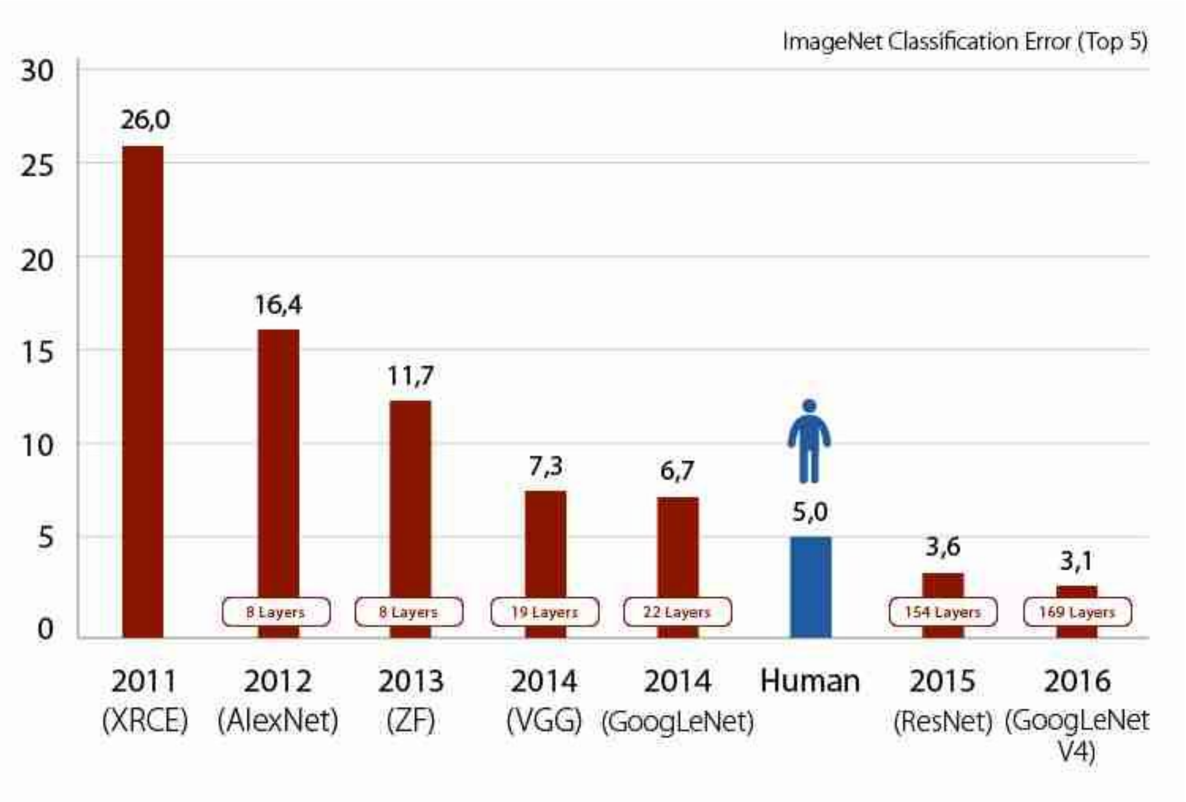
The Arcane Arts of Training Neural Nets

Santiago Hincapie-Potes

Deep Learning is Awesome



Deep Learning is Awesome

AlphaGo

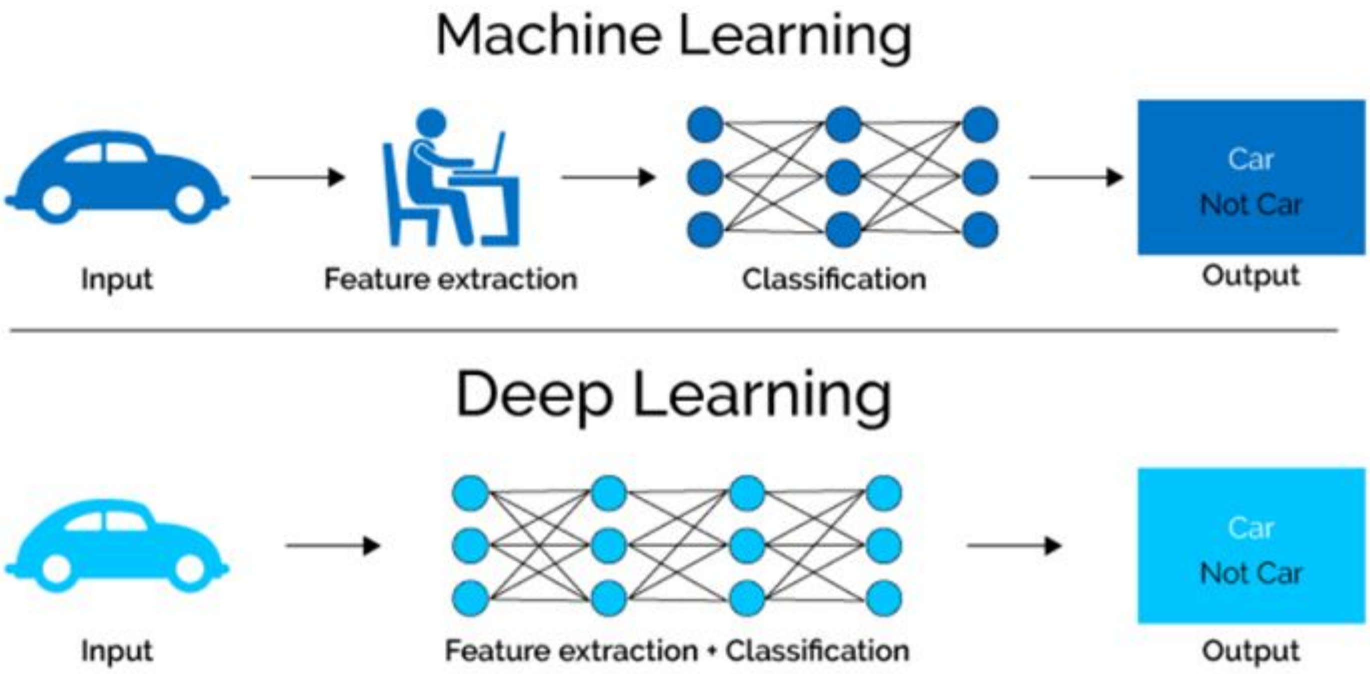
Deep Learning is Awesome

Deep Learning is Awesome

```
import os
import sys

# Count lines of code in the given directory, separated by file extension
def main(directory):
    line_count = {}
    for filename in os.listdir(directory):
        _, ext = os.path.splitext(filename)
        if ext not in line_count:
            line_count[ext] = 0
        for line in open(os.path.join(directory, filename)):
            line_count[ext] += 1
            line_count[ext] += 1                13%
            line_count[ext                Tab   20%
            line_count[ext] +=               3  14%
            line_count[ext].append(          4   3%
            line                             5  23%
```

End-to-end systems



Why deep learning now?

1952 Stochastic Gradient Descent

1958 Perceptron
- Learnable Weights

⋮

1986 Backpropagation
- Multi-Layer Perceptron

1995 Deep Convolutional NN
- Digit Recognition

⋮

Neural Networks date back decades, so why the resurgence?

1. Big Data

- Larger Datasets
- Easier Collection & Storage

IMAGENET

WIKIPEDIA
The Free Encyclopedia

2. Hardware

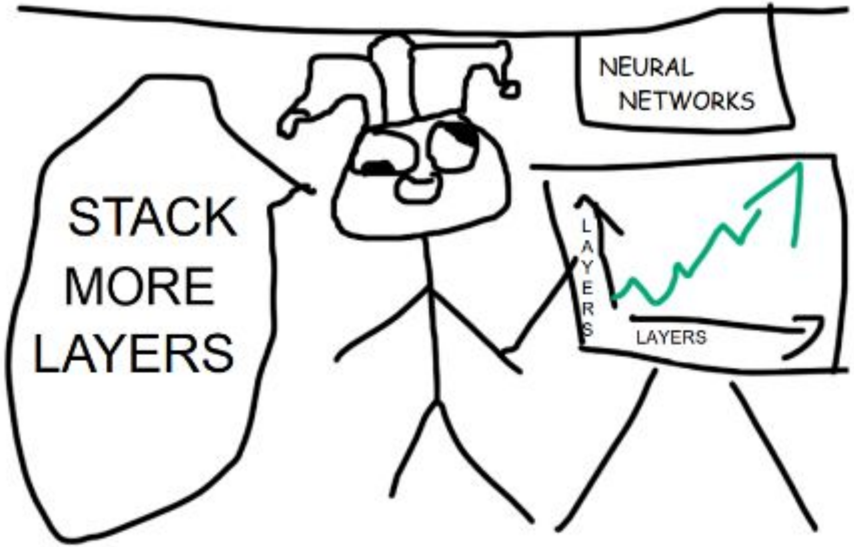
- Graphics Processing Units (GPUs)
- Massively Parallelizable

