Not All Features Are Equal: Feature Leveling Deep Neural Networks for Better Interpretation

Yingjing Lu*
Carnegie Mellon University
yingjinl@andrew.cmu.edu

Runde Yang*
Cornell University
ry82@cornell.edu

Abstract

Self-explaining models are models that reveal decision making parameters in an interpretable manner so that the model reasoning process can be directly understood by human beings. General Linear Models (GLMs) are self-explaining because the model weights directly show how each feature contribute to the output value. However, deep neural networks (DNNs) are in general not self-explaining due to the non-linearity of the activation functions, complex architectures, obscure feature extraction and transformation process. In this work, we illustrate the fact that existing deep architectures are hard to interpret because each hidden layer carries a mix of high level features and low level features. As a solution, we propose a novel feature leveling architecture that isolates low level features from high level features on a per-layer basis to better utilize the GLM layer in the proposed architecture for interpretation. Experimental results show that our modified models are able to achieve competitive results comparing to main-stream architectures on standard datasets while being more self-explaining. Our implementations and configurations are publicly available for reproductions†.

1 Introduction

Deep Neural Networks (DNNs) are viewed as back-box models because of their obscure decision making process. As a result, those models are hard to verify and are susceptible to adversarial attacks. Thus it is important for researchers to find ways to interpret DNNs improve their applicability.

One reason that makes deep neural network hard to interpret is that they are able to magically extract abstract concepts through multi-layer non-linear activations and end-to-end training. From a human perspective, it is hard to understand how features are extracted from different hidden layers and what features are used for final decision making.

In response to the challenge of interpretability, two paths are taken to unbox neural networks' decision learning process. One method is to design verifying algorithms that can be applied to existing models to backtrace their decision learning process. Another method is to design models that "explain" the decision making process automatically. The second direction is promising in that it is more organic and robust.

One class of the self-explaining models borrows the interpretability of Generalized Linear Models (GLMs) such as linear regression. GLMs are naturally interpretable in that complicated interactions of non-linear activations are not involved. The contribution of each feature to the final decision output can simply be analyzed by examining the corresponding weight parameters. Therefore, we take a step to investigate ways to to make DNNs as similar to GLMs as possible for interpretability purpose while maintaining competitive performance.

* indicates equal contribution

† Public Repo URL annonymized for review purpose - Work submitted to NIPS 2019

Preprint. Under review.

Fortunately, a GLM model naturally exists in the last layer of most common architectures of DNNs (See Supplemental for why the last layer is a GLM layer). However, the GLM only limit to the output generated by the last layer and this ouput is not easy to interpret because it potentially contains mixed levels of features. In the following section, we use empirical results to demonstrate this mixture effect. Based on this observation, one way to naturally improve interpretation is to prevent low level features extracted from higher layers from mixing with those extracted by lower layers. Rather, we directly pass features extracted by each layer directly to the final GLM layer. This can further improve interpretability by leveraging GLM layer to account for these features. Motivated by this observation, we design a feature leveling network structure that can automatically separate low level features from their high level counterparts to prevent mixture effect. In other words, if the low level feature extracted by hidden k^{th} can be readily used by the GLM layer, we should directly pass it to the GLM rather than to continue feeding it to the $k+1^{th}$ hidden layer. We also propose a feature leveling scale to measure different features' complexity in an unambiguous manner rather than simply using vague terms such as "low" and "high".

In the following sections we will first lay out the proposed definition of feature leveling. This definition aims to replace concepts of relative "low" and "high" level features as they have long been treated as vague concepts. We then will illustrate how different levels of features reside in the same feature space. Based on the above observations, we propose feature leveling network, an architectural modification on existing models that can easily isolate low level features from high level features within different layers of the neural network. In the experiment section, we will use empirical results to show that this modification can also be applied to reduce the number of layers in an architecture and thus reduce the complexity of the network. In this paper, we focus primarily on fully connected neural networks(FCNN) with ReLU activation function in the hidden layers. Our main contributions are as follows:

- We take a step forward to quantize feature complexity for DNNs.
- We investigated the phenomenon of mixture between features of different complexities in the hidden layers of DNNs.
- We propose a feature leveling architecture that is able to isolate low level features extracted from each hidden layers to improve interpretation.
- We further show that the proposed architecture is able to prune redundant hidden layers to reduce DNNs' complexity with little compromise on performance.

