# Theory of Hierarchical Learning via Gradient Descent

Margalit Glasgow
Stanford University
mglasgow@stanford.edu

## Abstract

*In this project, we set the stage for theoretical study of hierarchical learning on neural networks via gradient descent. Our main goal is to design good synthetic toy settings for studying this non-trivial setting of learning. We define hierarchical learning, and consider various desired properties of a hierarchical learning problem which make it meaningfully hierarchical, likely to be amenable for analysis, and difficult enough that existing tools in the theory literature will not succeed. We then design some toy problems, and show empirically that they have these properties.*

# 1 Introduction

This project is the starting point of a larger theoretical investigation of how neural networks are able to learn functions of hierarchical features. The ultimate goal is to be able to approach this problem theoretically to understand how gradient descent on simple neural networks can learn hierarchical functions. For this class project, the primary progress is designing some toy synthetic vision-inspired classification settings in which:

1. The ground truth classifier involves a *hierarchy* of feature-learning tasks (as defined in Section 2).

2. Existing theoretical tools cannot explain why learning is possible (See Section 3).

In this project summary, we detail the process of designing such problems. We explain what aspects of these problems make them satisfy these criteria, and additionally what aspects make them simple enough to (hopefully) be amenable to future theoretical analysis.

Note that we do not *intentionally* design problems in which existing theoretical tools fail. Instead, we show that in natural synthetic and real-image settings where our other criteria apply, the existing theoretical tools are not sufficient for analysis.

## 1.1 Background

Empirically, it has been shown that neural networks can learn hierarchical features, and a prime example of this is CNNs on image data, visualized by work of Zeiler and Fergus [9]. Theoretically understanding the mechanism behind this is more complicated. Two recent works of Abbe and coauthors [1, 2] analyze a network that is trained by gradient descent (GD) consecutively layer by layer, which we call *layer-wise training* (LWT). These works argue that hierarchical functions are learned by the networks first learning lower level features, and then progressing to higher level features composed from the lower level features. In contrast to this theory, two recent works of Allen-Zhu and Li [3, 4] suggest that the low and high level features are learned simultaneously via gradient descent, with the higher network layer "correcting" the lower level layers during training. While the consecutive theory of learning is simpler, it may not be as powerful as the simultaneous learning. Nevertheless, the simultaneous learning setting of [3, 4] only considers a limited kernel-like regime where the trained network does not travel far from its initialization.

## 1.2 Notation

We use the notation $[k]$ to denote the set $\{1, 2, \dots k\}$.

# 2 Hierarchical Learning

While *hierarchical learning* is not a well-defined concept, it typically denotes the learning a composition of functions.

We will consider this following running example from vision as inspiration throughout this write-up. Consider a simplified setting where a truck can be identified by a circle (wheel) plus a rectangle, while a dog can be identified by two circles next to each-other (eyes), with a triangle below (nose). This classification problem of dogs vs trucks can be viewed as having a ground truth function which is a composition of two levels of functions. At the lower level, we have three relevant features $F_\circ$, $F_\square$, and $F_\triangle$ which identify circle, square, and triangle respectively. These features can be

thought of as being applied as convolutional features such that they represent a first level of functions. At the second higher level, we have have a function that takes as input the output of the lower-level convolutions, and returns a class, dog or truck. This second level function, consisting of a few "ands" and "or" operations is relatively simple.

We formalize this definition of a $K$-class hierarchical learning problem as follows. For $h \in 1, 2, \cdots, H$, let $\mathcal{F}_i$ be a set of functions from some domain $\mathcal{X}_{i-1} \to \mathcal{X}_i$. Let $\mathcal{X} = \mathcal{X}_0$ be the input space of the learning problem and let $\mathcal{X}_H = \{1, \cdots, k\}$.

**Definition 2.1.** *A $K$-class level $H$ hierarchical learning problem $\mathcal{F}$ is a sequence of function classes $\mathcal{F}_1, \ldots \mathcal{F}_H$. Each instance $f \in \mathcal{F}$ is a sequence of functions $\{f_i \in \mathcal{F}_i\}_{i \in [H]}$, which on input $x \in \mathcal{X}$ outputs $f_H(f_{H-1}(\cdots f_1(x)))$.*

In this project we will restrict our attention to problems with two levels of hierarchy, ie., $H = 2$.