3. Software

- Improved Techniques
- New Models
- Toolboxes

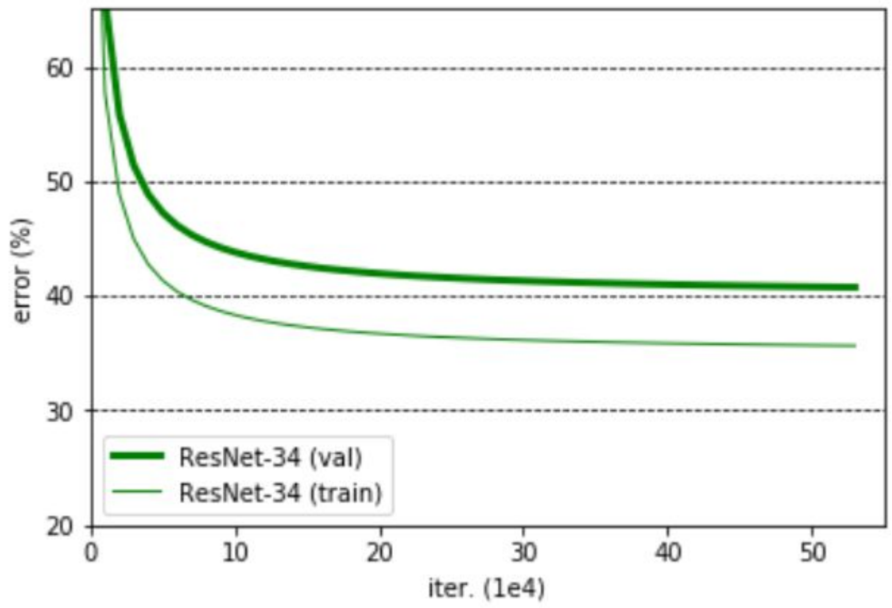
TensorFlow

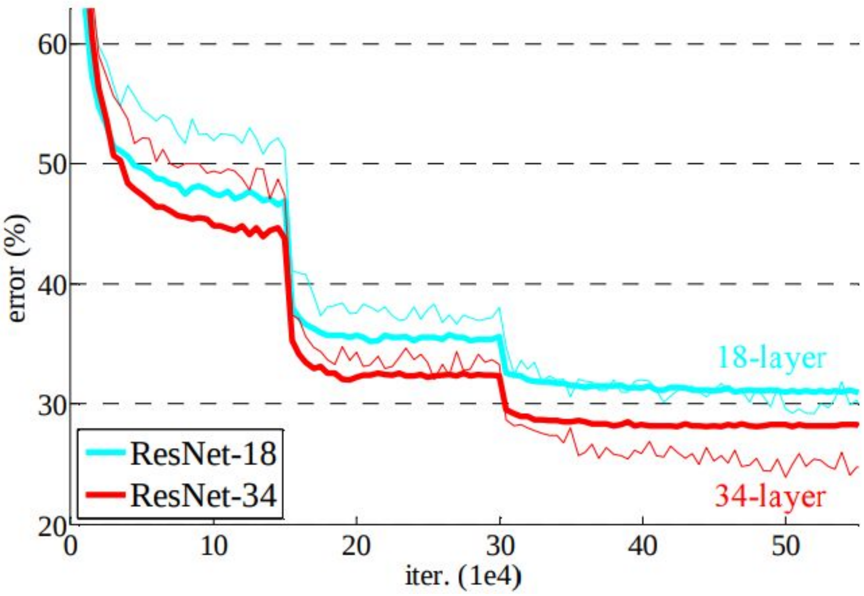

THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS


STACK MORE LAYERS
NEURAL NETWORKS
LAYERS
LAYERS

Deep learning is hard!

Why?





you vs the guy she tells you not to worry about

Why?

Andrej Karpathy
@karpathy

Debugging: first it doesn't compile. then doesn't link. then segfaults. then gives all zeros. then gives wrong answer. then only maybe works

2:30 am · 17 Jan 2014 · Twitter Web Client

5 Retweets **12** Likes

Bugs are invisible!!

- Labels out of order
- Incorrect shapes for your tensors
- Incorrect input to your loss function
- Forgot labels on data augmentation transformations.
- Initialized your weights from a pretrained checkpoint but didn’t use the original mean.
- Forgot `model.zero_grad()`
- Mortal typos
- Clipped loss instead of gradient
- ...

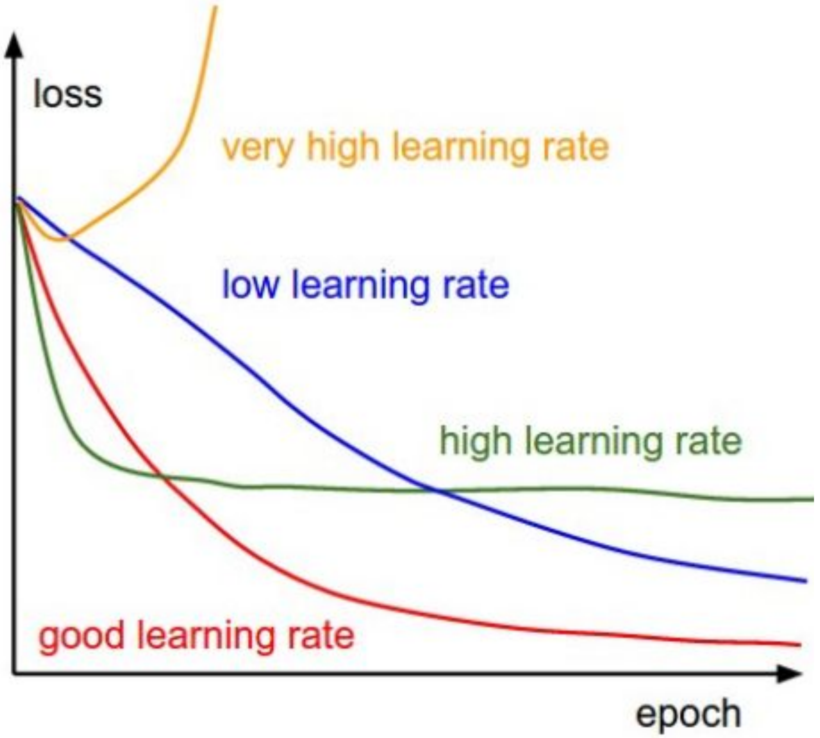
Bugs are invisible!!

- Labels out of order
- Incorrect shapes for your tensors
- Incorrect input to your loss function
- Forgot labels on data augmentation transformations.
- Initialized your weights from a pretrained checkpoint but didn't use the original mean
- Forgot `model.zero_grad()`
- Mortal typos
- Clipped loss instead of gradient
- ...

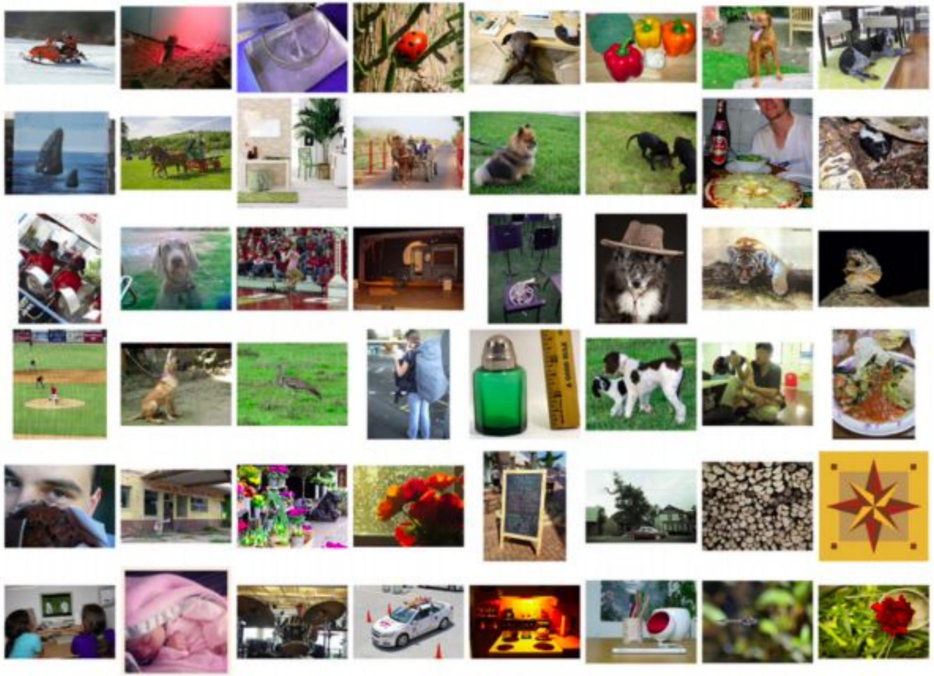
Bugs are invisible!!

- Labels out of order
- Incorrect shapes for your tensors
- Incorrect inputs to your loss function
- Forgot labels on data augmentation transformations.
- Initialized your weights from a pretrained checkpoint but didn't use the original mean
- Forgot `model.zero_grad()`
- Mortal typos
- Clipped loss instead of gradient
- ...

Hyperparameters metters!



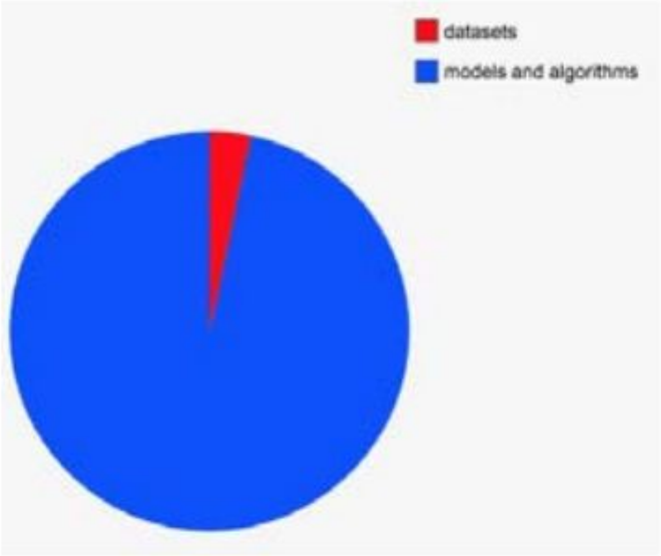
Dataset metters!



