Connecting Machine Learning with Shallow Neural Networks

Neural Networks and Deep Learning, Springer, 2018
Chapter 2, Section 2.1
Neural Networks and Machine Learning

• Neural networks are optimization-based learning models.

• Many classical machine learning models use continuous optimization:
  – SVMs, Linear Regression, and Logistic Regression
  – Singular Value Decomposition
  – (Incomplete) Matrix factorization for Recommender Systems

• All these models can be represented as special cases of shallow neural networks!
The Continuum Between Machine Learning and Deep Learning

- Classical machine learning models reach their learning capacity early because they are simple neural networks.

- When we have more data, we can add more computational units to improve performance.
The Deep Learning Advantage

- Exploring the neural models for traditional machine learning is useful because it exposes the cases in which deep learning has an advantage.
  - Add capacity with more nodes for more data.
  - Controlling the structure of the architecture provides a way to incorporate domain-specific insights (e.g., recurrent networks and convolutional networks).

- In some cases, making minor changes to the architecture leads to interesting models:
  - Adding a sigmoid/softmax layer in the output of a neural model for (linear) matrix factorization can result in logistic/multinomial matrix factorization (e.g., word2vec).
Recap: Perceptron versus Linear Support Vector Machine

(a) Perceptron
Loss = \( \max\{0, -y(W \cdot X)\} \)

(b) SVM
Loss = \( \max\{0, 1 - y(W \cdot X)\} \)

- The Perceptron criterion is a minor variation of hinge loss with identical update of \( W \leftarrow W + \alpha y X \) in both cases.

- We update only for misclassified instances in perceptron, but update also for “marginally correct” instances in SVM.
• Loss for positive class training instance at varying values of $\mathbf{w} \cdot \mathbf{x}$. 
What About the Kernel SVM?

- RBF Network for unsupervised feature engineering.
  - Unsupervised feature engineering is good for noisy data.
  - Supervised feature engineering (with deep learning) is good for learning rich structure.
Much of Machine Learning is a Shallow Neural Model

- By minor changes to the architecture of perceptron we can get:
  - Linear regression, Fisher discriminant, and Widrow-Hoff learning ⇒ Linear activation in output node
  - Logistic regression ⇒ Sigmoid activation in output node

- Multinomial logistic regression ⇒ Softmax Activation in Final Layer

- Singular value decomposition ⇒ Linear autoencoder

- Incomplete matrix factorization for Recommender Systems ⇒ Autoencoder-like architecture with single hidden layer (also used in word2vec)
Why do We Care about Connections?

- Connections tell us about the cases that it makes sense to use conventional machine learning:
  - If you have less data with noise, you want to use conventional machine learning.
  - If you have a lot of data with rich structure, you want to use neural networks.
  - Structure is often learned by using deep neural architectures.
- Architectures like convolutional neural networks can use domain-specific insights.
Neural Models for Linear Regression, Classification, and the Fisher Discriminant
[Connections with Widrow-Hoff Learning]

Neural Networks and Deep Learning, Springer, 2018
Chapter 2, Section 2.2
The perceptron (1958) was historically followed by Widrow-Hoff Learning (1960).

Identical to linear regression when applied to numerical targets.

- Originally proposed by Widrow and Hoff for binary targets (not natural for regression).

The Widrow-Hoff method, when applied to mean-centered features and mean-centered binary class encoding, learns the Fisher discriminant.

We first discuss linear regression for numeric classes and then visit the case of binary classes.
Linear Regression: An Introduction

• In linear regression, we have training pairs \((X_i, y_i)\) for \(i \in \{1 \ldots n\}\), so that \(X_i\) contains \(d\)-dimensional features and \(y_i\) contains a numerical target.

• We use a linear parameterized function to predict \(\hat{y}_i = \overline{W} \cdot \overline{X}_i\).

• Goal is to learn \(\overline{W}\), so that the sum-of-squared differences between observed \(y_i\) and predicted \(\hat{y}_i\) is minimized over the entire training data.

