Geosci. Model Dev. Discuss.,
https://doi.org/10.5194/gmd-2019-327-RC1, 2020

Interactive comment on "RadNet 1.0: Exploring deep learning architectures for longwave radiative transfer" *by* Ying Liu et al.

Anonymous Referee #1

Received and published: 19 February 2020

Summary

In this paper the authors use a neural network to emulate a radiation scheme used in GCMs. They use reanalysis states to train networks with several architectures and find that the radiative heating rates were accurately reproduced. They then couple the emulator to a single-column simulation and find that the climate of the neural net model differs in some ways from the reference climate.

This paper represents a valuable contribution to the growing literature of machine learning parameterizations and should be of interest to a range of potential readers. The paper is well written and the analysis is thorough. I do have a few comments about the methodology and interpretation of the results though which are listed below.

General comments

1. Study setup You have chosen to use ERA profiles as input which of course gives you realistic profiles. However, this comes with two major limitations: a) You have to manually create out-of-sample profiles (Dataset 2) which are potentially unrealistic (see comment below). b) You are limited to single-column model experiments for coupled verification. While I don't expect you to do this for this revision, I feel like you have the chance for a more insightful experimental setup using CliMT. You could have setup a global CliMT simulation that is somewhat realistic and used the RRTMG output from that simulation as training data. Then you could have also ran the same experiment with increased SSTs for a more realistic climate change experiment. Lastly, you could have used your RadNet in directly in the global simulation for a 3D coupled experiment. I think the results from such an experiment could be really insightful for the ML parameterization community.

2. Training data estimation for line-by-line computation As you mentioned in the paper, ideally one would use line-by-line radiation computations to train from. Since line-by-line is really expensive you would be limited in the amount of training data you can generate. Would it be possible to make an estimate of what a realistic amount of line-by-line data would be and whether this amount of data would be enough to train your networks

3. Neural network architectures One of the key goals of this paper is to compare different NN architectures, particularly fully connected versus convolutional. You present 5+ different architectures. I have several questions about these:

3.1 Model E is the same as Model C but with zero-padding but why do the number of channels also increase in the table from 4 to 6? I assume this is an error since the number of parameters also stays the same.

3.2 You could have tried a fully convolutional network by using zero-padding on each layer. That way you don't need the final fully connected layer, which means a lot fewer parameters. I would be interested to see how such a network would perform.

3.3 However, one problem I see with CNNs in this context is that they are based on translational invariance i.e. the computations are the same along the entirety of the vector. But with non-uniform grids like the one you are using is there any reason to assume that translation invariance should hold?

3.4 Another architecture you could try is a Resnet using skip connections.

3.5 You mention you use PReLUs to avoid vanishing gradients. Did you observe vanishing gradients for normal ReLUs? That would surprise me since your networks are not very deep.

3.6 Did you observe overfitting (train MSE < valid MSE)? Your validation dataset is from 2015-2016 while your train dataset is from 1979-1985, which means that the base state is probably changing. This will make it hard to check for overfitting since even with a model that does not overfit one would expect the train loss to be lower. A better test of overfitting would be to take a random subset of the train dataset. The reason I am interested in this is that I am surprised a bigger network (B vs A) did not give you a better score. If you are not yet overfitting, bigger networks should always result in better losses, or am I missing something? You mention overfitting when talking about the generalization experiments but that still doesn't explain why the skill for Dataset 1 wouldn't be better for model B.

——-

Specific comments

Line 56: You could also mention this recent paper: https://arxiv.org/abs/2001.03151. Also this follow up paper highlights some interesting issues: https://onlinelibrary.wiley.com/doi/abs/10.1029/2019MS001711

Line 90: Typo "colummn"

Line 102: I think it would be useful at this point to mention the horizontal and vertical resolution of ERA.

Line 105: So the size of your training dataset is around 12 million? Please mention this explicitly?

Line 110: If I understand correctly there is no vertical correlation in your perturbation, right? Are the resulting profiles therefore "unrealistic"? How do you think this affects the network's generalization? Do you think that your results are representative of the sorts of perturbations you would experience in real climate change (increased T)?

Line 150: Actually, convolutions are a form of regularization by introducing an architectural constraint. In fact, a convolution layer is simply a subset of a fully connected layer with shared weights (https://medium.com/impactai/cnns-from-different-viewpoints-fab7f52d159c). But yeah, in effect you end up focusing on local features.

Table 1: I think it would be helpful to list all the Models (including F) in the table.

Data: Great that you included an example file in the repo. But maybe you could also add instructions on how to download the full dataset you used.

Code: Kudos for including code. I looked at the implementation of the networks. Maybe just a recommendation: I think the code could have been a lot easier and clearer if you used Keras instead of plain TF. Especially with TF 2 its now very easy to take all the precoded parts of Keras and implement your own layers, etc. using TF. All the manually coded layers in your code (matmul, etc.) make it very hard to follow when a simple Dense() would suffice. This would make the code easier to read for others.
