Debugging and Visualization

11-785 / Spring 2019 / Recitation 4

Raphaël Olivier, Sarveshwaran Dhanasekar

Recap

Now you know :

- What Neural Networks are and what they do (lectures week 1)
- How to train networks(lectures week 2-4)
- How Pytorch helps you to define and train nets (rec 2)
- How to use Pytorch to simultaneously load data, build networks and train them efficiently (rec 3)

You have tried to use that knowledge in HW1P2.

It’s harder than recitations make you think.

Debugging deep learning

In Computer Science, debugging is always a big, painful part of the work.

In Deep Learning it's even bigger and more painful. Very often you :

- Have implemented a sweet model
- The code looks fine
- Accuracy is terrible/you get a weird error
- You have no idea why

Debugging deep learning

The reason DL debugging is especially hard is that there's a large number of phenomena that can make your code fail :

- Python-based error
- Pytorch-based error
- "Math" error (wrong minus somewhere)
- Modelization issue
- Training issue
- Testing issue
- ....

The hardest thing is to determine in which situation you are.

Plan for today

Today we'll cover these cases and what you can do about them (both **prevention** and **debugging**)

- General tips to **organize your code**
- Usual **Model-related errors** and how to find them
- Use metrics/hyperparams **visualization** to help you !

Apply these **before** coming to Office Hours !

General coding tips

Make your code modular

Design functions/classes for each of the main subtasks (data loading, model definition, model training,...).

Have these functions in separate files, and use a central script file that you call once.

Do not use a notebook, except for the script file.

General coding tips

Centralize your hyperparameters

Instead of hardcoding the hyperparameters you use (learning rate, nepochs, layer size, dropout) in the different files, write a configuration module for that (file or command line)

This may contain boolean flags too (use_cuda, test_only, etc).

General coding tips

Start small

Use simple models/training routines at first, with few configuration options. When things seem to work, increase complexity.

Implement a **sanity check**

General coding tips

Let’s look at an example.

Debugging

Coding error : when your model does not do what you want it to do (python, pytorch, math)

Training error : when your model does what you want it to do but is not learning well

Testing/decoding error : when your model is learning well but outputs bad results
(this one should be rare in HW1 but very usual when dealing with language in HW3/HW4. We won't talk too much about it.)

You should check for coding errors first, then training errors.

Coding errors

Signs you may have one :

- Loss does not decrease at all
- Outputs are constants
- Training stops mid-time for unclear reasons

Coding errors

How to find them : **Print everything** to look for the first moment the problem appears. Be methodical

What to check :

- Your **data** : Not iterating ? Instance-label misalignment ?
- Your **shapes** : everything consistent ?
- Your **hyperparameters** : when you print them, are they what they're supposed to be ?
- Your **parameters** : are they changing during training ? Are they going to 0 ? (they are in my_net.parameters())
- **Simpler cases** (ex : with batch size 1 does it work ?)

It's good to have a **sanity check** for that. Don't use cuda/AWS.

Time issues

Specific case of coding error : when things work but are too slow.

In your epochs, use the **time** module to check the duration of all your subtasks (data loading, forward, backward,...), and find the aberrant one.

(In HW1P2, one epoch should last ~5 minutes with 4-5 layers of sizes ~1024)

Training errors

For those errors, usually your loss does decrease, but not enough. If you see absurdly low performance (~random) it's probably a coding error.

Note : a random classification model would have a cross-entropy loss of ~log(Number_of_classes).

Training errors

Different problems :

Modelization issues : your model is too small to learn patterns (or not well designed when the problem is complex)

Optimization issues : you cannot train your model properly

Overfitting : your model is too big/you train too long

Modelization/optimization issues

Sign that you have one : the **training** loss does not go down well enough

To make sure :
A good model *will* overfit if you train it too long → you can use that to debug
Train your model on a small subset of your data for many epochs : the training loss should go to 0. If it doesn't, you have a problem.

Modelization issues

Is your model too weak ?

You should refer to literature/your experience to know that.
(In HW1P2 : 3 layers of size 512 are enough for 55% accuracy (with context))

If the model shouldn’t be weak, look for an **optimization issue**.

