Matrix factorization for representation learning
Reminders/Comments

• Speed of learners:
  • Sources of slowness: for-loops
  • In naive Bayes, for example, mostly have to loop through samples; however, using vector addition within might speed things up

• A more explicit Mini-project specification added to schedule
  • Gives better feedback about marking
  • Note: initial draft (due November 28) should be an almost complete final draft—only thing that can be missing is some results

• I will provide you will practice questions for the final
Neural networks summary

- Discussed basics, including

- Basic architectures (fully connected layers with activations like sigmoid, tanh, and relu)

- How to choose the output loss
  - i.e., still using the GLM formulation

- Learning strategy: gradient descent (called back-propagation)

- Basic regularization strategies

- After reading week, will discuss more advanced topics (for fun)
How else can we learn the representation?

• Discussed how learning can be done in simple ways even for “fixed representations”
  • e.g., learn the centres for radial basis function networks
  • e.g., learn the bandwidths for Gaussian kernel

• Discussed less constrained representation learning setting with neural networks
  • though still quite constrained in our architecture, not just learning any representation

• In general, this problem has been tackled for a long time in the field of unsupervised learning
  • where the goal is to analyze the underlying structure in the data
Representation learning

Neural network

\[
W^{(1)} \in \mathbb{R}^{k \times d}, \quad W^{(2)} \in \mathbb{R}^{m \times k}
\]
\[
d = 4, \quad k = 5, \quad m = 2
\]
\[
\hat{y} = f_2(W^{(2)}f_1(W^{(1)}x))
\]

Dictionary Learning models

or factor models

\[
D \in \mathbb{R}^{k \times d}, \quad W \in \mathbb{R}^{k \times m}
\]
\[
d = 4, \quad k = 5, \quad m = 2
\]
\[
\hat{y} = f_2(hW)
\]
\[
h = \arg \min_{h \in \mathbb{R}^{1 \times k}} L_x(hD, x)
\]
Using factorizations

• Many unsupervised learning and semi-supervised learning problems can be formulated as factorizations
  • PCA, kernel PCA, sparse coding, clustering, etc.

• Also provides a way to embed more complex items into a shared space using co-occurrence
  • e.g., matrix completion for Netflix challenge
  • e.g., word2vec
Intuition (factor analysis)

- Imagine you have test scores from 10 subjects (topics), for 1000 students
- As a psychologist, you hypothesize there are two kinds of intelligence: verbal and mathematical
- You cannot observe these factors (hidden variables)
- Instead, you would like to see if these two factors explain the data, where $x$ is the vector of test scores of a student
- Want to find: $x = d_1 h_1 + d_2 h_2$, where $d_1$ and $d_2$ are vectors $h_1 = \text{verbal intelligence}$ and $h_2 = \text{mathematical intelligence}$
- Having features $h_1$ and $h_2$ would give a compact, intuitive model
Matrix factorization

If $k < d$, then we obtain dimensionality reduction (PCA)
### Example: K-means

$$\begin{align*}
\|x - \sum_{i=1}^{2} 1(x \text{ in cluster } i) d_i\|_2^2 &= \|x - hD\|_2^2 \\
\text{where } h &= [1 \ 0] \text{ or } h = [0 \ 1] \text{ and } D = [d_1 \ ; \ d_2].
\end{align*}$$

Select cluster 1

<table>
<thead>
<tr>
<th>Sample 1</th>
<th>0.1</th>
<th>-3.1</th>
<th>2.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Mean cluster 1

<table>
<thead>
<tr>
<th>Mean cluster 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
</tr>
<tr>
<td>1.2</td>
</tr>
</tbody>
</table>

Mean cluster 2

<table>
<thead>
<tr>
<th>Mean cluster 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

$$X \approx H k$$
Dimensionality reduction

• If set inner dimension $k < d$, obtain dimensionality reduction

• Recall that the product of two matrices $H$ and $D$ has rank at most the minimum rank of $H$ and $D$

\[
\text{rank}(HD) \leq \min(\text{rank}(H), \text{rank}(D))
\]

• Even if $d = 1000$, if we set $k = 2$, then we get a reconstruction of $X$ that is only two-dimensional
  • we could even visualize the data! How?
Principal components analysis

- New representation is $k$ left singular vectors that correspond to $k$ largest singular values
  - i.e., for each sample $x$, the corresponding $k$-dimensional $h$ is the rep

- Not the same as selecting $k$ features, but rather projecting features into lower-dimensional space
Do these make useful features?