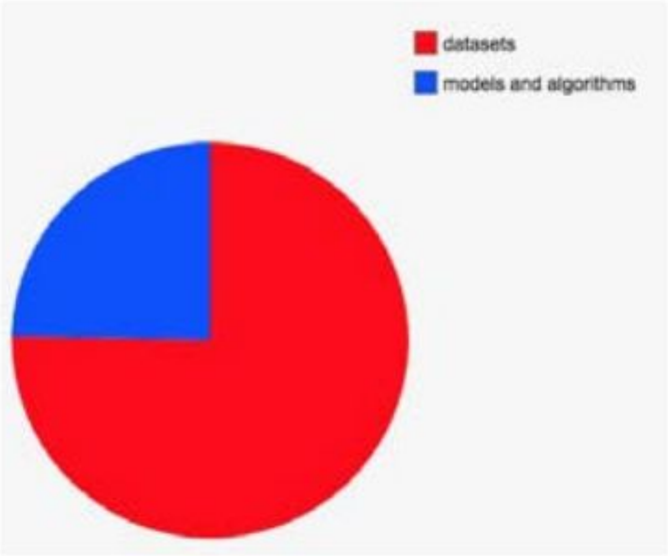
you vs the guy she tells you not to worry about

Dataset metters!

PhD



Tesla



Dataset metters!

Vicki Boykis @vboykis · 28 Jan 2019
Have been extremely curious about this for a while now, so I decided to create a poll.
"As someone titled 'data scientist' in 2019, I spend most of (60%+) my time:"
("Other") also welcome, add it in the replies.

Picking features/models	6.2%
Cleaning data/Moving data	**67.3%**
Deploying models in prod	3.6%
Analyzing/presenting data	22.9%

2,116 votes · Final results

Vicki Boykis @vboykis · 15 Jan 2019
Just a personal anecdote, but, in the past 2 years, % of any given project:
+ that involves ML: 15%
+ that involves moving, monitoring, and counting data to feed ML: 85%

mat kelcey @mat_kelcey · 11 Feb 2019
for my last few ML projects the complexity hasn't been in the modelling or training; it's been in input preprocessing. find myself running out of CPU more than GPU & in one project i'm actually unsure how to optimise the python further (& am considering c++ for one piece)

Katherine Scott @kscottz · 1 Feb 2019
One of the biggest failures I see in junior ML/CV engineers is a complete lack of interest in building data sets. While it is boring grunt work I think there is so much to be learned in putting together a dataset. It is like half the problem.

Dataset construction is hard!!

- Not enough data
- Class imbalances
- Noisy labels
- Train / test from different distributions
- Not enough variance
- Non representative samples
- ...

FAQs

- Optimal dataset size?
- Train/Validation/Test split proportion?
- Different training and test set distributions .-.
- Image resolution?
- Roadmap

FAQs

- Optimal dataset size?
- Train/Validation/Test split proportion?
- Different training and test set distributions .-.
- Image resolution?
- Roadmap

What’s the optimal size of a dataset?

- Short answer: We don’t know

What’s the optimal size of a dataset?

- Short answer: We don’t know
- Long answer:
  - We don’t know
  - Difficulty looks like a reasonable proxy.
  - How to measure the difficulty of a task?
    - We don’t know :c
    - Buuuut similar problems should have similar difficulties
    - Similar problems should need similar size dataset
- Use a known **proxy project** to evaluate how much data you need!

FAQs

- Optimal dataset size?
- Train/Validation/Test split proportion?
- Different training and test set distributions .-.
- Image resolution?
- Roadmap

Train / Validation / Test set

- Train
  - Run your learning algorithm
- Validation
  - Tune parameters
  - Make other decisions regarding the learning algorithm
- Test
  - Evaluate the performance of the algorithm
  - Not to make any decisions regarding what learning algorithm or parameters to use.

Train / Validation / Test set

- Train
  - Run your learning algorithm
- Validation
  - Tune parameters
  - Make other decisions regarding the learning algorithm
- Test
  - Evaluate the performance of the algorithm
  - Not to make any decisions regarding what learning algorithm or parameters to use.

Choose validation and test sets to reflect data you expect to get in the future and want to do well on

Train / Validation / Test set

- Validation set should be large enough to detect differences between algorithms.

Train / Validation / Test set

- Validation set should be large enough to detect differences between algorithms.
- Test set should be large enough to give high confidence in the overall performance of your system.

Train / Validation / Test set

- Validation set should be large enough to detect differences between algorithms.
- Test set should be large enough to give high confidence in the overall performance of your system.
- Validation / Test set must come from the same distribution!

Train / Validation / Test set

- Validation set should be large enough to detect differences between algorithms.
- Test set should be large enough to give high confidence in the overall performance of your system.
- Validation / Test set must come from the same distribution!
- If there is a mismatch between Train and Test sets distribution:
  - There will also be a mismatch between train and validation sets
  - Create an extra validation set (a subset of the training)
    - Measure the overfit of training set.
    - Measure data mismatch.

FAQs

- Optimal dataset size?
- Train/Validation/Test split proportion?
- Different training and test set distributions .-.
- Image resolution?
- Roadmap

FAQs

- Optimal dataset size?
- Train/Validation/Test split proportion?
- Different training and test set distributions .-.
- Image resolution?
- Roadmap

Roadmap

1. Data labeling
2. Data storage
3. Data versioning
4. Data workflow

Roadmap

1. Data labeling
2. Data storage
3. Data versioning
4. Data workflow

Data labeling

- Data labeling requires separate software stack, temporary labor, and quality assurance. **Makes sense to outsource.**
- If not, then at least use existing software.
- Hiring part-time makes more sense than trying to make crowdsourcing work.

Data labeling

- Data labeling requires separate software stack, temporary labor, and quality assurance. **Makes sense to outsource.**
- If not, then at least use existing software.
- Hiring part-time makes more sense than trying to make crowdsourcing work.
- How to make labeling more efficient?
  - Weak supervision (e.g. Snorkel)
  - Jeremy Howard's tool (platform.ai)

Data labeling

- Data labeling requires separate software stack, temporary labor, and quality assurance. **Makes sense to outsource.**
- If not, then at least use existing software.
- Hiring part-time makes more sense than trying to make crowdsourcing work.
- How to make labeling more efficient?
  - Weak supervision (e.g. Snorkel)
  - Jeremy Howard's tool (platform.ai)
- How to make labeling more accurate?
  - Make it simple!
  - Label quality control

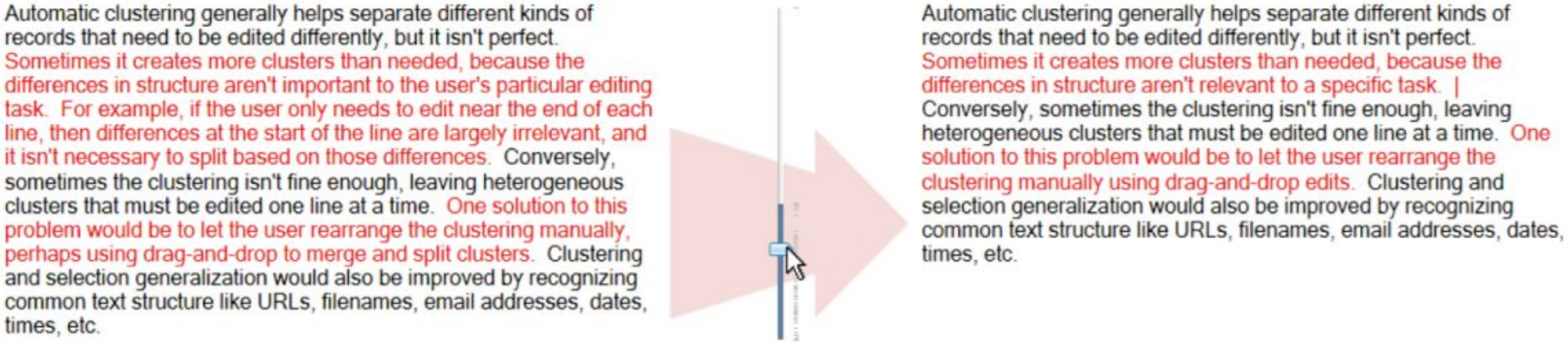
Soylent: A Word Processor with a Crowd Inside

Michael S. Bernstein[1], *Greg Little*[1], *Robert C. Miller*[1], *Björn Hartmann*[2], *Mark S. Ackerman*[3], *David R. Karger*[1], *David Crowell*[1], *Katrina Panovich*[1]

