Troubleshooting Deep Neural Networks

A Field Guide to Fixing Your Model

Josh Tobin (with Sergey Karayev and Pieter Abbeel)

Help me make this guide better!

Help me find:

- Things that are unclear
- Missing debugging tips, tools, tricks, strategies
- Anything else that will make the guide better

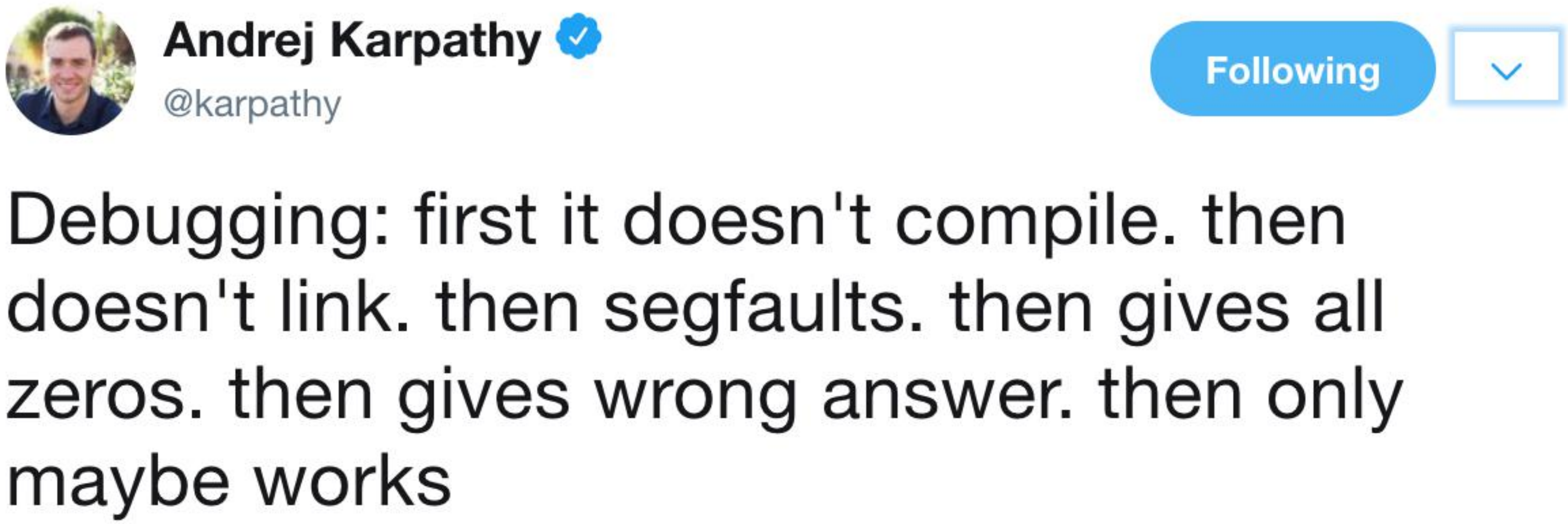
Feedback to:

- joshptobin [at] gmail.com
- Twitter thread (https://twitter.com/josh_tobin_)

Why talk about DL troubleshooting?

XKCD, https://xkcd.com/1838/

Why talk about DL troubleshooting?



Why talk about DL troubleshooting?

Common sentiment among practitioners:

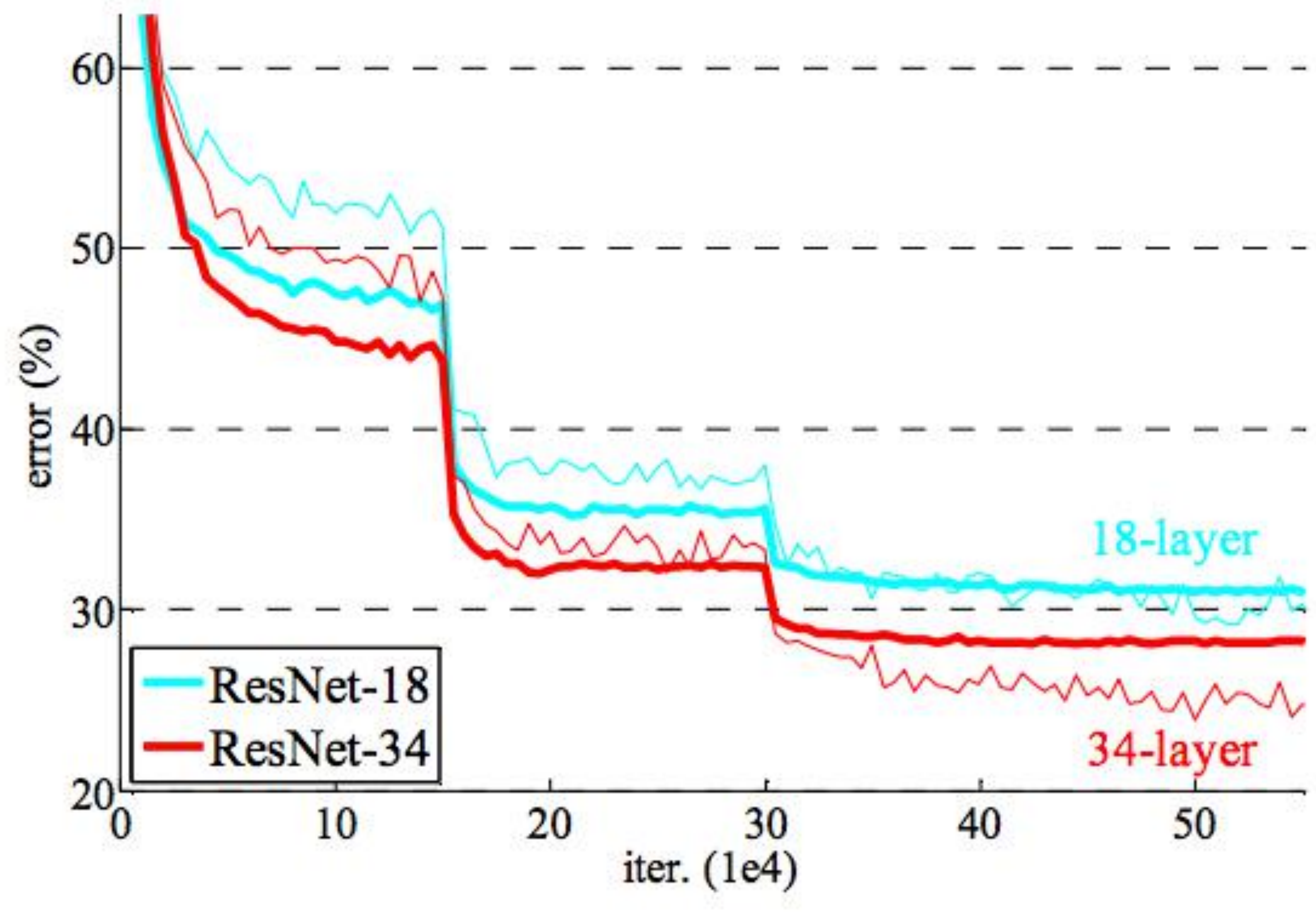
80-90% of time debugging and tuning

10-20% deriving math or implementing things

Why is DL troubleshooting so hard?

