Best Practices for Deep Learning for Science

Bethany Lusch
Asst. Computer Scientist, Argonne Leadership Computing Facility
blusch@anl.gov

www.anl.gov

Goals

- Have low error on your data
- Generalizes well to wherever you want to apply it
- Training is efficient
- Inference (applying it) is efficient
- Have good deep learning model ”soon” (limited human & computer hours)
- Didn’t need expensive dataset

Other “best practice” talks at this workshop

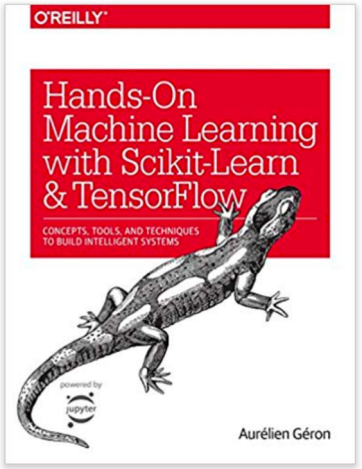
- “Deep Learning with Keras, Tensorflow, PyTorch, and Horovod on Theta” - Huihuo Zheng
- “Parallel I/O and Storage” - Kevin Harms
- “DeepHyper” - Prasanna Balaprakash and Romit Maulik
- And others…
- (I’ll try not to overlap)

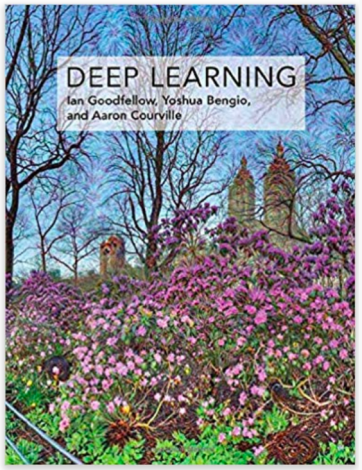
Excellent Reference

"Nuts and Bolts of Applying Deep Learning" by Andrew Ng, 1:20 video on YouTube

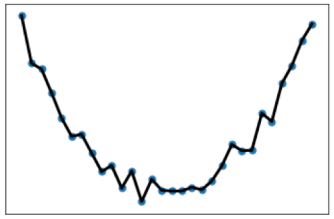
My #1 Tip

- Deep learning is hard, so save time to learn
- But don't just read, try things
- Some good resources:
  - Andrew Ng's Deep Learning sequence on Coursera
  - *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by Aurelien
  - Much more advanced/technical: *Deep Learning* by Goodfellow, et al.
    - Chapter 11 has great practical advice
    - Free at deeplearningbook.org





Generalizing Well



- Pro: neural networks can represent complicated functions
- Con: neural networks can represent complicated functions
- (due to the universal approximation theorem)

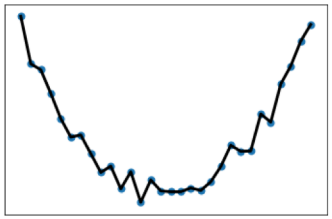
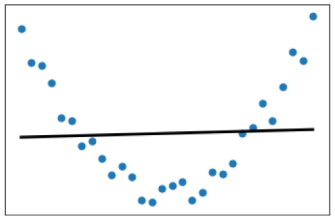
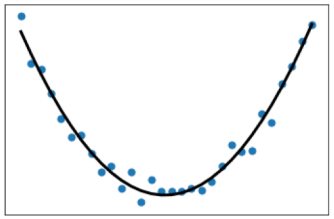
- Consequences:
  - Don’t need to be as clever about input features
  - Can handle complicated data like images
  - But then easy to overfit… (and extrapolation even harder than interpolation)

- Crucial concept: overfitting vs. underfitting (aka bias vs. variance)

Visual explanation of the universality theorem:
http://neuralnetworksanddeeplearning.com/chap4.html

Bias vs. variance

Want low training error, but also generalize well

High variance	High bias	Low bias, low variance
		
overfitting	underfitting	balanced

Neural networks can be very expressive (low bias), but easy to overfit

Picture similar to: Seema Singh, “Understanding the Bias-Variance Tradeoff”

Primary Tool: Validation Data

Non-negotiable aspect of machine learning: must do some validation (and report it in your paper)

Cross-validation typically too expensive for deep learning, so standard practice is to split your data once, at the beginning