● Before we were doing (huge) nonlinear expansions

● PCA takes input features and reduces the dimension

● This constrains the model, cannot be more powerful

● Why could this be helpful?
  ● Constraining the model is a form of regularization: could promote generalization
  ● Sometimes have way too many features (e.g., someone overdid their nonlinear expansion, redundant features), want to extract key dimensions and remove redundancy and noise
  ● Can be helpful for simply analyzing the data, to choose better models
What if the data does not lie on a plane?

• Can do non-linear dimensionality reduction

• Interestingly enough, many non-linear dimensionality reduction techniques correspond to PCA, but first by taking a nonlinear transformation of the data with a (specialized) kernel
  • Isomap, Laplacian eigenmaps, LLE, etc.

• Can therefore extract a lower-dimensional representation on a curved manifold, can better approximate input data in a low-dimensional space
  • which would be hard to capture on a linear surface
Isomap vs PCA

*Note: you don’t need to know Isomap, just using it as an example
Sparse coding

- For sparse representation, usually $k > d$
- Many entries in new representation are zero
Sparse coding illustration

\[ [a_1, \ldots, a_{64}] = [0, 0, \ldots, 0, 0.8, 0, \ldots, 0, 0.3, 0, \ldots, 0, 0.5, 0] \]

(feature representation)

Compact & easily interpretable

Slide credit: Andrew Ng
We’ll look at more examples of matrix factorizations later

Let’s look at now how to solve for these representations
  • for some settings we will have a closed-form solution (e.g., PCA)
  • for most setting, we will again have to derive an iterative update

Finish up example with auto-encoder
Why does the l1 regularizer give sparse representations?
  • behaves like the l0 regularizer

What about the l2 regularizer?
  • the l2 regularizer prefers to more uniformly squash values
  • in fact, picking an l2 regularizer on both H and D ends up corresponding to PCA (subspace representations) → the interaction of having an l2 on both seems to prefer to zero out entire rows of H and columns of D (relaxed rank PCA)
l1 regularizer and l0 regularizer

\[ l_0(w) = \sum_{i=1}^{d} 1(w_i \neq 0) = \text{# non-zero entries} \]

\[ l_1(w) = \sum_{i=1}^{d} |w_i| \]

- l1 regularizer in practice behaves similarly to l0 regularizer

- Before we used it for feature selection
  - regularized weights \( w \) in \( Xw = y \)

- Here we are using it on a matrix, so again we are doing feature selection, but separately for each sample

\[ \|H\|_{1,1} = \sum_{i=1}^{k} \sum_{j=1}^{t} |H_{ij}| \]
For other settings

- If there is no closed form solution, we will do as before: compute the gradient and do gradient descent

- Step 1: Compute gradient with respect to $H$, for fixed $D$, update $H = H - \alpha \text{grad}_H$

- Step 2: Compute gradient with respect to $D$, for fixed $H$, update $D = D - \alpha \text{grad}_D$

- Natural question: with neural networks, we updated both $W1$ and $W2$ simultaneously; why do we alternate between the two variables here?
Alternating methods

• Alternating steepest descent: step in direction of gradient in alternating fashion
  • seems to have nice time, convergence trade-offs

• Alternating minimization: solve for one variable, with the other fixed, in alternating fashion
  • each step corresponds to a batch gradient descent solution with one of the variables fixed
  • more traditional approach with well-known convergence properties

• Which one you use likely depends on your setting; alternating steepest descent is likely a better way in general, if there are computation time restrictions

• Note: this is related to EM, as we will see later (viterbi EM)
What are the distributional assumptions?

• If try to factorize $X$ into $HD$, making an assumption that $p(x \mid \mu = hD)$ is Gaussian, with some fixed covariance
  • weighted $l_2$-loss gives a different covariance for each entry

• What if the data is binary (not Gaussian) or Poisson distributed? (or some other distribution)
  • again, we can use generalized linear models to generalize the distribution $p(x \mid hD)$ to exponential families
  • See e.g., paper on exponential family PCA: “A generalization of principal component analysis to the exponential family”, Collins et al., 2002