Optimization issues

You should look for :

- Learning rate: if too small you will learn too slowly. If too large you will learn for a while then diverge.
  Default “good” : 0.001.
  It is recommended to do **learning rate decay** : start large, then decrease (for example when loss stops improving)
- Optimizer (default “good” : Adam)
- Initialization (default “good” : xavier)
- Batching (just the batch size on simple problems). Default “good” : from 32 to 128 if you can afford it.

Too deep models can create optimization problems too (vanishing gradients). They also lead to...

Overfitting issues

Overfitting symptom : **Training loss decreases but validation loss doesn't**.

You should ***always*** have a small validation set to look at every epoch.

Things to do there :

- Verify that you **shuffle your training data**
- Decrease your model size/depth
- Use some of the tricks you know that help generalization : **dropout, batchnorm, early stopping**, validation-driven **rate decay**

Note : adaptative optimizers (Adam,...) overfit more.

Overfitting issues

It's also possible to **overfit on the validation set**.
This happens when you try a very large amount of architectures/hyperparameters with the same validation set : you may find one that works "by chance" and won't generalize.

(In HW1P2 : if you do 200 attempts a day on kaggle, you may overfit on the public leaderboard and be disappointed by your results on the private leaderboard).

If you plan to look for many architectures, consider a better validation method like K-fold.

Testing/decoding issues

When your model learns, training *and* validation loss decrease, but accuracy is low.

Recall that losses (ex:cross-entropy) are differentiable surrogates for the metric you want (ex:accuracy). It's always possible to have a gap between the two.

On a simple classification problem like HW1P2 this shouldn't happen too much (unless bug in the prediction function). However, to be safe you should look at your validation accuracy along with your loss.

Visualize your metrics

We repeated many times that you should look at your metrics, compare training/validation loss,etc.

But, just printing them in the terminal is dirty and hard to read.

That's why you should **visualize them**

→ Second part of this recitation

Installation of TensorBoard(X)

- TensorFlow Users
  - TensorBoard comes pre-installed with TensorFlow when you use the conda installation
  - If you installed TensorFlow with pip, then you can install TensorBoard

```
pip install -U pip
pip install tensorboard
```

- PyTorch Users
  - TensorFlow has come out with the TensorBoardX package for PyTorch users
  - You can install the package using pip as follows

```
pip install tensorboardX
```

Need for Visualization

- Answers the question “Why am I learning?”
- To see how your weight matrix and gradients change over time during training of your model, which can help determine whether you need to
  - Remove extra layers when there is a redundancy in matrices
  - Add new layers to see if they learn something unique
- To predict the right time to stop training the model
  - It’s better to use tools to predict when to stop rather than logging loss and accuracies at each step of training

Need for Visualization

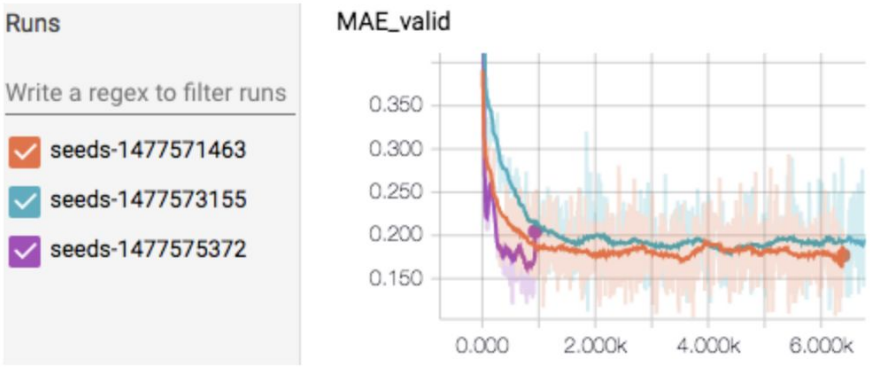
- Weight Initialization
  - To better understand which weight initialization method performs better for the given problem
  - We get to see why initializing with **zeroes** is not preferred
- How well are the Activation functions performing?
- Is the Dropout rate too high?
- In general, Visualization helps to fine tune the network for better or optimal performance

TensorBoard

- A Visual Logger
- To better understand, debug and optimize the problem at hand
- Among many, tf.summary()
  - It’s a public API available for use in multiple deep learning frameworks
  - Permits the logging of data to user defined directories
  - Allows logging of operands (similar to nodes in the TF data flow graph)
- Support for logging scalars, images, figures, histograms, audios, text, graphs and video summaries