**Example 2.2.** *For $j \in [J]$, let $c_j : \mathbb{R}^{d \times d} \to \mathbb{R}^{d \times d}$ be a convolutional filter that detects some feature $F_j$. (One can think of $F_j$ as being the set of all square filters of some dimension). For $(j_1, j_2, \cdots, j_{T_1}) \in [J]^{T_1}$, let $f_{(j_1, j_2, \cdots, j_t)} : \mathbb{R}^{d \times d} \to \mathbb{R}^{d \times d \times T_1}$ be such that $f_{(j_1, j_2, \cdots, j_{T_1})}(x) = (c_{j_1}(x), c_{j_2}(x), \ldots, c_{j_{T_1}}(x))$. Let $\mathcal{F}_1 = \{f_{(j_1, j_2, \cdots, j_t)}\}_{(j_1, j_2, \cdots, j_t) \in [J]^{T_1}}$.*

While many problems may have the hierarchical structure of Definition 2.1, we want to make sure the 2-level hierarchical toy problem we design involves meaningful learning at *both* levels (otherwise, it would secretly not be hierarchical). At the minimum this requires making sure both $\mathcal{F}_1$ and $\mathcal{F}_2$ are non-trivial function classes. It turns out, however, that this is not sufficient to require meaningful learning at both levels. To avoid such counterexamples, we state the following property, which we desire of our toy problems.

**Desired Property 1** (Learning Required at All Levels)**.** *If we only train any one layer of the neural network, we cannot achieve good train and test error.*

To reflect the interpretable low-level feature-learning that has been observed in [9], we want simple neural networks trained via gradient descent to identifyably learn the lower level features. In fact, we observe that in a simpler CNN trained on CIFAR-10, we can similarly interpret low-level features driven by the convolutional filters (see Example 5.1).

Interpretability is also an am important criteria for a problem being amenable to analysis: if we cannot interpret the trained neural network, it will likely be harder to analyze theoretically.

It turns out that this too is a non-trivial desiderata! In Table 1, we show that in seemingly hierarchical problems, the first level of the trained neural network does not significantly correlate with the individual underlying features.

Thus a second desired property of the toy problems we design involves interpretability:

**Desired Property 2** (Low-Level Feature Interpretability)**.** *Most first layer trained weights are correlated with a single underlying feature. This correlation is statistically significant compared to the correlations of the untrained weights with the underlying features.*

The meaning of "correlation with a single underlying feature" will become clear in Section 4, where we define our toy problems.

# 3 Existing Theoretical Tools in Hierarchical Learning

In this section, we describe a few settings in which previous theoretical work has been able to explain examples of hierarchical learning. One observation of this project is that both natural synthetic and real problems *cannot* be understood through these existing tools. This means that to push our theoretical understanding of neural network learning, we need to come up with new theoretical tools.

## 3.1 Kernel Learning

Kernel learning is equivalent to training a linear model on the representations obtained by applying a non-linear feature-map to the data points.

Kernel learning encompasses the neural tangent kernel (NTK) [5] and the random features model [6], which are particular regimes of neural network initialization and training where the network parameters do not deviate much from their initialization. In these cases, the final neural network can be viewed as a linearization around its initialization, thus inducing a kernel model.

Going beyond the NTK, but still staying close to the initialization of the network, one can study higher order Taylor approximations of a neural network. The work of Allen-Zhu and Li [3, 4] (which consider certain hierarchical functions) take place in such a regime.

The main weakness of kernel methods is that the feature-map is determined in advance (and sometimes randomly, by the initialization of the network), but cannot be *learned* by using the training data. This property makes kernel methods particularly limited on problems in sparsity is favored with respect to a certain unknown basis.[1] Explicit regularizing neural networks is known to sometimes alleviate this

[1] For instance, linear regression or ridge regression (which are kernel methods) require many more samples than $\ell_1$-regularized regression in settings where the ground truth classifier is sparse in the standard coordinate basis.

issue [8], and it is possible that the implicit regularization due to gradient descent may also have similar benefits. [7].

