Loopy Neural Nets: Imitating Feedback Loops in the Human Brain

Caswell, Isaac
icaswell@stanford.edu

Shen, Chuanqi
shencq@stanford.edu

Wang, Lisa
lisa1010@stanford.edu

Stanford University
450 Serra Mall, Stanford, CA 94305

Abstract

Artificial Neural networks purport to be biomimetic, but are by definition acyclic computational graphs. As a corollary, neurons in artificial nets fire only once and have no time-dynamics. Both these properties contrast with what neuroscience has taught us about human brain connectivity, especially with regards to object recognition. We therefore propose a way to simulate feedback loops in the brain by unrolling loopy neural networks several timesteps, and investigate the properties of these networks. We compare different variants of loops, including multiplicative composition of inputs and additive composition of inputs. We demonstrate that loopy networks outperform deep feedforward networks with the same number of parameters on the CIFAR-10 dataset, as well as nonloopy versions of the same network, and perform equally well on the MNIST dataset. In order to further understand our models, we visualize neurons in loop layers with guided backprop, demonstrating that the same filters behave increasingly nonlinearly at higher unrolling levels. Furthermore, we interpret loops as attention mechanisms, and demonstrate that the composition of the loop output with the input image produces images that look qualitatively like attention maps.

1. Introduction

Neural networks, in particular deep neural networks, comprise a nascent field that has seen much active research in recent years. A model that was inspired by the brain, whereby each neuron supports simple, primitive functions but an entire network of them allows the brain to perform complicated tasks, a neural network consists of a series of interconnected layers. Each layer of neurons computes a simple function of the previous layer, but amalgamating many layers together allows the final network to perform a range of arbitrarily complex tasks. Active research, fuelled by a huge increase in computing power, has transformed neural networks into a state-of-the-art technology. With neural networks, computers nowadays can act as personal assistants (Siri and Cortana), play video games [7], and identify cat videos [5].

However, artificial neural networks differ from their counterparts from nature in a distinct way; artificial neural networks are directed acyclic graphs (DAGs), but the network of neurons in our brains contain many feedback loops. In fact, neuroscience tells us that the architecture of the human brain is fundamentally cyclic. For instance, the well-documented "what pathway" and "where pathway", the two main visual object recognition systems in humans, contain many feedback loops, leading for instance to the phenomenon known as top down attention [1].

In this paper, we propose a new model, which we call loopy neural networks (LNNs). At a high level, a LNN mimics the cyclic structures in the human brain and is created by augmenting conventional neural networks with "loop layers" that allow information from deeper layers to be fed to lower layers.

2. Our Model

2.1. Theoretical Model

Fig 1 shows an simple example of our proposed model. In this example we use convolutional layers, but the same idea can also be applied to any type of layer. While neural networks are in general acyclic, the model above contains a loop. In particular, the output from loop undergoes elementwise addition with the input layer before being fed as input to the first layer again.

It would be ideal if the loop continued being processed until the result converged. However, in general convergence is not guaranteed, and neither would this be computationally feasible. Neither is this in fact especially biomimetic, as the input to a real brain is constantly changing. Therefore we approximate the loopy structure by simulating the execution of a small number of passes through the feedback loop.

To accomplish this, we propose a mechanism similar to that in recurrent neural networks (RNNs). The model is further defined by a parameter k (called the *unroll factor*),

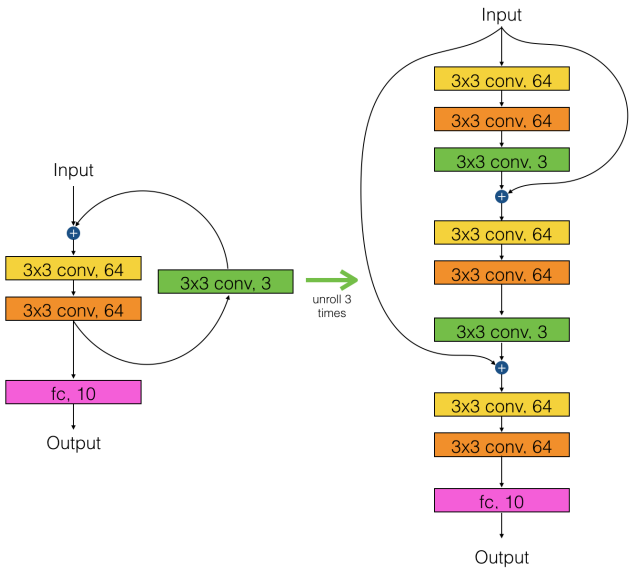

Figure 1. CIFAR-10 loopy model with loop layer. Unrolled network on the right.

which determines the number of times the loop will be processed. k is so named because it is very similar to the unrolling mechanism in RNNs. For example, the diagram on the right in fig. 1 depicts what happens when the network on the left is unrolled 3 times.