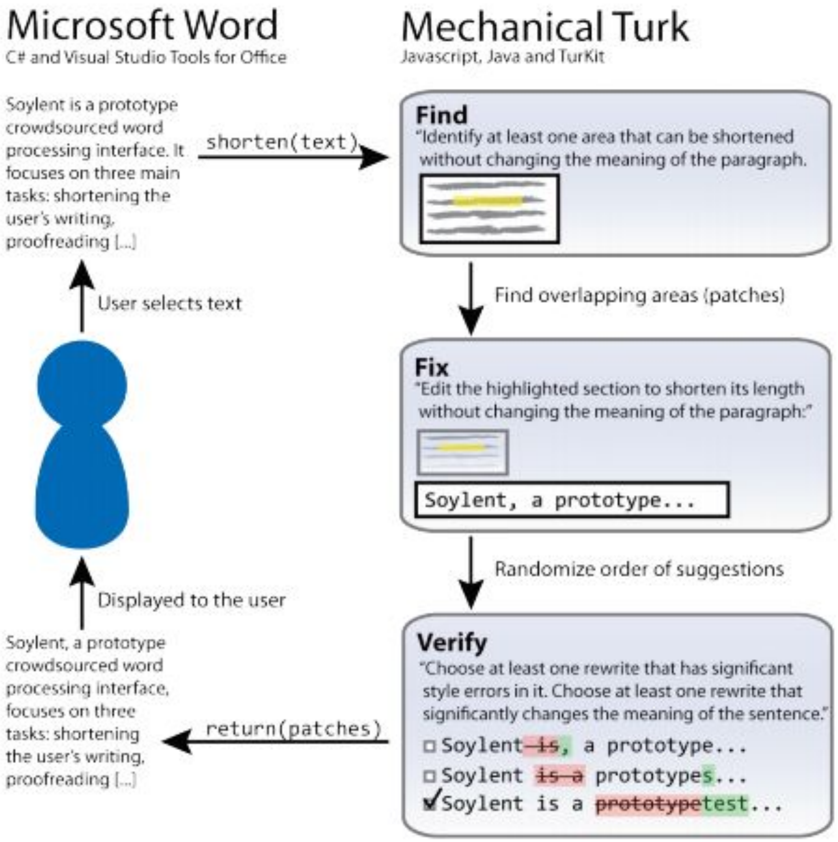
[1] MIT CSAIL
Cambridge, MA
{msbernst, glittle, rcm, karger, dcrowell, kp}@csail.mit.edu

[2] Computer Science Division
University of California, Berkeley
Berkeley, CA
bjoern@cs.berkeley.edu

[3] Computer Science & Engineering
University of Michigan
Ann Arbor, MI
ackerm@umich.edu



Soylent paper



Roadmap

1. Data labeling
2. Data storage
3. Data versioning
4. Data workflow

Data version control

- Level 0: unversioned
    - Inability to get back to a previous level of performance

Data version control

- Level 0: unversioned
  - Inability to get back to a previous level of performance
- Level 1: versioned via snapshot at training time
  - Would be far better to be able to version data just as easily as code.

Data version control

- Level 0: unversioned
  - Inability to get back to a previous level of performance
- Level 1: versioned via snapshot at training time
  - Would be far better to be able to version data just as easily as code.
- **Level 2: versioned as a mix of assets and code**
  - Heavy files stored in S3, with unique ids
  - Training data is stored as JSON or similar, referring to these ids and include relevant metadata
  - Store those JSON as code (git-lfs could be needed)

Data version control

- Level 0: unversioned
  - Inability to get back to a previous level of performance
- Level 1: versioned via snapshot at training time
  - Would be far better to be able to version data just as easily as code.
- **Level 2: versioned as a mix of assets and code**
  - Heavy files stored in S3, with unique ids
  - Training data is stored as JSON or similar, referring to these ids and include relevant metadata
  - Store those JSON as code (git-lfs could be needed)
- Level 3: specialized data versioning solution
  - Avoid these until you can fully explain how they will improve your project.
  - Leading solutions are DVC, Pachyderm, Quill.

Questions?

Deep learning is hard!

So… How to train deep learning models?

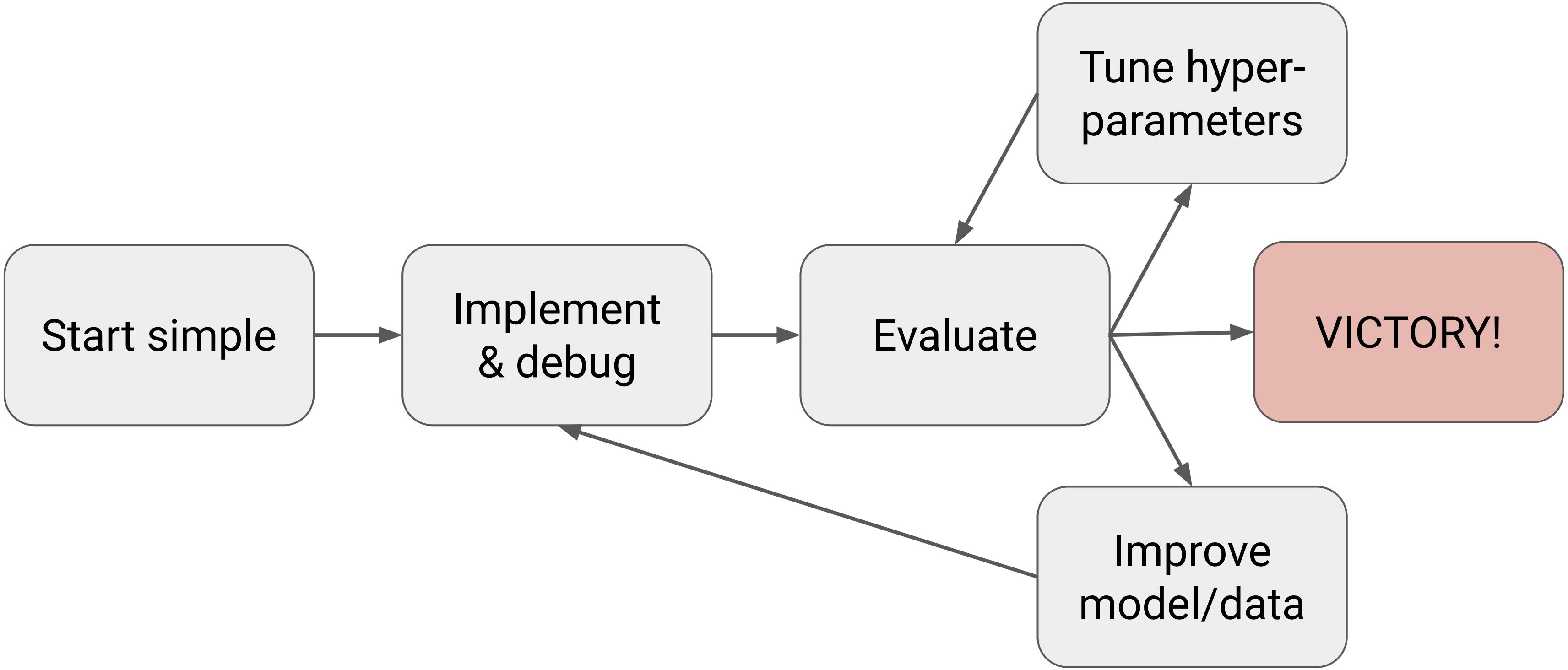
Pessimism!!

Pessimism!!

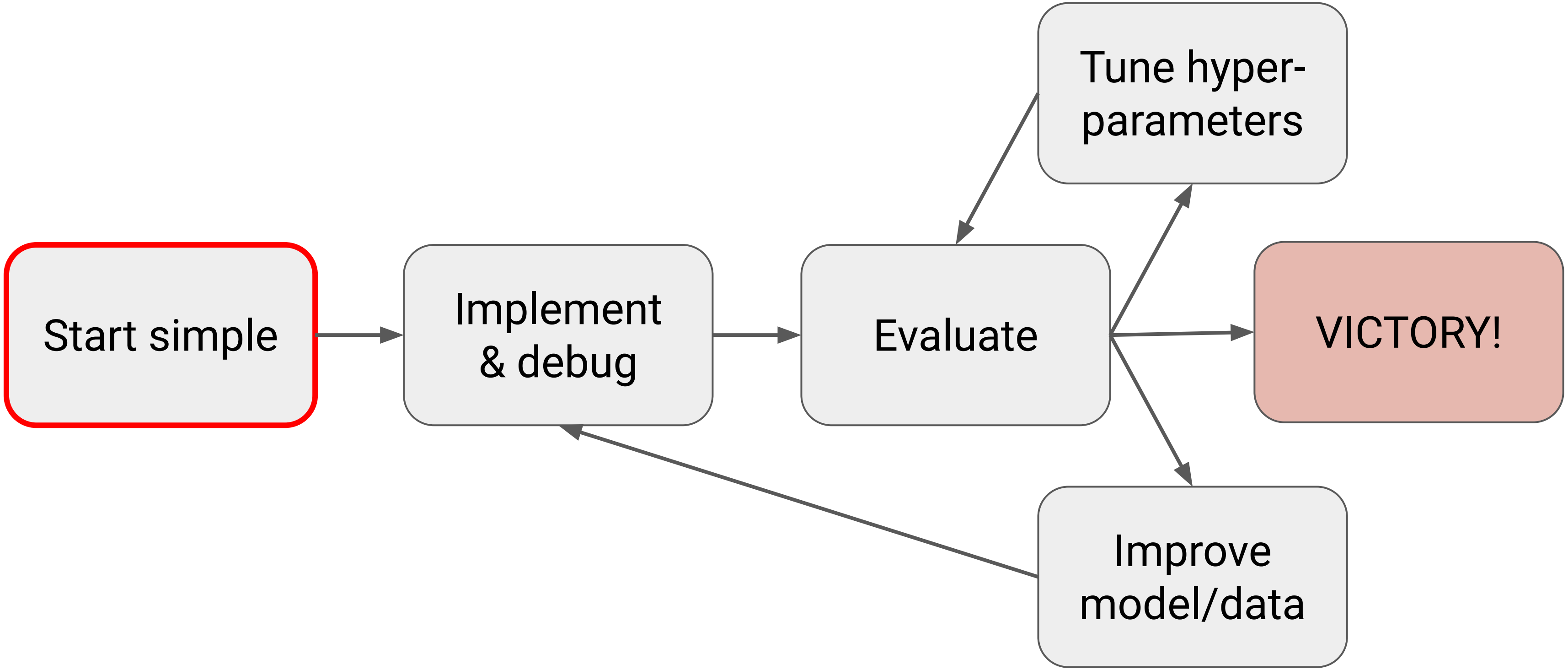
Since it’s hard to disambiguate errors...

