EMBEDDED METHODS FOR FEATURE SELECTION IN NEURAL NETWORKS

A PREPRINT

Vinay Varma K.
Machine Learning Engineer
Niveshi
New Delhi, PIN 110034
vinay@niveshi.com

October 13, 2020

ABSTRACT

The representational capacity of modern neural network architectures has made them a default choice in various applications with high dimensional feature sets. But these high dimensional and potentially noisy features combined with the black box models like neural networks negatively affect the interpretability, generalizability, and the training time of these models. Here, I propose two integrated approaches for feature selection that can be incorporated directly into the parameter learning. One of them involves adding a drop-in layer and performing sequential weight pruning. The other is a sensitivity-based approach. I benchmarked both the methods against Permutation Feature Importance (PFI) - a general-purpose feature ranking method and a random baseline. The suggested approaches turn out to be viable methods for feature selection, consistently outperform the baselines on the tested datasets - MNIST, ISOLET, and HAR. We can add them to any existing model with only a few lines of code.

1 Introduction

1.1 Motivation

Outside of domains like Computer Vision and Natural Language Processing (NLP), identifying relevant input feature set for the task is not obvious. For example, consider the problem of forecasting stock-returns. Here, potentially relevant features might range from the returns of the index and other stocks to the tweets of the CEO. Even in applications where Domain-expertise renders a good feature set, the predictions from the model are rarely the end-goal.\footnote{Complete end-to-end decision-making systems aren’t yet possible with machine learning, at least in high stakes environments. For instance, consider using machine learning for decision making in healthcare\cite{1}. Though, In theory, diagnosing and treating a patient can be formulated under active reinforcement learning, present machine learning models only augment the expertise of the human doctor.} The work aims to present two simple methods that reliably give feature importance ranking in neural networks. It’s important to note that the method needs to be embedded - use the specific predictor in this case neural networks and be integrated into the parameter learning. This ensures appropriate interaction effects relevant to the optimization criteria are taken into account and the redundancies thereby are removed. Though the results are presented on feed-forward neural networks in a classification setting, by construction the same methods can be used with any learning criteria and architecture.

1.2 Background

For a given dataset $X^{n \times d}$, any Feature ranking method involves returning a list $I = [f_1, f_2, \ldots, f_d]$. The magnitude of the elements in the list is used to rank order the features. Here I detail the Permutation Feature Importance(PFI)\cite{1}
A preprint - October 13, 2020

algorithm which I here used as a baseline. PFI is a widely used importance attribution technique for random forests. Here, I adapted the model agnostic version\cite{2} of the method for neural networks.

\textbf{Algorithm 1:} Permutation Feature Importance

\textbf{Input:} training data $X \in \mathbb{R}^{n \times d}$, training labels $Y$, neural network $f_\theta()$, number of random permutations $c$, accuracy measure of the model $A_{f_\theta}(X,Y)$

\begin{algorithmic}[1]
\FOR{$\text{dim in 1,\ldots, d}$}
\STATE Let the feature importance of the $\text{dim}$ be $f_c$
\STATE $f_c \leftarrow 0$
\FOR{$i \text{ in 1,\ldots, c}$}
\STATE Generate a new input data $X_{\text{perm}}$ with random permutation of the column $\text{dim}$ in $X$
\STATE $f_c \leftarrow f_c + A_{f_\theta}(X,Y) - A_{f_\theta}(X_{\text{perm}},Y)$
\ENDFOR
\STATE The quantity $\frac{f_c}{c}$ gives the feature importance for each $\text{dim}$ in $d$
\ENDFOR
\end{algorithmic}

PFI has numerous shortcomings, especially under correlated-features\cite{3}. Nevertheless, the generality and the simplicity of the method makes it a good baseline.

The contributions of the present work are two-fold:

1. A drop-in layer with iterative pruning of the weights is shown to be a viable feature selection method.
2. Many prediction-explanation\cite{4,5,6,7} methods are used in neural networks. These are post-hoc instance-wise explainers. This work uses a gradient-based sensitivity method for global feature selection. I also propose that the testing framework used herein, where the features are removed and retrained should be adopted to approximate the relative validity of any feature attribution technique. As the approach doesn’t distinguish between instance-specific and global attribution methods, the results would allow us to compare between them and also use them for the task of feature selection.