• Solution exists in closed form, but requires the inversion of a potentially large matrix.

• Gradient-descent is typically used anyway.
Linear Regression with Numerical Targets: Neural Model

- Predicted output is $\hat{y}_i = \mathbf{W} \cdot \overline{X}_i$ and loss is $L_i = (y_i - \hat{y}_i)^2$.

- Gradient-descent update is $\mathbf{W} \leftarrow \mathbf{W} - \alpha \frac{\partial L_i}{\partial \mathbf{W}} = \mathbf{W} + \alpha (y_i - \hat{y}_i) \overline{X}_i$. 
Widrow-Hoff: Linear Regression with Binary Targets

- For $y_i \in \{-1, +1\}$, we use same loss of $(y_i - \hat{y}_i)^2$, and update of $W \leftarrow W + \alpha (y_i - \hat{y}_i) X_i$.

  - When applied to binary targets, it is referred to as delta rule.

  - Perceptron uses the same update with $\hat{y}_i = \text{sign}\{W \cdot X_i\}$, whereas Widrow-Hoff uses $\hat{y}_i = W \cdot X_i$.

- **Potential drawback:** Retrogressive treatment of well-separated points caused by the pretension that binary targets are real-valued.

  - If $y_i = +1$, and $W \cdot X_i = 10^6$, the point will be heavily penalized for strongly correct classification!

  - Does not happen in perceptron.
Comparison of Widrow-Hoff with Perceptron and SVM

- Convert the binary loss functions and updates to a form more easily comparable to perceptron using $y^2_i = 1$:

- Loss of $(X_i, y_i)$ is $(y_i - W \cdot X_i)^2 = (1 - y_i[W \cdot X_i])^2$
  
  Update: $W \leftarrow W + \alpha y_i (1 - y_i[W \cdot X_i])X_i$

<table>
<thead>
<tr>
<th>Loss Update</th>
<th>Perceptron</th>
<th>$L_1$-Loss SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\max {-y_i(W \cdot X_i), 0}$</td>
<td>$\max {1 - y_i(W \cdot X_i), 0}$</td>
</tr>
<tr>
<td></td>
<td>$W \leftarrow W + \alpha y_i I(-y_i[W \cdot X_i] &gt; 0)X_i$</td>
<td>$W \leftarrow W + \alpha y_i I(1 - y_i[W \cdot X_i] &gt; 0)X_i$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Loss Update</th>
<th>Widrow-Hoff</th>
<th>Hinton’s $L_2$-Loss SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$(1 - y_i(W \cdot X_i))^2$</td>
<td>$\max {1 - y_i(W \cdot X_i), 0}^2$</td>
</tr>
<tr>
<td></td>
<td>$W \leftarrow W + \alpha y_i (1 - y_i[W \cdot X_i])X_i$</td>
<td>$W \leftarrow W + \alpha y_i \max{(1 - y_i[W \cdot X_i]), 0}X_i$</td>
</tr>
</tbody>
</table>
Some Interesting Historical Facts

• Hinton proposed the SVM $L_2$-loss three years before Cortes and Vapnik’s paper on SVMs.
  

  – Hinton’s $L_2$-loss was proposed to address some of the weaknesses of loss functions like linear regression on binary targets.

  – When used with $L_2$-regularization, it behaves identically to an $L_2$-SVM, but the connection with SVM was overlooked.

• The Widrow-Hoff rule is also referred to as ADALINE, LMS (least mean-square method), delta rule, and least-squares classification.
Connections with Fisher Discriminant

• Consider a binary classification problem with training instances \((X_i, y_i)\) and \(y_i \in \{-1, +1\}\).

  – Mean-center each feature vector as \(X_i - \overline{\mu}\).

  – Mean-center the binary class by subtracting \(\sum_{i=1}^{n} y_i/n\) from each \(y_i\).

• Use the delta rule \(\overline{W} \leftarrow \overline{W} + \alpha (y_i - \hat{y}_i) X_i\) for learning.

• Learned vector is the Fisher discriminant!

