkey</td>
</tr>
<tr>
<td>grille</td>
<td>mushroom</td>
<td>grape</td>
<td>spider monkey</td>
</tr>
<tr>
<td>pickup</td>
<td>jelly fungus</td>
<td>elderberry</td>
<td>titi</td>
</tr>
<tr>
<td>beach wagon</td>
<td>gill fungus</td>
<td>affordshire bullterrier</td>
<td>indri</td>
</tr>
<tr>
<td>fire engine</td>
<td>dead-man's-fingers</td>
<td>currant</td>
<td>howler monkey</td>
</tr>
</tbody>
</table>
Using neural nets to learn non-linear features

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]
Sequences and Recurrent Neural Networks

Machine Learning – CSE4546
Kevin Jamieson
University of Washington

November 28, 2017
Variable length sequences

Images are usually standardized to be the same size (e.g., 256x256x3)

But what if we wanted to do classification on country-of-origin for names?

**Neural Network**

**Recurrent Neural Network**

Scottish
English
Irish

Hinton
Variable length sequences

*Recurrent* Neural Network

Standard RNN

LSTM

Slide: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Basic Text/Document Processing

Machine Learning – CSE4546
Kevin Jamieson
University of Washington

November 28, 2017
n documents/articles with lots of text

How to get a feature representation of each article?

1. For each document $d$ compute the proportion of times word $t$ occurs out of all words in $d$, i.e. term frequency

$$ TF_{d,t} $$

2. For each word $t$ in your corpus, compute the proportion of documents out of $n$ that the word $t$ occurs, i.e., document frequency

$$ DF_t $$

3. Compute score for word $t$ in document $d$ as

$$ TF_{d,t} \log \left( \frac{1}{DF_t} \right) $$
BeerMapper - Under the Hood

Algorithm requires feature representations of the beers \( \{x_1, \ldots, x_n\} \subset \mathbb{R}^d \)

Two Hearted Ale - Input ~2500 natural language reviews

http://www.ratebeer.com/beer/two-hearted-ale/1502/2/1/

Reviews for each beer

Bag of Words weighted by TF*IDF

Get 100 nearest neighbors using cosine distance

Non-metric multidimensional scaling

Embedding in d dimensions
BeerMapper - Under the Hood

Algorithm requires feature representations of the beers \( \{ x_1, \ldots, x_n \} \subset \mathbb{R}^d \)

**Two Hearted Ale - Weighted Bag of Words:**

Reviews for each beer

- Bag of Words weighted by TF*IDF
- Get 100 nearest neighbors using cosine distance
- Non-metric multidimensional scaling
- Embedding in \( d \) dimensions
BeerMapper - Under the Hood

Algorithm requires feature representations of the beers \( \{x_1, \ldots, x_n\} \subset \mathbb{R}^d \)

**Weighted count vector for the \( i \)th beer:**

\[ z_i \in \mathbb{R}^{400,000} \]

**Cosine distance:**

\[ d(z_i, z_j) = 1 - \frac{z_i^T z_j}{||z_i|| ||z_j||} \]

**Two Hearted Ale - Nearest Neighbors:**
- Bear Republic Racer 5
- Avery IPA
- Stone India Pale Ale
- Founders Centennial IPA
- Smuttynose IPA
- Anderson Valley Hop Ottin IPA
- AleSmith IPA
- BridgePort IPA
- Boulder Beer Mojo IPA
- Goose Island India Pale Ale
- Great Divide Titan IPA
- New Holland Mad Hatter Ale
- Lagunitas India Pale Ale
- Heavy Seas Loose Cannon Hop3
- Sweetwater IPA

---

**Reviews for each beer**

**Bag of Words weighted by TF*IDF**

**Get 100 nearest neighbors using cosine distance**

**Non-metric multidimensional scaling**

**Embedding in \( d \) dimensions**
BeerMapper - Under the Hood

Algorithm requires feature representations of the beers \( \{x_1, \ldots, x_n\} \subset \mathbb{R}^d \)

Find an embedding \( \{x_1, \ldots, x_n\} \subset \mathbb{R}^d \) such that

\[
||x_k - x_i|| < ||x_k - x_j|| \quad \text{whenever} \quad d(z_k, z_i) < d(z_k, z_j)
\]

for all 100-nearest neighbors. \( 10^7 \) constraints, \( 10^5 \) variables

Solve with hinge loss and stochastic gradient descent.

(20 minutes on my laptop) \( d=2, \text{err}=6\% \) \( d=3, \text{err}=4\% \)

Could have also used local-linear-embedding, max-volume-unfolding, kernel-PCA, etc.

Reviews for each beer
Bag of Words weighted by TF*IDF
Get 100 nearest neighbors using cosine distance
Non-metric multidimensional scaling
Embedding in \( d \) dimensions
BeerMapper - Under the Hood

Algorithm requires feature representations of the beers \( \{x_1, \ldots, x_n\} \subset \mathbb{R}^d \)

- Reviews for each beer
- Bag of Words weighted by TF*IDF
- Get 100 nearest neighbors using cosine distance
- Non-metric multidimensional scaling
- Embedding in \( d \) dimensions
BeerMapper - Under the Hood

Algorithm requires feature representations of the beers \( \{x_1, \ldots, x_n\} \subset \mathbb{R}^d \)

Reviews for each beer

Bag of Words weighted by TF*IDF

Get 100 nearest neighbors using cosine distance

Non-metric multidimensional scaling

Embedding in \( d \) dimensions

Sanity check: styles should cluster together and similar styles should be close.
BeerMapper - Under the Hood

Algorithm requires feature representations of the beers \( \{x_1, \ldots, x_n\} \subset \mathbb{R}^d \)

- Reviews for each beer
- Bag of Words weighted by TF*IDF
- Get 100 nearest neighbors using cosine distance
- Non-metric multidimensional scaling
- Embedding in \( d \) dimensions
Other document modeling

Matrix factorization:

1. Construct word x document matrix of counts
2. Compute non-negative matrix factorization
3. Use factorization to represent documents
4. Cluster documents into topics

Also see latent Dirichlet factorization (LDA)