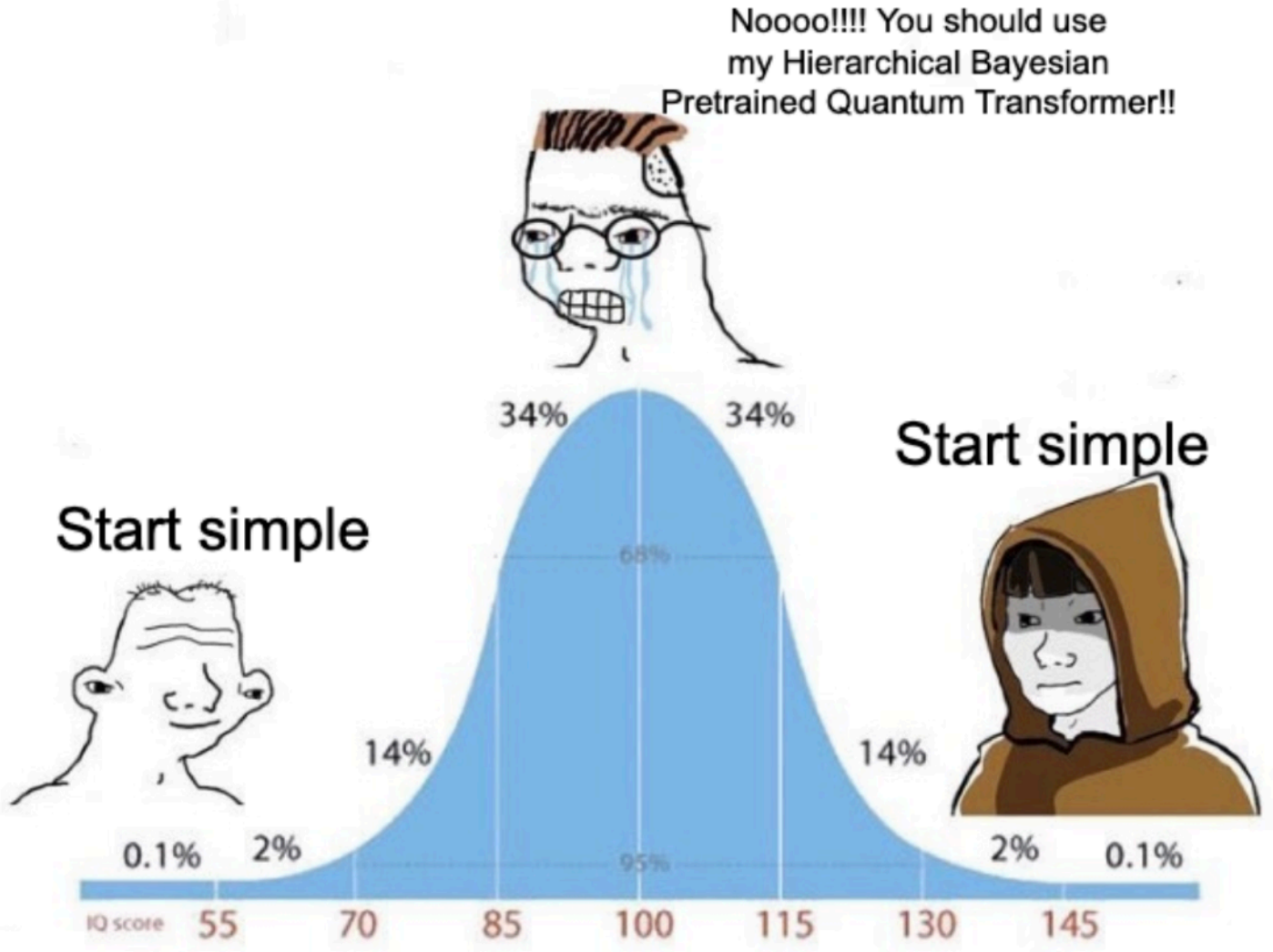
Machine Learning Best Practices



Miles Cranmer (Cambridge)

