Deep Neural Inspection Using DeepBase

Yiru Chen
yiru.chen@columbia.edu

Yiliang Shi
ys3130@columbia.edu

Boyuan Chen
bchen@cs.columbia.edu

Thibault Sellam
sellam@cs.columbia.edu

Carl Vondrick
cv2428@columbia.edu

Eugene Wu*
ewu@cs.columbia.edu

1 Introduction

There is currently excellent software and hardware infrastructure for every part of the neural network (NN) development lifecycle—creating models, training them, evaluating their accuracy, and deploying them. This has helped drive the excitement towards developing and deploying NN models in nearly every discipline and industry. Although neural networks today are largely evaluated on held-out test data, this does not guarantee models will behave reliably and correctly when deployed in practice. Models may encounter new situations that are statistically different from their training set and testing set, and researchers need to understand how their trained models will behave.

When learned models fail to work as expected, researchers often struggle to understand why, and how to fix it to ensure the same errors are not repeated. Unlike traditional software engineering where each line of code has clear semantic meaning that can be tested and verified, the logic of how a neural network behaves is embedded in thousands or millions of numerical weights. These weights do not directly correspond to high-level behavior that developers and users are able to easily reason about.

An emerging and promising technique, which we call *Deep Neural Inspection* (DNI), seeks to understand how the model internally behaves, rather than rely on end-to-end accuracy measures. To do so, researchers write complex scripts to extract activation "behaviors" for individual or groups of hidden units in a NN model, and develop measures to characterize these "behaviors" into higher order concepts such as "detected chair" [23] or "learned parsing rules" [19].

Manual approaches [21, 8], such as LSTMVis [21], visualize each unit's activations and let users manually check that the units behave as expected. Saliency Analysis, used in systems such as NetDissect [3] and other systems [20, 18, 24, 12, 15] models, seeks to identify the input symbols that have the largest "effect" on a single or group of units. For instance, we may want to find words that an LSTM's output is sensitive to [12], or the image pixels that most activates a unit [7]. This analysis may use different behaviors, such as the unit activation or its gradient. These procedures typically collect and identify the input symbols (e.g., pixels, characters) that triggered the highest unit activations or gradients. Further, some DNI analyses [9, 3, 13, 14, 2, 4] estimate statistical scores that compare the behaviors of individual or groups of units with image or text annotations. This approach seeks to identify correlations or statistical dependencies between hidden unit behaviors and e.g., pixels that represent a cat or text characters that constitute verbs or sentiment.

There are two primary limitations to these existing approaches towards DNI. First, much of the visualization or saliency-based analysis is predicated on manual, and limited, human judgment. Second, and more importantly, each analysis is specialized to a specific model and developed as a one-off script that must re-implement functionality for data and model management, behavior extraction, and performance optimization. It is akin to writing each database query as an one-off imperative program. This restricts the scope of developers that can inspect their models, and limits the

*All authors affiliated with Columbia University

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

complexity and scale at which model inspection can be performed. In short, ***there lacks a general abstraction and system to easily express and efficiently execute DNI analyses at scale.***

This paper describes a high level design of DeepBase, followed by two walk-throughs of performing deep neural inspection using the system. The purpose is to show how easily the developer can compare different models, compute different metrics that compare various hypotheses and combinations of hidden units. The primary bulk of code is to implement new metrics or new hypotheses, which can be accomplished through high-level APIs akin to user defined functions in database systems. ***Note that although we also report analysis results, the primary purpose is to highlight the easy of analysis.***

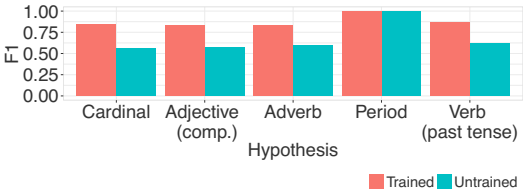
The first session analyzes a MiniPacman agent whose policies are predicted by a two-layer CNN trained via reinforcement learning. We use DeepBase to understand whether or not individual or groups of units learn low level game playing features such as identifying and focusing on pills and ghosts, as well as potentially higher level features such as estimating the distance between the Pacman and ghosts, or estimating where the ghost may move next. The second session analyzes facial recognition models and uses DeepBase to identify if the gender of a subject influences how a neural network finds facial landmarks. We include code snippets of each analysis, and highlight the small amount of changes needed to adapt the analysis to answer different questions. Both sessions are exploratory in nature, where we find value in the ability to rapidly iterate and try different variations of analyses.