Neural Models for Logistic Regression

Neural Networks and Deep Learning, Springer, 2018
Chapter 2, Section 2.2
Logistic Regression: A Probabilistic Model

- Consider the training pair \((X_i, y_i)\) with \(d\)-dimensional feature variables in \(X_i\) and class variable \(y_i \in \{-1, +1\}\).

- In logistic regression, the sigmoid function is applied to \(W \cdot X_i\), which predicts the probability that \(y_i\) is +1.

\[
\hat{y}_i = P(y_i = 1) = \frac{1}{1 + \exp(-W \cdot X_i)}
\]

- We want to maximize \(\hat{y}_i\) for positive class instances and \(1 - \hat{y}_i\) for negative class instances.

  - Same as minimizing \(-\log(\hat{y}_i)\) for positive class instances and \(-\log(1 - \hat{y}_i)\) for negative instances.

  - Same as minimizing loss \(L_i = -\log(|y_i/2 - 0.5 + \hat{y}_i|)\).

  - Alternative form of loss \(L_i = \log(1 + \exp[-y_i(W \cdot X_i)])\)
Maximum-Likelihood Objective Functions

• Why did we use the negative logarithms?

• Logistic regression is an example of a maximum-likelihood objective function.

• Our goal is to maximize the product of the probabilities of correct classification over all training instances.
  - Same as minimizing the sum of the negative log probabilities.
  - Loss functions are always additive over training instances.
  - So we are really minimizing $\sum_i -\log(|y_i/2 - 0.5 + \hat{y}_i|)$ which can be shown to be $\sum_i \log(1 + \exp[-y_i(W \cdot \bar{X}_i)])$. 
Logistic Regression: Neural Model

- Predicted output is $\hat{y}_i = 1/(1 + \exp(-W \cdot X_i))$ and loss is $L_i = -\log(|y_i/2 - 0.5 + \hat{y}_i|) = \log(1 + \exp[-y_i(W \cdot X_i)])$.

- Gradient-descent update is $W \leftarrow W - \alpha \frac{\partial L_i}{\partial W}$.

$$W \leftarrow W + \alpha \frac{y_i X_i}{1 + \exp[y_i(W \cdot X_i)]}$$
Interpreting the Logistic Update

• An important multiplicative factor in the update increment is $1/(1 + \exp[y_i(W \cdot X_i)])$.

• This factor is $1 - \hat{y}_i$ for positive instances and $\hat{y}_i$ for negative instances ⇒ Probability of mistake!

• Interpret as: $\overline{W} \leftarrow \overline{W} + \alpha \left[ \text{Probability of mistake on } (X_i, y_i) \right] (y_iX_i)$
Comparing Updates of Different Models

- The unregularized updates of the perceptron, SVM, Widrow-Hoff, and logistic regression can all be written in the following form:

\[
\overline{W} \leftarrow \overline{W} + \alpha y_i \delta(X_i, y_i) X_i
\]

- The quantity \( \delta(X_i, y_i) \) is a mistake function, which is:
  - Raw mistake value \((1 - y_i(\overline{W} \cdot X_i))\) for Widrow-Hoff
  - Mistake indicator whether \((0 - y_i(\overline{W} \cdot X_i)) > 0\) for perceptron.
  - Margin/mistake indicator whether \((1 - y_i(\overline{W} \cdot X_i)) > 0\) for SVM.
  - Probability of mistake on \((X_i, y_i)\) for logistic regression.
- Loss functions are similar (note Widrow-Hoff retrogression).
Other Comments on Logistic Regression

- Many classical neural models use repeated computational units with logistic and tanh activation functions in hidden layers.

- One can view these methods as feature engineering models that stack multiple logistic regression models.

- The stacking of multiple models creates inherently more powerful models than their individual components.
The Softmax Activation Function and Multinomial Logistic Regression
Binary Classes versus Multiple Classes

• All the models discussed so far discuss only the binary class setting in which the class label is drawn from \{-1, +1\}.