...Start simple and gradually ramp up complexity

Deep Learning recipe



Deep Learning recipe



Starting simple

- Choose a simple architecture
- Use sensible defaults
- Normalize inputs
- Simplify the problem

Demystifying architecture selection

	Start here	this later
Images	LeNet	ResNet
Sequences	LSTM with one hidden layer (or 1DConv)	Attention model or WaveNet-like model
Other	MLP with few hidden layer	Problem-dependent

Default settings

- **Optimizer:** Adam optimizer with learning rate 3e-4
- **Activations:**
    - ReLU (FC and Conv models)
    - tanh (LSTMs)
- **Initialization:**
    - He et al. normal (relu)
    - Glorot normal (tanh)
- **Regularization:** None

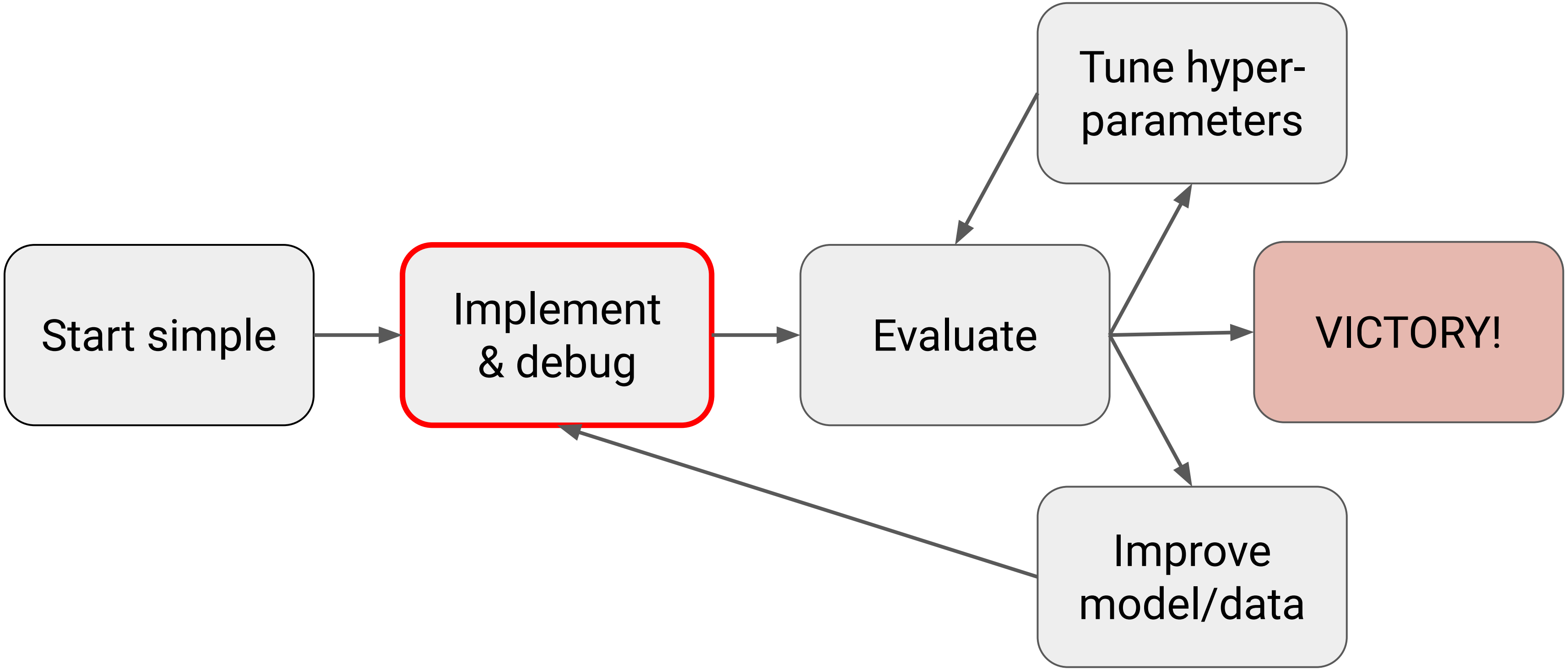
Starting simple

- Choose a simple architecture
- Use sensible defaults
- Normalize inputs
- Simplify the problem

Starting simple

- Choose a simple architecture
- Use sensible defaults
- Normalize inputs
- Simplify the problem

Deep Learning recipe



Top 10 most common DL bugs

- Incorrect shapes for your tensors
  - Accidental broadcasting: x.shape = (None,), y.shape = (None, 1), (x+y).shape = (None, None)
- Pre-processing inputs incorrectly
- Incorrect input to your loss function
  - Softmaxed outputs to a loss that expects logits
  - One-hoted outputs to a loss that expects sparse indices
- Forgot to set up train mode for the net correctly
- Numerical instability - inf/NaN
- Casting error
- Overwriting variable names
- Forgot to turn off bias when using batch norm
- Out of memory error

General advice for implementing your model

- Start with a lightweight implementation .
  - Minimum possible new lines of code
- Use off-the-shelf components.
  - Keras
  - Pytorch-lightning
- Start with a dataset you can load into memory.
- Debbugers are your friends.
- Verify loss at init.
- **Pro tip:** at this stage on each “iteration”, first run a couple of batches on test mode and then on train mode.