Good material from

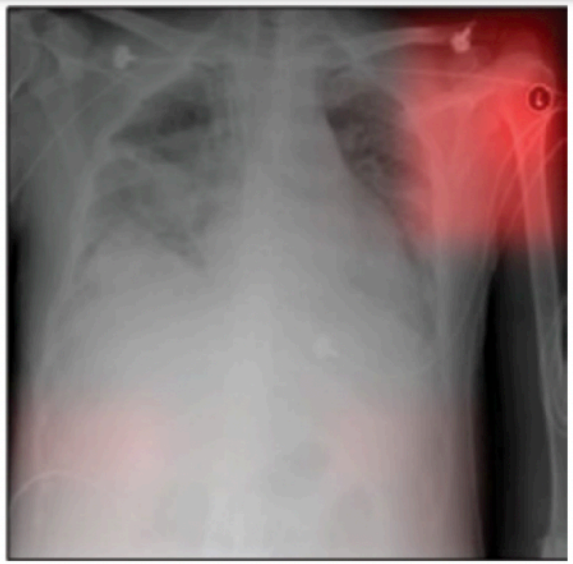
- Andrej Karpathy [blog]
- Ian Goodfellow et al. [text]
- Josh Tobin [notes]
- Slav Ivanov [notes]
- Jeremy Howard [blog/notes]

Overview

- General neural network training recipe
- Small-scale experimentation, scaling, debugging
- Review Karpathy's “Recipe”
- Discuss experiment tracking; reproducibility

Neural networks are powerful. ***Too*** powerful

Neural networks are powerful. ***Too*** powerful

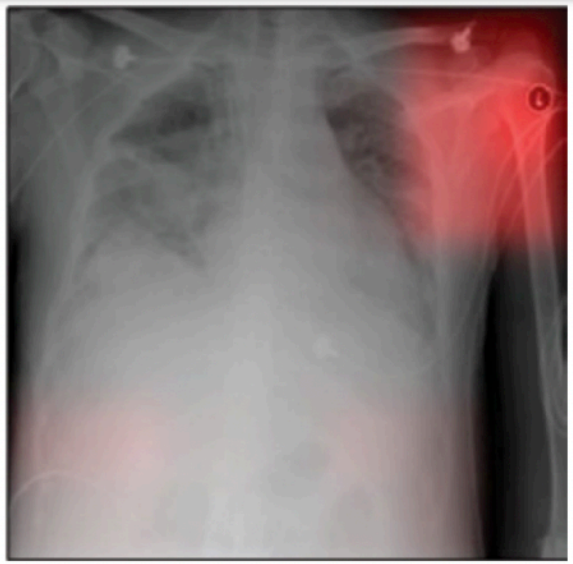


Zech 2018

Task for DNN	Recognize pneumonia
Problem	Fails on scans from new hospitals
Shortcut	Looks at hospital token, not lung

Neural networks are powerful. ***Too*** powerful

- They can fit ***anything***

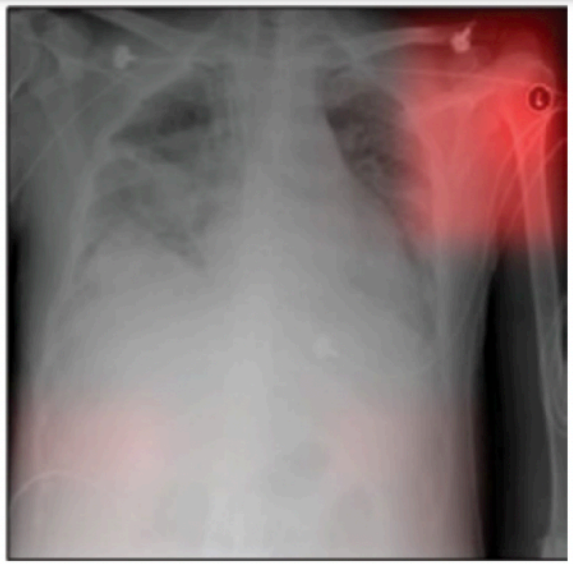


Zech 2018

Task for DNN	Recognize pneumonia
Problem	Fails on scans from new hospitals
Shortcut	Looks at hospital token, not lung

