Report

Training a tiny SupAmp model on easy tasks

The influence of failure rate on learning curves

Richard Möhn

24th January 2020

Contents

1 Introduction

(For a project overview and a glossary, see the home page of the Farlamp repository.)

BLUF: The learning behaviour is as roughly expected. X_{PA} compensates for some overseer failures, although not yet enough to be reliable. Mistakes I made prevent me from drawing further interesting conclusions from the experiment results so far. OQ and TODO mark questions and tasks for future research.

I ran experiments with a tiny SupAmp model on easy tasks. Small data, short training times and local execution allow for a tight feedback loop with few moving parts. Thus I could understand and adapt the code, and fix mistakes quickly.

Table 1: Explanations of the labels in the diagrams

label in diagrams	explanation using the terms of this report
`accuracy/targets`	validation accuracy of $\text{Amplify}^{H'}(X_{\text{PA}})$
`accuracy/teacher`	validation accuracy of X_{PA}
`accuracy/train`	training accuracy of X
`accuracy/validation`	validation accuracy of X
`accuracy_on/<d>/targets`	validation accuracy of $\text{Amplify}^{H'}(X_{\text{PA}})$ on questions/answers of depth d
`accuracy_on/<d>/teacher`	validation accuracy of X_{PA} on questions/answers of depth d
`accuracy/asker/q/train`	training accuracy of H' on imitating the sub-questions that H would ask
`accuracy/asker/q/validation`	validation accuracy of H' on imitating the sub-questions that H would ask
`accuracy/asker/a/train`	training accuracy of H' on imitating the root answers that H would give
`accuracy/asker/a/validation`	validation accuracy of H' on imitating the root answers that H would give

The drawback is that it doesn't yield many insights. One result (click to jump there), however, was encouraging: X_{PA} does learn to be more reliable than its overseer $\text{Amplify}^{H'}(X_{\text{PA}})$. The rest of the results fall into two categories: Results confirming my understanding of the code and learning behaviour. And results made useless by mistakes and misunderstanding of mine. Although the latter sounds bad, it is actually useful to have exposed these mistakes before running more expensive experiments.

In this report I quickly note and discuss the methods, observations and lessons learned before I move on. 'Quickly' also means that you have to content yourself with screenshots of TensorBoard instead of pretty plots. Please excuse the unpolished prose and other rough edges – I had to move on and start preparing for my AISC interview.

1.1 Terms

Questions are answered by $\text{Amplify}^{H'}(X_{\text{PA}})$. The resulting (question, answer) pairs are used to train X. From X $\underline{X_{\text{PA}}}$ is derived by Polyak averaging. This is the same as in Christiano et al. (2018), except that I introduce the symbol X_{PA} in order to separate the learner from the assistant. I distinguish them here, because they are distinguished in the code. From the code also derive the labels in the diagrams that you will see later. Table 1.1 explains these using the terms and symbols of this report.

The <u>depth</u> d of a question is equal to its height plus one. The height is defined in Overseer failures in SupAmp and ReAmp. They have the same definition, just a different starting point. Henceforth I will use depth instead of height, in order to be consistent with Christiano et al. (2018) and the SupAmp code.

$\underline{N_a} = N + 1$ is the number of answers a learner can give. – Any of the numbers of the domain plus a symbol for 'I don't know'.

2 Method

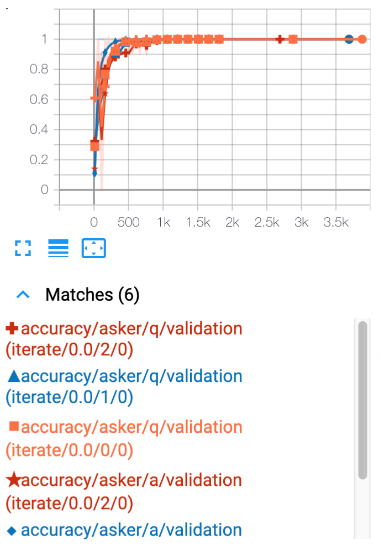
I ran SupAmp on the permutation powering task from Christiano et al. (2018). The code was written originally by Paul Christiano, as far as I know, then modified by William Saunders, then by me. Fifteen trials were run in total, three each with error probability $p_F = 0, 0.01, 0.1, 0.3, 1.0$ and the following configuration:

- One-layer transformer for both H' and X (if I understand it right).
- Permutations of four numbers ($N = 4$), up to the seventh power ($2 \leqq k \leqq 7$).
- Train X with 4000 batches à 50 contexts with 4 root questions and answers each. After X is done with 4000 batches, also quit the training of H', no matter how far it got.
- Other hyperparameters as they were when I got the code.

