

Debugging and Visualization

11-785 / Fall 2019 / Recitation 4

Liwei Cai, Natnael Daba

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
- Have no idea why

Debugging deep learning

The reason DL debugging is especially hard is that there are many things 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,...). You will find compartmentalizing your code to be useful especially for upcoming homeworks.

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**

Let's look at an example.

Debugging

Coding mistakes : when your model does not do what you want it to do (python, pytorch, math or logic).

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

Testing/decoding mistakes : when your model is learning well but outputs bad results.

(this one should be rare in HW1 but very usual when dealing with Language models in HW3/HW4. We won't talk too much about it in this recitation.)

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 mistakes

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 ? (spend a good amount of time checking if your data is sane. This is crucial especially if you're doing some preprocessing on your data before passing it to your network.)
- Your **shapes** : everything consistent ?
- Your **hyperparameters** : when you print them, are they what they're supposed to be ?

Coding mistakes

Some typical examples of coding errors are:

- You forgot to put your model in eval mode during inference time and your model is now producing garbage.
- You passed softmax outputs to a loss that expects raw logits. E.g. passing softmax outputs to `nn.CrossEntropyLoss()` while the documentation specifies:

“The input is expected to contain raw, unnormalized scores for each class”

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.

Training mistakes

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

Note: a random classification model would have a cross-entropy loss of $\sim \log(\text{Number_of_classes})$.

Training mistakes

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

One common mistake that most people make is forgetting to overfit a single batch (or any random subset of your training data) first.

This is important to check if your model is capable of doing anything at all and also fix any bugs in your model quickly.

Read more about this from here: <http://karpathy.github.io/2019/04/25/recipe/> an excellent blog on tips and tricks to train a neural net by Andrej Karpathy.

Modelization/optimization issues

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

A good model (a simple but big enough model with no bells and whistles like regularizations) will overfit a small random subset of your training set (e.g. a single batch) if you train it too long → you can use that to debug.

The training loss should go to 0. If it doesn't, you have a problem.

Once your training loss decreases to 0, you can slowly add regularizations and train it on the entire training set and start fine tuning.

Optimization issues

You should check:

- **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 256 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 **learning 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 (e.g. cross-entropy) are differentiable surrogates for the metric you want (e.g. 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 there is a bug in the prediction/inference code).

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

Why visualize?

- Answers the question “What 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
- 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

Tensorboard

TensorBoard is a visualization library for TensorFlow that is useful in understanding training runs, tensors, and graphs.

There have been 3rd-party ports such as tensorboardX but no official support until now.

In PyTorch 1.1.0, TensorBoard was experimentally supported in PyTorch, and with PyTorch 1.2.0, it is no longer experimental - you can simply type

```
from torch.utils.tensorboard import SummaryWriter
```

to get started.

Why visualize?

- 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

Installing Tensorboard

However, if your PyTorch version is below 1.2.0, the import statement might not work because tensorboard was experimental for versions below 1.2.0. Therefore, check your PyTorch version first using:

```
import torch
print(torch.__version__)
```

If your PyTorch version is $\geq 1.2.0$, then you're all set. If your PyTorch version is $< 1.2.0$, then follow this blog: <https://www.endtoend.ai/blog/pytorch-tensorboard/> to upgrade PyTorch and install tensorboard.

Starting Tensorboard

Before starting tensorboard, we have to first do local port forwarding. Remember the `-L` flag you used in your ssh command in Recitation 1?

