Modifications of Foldiak's Algorithm for Imbalanced Data

Luke Bemish

Under the direction of

Kevin Chu
CEO
Velexi Corporation

Research Science Institute
July 27, 2020

Abstract

Creating sparse representations often helps to analyze data. Neurally inspired networks can provide methods for creating sparse representations from a dataset. Földiák's algorithm, a neurally inspired sparse coding method, is unable to extract accurate features from imbalanced datasets where features are over or under-represented. Here, we demonstrate a modification of Földiák's algorithm that allows it to find features in imbalanced datasets by allowing the bit probabilities of the extracted features to change over the course of training. On a imbalanced dataset created from known features and bit probabilities, the modified algorithm found the bit probabilities used to construct the data. The modified algorithm's learned memories and bit probabilities matched to the features and frequencies used to create the dataset, while those of Földiák's original algorithm did not.

Summary

Data can often be easier to analyze when it is expressed differently. A sparse representation of data is a way of expressing it as a combination of various memories. Sparse coding refers to the methods used to create these memories and find out which are present in a sample. Földiák's algorithm is a method for creating sparse representations. However, it does not perform well when some memories are over or under-represented in a dataset. Additionally, we have to know in advance how common all of the memories are going to be in order to find them. However, by allowing the frequency of the memories to change over training, Földiák's algorithm can be modified to adapt to the data it is shown and let the frequencies of the memories arise naturally from the data, which lets it perform better on imbalanced datasets.

1 Introduction

Machine learning seeks to decrease the dimensionality of input data into something that humans can understand and analyse. Categorization problems are generally solved by creating a sparse representation of input data, and then matching the dimensions of the representation with known categories. The methods used to create sparse representations are referred to as sparse coding [1]. Sparse coding involves creating a set of dictionary or basis vectors from a dataset, as well as calculating how much of each of them is present in each sample [2]. This allows a sparse representation to be created from the dataset, which could be used for categorization.

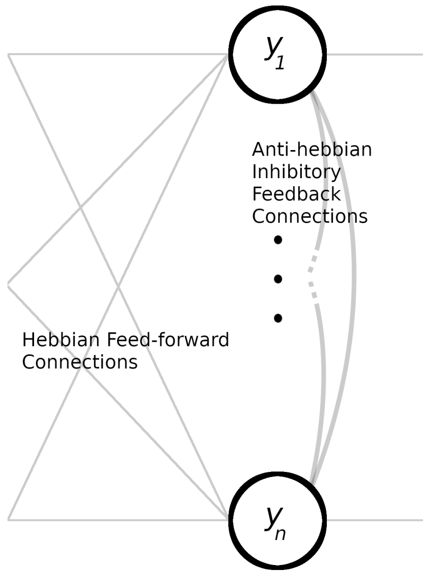
y_1
Anti-hebbian Inhibitory Feedback Connections
Hebbian Feed-forward Connections
y_n

Figure 1: A diagram of Földiák's algorithm. Hebbian feed-forward connections strengthen when both the input and output node are on, while anti-Hebbian feedback connections become increasingly negative to inhibit correlations when both endpoints are on.

Földiák's algorithm, developed by Peter Földiák, is a neural inspired method of sparse coding. Several output "nodes" are created, and a bit probability p, the chance of any given node being on for a given output, is assigned. Connections each have a bias or weight. Each node is, for a given input, either on or off. To determine the state of a node, the biases of incoming connections are multiplied with the values of incoming connections and summed; this sum is then compared to a threshold value. Positive feed-forward connections are created from the inputs of the network to the node, and negative feedback connections are created from the outputs to the node. The feed-forward connections follow associative or Hebbian learning, strengthening when both ends of the connections are on, while the feedback connections become increasingly negative when both outputs are on at the same time [3].

The feed-forward, or Hebbian connections allow the outputs to each "learn" a feature from the input. For any given output, the biases of that output's Hebbian connections make up an image in the same shape as the input, storing a "memory" of sorts of what feature that output is matching [3]. The feedback, or anti-Hebbian, connections prevent outputs from becoming correlated with each other, by decreasing the values of any two outputs when they are both on; this causes all of the outputs to learn distinct features from the data [3]. The combination of these two types of connections allows the algorithm to separate out important, distinct features or "memories" from a dataset, without needing to minimize an error of any sort.