3 Observations and discussion

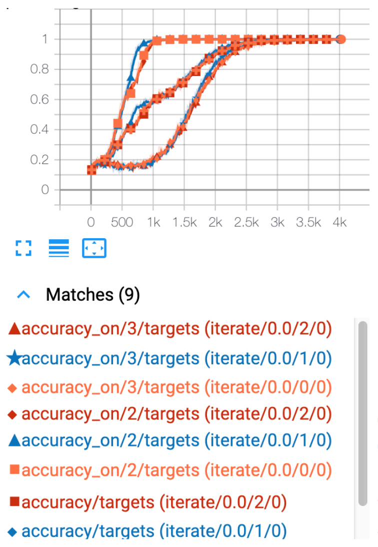
In the following screenshots from TensorBoard the smoothing factor is 0.6 unless noted otherwise.

3.1 Without failures ($p_F = 0$)



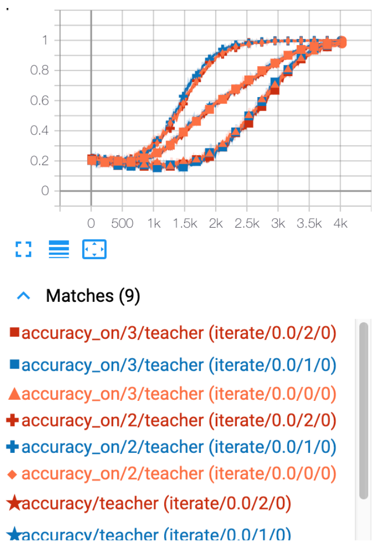
Matches (6)

✚accuracy/asker/q/validation (iterate/0.0/2/0)
▲accuracy/asker/q/validation (iterate/0.0/1/0)
■accuracy/asker/q/validation (iterate/0.0/0/0)
★accuracy/asker/a/validation (iterate/0.0/2/0)
◆ accuracy/asker/a/validation



Matches (9)

▲accuracy_on/3/targets (iterate/0.0/2/0)
★accuracy_on/3/targets (iterate/0.0/1/0)
◆ accuracy_on/3/targets (iterate/0.0/0/0)
◆ accuracy_on/2/targets (iterate/0.0/2/0)
▲accuracy_on/2/targets (iterate/0.0/1/0)
■accuracy_on/2/targets (iterate/0.0/0/0)
■accuracy/targets (iterate/0.0/2/0)
◆ accuracy/targets (iterate/0.0/1/0)

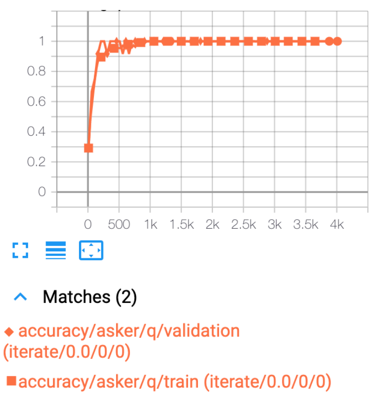


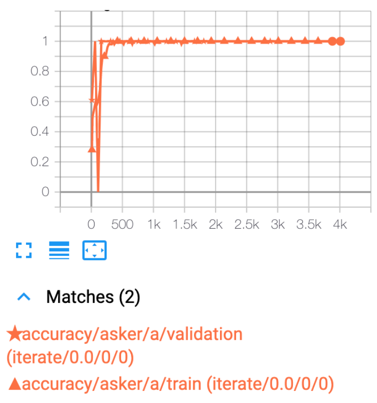
Matches (9)

■accuracy_on/3/teacher (iterate/0.0/2/0)
■accuracy_on/3/teacher (iterate/0.0/1/0)
▲accuracy_on/3/teacher (iterate/0.0/0/0)
✚accuracy_on/2/teacher (iterate/0.0/2/0)
✚accuracy_on/2/teacher (iterate/0.0/1/0)
◆ accuracy_on/2/teacher (iterate/0.0/0/0)
★accuracy/teacher (iterate/0.0/2/0)
★accuracy/teacher (iterate/0.0/1/0)

Observation The learning curves for all three trials are similar, the variation between trials is low.