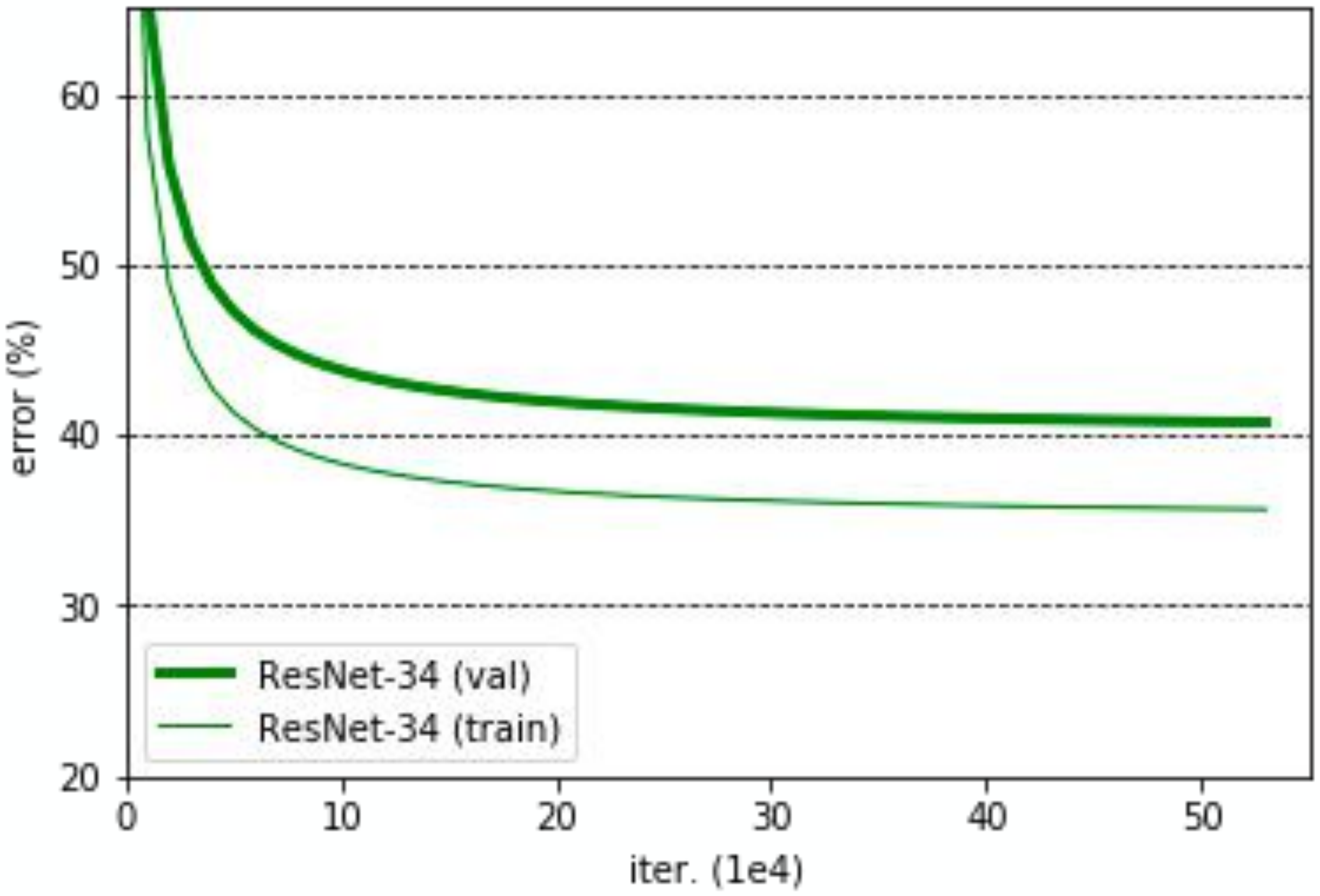
Suppose you can't reproduce a result

Learning curve from the paper



He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