The remaining content are organized as follows: In section 2 we first introduce our re-definitions for low and high level features and use a toy example to show the phenomenon of low and high level feature mixture in hidden layers. In section 3 we give a detailed account of our proposed feature leveling network that could effectively isolate low level and high level features. In section 4, we provide a high level introduction to some related works that motivated our architectural design. In Section 5, We test and analyze our proposed architecture on various real world datasets and prove that our architecture is able to achieve competitive performance while improving interpretability. In section 6, we show that our model is also able to automatically prune redundant hidden layers, thus reducing the complexity of DNNs.

2 Low level and high level features for neural networks

The concepts of low level and high level features are often brought up within the machine learning literature. However, their definitions are vague and not precise enough for applications. Intuitively, Low level features are usually "simple" concepts or patterns whereas high level features are "abstract" or "implicit" features.

Within the scope of this paper, we take a step forward to give a formal definition of different levels of features in a deep neural network. We will use a toy example to demonstrate how features can have different levels and explain why separating different levels of features could improve interpretability.

2.1 A toy example

We create a toy dataset called Independent XOR(IXOR). IXOR consists of a set of uniformally distributed features $\mathcal{X} : \{(x^1, x^2, x^3) | x^1 \in [-2, 2], x^2 \in [-2, 2], x^3 \in [0, 1]\}$ and a set of labels

$\mathcal{Y}:\{0,1\}$. The labels are assigned as:

$$\begin{cases} y=1 & x^1 \times x^2 > 0 \wedge x^3 > 0.5 \\ y=0 & otherwise \end{cases}$$

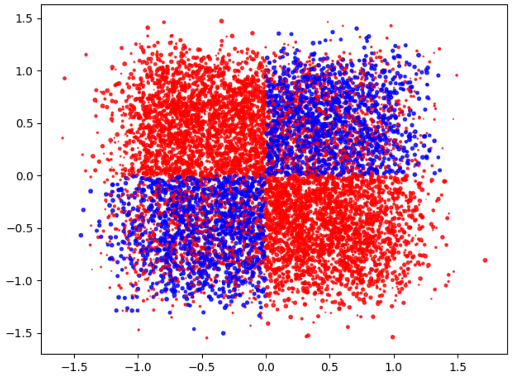

Figure 1: Visualization of the toy IXOR dataset

In this dataset (x^1, x^2, x^3) clearly have different levels of feature for DNNs. x^3 can be directly used by the GLM layer as it has a linear decision boundary. (x^1, x^2) is more complex as they form an XOR pattern and cannot be linearly separated, thus requiring further decomposition to become features ready for the GLM layer. To learn the patter, the DNN should use one layer to decompose the XOR decision boundary into lower level features, and maintain x^3's value to into the GLM layer.

2.2 Characterize low and high level features

From IXOR we can see that not all features have the same level of "complexity". Some could be directly fed into the GLM layer, others may need to go through one or more hidden layers to be transformed to features that can directly contribute to decision making.

And thus, instead of using "low level" and "high level" to characterize features, we propose to create a more precise concept of feature leveling to characterize how many levels of complexity certain feature has in the context of our proposed architecture - the feature leveling network.

For a dataset $\mathcal{D}$ consisting of N i.i.d samples with features and their corresponding labels $\{(x_1, y_1), ..., (x_N, y_N)\}$. We first assume that each sample x_i contains features that requires at most K hidden layers to extract.

We train a DNN with K hidden layers and a GLM layer. We define the set of k^{th} level feature as the set of features that requires at least $k-1$ hidden layers to extract to be sufficiently utilized by the GLM layer. In the following paragraphs, we denote $l_k \in L_k$ as the k^{th} level features extracted from one sample and L_k denotes the set of all k^{th} level feature to be learned in the target distribution. The rest of high level features are denoted by h_k that should be passed to the k^{th} layer to extract further level features. In this case, l_k and h_k should be disjoint, that is $l_k \bigcap h_k = \emptyset$. In the case of the toy example, x^3 is l_1, level one feature, as it is learned by the first hidden layer to directly transport its value to the GLM layer. (x^1, x^2) is h_1. The XOR can be decomposed by one hidden layer with sufficient number of parameters to be directly used by the GLM layer to make accurate decisions. Assuming the first hidden layer f_1 has sufficient parameters, it should take in h_1 and output l_2.