\section{Proposed Methods}

\subsection{Sensitivity based Selection (SBS)}

Specific to neural networks several feature attribution methods exist. These approaches typically involve backpropagating an importance signal from the output neuron to the input features. For instance, Simonyan et al. used saliency maps\cite{7} for images that involve computing the gradient of the output w.r.t. pixels of an input image. To the best of my knowledge, they are designed as a Visualization-aid and are not used for feature selection. Sensitivity based selection method can be summarized as follows. Considering the classification problem with input $X \in \mathbb{R}^{n \times d}$. First, train the model on this data. Now, for each instance of the data point $x \in X$, compute the gradient of the output label w.r.t $x$ on this model. Take the absolute value of the average of this gradient across all the data instances. The resulting list of values $I = \{f_1, f_2, \ldots, f_d\}$ gives the global feature sensitivity of the model under consideration. Similarly, In a regression setting gradient can be taken w.r.t the error signal. The intuition is that the absolute value of the partial derivative of input feature w.r.t relevant output neuron, In a classification setting it the corresponding class-probability, quantify its importance. This method can be seen as adapting PFI for the special case of neural networks, using the gradient information. PFI, being model agnostic randomly permutes a feature and uses the drop in accuracy (or another similar metric) as a proxy for feature importance. Moreover, with PFI computational cost scales linearly with the dimensionality of the dataset. Neural Networks implemented with libraries like Pytorch\cite{8} allows the computation of this feature sensitivity in a single forward and backward pass through the network.

\subsection{Stepwise Weight Pruning Algorithm (SWPA)}

In Neural networks, weight-pruning refers to systematically removing the parameters of the existing network. For a given reduction in parameters, pruning techniques aim to minimize the loss in the performance of the original model\cite{2}. Here I use a drop-in layer that can be incorporated into any neural network and sequentially prune its weights to arrive at the most important ones. The idea is if these weights $W \in \mathbb{R}^{1 \times d}$ are in the first layer of the network and are all initialized to ones and the output of the first layer $O = \{w_1.x_1, \ldots, w_d.x_d\}$ is the multiplication of the corresponding

\footnote{Performance doesn’t necessarily always decrease with pruning. see\cite{10}}
The Inputs of the original model are first passed through the Drop-in Layer. Then, the weights $w_i$ are dropped sequentially during the training according to Algorithm 2.

Algorithm 2: Stepwise Weight Pruning Algorithm (SWPA)

Input: training data $X \in \mathbb{R}^{n \times d}$, training labels $Y$, base network $f_\theta(\cdot)$, Drop-in Layer $W$, Step Counter $n \in \mathbb{Z}_{\geq 1}$, Selection factor $f \in [0, 1]$

1. for count in 1, ..., $n + 1$ do
2.   $O \leftarrow \{w_1.x_1, \ldots, w_d.x_d\}$
3.   if count > 1 then
4.     $k \leftarrow (1-f) \times d$
5.     Sort the weights $W$ of the Drop-in Layer based on their absolute value.
6.     Set the least $k$ of them to 0.
7.   end
8.   Train the base network on $O$
9. end
10. Take the features corresponding to the top $f$ fraction of the weights in $W$ based on their absolute value and train them on the base network.

3 Experiments

3.1 Data and Setup

Here, I give a brief description of the datasets used.

Smartphone Dataset for Human Activity Recognition (HAR) is the data collected from smartphone sensors mounted on to the humans performing various activities like WALKING, LYING, STANDING, etc. For each data instance, 561 features are given.

ISOLET is a widely used speech dataset collected from 150 individuals speaking each letter of the alphabet. The data is preprocessed to include 617 attributes like spectral coefficients; contour and sonorant features.

MNIST is one of the most commonly used datasets in the machine learning community. It consists of 28-by-28 gray-scale images of handwritten digits from 0-to-9. All the images in the dataset are centered. Thus, each pixel can be safely treated as a separate feature.

In all the experiments data is randomly divided into 60-20-20 split between train, validation, and test set respectively. Training is done with a budget of 20000 epochs, and the patience factor is set to 2000 epochs. This essentially means the model is allowed to be trained for up to 20000 epochs only if there is no degradation of the performance on the validation set on any continuous set of 2000 epochs. The best performing model on the validation set is chosen and tested on the test set. A 3 layer feedforward neural network with a reduction factor of 2 is used in all experiments. Complete details of the network architecture are given in Appendix B. SGD optimizer is used with a learning rate of 0.001 and a
momentum of 0.9. Other hyperparameters include the Step Counter \( n \) of the SWPA which is set to 4, Selection Factor \( f \) to 0.1, and the number of random permutations \( c \) in PFA to 10.