Since deep neural inspection is a highly structured research task, our goal in designing DeepBase is to make this analysis fast and easy to perform. We hope that this empowers a wider audience of neural network ~~practitioners to perform~~ deep neural inspection on their models in novel ways.

2 Architecture and API Overview

```
import deepbase
models = [load_model("OpenNMT"),
          load_model("UntrainedOpenNMT")]
dataset = load_dataset("WMT 2015")
# Logistic regression acc between units and hypotheses
scores = [metrics.LogRegF1(thresh=0.01)]
# Code to parse dataset and translate parse trees
# into hypothesis functions
hypotheses = make_hyp_funcs(dataset)
deepbase.inspect(models, hypotheses, scores, dataset)
```

(a) Input Code



(b) Output of Our Prototype

Figure 1: Deep neural inspection on OpenNMT translation model using DeepBase. Results compared against an untrained OpenNMT model. (a) Python code snippet. (b) Logistic Regression F1 measure shows that both models learn low-level hypotheses (e.g., Period), but only the trained model learns higher level concepts.

This section introduces DeepBase's Python API, system design, and terminology through an example that inspects two English to German neural machine translation models from OpenNMT [10] to understand whether hidden units in the models learn to identify words based on their part of speech (verbs, nouns, etc) (Figure 1b). Details about the system are available in the technical report [17]. Figure 1a imports the DeepBase module, loads trained and untrained OpenNMT models, and the WMT 2015 competition dataset[2]. Each data point is a pair of English and German sentences. The inspection will be based on how the models behave when executed on the WMT corpus.

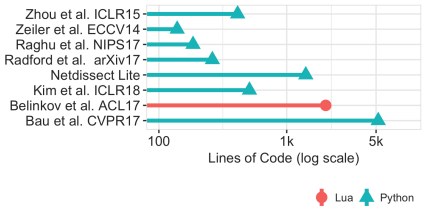


Figure 2: Lines of primary analysis code in papers that have publicly available implementations.

The code then loads *hypothesis functions* that encode higher-level language features as vectors over input symbols. Each language feature, such as verb, is translated into a vector that is 1 if the word is a verb and 0 otherwise. For image datasets, the hypothesis may output an annotation for each image pixel, or a single value for the entire image. The

[2] http://statmt.org/wmt15

developer then specifies that the affinity between the hidden unit activations and the language features should be quantified by measuring the cross-validation F1 prediction accuracy of a logistic regression model. Finally, the developer calls `deepbase.inspect()`.

To contrast with the above code snippet, we conducted a brief survey of existing DNI analysis implementations. Out of 10 papers, we found public implementations for 7 (including two versions of Netdissect). We attempted to identify code that was related to performing deep neural inspection (as opposed to visualization or imported library code). Figure 2 plots the approximate lines of code for each paper, and shows that every analysis is at least several hundred lines of code, and in some cases thousands of lines. This is an imperfect measure of complexity, but provides a sense of current analysis complexity.

On the right, Figure 1b shows the output produced by our proposed system. First, it confirms recent work showing that model architecture can act as a strong prior [1]. The untrained model has high affinity low level language features (e.g, periods), but low affinity for almost all high-level features. On the other hand, the trained model has far higher affinity to part of speech tags and phrase structure, such as Verbs, Cardinals, Adverbs, and Adjectives.