2.3 Mixture of low and high level features in DNNs

However, the basic form of neural network does not separate out each level of features explicitly. In our toy example, f_1 does not just take in (x^1, x^2), but x^1, x^2, x^3. Without explicit isolation, different levels of features could be mixed together and cause the network to be hard to interpret as demonstrated by a FCNN trained on the toy example in Figure 2.

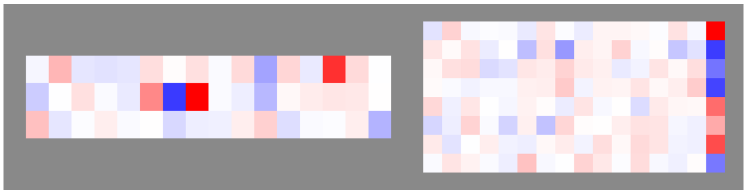

Figure 2: An example DNN architecture with 3-16-8-2 ReLU architecture. The weights of each layer are shown as heatmap where blue denotes negative weight, white as 0 and red as positive weights. Left is the weight heatmap for the first hidden layer and right is for the second. Feature x^3 in layer 1 is copied by the last column vector as the last input feature to the second layer. In the second hidden layer the feature x^3 is mixed within all the features extracted by layer 2 by inspecting the corresponding last weight column vector. Every output of layer 2 carries some form of x^3.

3 Our proposed architecture

Inspired by our definition of different levels of features and to resolve the mixture of feature problem, we design an architecture that is able to recursively filter the k^{th} level feature from the k^{th} layer inputs and allow them to be directly passed to the final GLM layer.

We start with a definition of neural network and extend that to our model: We aim to learn a function $\mathcal{F}$ parametrized by a neural network with K hidden layers. The function $\mathcal{F}$ can be written as:

$$\mathcal{F} = d\Big(f_K(f_{K-1}(...f_1(x;\theta_1));\theta_K)\Big) \tag{1}$$

with f_k being the k^{th} hidden layer function with parameters θ_k, $d(\cdot)$ be the GLM model used for either classification, or regression. Thus the goal is to learn the function $\mathcal{F}$ such that:

$$\mathcal{R}(\theta) = \frac{1}{N}\left(\sum_{i=1}^{N}\mathcal{L}(\mathcal{F}(x_i;\theta), y_i)\right) \qquad \theta^* = \arg\min_{\theta}(\mathcal{R}(\theta)) \tag{2}$$

In the above formulation, only f_K exports l_{K+1} whereas the output of other layers are mixed. Instead, in our formulation, each hidden layer can be viewed as separator for the k^{th} level feature and extractor for further level features. Thus the output of f_k has two parts: l_k is the set of k^{th} level feature being extracted from inputs and can be readily transported to the GLM layer for decision making. And h_k is the abstract features that require further transformations by f_k. In formal language, we can describe our network with the following equation ("$-$"denotes set subtraction):

$$\mathcal{F} = d\Big(l_1, l_2, ...l_K, f_K(f_{K-1}(...f_1(x - l_1;\theta_1)) - l_K)\Big) \tag{3}$$

In order for f_k to learn mutually exclusive separation, we propose a gating system for layer k, paramatrized by ϕ_k, that is responsible for determining whether a certain dimension of the input feature should be in l_k or h_k. For a layer with input dimension J, the gate $\{z_k^1, ...z_k^J\}$ forms the corresponding gate where $z_k^j \in \{0, 1\}$. ϕ_k is the parameter that learns the probability for the gate z_k^j to have value 1 for the input feature at j^{th} dimension to be allocated to h_k and l_k otherwise.

In order to maintain mutual exclusiveness between l_k and h_k, we aim to learn ϕ_k such that the it allows feature to pass to l_k if and only if the gate is exactly zero. Otherwise the gate is 1 and the feature goes to h_k. Thus we can rewrite the neural network $\mathcal{F}$ with the gating mechanism for the i^{th} sample x_i from the dataset:

$$\mathcal{F} = d\Big(B(z_1)\odot x_i, B(z_2)\odot f_1(z_1 \odot x_i), ..., f_K(z_K \odot f_{K-1}(z_{K-1} \odot f_{K-2}(...f_1(z_1 \odot x_i)))))\Big) \tag{4}$$

Here $\odot$ acts as element-wise multiplication. The function B acts as a binary activation function that returns 1 if and only if the value of z is 0 and 0 otherwise. The function B allows level k feature $l_k = B(z_k) \odot f_{k-1}$ to be filter out if and only if it does not pass into the next layer at all.