With the addition of loops, it is hoped that LNNs improve upon vanilla neural networks in the following ways.

1. **Feedback mechanism** By allowing lower-level layers to know the weights of higher-level features, a more refined choice of weights for the lower-level layers may be possible.

2. **Compact Representation** Even a shallow LNN can resemble a deep neural network when unrolled several times. Yet, the unrolled network uses far fewer parameters when compared to a deep neural network of the same depth. It is hoped that both networks can provide similar expressive power despite the discrepancy in the number of parameters. If this is found to be true, then LNNs can serve as a compact representation of complicated, deep models.

Training a LNN is very similar to training a vanilla neural network. After a LNN is unrolled, forward propagation can be performed in the standard manner. Backward propagation can also be done on unrolled LNN in the standard manner. However, due to shared parameters between layers in the unrolled network, the gradient needs to be updated differently. For example, in fig. 1, we will treat each layer in each unrolled layer as distinct and perform backward propagation as normal. However, the actual gradient of W_1 will be the sum of gradients of W_1 in each layer. In other words,

$$dW_i = \sum_{j=1}^{k} dW_{i,j}$$

where dW_i is the gradient of layer W_i and $dW_{i,j}$ is the gradietn of W_i at the j^{th} unroll.

2.2. Architecture

We wrote a library based on Lasagne [1] that allows us to specify the layer details, loop configurations, relevant hyperparameters etc in a config file[2]. We accomplish layer duplication in the unrolled case by tying the weights among layers which correspond to the same layer in the loopy model. This is as simple as passing in the same Theano shared variable to all unrolled layers.

3. Data Sets

Since our project is focusing on the design of a new neural net architecture, we decided to use two standard Computer Vision datasets, MNIST and CIFAR-10.

3.1. MNIST

MNIST is a dataset of handwritten digits [6]. It is considered a standard dataset in computer vision and pattern recognition. We started with this dataset since it requires less memory than datasets with color images. We used 40,000 images for training, 10,000 for validation and 10,000 for test. There 10 classes, one for each digit. The images are in grey-scale and the size of each image is 28x28 pixels. We did not do any other processing on the MNIST images before feeding them into our neural nets.

3.2. CIFAR-10

CIFAR-10 is a subset of the Tiny Images dataset. CIFAR-10 contains 60,000 color images from ten object classes and is also considered a standard dataset in computer vision. The ten classes are airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The distribution of classes across the data set is even and each image can only belong to one class of the ten classes. Examples from the CIFAR-10 dataset can be seen in (fig. 2)
Since the images are in color, each image is represented in 3 color channels. The image size is 32x32 pixels, resulting in a 3x32x32 representation for each image. We used 20,000 for training, 1000 for validation and 1000 for test. We did not do any other processing on the CIFAR-10 images before feeding them into our neural nets.

[1] which is based on Theano, which is based on C, which is based on assembly...a deep framework for a deep problem.

[2] See `https://github.com/icaswell/loopy_CNNs`

Figure 2. Sample images from CIFAR-10.

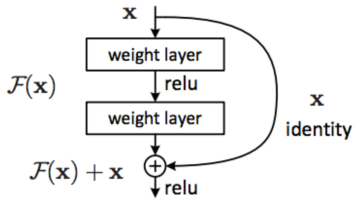

Figure 3. Building block of residual learning.

4. Related Work

There are models in existing literature that also try to incorporate loops into the model, and we derived some of our ideas from these models.

Recurrent Neural Networks. A Recurrent Neural Network (RNN) is a model that takes at each timestep an input from the external world and an input from itself from a previous timestep, summarizing all the information it's seen so far. The way that we unroll our networks leads to a very similar architecture: each copy of the network (each 'unroll') takes as input all outputs from loops from the previous unroll, as well as the output of the highest layer in the network with no upstream loop

The unrolled structure looks very much like an RNN, with the same input at every timestep. There are then two main conventional differences between LNN and RNN: firstly, as noted, the input is static, and secondly, there may be an arbitrary number of 'hidden states' output by a cell that are inserted into different points in the computational machinery of the next cell, usually by direct elementwise composition rather than pre-processing through a fully connected layer first. These are conventional differences in that they differ in the convention of how RNNs are constructed, rather than in their definition: one does not tend to see RNNs with elementwise multiplication of hidden state with input, for instance, or internal convolutional layers, or deep processing of input variables.

