The Use of Matrix Decompositions to Initialize Artificial Neural Networks

April 25, 2021

Anna Van Boven

Introduction to Neural Networks

An artificial neural network is a computer science algorithm widely used across the field of Artificial Intelligence that uses supervised learning to predict the outcome of future events. Supervised learning works with labeled data to learn a data set and match inputs to outputs. This is opposed to unsupervised learning, which works with unlabeled data to detect structures among the data or cluster the data.
The artificial neural network, thus forward referred to as simply the neural network, models the process of neurons travelling through the brain by recreating the theory that neural pathways that are more commonly used have stronger connections. As Richard Nagyfi states in their paper, *The differences between Artificial and Biological Neural Networks*,“signals can be received from dendrites, and sent down the axon once enough signals were received. This outgoing signal can then be used as another input for other neurons, repeating the process” (Nagyfi). Neural networks recreate this process through the use of perceptrons. Perceptrons are nodes that take in data points x_i each with their own weights, $w_i, 1 \leq i \leq n$. The perceptron then outputs the value $g(\sum_{i=1}^{n} w_i x_i)$, where $g(x)$ is an activation function and the perceptron is only“activated” if the inputs are considered ”strong enough”. Activation functions commonly output either a 0 or 1, or some real number between 0 and 1 (though depending on implementations of a neural network, some functions may output any real number value). The sigmoid function is a common activation function of the form $S(n) = (1 + e^{-n})^{-1}$. A feed-forward neural network consists of multiple columns where the first column is the input layer, followed by k hidden layers of perceptrons where $k \geq 1$, followed by the output layer. The input layer is an $m \times n$ data matrix, where there are n data points each with m observations. Each column is a data point that passes its data to every perceptron in column 1. Each perceptron in column j passes its output as an input to every perceptron in column $j + 1$, where $1 \leq j \leq k$. The kth column of perceptrons is the output layer. A deep learning neural network is one where $k > 1$. This paper will introduce techniques used to implement a single layer neural network, and will discuss ways to adapt these techniques to suit a multi-layer network.
A neural network“learns” by finding weights and thresholds such that the training data is classified, but the model still generalizes well to unseen data. The weights of each edge are adjusted through a process called backpropagation which computes the gradient of the loss function with respect to the weights of the neural network. This process can take a long time, and can substantially impact the runtime for a network, which is very dependent on initial weights and the threshold value of the activation function. There is no principled way to know what starting weights to use for a neural network; the general technique is to randomly initialize the weights, and a poor random initialization can slow down the algorithm. This paper will explore the use of different matrix decompositions to initialize the weights for a neural network and potentially decrease the runtime.

Singular Value Decompositions to Initialize Weights

Before discussing the different approaches to implementing a neural network, it would be beneficial to have an example of how one works. The following example is based on a data set provided by America Chambers. Say each data point is a politician. Each politician either voted yea (1) or nea (0) on ten different issues and can be classified as either a Democrat (0) or a Republican (1). The goal of the neural network, then, would be to take in information about a single politician and be able to predict their party affiliation. Put differently, if y_i is the output by the neural network for data point i and t_i is the known party affiliation for data point i, the neural network seeks to minimize $\sum_{i=1}^{n} \|t_i - y_i\|_F$, where $\|.\|_F$ is the Frobenius norm, defined for an $m \times n$ matrix M as

$$\|M\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2}.$$

The weight matrix W holds the weights of all the edges between two layers in a neural network. W is an $n \times k$ matrix, where k is the number of perceptrons in a given layer. $[W]_{ij}$ holds the weight of data X_i sending to perceptron j. When working with a single layer neural network, for the ideal data set there would exist a weight matrix W such that $T = WX$, where each output can be perfectly predicted by correctly weighting the input data. This is rarely the case; fortunately, for a single layer neural network a projection of W labeled $\hat{W}$ can be computed using a pseudoinverse constructed on the singular value decomposition of the data matrix X.
Let X be a $m \times n$ data matrix with rank r and a singular value decomposition $X = U\Sigma V^*$. Let the SVD matrices be rewritten as follows:

$$U = \begin{bmatrix} U_r & U_{m-r} \end{bmatrix}$$
$$\Sigma = \begin{bmatrix} \Sigma_r & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$$
$$V = \begin{bmatrix} V_r & V_{n-r} \end{bmatrix}$$