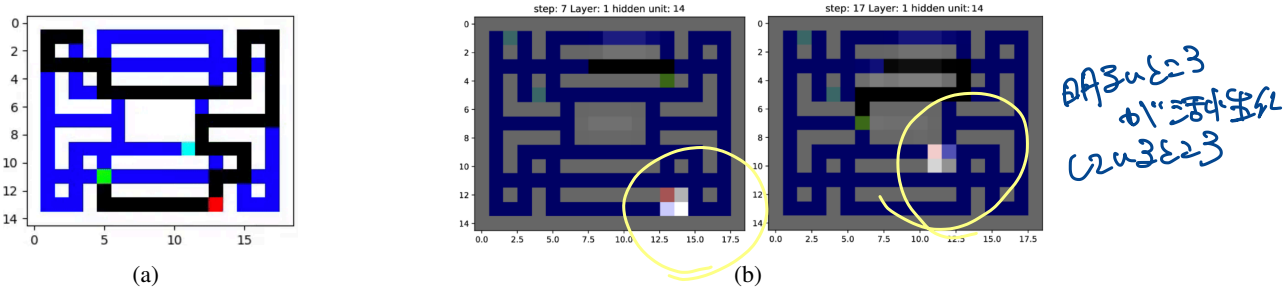
More importantly, notice that ***DeepBase internally handles the core analysis complexity***: it extracts hidden unit activations, manages the large intermediate activation and hypothesis data (easily 10s of GB for even simple models), and calculates affinity scores. The developer only needs to select the inputs of the inspection analysis she wants to run.

3 Experiments

This section shows how DeepBase helps inspect models in two different domains. We walk through how each analysis can be expressed using DeepBase. Our focus is to create a system that makes visualization and interpretation of neural networks easy to use for everyone.

3.1 Inspecting Agents in Games using MiniPacman

MiniPacman [22] is a game environment for learning policies for Pacman and Ghost agents. We are interested in analyzing and characterizing the learned individual or groups of units.



(a) (b)

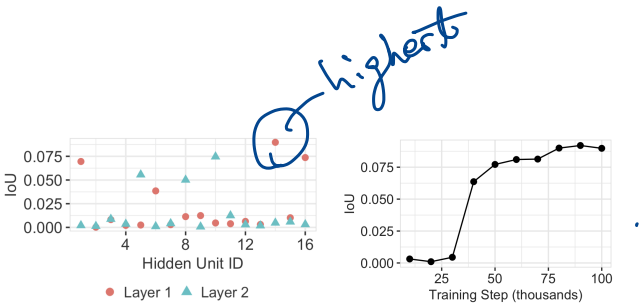
Figure 3: (a) Frame from Minipacman. Pacman is green, ghost is red, power pills are cyan, food is dark blue, and empty corridors are black. (b) Unit 14's activations are centered around the ghost.

3.1.1 Experimental Setup

The game consists of a 15×19 pixel map (Figure 3a), where the agents can move one pixel in any non-wall direction. The walls are fixed, and food that the Pacman can eat is found throughout the maze. There are two power pills that turns Pacman faster and able to eat the ghosts for a short amount of time. The number of ghosts is set to 1. We use actor-critic to train the agent models [11], and use the baseline 2-layer fully-connected convolutional model with 16 units per layer, as described in [22]. We use a deterministic policy for the ghost, and train the Pacman agent to maximize its reward function. We use the original reward function from [22]. However, what type of policy has the model learned? What features does the model use, and does the policy plan ahead by focusing on future paths that the Pacman or ghost will take? We show how DeepBase can help perform this analysis.

3.1.2 Results

```
    models = [torch.load("pacman")]
#   models = <load pacman at training steps>
      data = pickle.load(open("dataset.pkl"))
    scores = [metrics.LocalIoU()]
     units = [(1, '*'), (2, '*')]
#    units = [(1, 14)]  # (layer 1, unit 14)
       hyp = [f_ghost]
deepbase.inspect(models, hyp, scores, data, units)
```



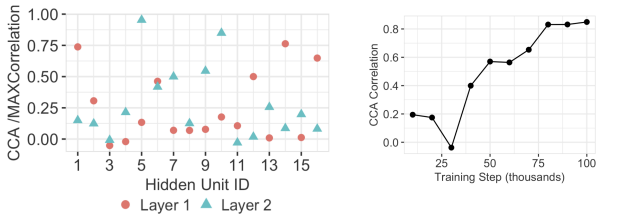
(a) (b) (c)

Figure 4: (a) Code to analyze model focus. (b) IoU score for each hidden unit. (c) IoU score at different training steps for unit 14 in layer 1.

Which units track the ghost and Pacman? We developed the hypothesis `f_ghost` that annotates the location of the ghost in each frame, and chose to compute the IoU (jaccard) measure between each unit's highest activation pixels and the annotated frame. IoU was proposed in NetDissect [23] to identify units that uniquely activate for objects in images. Figure 4a shows the code snippet to conduct this analysis for units in the first convolutional layer. The main coding effort is to implement the hypothesis annotation, which is straightforward. Figure 4b shows the IoU score for each hidden unit in the first convolutional layer ("conv1") and the second convolutional layer ("conv2"); unit 14 in "conv1" exhibits the highest score 8.9%.