Neural networks are powerful. ***Too*** powerful

- They can fit ***anything***
- This means that they can fit **around** bugs

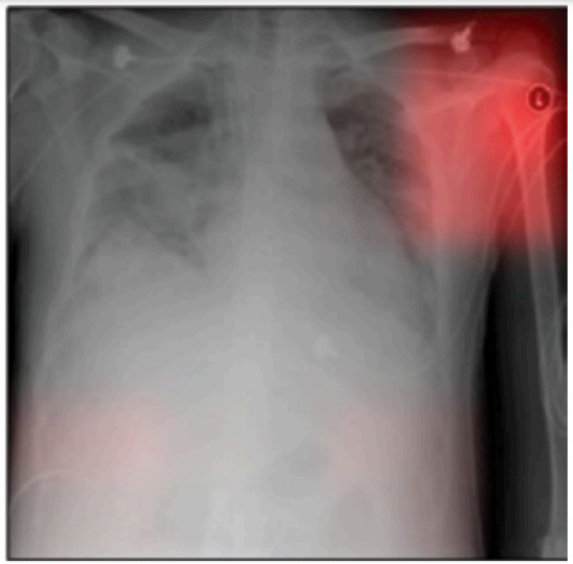


Zech 2018

Task for DNN	Recognize pneumonia
Problem	Fails on scans from new hospitals
Shortcut	Looks at hospital token, not lung

Neural networks are powerful. ***Too*** powerful

- They can fit ***anything***
- This means that they can fit **around** bugs
- Karpathy calls this a “leaky abstraction" - the network can learn something, but it might not be what you intended!

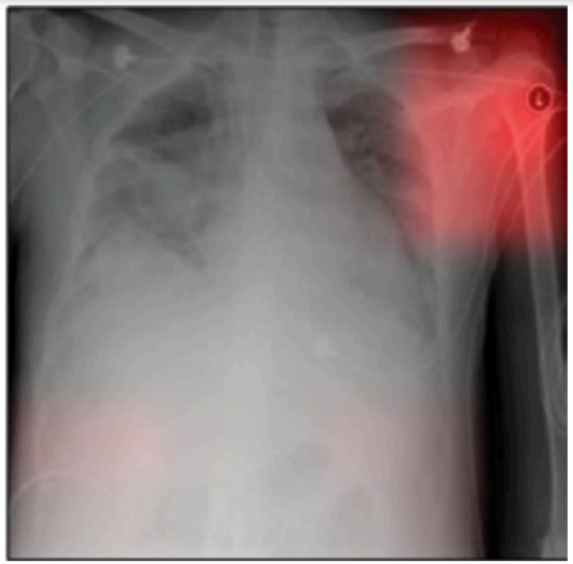


Zech 2018

Task for DNN	Recognize pneumonia
Problem	Fails on scans from new hospitals
Shortcut	Looks at hospital token, not lung

Neural networks are powerful. ***Too*** powerful

- They can fit ***anything***
- This means that they can fit **around** bugs
- Karpathy calls this a “leaky abstraction" - the network can learn something, but it might not be what you intended!