3.2 Results

In Table 1, both the top and the bottom 10% of the features are selected based on the proposed methods and the corresponding accuracies on the test set are presented. Both the methods are able to effectively rank order the input features. Of the two, SWPA outperforms on all the data sets. Table 2 summarizes the results on the baseline - Permutation Feature Importance (PFI). As expected, PFI reduces the feature importance when a correlated feature is added \(^3\), and shares it between the two. This explains the small the accuracy gap between the top and the bottom as informative features are spread between both the groups \(^3\). But, it performs better than random assignment, acting as a reasonable baseline. For the accuracies on the randomly chosen features, 10 runs are used.

| Dataset | Top SWPA | Btm SWPA | Top SBS | Btm SBS |
|---------|----------|----------|---------|---------|
| HAR     | 0.976    | 0.654    | 0.932   | 0.574   |
| ISOLET  | 0.900    | 0.411    | 0.841   | 0.498   |
| MNIST   | 0.941    | 0.134    | 0.880   | 0.153   |

Table 1: Classification accuracies on the features chosen by the proposed methods

| Dataset | Top PFI | Btm PFI | Random Worst | Random Avg |
|---------|---------|---------|--------------|------------|
| HAR     | 0.917   | 0.814   | 0.306        | 0.504      |
| ISOLET  | 0.787   | 0.700   | 0.717        | 0.766      |
| MNIST   | 0.893   | 0.867   | 0.323        | 0.714      |

Table 2: Classification accuracies on PFI based features and the randomly assigned ones.

4 Ablations

4.1 Effect of Step Counter

In this section, I answer the question: Is the Step Counter \( n > 1 \) (SWPA) adding any value? I compared the accuracy of the resulting models with \( n = 1 \) and \( n = 4 \). It is interesting to note that Sequential pruning is adding a noticeable improvement in all the cases. In figure 2, I have chosen two instances of the MNIST dataset where SWPA \((n=1)\) misclassified them. About 35% of the features in \( SWPA(n=4) \) are different from \( SWPA(n=1) \). Given, the performance of the model on \( SWPA(n=4) \), it indicates that these features are responsible for the marginal gains.

| Dataset  | SWPA with \( n = 4 \) | SWPA with \( n = 1 \) |
|----------|-----------------------|-----------------------|
| HAR      | 0.976                 | 0.949                 |
| ISOLET   | 0.900                 | 0.835                 |
| MNIST    | 0.941                 | 0.877                 |

Table 3: SWPA with stepsize = 1

4.2 Effects of Adding constraints to the objective function

Here, I tested the effect of adding various constraints to the original objective function \( L(f_o, W) \). First, I added \( \ell^1 \) regularization on the weights \( W \) of the Drop-in Layer.

\(^3\)All the used datasets have high degree of correlated features.
The second and the third column highlights the final selected features by SWPA ($n = 1$) and SWPA ($n = 4$) respectively. In both the instances, SWPA ($n = 1$) misclassifies the input. Visually, it can be inferred that SWPA ($n = 4$) chooses slightly more informative features.

\[ H = \lambda \| W \|_1 \]  

(1)

\[ wvl = -\gamma \text{Var}[\text{Sigmoid}(20 \ast W - 0.5)] \]  

(2)

Further, to make the weights more spread out and hence distinctive, I also considered adding weight variance loss ($wvl$) as in (2). In all the experiments I chose $n = 1$ for simplicity. $\lambda, \gamma$ are set to 1 and 10 respectively.

| Dataset  | base  | only $l_1$ | only $wvf$ | $l_1$ and $wvf$ |
|----------|-------|------------|------------|-----------------|
| HAR      | 0.949 | 0.963      | 0.961      | 0.960           |
| ISOLET   | 0.835 | 0.833      | 0.816      | 0.828           |
| MNIST    | 0.877 | 0.879      | 0.877      | 0.867           |

Table 4: Effects of adding constraints - $l_1$ and Weight variance factor ($wvf$). Comparison of the features selected in each of these methods is given in Appendix A.

As there is no considerable difference in performance, I conclude that the marginal benefit of adding constraints like $l_1$ or $wvf$ is negligible for the task of selecting informative features.

5 Conclusion

As neural network based methods are becoming the default choice for modeling problems in high stakes applications like Finance and Medicine, maintaining the model and being able to reason about its predictions is crucial. Moreover, in real-world applications performance on the validation set or even the test set doesn’t give a complete picture as the statistical correlations can always break and new ones can emerge in the future. In this regard, a simpler model with fewer features is always desirable. In this work, I proposed two approaches for feature selection in neural networks, of which a simple pruning based method (SWPA) outperformed on all the datasets.

References

[1] Breiman, Leo. Random Forests. In Machine Learning, pages 5–32. Springer, 2001.

[2] Aaron Fisher, Cynthia Rudin, Francesca Dominici. All Models are Wrong, but Many are Useful: Learning a Variable’s Importance by Studying an Entire Class of Prediction Models Simultaneously. pages 1–81. JMLR, 2019.