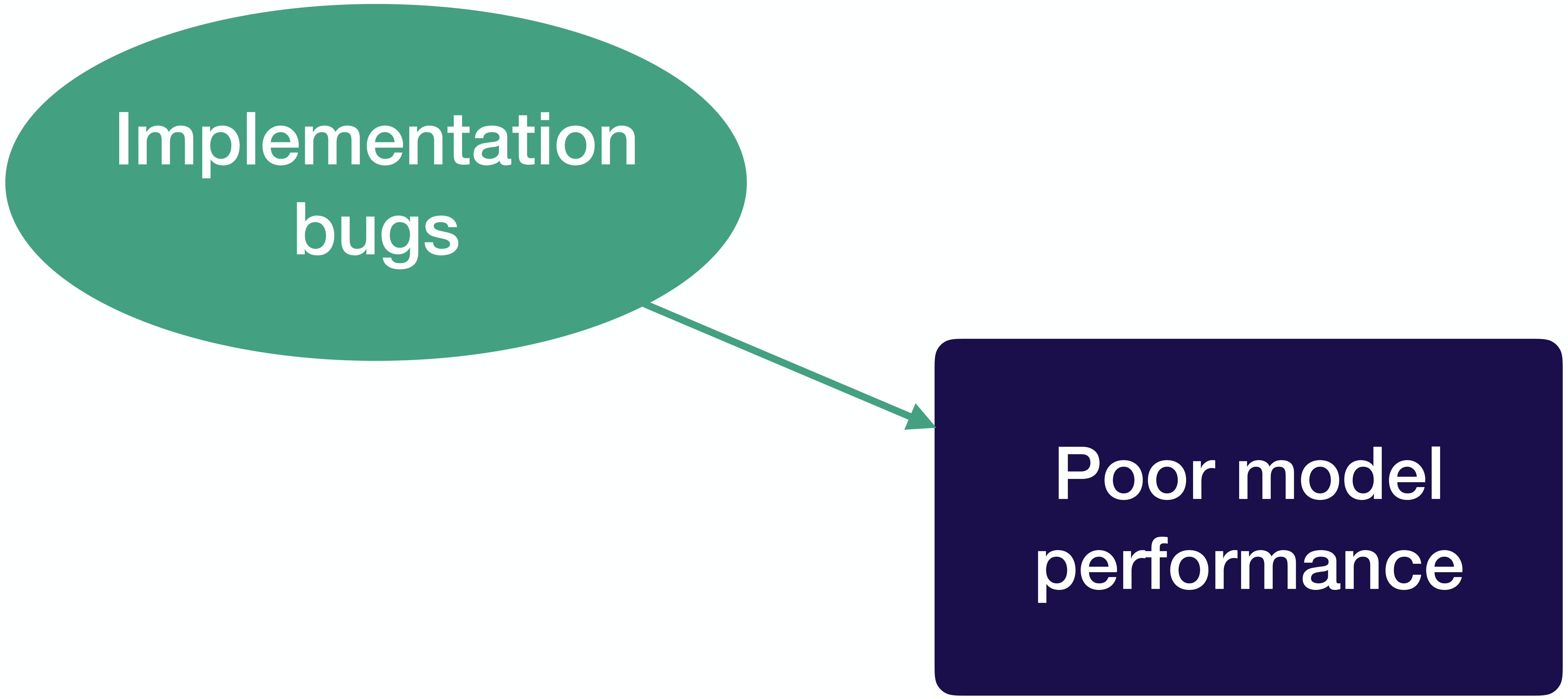
Your learning curve



Why is your performance worse?

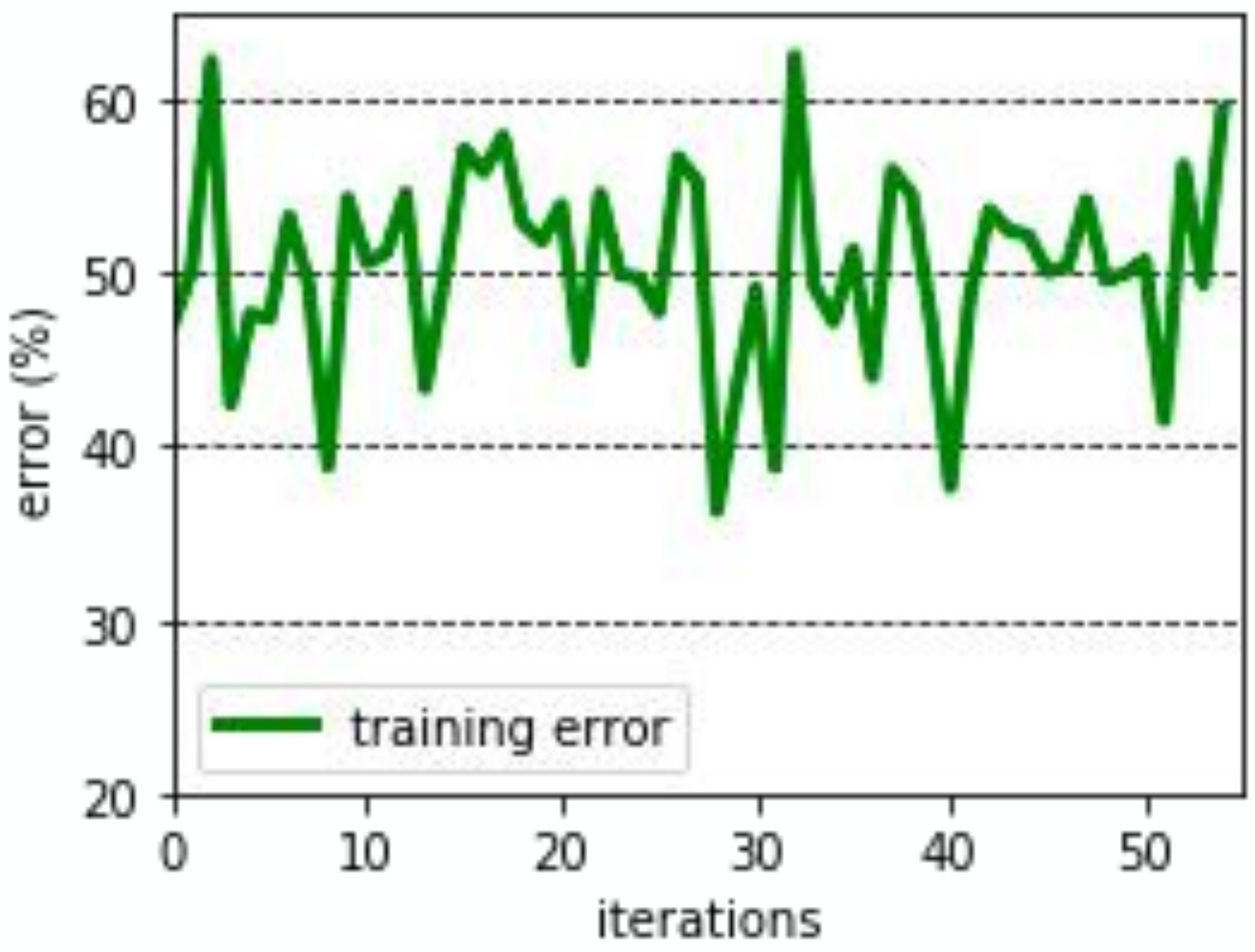
Poor model performance

Why is your performance worse?



