Learning Relevant Features of Data with Multi-Scale Tensor Networks

Presenters: Tayssir Doghri, Aayushi Kulshrestha, Tapopriya Majumdar

E. Miles Stoudenmire

March 19, 2019
Paper Outline

- Introduction
- Applying decomposition technique to matrices, which would then be extended to high order tensors
- Unsupervised Coarse Graining
- Experiment on classification task
- Mixing prior
- Conclusion
High Level Idea

- Provides a method to compress data originally in high dimensional space
- Preserves data properties at large scale while normalizing over smallest length scales
- Main idea comes from physics
- Eg. Looking at temperature of the entire system to understand state instead of dynamics about each particle
- Significantly reduces the size of feature space

(Stoudenmire., 2018)
Research Significance

- Computational Efficiency
- Building block for many machine learning tasks
- The algorithm proposed is unsupervised
- Can be applied to very large datasets with a large set of features
Main Contributions

- Uses tensor networks to produce a hierarchical representation of data using low order tensors
- Unsupervised learning based on statistical properties of data
- Only a single topmost layer of tensor needs to be optimized based on task
- Can be used with prior estimates of weights to make learning faster
Key Points

- Compressed space is represented using a layered tree tensor network
- The algorithm scales linearly with both the dimension of the input and training set size
- Uses kernel learning
- The tree tensor network obtained is generalizable to various tasks

(Stoudenmire., 2018)
Consider a model $f(x) = W \cdot \Phi(x)$, where $\Phi(x)$ is the kernel space mapping of the training data.

The optimal weights belong to the span of the training data within feature space.

Using *Representer Theorem*, $W$:

$$W = \sum_{j=1}^{N_T} \alpha_j \Phi^T(x_j)$$

(1)

Quadratic or worse dependence on training set size.
Dealing with scale

\[ W = \sum_{j=1}^{N_T} \alpha_j \Phi^T(x_j) \] (2)

- \( W \) resides in the span of the \( \{\Phi^T(x_j)\}_{j=1}^{N_T} \)

\[ W = \sum_n \beta_n U_n^T \] (3)

where \( U_n^T \) spans the same space as \( \Phi^T(x_j) \)

- One way to obtain \( U^T \) could be by performing SVD on \( \{\Phi(x_j)\} \).

\[ \Phi_j^s = \sum_{nn'} U_{nn'}^s S_{nn'}(V^T)_{nn'} \] (4)

- Truncating singular values very close to zero, \( U^T \) will give the transformation from entire feature space to the reduced parameter space
Covariance to the Rescue!

- SVD is computationally challenging for large datasets
- Alternative method:

\[ \rho_s' = \frac{1}{N_T} \sum_{j=1}^{N_T} \Phi_j^s' (\Phi_j^s)^T = \sum_n U_n^s' P_n (U_n^T)_s \]

(5)

Thus, \( U \) diagonalizes the feature space covariance matrix \( \rho \)

- Truncate directions along which \( \rho \) has a very small projection to rapidly reduce the size of the space needed to carry out learning tasks.

(Stoudenmire., 2018)
Generalization to Tensors

- We define a local feature map $\phi : \mathbb{R} \rightarrow \mathbb{R}^d$, and

$$\Phi(x) = \phi(x_1) \circ \cdots \circ \phi(x_N),$$  \hspace{1cm} (6)

so that now $W$ is a tensor of order $N$ with $d^N$ weight parameters.

$$f(x) = \Phi(x)$$

(c) $W = \begin{array}{c}
\end{array}$
As before, the idea is to compute the eigenvectors of $\rho$, then discard those with smallest eigenvalues.

We think of the collection of feature vectors $\{\Phi(x_j)\}_{j=1}^{NT}$ as a single tensor of order $N + 1$, so that $\rho$ is formed by contracting $\Phi$ and $\Phi^T$ over the index $j$.

$$\Phi^{s_1s_2\cdots s_N}(x_j) = \Phi_j^{s_1s_2\cdots s_N} = \Phi(x_j) \Phi^T(x_j)$$

$\rho = \Phi \Phi^\dagger = \frac{1}{NT} \sum_{j=1}^{NT} \Phi(x_j) \Phi^T(x_j)$

(Stoudenmire., 2018)
Local Isometry

As it is not feasible to diagonalize $\rho$ directly, we look at local isometries, which are third-order tensors $U_{t}^{s_1 s_2}$ satisfying $\sum_{s_1 s_2} U_{t}^{s_1 s_2} U_{s_1 s_2}^{t'} = \delta_{t t'}$

We define $U_1$ such that when it acts on the first two feature space indices, it maximizes the fidelity

$$F = \text{Tr}[\rho] = \frac{1}{N_T} \sum_j \Phi^T \Phi$$

(Stoudenmire., 2018)
Local Isometry (Contd.)

- The fidelity of the approximated $\rho$ is
  \[ F_1 = \frac{1}{N_T} \sum_j \Phi^T U_1 U_1^T \Phi \]  
  (8)

- $F_1 \leq F$.