In summary: LNNs should be thought of as a generalization or a reformulation of an RNN with different semantics, approaching the problem of refining judgment on a static input rather than characterizing timeseries data.

ResNet: Deep Residual Learning. Deep residual nets try to address the problems that arise with very deep networks by reformulating layers as learning residual functions [3]. In their work, He et al. propose ResNet, a residual net with 152 layers, which won 1st places in all five main tracks of the ILSVRC & COCO 2015 competitions. Over the past six years of ILSVRC competitions from 2010 to 2015, the winning neural net architecture always had more layers than the winning net of the previous year. This suggested that high accuracies on image recognition tasks are correlated with the depth of the neural net.

However, with deeper networks, the problem of degradation arises. With increasing the depth, the validation accuracy saturates at some point and degrades afterwards. According to He et al., this is not a problem with overfitting, since the training accuracy goes down as well. Deep residual learning addresses degradation by adding identity shortcut connections, which skip layers. Fig. 3 shows a building block of residual learning with the identity shortcut. Our loopy network, once unrolled, is similar to a network with skip connections, with the difference that the loop connections feed into layers that share parameters with previous layers. Our addition loops are similar to these identity skip connections since they do not introduce extra parameters and feed into an addition node that combines two outputs from different layers. However, unlike the layers between the loop connection in our loopy networks, the layers of residual networks do not share parameters. Each weight layer in ResNet has its own individual set of parameters.

Recurrent CNN[9] The authors tackled the scene labelling problem i.e. provide each pixel of the image with a label that determines which class the pixel belongs to. To do so, they used a "recurrent convolutional neural network" model. The underlying CNN takes as input an image and the label predictions from the previous run (for the first run this is initialized to 0), and outputs the label scores for each context patch of the input image. The new set of label predictions is then fed into the CNN again with a scaled[3] version of the original image. This process continues for k iterations for a predetermined k. This is very similar to a specific subset of LNNs, namely those which have a loop output composed with the input image. The parameter k is analogous to our unrolling depth, specifying in both cases how often to digest the input image. Recurrent CNN attained 80.2% pixel accuracy and 69.9% class accuracy on the Stanford Background Dataset, on par with then-state-of-the-art results.

dasNet[10] The authors worked on the image classifica-

[3] scaled to the same size as the new set of label scores

tion problem on the CIFAR-10 dataset and employed reinforcement learning on a pretrained model. Their motivation, like ours, was specifically to imitate feedback loops in the brain. First, the authors constructed a pre-trained Maxout network[2] that could achieve an accuracy of 90.39%. Then, they used reinforcement learning to train a deterministic policy π based on the Maxout model. The policy outputs a vector a that scales each filter by a certain value (hence causing the model to focus its attention on certain filters/features). Initially a is set to all 1s (so all filters have equal importance). The input image is fed through the Maxout network and certain set of statistical values o are computed based on the output of each filter. o is then fed into π to obtain a', after which the input image is fed through the Maxout network again, this time parameterized by a'. This process continues for T iterations for a predefined T, and the class scores of the final run gives the desired prediction. With this new model, the authors achieved an accuracy of 91.19%.

This model is created under similar motivations, and its realization (multiple passes through an almost identical network) is also similar to our loop mechanism. However, while dasNet improves upon a network that already performs very well, we train models from scratch.

Aside: Loopy Belief propagation The loopy model is reminiscent of a cyclic probabilistic graphical model/Markov random field. For such instances, a popular class of inference techniques is loopy belief propagation (e.g. [8]), in which gradient-like signals are passed from each node (neuron) to all those nodes dependent on them, until convergence comes (in the ideal case). Unrolling could be seen as a variant of loopy belief propagation in with a specified order of message passing and a fixed number of iterations. Pure loopy belief propagation would have the advantage that it could be parallelized for each layer and even each filter, lending the possibility that given the right architecture, learning our models could be sped up significantly. This asynchronous, parallel approach is also more closely related to the neural reality we're trying to imitate.