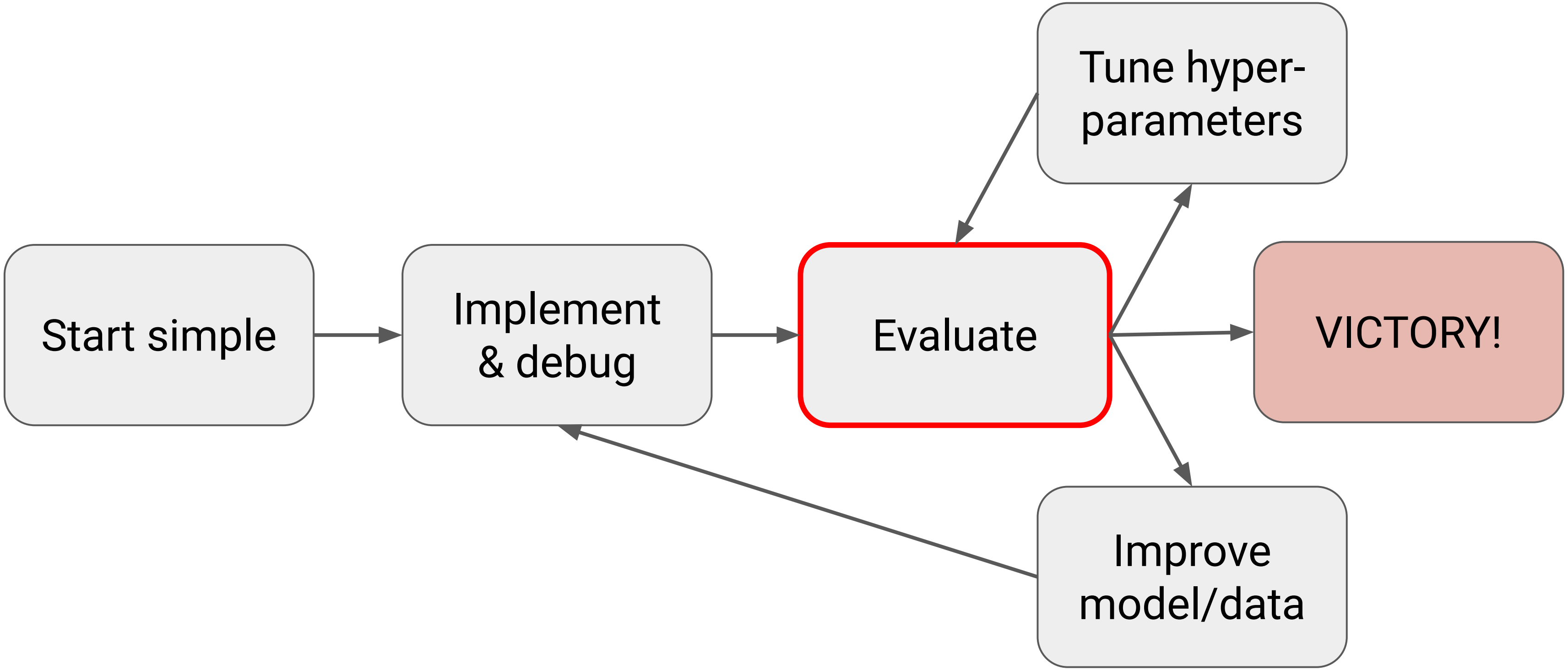
Overfit a single batch

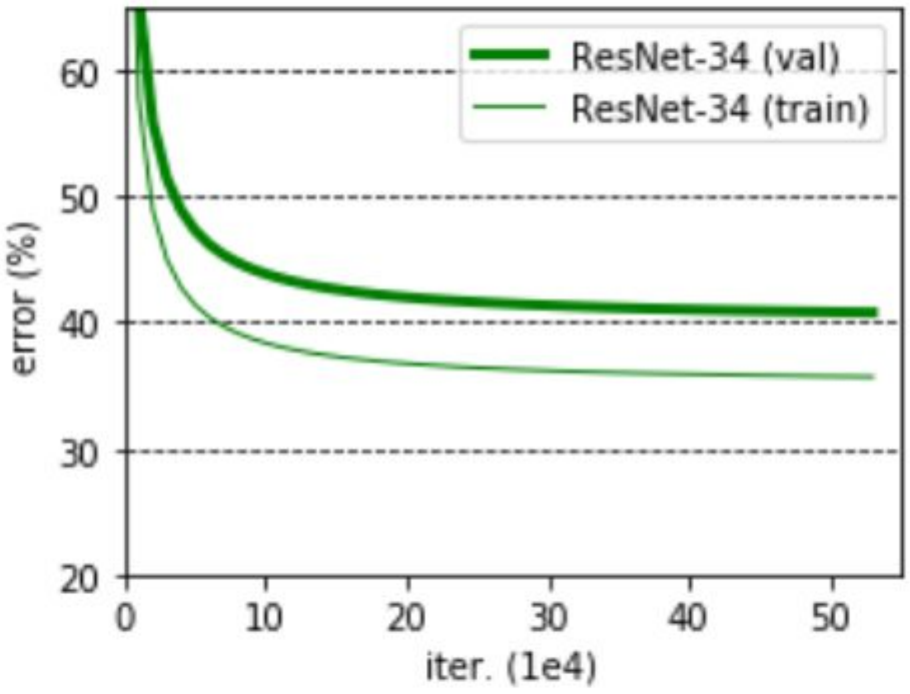
- If your neural network can't overfit a single data point, something is seriously wrong with the architecture.
- If error goes up:
  - Flipped loss/gradient sign
  - Learning rate too high
  - Softmax taken over wrong dimension
  - Numerical issues (Check exp, log, and div)
- If error plateaus:
  - Learining rate too low
  - Gradients not flowing though the whole model
  - Incorrect input to loss function (e.g. softmax (or even ReLU) instad of logits)
  - Data or labels corrupted

Compare your results!

1. Official implementation evaluated on similar dataset
   a. Walk through code line-by-line.
   b. Ensure your performance is up to par with expectations
2. Official implementation evaluated on benchmark
   a. Walk through code line-by-line.
3. Unofficial model implementation
   a. Walk through code line-by-line (with lower confidence)
4. Results from the paper (with no code)
   a. Ensure your performance is up to par with expectations
5. Results from your model on a benchmark dataset
   a. Make sure your model performs well in a simpler setting
6. Results from a similar model on a similar dataset
   a. Get a general sense of what kind of performance can be expected

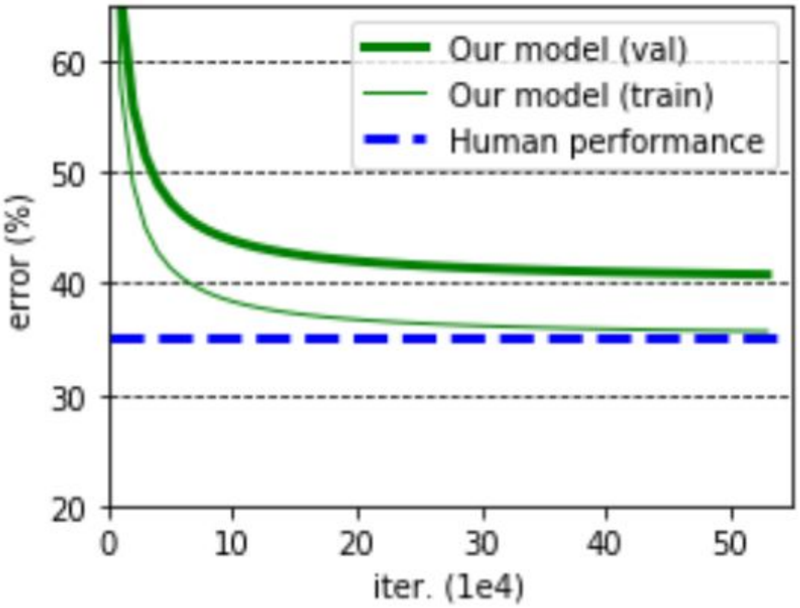
Deep Learning recipe

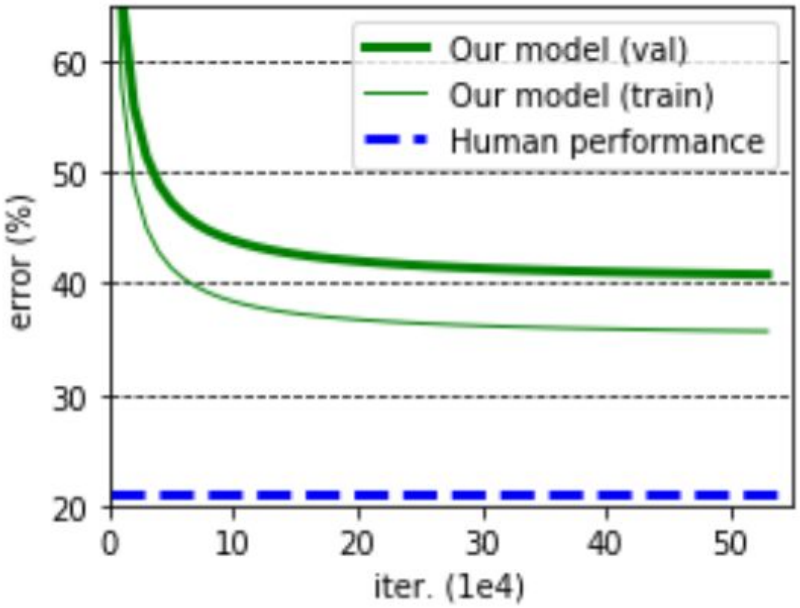




Quiz: overfit or underfit?

We need baselines!

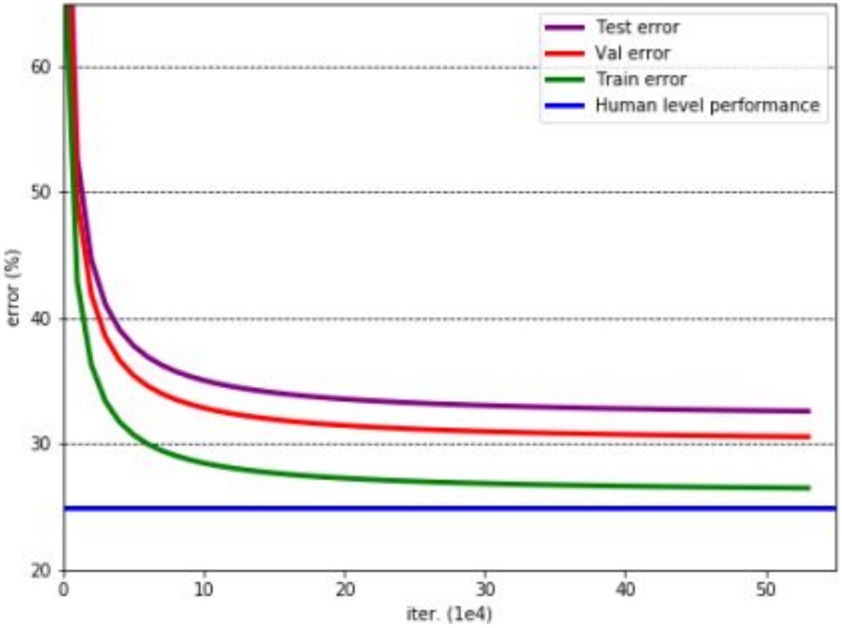




Bias-variance decomposition

Test error = irreducible error + bias + variance + val overfitting

Bias-variance decomposition



Error source	Value
Goal performance	1%
Train error	20%
Validation error	27%
Test error	28%

Bias-variance decomposition

Error source	Value
Goal performance	1%
Train error	20%
Validation error	27%
Test error	28%

Bias-variance decomposition

Error source	Value
Goal performance	1%
Train error	20%
Validation error	27%
Test error	28%

Train - goal = 19%
(under-fitting)

Bias-variance decomposition

Error source	Value
Goal performance	1%
Train error	20%
Validation error	27%
Test error	28%