If your ssh command that you use to connect to AWS was something like this:

```
ssh -i KeyTest.pem -L 8000:localhost:8888  
ubuntu@ec2-34-227-222-100.compute-1.amazonaws.com
```

Then now you just need to add an additional `-L` flag forwarding Tensorboard's port like this:

```
ssh -i KeyTest.pem -L 8000:localhost:8888 -L 6007:localhost:6006  
ubuntu@ec2-34-227-222-100.compute-1.amazonaws.com
```

Starting Tensorboard

To start Tensorboard, run the following command from your terminal in AWS (don't forget to activate your `pytorch_p36` virtual environment first):

```
tensorboard --logdir=./runs
```

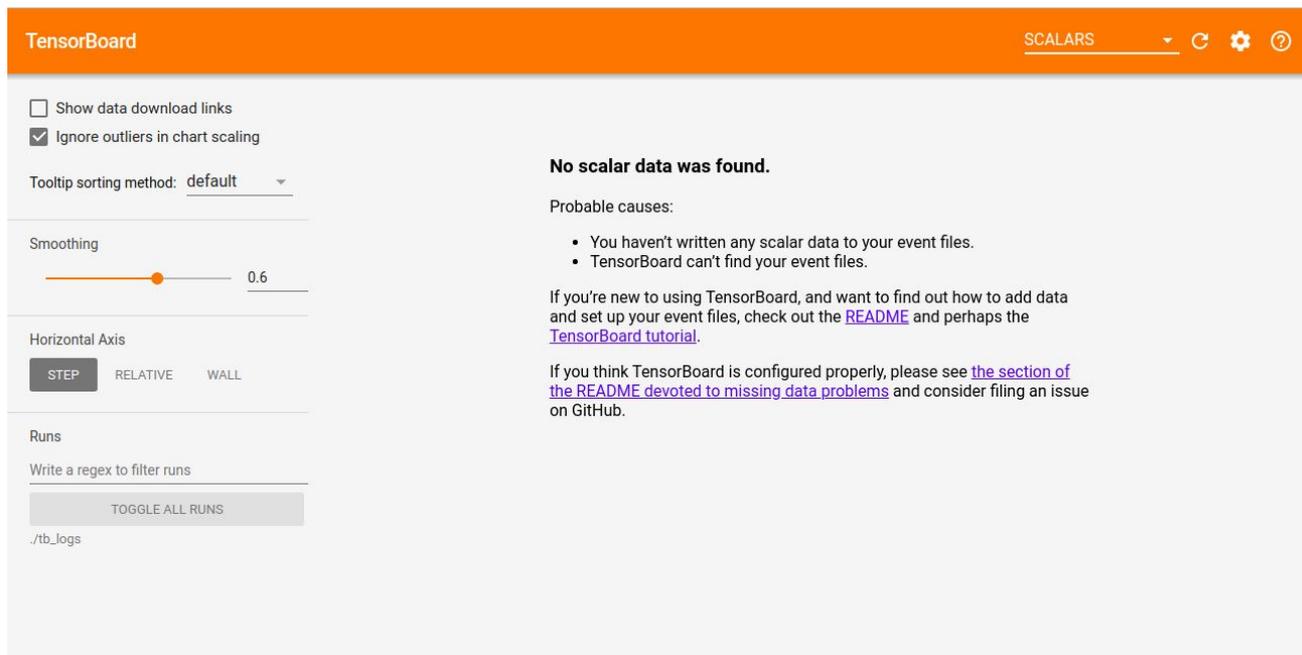
Where `./runs` is the directory for Tensorboard to search for the event files.

Next, type the following link in your local browser to view Tensorboard's main dashboard:

```
localhost:6007
```

Starting TensorBoard

If you correctly follow the instructions in the previous two slides, you should be seeing something like this in your browser:



The screenshot shows the TensorBoard web interface. At the top, there is an orange header with the text "TensorBoard" on the left and "SCALARS" with a dropdown arrow, a refresh icon, a settings gear icon, and a help question mark icon on the right. Below the header, the interface is divided into two main sections. On the left, there are several configuration options: a checkbox for "Show data download links" (unchecked), a checked checkbox for "Ignore outliers in chart scaling", a "Tooltip sorting method" dropdown menu set to "default", a "Smoothing" slider set to 0.6, and a "Horizontal Axis" section with three buttons: "STEP" (selected), "RELATIVE", and "WALL". Below these is a "Runs" section with a text input field for filtering runs and a "TOGGLE ALL RUNS" button. At the bottom left, the path "/tb_logs" is visible. The right section of the interface displays a message: "No scalar data was found." followed by "Probable causes:" and a bulleted list: "You haven't written any scalar data to your event files." and "TensorBoard can't find your event files." Below this, there is a paragraph of text: "If you're new to using TensorBoard, and want to find out how to add data and set up your event files, check out the [README](#) and perhaps the [TensorBoard tutorial](#)." and another paragraph: "If you think TensorBoard is configured properly, please see [the section of the README devoted to missing data problems](#) and consider filing an issue on GitHub."

