Ziptie: Learning Useful Features
November 20, 2023

Brandon Rohrer

A robot learning to navigate the world finds that the combination of certain sensors gives more information than either of them alone. The combination of x- *and* y-position tells more about when to watch for obstacles than either variable on its own. The speed *and* position of arm tells more about what shoulder torque needs to be generated than either variable on its own. These cases of sensor interaction can be hard coded manually. Human designers often exploit their knowledge of the system to do this via feature engineering, but in cases where the robot system is too complex to intuit these useful interactions, they can be learned. Automatically creating these predictive features is the goal of the **Ziptie** algorithm.

Ziptie makes a non-traditional assumption that all sensor signals, α_i, (otherwise known as **features**) are **Fuzzy Categorical variables** with $\alpha_i \in [0, 1]$. It also assumes that a fixed number of features, n, are received at discrete time intervals in a vector $\mathbf{A}$.

$$\mathbf{A} = (\alpha_1, \alpha_2, \alpha_3, ..., \alpha_n) \tag{1}$$

The n features of the sensor data can be imagined as n separate **cables**, ϕ_i, each carrying a single signal (as they often are in robots). The challenge of clustering these cables into informative combinations can then be imagined as creating **bundles** of cables, ϕ_{ij}, as with a ziptie.

A new bundle is created when the **agglomeration energy**, γ_{ij} between two cables exceeds a threshold, C_γ. Agglomeration energy is the accumulated **coactivation**, κ_{ij}, of the cable pair, where the coactivation at each time step is given by the product of their two activities.

$$\kappa_{ij} = \alpha_i \alpha_j \tag{2}$$

Once bundle ϕ_{ij} is created, it gets first dibs at representing any amount of signal carried on both cables ϕ_i and ϕ_j.

$$\alpha_{ij} = \min(\alpha_i, \alpha_j) \tag{3}$$

The member cables of ϕ_{ij} retain only the residual signal.

$$\alpha_{\hat{i}} = \alpha_i - \alpha_{ij} \tag{4}$$
$$\alpha_{\hat{j}} = \alpha_j - \alpha_{ij} \tag{5}$$

This approach constrains bundles' activities to remain on [0, 1] as well. Bundles can be coactive with cables and with other bundles. Any cable-cable, cable-bundle, or bundle-bundle pair whose agglomeration energy exceeds C_γ will nucleate a new bundle.

As Ziptie continues to operate on the stream of cable activities, the total number of bundles will continue to grow. As bundles are bundled again together, the number of cables in the largest bundles will grow too. These can be limited from growing too large by introducing a **stopping condition**, such as a maximum number of bundles or fixed number of time steps.

The rest of this paper attempts to answer the questions you might have:

- What's **all this** about?
- What is Ziptie **good for?**
- How does the **Ziptie** algorithm work in practice?
- What are **Fuzzy Categorical variables** and why do they matter?
- How is Ziptie related to what **biological brains** do?
- How can I use **Ziptie code**?

1. Concepts and Related Work

1.1. Feature Learning

Feature engineering is the practice of cleverly combining several features to get at information that none of them could provide on their own. For example, the x- and y-velocities of an object can be combined to give its overall speed, or specific patterns in a 3×3 collection of pixels can be used to detect edges. Feature learning, also known as **automated feature engineering**, is when features are generated through heuristics or the result of algorithms. There are a collection of **open source tools** for this, which largely focus on time series data sets.

Feature learning is also referred to as **representation learning**. **Principal components analysis** (PCA) is the poster child for unsupervised representation learning. PCA resembles Ziptie in that it finds combinations of features that tend to co-occur. PCA is focused on *dimensionality reduction* in that its goal is to reduce the total number of features used to a small set that distills out most of the informtion in the data.

1.2. Sparse Coding

Ziptie is an example of specific variant of feature learning called **sparse coding** or **sparse dictionary learning**, a family of methods for learning concise ways to represent data.

The goal of sparse coding is to represent an observation using as few features as possible. To do this, sparse coding methods often learn an overcomplete dictionary of basis functions. (Extending the analogy of a dictionary of the English language, the basis functions are the words. Overcomplete means that there is more than one combination of words that express the same idea. Overcompleteness allows for sparse representation, because you can choose the represent the idea concisely, using the representation with the fewest words.)