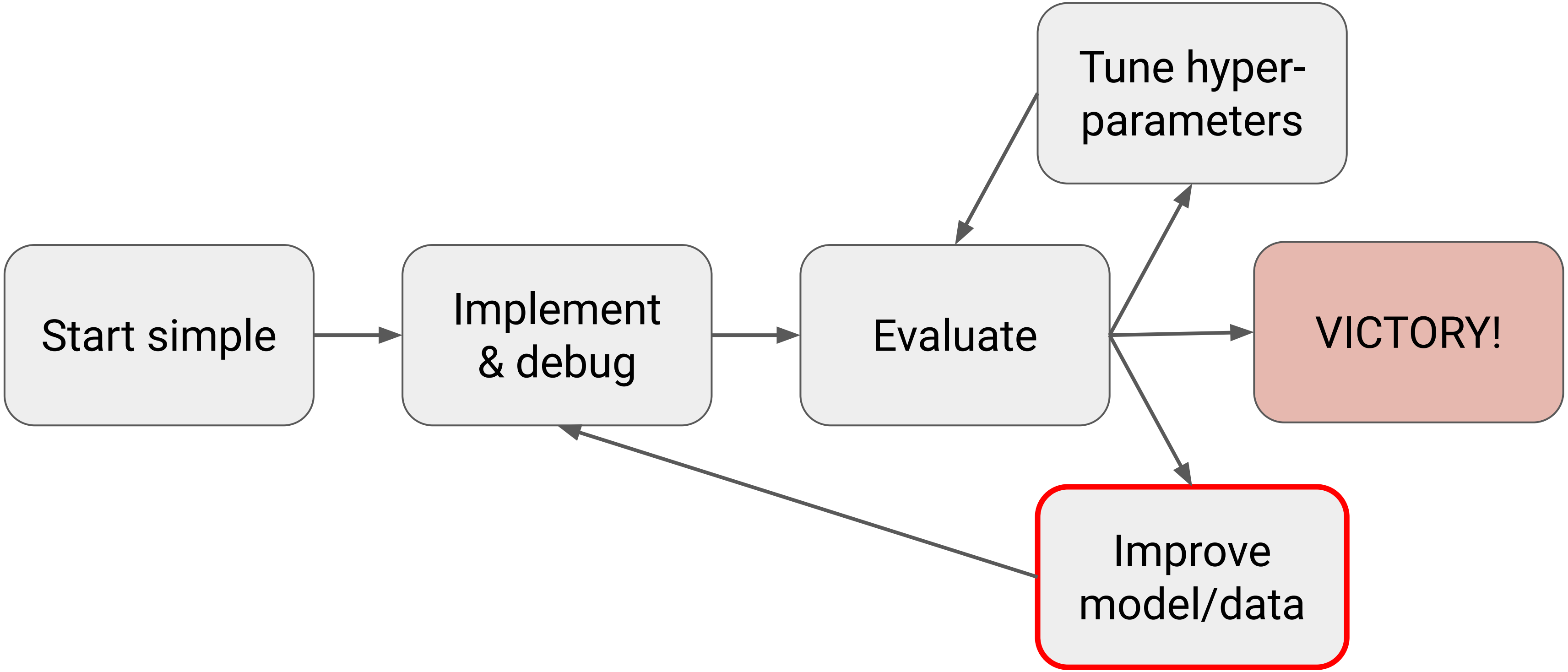
Val - Train= 7%
(over-fitting)

Bias-variance decomposition

Error source	Value
Goal performance	1%
Train error	20%
Validation error	27%
Test error	28%

Val - Test = 1%
(looks good!!)

Deep Learning recipe



Prioritizing improvements

1. Address under-fitting
2. Address over-fitting
3. Address distribution shift

Prioritizing improvements

1. Address under-fitting
2. Address over-fitting
3. Address distribution shift

Addressing under-fitting

1. Make your model bigger
2. Use pre-trained weights
3. Reduce regularization
4. Error analysis
5. Choose a different model architecture
6. Tune hyper-parameters
7. Add features

Bias-variance decomposition

Add more layers
to the ConvNet

Error source	~~Value~~	Value
Goal performance	1%	1%
Train error	20%	7%
Validation error	27%	19%
Test error	28%	20

Bias-variance decomposition

Switch to
ResNet-101

Error source	~~**Value**~~	~~**Value**~~	**Value**
Goal performance	1%	1%	1%
Train error	20%	7%	3%
Validation error	27%	19%	10%
Test error	28%	20%	10%

Bias-variance decomposition

Add learning
rate schedule

Error source	~~Value~~	~~Value~~	~~Value~~	Value
Goal performance	1%	1%	1%	1%
Train error	20%	7%	3%	0.8%
Validation error	27%	19%	10%	12%
Test error	28%	20%	10%	12%

Prioritizing improvements

1. Address under-fitting
2. Address over-fitting
3. Address distribution shift

Addressing over-fitting

1. Add more training data (if possible!)
2. Add normalization (e.g., batch norm)
3. Add data augmentation
4. Increase regularization (e.g., dropout, L2)
5. Error analysis
6. Choose a different model
7. Tune hyperparameters
8. Early stopping
9. Remove features
10. Reduce model size

Addressing over-fitting

1. Add more training data (if possible!)
2. Add normalization (e.g., batch norm)
3. Add data augmentation
4. Increase regularization (e.g., dropout, L2)
5. Error analysis
6. Choose a different model
7. Tune hyperparameters
8. Early stopping
9. Remove features
10. Reduce model size

Bias-variance decomposition

Increase dataset

Error source	~~Value~~	Value
Goal performance	1%	1%
Train error	0.8%	1.5%
Validation error	12%	5%
Test error	12%	6%

Bias-variance decomposition

Add
weight decay

Error source	~~Value~~	~~Value~~	Value
Goal performance	1%	1%	1%
Train error	20%	1.5%	1.7%
Validation error	27%	5%	4%
Test error	28%	6%	4%

Bias-variance decomposition

Add data augmentation

Error source	~~Value~~	~~Value~~	~~Value~~	Value
Goal performance	1%	1%	1%	1%
Train error	20%	7%	1.7%	0.8%
Validation error	27%	19%	4%	2.5%
Test error	28%	20%	4%	2.6%

Bias-variance decomposition

Tune num layers, optimizer params, weight initialization, kernel size, weight decay

Error source	~~**Value**~~	~~**Value**~~	~~**Value**~~	~~**Value**~~	**Value**
Goal performance	1%	1%	1%	1%	1%
Train error	20%	7%	1.7%	0.8%	0.6%
Validation error	27%	19%	4%	2.5%	0.9%
Test error	28%	20%	4%	2.6%	**1.0%**

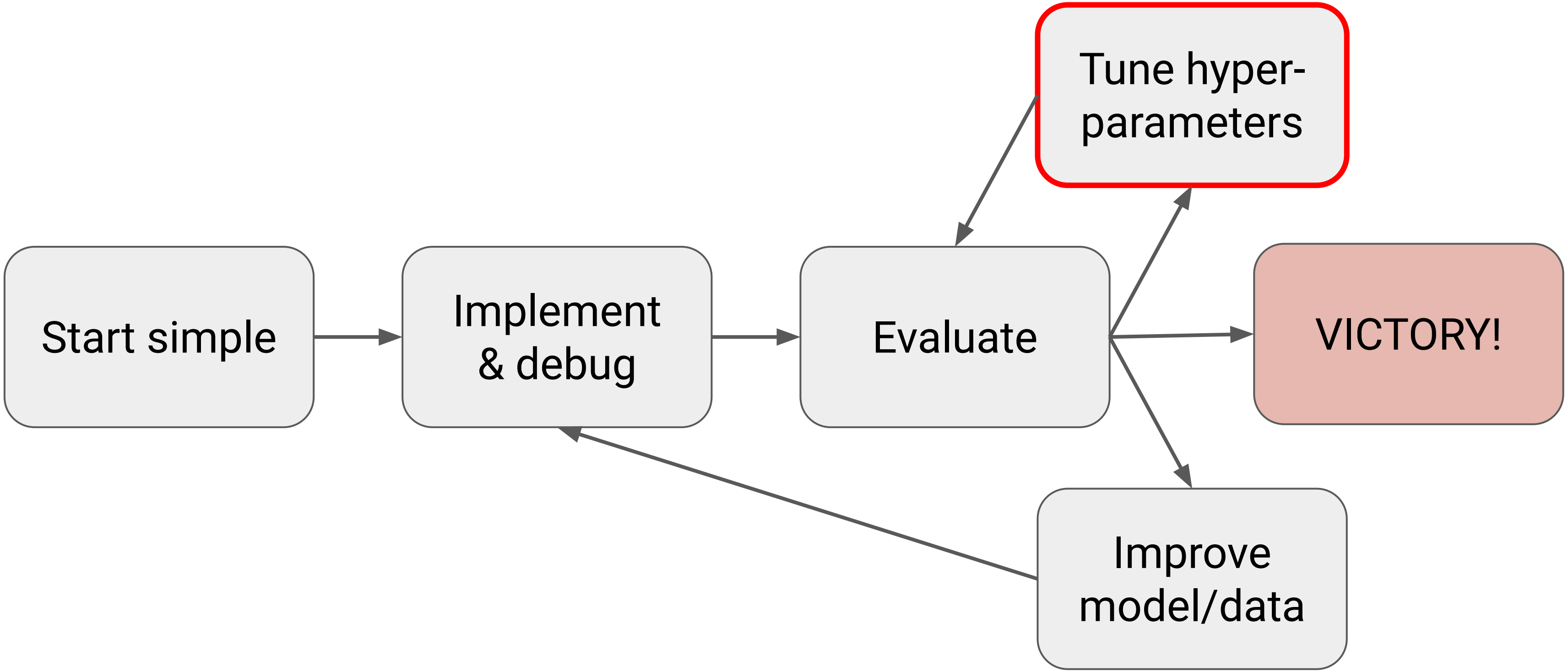
Prioritizing improvements

1. Address under-fitting
2. Address over-fitting
3. Address distribution shift
    a. Analyze test-val set errors & collect more training data to compensate
    b. Analyze test-val set errors & synthesize more training data to compensate
    c. Apply domain adaptation techniques to training & test distributions

Track your experiments!!