Each output y_i has an internal threshold t_i that its inputs are compared to. Each iteration, the threshold is changed so as to bring the expected value of y_i closer to some value p, the bit probability of the network. If the algorithm is trained indefinitely, y_i will approach p [3]. This bit probability has a huge impact on the eventual state of the network, because it decides what value the threshold will eventually converge on. If the network is trained indefinitely, the expected value of y_i will eventually reach p. However, this means that p must be known before training begins. Many techniques of sparse coding overlook features that are over or under-represented in datasets [4]. In Földiák's algorithm, the frequency of the outputs in the original dataset is specified prior to training, meaning that Földiák's algorithm falls prey to this same problem.

Földiák's algorithm holds the potential to become an extremely powerful tool for creating sparse representations. However, the bit probabilities of the basis vectors of a dataset are not necessarily known prior to training, and they are not necessarily the same for all basis vectors. Optimally, an unguided algorithm for creating sparse representations would allow values of p to arise naturally from the data for each output.

Here we show how Földiák's algorithm can be modified in order to let the bit probabilities arise naturally from the data. The modified version of the algorithm performs well on datasets with known basis vectors that Földiák's algorithm is unable to find the basis vectors of due to the basis vectors having different or unknown bit probabilities. This is accomplished by letting the bit probabilities of each output change over time to match the data.

2 Methods

Rules

The modified network uses the same Hebbian and anti-Hebbian connections described by Földiák. Only the anti-Hebbian and threshold rules depend on p; here, each output is given its own value of p, with output y_i having a bit probability of p_i.

The thresholds are changed by the following rule, where γ is a learning rate constant [3]:

$$\Delta t_i = \gamma(y_i - p_i), \tag{1}$$

The Hebbian connections follow the following rule (q_{ij} is the bias of the connection from input j to output i) [3]:

$$\Delta q_{ij} = \beta y_i(x_j - q_{ij}). \tag{2}$$

the anti-Hebbian connections follow the following rule (w_{ij} is the bias between outputs i and j; w_{ij} is always kept negative, and $w_{ii} = 0$) [3]:

$$\Delta w_{ij} = -\alpha(y_i y_j - p_i p_j) \tag{3}$$

The previous rules were kept identical to Földiák's original algorithm. However, in order to allow p to arise naturally for each feature, p is changed over time so that it approaches the same value as $E(y_i)$, the expected value of output y_i. p_i is modified by the following rule:

$$\Delta p_i = \delta(y_i - p_i) \tag{4}$$

When shown a given sample, the feedback connections mean that the output values cannot be determined by a single sum. However, since the anti-Hebbian connections are symmetrical, it has been shown that a system like this, if left to stabilize, will reach a single value [5]. This value is found by numerically solving differential equations for each output [3].