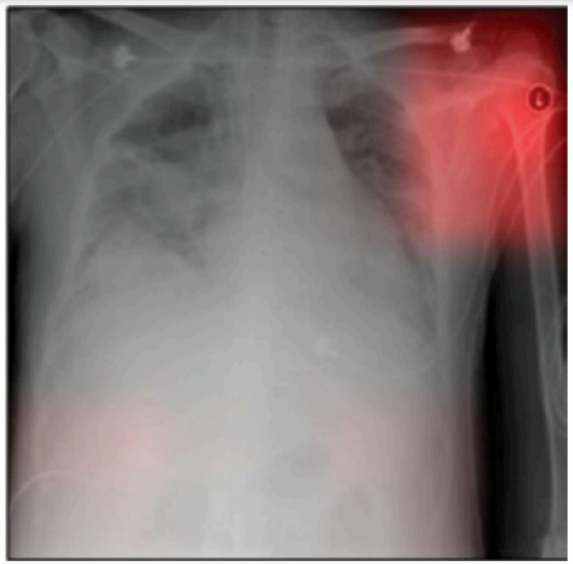


Zech 2018

Task for DNN	Recognize pneumonia
Problem	Fails on scans from new hospitals
Shortcut	Looks at hospital token, not lung

Neural networks are powerful. ***Too*** powerful

- They can fit ***anything***
- This means that they can fit **around** bugs
- Karpathy calls this a “leaky abstraction" - the network can learn something, but it might not be what you intended!

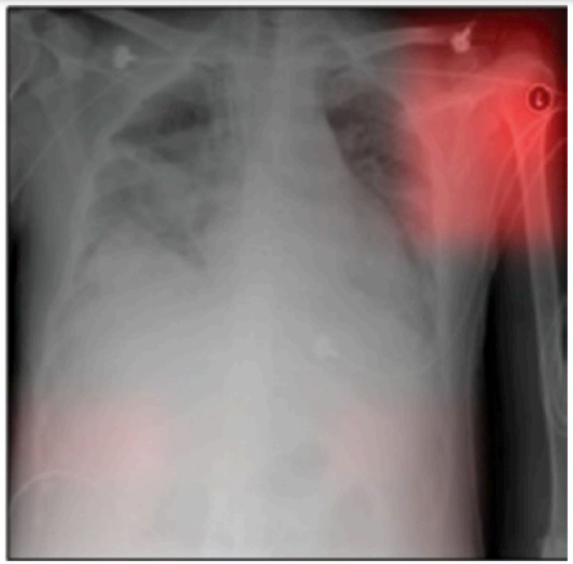


Zech 2018

Task for DNN	Recognize pneumonia
Problem	Fails on scans from new hospitals
Shortcut	Looks at hospital token, not lung

Neural networks are powerful. ***Too*** powerful

- They can fit ***anything***
- This means that they can fit **around** bugs
- Karpathy calls this a “leaky abstraction" - the network can learn something, but it might not be what you intended!



Zech 2018

Task for DNN	Recognize pneumonia
Problem	Fails on scans from new hospitals
Shortcut	Looks at hospital token, not lung

- They often fail ***quietly***

Importance of being meticulous

Importance of being meticulous

- When working with deep learning, you should basically be a “conspiracy theorist”

Importance of being meticulous

- When working with deep learning, you should basically be a “conspiracy theorist"
- Don't believe anything is working; be skeptical about every single step.

Importance of being meticulous

- When working with deep learning, you should basically be a “conspiracy theorist”
- Don't believe anything is working; be skeptical about every single step.
- Don't believe that your data is good.

Importance of being meticulous

- When working with deep learning, you should basically be a “conspiracy theorist”
- Don't believe anything is working; be skeptical about every single step.
- Don't believe that your data is good.
- Don't believe that your loss function is computed correctly.

Importance of being meticulous

- When working with deep learning, you should basically be a “conspiracy theorist”
- Don't believe anything is working; be skeptical about every single step.
- Don't believe that your data is good.
- Don't believe that your loss function is computed correctly.

Importance of being meticulous

- When working with deep learning, you should basically be a “conspiracy theorist”
- Don't believe anything is working; be skeptical about every single step.
- Don't believe that your data is good.
- Don't believe that your loss function is computed correctly.

- Verify, verify, verify! Bugs in neural networks are much more sinister than bugs in regular software.

Recipe

Recipe

1. Become one with the data

Recipe

1. Become one with the data
2. Keep it simple; build an end-to-end training pipeline

Recipe

1. Become one with the data
2. Keep it simple; build an end-to-end training pipeline
    - Make this dumb. Linear models!

