4. Modeling of Research Data

Deep learning – Tools and Tricks

I wish I had known earlier

Peter Lippmann

13.02.2025

Scientific AI Lab, IWR, Heidelberg University

Deep learning overview

$$\min_{\theta} \ \mathbb{E}_{(x,y)\sim\mathcal{D}} \ L(y, f_\theta(x))$$

$\mathcal{D}$: data distribution, typically approximated by a set of finite set of input-target pairs $\{x_i, y_i\}$

f_θ: neural network with learnable parameters θ

L: differentiable loss function, typically minimal if $y = f_\theta(x)$

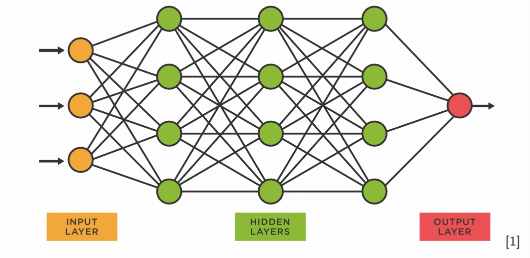
In practice:

$$\min_{\theta} \sum_{\substack{\text{sample } (x_i, y_i) \\ \text{in training set}}} L(y_i, f_\theta(x_i))$$

Optimization via (mini batch) gradient descent:

$$\theta^{(t+1)} = \theta^{(t)} \quad - \quad \alpha \sum_{\substack{\text{sample } (x_i, y_i) \\ \text{in mini batch}}} \nabla_\theta L(y_i, f_\theta(x_i))\Big)$$

The neural network



- Network types: CNNs for images, GNNs for graphs, sequence models for language, recurrent NNs for time series data, ... (often clear what to choose)
- BUT many representatives, e.g. many different CNN architectures (less clear)
- For given architecture, several hyperparameters (educated guess + trying out)

The loss function

$$\min_{\theta} \sum_{\substack{\text{sample } (x_i, y_i) \\ \text{in training set}}} L(y_i, f_\theta(x_i))$$

- heavily depends on your task, e.g.:
  - classification $\leftrightarrow$ Cross Entropy loss,
  - regression $\leftrightarrow$ MSE (a.k.a. L2-loss) $\|y_i - f_\theta(x_i)\|^2$
- Tip: for regression L1-loss $|y_i - f_\theta(x_i)|$ can be more robust to outliers
- sometimes a combination of losses is used (weighting them can be tricky)

The data

- more data is better, higher quality data is better
- visualize your data before: PCA, UMAP, t-SNE
- check your data is balanced (e.g. in instances per class)
- split your data into train, validation and test set
- standardize your data (both input and output)
- use data augmentation if possible

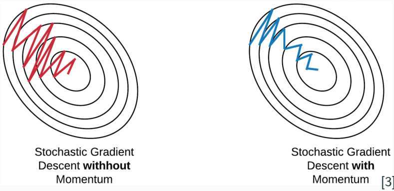
[2]

Data augmentation

The optimizer

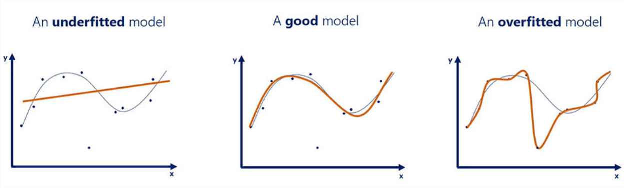
Use AdamW (Adaptive Moment Estimation + weight decay):

- adaptive learning rate helps against too small or too large gradients
- momentum stabalizes the gradient descent
- weight decay helps against overfitting



[3]