5. Models

5.1. MNIST Models

We used a simple architecture with 3 layers for MNIST. All 3 layers consist of 3x3 filters, and the 3 layers have 16, 8 and 1 filter(s) respectively (fig. 4). There also exists a loop from the third layer to the first layer via an addition node. The model is unrolled 1[4], 2, 3 times and the results are analyzed. As simple models could achieve high test accuracy on MNIST, we did not experiment with different models on this dataset.

[4]Note that unrolling the model once means the loop is not utilized

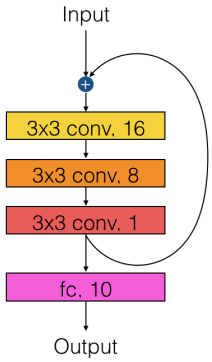

Figure 4. MNIST Loopy network with 3 conv layers.

5.2. CIFAR-10 Models

5.2.1 Degrees of Freedom

To compare the effectiveness of our proposed loopy network architecture with non-loopy models, we experimented with the following variations:

1. **Loop vs. no loop** We want to evaluate whether adding a loop without changing the number of parameters can affect accuracies and training behaviors.

2. **Unrolled loopy network vs. deep network** Adding loops might help networks learn more features and be more expressive. It could also offer a more compact model compared to a deeper network. Hence, we are interested in comparing a loopy net with its deep equivalent, i.e. a deep network with the same architecture but untied weights.

3. **Loop without parameters vs. loop with parameters** Finally, we can also add parameters within the loop itself. We would like to explore how the loop parameters influence the learning behavior by comparing its performance to a similar network that does not have parameters in the loop.

Based on these parameters, we designed the following four models:

1. **Vanilla**. The Vanilla net is a simple architecture with three convolutional layers. The first two conv layers have 64 3x3 filters, followed by a conv layer with 3 3x3 filters and a fully connected layer (fig. 5). The third conv layer has exactly 3 filters to allow us to add in a loop back into the first layer for the Loopy model.

2. **Loopy.** By extending the vanilla model with a loop around all three convolutional layers and unrolling three times, we get the loopy model. Note that the input has a depth of 3, hence the output from the loop

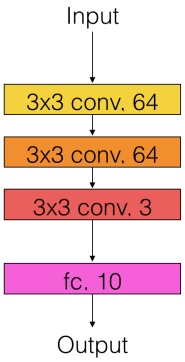

Figure 5. CIFAR-10 vanilla network with 3 conv layers.

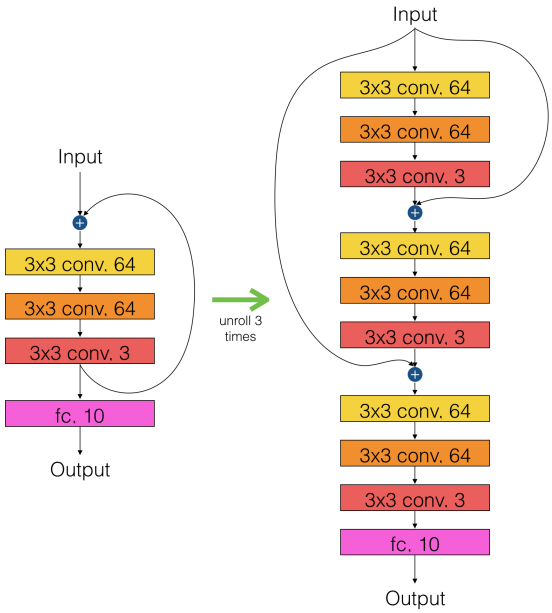

Figure 6. CIFAR-10 loopy network with 3 conv layers and 3 unrolls.

must also have a depth of 3. This explains why the third conv layer has exactly 3 filters. It has exactly the same number of parameters as the Vanilla network, since the loop does not introduce any new parameters. Comparing the results of `Vanilla` and `Loopy` allows us to evaluate the first metric and determine whether adding loops can improve the expressiveness or learning behavior of neural nets. Fig. 6 shows the `Loopy` net and its unrolled architecture.

3. **Deep.** This net has a very similar architecture compared to `Loopy`, but the layers in the unrolled network do not share parameters. It has three times more parameters than `Vanilla` and `Loopy`, and has the same depth as `Loopy`. We can compare the results of `Loopy` and `Deep` to evaluate whether a loop on a shallow net can imitate the behavior of a deeper net. Fig. 7 illustrates the architecture of the deep net.

