Understanding intermediate layers using linear classifier probes

Guillaume Alain & Yoshua Bengio[*]
Department of Computer Science and Operations Research
Université de Montréal
Montreal, QC. H3C 3J7
`guillaume.alain.umontreal@gmail.com`

Abstract

Neural network models have a reputation for being black boxes. We propose a new method to understand better the roles and dynamics of the intermediate layers. This has direct consequences on the design of such models and it enables the expert to be able to justify certain heuristics (such as the auxiliary heads in the Inception model). Our method uses linear classifiers, referred to as "probes", where a probe can only use the hidden units of a given intermediate layer as discriminating features. Moreover, these probes cannot affect the training phase of a model, and they are generally added after training. They allow the user to visualize the state of the model at multiple steps of training. We demonstrate how this can be used to develop a better intuition about a known model and to diagnose potential problems.

1 Introduction

The recent history of deep neural networks features an impressive number of new methods and technological improvements to allow the training of deeper and more powerful networks.

The model themselves had a reputation for being black boxes, and they still have that reputation. Neural networks are criticized for their lack of interpretability, which is a tradeoff that we accept because of their amazing performance on many tasks. Efforts have been made to identify the role played by each layer, but it can be hard to find a meaning to individual layers.

There are good arguments to support the claim that the first layers of a convolution network for image recognition contain filters that are relatively "general", in the sense that they would work great even if we switched to an entirely different dataset of images. The last layers are specific to the dataset being used, and has to be retrained when using a different dataset. In Yosinski *et al.* (2014) the authors try to pinpoint the at which this transition occurs, but they show that the exact transition is spread across multiple layers.

In this paper, we introduce the concept of the *linear classifier probe*, referred to as a "probe" for short when the context is clear. We start from the concept of *Shanon entropy*, which is the classic way to describe the information contents of a random variable. We then seek to apply that concept to understand the roles of the intermediate layers of a neural network, to measure how much information is gained at every layer (answer : technically, none). We show that it fails to apply, and so we propose an alternative framework to ask the same question again. This time around, we ask what would be the performance of an optimal linear classifier if it was trained on the inputs of a given layer from our model. We demonstrate how this powerful concept can be very useful to understand the dynamics involved in a deep neural network during training and after.

[*]Yoshua Bengio is a senior CIFAR Fellow

2 Information theory

In section 2.1 we do a quick review of the basic concept of *Shanon entropy*. In section 2.2 we then give Proposition 1 that demonstrates how *entropy* fails to capture the essence of a lot of interesting questions regarding neural networks.

- What happens when we add more layers ?
- Where does information flow in a neural network with multiple branches ?
- Does having multiple auxiliary losses help ? (e.g. Inception model)

This paves the way for our concept of the *linear classifier probe* that we introduce in section 3.1.

2.1 Basics of Shannon entropy

It was a great discovery when Claude Shannon repurposed the notion of *entropy* to represent information contents in a formal way. It laid the foundations for the discipline of information theory.

We would refer the reader to first chapters of MacKay (2003) for a good exposition on the matter.

One of the particularly interesting aspects of information theory is the ability to quantify in a useful manner the amount of uncertainty left in a discrete random variable X taking values in $\mathcal{A}_X$. The entropy is defined as

$$\mathbb{H}\left[X\right] = - \sum_{x \in \mathcal{A}_X} P(x) \log P(x) \tag{1}$$

and it is always non-negative. It is equal to zero if and only if X has all its probability mass at a single value. That is, the entropy is zero only when X is a deterministic value, when there is no uncertainty left in X.

Part of the genius of the notion of entropy is that is distills the essence of information to a quantity that does not depend on the particular representation.

One can also define the condition entropy of Y given X as

$$\mathbb{H}\left[Y|X\right] = - \sum_{x \in \mathcal{A}_X} P(x) \left[\sum_{y \in \mathcal{A}_Y} P(y|x) \log P(y|x) \right]. \tag{2}$$

Now, those formulas can be translated directly to the case of a continuous random variable X with probability density function $f_X(x)$, using integrals instead of summations, However, some of the nice properties no longer hold. In the continuous setting, the entropy is no longer scale-invariant, nor is it even guaranteed to be non-negative (we can find a good criticism of this in Marsh (2013)).