In this project, we design our toy problems to be sparse because of this particular weakness of kernel learning. While in this project we don't theoretically *prove* that kernel methods (or the higher-order Taylor expansion approaches of [3, 4]) cannot work, we empirically suggest that any trained neural network with good generalization travels far from its initialization. This precludes any of the kernel-like analyses above. For emphasis, we state this as a desired property of our toy problems.

**Desired Property 3** (Kernel Regimes Inapplicable)**.** *For any learning rate and batch size, with high probability over the initialization of the neural network and the training process, the following holds. If we train via SGD and achieve good train and test error, then the trained network $W_t$ travels far from its initialization $W_0$, that is,*

$$\frac{\|W_t - W_0\|_F}{\|W_0\|_F} \geq 1. \tag{3.1}$$

*Here $W$ represents the first layer weights of the network.*

While it is impossible to empirically prove that the problems we train on have this property, we expect it may hold because in all training regimes we tried, we observed it to be true (see Section 5).

### 3.2 Layerwise Staircase Learning

The works of Abbe et al. in [1, 2] consider a certain simplified setting of hierarchical learning where the ground truth satisfies a certain "staircase" property. The study a Boolean function setting where the data points $x$ lie $\{-1, 1\}^d$. In the simplest version, the function class they consider consists of functions $f_\pi : \{-1, 1\}^d \to \mathbb{R}$ parameterized by permutations on $[d]$, where $f_\pi(x) = x_{\pi(1)} + x_{\pi(1)}x_{\pi(2)} + x_{\pi(1)}x_{\pi(2)}x_{\pi(3)} + \cdots + x_{\pi(1)}x_{\pi(2)}x_{\pi(3)} \cdots x_{\pi(d)}$.

In both [1] and [2], the authors prove that this sort of staircase function class can be learned via *layer wise training* (LWT), where gradient descent is first performed on the first layer of the network, then on the second, and so on until the final layer is trained. In [1] they argue that this method of training works because first the first layer of the network learns the linear function $x_{\pi(1)}$, then, with this learned, the second layer can easily learn $x_{\pi(1)}x_{\pi(2)}$, and so on, up the layers.

While LWT provably works for staircase functions, one can ask whether their analysis techniques, which are premised on LWT, can succeed for more general hierarchical functions which closer reflect reality, or have the Properties 1 and 2. Based on our experiments, we observe that the hierarchical problems we design with Properties 1 and 2 cannot be easily learned via LWT. Similarly, we observe empirically that a CNN trained with LWT on CIFAR-10 does not learn well (See Example 5.2). We state this final property for emphasis.

**Property 4** (Layer-Wise-Training Fails)**.** *If we train layer by layer from the bottom up for a suitable number of epochs, we do not achieve good test error.*

## 4 Problems

To highlight the problem selection process, we list a number of toy problems we experimented with. Only some of them satisfy all of the desired properties, but we still believe our empirical observations on the others may be interesting and useful for future theoretical work.

To simplify our setting, we work with data points $x \in \mathbb{R}^d$, which can be thought of as a single row of "pixels".

For our first layer features/functions, we consider patterns of length $T$ in $\{-1, 1\}^T$. To keep the problem sparse, we consider $k$ such patterns, randomly chosen from $\{-1, 1\}^T$, where $k$ and $T$ are problem-dependent parameters. We call these patterns $F_1, F_2, \ldots, F_k$.

The second level functions are dependent on the presence, and possibly locations, of the $k$ different patterns. We elaborate in the specific problems. In most problems we consider only 2 or 4 classes, though in some we consider more classes.

Besides any specified features, all other pixels are set to 0. Note that we experimented with having small variance in the other pixels and observed similar results.