Testing Tensorboard

Run the following python script to log random data to the directory you specified when you started tensorboard:

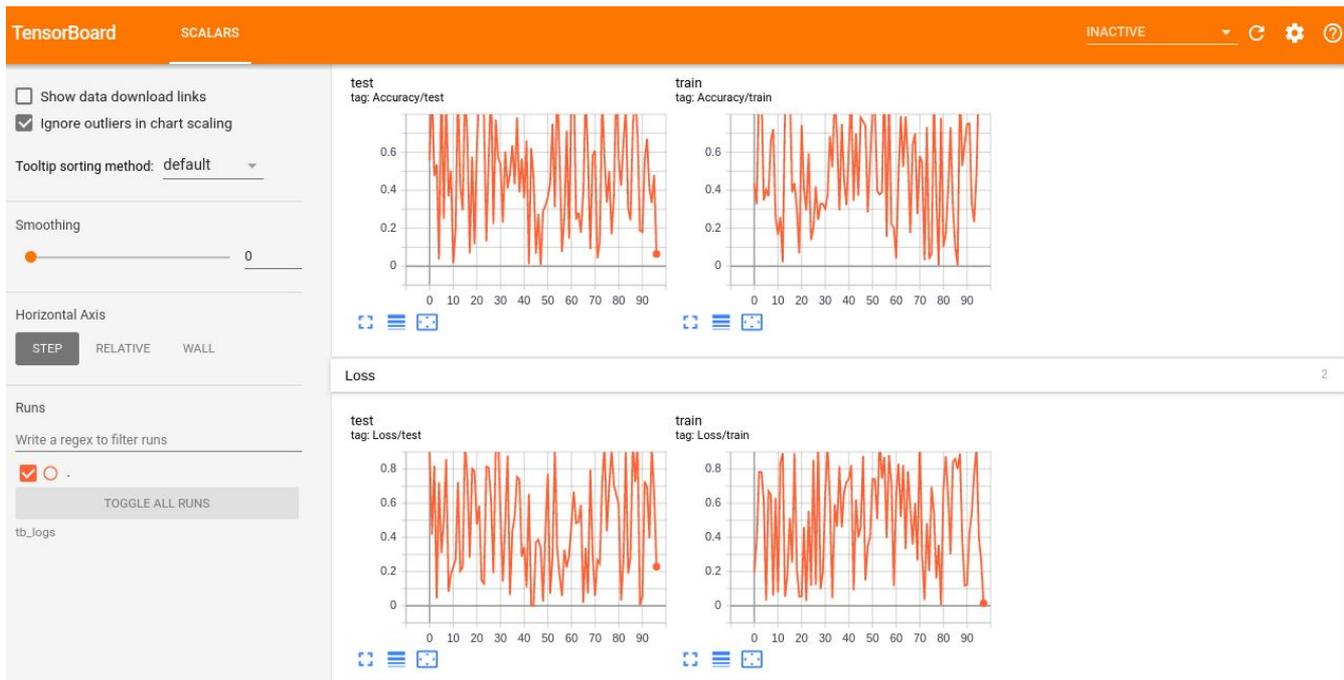
```
from torch.utils.tensorboard import SummaryWriter
import numpy as np

writer = SummaryWriter("./runs/test")

for n_iter in range(100):
    writer.add_scalar('Loss/train', np.random.random(), n_iter)
    writer.add_scalar('Loss/test', np.random.random(), n_iter)
    writer.add_scalar('Accuracy/train', np.random.random(), n_iter)
    writer.add_scalar('Accuracy/test', np.random.random(), n_iter)
```

Testing Tensorboard

Again, if everything is working properly then this is what you should be seeing in your dashboard:



Create SummaryWriter

- A SummaryWriter writes all values we want to visualize to event files in a given directory.
- You should use different run directories (“example”) in a common root directory (“./runs”) for different runs of your model.

```
from torch.utils.tensorboard import SummaryWriter  
writer = SummaryWriter("./runs/example")
```

Add values

All methods of SummaryWriter take 3 parameters: tag, value, step.