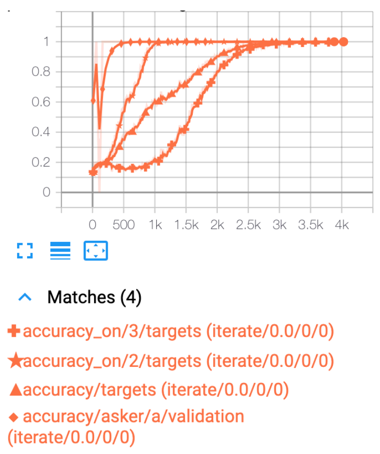
Discussion This allows me to select only the first trial for further analysis.

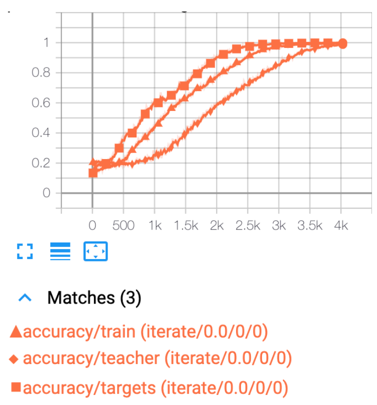




Observation (Smoothing turned off.) The accuracy of H' bounces around in the beginning.

Discussion I don't know why. One reason is that it is measured only every fifty batches, OQ much more seldom than the other accuracies. The bouncing is no problem, since H' reaches perfect accuracy quickly.

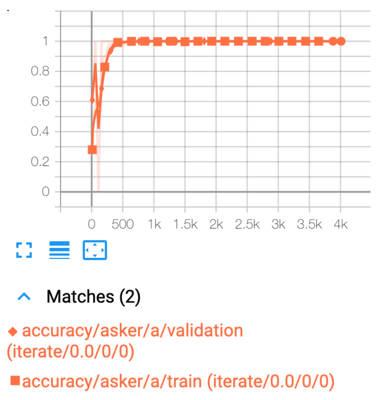


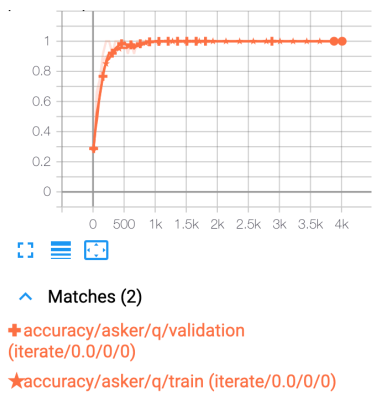


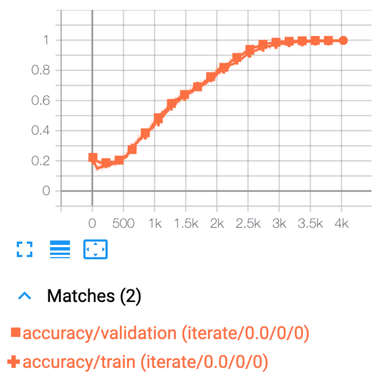
Observation H' reaches 1.0 quickest (talking about accuracy), then $\text{Amplify}^{H'}(X_{\text{PA}})$ on depth-2 questions. Thereafter, $\text{Amplify}^{H'}(X_{\text{PA}})$ on depth-3 questions and on all questions reach 1.0 at the same time. X lags behind $\text{Amplify}^{H'}(X_{\text{PA}})$, and X_{PA} lags behind X.