**Toy Problem 1.** $k = 2$, $T = 16$
*Class 1: $F_1$ in random location*
*Class 2: $F_2$ in random location*

**Toy Problem 2.** $k = 2$, $T = 16$
*Class 1: $F_1$ AND $F_2$ in random locations OR nothing.*
*Class 2: $F_1$ OR $F_2$ in random locations, but not both.*

**Toy Problem 3.** $k = 2$, $T = 16$
*Class 1: $F_1$ and $F_2$ in random locations*
*Class 2: $F_1$ OR $F_2$ in random locations, but not both.*
*Class 3: $F_1$ in two random locations.*
*Class 4: $F_2$ in two random locations both.*

**Toy Problem 4.** $k = 10$, $T = 16$
*Class 1: Exactly one occurrence of $F_1, F_2, \ldots$, or $F_{k/2}$ in random location.*
*Class 2: Exactly one occurrence of $F_{k/2+1}, F_{k/2+2} \ldots$, or $F_k$ in random location.*

**Toy Problem 5.** $k = 4$, $T = 16$
*Class 1: = $F_1$ AND $F_2$ in random location, sequentially with a spacing of $T$ OR $F_3$ AND $F_4$ in random location, sequentially with a spacing of $T$.*

*Class 2: = $F_1$ AND $F_3$ in random location, sequentially with a spacing of $T$ OR $F_2$ AND $F_4$ in random location, sequentially with a spacing of $T$.*

# 5 Experiments

We detail our experiments in this section. Most of our results are aggregated in Table 1.

**Architecture** In all problems described in Section 4, we train on a 3 layers neural networks with the following layers:

1. 1-D Convolutional filter of length $T$ with 20 channels, then relu.
2. Fully connected affine layer with 100 hidden units, then relu.
3. Fully connected affine layer with $C$ outputs, where $C$ is the number of classes.

Note, we also experimented with a 2 layer neural network. In some cases our observation were the same, but we have presented our results only for the 3-layer network.

**Training (Simultaneous and Layer-Wise)** We trained via SGD on $n = 1024$ examples with a batch size of 64. The number of epochs is given stated in the table of our results (Table 1); typically no more than 25 epochs were necessary in simultaneous training.

For the LWT, we trained for 200 epochs on each layer, which is (often far) more than 4 times the number of epochs needed to achieve accuracy of 0.1 with simultaneous training of layers.

We experimented with learning rates between 0.01 and 0.5, insuring that the best results were never achieved at the boundary of that set of learning rates.

**Deviation From initialization.** We report in Table 1 the value $\frac{\|W_t - W_0\|_F}{\|W_0\|_F}$, where $W_t$ represents the first (convolutional) layer after $t$ epochs of training. Here $t$ is the value in the first column, representing the number of epochs we trained until to get 0.1 test error. This corresponds to Property 3.

**Interpretability.** For each filter, we measure its dot product and its convolution with each features $F_1, \ldots F_k$. We find the max dot product among any feature, and the max value of the all-ones vector dotted with the convolution among any features. We compare the mean of these values for the trained convolutional filters to the mean for the untrained convolutional filters. We report two values in Table 1. The first value is the normalized difference between the trained and untrained max dot-products, and the second value is the normalized difference between the trained and untrained max convolutions. (Here we normalize by the untrained value).

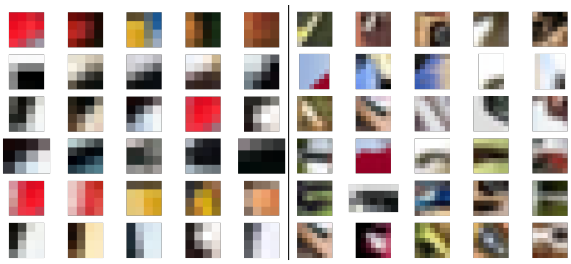

Figure 1: Visualization of Filters in Simple CNN. Left: Six Layer 1 filters. Right: Six Layer 2 filters. Each row represent 1 filter with 5 patches that maximally activate it.

## 5.1 Experiments with CIFAR-10 on Simple CNN

In the following two examples concerning a simple CNN trained on CIFAR-10, we use the following sequential architecture:

1. conv-relu (32 4x4 filters)
2. conv-relu-pool (32 4x4 filters)
3. affine-relu
4. affine - softmax

**Example 5.1** (Intepretability of Simple CNN on CIFAR-10)**.** *In Figure 1, we show the patches which most activated the first and second layer convolutional filters in the network above. The network was trained for 2 epochs on 49000 data points. Then, from the first minibatch of 64 images, we selected the 5 image patches which maximally activated each of the 32 filters in each convolutional layer. To save space, we only picture 6 filters from each layer.*