[3] Molnar, Christoph. Interpretable machine learning. A Guide for Making Black Box Models Explainable. https://christophm.github.io/interpretable-ml-book/
[4] Leila Arras, Grégoire Montavon, Klaus-Robert Müller, Wojciech Samek. Explaining recurrent neural network predictions in sentiment analysis. In Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, pages 159–168, 2017.

[5] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In Doina Precup and Yee Whye Teh (eds.), Proceedings of the 34th International Conference on Machine Learning, volume 70 of Proceedings of Machine Learning Research, pages 3319—3328. PMLR, 2017.

[6] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In Doina Precup and Yee Whye Teh (eds.), Proceedings of the 34th International Conference on Machine Learning, volume 70 of Proceedings of Machine Learning Research, pages 3145–3153. PMLR, 2017.

[7] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In ICLR Workshop, 2014.

[8] Paszke, Adam and Gross, Sam and Massa, Francisco and Lerer, Adam and Bradbury, James and Chanan, Gregory and Killeen, Trevor and Lin, Zeming and Gimelshein, Natalia and Antiga, Luca and Desmaison, Alban and Kopf, Andreas and Yang, Edward and DeVito, Zachary and Raison, Martin and Tejani, Alykhan and Chilamkurthy, Sasank and Steiner, Benoit and Fang, Lu and Bai, Junjie and Chintala, Soumith. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems 32, pages 8024–8035. 2019

[9] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, John Guttag What is the State of Neural Network Pruning? In Proceedings of Machine Learning and Systems, 2020.

[10] Jonathan Frankle, Michael Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In ICLR, 2019.

[11] Chao Yu, Jiming Liu, Shamim Nemati. Reinforcement Learning in Healthcare: A Survey. arXiv:1908.08796, 2019.

[12] George Kour and Raid Saabne. Real-time segmentation of on-line handwritten arabic script. In Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on, pages 417–422. IEEE, 2014.

[13] George Kour and Raid Saabne. Fast classification of handwritten on-line arabic characters. In Soft Computing and Pattern Recognition (SoCPaR), 2014 6th International Conference of, pages 312–318. IEEE, 2014.

[14] Guy Hadash, Einat Kermany, Boaz Carmeli, Ofer Lavi, George Kour, and Alon Jacovi. Estimate and replace: A novel approach to integrating deep neural networks with existing applications. arXiv preprint arXiv:1804.09028, 2018.
A Feature Similarity on SWPA with various constraints

The numbers in the below tables indicate fraction of Features that are similar between the methods. Again, for all comparison the top 10% of the features are taken.

| Experiment | base | $l_1$ | $wvl$ | $l_1$ and $wvl$ |
|------------|------|-------|-------|-----------------|
| base       | 1.0  | 0.803 | 0.839 | 0.839           |
| $l_1$      | 0.803| 1.0   | 0.857 | 0.839           |
| $wvl$      | 0.839| 0.857 | 1.0   | 0.785           |

Table 5: HAR

| Experiment | base | $l_1$ | $wvl$ | $l_1$ and $wvl$ |
|------------|------|-------|-------|-----------------|
| base       | 1.0  | 0.786 | 0.852 | 0.786           |
| $l_1$      | 0.786| 1.0   | 0.786 | 0.852           |
| $wvl$      | 0.852| 0.786 | 1.0   | 0.786           |

Table 6: ISOLET

| Experiment | base | $l_1$ | $wvl$ | $l_1$ and $wvl$ |
|------------|------|-------|-------|-----------------|
| base       | 1.0  | 0.820 | 0.807 | 0.897           |
| $l_1$      | 0.820| 1.0   | 0.833 | 0.807           |
| $wvl$      | 0.807| 0.833 | 1.0   | 0.820           |

Table 7: MNIST

B Network

A 3 layer feed-forward neural network is used with a reduction factor of 2, meaning at each successive layer the dimensionality of the input is reduced by half. The final layer has the dimensionality equal to the number of distinct classes in the dataset. Table 8 and 9 summarizes the number of neurons in each layer for the networks on both reduced and full feature sets.

| Dataset | #feat | #layer 1 | #layer 2 | #layer 3 |
|---------|-------|----------|----------|----------|
| HAR     | 56    | 28       | 14       | 6        |
| ISOLET  | 61    | 30       | 15       | 26       |
| MNIST   | 78    | 39       | 19       | 10       |

Table 8: Network on reduced feature sets.

| Dataset | #feat | #layer 1 | #layer 2 | #layer 3 |
|---------|-------|----------|----------|----------|
| HAR     | 561   | 280      | 140      | 6        |
| ISOLET  | 617   | 308      | 154      | 26       |
| MNIST   | 784   | 392      | 196      | 10       |

Table 9: Network on full data set.