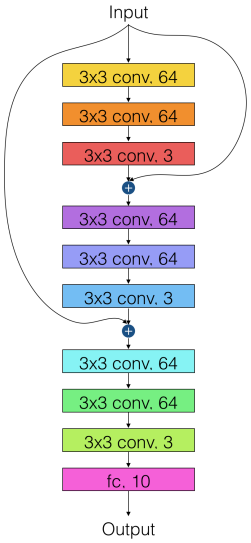

Figure 7. CIFAR-10 deep network with 9 conv layers.

4. **Loopy with loop parameters.** This net has one loop around the conv layers in the main stack and a conv layer in the loop itself. To keep the number of parameters the same, we reduced the number of conv layers in the main stack to two. We unrolled this model three times. Interestingly, once unrolled, this network looks almost identical to `Loopy`, but without the final 3 filter conv layer. This top layer can be seen as an informational bottleneck in `Loopy` as it compresses the final activations to only three filters (Fig 1), and as a result we might expect `Loopy with loop parameters` to be more expressive. In addition, having loop parameters implies that the the last conv layer before the fully-connected layer doesnt have to "split its purpose" between predicting labels and directing attention, since the semantics of attention directing can be handled entirely in the loop layer.

6. Results and Discussion

6.1. Training Time

Due to time constraints, we trained the models over a variety of machines with different configurations. However, the ratios between training times of the different models are consistent across all platforms used. Vanilla models have the shortest training time, while loopy and deep models have almost identical training times, taking about k times the training time used for the vanilla model. This coincides with our expectations, as the loopy model is trained after be-

train acc	vanilla	loopy	+layer	×layer	deep
batch 5	61.9%	72.6%	**90.5%**	89.9%	69.8%
batch 32	43.7%	45.2%	-	-	35.6%
val acc	vanilla	loopy	+layer	×layer	deep
batch 5	46.6%	**57.4%**	51.8%	37.7%	56.1%
batch 32	47.3%	42.45%	-	-	28.3%
test acc	vanilla	loopy	+layer	×layer	deep
batch 5	49.7%	**56.4%**	53.5%	45.1%	50.2%
batch 32	37.9%	40.8%	-	-	34.7%

Figure 8. CIFAR-10 accuracy results after 24 epochs, comparing different architectures with the same number o parameters: (1) vanilla network, (2) loopy network, (3) loopy network with loop layer and addition node, (4) loopy network with loop layer and multiplication node, and (5) deep network with the same architecture as (2) but untied weights. The two loop layer models were not trained with batchsize 32.

ing unrolled, and unrolled loopy model has the same configuration as that of the deep model. Therefore, while LNNs can provide savings in terms of memory, with the current implementation there is no savings in terms of training time.

6.2. Benchmarking Results

6.2.1 MNIST

With one unroll, a validation accuracy of 97% was achieved without any hyperparameter search. Increasing the number of unrolls did not decrease this value. This at least shows that adding loops does not make the model worse. It is hard, however, to distinguish between the performances of the models due to high accuracies.

6.2.2 CIFAR-10

Fig 8 shows our experimental results for CIFAR-10. We can observe several interesting trends.

1. **Loop vs. no loop** As expected, the loopy model performs significantly better than the vanilla model in all cases. This shows that LNNs are indeed an improvement over vanilla neural networks and give the model more expressive power.

2. **Unrolled loopy network vs. deep network** The loopy model is in fact able to outperform the deep network in all cases, confirming our hope that an LNN can be as expressive as (if not more than) a deep neural network of the same unrolled depth, despite the smaller number of parameters.

   Contrary to our expectations, the deep model sometimes performed worse than the vanilla model. There are several possible reasons for this. Firstly it could be due to computational constraints, as we did little hyperparameter tuning, and it could be that the loopy model is more robust to perturbations in hyperparameters than the deep model. Secondly, the deep model may require more epochs to attain better results. Thirdly, a deep neural network with the same depth as an unrolled LNN can be very deep when k is large, and such a network can experience problems with vanishing or exploding gradients. Because the gradient of a layer in an LNN is the summation of gradients of the layer in each unroll, layers in LNN are more robust to vanishing gradients. More experiments may need to be carried in this area to explore the discrepancies in the accuracy of the models.