Recipe

1. Become one with the data
2. Keep it simple; build an end-to-end training pipeline
    - Make this dumb. Linear models!
    - Newcomers to ML like to try the fanciest thing they've read about first.

Recipe

1. Become one with the data
2. Keep it simple; build an end-to-end training pipeline
    - Make this dumb. Linear models!
    - Newcomers to ML like to try the fanciest thing they've read about first.
    - That is very bad and you will get hurt!

Recipe

1. Become one with the data
2. Keep it simple; build an end-to-end training pipeline
    - Make this dumb. Linear models!
    - Newcomers to ML like to try the fanciest thing they've read about first.
    - That is very bad and you will get hurt!
3. Try to overfit on a tiny subset of your data (~1%), with a tiny version of your model (~1%)

Recipe

1. Become one with the data
2. Keep it simple; build an end-to-end training pipeline
    - Make this dumb. Linear models!
    - Newcomers to ML like to try the fanciest thing they've read about first.
    - That is very bad and you will get hurt!
3. Try to overfit on a tiny subset of your data (~1%), with a tiny version of your model (~1%)
4. Increase model capacity until can fit

Recipe

1. Become one with the data
2. Keep it simple; build an end-to-end training pipeline
    - Make this dumb. Linear models!
    - Newcomers to ML like to try the fanciest thing they've read about first.
    - That is very bad and you will get hurt!
3. Try to overfit on a tiny subset of your data (~1%), with a tiny version of your model (~1%)
4. Increase model capacity until can fit
5. Regularize

Recipe

1. Become one with the data
2. Keep it simple; build an end-to-end training pipeline
    - Make this dumb. Linear models!
    - Newcomers to ML like to try the fanciest thing they've read about first.
    - That is very bad and you will get hurt!
3. Try to overfit on a tiny subset of your data (~1%), with a tiny version of your model (~1%)
4. Increase model capacity until can fit
5. Regularize
6. Generally: do small-scale tuning; and only then do large-scale runs

1. Become one with the data

1. Become one with the data

- Don't touch any ML yet.

1. Become one with the data

- Don't touch any ML yet.
- Open up your dataset in an interactive Jupyter notebook, and visualize it.

1. Become one with the data

- Don't touch any ML yet.
- Open up your dataset in an interactive Jupyter notebook, and visualize it.
- Are there duplicates?

1. Become one with the data

- Don't touch any ML yet.
- Open up your dataset in an interactive Jupyter notebook, and visualize it.
- Are there duplicates?
- Look at the max and min of each feature

1. Become one with the data

- Don't touch any ML yet.
- Open up your dataset in an interactive Jupyter notebook, and visualize it.
- Are there duplicates?
- Look at the max and min of each feature
- Look at class imbalances

1. Become one with the data

- Don't touch any ML yet.
- Open up your dataset in an interactive Jupyter notebook, and visualize it.
- Are there duplicates?
- Look at the max and min of each feature
- Look at class imbalances
- Try to find corrupted labels

1. Become one with the data

- Don't touch any ML yet.
- Open up your dataset in an interactive Jupyter notebook, and visualize it.
- Are there duplicates?
- Look at the max and min of each feature
- Look at class imbalances
- Try to find corrupted labels
- Note ***anything at all*** that could mess with training!

1. Become one with the data

- Don't touch any ML yet.
- Open up your dataset in an interactive Jupyter notebook, and visualize it.
- Are there duplicates?
- Look at the max and min of each feature
- Look at class imbalances
- Try to find corrupted labels
- Note ***anything at all*** that could mess with training!
- Can you “fit" the data, by eye? What is ***your*** performance on the test set?

1. Become one with the data

- Don't touch any ML yet.
- Open up your dataset in an interactive Jupyter notebook, and visualize it.
- Are there duplicates?
- Look at the max and min of each feature
- Look at class imbalances
- Try to find corrupted labels
- Note ***anything at all*** that could mess with training!
- Can you “fit" the data, by eye? What is ***your*** performance on the test set?