Most DL bugs are invisible

```
features = glob.glob('path/to/features/*')
labels = glob.glob('path/to/labels/*')
train(features, labels)
```

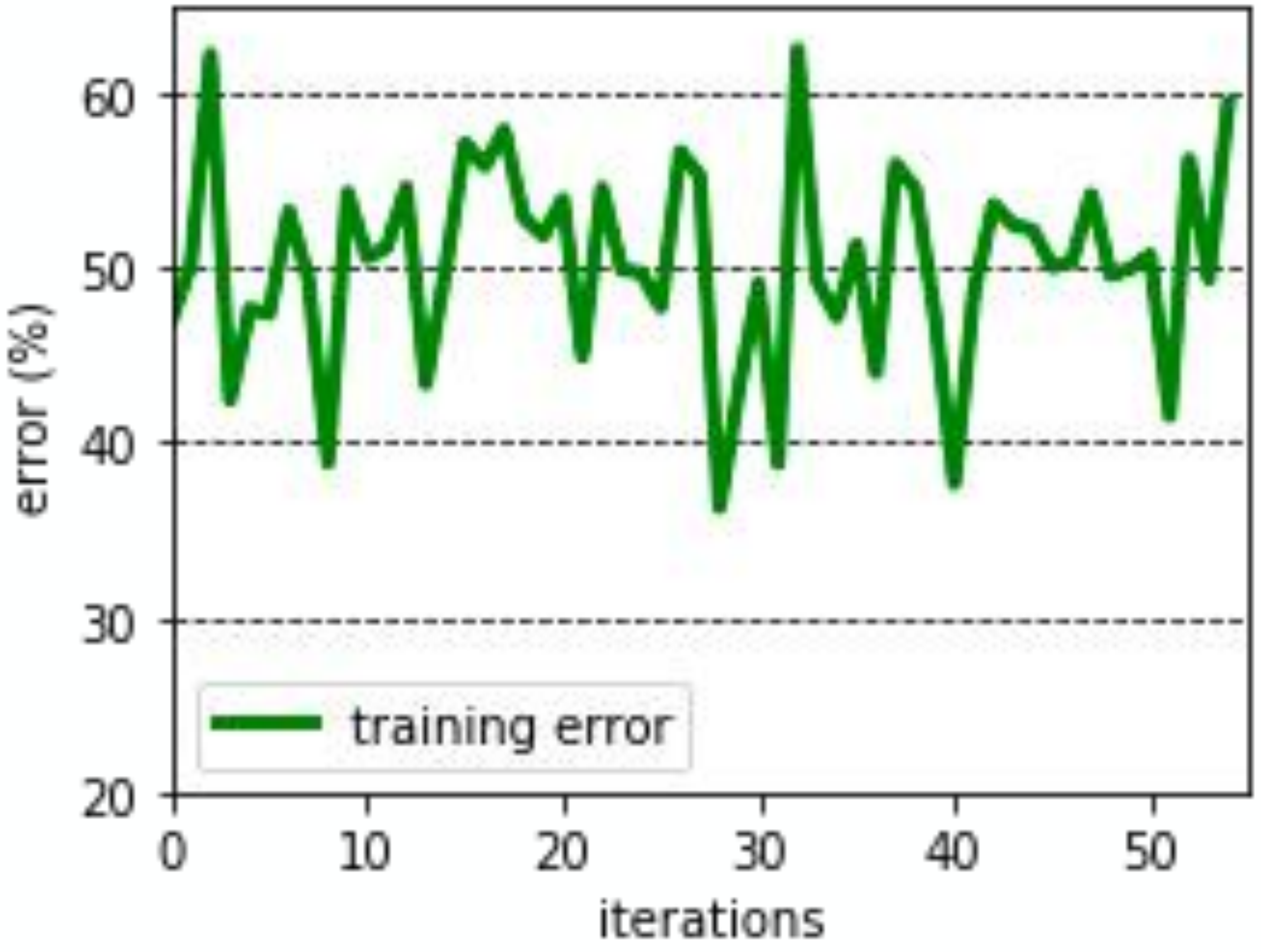