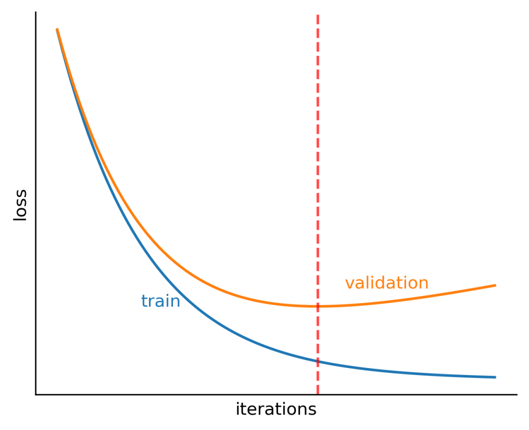
(a) Effect of momentum



[4]

(b) Overfitting model

More tricks against overfitting

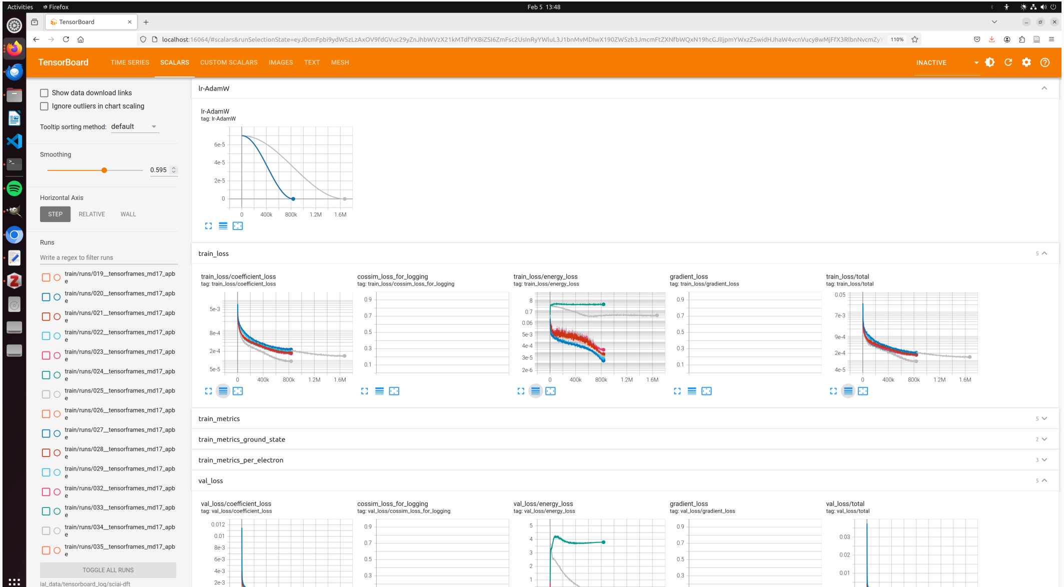


- Dropout: randomly drop/ignore neurons during training
- Save checkpoints of your model: last (if training crashes) & k-many best

More Training Tricks

- try to overfit on a single sample to debug your pipeline
- set a seed during training for reproducibility
- use enough CPU workers in dataloader to properly use GPU
- in dataloader use `shuffle=True` and `drop_last=True`
- try gradient clipping in the optimizer against instable training
- use a learning rate scheduler (e.g. cosine schedule)
- try learning rate warmup
- definitely try normalization layers: helps to standardize activations

Logging via Tensorboard



Pytorch Lightning

Pytorch lightning module combines pytorch model + optimizer + logging

- abstracts away `to("cuda")`, `loss.backward()`, `model.eval()` and much more
- makes complicated things which many people use easy, e.g. multi GPU support

```python
import lightning as L
import torch

from lightning.pytorch.demos import Transformer


class LightningTransformer(L.LightningModule):
    def __init__(self, vocab_size):
        super().__init__()
        self.model = Transformer(vocab_size=vocab_size)

    def forward(self, inputs, target):
        return self.model(inputs, target)

    def training_step(self, batch, batch_idx):
        inputs, target = batch
        output = self(inputs, target)
        loss = torch.nn.functional.nll_loss(output, target.view(-1))
        return loss

    def configure_optimizers(self):
        return torch.optim.SGD(self.model.parameters(), lr=0.1)
```