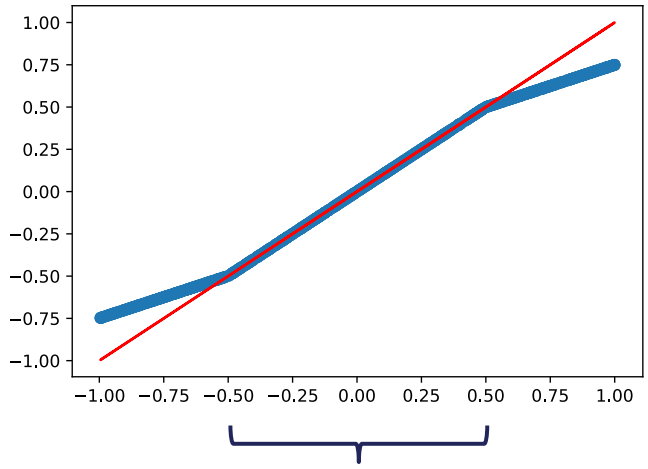
- i.e. 70% training, 20% validation, 10% test
- Use training data for the training of the NN (fitting the data)
- Use validation data to choose between versions (i.e. training multiple times, or deciding when to stop training)
- Check error on test data **at the very end** (i.e. when writing paper) to make sure you didn't overfit the validation data
- Training error unsatisfactory? Then underfitting
- Training error ok, but not validation error? Then overfitting

Validate Carefully!

- Standard: split your dataset at beginning and calculate loss on each
- But…
  - Is your dataset representative of where you want to apply the model?
  - Is your loss the same as what really matters, or just a proxy?
    - i.e. 1-step prediction error vs. multi-step prediction error
    - Might be using smoothed version of real quantity of interest
  - Does randomly splitting your data approximate how you would apply the model?
    - Example: if you want to apply model to materials that the model has never seen, then split data so that some materials are in validation data and not training data
  - Deep learning models do not extrapolate well! Interpolation is best you can do.

Deep learning models cannot extrapolate

- Easy 1-D problem (small data & simple f)
- Input from $[-.5, .5]$
- Output is just the identity: $f(x) = x$
- Trained tiny neural network
- Know perfect analytic solution



Low training & validation error
But poor outside of domain

This visual explanation of the universality theorem also gives some intuition on why you can't extrapolate:
http://neuralnetworksanddeeplearning.com/chap4.html

Aside: Hyperparameter search

- Manual tweaking
- Grid search
- Random search
- Optimized search, i.e. DeepHyper

If using automated search, still helpful to understand hyperparameters

- Need to choose search space
- Need to troubleshoot if not satisfied with search results

Personal strategy:

- Start with hyperparameters good for related problem
- Some manual tweaking
- If not successful, use deephyper

Overfitting advice

- more data (either by collecting more or using data augmentation)
- more regularization (i.e. L1, L2, dropout)
- simpler/smaller model
- early stopping
- different architecture
- reduce noise in training data (e.g. fix errors and remove outliers).

Hands-On Machine Learning with Scikit-Learn & TensorFlow by Aurélien Géron (2017)
Deep Learning by Josh Patterson; Adam Gibson (2017)
Chapter 11 of *Deep Learning* by Ian Goodfellow, Yoshua Bengio, and Aaron Courville (2016),
"Many of the recommendations in this chapter are adapted from Ng (2015)"
https://www.deeplearningbook.org/contents/guidelines.html