$X = U_r \Sigma_r V_r^*$ is called the reduced SVD. Note that Σ_r has only nonzero singular values down the diagonal, and so it is invertible. Note also that since U_r and V_r are unitary, $U^T U = I$ and $V^T V = I$. Let $\boldsymbol{y}$ and $\boldsymbol{b}$ be arbitrary vectors. Then,

$$\begin{aligned} X\boldsymbol{y} &= \boldsymbol{b}, \\ U_r \Sigma_r V_r^T \boldsymbol{y} &= \boldsymbol{b}, \\ \Sigma_r V_r^T \boldsymbol{y} &= U_r^* \boldsymbol{b}, \\ V_r^T \boldsymbol{y} &= \Sigma_r^{-1} U_r^* \boldsymbol{b}. \end{aligned}$$

Knowing that $VV^T\boldsymbol{y}$ gives the projection of $\boldsymbol{y}$ onto the column space of V, this gives

$$\hat{\boldsymbol{y}} = V_r\Sigma_r^{-1}U_r^*\boldsymbol{b}.$$

The SVD pseudoinverse of the matrix X, then, is $V_r\Sigma_r^{-1}U_r^*$.
Going back to finding the weight matrix, $\hat{W}$ can be computed using the SVD pseudoinverse of X:

$$\begin{aligned} T &= WX, \\ \hat{W} &= V_r\Sigma_r^{-1}U_r^*T. \end{aligned}$$

It is important to note that this computation only works in a single layer neural network. With multiple layers, there are multiple weight matrices and the equation $T = WX$ is no longer a suitable goal. The next section will introduce another technique for computing weight matrices that is also well-suited for an adapted neural network. This new technique can also be used to initialize the weights for a deep learning neural network.

Non-negative Matrix Factorization

A variation on the standard feed-forward neural network involves the implementation of autoencoders. Autoencoders are a form of semi-supervised learning – where supervised learning works with labeled data to predict the output, semi-supervised learning works with labeled and unlabeled data to form a representation that can approximate its own input. To implement this, one or more of the hidden layers in the network has less perceptrons than there are data points. The reasoning behind this is that if there are a uniform number of perceptrons in each layer, the neural network will "learn" to pass a data point through a single row without needing to generalize at all. This technique is called bottlenecking, and it forces a compressed knowledge representation as the output layer tries to accurately reconstruct the input layer after passing it through the bottleneck. The goal of autoencoding is to create a network that is sensitive enough to build a reconstruction, but insensitive enough that the model does not overfit to the training data, meaning that it has the ability to reconstruct data from a test set with similar precision to that of the training set.
This paper will discuss a strategy for implementing autoencoders by using a non-negative matrix factorization to approximate the data matrix X, then use these values to initialize the weights between the autoencoded layers. Let the data matrix X be an $m \times n$ matrix with all non-negative entries. Then, X can be approximated by a $m \times p$ non-negative matrix A and a $p \times n$ non-negative matrix S such that X can be approximated by AS, $X \approx AS$. In the above statement, p is the factorization rank and it is guaranteed that $p < \min(m, n)$, and will be discussed in a later section. In non-negative matrix factorizations (NMFs), the matrix A represents the basis of a new subspace where each column is a basis element, and the matrix S holds the coefficients of all the data, where each column of S gives the coordinate of a data point X in the basis A. For the matrix X with the NMF $X \approx AS$, let X_n represent a column of X. Then, $X_n \approx AS_n$. Each vector X_n can be approximated by a linear combination of the columns of A, where the scalars used for this combination are the elements of the vector S_n.
The goal of an NMF is as follows:

$$\begin{aligned}
&\min \|X - AS\|_F \\
&\text{such that} A_{ij} \geq 0, S_{kl} \geq 0 \\
&\text{For every } i \leq m,\ i, j \leq p,\ , l \leq n.
\end{aligned}$$