Discussion This is what I roughly expected. These accuracies all depend on each other. H' needs to become good enough for the amplification to generate sensible training data for X. At first the only correct inputs $\text{Amplify}^{H'}(X_{\text{PA}})$ gets are answers to depth-1 (ie. primitive) questions. With the answers to depth-1 questions, $\text{Amplify}^{H'}(X_{\text{PA}})$ can answer depth-2 questions, which is the next accuracy that goes up. Once depth-2 answers hit the mark often enough, the training for depth-3 answers becomes effective. Depth-2 and depth-3 answers make up all answers, so the accuracy on all answers is in between.

Amplify$^{H'}(X_{\text{PA}})$ delivers the training data for X, and X_{PA} is derived from X. This explains the respective lags.

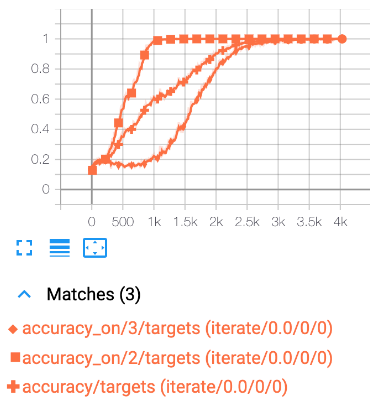


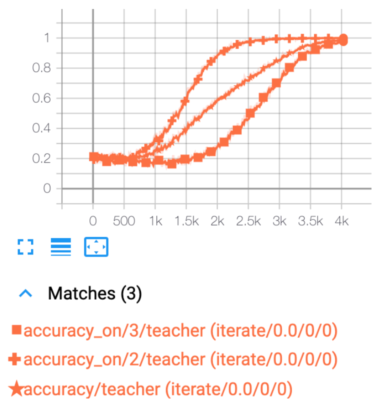


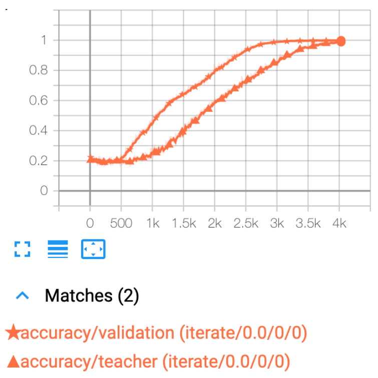


Observation Training and validation accuracy curves are remarkably close both for X and H'.

Discussion I don't know why this is. It might be that the validation data are too similar OQ
to the already trained data and therefore not good enough. This might be because the domains are too small. Or perhaps the learners barely overfit, because the training data are always random and new.

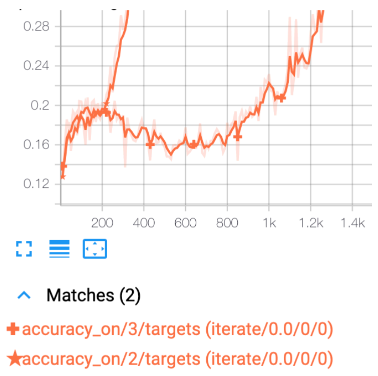


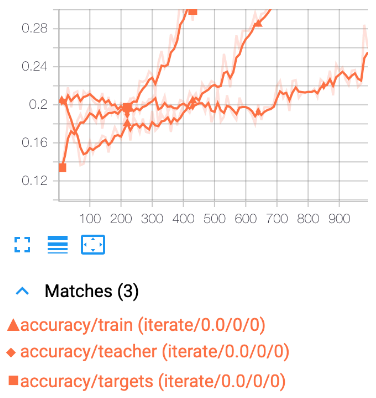




Observation The learning curves of X_{PA} are smoother than those of Amplify$^{H'}(X_{\text{PA}})$and X.

Discussion I guess that this is because of the Polyak averaging. It averages the weights of the last 1000 steps. This biases the predictions to be similar to the predictions during the last 1000 steps, and therefore reduces the variance in the accuracy.





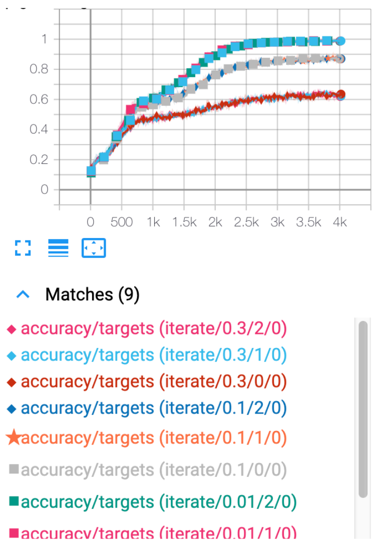
Observation At the beginning of the training $\text{Amplify}^{H'}(X_{\text{PA}})$, X and X_{PA}, in this order, go through a section of less-than-chance accuracy. (The accuracy of random outputs would be $\frac{1}{N_a}$.)