Then the optimization objective becomes:

$$\mathcal{R}(\theta, \phi) = \frac{1}{N}\left(\sum_{i=1}^{N}(\mathcal{L}(\mathcal{F}(x_i, z; \theta, \phi, B), y_i)\right) + \lambda \sum_{k=1}^{K} ||z_k||_0 \ , \quad z_k = g(\phi_k) \tag{5}$$

With an additional L_0 regularization term to encourage less h_k to pass into the next network but more l_k to flow directly to the GLM layer. $g(\phi)$ act as a transformation function that maps the parameter ϕ to the corresponding gate value.

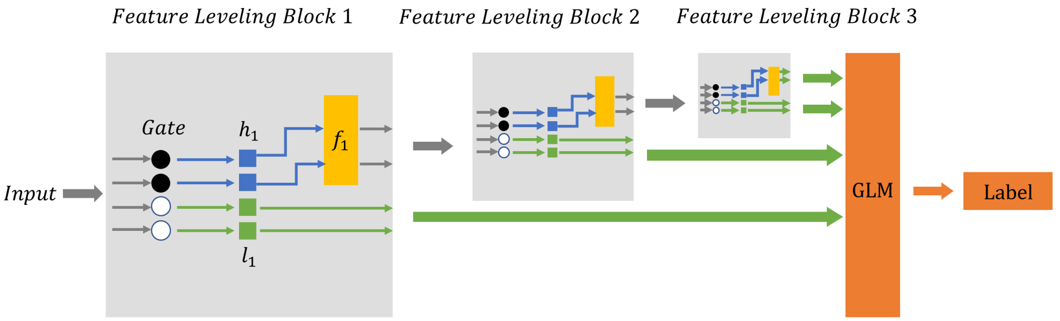

Figure 3: Illustration of the model with three hidden layers. Yellow denotes hidden layer typically with ReLU activations and greens denotes the k^{th} level feature separated out by the gates. Thick arrows denotes vector form of input and output. The dimension between the input of the hidden layers and the output can be different.

To achieve this discrete gate construction, we propose to learn the gating parameters under the context of L_0 regularization. To be able to update parameter values through backpropogation, we propose to use the approximation technique developed by [17] on differentiable L_0 regularization. We direct interested readers to the original work for full establishment of approximating L_0 and will summarize the key concept in terms of our gating mechanism below.

Although the gate value $z \in \{0, 1\}$ is discrete and the probability for a certain gate to be 0 or 1 is typically treated as a Bernoulli distribution, the probability space can be relaxed by the following: Consider s be a continuous random variable with distribution $q(s|\phi)$ paramaterized by ϕ. The gate could be obtained by with transformation function $m(\cdot)$ as:

$$s \sim q(s|\phi), \ z = m(s) = min(1, max(0, s)) \tag{6}$$

Then the underlying probability space is continuous because s is continuous while achieving exactly 0 gate value. The probability for the gate to be non-zero is calculated by the cumulative distribution function Q:

$$q(z \neq 0|\phi) = 1 - Q(s \leq 0|\phi) \tag{7}$$

The authors furthers use the reparameterization trick to create a sampling free noise $\epsilon \sim p(\epsilon)$ to obtain s: $s = n(\epsilon, \phi)$ with a differentiable transformation function $n(\cdot)$, and thus $g(\cdot)$ is equivalent to $m \circ n$ where $\circ$ denotes function composition.

Then the objective function under our feature leveling network is:

$$\mathcal{R}(\theta, \phi) = \frac{1}{N}\left(\sum_{i=1}^{N}(\mathcal{L}(\mathcal{F}(x_i, z; \theta, \phi, B, g), y_i)\right) + \frac{\lambda}{K}\sum_{k=1}^{K}\Big(1 - Q(s_k \leq 0|\phi)\Big)$$
$$z_k = g(\phi_k, \epsilon), \quad g(\phi_k, \epsilon) = m \circ n(\phi_k, \epsilon), \quad \epsilon \sim p(\epsilon) \tag{8}$$

4 Related work

Interpreting existing models: The ability to explain the reasoning process within a neural network is essential to validate the robustness of the model and to ensure that the network is secure against adversarial attacks [19, 4, 5]. In recent years, Many works has been done to try to explain the reasoning process of an existing neural network model either through extracting the decision boundary

[3, 24, 25, 29], or through a variety of visualization methods [18, 30, 15]. Most of those methods are designed for validation purpose. However, their results cannot be easily used to improve the original models.