After initializing A and S with non-negative entries, the two matrices are continually updated with the following update equations until the chosen stopping criteria is met, where the symbol $\otimes$ denotes component-wise multiplication between two matrices, known as the

Hadamard Product:

$$A \leftarrow A \otimes \frac{XS^T}{ASS^T},$$
$$S \leftarrow S \otimes \frac{A^TX}{A^TAS}.$$

Autoencoding in a single-layer neural network is seen as solving the minimization problem

$$\min \sum_{n=1}^{N} \|X_n - f(x_n, W)\|_F$$

where $f(x_n, W)$ is the output of a perceptron in the output layer and W is the weights matrix.
Remember that the columns of S hold the coordinates to map a data point in X to the basis elements in A. Therefore $x_n \approx AS_n$. Each $f(x_n, W)$ approximates an element of the input data, meaning it also approximates the vector AS_n. Keeping in mind the minimization goal of an NMF,

$$\|X_n - f(x_n, W)\|_F \approx \|X_n - AS_n\|_F.$$

The matrix A is not guaranteed to be invertible. However, using the Moore-Penrose pseudoinverse on A gives $A^\circ = (A^*A)^{-1}A^*$, and the matrix S can be cut out of the minimization entirely:

$$\|X_n - AS_n\| \approx \|X_n - AA^\circ x_n\|.$$

With this new equation, A° holds the optimal initial weights for the single-layer neural network. To expand this to a deep-learning neural network (one with multiple layers), the outputs to the last layer construct the new "data matrix", and the same process can be repeated to find the weights between the next layer. Doing this also requires the incorporation of a pooling layer. In short, a pooling layer reduces the dimension of the data set at a certain layer by grouping similar data points together, which is done to reduce the complexity of a network. This is a broad overview of its implementation, but it is enough to give an insight as to why NMFs can be used over SVDs to initialize the weights of a deep-learning neural network. The decomposition is not based on the neural network output, and does not need to be based on the data matrix at all, but rather it bases the factorization off of any non-negative matrix. This means that the outputs of a hidden layer can be combined into the columns of a new matrix, and that matrix can become the new X. Then, the weights between any two layers can be computed using the NMF.
Non-negative matrix factorizations are useful because they can easily generate sparse and

meaningful features in the data. To demonstrate the usefulness of NMFs, the following example is based on an example done by Nicholas Gillis in his paper, *The Why and How of Nonnegative Matrix Factorization.*
Suppose a model is given a data matrix X that represents a group of documents, and the model is looking to classify these documents. Each column of X represents a document, and each row represents a word. The words represented can range from being every word that appears in every document, or being key words that would aid in classification – When looking for documents discussing political parties, searching for the words *Democrat* and *Republican* would quickly weed out irrelevant documents. X_{ij} would represent the frequency of word i in document j. Solving the NMF for X gives $X \approx AS$. The columns of A are the basis elements for X, and each one represents a document. Because $n > p$, there are more documents in total than there are in the columns of A, and the columns of A do not represent the original documents; rather, they hold different combinations of the original documents, which can be considered different topics. The S matrix gives the coordinates of a data point in X in the A matrix – essentially, S holds the weight of how much each document in X is related to each topic in A.
The NMF's ability to pick out meaningful features in the data is what makes them so useful in autoencoding: a NMF will not be constructed around specific points in a data set, but will reflect features in the data. It is important to note that the NMF is not unique. Let AS be the NMF for a matrix X, and let B be an invertible $p \times p$ matrix. Then,

$$AS = ABB^{-1}S.$$

So long as AB and $B^{-1}S$ are nonnegative, then they are valid NMFs of X. One area of research is experimenting with different variations on NMFs to see if a specific variation has an optimal impact on runtime.
One significant drawback to NMFs is that they can only be used when the initial data is non-negative. For nominal data, NMFs can be easily introduced. Nominal data is data used to label variables (for example, 0 for male, 1 for female) and if it is not initially nonnegative, it can be asily converted. However, there are other low-rank matrix approximations that work similar to NMFs that can be applied to data sets with negative entries.