Despite this, it still makes sense to compare the conditional entropy $\mathbb{H}\left[Y|A\right]$ to the conditional entropy $\mathbb{H}\left[Y|B\right]$, for other random variables A and B. Moreover, the relative entropy, in the form of the KL-divergence, is also very useful when dealing with continuous variables.

2.2 Entropy with intermediate layers of neural network

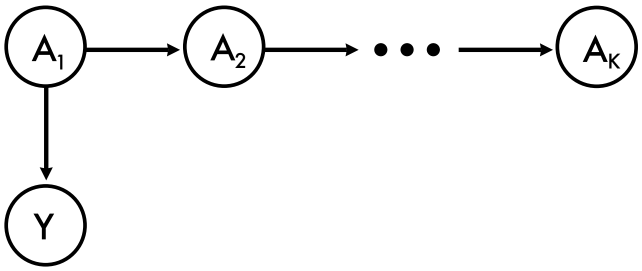
Question 1 **Given an MLP with data (X, Y), do any of the intermediate layers contain more information than X about Y ?**

To rephrase this question in terms of entropy, we are asking if the conditional entropy $\mathbb{H}\left[Y|A\right]$ is ever smaller than $\mathbb{H}\left[Y|X\right]$, where A refers to any intermediate layer of the MLP.

The answer to that is *No*.

We present the following proposition which can be applied directly to an MLP to settle this case.

Proposition 1. *Let $Y, A_1, A_2, \ldots, A_K$ be a set of continuous (or discrete) random variables such that their joint distribution factors according to the following graphical model.*



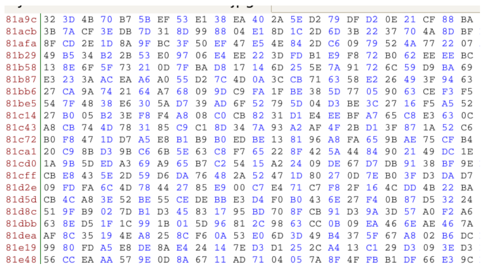
$$\begin{tikzcd}
A_1 \arrow[r] \arrow[d] & A_2 \arrow[r] & \cdots \arrow[r] & A_K \\
Y & & &
\end{tikzcd}$$

Then we have that the conditional entropy of Y given each A_k is ordered as follow :

$$\mathbb{H}\left[Y|A_1\right] \leq \mathbb{H}\left[Y|A_2\right] \leq \ldots \leq \mathbb{H}\left[Y|A_K\right] \tag{3}$$

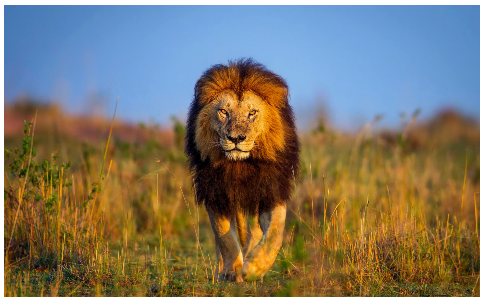
This proposition can be applied directly to the structure of a traditional MLP. It applies also to a MLP where certain layers have a component of randomness (e.g. Dropout during training).