**Example 5.2** (LWT of Simple CNN on CIFAR-10)**.** *As a preliminary experiment, we tried LWT on the CIFAR-10 dataset, using the architecture above. For this architecture, we achieved an accuracy of over 60% on this architecture after a* single *epoch of training.*

*For the LWT, we train each of these four parts in sequence for one epoch, starting with the layers closest to the input. We observed significantly worse results both at train time and at test time for the LWT. Below are preliminary plots of training loss, training accuracy, and validation accuracy.*

*While the losses and accuracies seem to improve after the transition from training the first part to the second part,*

| Problem | Simultaneous Training | Layer-Wise Training | Train Single Layer | Dev from Init. | Interpretability |
|---|---|---|---|---|---|
| 1 | 4 | 20 | 137 (Second Layer) | 0.691 | $< 0$ |
| 2 | 50 | N/A | N/A | 2.004 | (0.084, 0.374) |
| 3 | 20 | 0.16 error after 200 | N/A | 1.934 | (-0.001, 2.699) |
| 4 | 44 | 0.29 error after 200 | N/A | 1.396 | ( -0.045, 4.135) |
| 5 | 46 | 70 | 351 (2nd layer) | 1.006 | (0.092, 1.793) |

Table 1: Summary of Results for Toy Problems, focusing on Properties 1-4. Number listed is epochs to get to 0.1 test error. N/A means it did not get to 0.1 after 200 epochs. If it got decent test error after 200 epochs, we list the error for completeness.

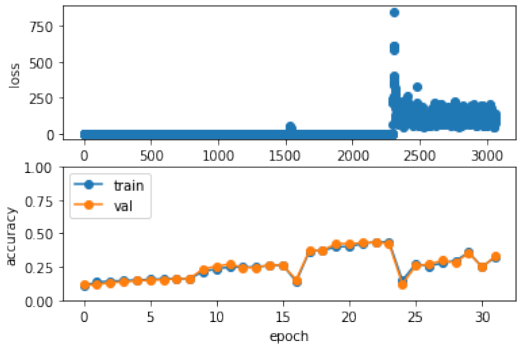


Figure 2: Training Dynamics for Layer-Wise-Training

*there is a big spike in loss (and drop in accuracies) after the third and forth parts begin to get trained. Note: we expect some of this issue is due to needing to choose new hyperparameters in each of the four periods of LWT. Since this was a preliminary experiment before the direction of my project changed, we did not focus on optimizing these hyperparameters.*

# 6 Conclusion

We experimented with a series of problems, and found that there are several (eg. problem 3 and 4) that seem to satisfy all the desired properties. Problem 1 was too simple, while Problem 2 was not interpretable, and problem 5 seemed interesting, but potentially could be understood via the kernel or LWT perspectives.

# References


[1] E. Abbe, E. Boix-Adsera, M. S. Brennan, G. Bresler, and D. Nagaraj. The staircase property: How hierarchical structure can guide deep learning. *Advances in Neural Information Processing Systems*, 34:26989–27002, 2021. 1, 3

[2] E. Abbe, E. Boix-Adsera, and T. Misiakiewicz. The merged-staircase property: a necessary and nearly sufficient condition for sgd learning of sparse functions on two-layer neural networks. *arXiv preprint arXiv:2202.08658*, 2022. 1, 3

[3] Z. Allen-Zhu and Y. Li. What can resnet learn efficiently, going beyond kernels? *Advances in Neural Information Processing Systems*, 32, 2019. 1, 2, 3

[4] Z. Allen-Zhu and Y. Li. Backward feature correction: How deep learning performs deep learning. *arXiv preprint arXiv:2001.04413*, 2020. 1, 2, 3

[5] A. Jacot, F. Gabriel, and C. Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018. 2

[6] A. Rahimi and B. Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007. 2

[7] N. Razin and N. Cohen. Implicit regularization in deep learning may not be explainable by norms. *Advances in neural information processing systems*, 33:21174–21187, 2020. 3

[8] C. Wei, J. D. Lee, Q. Liu, and T. Ma. Regularization matters: Generalization and optimization of neural nets vs their induced kernel. *Advances in Neural Information Processing Systems*, 32, 2019. 3

[9] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014. 1, 2