A Dataset With Known Features

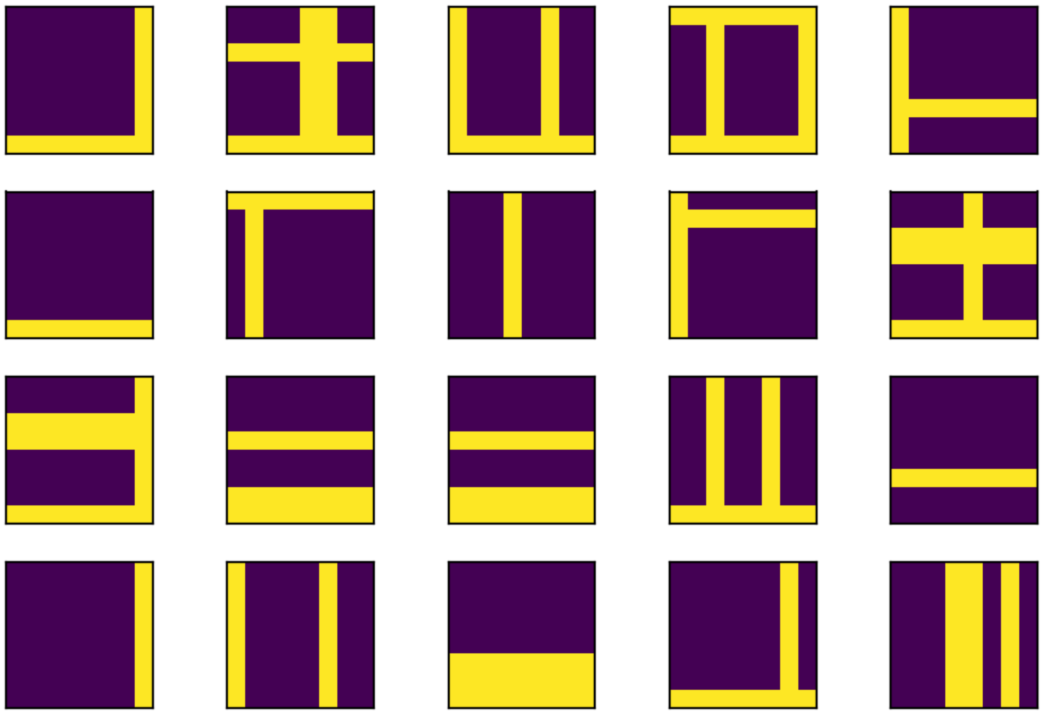

Figure 2: Examples of the data created to test the modified algorithm. In the data here, the lowermost horizontal line is 4 times as common as the other 15 lines; while Földiák's algorithm would struggle with this dataset due to this imbalance in the data, the modified algorithm lets its internal target probabilities change over time.

In order to test the modified network, a dataset was created with known bit probabilities and features so that error could be measured. Each sample was an 8 by 8 pixel image of horizontal and vertical lines, with each pixel as either a 1 or 0. Each of the 8 vertical lines had a $\frac{1}{8}$ chance of being on in any given image, while the first 7 of the horizontal lines had a $\frac{1}{8}$ chance of being on. The final horizontal line had a $\frac{1}{2}$ chance of being on.

Were the algorithm to perfectly learn the features and bit probabilities of this dataset, the output that learned the more common line would have $p = \frac{1}{2}$, while the others would have $p = \frac{1}{8}$.

This dataset was chosen as Földiák's algorithm has performed well on a balanced version of the dataset, with each line having a frequency of $\frac{1}{8}$ [3].

Error and Memories

Every 1000 samples, the stored memories or biases were compared with the known basis vectors or features. The memories were scaled to the range $(0, 1)$, and each known basis vector, or line, was matched to the scaled memory with the smallest difference. This allowed me to use the algorithm for categorization and to measure error.

The error we measured was the ratio of the number of incorrectly categorized basis vectors in a sample (either false positives or negatives) to the total number of basis vectors actually present in the sample.

The algorithm was run twice. Once, with the modified rules and parameters described above. The second time, it was run with $\delta = 0$ and $p_i = \frac{1}{8}$, which makes it identical to the algorithm run in Földiák's paper.

Training

To train the network, we used the parameters used by Földiák, with $\alpha = 0.1$ and $\beta = \gamma = 0.02$ [3]. we also used $\delta = 0.02$ for the p_i training rate. Prior to training, we ran the network with $\alpha = \beta = \delta = 0$ and $\gamma = 0.5$ with noise as an input for 200 iterations, which allows the thresholds to reach a stable value prior to training. The network was trained for 50,000 samples, with the error and p_i values recorded every 1000 samples.

3 Results

In Figure 3, the p_i values of the 16 outputs are shown over training.