A culinary example of sparse coding is “Moose Tracks” flavor ice cream. It could just as accurately be described as “vanilla with small peanut butter cups and fudge ripples”. The name “Moose Tracks” is superfluous; it means exactly the same thing. Its existence shows overcompleteness in the dictionary of ice cream flavor names. But it allows for a concise representation.

Another example of sparse coding shows up in music. As seen in Figure 1, combinations of several keys (features) can be represented as chords (basis functions). There are many more possible chords than keys. Moving to a chord representation does not make a shorter dictionary than working with keys directly. But what it buys us is the ability to represent a collection of several keys with a single feature instead of having to enumerate all the keys involved.

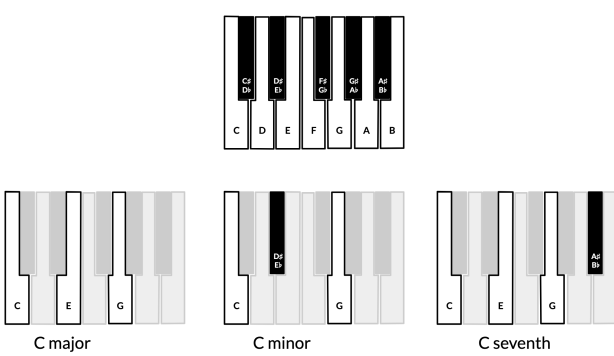

Figure 1. Piano chords as a sparse coding

1.3. ℓ^0 vs ℓ^1 Norms

In its purest form, the goal of sparseness is have as many features be zero as possible. An ideal sparse coding would be able to represent each set of inputs with exactly one feature. The number of non-zero features is also called the ℓ^0 norm. An ℓ^0 sparse coding will do it’s best to represent each input with as few features as possible.

Unfortunately, minimizing the ℓ^0 norm for feature representation is hard. Our regular machine learning bag of tricks for optimizing things requires differentiability, and the ℓ^0 norm is not differentiable. The count of non-zero features changes sharply by one, even if the feature value only changes from 0 to 0.001. And as that feature value changes from 0.001 to 0.1 to 10, the ℓ^0 norm doesn’t change at all.

As any experienced algorithms person will tell you, when presented with a problem you can’t solve, just ignore it and solve a different one. In sparse coding, if “sparse” is redefined to minimizing the ℓ^1 norm, instead of the ℓ^0 norm, then it becomes differentiable, and can be solved with familiar tools, including backpropagation and neural networks.

However, all hope is not lost. There is a whole field of optimization for hard problems like this. (If you’re trying to hire a person to do this their past job titles will probably be Operations Research, rather than Machine Learning.) A 2018 paper [1] described a method for ℓ^0 sparse coding using one of these optimization methods (mixed integer quadratic programming).

I applaud them for tackling the honest-to-goodness sparse coding problem. The only downside for practical use is that it is quite computationally expensive. The authors don’t say exactly how expensive, but when they describe it as an Achilles’ heel, that suggests it’s not quite ready for real time applications.

Another way to make sparse optimization easier to solve is to keep the ℓ^0 norm, but to relax the minimization requirement. What if we had a method that was just pretty OK? What if it didn’t necessarily find the way to use the absolute minimum number of features each time, but found a way to get kind of close? And what it it only took one-millionth of the computation? This describes the path of heuristic approximation, or to use less floofed-up language, a hack. The hack that Ziptie uses is agglomerative clustering.

[1] Liu, Y., Canu, S., Honeine, P., Ruan., S. (2018) K-SVD with a real L0 optimization: application to image denoising. Proc. 28th IEEE workshop on Machine Learning for Signal Processing (MLSP), Aalborg, Denmark. pp.1 - 6.

1.4. Agglomerative Clustering

This is a method of grouping observations or data points in which the most similar are grouped together right away, then the slightly less similar are added to those clusters. As clusters grow they can also glom on to each other. This process of similar observations and clusters repeatly combining to form larger clusters is **agglomerative clustering**. It's also called hierarchical clustering because when you trace the lineage observations and mini-clusters combining, it forms a tree showing the hierarchy of similarity between them.

1.5. Multi-membership Clustering

Ziptie is an unusual variant of agglomerative clustering where an item can belong to multiple clusters. Here the ziptie analogy of Figure 2 can be helpful. In a set of wires, imagine pulling out five of them and wrapping them with one ziptie. Then imagine taking just two of those five, selecting another two loose wires, and wrapping those four with another ziptie. Two of the wires are included in both zipties. Those represent elements with multiple cluster membership.