Self explaining models are proposed by [2] and it refers to models whose reasoning process is easy for human beings to interpret. This class of models is advantageous in that it does not require a separate validation process. Many works have focused on designing self-explaining architectures that can be trained end-to-end[31, 27, 16, 12, 10]. However, most self-explaining models sacrifice certain amount of performance for interpretability. Two noticeable models among those models are able to achieve competitive performance on standard tasks while maintaining interpretability. The NIT framework [23] is able to interpret neural decision process by detecting feature interactions in a Generalized Additive Model style. The framework is able to achieve competitive performance but is only able to disentangle up to K groups of interactions and the value K needs to be searched manually during the training process. The SENN framework proposed by [2] focuses on abstract concept prototyping. It aggregates abstract concepts with a linear and interpretable model. Compared to our model, SENN requires an additional step to train an autoencoding network to prototype concepts and is not able to disentangle simple concepts from more abstract ones in a per-layer basis.

Sparse neural network training refers to various methods developed to reduce the number of parameters of a neural model. Many investigations have been done in using L_2 or L_1 [7, 20, 26, 6] regularization to prune neural network to avoid overfitting while maintaining differentiability for back propagation. Another natural choice for regularization and creating sparsity is the L_0 regularization. However, due to its discrete nature, it does not support parameter learning through backpropagation. A continuous approximation of L_0 is proposed in regard to resolve this problem and has shown effectiveness in pruning both FCNN and Convolutional Neural Networks (CNNs) in an end to end manner [17]. This regularization technique is further applied not only to neural architecture pruning but to feature selections [28]. Our work applies the L_0 regularization's feature selection ability in a novel context to select maximum amount of features as direct inputs for the GLM layer.

5 Experiments

We validate our proposed architecture through three commonly used datasets - MNIST, California Housing and CIFAR-10. For each task, we use the same initial architecture to compare our proposed model and FCNN baseline. However, due to the gating effect of our model, some of the neurons in the middle layers are effectively pruned. The architecture we report in this section for our proposed model is the pruned version after training with the gates. The second to last layer of our proposed models is labeled with a star to denote further concatenation with additional features. This is because the second to last dimension is the summary of dimensions of previous l_k and the dimension of the last hidden layer's output. For example, for California Housing architecture, both proposed and FCNN baseline started with $13 - 64 - 32 - 1$ as the initial architecture, but due to gating effect on higher layer, the layer with $32*$ neurons should have $32 + (13 - 10) + (64 - 28) = 36$ neurons accounting for previously gated features. ($13 - 10 = 3$ for l_1, $64 - 28 = 36$ for l_2).

The two objectives of our experiments are: 1) To test if our model is able to achieve competitive results, under the same initial architecture, compared to FCNN baseline and other recently proposed self-explaining models. This test is conducted by comparing model metrics such as root mean square error (RMSE) for regression tasks, classification accuracy for multi-class datasets and area under ROC curve (AUC) for binary classification. 2) To test if the k^{th} level features gated from the pre-GLM layer make similarly important contributions to the result as features extracted by entirely through hidden layers and fed to the GLM. In order to account for how much each layer's feature contribute to the final decision making, we propose to use the average of absolute values (AAV) of the final GLM layers weights on the features selected by the gates. If the AAV of each level's features is similar, it shows that these features make similar influence to the final decision as the features extracted by all hidden layers. It also demonstrates the fact that all levels of features could make equal contribution to the output of a neural network.