• Many natural applications contain multiple classes without a natural ordering among them:
  – Predicting the category of an image (e.g., truck, carrot).
  – *Language models*: Predict the next word in a sentence.

• Models like logistic regression are naturally designed to predict two classes.
Generalizing Logistic Regression

- Logistic regression produces probabilities of the two outcomes of a binary class.

- *Multinomial* logistic regression produces probabilities of multiple outcomes.
  - In order to produce probabilities of multiple classes, we need an activation function with a vector output of probabilities.
  - The *softmax activation function* is a vector-based generalization of the sigmoid activation used in logistic regression.

- Multinomial logistic regression is also referred to as softmax classifier.
The Softmax Activation Function

• The softmax activation function is a natural vector-centric generalization of the scalar-to-scalar sigmoid activation function \( \Rightarrow \) vector-to-vector function.

• Logistic sigmoid activation: \( \Phi(v) = \frac{1}{1 + \exp(-v)} \).

• Softmax activation: \( \Phi(v_1 \ldots v_k) = \frac{1}{\sum_{i=1}^{k} \exp(v_i)} [\exp(v_1) \ldots \exp(v_k)] \)
  
  – The \( k \) outputs (probabilities) sum to 1.

• Binary case of using \( \text{sigmoid}(v) \) is identical to using 2-element softmax activation with arguments \( (v, 0) \).
  
  – Multinomial logistic regression with 2-element softmax is equivalent to binary logistic regression.
Loss Functions for Softmax

- Recall that we use the negative logarithm of the probability of observed class in binary logistic regression.
  - Natural generalization to multiple classes.
  - Cross-entropy loss: Negative logarithm of the probability of correct class.
  - Probability distribution among incorrect classes has no effect.

- Softmax activation is used almost exclusively in output layer and (almost) always paired with cross-entropy loss.
Cross-Entropy Loss of Softmax

- Like the binary logistic case, the loss \( L \) is a negative log probability.

\[
\text{Softmax Probability Vector } \Rightarrow [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_k]\\
[\hat{y}_1 \ldots \hat{y}_k] = \frac{1}{\sum_{i=1}^{k} \exp(v_i)} [\exp(v_1) \ldots \exp(v_k)]
\]

- The loss is \(-\log(\hat{y}_c)\), where \( c \in \{1 \ldots k\} \) is the correct class of that training instance.

- Cross entropy loss is \(-v_c) + \log[\sum_{j=1}^{k} \exp(v_j)]\)
Loss Derivative of Softmax

- Since softmax is almost always paired with cross-entropy loss $L$, we can directly estimate $\frac{\partial L}{\partial v_r}$ for each pre-activation value from $v_1 \ldots v_k$.

- Differentiate loss value of $-v_c + \log[\sum_{j=1}^{k} \exp(v_j)]$

- Like the sigmoid derivative, the result is best expressed in terms of the post-activation values $\hat{y}_1 \ldots \hat{y}_k$.

- The loss derivative of the softmax is as follows:

\[
\frac{\partial L}{\partial v_r} = \begin{cases} 
\hat{y}_r - 1 & \text{If } r \text{ is correct class} \\
\hat{y}_r & \text{If } r \text{ is not correct class}
\end{cases}
\]
Multinomial Logistic Regression

- The $i$th training instance is $(\overline{X}_i, c(i))$, where $c(i) \in \{1 \ldots k\}$ is class index ⇒ Learn $k$ parameter vectors $W_1 \ldots W_k$.
  
  - Define real-valued score $v_r = W_r \cdot \overline{X}_i$ for $r$th class.
  
  - Convert scores to probabilities $\hat{y}_1 \ldots \hat{y}_k$ with softmax activation on $v_1 \ldots v_k$ ⇒ Hard or soft prediction
Computing the Derivative of the Loss

- The cross-entropy loss for the $i$th training instance is $L_i = -\log(\hat{y}_{c(i)})$.

- For gradient-descent, we need to compute $\frac{\partial L_i}{\partial W_r}$.