1. Become one with the data

- Don't touch any ML yet.
- Open up your dataset in an interactive Jupyter notebook, and visualize it.
- Are there duplicates?
- Look at the max and min of each feature
- Look at class imbalances
- Try to find corrupted labels
- Note ***anything at all*** that could mess with training!
- Can you “fit" the data, by eye? What is ***your*** performance on the test set?

- It is good to spend ***hours*** doing this process for a new project; resist the urge of throwing ML at it!

1. Become one with the data

- Don't touch any ML yet.
- Open up your dataset in an interactive Jupyter notebook, and visualize it.
- Are there duplicates?
- Look at the max and min of each feature
- Look at class imbalances
- Try to find corrupted labels
- Note ***anything at all*** that could mess with training!
- Can you “fit" the data, by eye? What is ***your*** performance on the test set?

- It is good to spend ***hours*** doing this process for a new project; resist the urge of throwing ML at it!

1. Become one with the data

- Don't touch any ML yet.
- Open up your dataset in an interactive Jupyter notebook, and visualize it.
- Are there duplicates?
- Look at the max and min of each feature
- Look at class imbalances
- Try to find corrupted labels
- Note ***anything at all*** that could mess with training!
- Can you “fit" the data, by eye? What is ***your*** performance on the test set?

- It is good to spend ***hours*** doing this process for a new project; resist the urge of throwing ML at it!

- ^Manual inspection reveals anomalies that will ***sabotage*** training! Very important to do this FIRST, rather than run around in circles fixing a bug only to realise it was the data all along!

2. Keep it simple; build an end-to-end training pipeline

- Set up an end-to-end training loop
- Start with a single linear model, or another “dumb" model
- Evaluate the training and validation loss, and ***confirm*** it is what you expect (should usually be better than random)
- Use this opportunity to create plotting/analysis scripts of trained models. You want to make these ***before*** you do any crazy ML, so you can verify they work

- You want to ***prove*** your code can learn something; and that your pipeline is correct

Check initial results

- Verify, by hand - does the loss start at the expected value (e.g., -log(1/num_classes))
- Does the model beat an ***input-independent*** baseline?
- Does the linear model train stably for at least a few epochs?

- Fix all issues before complicating the model!

3. Overfit on a small batch

- Overfit on ~1% of the dataset. (Now, use a slightly larger model).
- Focus on driving training loss down.
- This confirms no hidden bugs or capacity limitations.
- Once you can overfit, you know the pipeline's fundamentals are sound and ***all wires connect***

4. Increase model capacity

- Now, try to ***overfit the full training set***
- Use a larger model until you can do this (but don't complicate it yet)
- If you cannot overfit, there might be a problem!

Note: Inspect the data as it enters the model

- Even if you've thoroughly inspected your dataset, you might still have an issue at the preprocessing stage
- Solution: visualize the data ***as it enters the model***
- Right before it enters, you should add a hook that captures and logs the data - you can visualise it right there

5. Regularize

- After proving you can overfit, you want to improve generalization
- Techniques include weight decay, dropout, data augmentation, label smoothing, and early stopping.
- Gradually introduce each method, confirming your validation performance improves.
- Avoid adding too many new pieces at once.
- Monitor both training and validation losses carefully.

Augment dataset

- Data augmentation is often the strongest regularizer for images, text, or audio.
- Meaningful transformations (flips, crops, color jitter, synonyms) enlarge your dataset artificially.
- For physics datasets, there are often physically-motivated augmentations - these are very smart to use!
- Ensure they remain label-preserving. Overly aggressive augmentations can harm performance. Start simple (random flips or slight crops) and expand. Always check sample outputs to confirm correctness thoroughly.

6. Tune hyperparameters

- Tuning on small scale is very important
- Try to see the trends in hyper parameters, BEFORE moving to large-scale
- By staying at small scale first, you get a very quick prototype loop. You can quickly test new ideas
- There are some known scaling relations you can use for hyper parameters - such as