Initializing NMFs

The discussion of NMFs thus far has left two unknowns: finding the value p where for the NMF $X \approx AS$, where A is an $m \times p$ matrix and S is a $p \times n$ matrix, and providing initial values for A and S.
The value p is known as the factorization rank. The goal for determining a value for p is to pick a value large enough that the approximation is accurate, but small enough that it is still able to pick up on key features in the data. In practice, there is no standard technique for picking p. Researchers will vary p and compute the NMF several times for each value, then compare results. Studying the data may give a general range for p – in the above example about document classification, p could be chosen around the estimated number of topics amongst the documents. One technique is to base p off of the rate of decay of the singular values by choosing a value of p that is correlated with the singular values becoming small. Theorem 2 in the section **Theorems and Proofs** gives a way to calculate the Frobenius norm for a square matrix using the singular values of a matrix, $\|M\|_F = \sqrt{\sum \sigma_i^2}$.
The Choosing rule is a method for choosing p based on the latter of the three techniques. Let X be a non-negative $m \times n$ matrix of rank r with the singular value decomposition $X = U\Sigma V$.
Let Σ' be an $r \times r$ diagonal matrix with diagonal entries $\sigma_i, 1 \leq i \leq r$, where $\sigma_i \geq \sigma_{i+1}$. Take

$$s_r = \sum_{i=1}^{r} [\Sigma']_{ii},$$
$$s_k = \sum_{i=1}^{k} [\Sigma']_{ii}, 1 \leq k \leq r.$$

The value of p will be chosen such that $p = k$, and the following inequalities hold:

$$\frac{s_p}{s_r} < \alpha,$$
$$\frac{s_{p+1}}{s_r} \geq \alpha.$$

The value of α normally varies around 0.9 and cannot go above 1. The value is kept high so it is more likely to hold the larger σ values in the set $\{\sigma_1, \ldots, \sigma - p - 1\}$. A lower α will be more likely to exclude large σ_ks and will not accurately estimate the decay of the singular values.
The second unknown is how to initialize the NMFs, A and S. Again, a common strategy is to choose random non-negative values and use the update algorithm to refine the matrices. Three techniques have gained significant traction – one technique works with k-means clustering to initialize the NMF. Another involves constructing the matrix A from the original data. The other works with the SVD of the data matrix. This section will discuss the third technique.

The understanding of this technique requires the Eckhart Young Theorem on Low-Rank Approximation, Theorem 1 in the section **Theorems and Proofs**. The theorem provides a way to minimize the Frobenius distance between a given matrix and an approximating matrix with a lower rank. This theorem can easily be used to gather good initializations for a NMF. Let Z be a non-negative matrix with size $m \times n$ whose singular value decomposition is $Z = U\Sigma V^T$ and for who there exists a non-negative matrix factorization $Z = AS$. Then, by the Eckhart Young Theorem, there exists a matrix $\hat{Z}$ of rank p where $\hat{Z} = U_1\Sigma_1 V_1^T$ that achieves the goal of the NMF, so $AS = U_1\Sigma_1 V_1^T$. The issue with this is that values of $\hat{Z}$ are not guaranteed to be non-negative. To overcome this, let the initial values of A and S be as follows:

$$\begin{aligned} A_0 &= |U_1|, \\ S_0 &= |\Sigma_1 V_1^T|. \end{aligned}$$

Conclusion

Analysis of the Effect of SVD and NMF on Runtime

When backprogagating through a neural network to adjust the weights, there is no guarantee that a low threshold will be met. For this reason, the runtime of a neural network is highly dependent on the initial weights. For large data sets, techniques such as NMF that can point out patterns in the data after a single decomposition have the potential to significantly reduce the runtime by avoiding continual backpropogation over randomly initialized weighted matrices. However, this is not a guarantee.