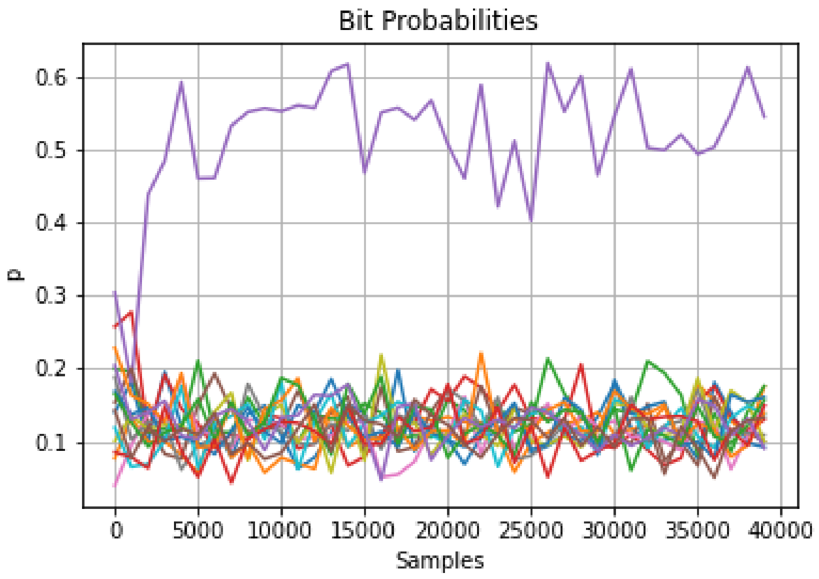

Figure 3: Values of p_i over time with $\delta = 0.02$; recorded every 1000 samples. 15 of the 16 outputs show a p_i that stays within 0.1 of 0.125, while the 16th shows a p_i that stays within 0.1 of 0.5. These p_i values match what would be expected of the dataset used, and demonstrate that the modified algorithm's bit probabilities can correctly adapt to the data used.

As training progresses, after a short initial training segment, the p_i values differentiate 1 output from the other 15, consistent with the basis vectors of the data set. Additionally, the

p_i value approached by the most common output appears to be close to 0.5, while the other 15 outputs have p_i values of close to 0.125. However, the p_i values vary substantially over training for any given node, even after longer training times when the network's memories have stabilized.

Figure 4 shows the error, measured as the ratio of false positive or negative lines to total lines present, over training for both Földiák's original algorithm, with a static p_i; and the modified algorithm, with a changing p_i.

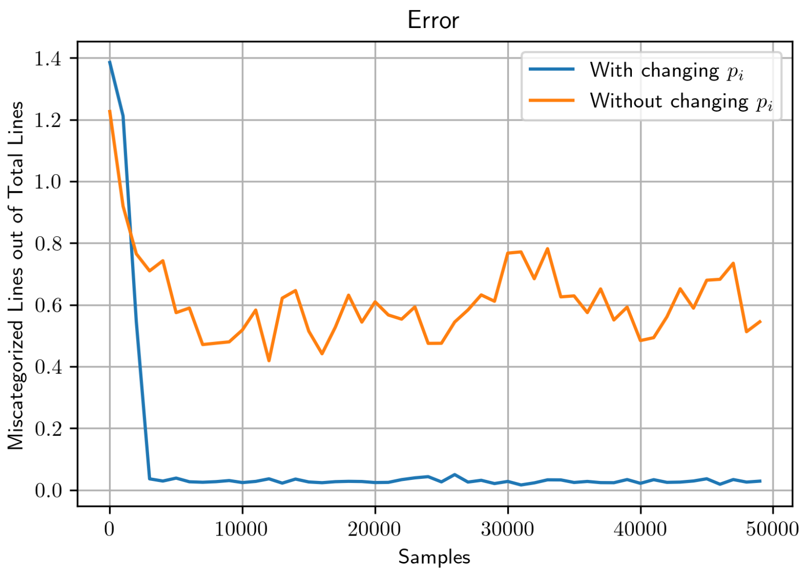

Figure 4: The modified version of Földiák's algorithm, with a changing p_i (shown in blue) performs substantially better than Földiák's original algorithm (shown in orange) on the dataset with known basis vectors and different bit probabilities.

Földiák's original algorithm, without a changing value of p_i, does not perform well on this dataset, though it correctly learns the memories of a dataset without where all 16 basis vectors had a bit probability of $\frac{1}{8}$ [3].

The modified version with a changing p_i, however, performs much better. Error, measured as the ratio of miscategorized lines to total lines, approached 0.0275 after training for 4000 samples, and stayed constant over extended training, while Földiák's original algorithm approached an error of close to 0.6 with large variances.