Going deeper into the multi-layer neural network, we are pretty much only destroying information. We start with the raw data X, and then it's just downhill. If X contains the image of the savannah, and $Y \in \{0, 1\}$ refers to whether it contains a lion or not, then none of the subsequent layers are truly more informative than X itself. Refer to figure 1.



```
81a9c 32 3D 4B 70 B7 5B EF 53 E1 38 EA 40 2A 5E D2 79 DF D2 0E 21 CF 88 BA
81acb 3B 7A CF 3E DB 7D 31 8D 99 88 04 E1 8D 1C 2D 6D 3B 22 37 70 4A 8D BF
81afa 8F CD 2E 1D 8A 9F BC 3F 50 EF 47 E5 4E 84 2D C6 09 79 52 4A 77 22 07
81b29 49 B5 34 B2 2B 53 E0 97 06 E4 EE 22 3D FD B1 E9 F8 72 B0 62 EE EE BC
81b58 13 8E 6F 5F 73 21 0D 7F BA D8 17 14 6D 25 5E 7A 91 72 6C 59 D9 BA 69
81b87 E3 23 3A AC EA A6 A0 55 D2 7C 4D 0A 3C CB 71 63 58 E2 26 49 3F 94 63
81bb6 27 CA 9A 74 21 64 A7 68 09 9D C9 FA 1F BE 38 5D 77 05 90 63 CE F3 F5
81be5 54 7F 48 38 E6 30 5A D7 39 AD 6F 52 79 5D 04 D3 BE 3C 27 16 F5 A5 52
81c14 27 B0 05 B2 3E F8 F4 A8 08 C0 CB 82 31 D1 E4 EE BF A7 65 C8 E3 63 0C
81c43 A8 CB 74 4D 78 31 85 C9 C1 8D 34 7A 93 A2 AF 4F 2B D1 3F 87 1A 52 C6
81c72 B0 F8 47 1D D7 A5 E8 B1 B9 B0 ED BE 13 81 96 A8 FA 65 9B AE 75 CF B4
81ca1 20 C9 8B D3 9B C6 6B 5E 63 C8 F7 65 22 8F 42 5A 44 84 90 21 49 DC 1E
81cd0 1A 9B 5D ED A3 69 A9 65 B7 C2 54 15 A2 24 09 DE 67 D7 DB 91 38 BF 9E
81cff CB E8 43 5E 2D 59 D6 DA 76 48 2A 52 47 1D 80 27 0D 7E B0 3F D3 DA D7
81d2e 09 FD FA 6C 4D 78 44 27 85 E9 00 C7 E4 71 C7 F8 2F 16 4C DD 4B 22 BA
81d5d CB 4C A8 3E 52 BE 55 CE DE BB E3 D4 F0 B0 43 6E 27 F4 0B 87 D5 32 24
81d8c 51 9F B9 02 7D B1 D3 45 83 17 95 BD 70 8F CB 91 D3 9A 3D 57 A0 F2 A6
81dbb 63 8E D5 1F 1C 99 1B 01 5D 96 81 2C 98 63 CC 0B 09 EA 46 6E AE 46 7A
81dea AF 8C 35 19 4E A8 25 8C F6 0A 53 E0 6D 3D 49 B4 37 5F 67 A8 02 B6 DC
81e19 99 80 FD A5 E8 DE 8A E4 24 14 7E D3 D1 25 2C A4 13 C1 29 D3 09 3E D3
81e48 56 CC EA AA 57 9E 0D 8A 67 11 AD 71 04 05 7A 8F 4F FB B1 DF 66 E3 9C
```

(a) hex dump of picture of a lion



(b) same lion in human-readable format

Figure 1: The hex dump represented on the left has more information contents than the image on the right. Only one of them can be processed by the human brain in time to save their lives. Computational convenience matters. Not just entropy.

3 Linear classifier probes

In section 3.1 we present the main concept from this paper. We illustrate the concept in section 3.2 We then present a basic experiment in section 3.3. In section 3.4 we modify a very deep network in two different ways and we show how probes allow us to visualize the (sometimes disastrous) consequences of our design choices.

3.1 Probes

Despite what we highlighted in the previous section 2, there is indeed a good reason to use many deterministic layers, and it is because they perform useful transformations to the data with the goal of *ultimately fitting a linear classifier at the very end*. That is the purpose of the many layers. They are a tool to massage data into a form to be fed to a boring linear classifier.

With this in mind, it is natural to ask if that transformation is sudden or progressive, and whether the intermediate layers already have a representation that is immediately useful to a linear classifier. We refer the reader to figure 2 for a little diagram of probes being inserted in the usual deep neural network.

The conceptual framework that we propose is one where the intuitive notion of *information* is equivalent with *immediate suitability for a linear classifier* (instead of being related to entropy). It is with that notion that we study multiple scenarios in sections 3.2, 3.3, 3.4.

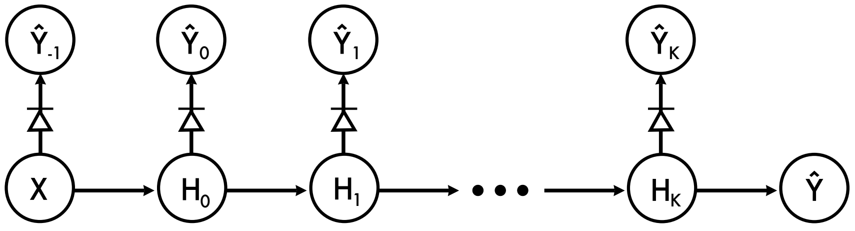


Figure 2: Probes being added to every layer of a model. These additional probes are not supposed to change the training of the model, so we add a little diode symbol through the arrows to indicate that the gradients will not backpropagate through those connections.