Techniques for initializing a neural network such as those discussed in this paper are not widely used. This is in part because neural networks are a generally new field of study and different implementations are still being explored. However, part of the reason there is no universally acknowledged way to implement a neural network is because the benefits of different implementations are entirely dependent on both the type of data being used and the size of the data set. The time complexity of computing the singular value decomposition for a matrix is $O(\min(n, m))$. For large enough data sets, this can have a significant impact on the runtime. None of the techniques discussed, or any other technique explored, have been proved to put an upper bound on the distance between the initialized point to the optimal solution – all of the results of the effectiveness of using SVDs and NMFs for initialization come from their observed implementation in neural networks. Therefore there is no guarantee that using these methods will not significantly slow down the runtime of a neural network rather than speed it up. This can easily be seen with NMFs. Finding a NMF is an NP-hard problem, and the benefit of using NMFs is that it will slightly speed up the construction of a neural network, but this does not outweigh the runtime cost it adds for big enough problems.

Applications of Neural Networks

Neural networks are a relatively new algorithm in the field of Artifical Intelligence that can be used with a wide variety of applications. The example in this paper demonstrated the application of networks in text classification, which is used for web searching and language classification. A related topic, Named Entity Recognition, uses neural networks to classify data with general labels: recognizing "Vincent Van Gogh" as a name, or "Tacoma" as a location. Networks are also used for speech recognition and spell checking.

In a broad sense, a neural network is a way for a computer to learn implied aspects of data that come naturally to a person, making it an integral part of the structure and future of Artificial Intelligence. In the near future, enough research will be done on the techniques introduced in this paper that there will likely be conclusive answers on the way to initialize a neural network that will produce the optimal runtime.

Theorems and Proofs

Theorem 1. *Let D be an $m \times n$ matrix with rank r and the singular value decomposition $D = U\Sigma V^T$. Then the problem*

$$\min \|D - \hat{D}\|_F$$
$$st\ rank(\hat{D}) \leq r$$

has the following solution:

$$U = \begin{bmatrix} U_1 & U_2 \end{bmatrix}$$
$$\Sigma = \begin{bmatrix} \Sigma_1 & \mathbf{0} \\ \mathbf{0} & \Sigma_2 \end{bmatrix}$$
$$V = \begin{bmatrix} V_1 & V_2 \end{bmatrix}$$

The submatrix U_1 has size $m \times k$, the submatrix Σ_1 has size $k \times k$, and the submatrix V_1 has size $k \times n$, where k is the rank of the approximating matrix and $k \leq r$.
Let $\hat{D}^* = U_1 \Sigma_1 V_1^T$. Then,

$$\|D - \hat{D}^*\|_F = \min_{\hat{D}} \|D - \hat{D}\|_F = \sqrt{\sigma_{k+1}^2 + \cdots + \sigma_r^2}.$$

Theorem 2. *The Frobenius norm for an $n \times n$ matrix A, $\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$, can also be defined as $\|A\|_F = \sqrt{\sum \sigma_i^2}$.*

Let $A = U\Sigma V^T$ be the SVD of the matrix A. Then,

$$\begin{aligned}
\|A\|_F &= \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}, \\
&= \sqrt{tr(A^TA)}, && \text{Theorem 3} \\
&= \sqrt{tr((U\Sigma V^T)^T U\Sigma V^T)}, \\
&= \sqrt{tr((V^T)^T\Sigma^T U^T U\Sigma V^T)}, && \text{Theorem Matrix Multiplication and Transposes} \\
&= \sqrt{tr(V\Sigma U^T U\Sigma V^T)}, && \Sigma \text{ only has nonzero entries on the diagonal} \\
&= \sqrt{tr(V\Sigma\Sigma V^T)}, && U \text{ is unitary} \\
&= \sqrt{tr(V\Sigma^2 V^T)}, \\
&= \sqrt{tr(VV^T\Sigma^2)}, && \text{Theorem 4} \\
&= \sqrt{tr(\Sigma^2)}, && V \text{ is unitary} \\
&= \sqrt{\sum_{i=1}^{n}\sigma_i^2}.
\end{aligned}$$