   Building off this idea, another way of viewing the success of LNNs in our experiments is from a training-efficiency perspective. In the tied-weights scenario (Loopy), each update higher in the network simultaneously updates parameters at many layers in the network. Therefore, even in the case that the gradient vanishes completely before reaching the lowest layers, they will still receive meaningful updates. In addition[5], since there are multiple updates to each matrix per backwards pass, the training process may be sped up. We speculate that these may be some of the most important factors in our increased performance, and even if the idea of an LNN as we conceive it is completely superseded, this paradigm (updating parameters at a variety of depths with the higher, less corrupted error signal) may be used to train prohibitively large networks more effectively. (ResNet is an example of a similar principle.)

3. **Loop without parameters vs. loop with parameters** Both models with loop layers (multiplicative composition and additive composition) performed worse than the loopy model with lower validation and test accuracy—yet the models with loop layer have significantly higher training accuracy. This suggests that models with loop parameters overfit much more severely than models without loop parameters, indicating that they have higher expressive capacity. One suspects that with enough hyperparameter tuning, these models can outperform the other models.

   Curiously, additive composition performs significantly better than multiplicative composition, and multiplicative composition in fact has lower test accuracy than the vanilla network. We are unable to explain why this is so.

Aside: we initially used a batch size of 32 but decreased it to 5 in order to train different models more quickly. While

[5] pun intended

we expected models trained with larger batch size to perform better (because they are more resistant to the inherent randomness in stochastic gradient descent), we obtained consistently better results with batch size 5. We speculate that this is because the function space contains many local minima that are close together. Therefore, while a smaller batch size results in noisier gradients, it helps to escape local minima.

6.3. Guided Backprop

To get a better idea of what different filters in different layers respond to, we implemented guided backpropagation [4]. This entails introducing an artificial gradient at the filter wherein the gradient values for that filter are set to 1 and the values for all other filters in that layer are set to 0. This signal is then propagated into the original image. *Guided* backpropagation refines this technique, with the addition that all negative values in intermediate gradients are masked out, with the justification that this prevents intermediate neurons from decreasing the activation of the higher layer unit one aims to visualize.

The results of a sample run of guided backpropagation are shown in Fig 10 and 11. These figures show the reaction of the same filter at different levels of unrolling. Importantly, the same neuron seems to react in a more refined and nonlinear fashion when the number of unrolls increases. This lends further credence to the idea that LNNs can simulate the expressive power of a deeper network.

6.4. Merge Layer Visualizations

In our experiments we had exactly one loop, which was pointwise composed directly with the input. For this reason we also had the opportunity to visualize this merge node directly. (Recall that a merge node is the pointwise addition or multiplication of a loop output with the output of some other layer, in our case the input image.) The output of the merge node is a modified version of the input, and as such can be seen as a weighting of what higher levels in the network deem to be important elements of the input image. In the language of attention models, it can be interpreted as an attention map.

What we found is that for multiplication nodes, the modified input resembled an attention map, supporting the hypothesis that that loop output learns to weight the important parts of the input. (Curiously, for addition nodes, merge layers were very close to the original image.) Fig 9 shows images after the loop output from the first unroll has highlighted the important elements of the image, and then after the loop output from the second unroll has done the same. The higher the unroll layer, the more precise the attention map.

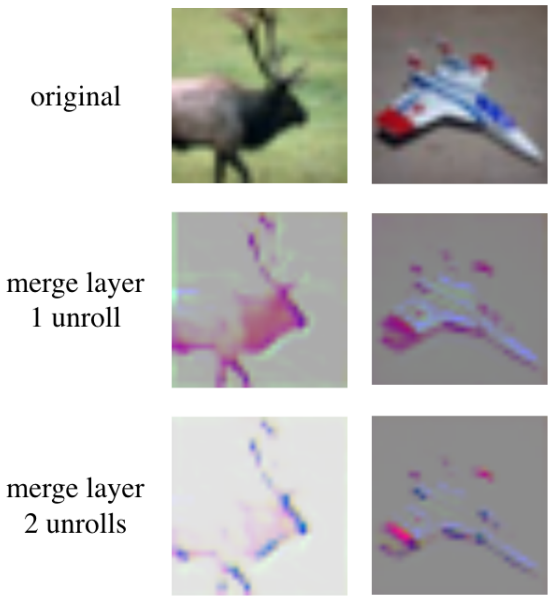

Figure 9. Original CIFAR image (row 1), followed by its attention map after the first and then second unrolling.