Most DL bugs are invisible

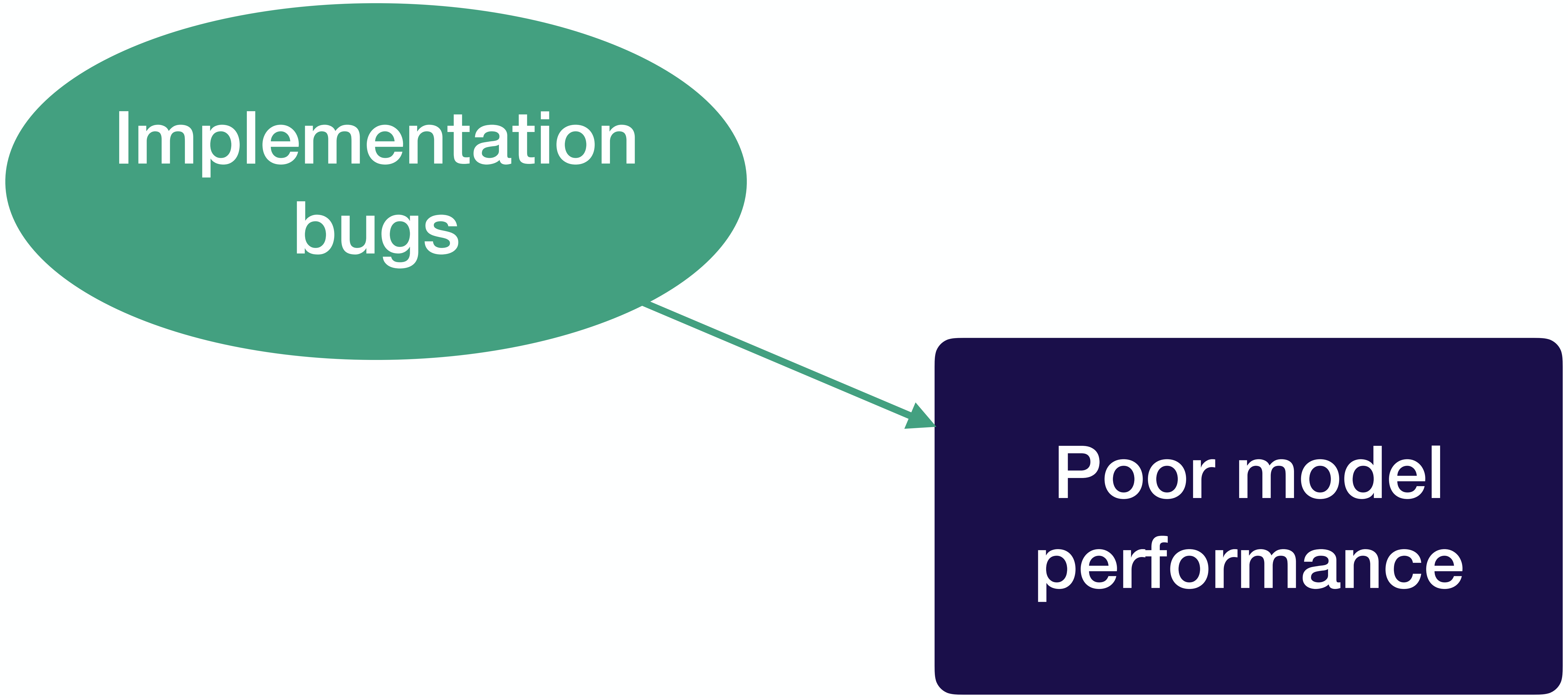
Labels out of order!

```
features = glob.glob('path/to/features/*')
labels = glob.glob('path/to/labels/*')
train(features, labels)
```