- Spreadsheet + TensorBoard:
  - Experiment id
  - **Git hash**
  - Model / dataset version
  - Hyperparameters
  - Metrics
  - Use JSON-like file in order to store parameters (I love gin-config)
  - Separate logbook to track ideas
- Weights & Biases
  - Common place for all the team
  - Create fancy reports
  - Nice integration with different frameworks
  - Transparency!
  - See example

Deep Learning recipe



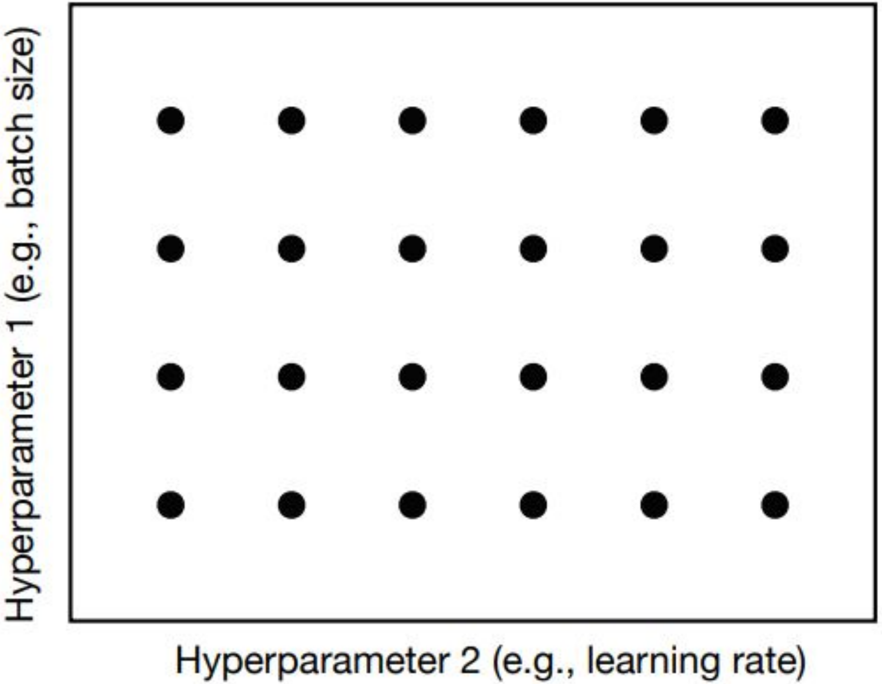
Which hyper-parameters to tune?

- Loss function
- Learning rate
- Learning rate schedule
- Layer size
- Weight initialization
- Model depth
- Layer params (e.g., kernel size)
- Weight of regularization
- Optimizer choice
- Other optimizer params (e.g., Adam beta1)
- Batch size
- Nonlinearity

Method 1: manual hyperparam optimization

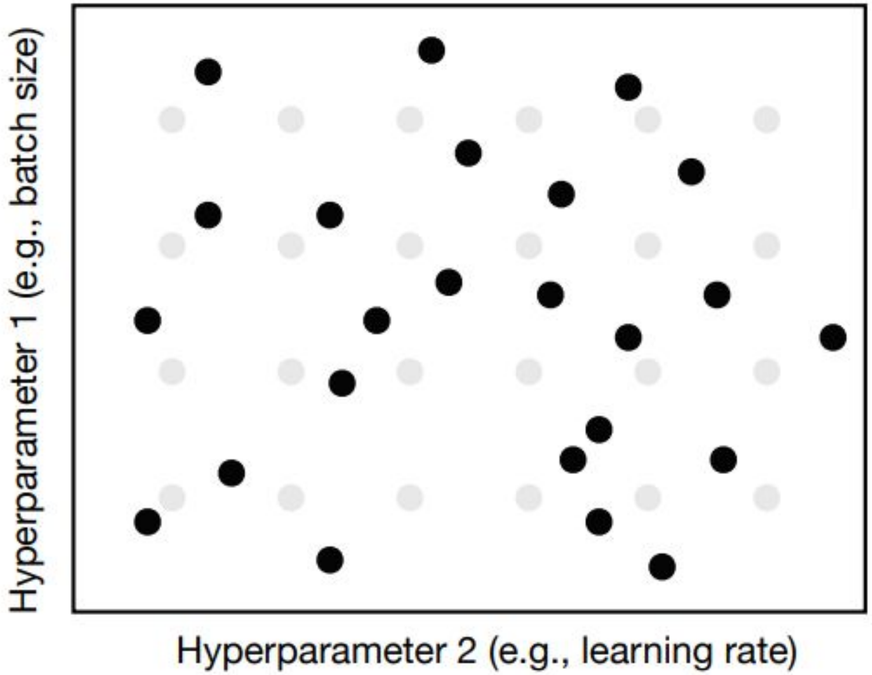
- Understand the algorithm
  - higher learning rate means faster less stable training
- Train & evaluate model
- Guess a better hyperparam value & reevaluate
- Pros:
  - For a skilled practitioner, may require least computation to get good result
- Cons:
  - Requires detailed understanding of the algorithm
  - Time-consuming

Method 2: grid search



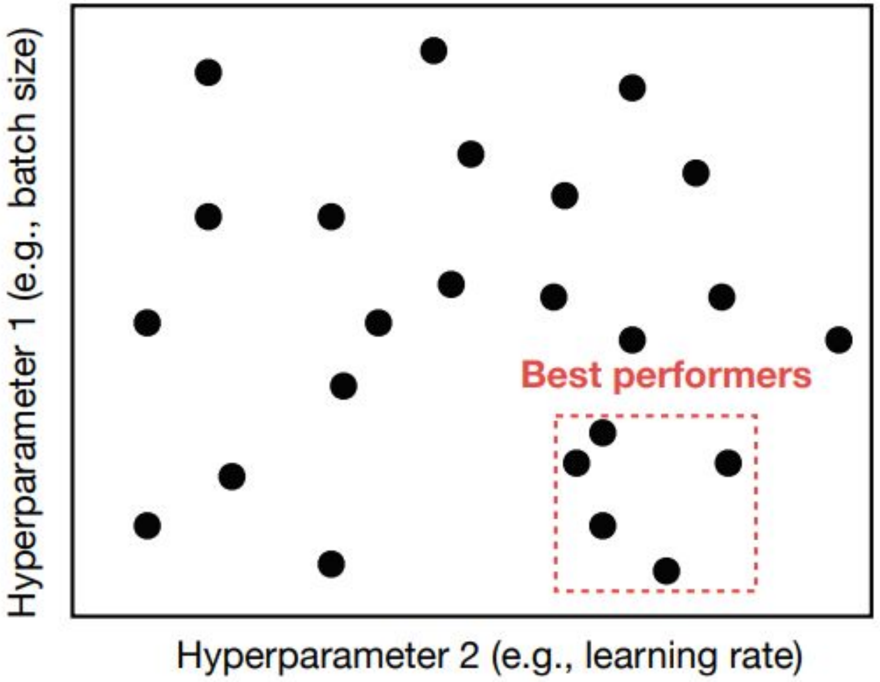
- Pros:
  - Super simple to implement
  - Can produce good results
- Cons:
  - Not very efficient: need to train on all cross-combos of hyper-parameters
  - May require prior knowledge about parameters to get good results

Method 3: random search



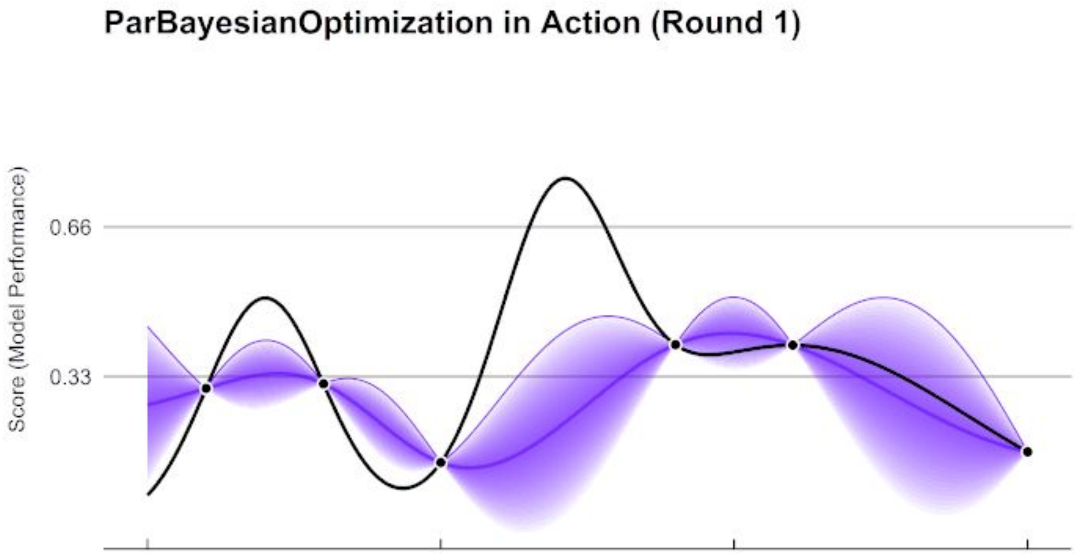
- Pros:
  - Easy to implement
  - Often produces better results than grid search
- Cons:
  - Not very interpretable
  - May require prior knowledge about parameters to get good results

Method 4: Grad student descent



- Pros:
  - Can narrow in on very high performing hyperparameters
  - Most used method in practice
- Cons:
  - Somewhat manual process

Method 5: Specialized methods



OPTUNA

Questions?

Thanks!

@shpotes