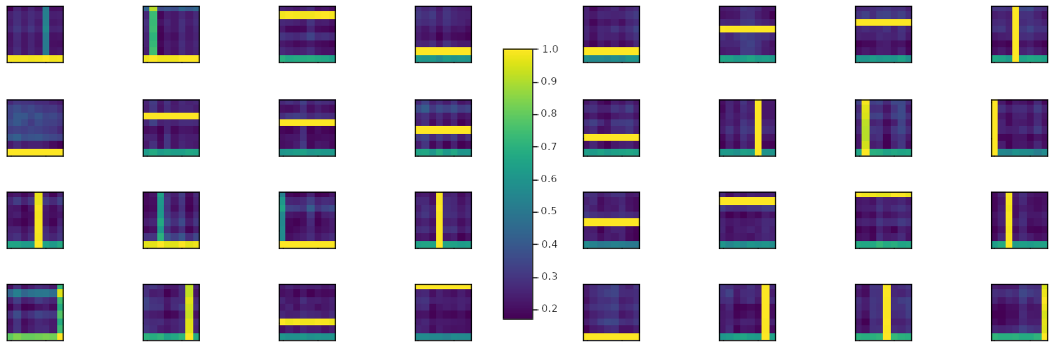

Figure 5: "Memories," or feed-forward bias vectors, stored by Földiák's algorithm (left) and the modified algorithm (right) after training. The modified algorithm better matches the original basis vectors than Földiák's original algorithm.

Figure 5 shows the stored bias vectors or memories after training of Földiák's algorithm and the modified version. In each memory, higher biases show the values that the memory matched best. In the modified algorithm (right), each memory has a strong line, with a bias of 1.0, representing one of the basis vectors of the dataset. Every basis vector is represented, and each output represents a different basis vector. The more common basis vector, the bottom horizontal line, is visible in every memory, but is always dimmer than the basis vector the memory best matches. In Földiák's algorithm (left), only 11 of the 16 basis vectors are represented by a memory, with 6 memories best matching the most common basis vector, the bottom horizontal line. Were the memories of the over-represented line to always give a false positive, they would give a false positive just less than $\frac{7}{16}$ of the time for each of the 5 features, leading to a predicted error of close to 0.9, which means that the actual false positive rate was less than $\frac{7}{16}$. This is likely due to the fact that these memories still contain bits of other features. While the outputs of the modified algorithm were each able to learn a distinct basis vector of the dataset, including the basis vector with a higher bit probability, the outputs of Földiák's original algorithm were not able to learn distinct basis vectors.

4 Discussion

On an imbalanced dataset, where Földiák's original algorithm struggles to create meaningful memories, a version modified to allow bit probabilities of individual nodes to vary over time performs significantly better. Földiák's algorithm is unable to form memories that match the features of the dataset, since it uses a single, static bit probability.

By allowing the bit probabilities to vary with the thresholds, the bit probabilities can arise naturally from the data instead of being specified at the beginning of training. The data used was imbalanced, since one of the features was substantially more common than the other 15. The algorithm correctly found bit probabilities of $\frac{1}{2}$ and $\frac{1}{8}$. However, the algorithm did not perform perfectly. The stored bit probabilities varied substantially as the model was continuously trained, and though it didn't seem to have a substantial effect on the error, it was likely responsible for some of the error present. Unfortunately, this instability cannot be removed by lowering δ, the bit probability training rate, due to the odd effects created by a difference between δ and γ.

Though more in depth evaluation is merited, the modified algorithm's p_i values and error appeared to only converge when δ, the training rate of p_i, was close to or equal to γ, the threshold training rate. If any difference was created between these two, even as small as 0.01, the p_i values seemed not to converge, and the algorithm exhibited a larger error. This could mean that a limit of some form is needed on one of the two rules, or that the state where the two rules converge can be expressed as a single equation.

While Földiák's algorithm could not learn the memories of the imbalanced dataset, the memories learned by the modified algorithm matched up to the known features used to create the dataset. These memories were not perfect, with noise caused by the other lines present at the same time as the line in the memory. Though the anti-Hebbian connections helped to prevent this, the over-represented feature showed up in all of the learned memories, though at a much fainter level than the main line in each memory. An assumption made was that the anti-Hebbian rule could remain unchanged without affecting the memories when the changing bit probabilities were added; this assumption may have been false.