Many great tutorials at
`https://lightning.ai/docs/pytorch/stable/starter/introduction.html`

Config management with Hydra

- save on boilerplate by "programming" in configs (customize models in config not in code)

```yaml
net:
  _target_: mldft.ml.models.components.graphformer.Graphformer
  edge_mlp:
    _target_: mldft.ml.models.components.mlp.MLP
    in_channels: 128
    hidden_channels: [768, 32]
    activation_layer:
      _target_: hydra.utils.get_class
      path: torch.nn.SiLU
    dropout: 0.
  energy_mlp:
    _target_: mldft.ml.models.components.mlp.MLP
    in_channels: 768
    hidden_channels: [768, 1]
    activation_layer:
      _target_: hydra.utils.get_class
      path: torch.nn.SiLU
    dropout: 0.
    disable_dropout_last_layer: True
    disable_activation_last_layer: True
    disable_norm_last_layer: True
```

- easy to add new models, datasets, tasks and experiments
- uses OmegaConf for configuration management

Great Hydra + Lightning template at
`https://github.com/ashleve/lightning-hydra-template`

Thank You!

Any Questions?

Image References

[1] https://www.marktechpost.com/wp-content/uploads/2022/09/Screen-Shot-2022-09-23-at-10.46.58-PM.png

[2] https://media.datacamp.com/legacy/image/upload/v1669203370/Data_Augmentation_Header_f42227f2cb.png

[3] https://i.sstatic.net/epW89.jpg

[4] https://static.wixstatic.com/media/0ed3e8_a9b7d6d3dc6b4d5cbcb30c8b2fd4782b~mv2.jpg/v1/fill/w_1000,h_449,al_c,q_90,usm_0.66_1.00_0.01/0ed3e8_a9b7d6d3dc6b4d5cbcb30c8b2fd4782b~mv2.jpg

Model Metadata

Model card

- Model details
  - Architecture, parameters, citation information, license information
- Intended use
  - Use cases within the model’s scope
- Performance metrics
  - Intended performance on given data
- Training data
  - Description of training data and data distribution
- Quantitative analysis
  - Potential biases and limitations
- Ethical consideration
  - Privacy and fairness concerns, impact on society
- https://huggingface.co/spaces/huggingface/Model_Cards_Writing_Tool
- https://github.com/openai/gpt-3/blob/master/model-card.md

Where to share/publish/deploy your models

Model sharing platforms

You can make models available for others on model sharing platforms like

- Hugging face,
- OpenML,
- Kaggle.

Advantages: Public platform with version control and model cards, you can link the data into the repo, allows others to use your model for production or fine-tuning.

How to test your software that is based on ML models

Testing of non-deterministic processes

Try to make processes deterministic
For example: Use specified random seed.

Separate deterministic and non-deterministic processes and test separately
For example: Input processing can be tested separately from model prediction.

Test for output parameters and properties that remain constant
For example: Number of predictions, feature length, etc.

Include multiple valid outputs in your tests
For example: Three most probable classifications.

Robustness: A robust model is more likely to behave like a deterministic system
Make sure your model output is stable under a range of conditions.

Accuracy: The model accuracy will affect the testing strategy
A higher accuracy leads to more consistent predictions.

Distribution: You can also test for the distribution of results rather than the results themselves

Deploying machine-learning models and software

Model deployment

In addition to making models and software available for others to use in their own code, you can also directly deploy the model - together with your code - directly so that it can be used.

Examples:

- Diffusers: google colab https://colab.research.google.com/github/huggingface/notebooks/blob/main/diffusers/stable_diffusion.ipynb
- https://lightning.ai/ for paid service and deployable models
- See https://www.freecodecamp.org/news/deploy-your-machine-learning-models-for-free/ for tutorials and services