- Using chain rule of differential calculus, we get:

$$\frac{\partial L_i}{\partial W_r} = \sum_j \left( \frac{\partial L_i}{\partial v_j} \right) \left( \frac{\partial v_j}{\partial W_r} \right) = \frac{\partial L_i}{\partial v_r} \frac{\partial v_r}{\partial W_r} + \text{zero-terms}$$

$$= \begin{cases} -\bar{X}_i(1 - \hat{y}_r) & \text{if } r = c(i) \\ \bar{X}_i \hat{y}_r & \text{if } r \neq c(i) \end{cases}$$
Gradient Descent Update

- Each separator $\overline{W_r}$ is updated using the gradient:

$$\overline{W_r} \leftarrow \overline{W_r} - \alpha \frac{\partial L_i}{\partial \overline{W_r}}$$

- Substituting the gradient from the previous slide, we obtain:

$$\overline{W_r} \leftarrow \overline{W_r} + \alpha \begin{cases} X_i \cdot (1 - \hat{y}_r) & \text{if } r = c(i) \\ -X_i \cdot \hat{y}_r & \text{if } r \neq c(i) \end{cases}$$
Summary

• The book also contains details of the multiclass Perceptron and Weston-Watkins SVM.

• Multinomial logistic regression is a direct generalization of logistic regression.

• If we apply the softmax classifier with two classes, we will obtain $W_1 = -W_2$ to be the same separator as obtained in logistic regression.

• Cross-entropy loss and softmax are almost always paired in output layer (for all types of architectures).
  - Many of the calculus derivations in previous slides are repeatedly used in different settings.
The Autoencoder for Unsupervised Representation Learning

Neural Networks and Deep Learning, Springer, 2018
Chapter 2, Section 2.5
Unsupervised Learning

• The models we have discussed so far use training pairs of the form \((\bar{X}, y)\) in which the feature variables \(\bar{X}\) and target \(y\) are clearly separated.
  
  – The target variable \(y\) provides the *supervision* for the learning process.

• What happens when we do not have a target variable?
  
  – We want to capture a model of the training data without the guidance of the target.
  
  – This is an *unsupervised* learning problem.
Example

- Consider a 2-dimensional data set in which all points are distributed on the circumference of an origin-centered circle.

- All points in the first and third quadrant belong to class $+1$ and remaining points are $-1$.
  - The class variable provides focus to the learning process of the supervised model.
  - An unsupervised model needs to recognize the circular manifold without being told up front.
  - The unsupervised model can represent the data in only 1 dimension (angular position).

- Best way of modeling is data-set dependent $\Rightarrow$ Lack of supervision causes problems
Unsupervised Models and Compression

- Unsupervised models are closely related to compression because compression captures a model of regularities in the data.
  - Generative models represent the data in terms of a compressed parameter set.
  - Clustering models represent the data in terms of cluster statistics.
  - Matrix factorization represents data in terms of low-rank approximations (compressed matrices).

- An autoencoder also provides a compressed representation of the data.
Defining the Input and Output of an Autoencoder

- All neural networks work with input-output pairs.
  - In a supervised problem, the output is the label.

- In the autoencoder, the output values are the same as inputs: replicator neural network.
  - The loss function penalizes a training instance depending on how far it is from the input (e.g., squared loss).
Encoder and Decoder

- Reconstructing the data might seem like a trivial matter by simply copying the data forward from one layer to another.
  - Not possible when the number of units in the middle are constricted.
  - Autoencoder is divided into encoder and decoder.
Basic Structure of Autoencoder

• It is common (but not necessary) for an $M$-layer autoencoder to have a symmetric architecture between the input and output.
  
  – The number of units in the $k$-th layer is the same as that in the $(M - k + 1)$th layer.

• The value of $M$ is often odd, as a result of which the $(M + 1)/2$th layer is often the most constricted layer.
  
  – We are counting the (non-computational) input layer as the first layer.

  – The minimum number of layers in an autoencoder would be three, corresponding to the input layer, constricted layer, and the output layer.
Undercomplete Autoencoders and Dimensionality Reduction

- The number of units in each middle layer is typically fewer than that in the input (or output).
  - These units hold a reduced representation of the data, and the final layer can no longer reconstruct the data exactly.