The reduced covariance matrix $\rho_{12}$ is defined by tracing over all indices of $\rho$ other than $s_1$ and $s_2$, so that
  \[ F_1 = \sum_{s_1 s_2 s_1' s_2'} (U_1^T)^{t}_{s_1 s_2} \rho_{12}^{s_1 s_2} U_1^{s_1 s_2} U_1 U_1^T \]  
  (9)

(Stoudenmire., 2018)
Reduced Covariance matrix

- $\rho_{12} = U_1 P_{12} U_1^T$.
- $U_1$ is truncated keeping the eigenvectors corresponding to the $D$ largest eigenvalues of $\rho_{12}$, where the choice of $D$ depends on a given truncation error cutoff $\epsilon$.

\[
\rho_{12} = \sum_j s^j_{12} \implies \rho_{12} = s_{12}^1 \otimes s_{12}^2 = P_{12} U_{12} U_{12}^T
\]

\[
\rho_{34} = \sum_j s^j_{34} \implies \rho_{34} = s_{34}^1 \otimes s_{34}^2 = P_{12} U_{12} U_{12}^T
\]
Diagonalizing $\rho$

We use the isometry layer to coarse grain the feature vectors, and iterate to diagonalize $\rho$ in $\log_2(N)$ steps.

(Stoudenmire., 2018)
Defining the model

Having determined $\mathcal{U}$, our model is:

$$f(x) = \sum_{t_1 t_2} w_{t_1 t_2} \tilde{\Phi}^{t_1 t_2}(x)$$  \hspace{2cm} (10)

where

$$\tilde{\Phi}^{t_1 t_2}(x) = \sum_{t_1 t_2} \mathcal{U}_{s_1 s_2 \cdots s_N}^{t_1 t_2} \Phi^{s_1 s_2 \cdots s_N}(x)$$  \hspace{2cm} (11)
The local feature map $\phi^{s_n}(x_n)$ is defined by

$$\phi^{s_n=1}(x_n) = 1$$

$$\phi^{s_n=2}(x_n) = x_n$$

We use conjugate gradient to optimize the top tensor $\mathcal{W}$.

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>Accuracy on training set (%)</th>
<th>Accuracy on test set (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{-3}$</td>
<td>107</td>
<td>151</td>
<td>98.75</td>
<td>97.44</td>
</tr>
<tr>
<td>$6 \times 10^{-4}$</td>
<td>328</td>
<td>444</td>
<td>99.68</td>
<td>98.08</td>
</tr>
</tbody>
</table>

Table: Results on MNIST dataset using unsupervised / supervised algorithm
Mixed task-specific / unsupervised algorithm

- Mix the feature space covariance matrix $\rho$ with another matrix based on a specific task:
  \[ \rho_\mu = \mu \hat{\rho}_W + (1 - \mu) \hat{\rho} \]  
  (12)

- Given a prior guess for supervised task weights:
  \[ \hat{\rho}_W = \frac{1}{\text{Tr}(W^T W)} W^T W, \quad \hat{\rho} = \frac{1}{\text{Tr}(\rho)} \rho \]

<table>
<thead>
<tr>
<th>$\mu$</th>
<th>$\epsilon$</th>
<th>$t_1$</th>
<th>$t_2$</th>
<th><strong>Accuracy on training set</strong> (%)</th>
<th><strong>Accuracy on test set</strong> (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>$4 \times 10^{-4}$</td>
<td>279</td>
<td>393</td>
<td>99.798</td>
<td>98.110</td>
</tr>
</tbody>
</table>

Table: Results on MNIST dataset using mixed task-specific / unsupervised algorithm
Partial coarse graining: tree curtain model

- Consider the weights $\mathcal{W}$ as a matrix product state

$$f^\ell(x) = \prod_{i=1}^{\ell} \Phi(x)$$

<table>
<thead>
<tr>
<th>$\mu$</th>
<th>$\epsilon$</th>
<th>Accuracy on training set (%)</th>
<th>Accuracy on test set (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td>$2 \times 10^{-9}$</td>
<td>95.38</td>
<td>88.97</td>
</tr>
</tbody>
</table>

Table: Results on fashion-MNIST dataset using partial coarse graining / unsupervised algorithm

<table>
<thead>
<tr>
<th>Approach</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGBoost</td>
<td>89.8</td>
</tr>
<tr>
<td>AlexNet</td>
<td>89.9</td>
</tr>
<tr>
<td>Two-layer convolutional neural network trained with Keras</td>
<td>87.6</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>93.7</td>
</tr>
</tbody>
</table>

Table: Results for state-of-the-art approaches without preprocessing
Constructing a model using a tree tensor network $U$ and a top tensor $W$.

The algorithm scales linearly in both training set size and input space dimension.

This can be reduced to sublinear using stochastic optimization techniques.

Experimentation can be done with different choices of the covariance matrix $\rho$ and feature map.

Stochastic Gradient Descent can be used for optimization of the top tensor to improve accuracy.

Instead of using tree tensor network, use MERA tensor network.
References