The large error found in Földiák's algorithm is likely due to its inability to learn comprehensive memories. In some memories, such as the memory top second from the left, a line is faintly present that is missing from all other memories. However, since the algorithm looked for a constant bit probability of $\frac{1}{8}$, it appears that the over-represented feature got "spread out" over multiple memories to provide the necessary bit probabilities. This means that though the memories for some lines may trigger when the line is present, many will also trigger when the over-represented feature is present. Though the anti-Hebbian biases help to decorrelate similar memories, the degree to which they do this is dependent on p, and in Földiák's algorithm, p is constant and much less than the frequency of the over-represented feature.

The modified algorithm could be used in more extensive applications than the original algorithm. In applications where a dataset has no or very few keys, sparse coding could be used as prepossessing. The ability to not over-represent the over-represented features in the memories created makes the modified algorithm a possible option in these applications.

The tests done here were limited by the data used. The imbalances in this data took the form of an over-represented feature. However, in datasets or real world applications, a feature could instead be under-represented. This algorithm may or may not work to correctly find features in such a case. Additionally, comparison was only done to Földiák's original algorithm. The algorithm was not compared to other modern sparse coding techniques, many of which can be implemented as neural architectures [6].

5 Future Work

Though the initial results of the algorithm are promising, further analysis and comparison of the modified algorithm is needed. The algorithm could be compared to others used for sparse coding, such as neurally inspired algorithms like NOODL [6].

The bit probability and threshold values appear to be reasonably unstable over time. Though the instability could be decreased by decreasing the training rate $\delta = \gamma$, this comes at the cost of increased training time. A future improvement on the algorithm could be to implement a "certainty" value, to force the bit probabilities to change at a slower rate the more they match the data.

Additionally, further analysis is needed on why the algorithm performs best when the p_i and threshold learning rates are equal. This could be accomplished by combining the two into a single rule; in the algorithm as it currently stands, both rules accomplish the same purpose of making the expected value of the outputs y_i equal to p_i.

The algorithm should also be tested on datasets with a larger, possibly unknown set of features. Its performance could be measured by reconstruction error [1]. In the images of lines used here, there is only one set of features that reasonably represents the dataset. However, a dataset could be represented by more than one set of memories, and the ability for the bit probabilities to change increases the number of possible representations.

There are many possible applications for a tool to create accurate and useful sparse representations from imbalanced data. The algorithm could be applied to data to create sparse representation before the data is passed into a regression or classification algorithm, such as an artificial neural network.

6 Conclusion

Földiák's algorithm is limited by its inability to adapt to features without a constant or known bit probability. After being modified to allow the bit probabilities to change over time, the algorithm can find its bit probabilities from the data it is shown. This modified algorithm was tested on a dataset with known, imbalanced features, and its memories and error in predictions were compared to Földiák's original algorithm.

The modified algorithm performed substantially better than Földiák's original algorithm on the imbalanced data, and its memories learned the correct features while the original algorithm learned many copies of the over-represented memory. Over training, the modified algorithm found correct features quite often, while the original algorithm was unable to find the correct features. The bit probabilities learned by the modified algorithm, though unstable, were near the values used to create the dataset. The modified algorithm can create accurate sparse representations of imbalanced data, without over-representing the common features in its memories.

7 Acknowledgments

I would like to thank my mentor, Dr. Kevin Chu, my tutor, Ana Lyons, the Research Science Institute, Massachusetts Institute of Technology, the Center for Excellence in Education, and the sponsors of the Center for Excellence in Education. Thank you to RSI Alum Charles Tam for their feedback and comments on earlier versions of this manuscript.

References

[1] S. Arora, R. Ge, T. Ma, and A. Moitra. Simple, efficient, and neural algorithms for sparse coding. 2015.

[2] C. J. Rozell, D. H. Johnson, R. G. Baraniuk, and B. A. Olshausen. Sparse coding via thresholding and local competition in neural circuits. *Neural computation*, 20(10):2526–2563, 2008.

[3] P. Földiak. Forming sparse representations by local anti-hebbian learning. *Biological cybernetics*, 64(2):165–170, 1990.

[4] B. M. Whitaker. *Modifying sparse coding to model imbalanced datasets*. PhD thesis, Georgia Institute of Technology, 2018.

[5] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[6] S. Rambhatla, X. Li, and J. D. Haupt. NOODL: provable online dictionary learning and sparse coding. *CoRR*, abs/1902.11261, 2019.

A Source Code

For source code, see https://github.com/velexi-corporation/rsi-2020-bemish.