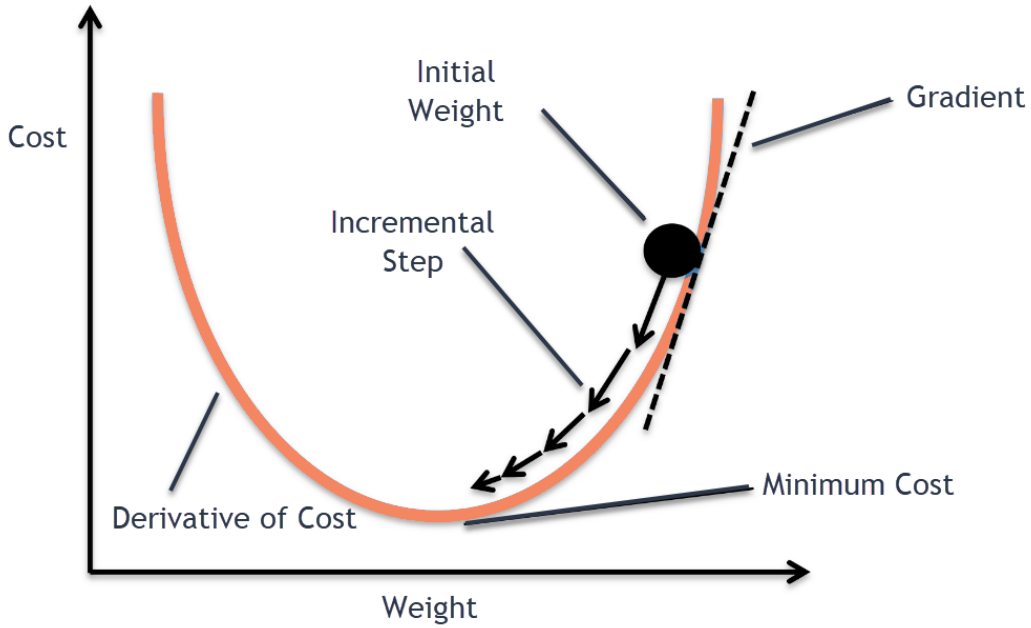
Underfitting advice

- more complex/bigger model
- reduce constraints (including reducing regularization)
- train longer
- try different optimization algorithm
- different architecture
- feature engineering

Hands-On Machine Learning with Scikit-Learn & TensorFlow by Aurélien Géron (2017)
Deep Learning by Josh Patterson; Adam Gibson (2017)
Chapter 11 of *Deep Learning* by Ian Goodfellow, Yoshua Bengio, and Aaron Courville (2016),
"Many of the recommendations in this chapter are adapted from Ng (2015)"
https://www.deeplearningbook.org/contents/guidelines.html

Learning rate

- Not clear relationship with overfitting vs. underfitting
- Andrew Ng says most important hyperparameter



Making the optimization easier

- Big gap between theory and practice (i.e. possible to fit good NN to this data, but optimization can't find it)
- Adam is popular optimization algorithm, but could help to try another
- May help to use learning rate schedule
- Initialization scheme can matter a lot because non-convex problem
  - Can run multiple times and choose best (based on val. data)
  - Other schemes might be better for your problem
- Deeper networks are more parameter-efficient, but harder to train
- Rescale each feature, i.e. [-1, 1] (easy in scikit-learn)

Constraining the network

- Many local minima: can you ”give hints”
- Example: theoretically possible to use basic NNs for images, but convolutional layers help immensely
- Lots of flexibility to add custom loss functions
  - i.e. incorporate your scientific domain knowledge

Hyperparameter Starting Points

- Batch normalization: yes, especially with CNNs and/or sigmoids
- Regularization:
  - Include mild L2 regularization immediately unless 10s of millions of examples
  - Use early stopping almost universally
  - Dropout is commonly excellent
- Layer widths: search on ≈ log2 scale, i.e. [50, 100, 200, 500, 1000, 2000]
- Learning rate: search on ≈ log10 scale, i.e. [0.1, 0.01, 0.001, 0.0001, 0.00001]. 0.001 common first try
- L2 regularization: search on ≈ log10 scale
- Batch size: search on ≈ log2 scale

Hands-On Machine Learning with Scikit-Learn & TensorFlow by Aurélien Géron (2017)
Deep Learning by Josh Patterson; Adam Gibson (2017)
Deep Learning by Ian Goodfellow, et al. (2016), which references Andrew Ng
Deep Learning with Python by François Chollet (2017)

Hyperparameter Starting Points

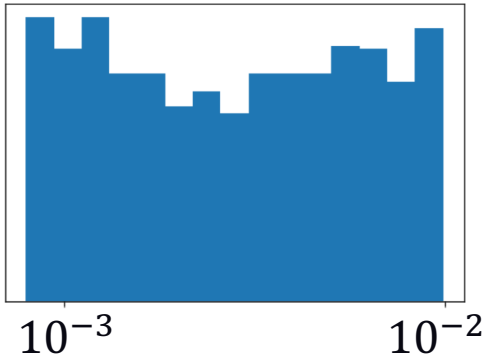
- ReLU is common, or ELU with α=1
- Optimization algorithm: usually Adam, but others default to:
  - SGD + momentum + decaying learning rate (i.e. decay linearly until fixed minimum, then decay exponentially, or decrease by factor of 2-10 when plateau)
  - Nesterov Accelerated Gradient
- "Stretch pants approach": go with a bigger network than you need, but include early stopping & plenty of regularization (Vincent Vanhoucke)
- In large CNNs: two convolutional layers stacked before every pooling layer
- For very deep nets: residual connections, batch normalization, and depthwise separable convolutions
- Initialization: Xavier/Glorot for logistic activation, He otherwise

Hands-On Machine Learning with Scikit-Learn & TensorFlow by Aurélien Géron (2017)
Deep Learning by Josh Patterson; Adam Gibson (2017)
Deep Learning by Ian Goodfellow, et al. (2016), which references Andrew Ng
Deep Learning with Python by François Chollet (2017)

Why search on log scale?