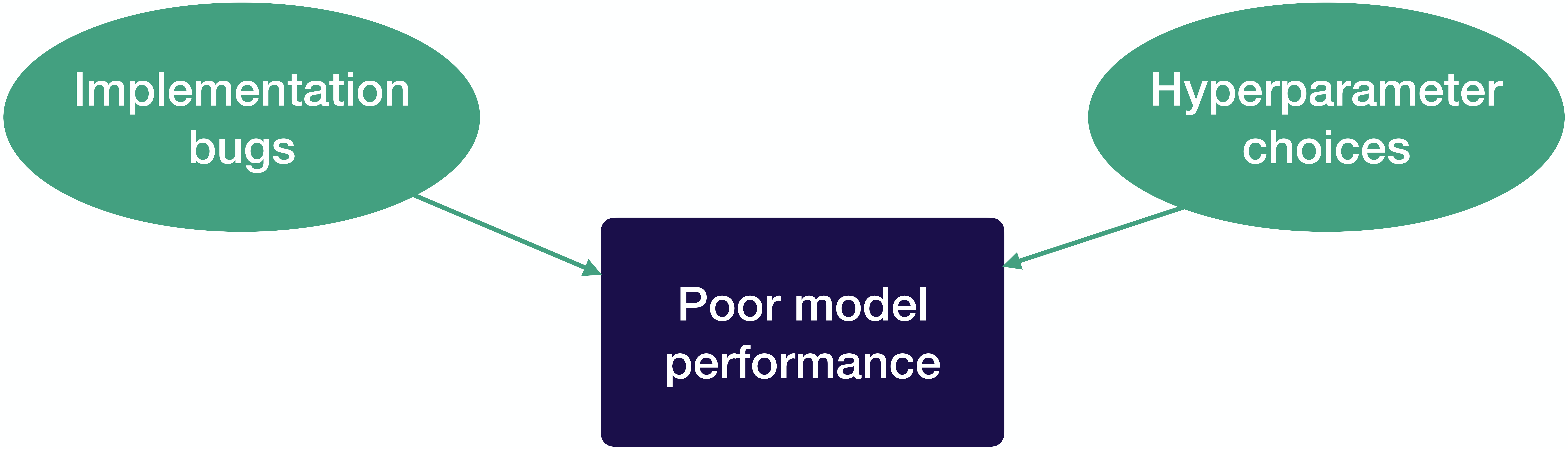
(real bug I spent 1 day on early in my PhD)



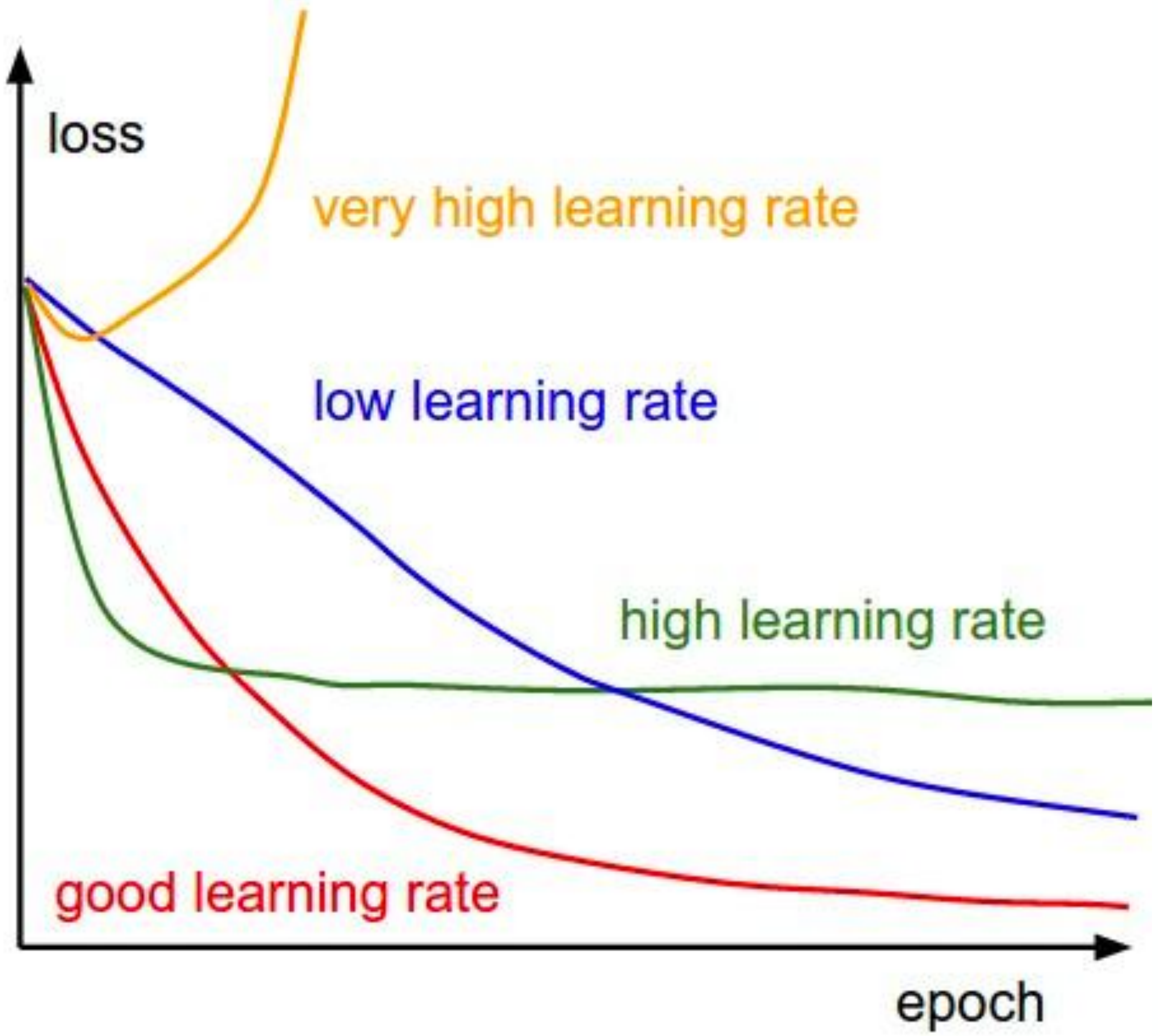
Why is your performance worse?