Theorem 3. *The square of the Frobenius norm for an $n \times n$ matrix A is the trace of A^TA,*

$$tr(A^TA) = \sum_{i=1}^{n}\sum_{j=1}^{n}[A]_{ij}^2.$$

First, define the trace function for the $n\times n$ matrix A as $tr(A) = \sum_{i=1}^{n}[A]_{ii}$. It is important to note that $tr(A) = tr(A^T)$, as the diagonal entries remain the same while transposing a matrix. Then,

$$\begin{aligned} tr(A^TA) &= \sum_{i=1}^{n} [A^TA]_{ii}, \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} [A^T]_{ji}[A]_{ij}, && \text{entry-wise matrix multiplication} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} [A]_{ij}[A]_{ij}, && \text{definition of transpose} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} [A]_{ij}^2. \end{aligned}$$

Theorem 4. *The trace of a square matrix is communicative, $tr(AB) = tr(BA)$.*

Let A and B be two matrices whose product is a square matrix of size n. Then,

$$\begin{aligned} tr(AB) &= \sum_{i=1}^{n} [AB]_{ii}, \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} [A]_{ij}[B]_{ji}, && \text{entry-wise matrix multiplication} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} [B]_{ji}[A]_{ij}, && \text{scalar multiplcation} \\ &= \sum_{i=1}^{n} [BA]_{ii}, \\ &= tr(BA). \end{aligned}$$

Sources

Atif, Syed Muhammad, et al. “Improved SVD-Based Initialization for Nonnegative Matrix Factorization Using Low-Rank Correction.” ScienceDirect, vol. 122, 1 May 2019.

Barata, J. C. A., and M. S. Hussein. “The Moore-Penrose Pseudoinverse. A Tutorial Review of the Theory.” Cornell University, 31 Oct. 2011.

Flenner, Jennifer, and Blake Hunter. “A Deep Non-Negative Matrix Factorization Neural Network.” Claremont Mckenna College, 2016. Jordan, Jeremy. “Introduction to Autoencoders.” Jeremyjordan, 19 Mar. 2018, www.jeremyjordan.me/autoencoders/.

Schafer, Casey. “The Neural Network, Its Techniques and Applications.” Whitman University, 12 Apr. 2016.

Lee, D., Seung, H. Learning the parts of objects by non-negative matrix factorization. Nature 401, 788–791 (1999). https://doi.org/10.1038/44565

Gillis, N. (2014). The why and how of nonnegative matrix factorization. Regularization, Optimization, Kernels, and Support Vector Machines, 12, 257–291.

Gillis, N., amp; Glinuer, F. (2012). A multilevel approach for nonnegative matrix factorization. ScienceDirect, 236(7). Retrieved 2021, from https://www.sciencedirect.com/science/article/pii/S0377042711005334

Nagyfi, R. (2018, September 4). The differences between Artificial and Biological Neural Networks [Web log post]. Retrieved from https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7 Qiao, H. (2014, October 10). New SVD based initialization strategy for Non-negative Matrix Factorization [PDF]. Ithaca, NY: Cornell University.

Squires, S., Prugel-Bennett, A., Miranjan, M. (2017). Rank Selection in Nonnegative Matrix Factorization using Minimum Description Length [PDF]. Southampton, UK: University of Southampton.

Https://medium.com/@datamonsters/artificial-neural-networks-in-natural-language-processing-bcf62aa9151a [Web log post]. (2017, August 17). Retrieved 2021, from Applications of Artificial Neural Networks in Natural Language Processing