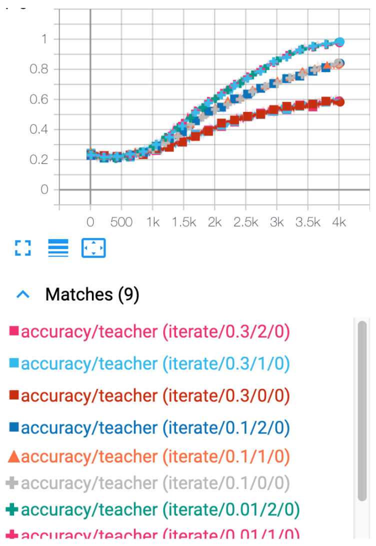
Discussion I don't know why this happens. It might have to do with H' not being fully trained, or with the special response 'I don't know', which is given non-randomly. It is not important to find out now. OQ

3.2 With failures injected ($p_F > 0$)

Unfortunately I made two mistakes: I let the trials run for too few steps, stopping it before training converged for depth-3 questions. This means that the accuracy numbers of X_{PA} for these are almost useless. And I was confused about the injecting of errors, putting it in the wrong place or measuring accuracy wrongly or calculating the theoretical accuracy wrongly – or all of these together. I haven't decided yet how to resolve the issue. TODO The result is that the accuracy numbers of $\text{Amplify}^{H'}(X_{\text{PA}})$ on depth-3 questions are almost useless as well.

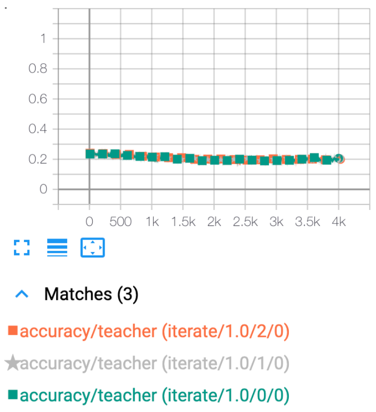
For now I don't have time to clarify my thoughts, fix the code and rerun the experiments. So I will only note and discuss the few valid observations that are left.





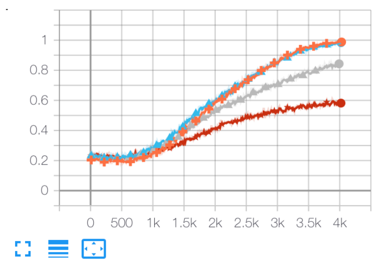
Observation Also here the results for equal configurations are almost the same.

Discussion I will only include the first trial of each configuration.



Observation The accuracy for $p_F = 1.0$ starts at slightly better than chance, then goes down to as good as chance.

Discussion Ending at as-good-as-chance is not surprising. $p_F = 1.0$ means that the training data are random. With random training data it is impossible to achieve better-than-chance accuracy. The start at a higher accuracy probably has to do with 'I don't OQ know' responses. I don't want to take time now to understand it completely. And I will leave the training curve for $p_F = 1.0$ out of the further discussion.

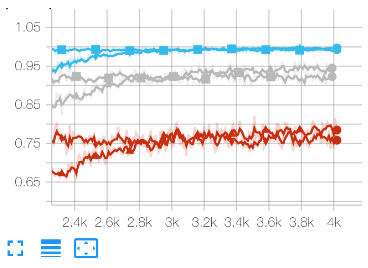


Matches (4)

★accuracy/teacher (iterate/0.3/0/0)
▲accuracy/teacher (iterate/0.1/0/0)
▲accuracy/teacher (iterate/0.01/0/0)
✚accuracy/teacher (iterate/0.0/0/0)

Observation The learning curves for all p_F, including $p_F = 0$ rise at about the same time. Only the invisible ceiling that they hit differs.