Just to be absolutely clear about what we call a *linear classifier*, we mean a function

$$\begin{aligned} f\colon H &\to [0,1]^d \\ h &\mapsto \text{softmax}\left(Wh+b\right). \end{aligned}$$

where $h \in H$ are the features of some hidden layer, $[0, 1]^d$ is the space of one-hot encodings of the target classes, and (W, b) are the probe weights and biases to be learned so as to minimize the usual cross-entropy loss.

Over the course of training a model, the parameters of the model change. This is what is commonly understood when we refer to "learning". However, probes only make sense when we refer to a given training step. We can talk about the probes at iteration n of training, when the model parameters are θ_n. **These parameters are not affected by the probes.** We prevent backpropagation through the model either by stopping the gradient flow (done with `tf.stop_gradient` in tensorflow), or simply by specifying that the only variables to be updated are the probe parameters, while we keep θ_n frozen.

We train the probe parameters up to convergence (or with early stopping). The concept of the probe makes sense only when we assume that it is the optimal possible linear classifier (or very close to that in practice).

Note that training those probes represents a convex optimization problem. In practice, this does mean that it is an easy thing to accomplish, but it is reassuring because it means that probes taken at time θ_n could be used as initialization for probes at time θ_{n+1} if we were not concerned about overfitting.

3.2 Probes on untrained model

We start with a toy example to illustrate what kind of plots we expect from probes. We use a 32-layer MLP with 128 hidden units. All the layers are fully-connected and we use LeakyReLU(0.5) as activation function.

We will run the same experiment 100 times, with a different toy dataset each time. The goal is to use a data distribution (X, Y) where $X \in \mathbb{R}^{128}$ is drawn $\mathcal{N}(0, I)$ and where $Y \in \{-1, 1\}$ in linearly separable (i.e. super easy to classify with a one-layer neural network). To do this, we just pick a $w \in \mathbb{R}^{128}$ for each experiment, and let the label y_n be the sign of $x_n^T w$.

We initialize this 32-layer MLP using *glorot_normal* initialization, we do not perform any training on the model, and we add one probe at every layer. We optimize the probes with RMSProp and a sufficiently small learning rate.

In figure 3, we show the prediction error rate for every probe, averaged over the 100 experiments. The graph includes a probe directly on the inputs X, where we naturally have an error rate that is essentially zero (to be expected by the way we constructed our data), and which serves as a kind of sanity check. Given that we have only two possible labels, we also show a dotted horizontal line at 0.50, which is essentially the prediction error that we would get by flipping a coin. We can see that the prediction error rate climbs up towards 0.50 as we go deeper in the MLP (with untrained parameters).

This illustrates the idea that the input signal is getting mangled by the successive layers, so much so that it becomes rather useless by the time we reach the final layer. We checked the mean activation

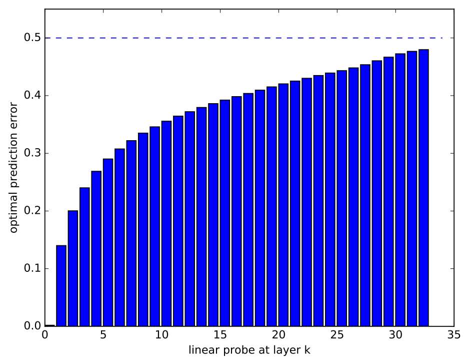


Figure 3: Toy experiment described in section 3.2, with linearly separable data (two labels), an untrained MLP with 32 layers, and probes at every layer. We report the prediction error for every probe, where 0.50 would be the performence of a coin flip and 0.00 would be ideal. Note that the layer 0 here corresponds to the raw data, and the probes are indeed able to classify it perfectly. As expected, performance degrades when applying random transformations. If many more layers were present, it would be hard to imagine how the final layer (with the model loss) can get any useful signal to backpropagate.

norm of the hidden units at layer 32 to be sure that numerical underflow was not the cause for the degradation.