All models are implemented in TensorFlow[1] and hyperparameters configurations could be found in our public repository or supplemental code. Model name with citation denotes that the result is obtained from the original paper. SEEN's architecture listed is the prototyping network while we use similar architecture for autoencoder parts. All SENN models are re-implemented with fully

Table 1: MNIST classification and California Housing price prediction

MNIST			California Housing		
Model	Architecture	Accuracy	Model	Architecture	RMSE
FCNN	784-300-100-10	0.984	FCNN	13-64-32-1	0.529
L0-FCNN [17]	219-214-100-10	0.986	GAM [23]	-	0.506
SENN (FCNN)	784-300-100	0.963	NIT [23]	8-400-300-200-100-1	0.430
Proposed	291-300*-10	0.985	Proposed	10-28-32* -1	0.477

Table 2: CIFAR-10 Binary

Model	Architecture	AUC
FCNN	3072-2048-1024-2	0.855
GAM [23]	-	0.829
NIT [23]	3072-400-400-1	0.860
SENN (FCNN)	3072-2048-1024-2	0.856
Proposed	3072-130- 1024*-2	0.866

connected networks for comparison purposes. More detailed dataset preprocessing are available in supplemental.

5.1 Datasets & performances

The MNIST hand writing dataset [14] consists of pictures of hand written digits from 0 to 9 in 28×28 grey scale format. It contains 60000 training samples and 10000 testing samples. We use a $784 - 300 - 100 - 10$ architecture for both FCNN baseline and the proposed model. This is the same architecture used in the original implementations of [17]. Out model is able to achieve similar result as those state-of-the-art architectures using ReLU activated FCNNs while having less number of layers. The feature gates completely eliminated message passing to the 100 neuron layer which implies that our model only need level 1 and level 2 layers for feature extractions to learn the MNIST datasets effectively.

The California Housing dataset [21] is a regression task that contains various metrics, such as longitude and owners' age to predict the price of a house. It contains 8 features and one of the features is nominal. We converted the nominal feature into one-hot encoding, thus there are 13 features in total. Since California Housing dataset does not contain standard test set, we split the dataset randomly with 4:1 train-test ratio. Our proposed model could beats the FCNN baseline with the same initial architecture. We also witness that only 3 out of 13 original features are directly passed to the GLM layer, meaning that cal-housing's input features are mostly second and third level features.

The CIFAR-10 Dataset [13] consists of 32×32 RGB images of 10 different classes. We test our model's ability to extract abstract concepts. For comparison, we follow the experiments done in the NIT paper and choose the class cat and deer to perform binary classification. The resulting architecture shows that for FCNN networks, most of the the two chosen classes are mainly differentiated through their second level feature. Non of the raw inputs are used for direct classification. This corresponds to the assumptuion that RGB image of animals are relatively high level features.

5.2 k^{th} level feature passage and AAV of GLM weights

We also validate the percentage of input to the k^{th} hidden layer , which is the k^{th} level features selected by the gates. We also measures to what extent could these features contribute to the final decision made by the GLM layer (Figure 4). Through inspecting the percentage of features that flow to the GLM layer (the total number of gate with 1 as its value) and the AAV metric that we mentioned in the prior section, we notice that k^{th} level features generally have similar, if not higher, AAV weights compared to the features extracted through all hidden layers. This implies that the k^{th} level features are making similar contribution to the decision as those features extracted by FCNNs alone.

6 Strength in pruning redundant hidden layers

Due to our proposed model's ability to encourage linearity, our model is also able to reduce its network complexity automatically by decreasing the number of hidden layers. Empirically, as training goes on, each layer observes increasing number of features flowing to the GLM. Thus, more l_k features are transported directly to the GLM, reducing complexity of our model. Intuitively, the training process can be characterized by the following steps: 1) our model learns to use the hidden layers alone to learn the task (no l_k feature fed into the GLM layer), similar to traditional networks. 2) Gradually, our model is able to assign features to different levels strategically and effectively, creating more paths from each level of layer to the GLM. This implies that our network is learning to use more l_k features directly in GLM as opposed to transforming these features in further hidden layers.

We also observe that for some tasks such as MNIST classification, when the dataset feature level is less than the number of hidden layers, our proposed model can learn to prune excess hidden layers automatically. As the network learns not to pass information to further hidden layers. As a result, the number of hidden layers are effectively reduced. Therefore, We believe that our framework is helpful for architectural design by helping researchers to probe the ideal number of hidden layers to use as well as understanding the complexity of a given task.

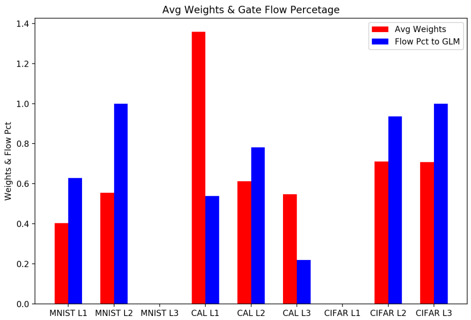

Figure 4: The percentage of gated features and average absolute weight (AAV) in GLM at different levels for all test models. Cal-Housing's AAVs are scaled down for graphing clarity.