- This type of reconstruction is inherently *lossy*.

- The activations of hidden layers provide an alternative to linear and nonlinear dimensionality reduction techniques.
Overcomplete Autoencoders and Representation Learning

- What happens if the number of units in hidden layer is equal to or larger than input/output layers?
  
  - There are infinitely many hidden representations with zero error.
  
  - The middle layers often do not learn the identity function.
  
  - We can enforce specific properties on the redundant representations by adding constraints/regularization to hidden layer.

* Training with stochastic gradient descent is itself a form of regularization.

* One can learn sparse features by adding sparsity constraints to hidden layer.
Applications

• Dimensionality reduction ⇒ Use activations of constricted hidden layer

• Sparse feature learning ⇒ Use activations of constrained/regularized hidden layer

• Outlier detection: Find data points with larger reconstruction error
  – Related to denoising applications

• Generative models with probabilistic hidden layers (variational autoencoders)

• Representation learning ⇒ Pretraining
Singular Value Decomposition with Autoencoders

Neural Networks and Deep Learning, Springer, 2018
Chapter 2, Section 2.5
Singular Value Decomposition

- Truncated SVD is the *approximate* decomposition of an $n \times d$ matrix $D$ into $D \approx Q \Sigma P^T$, where $Q$, $\Sigma$, and $P$ are $n \times k$, $k \times k$, and $d \times k$ matrices, respectively.

  - Orthonormal columns of each of $P$, $Q$, and nonnegative diagonal matrix $\Sigma$.

  - Minimize the squared sum of residual entries in $D - Q \Sigma P^T$.

  - The value of $k$ is typically much smaller than $\min\{n, d\}$.

  - Setting $k$ to $\min\{n, d\}$ results in a zero-error decomposition.
Relaxed and Unnormalized Definition of SVD

- **Two-way Decomposition:** Find and \( n \times k \) matrix \( U \), and \( d \times k \) matrix \( V \) so that \( \|D - UV^T\|^2 \) is minimized.

  - Property: At least one optimal pair \( U \) and \( V \) will have mutually orthogonal columns (but non-orthogonal alternatives will exist).

  - The orthogonal solution can be converted into the 3-way factorization of SVD.

  - Exercise: Given \( U \) and \( V \) with orthogonal columns, find \( Q, \Sigma \) and \( P \).

- In the event that \( U \) and \( V \) have non-orthogonal columns at optimality, these columns will span the same subspace as the orthogonal solution at optimality.
Dimensionality Reduction and Matrix Factorization

- Singular value decomposition is a dimensionality reduction method (like any matrix factorization technique).

\[ D \approx UV^T \]

- The \( n \) rows of \( D \) contain the \( n \) training points.

- The \( n \) rows of \( U \) provide the reduced representations of the training points.

- The \( k \) columns of \( V \) contain the orthogonal basis vectors.
The Autoencoder Architecture for SVD

- The rows of the matrix $D$ are input to encoder.

- The activations of hidden layer are rows of $U$ and the weights of the decoder contain $V$.

- The reconstructed data contain the rows of $UV^T$. 
Why is this SVD?

• If we use the mean-squared error as the loss function, we are optimizing $||D - UV^T||^2$ over the entire training data.

  – This is the same objective function as SVD!

• It is possible for gradient-descent to arrive at an optimal solution in which the columns of each of $U$ and $V$ might not be mutually orthogonal.

• Nevertheless, the subspace spanned by the columns of each of $U$ and $V$ will always be the same as that found by the optimal solution of SVD.
Some Interesting Facts

• The optimal encoder weight matrix \( W \) will be the pseudo-inverse of the decoder weight matrix \( V \) if the training data spans the full dimensionality.

\[
W = (V^T V)^{-1} V^T
\]

– If the encoder and decoder weights are tied \( W = V^T \), the columns of the weight matrix \( V \) will become mutually orthogonal.