One of the popular explanation for training difficulties in very deep models is that of the exploding/vanishing (Hochreiter, 1991; Bengio *et al.*, 1993). Here we would like to offer another complementary explanation, based on the observations from figure 3. That is, at the beginning on training, the usefulness of layers decays as we go deeper, reaching the point where the deeper layers are utterly useless. The values contained in the last layer are then used in the final softmax classifier, and the loss backpropagates the values of the derivatives. Since that derivative is based on garbage activations, the backpropagated quantities are also garbage, which means that the weights are all going to be updated based on garbage. The weights stay bad, and we fail to train the model. The authors like to refer to that phenomenon as *garbage forwardprop, garbage backprop*, in reference to the popular concept of *garbage in, garbage out* in computer science.

The works of Pascanu *et al.* (2013) and Montufar *et al.* (2014) explore the possibility of counting the exponentially-many regions of space that are defined define using the composition of many sequential layers in a deep neural network. In a way, the ability of an untrained neural network to "fold" a nice manifold into something unusable should not come as a surprize. Just a few randomly-initialized layers are sufficient to ruin a linearly-separable dataset. We leave out the idea of adversarially-tuned layers for another day.

3.3 Probes on MNIST convnet

In this section we run the MNIST convolutional model provided by the `tensorflow` github repo (`tensorflow/models/image/mnist/convolutional.py`) We selected that model for reproducibility and to demonstrate how to easily peek into popular models by using probes.

We start by sketching the model in figure 4. We report the results at the beginning and the end of training on figure 5. One of the interesting dynamics to be observed there is how useful are the first layers, despite the fact that the model is completely untrained.

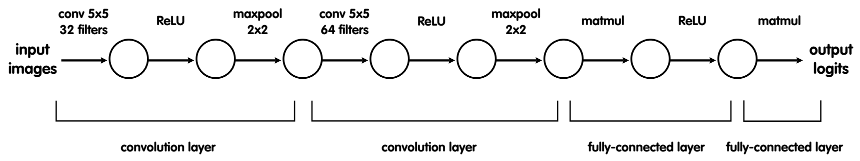
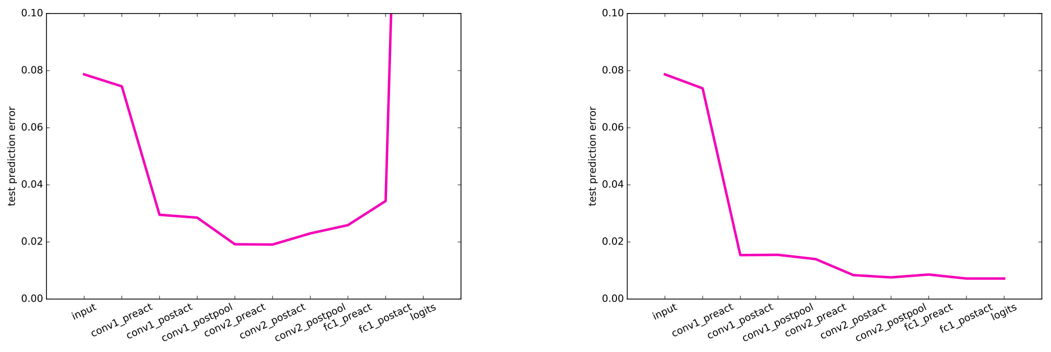


Figure 4: This graphical model represents the neural network that we are going to use for MNIST. The model could be written in a more compact form, but we represent it this way to expose all the locations where we are going to insert probes. The model itself is simply two convolutional layers followed by two fully-connected layer (one being the final classifier). However, we insert probes on each side of each convolution, activation function, and pooling function. This is a bit overzealous, but the small size of the model makes this relatively easy to do.



(a) After initialization, no training.

(b) After training for 10 epochs.

Figure 5: We represent here the test prediction error for each probe, at the beginning and at the end of training. This measurement was obtained through early stopping based on a validation set of 10^4 elements. The probes are prevented from overfitting the training data. We can see that, at the beginning of training (on the left), the randomly-initialized layers were still providing useful transformations. The test prediction error goes from 8% to 2% simply using those random features. The biggest impact comes from the first ReLU. At the end of training (on the right), the test prediction error is improving at every layer (with the exception of a minor kink on `fc1_preact`).

3.4 Auxiliary loss branches and skip connections

Here we investigate two ways to modify a deep model in order to facilitate training. Our goal is not to convince the reader that they should implement these suggestions in their own models. Rather, we want to demonstrate the usefulness of the linear classifier probes as a way to better understand what is happening in their deep networks.