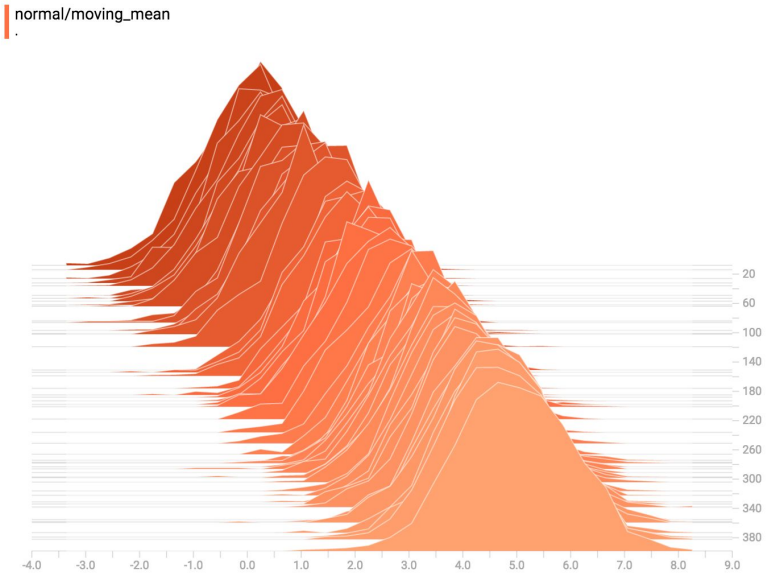
TensorBoard

- “scalars” tf.summary.scalar()
  - As the name suggests, plots 1-D operands on a 2-D space where x-axis is time (num epochs/steps) and y-axis is the operand value
  - Used to keep track of scalars like “loss”, “prediction_accuracy”, “learning_rate”, etc..



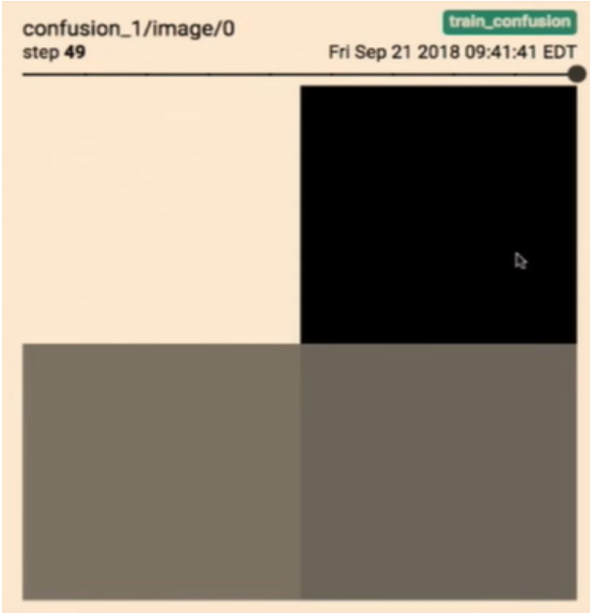
TensorBoard

- “histograms” tf.summary.histogram()
  - In general, all weights and biases are 2-D matrices which are then easily visualized using a histogram in 3-D space
  - In this case, the z-axis corresponds to the time/num-epochs, x-axis is operand value and y-axis is the frequency of the operand value



TensorBoard

- “images” tf.summary.images()
  - Generally used to log training images and and visualize which image is harder for the model to learn by looking at the loss function
  - Also used to plot weight matrices, CNN kernels or filters as a heat map and sometimes even a confusion matrix.
  - It normalizes the values provided between [0-255]



Training on AWS?

- TensorBoard should be pre-installed on both `tensorflow_p36` and `pytorch_p36` environments
- You however might want to install TensorBoardX into your PyTorch environment
- Use the Local Port Forwarding argument (-L) when you "**ssh**" into your instance
- Default port for TensorBoard(X) is 6006
- `ssh -i key.pem -L your_machine_port:127.0.0.1:6006 ubuntu@ec2-xyz.amazonaws.com`

References

- Graham Neubig, slides of *11-747 Neural Networks for NLP* about debugging neural nets, Spring 2018