We then modify the analysis (uncomment the gray lines) to load versions of the model at different steps during training, and using a tuple to specify that we want to analyze layer 1 unit 14. Figure 4c shows that after 20k training iterations, the unit quickly learns to focus on the ghost. 3b renders unit 14's activation for two frames in the game (lighter is higher activation). We can see the hidden unit indeed detects the ghost location.

```
    models = [torch.load("pacman")]
      data = pickle.load(open("dataset.pkl"))
      data = <filter data for edible states>
    scores = [metrics.CCA()]
     units = [(1,'*'), (2, '*')]    # layers 1 and 2
       hyp = [rollforward(3, 'ghost', models[0])]
deepbase.inspect(models, hyp, scores, data, units)
```



(a) (b) (c)

Figure 5: (a) Code to analyze planning ahead. (b) CCA correlation scores for each hidden unit. (c) CCA scores over different training steps for unit 10 in layer 2.

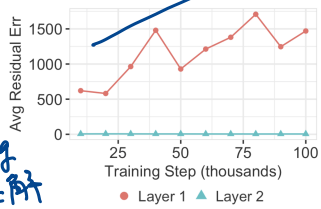
Planning Ahead: Does the learned policy react to the environment or plan ahead? In other words, does the model predict where the ghost will be in future time steps? To study this, we develop a hypothesis function `rollforward` that unrolls the game three steps into the future, and annotates the ghost's location for each of those time steps. In addition, we filter the dataset for only the frames where Pacman has recently eaten a power pill, so that the ghost will always run away from Pacman. This is simply a filter over the game state. We select two groups of units, layer 1 and layer 2, to analyze. Finally, we want to understand if a unit's activation is correlated with the ghost trajectory. Since the internal representations may be distributed, we fit a linear transformation with CCA on both the hypothesis and hidden activation space to maximize correlation, and report the correlation on a held-out test set. Adapting the program simply requires changing the red parts of the code snippet (Figure 5a).

Figure 5b shows that units 1, 14 in the first layer and units 5, 10 in the second layer are highly correlated with the edible ghost trajectories, with correlation of of over 75%. Figure 5c further shows the correlation for a single unit 10 in layer 2 throughout training, and we see that it is learned linearly over time.

Internal State: The final example illustrates how DeepBase can be used to identify whether any combination of hidden units behave in a way that is similar to tracking state information about the

```
models = [torch.load("pacman")]
  data = pickle.load(open("dataset.pkl"))
scores = [metrics.LinReg()]
 units = [(1,'*'), (2, '*')]  # layers 1 and 2
   hyp = [shortestpath]
deepbase.inspect(models, hyp, scores, data, units)
```

(a)



(b)

Figure 6: (a) Code to analyze shortest path distance between Pacman and ghost. (b) Average residual error for regression models trained on activations for each layer. Layer 1 gets worse for models at later NN training steps, while layer 2 has low prediction error.

game. Specifically, we encode a hypothesis function that returns the shortest-path distance between the Pacman and ghost. This function outputs one scalar for each frame, rather than one value for each pixel. We then switch the affinity measure to logistic regression that measures whether the unit activations are highly predictive of the distance. Similar to the previous examples, the code simply changes in the hypothesis and measure that is selected.

Figure 6b shows the average prediction error for the regression models trained on each layer for snapshots of the model at different training steps. We find that further training *degrades* the prediction accuracy for the layer 1 regression model, while the accuracy for layer 2 starts and remains low. Such surprising results are likely to be common case as researchers seek to better understand the dynamics of NN models, and underscore the value of systems to enable fast analysis iteration.

3.2 Facial Identification Inspection

DeepBase allows users to easily perform inspection on new datasets and trends to verify various hypothesis. Here, we showcase DeepBase's utility in simplifying the investigation of relatively complex models. Through its API, we are able to quickly generate a table representing the relationship between hidden units activation and properties we are concerned with, namely the presence of gender bias in a network.