In both cases we are going to use a toy model with 128 fully-connected layers with 128 hidden units in each layer. We train on MNIST, and we use Glorot initialization along with leaky ReLUs.

We choose this model because we wanted a *pathologically deep* model without getting bogged down in architecture details. The model is pathological in the sense that smaller models can easily be designed to achieve better performance, but also in the sense that the model is so deep that it is very hard to train it with gradient descent methods. From our experiments, the maximal depth where things start to break down was depth 64, hence the choice here of using depth 128.

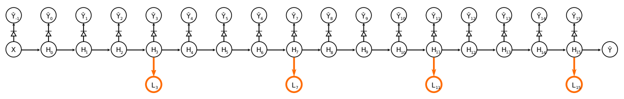
In the first scenario, we add one linear classifier at every 16 layers. These classifiers contribute to the loss minimized. They are not probes. This is very similar to what happens in the famous Inception model where "auxiliary heads" are used (Szegedy *et al.*, 2015). This is illustrated in figure 6a, and it works nicely. The untrainable model is now made trainable through a judicious use of auxiliary classifier losses. The results are shown in figure 7.

In the second scenario, we look at adding a bridge (a skip connection) between layer 0 and layer 64. This means that the input features to layer 64 are obtained by concatenating the output of layer 63 with the features of layer 0. The idea here is that we might observe that the model would effectively train a submodel of depth 64, using the skip connection, and shift gears later to use the whole depth of 128 layers. This is illustrated in figure 6b, and the results are shown in figure 8. It does not work as expected, but the failure of this approach is visualized very nicely with probes and serves as a great example of their usefulness in diagnosing problems with models.

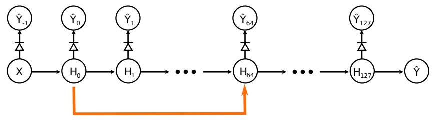
In both cases, there are two interesting observations that can be made with the probes.

Firstly, at the beginning of training, we can see how the raw data is directly useful to perform linear classification, and how this degrades as more layers are added. In the case of the skip connection in figure 8, this has the effect of creating two bumps. This is because the layer 64 also has the input data as direct parent, so it can fit a probe to that signal.

Secondly, the classification error goes down from the first probe to the last probe. This is even more apparent on the full video (instead of the 3 frames provided here). This is a ripple effect, where the prediction error in figure 6b is visually spreading from the left of the plot to the right. That is, during most of the training the prediction error goes down for a parent layer before it comes down for a layer itself.

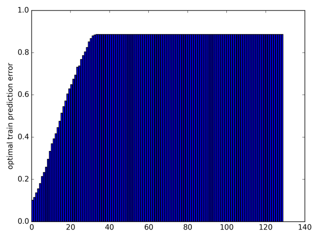


(a) Model with 16 layers, one guide at every 4 layers.

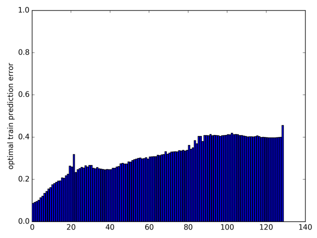


(b) Model with 128 layers. A skip connection goes from the beginning straight to the middle of the graph.

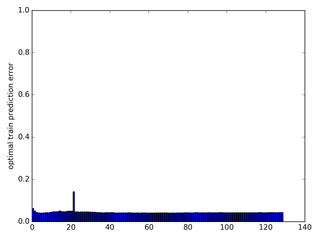
Figure 6: Examples of deep neural network with one probe at every layer (drawn above the graph). We show here the addition of extra components to help training (under the graph, in orange).