7. Future Work and Conclusion

We have demonstrated that LNNs are promising models that not only are more robust, but also give better performance when compared to their unrolled deep counterparts. The visualization experiments also suggest that even shallow layers in an LNN can exhibit characteristics of deep layers when unrolls are performed. Nevertheless, there still exist many experimental angles left to explore, and we discuss some of them below.

1. **Loop location/tightness** - All LNNs we explored contain a loop from the last convolutional layer to the first convolutional layer. It would be interesting to explore how loop tightness (e.g. loops between intermediate (even adjacent) convolutional layers) affect performance. We expect loops that are too tight to have little utility and possibly drown out higher level features.

2. **Loop complexity** - currently we test models with a single loop. An LNN with multiple loops may be able to achieve better performance.

3. **Optimizing Training Time** Even though LNNs contain much fewer parameters than their unrolled deep counterparts, the training time for both models is very similar. This is because we explicitly unroll the LNN and treat it as an unrolled deep network when performing forward and backward propagation. Given that each unroll is basically performing the same compu-

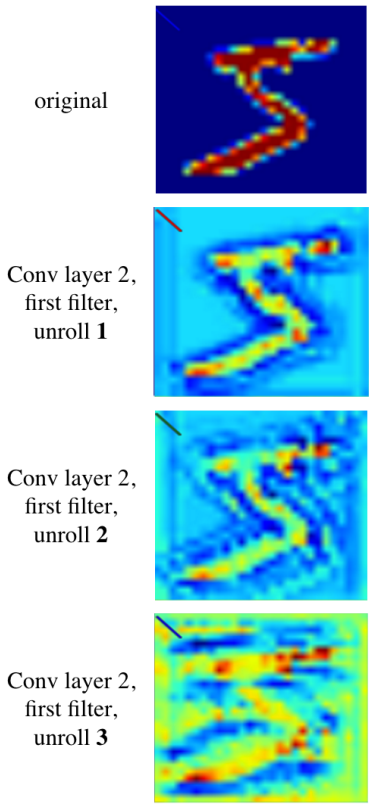

Figure 10. Guided back propagation on the same filter at different unrollings for an MNIST image. Note how it responds to different stimuli at different levels.

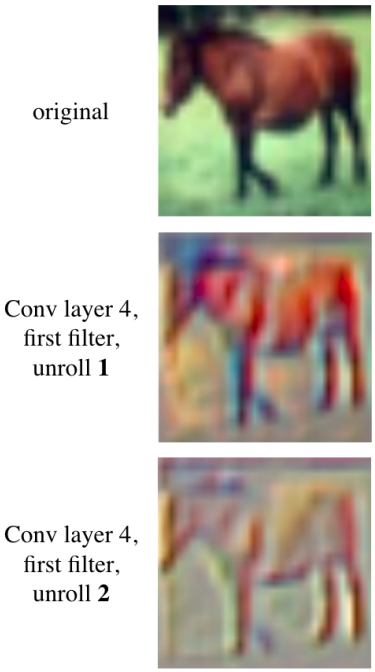

Figure 11. Guided back propagation on the same filter at different unrollings for an CIFAR-10 image. The responses become more refined.

tations again on a different "input", perhaps there exists a way to speed up this process. An example of such could be a closer imitation of loopy belief propagation.

4. **Experiments with larger models and more unrolls** These were out of our computational budget this time through, but perhaps next unrolling of this project we can demonstrate more expressive results...

References

[1] M. A. Goodale and A. D. Milner. Separate visual pathways for perception and action. *Trends in neurosciences*, 15(1):20–25, 1992.

[2] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *CoRR*, abs/1407.3068, 2013.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[4] T. B. M. R. Jost Tobias Springenberg, Alexey Dosovitskiy. Striving for simplicity: the all convolutional net. *ICLR*, 2015.

[5] Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, and A. Ng. Building high-level features using large scale unsupervised learning. In *International Conference in Machine Learning*, 2012.

[6] Y. LeCun, C. Cortes, and C. J. Burges. The mnist database of handwritten digits, 1998.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[8] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. *CoRR*, abs/1301.6725, 2013.

[9] P. H. O. Pinheiro and R. Collobert. Recurrent convolutional neural networks for scene parsing. *CoRR*, abs/1306.2795, 2013.

[10] M. Stollenga, J. Masci, F. J. Gomez, and J. Schmidhuber. Deep networks with internal selective attention through feedback connections. *CoRR*, abs/1407.3068, 2014.