The presence of bias in computer vision systems is a serious issue that is increasingly prominent in recent years. In [5] for instance, the authors find that commercial facial analysis systems have a higher gender classification failure rate for women and people of color compared to white men. To discover and minimize such issues, easy to use tools that expose the inner workings of complex neural networks is thus necessary.

This experiment shows how DeepBase can be used to perform an analysis to study whether units in a facial analysis model learns basic facial features such as eyes and noses. We then check to see if the gender of faces influence the behavior of units, a sign of possible bias.

3.2.1 Experiment Setup

We inspected the ResNet50 identity classification network, used in [6] to evaluate the Vggface2 dataset. The network has 50 layers, with up to 1023 hidden units in a given layer. The authors of [6] provided a trained model with a 3.9% top-1 classification error on new identities, which implies the network encodes general feature well. We also examine the same network with untrained weights as a control.

```
models = [resnet50, resnet50untrained]
  data = <load dataset>
scores = [metrics.LocalIoU()]
 units = [(i,'*') for i in [2, 5, 10, 15, 20,
                            25, 30, 35, 40, 45, 48]]
   hyp = <load annotations>
deepbase.inspect(models, hyp, scores, data, units)
```

Figure 7: Code snippet for inspecting VGG model

We develop hypothesis functions based on annotations of facial landmarks. Specifically, pixel-level annotations of the eyes, nose, mouth, and face, as well as a bounding box of the face. For each landmark, the corresponding hypothesis function generates a binary mask that marks the corresponding landmark pixels.

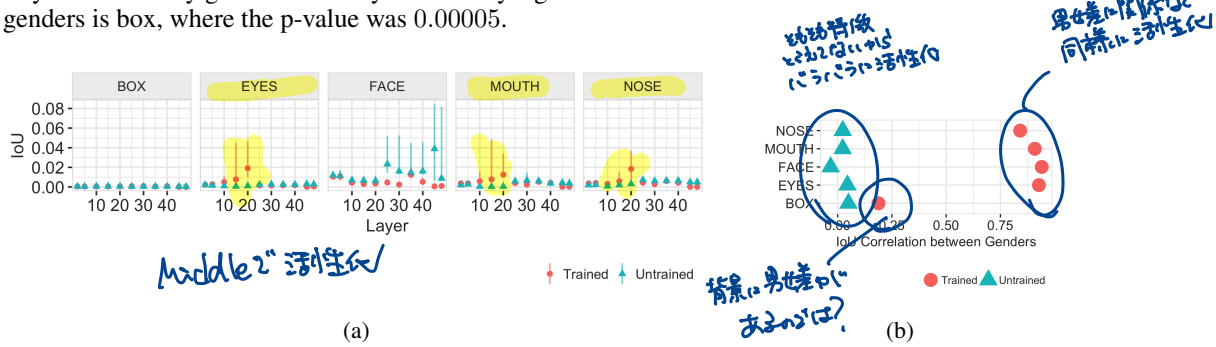
We expect low unit behavior correlation between the male and female test data for the untrained model and a high correlation for the trained model, as activation should be random for the first and targeted for the latter. A difference for the trained model would imply that the the neural network separately evaluates the feature landmarks for males and female.

Figure 7 shows the code snippet to compare 11 evenly spaces layers in the network with hypothesis functions. The code is essentially identical to the code snippets above, and the main change is the specification of the desired layers. This allows us to compute over 80,000 IoU scores with very few lines of code. Similarly, it is straightforward to partition the test dataset by gender and compute scores for each partition.

3.3 Results

Figure 8a plots the upper 10^{th} percentile IoU scores. Each point is the 95th percentile, and the whiskers show the 90th and 99th percentile IoU scores. For the eyes, mouth, and nose hypotheses, trained middle layers have top activations much higher than the untrained. This indicates that the middle layers could be responsible for detecting these features. The untrained IoU score is unexpectedly higher than the trained for face in mid and later layers, which could be due to face having a larger annotation area than other features.

We also compared the distributions of IoU scores for the trained model between 2 input data sets, female and male, to test if the network activates differently based on gender. Figure 8b reports the correlations between scores for female and male images on trained and untrained models. We find that their IoU scores are nearly perfectly correlated for all hypotheses except the bounding box, which we hypothesize is because the bounding box marks many pixels that are not strictly part of the face. It is possible that the model learns to recognize different characteristics of the facial background in a way that differs by gender. The only statistically significant difference in feature IoU scores between genders is box, where the p-value was 0.00005.