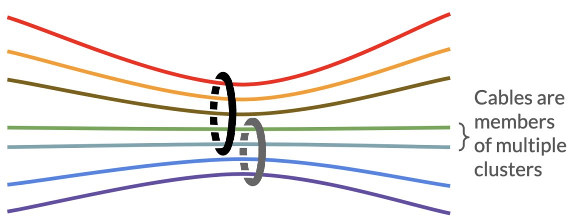

Figure 2. Clustering with multiple membership.

On its surface, scikit-learn's feature agglomeration [2] is similar to Ziptie. It uses agglomerative clustering to group features into larger groups of features. However, it is different in some important ways, which are illustrated in Section 3.1.

1.6. Continual Learning

Continual learning is a particular case of machine learning where the algorithm never stops evolving in response to its inputs, [3] also called incremental learning or lifelong learning. Ziptie is a specific flavor of continual learning called **online learning** where the algorithm does a small update after every new data point is collected. Ziptie also falls into the niche category of *unsupervised* continual learning like [4] and [5] because it isn't learning how to perform a specific task, but instead is learning how to organize and represent its data.

Taken all together, Ziptie sits at the intersection of several families of methods, as in Figure 3.

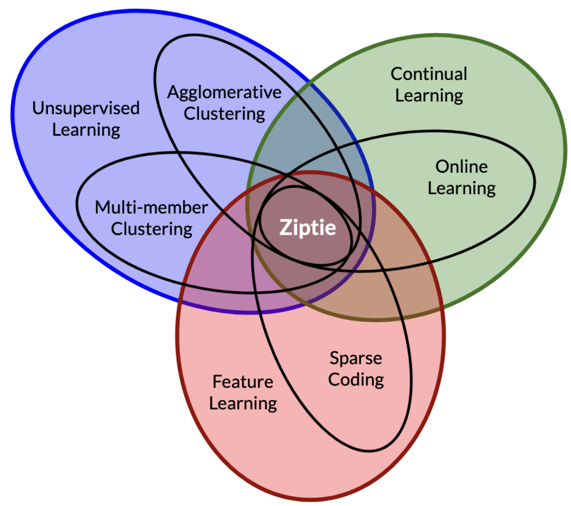

Figure 3. Ziptie at the crossroads.

- ℓ^0 Sparse Coding
- Agglomerative Clustering
- Multi-member Clustering
- Online Learning

2. Applications

It washes the windows and does your taxes.

3. How the Ziptie Algorithm Works

How it works, step-by-step.

[2]Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E. (2011) Scikit-learn: Machine Learning in Python. Feature agglomeration. *JMLR*, 12, 2825–2830.

[3]Wang, L., Zhang, X., Su, H., and Zhu, J. (2015) A Comprehensive Survey of Continual Learning: Theory, Method and Application. arXiv. https://arxiv.org/abs/2302.00487

[4]Ashfahani, A. and Pratama, M. (2021). Unsupervised Continual Learning in Streaming Environments. arXiv. https://arxiv.org/pdf/2109.09282.pdf

[5]Rao, D., Visin, F., Rusu, A. A., Teh, Y. W., Pascanu, R., and Hadsell, R. (2019) Continual Unsupervised Representation Learning. Paper presented at 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada. https://proceedings.neurips.cc/paper_files/paper/2019/file/861578d797aeb0634f77aff3f488cca2-Paper.pdf

3.1. Why coactivation matters

Feature agglomeration was mentioned in Section /refsubsec:featureagg. Feature agglomeration groups features based on how often they have similar values. It tries to group nearly identical features first. Ziptie on the other hand groups features based on how often they are co-active. (When they are both zero, that doesn't increase their similarity.) This creates groups of features that are simultaneously active, things that happen at the sime time. While this will capture features that are identical, it will also capture unrelated features whose co-occurence gives valuable information.

4. Fuzzy Categorical Variables

symbolic-connectionist divide

5. Biological Motivation

This is biologically motivated.

6. The Ziptie Python Package

Instructions for using the Ziptie library.

7. Versioning and History

The latest version of this document, and all the files needed to render it, are in this Codeberg repository. There's a backup copy in this repositofy on GitHub.

8. Licensing