NN RADIATION SOLVER IN RAPS

Jakob Progsch, July 2nd 2018
GOAL: MAKE NN MODELS RUN IN RAPS

Five simple steps

1. Find a way to run the NN models: Tensorflow
2. Come up with a way to call them from FORTRAN: Glue code library
3. Insert the calls in the right places in RAPS
4. Train a NN that can replace the radiation solver
5. Run it
RESULTS
ML training is magic!
RESULTS?

Well, I mean it's close enough, right?
RESULTS??
Uhm...?

Venus level weather?
RESULTS???
Evidently not
WHAT THE NN ACTUALLY DOES

array inputs
- Pressure
- Pressure gradient
- Temperature
- q_cloud_fraction
- q_rain
- q_snow
- q_vmr
- latitude
- aerosol_1
- skin_temperature
- albedo
- cos_solar_zenith

scalar inputs
- ...

array outputs
- flux_up
- flux_up_clear
- flux_up_sh
- flux_up_sh_clear
- flux_up_sw
- flux_up_sw_clear
- flux_down
- flux_down_clear
- flux_down_sh
- flux_down_sh_clear
- flux_down_sw
- flux_down_sw_clear
- direct

138 layers

Predict
WHAT THE NN ACTUALLY DOES

Example

Predict
THE CYCLE OF ML

RAPS

Import Model

Training Data

Neuronal Network

ipython Notebook

Train and Export
CHALLENGES

FORTRAN

• Machine learning frameworks are not written in FORTRAN.

• We can't simply link Tensorflow against RAPS since it uses its own tensor data types etc.

• We also want an abstracted simple interface in case we want to use something different than Tensorflow at some point

Solution:

• Glue code wrapping the required Tensorflow code in C calls we can then link from Fortran
CHALLENGES

Data Consistency

Simulation and training data need to match in:

- Units
- Layout (row vs column major etc.)
- Pre- and postprocessing of derived data

Solution:

- Extract the training data from the exact location in the code that the network will be used in later
- Use a dummy network that only forwards the inputs to verify that layouts etc. are consistent throughout the chain
CHALLENGES

Numerics

- Subtle numeric issues introduced by the NN might cause ripple effects throughout the simulation and produce issues such as:
  - Violating physical constraints (conserved quantities, non-negativity etc.)
  - Violating assumptions by other pieces of code about the output of the replaced routine

Possible solutions:

- Lots and lots of "data debugging"
- Enforce hard constraints after the network
- Include constraints in the loss function
- Design the network such that constraints are inherently fulfilled
AN EXAMPLE
The story of the great SIGFPE crisis

• Despite network outputs looking plausible, using them in RAPS would always lead to floating point exceptions in other parts of the code

• We thoroughly checked all the interface code to make sure there are no uninitialized, out of bounds or transposed variables

• Staring at the data revealed some "blown up" profiles (see Venus weather from before) as a result of strong outliers that were not represented enough in the training data

• Enforcing positivity and monotonicity constraints after the network didn't fix the problem entirely but had the simulation "survive" more iterations

• Smoothing the output allowed the simulation to not crash (YAY!) but it finishes with very high error relative to reference (BOO!)
CHALLENGES

Network Design

• Networks have to be powerful enough to capture the behavior but should also be small enough to avoid overfitting and slow performance

• For localized phenomena on grids (1 dimensional here) convolutional networks seem like a natural fit

• Problem: convolutional layers are good at detecting local phenomena but are bad at expressing long range interactions

• Possible solutions:
  • Use a diabolo shaped network
  • Predict gradients (or related even nicer quantities) instead of the flux values
• Purely convolutional networks are good at detecting local phenomena

• But they also have difficulties expressing long range interactions caused by them
• Down- and upsampling can provide the "reach" but also smears out fine details

• Introducing shortcut connections between the down and upsampling stages of the network can preserve the finer details

• Similar structure to a Multigrid solver?
PREDICT SOMETHING EASIER

Gradients!

• Instead of predicting the values that require producing step functions etc. predict the gradient

• Local phenomena stay local

• Conveniently allows enforcement of some constraints
  • Monotonicity turns into positivity
  • Known starting values are easily accounted for

• Downside: any small bias/error will offset the rest of the curve
CONCLUSIONS

- Interfacing machine learning frameworks with existing code can be messy
  - What would be a straightforward bug in a traditional application often manifests as a numerical problem in an NN application
- Training a network to some degree of good looking convergence is easy
  - That doesn't mean it will actually work in the context of a simulation
- Enforcing hard constraints is tricky but necessary. However, we can't expect the network to learn them exactly just from the data
  - Guaranteeing hard constraints might require new approaches besides external enforcement or loss functions alone