– Easily shown by substituting \( W = V^T \) above and postmultiplying with \( V \) to obtain \( V^T V = I \).

– This is exactly SVD!

• Tying encoder-decoder weights does not lead to orthogonality for other architectures, but is a common practice anyway.
Better reductions are obtained by using increased depth and nonlinearity.

Crucial to use nonlinear activations with deep autoencoders.
Row-Index to Row-Value Autoencoders: Incomplete Matrix Factorization for Recommender Systems

Neural Networks and Deep Learning, Springer, 2018
Chapter 2, Section 2.6
Recommender Systems

- Recap of SVD: Factorizes $D \approx UV^T$ so that the sum-of-squares of residuals $||D - UV^T||^2$ is minimized.
  
  - Helpful to watch previous lecture on SVD

- In recommender systems (RS), we have an $n \times d$ ratings matrix $D$ with $n$ users and $d$ items.
  
  - Most of the entries in the matrix are unobserved
  
  - Want to minimize $||D - UV^T||^2$ only over the observed entries
  
  - Can reconstruct the entire ratings matrix using $UV^T$ ⇒ Most popular method in traditional machine learning.
Difficulties with Autoencoder

- If some of the inputs are missing, then using an autoencoder architecture will implicitly assume default values for some inputs (like zero).
  - This is a solution used in some recent methods like AutoRec.
  - Does not exactly simulate classical MF used in recommender systems because it implicitly makes assumptions about unobserved entries.

- None of the proposed architectures for recommender systems in the deep learning literature exactly map to the classical factorization method of recommender systems.
Row-Index-to-Row-Value Autoencoder

- Autoencoders map row values to row values.
  - Discuss an autoencoder architecture to map the one-hot encoded row index to the row values.
  - Not standard definition of autoencoder.
  - Can handle incomplete values but cannot handle out-of-sample data.
  - Also useful for representation learning (e.g., node representation of graph adjacency matrix).

- The row-index-to-row-value architecture is not recognized as a separate class of architectures for MF (but used often enough to deserve recognition as a class of MF methods).
Encoder and decoder weight matrices are $U$ and $V^T$.

- Input is one-hot encoded row index (only in-sample)
- Number of nodes in hidden layer is factorization rank.
- Outputs contain the ratings for that row index.
How to Handle Incompletely Specified Entries?

- Each user has his/her own neural architecture with missing outputs.
- Weights across different user architectures are shared.
Equivalence to Classical Matrix Factorization for RS

- Since the two weight matrices are $U$ and $V^T$, the one-hot input encoding will pull out the relevant row from $UV^T$.

- Since the outputs only contain the observed values, we are optimizing the sum-of-square errors over observed values.

- Objective functions in the two cases are equivalent!
Training Equivalence

- For $k$ hidden nodes, there are $k$ paths between each user and each item identifier.

- Backpropagation updates weights along all $k$ paths from each observed item rating to the user identifier.
  - Backpropagation in a later lecture.

- These $k$ updates can be shown to be identical to classical matrix factorization updates with stochastic gradient descent.

- Backpropagation on neural architecture is identical to classical MF stochastic gradient descent.
Advantage of Neural View over Classical MF View

• The neural view provides natural ways to add power to the architecture with nonlinearity and depth.
  – Much like a child playing with a LEGO toy.
  – You are shielded from the ugly details of training by an inherent modularity in neural architectures.
  – The name of this magical modularity is backpropagation.

• If you have binary data, you can add logistic outputs for logistic matrix factorization.

• *Word2vec* belongs to this class of architectures (but direct relationship to nonlinear matrix factorization is not recognized).
Importance of Row-Index-to-Row-Value Autoencoders

• Several MF methods in machine learning can be expressed as row-index-to-row-value autoencoders (but not widely recognized—RS matrix factorization a notable example).