- Example: randomly choose learning rate uniformly from $[10^{-4}, 10^{-2}]$
- Will have $\approx 90\%$ on order of 10^{-2}, $\approx 9\%$ on order of 10^{-3}, $\approx 1\%$ on order of 10^{-4}
- Order of magnitude matters for learning rate, so would prefer $\approx 33\%$ on order of 10^{-2}, $\approx 33\%$ on order of 10^{-3}, $\approx 33\%$ on order of 10^{-4}
- Instead, search on log10 scale
- One way: choose exponent k uniformly, then set learning rate = 10^{-k}

Histogram of learning rates, $[10^{-4}, 10^{-2}]$



Transfer learning

Especially for images, helps to start with known good network

- Completely retrain on your data
- Just fine-tune on your data
- Fix earlier layers (“feature extraction layers”) and only retrain later layers
- Just use hyperparameters & architecture as inspiration

Yes, your data is weird

- Deep learning is moving fast and is an art
- Doesn't help that we all have unusual applications (a lot of advice is for "mainstream" problems)
- Most people are working on GPUs
- But, can balance learning from "mainstream" ML and domain-specific ML
- Fun to see connections between problems!

Fun fact: people have even used networks trained on ImageNet to help with x-ray imges

Helpful in my experience

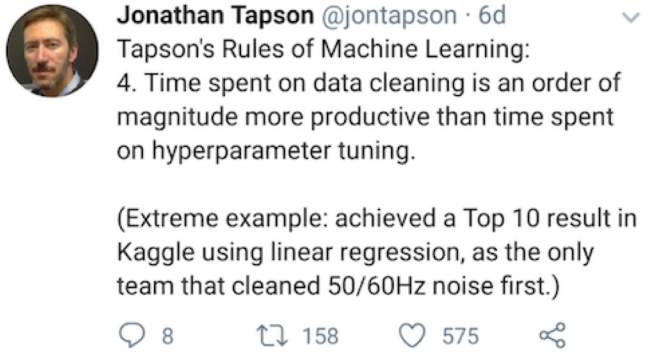
- Can you enforce constraints you know with extra loss functions or unusual network structure? (i.e. conservation of energy, symmetry)
- Is dimensionality reduction feasible?
  - If yes, can make DL easier to reduce first
  - If no, don't want any layers that are too skinny (forcing low-dimensional representation of data and losing information)
- Often helps to consider end goal and have one joint optimization
  - Co-train feature selection and your particular DL problem
  - Co-train autoencoder with how you want to use the low-dimensional representation

Helpful in my experience

- Make your loss function closer to true end goal (i.e. if you want multi-step prediction, penalize that during training)
- Visualize data & results for troubleshooting
- Residual networks can be hugely helpful, especially if output is similar to input, i.e. train output$(x) = NN(x) + x$ instead of $= NN(x)$
  - Related idea: discrepancy modeling
    - If you have a fast but inaccurate simulation and you want to replace it with a more accurate NN, could learn error instead of replacing everything
    - Here, you take advantage of the simulation you have
  - Can you use sparsity to improve interpretability?
    - i.e. Force one part to be sparse combination of patterns I expect
  - Although DL learns features for you, can still help to use domain knowledge to pick good features!

Troubleshooting

- Look at inputs & outputs for random examples
- Think about different types of data in your dataset
  - Are they evenly distributed?
  - How are the errors distributed?
- Use your domain knowledge: what should be true about the outputs?
- Can you iterate on a small subset of your data? A simpler problem?
- Does your data need to be cleaned up?
- Can you add informative features?
- Make a list of your assumptions.
- Do you have vanishing/exploding gradients?
- Ask a coworker! Experience is huge in DL.



Conclusions

- Rule #1: think carefully about validation and generalization
- Rule #2: deep learning cannot extrapolate

- This field is moving quickly
  - Keep learning (balancing from mainstream ML & scientific ML)
  - Ask for help!
  - Gain intuition by trying small problems
- Take advantage of your domain knowledge
- Look at your data again
- Look at the errors again


Thank You!
blusch@anl.gov