(a) probes after 0 minibatches

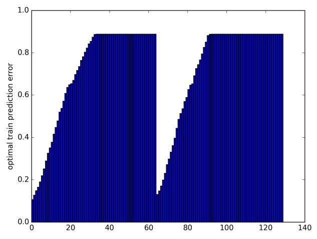


(b) probes after 500 minibatches

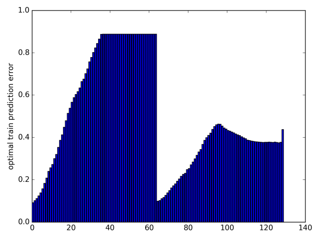


(c) probes after 5000 minibatches

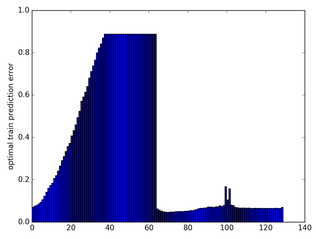
Figure 7: A pathologically deep model with 128 layers gets an auxiliary loss added at every 16 layers (refer to simplified sketch in figure 6a if needed). This loss is added to the usual model loss at the last layer. We fit a probe at every layer to see how well each layer would perform if its values were used as a linear classifier. We plot the train prediction error associated to all the probes, at three different steps. Before adding those auxiliary losses, the model could not successfully be trained through usual gradient descent methods, but with the addition of those intermediate losses, the model is "guided" to achieve certain partial objectives. This leads to a successful training of the complete model. The final prediction error is not impressive, but the model was not designed to achieve state-of-the-art performance.



(a) probes after 0 minibatches



(b) probes after 500 minibatches



(c) probes after 2000 minibatches

Figure 8: A pathologically deep model with 128 layers gets a skip connection from layer 0 to layer 64 (refer to sketch in figure 6b if needed). We fit a probe at every layer to see how well each layer would perform if its values were used as a linear classifier. We plot the train prediction error associated to all the probes, at three different steps. We can see how the model completely ignores layers 1-63, even when we train it for a long time. The use of probes allows us to diagnose that problem through visual inspection.

4 Discussion and future work

We presented results here for some toy models. While we think that this is sufficient to demonstrate the concept of the linear classifier probe and its usefulness, we really wanted to show the same kind of plots (and videos) on the Inception model running on the huge ImageNet dataset (~150GB).

After training the model for 2-3 weeks, saving a checkpoint every hour, we realized that training ~20 probes for every checkpoint was absolutely impossible. Realistically, given the size of the training set, it is even challenging to train single probe on a single checkpoint.

Training the probes is equivalent to solving a convex optimization problem. This means that we can make use of many convex optimization algorithms that the deep learning community are not currently using to train the deep networks themselves. We have not yet tried those methods, but we are looking at trying the Stochastic Average Gradient, and Stochastic Variance-Reduced Gradient methods (Roux *et al.*, 2012; Schmidt *et al.*, 2013; Harikandeh *et al.*, 2015).

5 Conclusion

In this paper we introduced the concept of the *linear classifier probe* as a conceptual tool to better understand the dynamics inside a neural network and the role played by the individual intermediate layers. We are now able to ask new questions and explore new areas. We have demonstrated how these probes can be used to identify certain problematic behaviors in models that might not be apparent when we traditionally have access to only the prediction loss and error.

We hope that the notions presented in this paper can contribute to the understanding of deep neural networks and guide the intuition of researchers that design them.

Acknowledgments

The authors would like to acknowledge the support of the following agencies for research funding and computing support: NSERC, FQRNT, Calcul Québec, Compute Canada, the Canada Research Chairs and CIFAR.

References

Bengio, Y., Frasconi, P., and Simard, P. (1993). The problem of learning long-term dependencies in recurrent networks. In *Neural Networks, 1993., IEEE International Conference on*, pages 1183–1188. IEEE.

Harikandeh, R., Ahmed, M. O., Virani, A., Schmidt, M., Konečnỳ, J., and Sallinen, S. (2015). Stopwasting my gradients: Practical svrg. In *Advances in Neural Information Processing Systems*, pages 2251–2259.

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, page 91.

MacKay, D. (2003). *Information Theory, Inference and Learning Algorithms*. Cambridge University Press.

Marsh, C. (2013). Introduction to continuous entropy.

Montufar, G. F., Pascanu, R., Cho, K., and Bengio, Y. (2014). On the number of linear regions of deep neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2924–2932. Curran Associates, Inc.

Pascanu, R., Montufar, G., and Bengio, Y. (2013). On the number of response regions of deep feed forward networks with piece-wise linear activations. In *International Conference on Learning Representations 2014 (ICLR 2014), Banff, Alberta, Canada*.

Roux, N. L., Schmidt, M., and Bach, F. R. (2012). A stochastic gradient method with an exponential convergence _rate for finite training sets. In *Advances in Neural Information Processing Systems*, pages 2663–2671.

Schmidt, M., Roux, N. L., and Bach, F. (2013). Minimizing finite sums with the stochastic average gradient. *arXiv preprint arXiv:1309.2388*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328.