• Several row-index-to-row-value architectures in NN literature are also not fully recognized as matrix factorization methods.
  – The full relationship of word2vec to matrix factorization is often not recognized.
  – Indirect relationship to linear PPMI matrix factorization was shown by Levy and Goldberg.
  – In a later lecture, we show that word2vec is directly a form of nonlinear matrix factorization because of its row-index-to-row-value architecture and nonlinear activation.
Word2vec: The Skipgram Model

Neural Networks and Deep Learning, Springer, 2018
Chapter 2, Section 2.7
Word2Vec: An Overview

- *Word2vec* computes embeddings of words using sequential proximity in sentences.
  
  - If *Paris* is closely related to *France*, then *Paris* and *France* must occur together in small windows of sentences.
    
    * Their embeddings should also be somewhat similar.

  - Continuous bag-of-words predicts central word from context window.

  - Skipgram model predicts context window from central word.
Words and Context

- A window of size $t$ on either side is predicted using a word.

- This model tries to predict the context $w_{i-t}w_{i-t+1}\ldots w_{i-1}$
  $w_{i+1}\ldots w_{i+t-1}w_{i+t}$ around word $w_i$, given the $i$th word in
  the sentence, denoted by $w_i$.

- The total number of words in the context window is $m = 2t$.

- One can also create a $d \times d$ word-context matrix $C$ with
  frequencies $c_{ij}$.

- We want to find an embedding of each word.
Where have We Seen this Setup Before?

- Similar to recommender systems with *implicit feedback*.

- Instead of user-item matrices, we have square word-context matrices.
  
  - The frequencies correspond to the number of times a contextual word (column id) appears for a target word (row id).
  
  - Analogous to the number of units bought by a user (row id) of an item (column id).
  
  - An unrecognized fact is that skipgram *word2vec* uses an almost identical model to current recommender systems.

- Helpful to watch previous lecture on recommender systems with row-index-to-value autoencoders.
**Word2Vec: Skipgram Model**

- Input is the one-hot encoded word identifier and output contains $m$ identical softmax probability sets.
Word2Vec: Skipgram Model

MINIBATCH THE $m$ d-DIMENSIONAL OUTPUT VECTORS IN EACH CONTEXT WINDOW DURING STOCHASTIC GRADIENT DESCENT. THE SHOWN OUTPUTS $y_{jk}$ CORRESPOND TO THE jth OF m OUTPUTS.

- Since the $m$ outputs are identical, we can collapse the $m$ outputs into a single output.

- Mini-batch the words in a context window to achieve the same effect.

- Gradient descent steps for each instance are proportional to $d \Rightarrow$ Expensive.
Word2Vec: Skipgram Model with Negative Sampling

MINIBATCH THE $m$ $d$-DIMENSIONAL OUTPUT VECTORS IN EACH CONTEXT WINDOW DURING STOCHASTIC GRADIENT DESCENT. THE SHOWN OUTPUTS $y_{jk}$ CORRESPOND TO THE jth OF m OUTPUTS.

- Change the softmax layer into sigmoid layer.
- Of the $d$ outputs, keep the positive output and sample $k$ out of the remaining $d - 1$ (with log loss).
- Where have we seen missing outputs before?
Can You See the Similarity?

\[ U = [u_{jq}] \]
\[ V = [v_{qj}] \]

The vast majority of zero outputs are missing (negative sampling).

- Main difference: Sigmoid output layer with log loss.
Word2Vec is Nonlinear Matrix Factorization

- Levy and Goldberg showed an *indirect* relationship between *word2vec* SGNS and PPMI matrix factorization.

- We provide a much more direct result in the book.
  - Word2vec is (weighted) logistic matrix factorization.
  - Not surprising because of the similarity with the recommender architecture.
  - Logistic matrix factorization is already used in recommender systems!
  - Neither the *word2vec* authors nor the community have pointed out this *direct* connection.
Other Extensions

• We can apply a row-index-to-value autoencoder to any type of matrix to learn embeddings of either rows or columns.

• Applying to graph adjacency matrix leads to node embeddings.
  – Idea has been used by DeepWalk and node2vec after (indirectly) enhancing the matrix entries with random-walk methods.
  – Details of graph embedding methods in book.