Discussion This observations appears rather uninteresting. But at least it shows that in this configuration the injected failures only influence the attainable accuracy and don't destabilize the training.



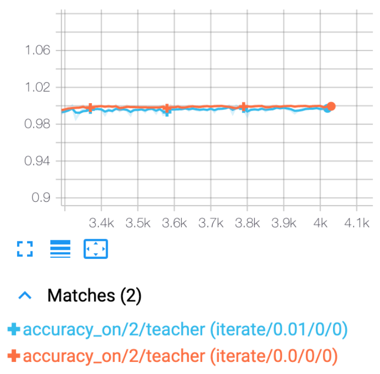
Matches (6)

▲accuracy_on/2/teacher (iterate/0.3/0/0)
◆ accuracy_on/2/teacher (iterate/0.1/0/0)
◆ accuracy_on/2/teacher (iterate/0.01/0/0)
◆ accuracy_on/2/targets (iterate/0.3/0/0)
■accuracy_on/2/targets (iterate/0.1/0/0)
■accuracy_on/2/targets (iterate/0.01/0/0)

Observation The accuracy of $\text{Amplify}^{H'}(X_{\text{PA}})$ on depth-2 questions matches the theoretically calculated accuracy. The accuracy of X_{PA} surpasses it in all remaining settings of p_F.

Discussion The former observation is not surprising, since for depth-2 questions H' receives correct inputs (sub-answers to primitve questions) and only its outputs get laced with randomness. There is nothing between the mechanism that injects randomness and the accuracy measurement. Therefore the measured accuracy must match what one would theoretically calculate, given the error injection mechanism.

The latter observation, however, is striking. (Perhaps the only worthwhile fact in this report.) X_{PA} achieves a higher accuracy than $\text{Amplify}^{H'}(X_{\text{PA}})$, from which its training data come. This supports my hypothesis that regularization in the distillation step can compensate for some of the failures introduced in the amplification. (What regularization? SupAmp uses Dropout, batch normalization and layer normalization (Christiano et al. 2018). Polyak averaging as a quasi-ensemble method (Brownlee 2019) probably acts as a regularizer as well.)



Observation With $p_F = 0.01$, the final accuracy of X_{PA} on depth-2 questions is 0.9972 (according to TensorBoard, with the smoothing parameter set to 0.6). This is lower than 0.9997 when $p_F = 0$.

Discussion This speaks against my main hypothesis. – If there was a failure tolerance, how low can it be? – The reason might be that even the tiny SupAmp model has too great capacity for this easy task, or, relatedly, that we need more regularization.

The hypothesis that more regularization leads to a greater failure tolerance, can be tested. For example, one could increase the dropout probability (currently 0.1) and see TODO
if the accuracy rises back to $p_F = 0$ levels. Of course, the higher p_F, the harder it will be to find a dropout rate that regularizes away faulty inputs while still allowing learning progress.

4 Points to improve

4.1 Length of training

I noted above that not all learners converged within the 4000 steps that each trial ran for. In order to ensure convergence and avoid useless results, I could increase the number of training steps indiscriminately. Better would be to have the weights saved at the TODO
end of training, or even at regular checkpoints, so that not-yet-converged trials can be continued. This will be vital if I move the training to AWS Spot instances, which I'm considering.

4.2 Performance

Much computation is wasted: Measuring code is run frequently, often at every batch. And a large part of it runs in Python or NumPy space, rather than more optimized TensorFlow space. H' reaches perfect accuracy within the time it takes X to complete 1000 training steps, but keeps getting trained at full speed. X reaches perfect accuracy on each successive question depth, yet questions of all depths continue being sampled uniformly. (The latter might be different when curriculum learning is turned on.) One should be able to reduce training time by concentrating ressources where they're needed.

References

Brownlee, Jason (2019). *How to Calculate an Ensemble of Neural Network Model Weights in Keras (Polyak Averaging)*. URL: https://machinelearningmastery.com/polyak-neural-network-model-weight-ensemble/ (visited on 2020-01-22).

Christiano, Paul, Buck Shlegeris and Dario Amodei (2018). 'Supervising strong learners by amplifying weak experts'. In: arXiv: 1810.08575 [cs.LG].