- Step is an increasing integer, usually the number of iterations, that serves as the x-axis value of the plot.

Each method appends a data point to a type of plot. Common methods are:

- `add_scalar`: line plot of one variable (value is a scalar, i.e. a float number).
- `add_scalars`: line plot of multiple variables (value is a dict of scalars).
- `add_histogram`: histogram of distributions of values (value is a tensor). Useful for understanding the dynamics of the network.

You can even add fancier values, like “`add_image`”, “`add_audio`”.

Example

Also included in
DataVisualization.ipynb

Notice the usage of
add_scalars and
add_histogram



```
from tqdm import tqdm

def train(model, writer, epochs=1000):
    criterion = nn.BCEWithLogitsLoss()
    optimizer = torch.optim.Adam(model.parameters())
    for epoch in tqdm(range(epochs)):
        model.zero_grad()
        out = model(X_train).flatten()
        loss = criterion(out, y_train.float())
        train_loss = loss.item()
        train_acc = ((out > 0) == y_train).float().mean().item()

        loss.backward()
        # Plot histogram of gradient of all parameters
        for name, param in model.named_parameters():
            writer.add_histogram('grad_' + name, param.grad.data, epoch)
            optimizer.step()

        with torch.no_grad():
            out = model(X_val).flatten()
            val_loss = criterion(out, y_val.float()).item()
            val_acc = ((out > 0) == y_val).float().mean().item()
        # Plot loss and accuracy on train and val
        writer.add_scalars('loss', {'train': train_loss, 'val': val_loss}, epoch)
        writer.add_scalars('acc', {'train': train_acc, 'val': val_acc}, epoch)
```

TensorBoard UI

Select type of values to display

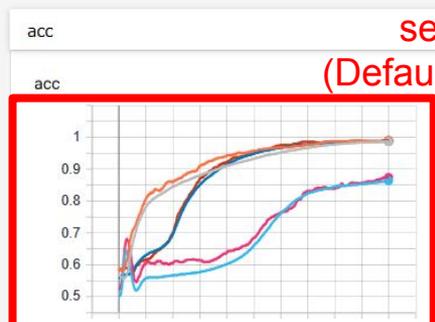
Manual refresh & set default refresh rate (Default is 30s, pretty slow)

Show data download links
 Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing  0.6

Horizontal Axis **STEP** RELATIVE



Filter runs to display

Runs

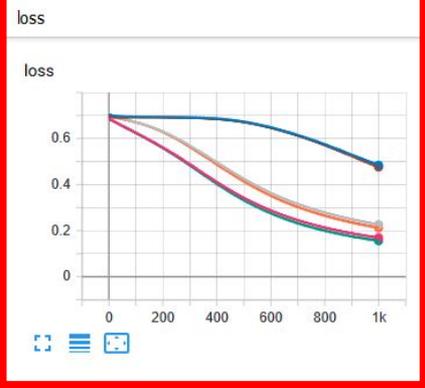
Write a regex to filter runs

- sigmoid
- sigmoid/loss_train
- sigmoid/loss_val
- sigmoid/acc_train
- sigmoid/acc_val
- tanh
- tanh/loss_train
- tanh/loss_val
- tanh/acc_train
- tanh/acc_val
- relu
- relu/loss_train
- relu/loss_val

TOGGLE ALL RUNS

./runs

Hover over the plots to see numerical details (see left)



Understanding histogram

- Plot histograms of gradients of network parameters.
- The gradient vanishing issue with sigmoid activation is obvious on the histogram (orange).