(a) (b)

Figure 8: (a) IoU Scores for each layer (x-axis) and each hypothesis (facets) (b) Correlation between IoU scores of model trained on Male and Female datasets.

4 Conclusions and Discussion

DeepBase generalizes and makes it easier to implement and perform deep neural inspection on neural network models. We showed how it can be used to inspect sequence based natural language models, as well as convolutional models used to make game-playing policy decisions as well as for facial recognition. DeepBase is initialized with a default library of affinity measures (CCA, linear regression, linear classification, and jaccard/IoU). It has adapters to support Tensorflow, Keras, and (some) PyTorch models. The main programming effort is to define hypothesis functions, and to choose the models, datasets, and hidden units to inspect.

The Python API can be somewhat unwieldy if the user wants to perform more complex pre/post processing, and we are developing a SQL-like query interface to simplify such analyses. As described in an earlier vision paper [16], we believe this is a building block for infrastructure to inspect, understand, and compare neural network models at scale, in a similar way to software infrastructure to analyze and test imperative software programs.

References

[1] J. Adebayo, J. Gilmer, I. Goodfellow, and B. Kim. Local explanation methods for deep neural networks lack sensitivity to parameter values. *ICLR*, 2018.

[2] G. Alain and Y. Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

[3] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba. Network dissection: Quantifying interpretability of deep visual representations. *arXiv preprint arXiv:1704.05796*, 2017.

[4] Y. Belinkov, N. Durrani, F. Dalvi, H. Sajjad, and J. Glass. What do neural machine translation models learn about morphology? *arXiv*, 2017.

[5] J. Buolamwini and T. Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on Fairness, Accountability and Transparency*, pages 77–91, 2018.

[6] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman. Vggface2: A dataset for recognising faces across pose and age. In *Automatic Face & Gesture Recognition (FG 2018), 2018 13th IEEE International Conference on*, pages 67–74. IEEE, 2018.

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[8] A. Karpathy, J. Johnson, and L. Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[9] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. B. Viégas, and R. Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *ICML*, 2018.

[10] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. Opennmt: Open-source toolkit for neural machine translation. In *ACL*, 2017.

[11] V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[12] J. Li, X. Chen, E. Hovy, and D. Jurafsky. Visualizing and understanding neural models in nlp. *arXiv preprint arXiv:1506.01066*, 2015.

[13] A. S. Morcos, D. G. Barrett, N. C. Rabinowitz, and M. Botvinick. On the importance of single directions for generalization. *arXiv*, 2018.

[14] M. Noroozi and P. Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pages 69–84. Springer, 2016.

[15] A. Radford, R. Jozefowicz, and I. Sutskever. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*, 2017.

[16] T. Sellam, K. Lin, I. Y. Huang, C. Vondrick, and E. Wu. "i like the way you think!" inspecting the internal logic of recurrent neural networks. *SylML*, 2018.

[17] T. Sellam, K. Lin, I. Y. Huang, M. Yang, C. Vondrick, and E. Wu. Deepbase: Deep inspection of neural networks. *CoRR*, abs/1808.04486, 2018.

[18] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *See https://arxiv. org/abs/1610.02391 v3*, 7(8), 2016.

[19] X. Shi, I. Padhi, and K. Knight. Does string-based neural mt learn source syntax? In *EMNLP*, 2016.

[20] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[21] H. Strobelt, S. Gehrmann, H. Pfister, and A. M. Rush. Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE transactions on visualization and computer graphics*, 24(1):667–676, 2018.

[22] T. Weber, S. Racanière, D. P. Reichert, L. Buesing, A. Guez, D. J. Rezende, A. P. Badia, O. Vinyals, N. Heess, Y. Li, R. Pascanu, P. Battaglia, D. Silver, and D. Wierstra. Imagination-augmented agents for deep reinforcement learning. *CoRR*, 2017.

[23] B. Zhou, D. Bau, A. Oliva, and A. Torralba. Interpreting deep visual representations via network dissection. *arXiv preprint arXiv:1711.05611*, 2017.

[24] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*, 2014.