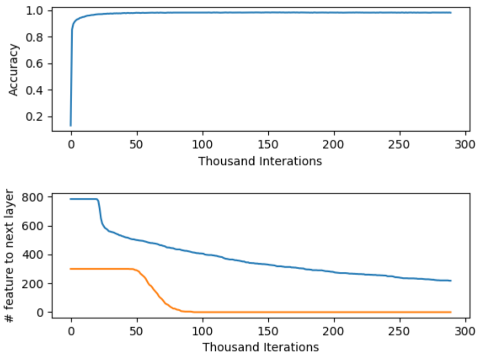

Figure 5: MNIST training performance curve and number of inputs passed to the following hidden layer (blue denotes the number of features passed to the firs hidden layer. Orange curve denotes the second).

7 Discussion

In this work we propose a novel architecture that could perform feature leveling automatically to boost interpretability. We use a toy example to demonstrate the fact that not all features are equal in complexity and most DNNs take mixed levels of features as input, decreasing interpretability. We then characterize absolute feature complexity by the number of layers it requires to be extracted to make GLM decision. To boost interpretability by isolating the k^{th} level feature from further mixed in following hidden layers, we proposed feature leveling network with a gating mechanics and a end-to-end training process that allow the k^{th} level feature to be directly passed to the GLM layer. We use experiment to show that our feature leveling network is able to successfully separate the k^{th} level feature without compromising performance.

There are two major directions that we see our proposed architecture will be developed: One is to extend our current construction in the context of convolutional neural networks. Another direction is to associate our network's identity mapping of lower level feature with existing network architectures with residual operations such as ResNet [9], Highway Network [22] and Dense Network [11] and try to explain their success. The residual architecture implicitly preserve the identity of the low level features and thus making them intact from further transformations of other features [8].

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] David Alvarez Melis and Tommi Jaakkola. Towards robust interpretability with self-explaining neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7775–7784. Curran Associates, Inc., 2018.

[3] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Advances in Neural Information Processing Systems*, pages 2494–2504, 2018.

[4] Tom B Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *arXiv preprint arXiv:1712.09665*, 2017.

[5] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2018.

[6] Federico Girosi, Michael Jones, and Tomaso Poggio. Regularization theory and neural networks architectures. *Neural computation*, 7(2):219–269, 1995.

[7] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[8] Moritz Hardt and Tengyu Ma. Identity matters in deep learning. *arXiv preprint arXiv:1611.04231*, 2016.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, volume 3, 2017.

[11] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[12] Hyunjik Kim and Andriy Mnih. Disentangling by factorising. *arXiv preprint arXiv:1802.05983*, 2018.

[13] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55, 2014.

[14] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2:18, 2010.

[15] Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. Visualizing and understanding neural models in nlp. *arXiv preprint arXiv:1506.01066*, 2015.

[16] Oscar Li, Hao Liu, Chaofan Chen, and Cynthia Rudin. Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[17] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017.

[18] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5188–5196, 2015.

[19] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[20] Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

[21] R Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297, 1997.

[22] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385, 2015.

[23] Michael Tsang, Hanpeng Liu, Sanjay Purushotham, Pavankumar Murali, and Yan Liu. Neural interaction transparency (nit): Disentangling learned interactions for improved interpretability. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 5804–5813. Curran Associates, Inc., 2018.

[24] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. *arXiv preprint arXiv:1804.02477*, 2018.

[25] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*, pages 6367–6377, 2018.

[26] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pages 2074–2082, 2016.

[27] Daniel E. Worrall, Stephan J. Garbin, Daniyar Turmukhambetov, and Gabriel J. Brostow. Interpretable transformations with encoder-decoder networks. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[28] Yutaro Yamada, Ofir Lindenbaum, Sahand Negahban, and Yuval Kluger. Deep supervised feature selection using stochastic gates. *arXiv preprint arXiv:1810.04247*, 2018.

[29] Radosiaw R Zakrzewski. Verification of a trained neural network accuracy. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, volume 3, pages 1657–1662. IEEE, 2001.

[30] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[31] Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. Interpretable convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8827–8836, 2018.