Why is your performance worse?

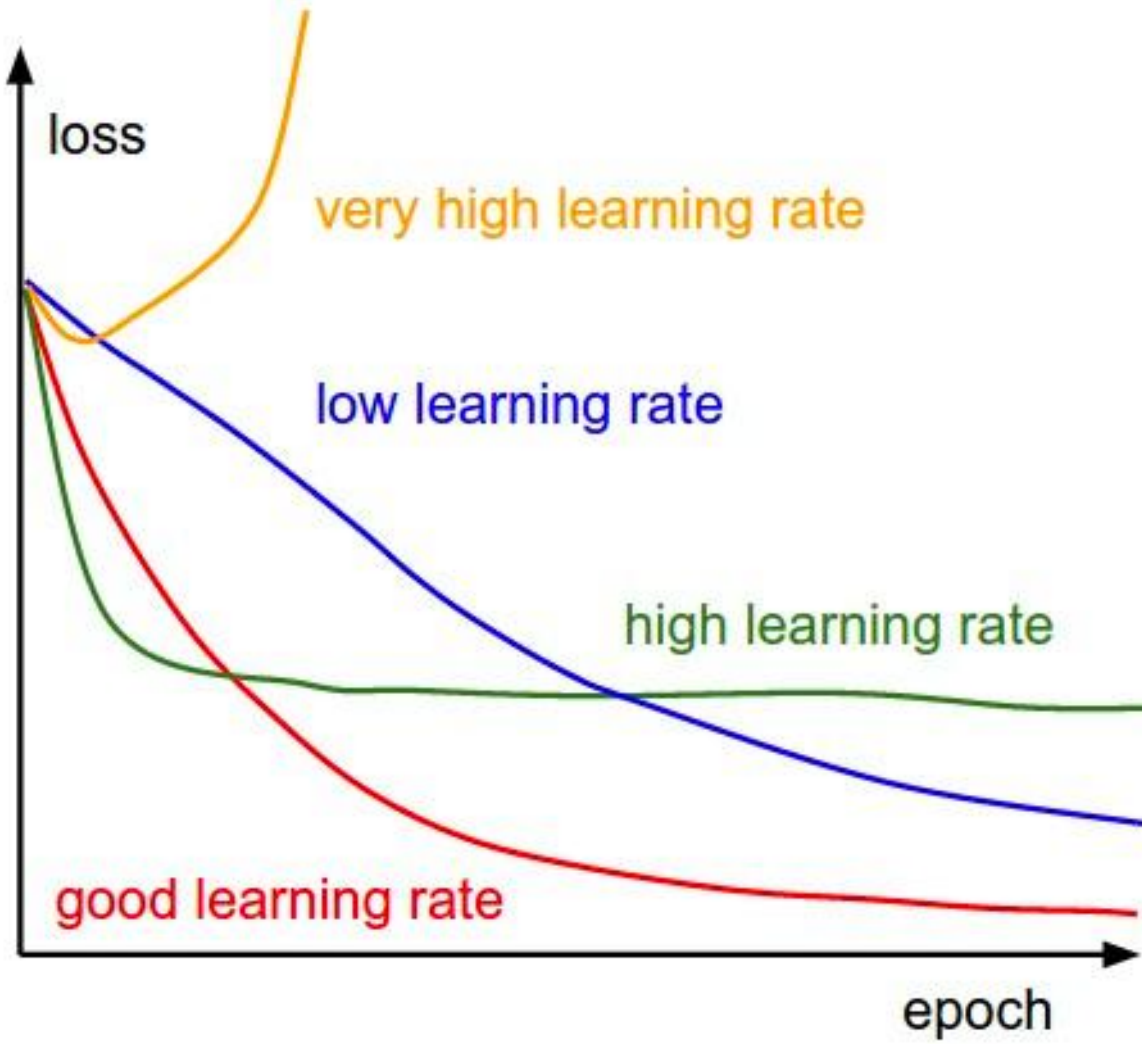


Models are sensitive to hyperparameters



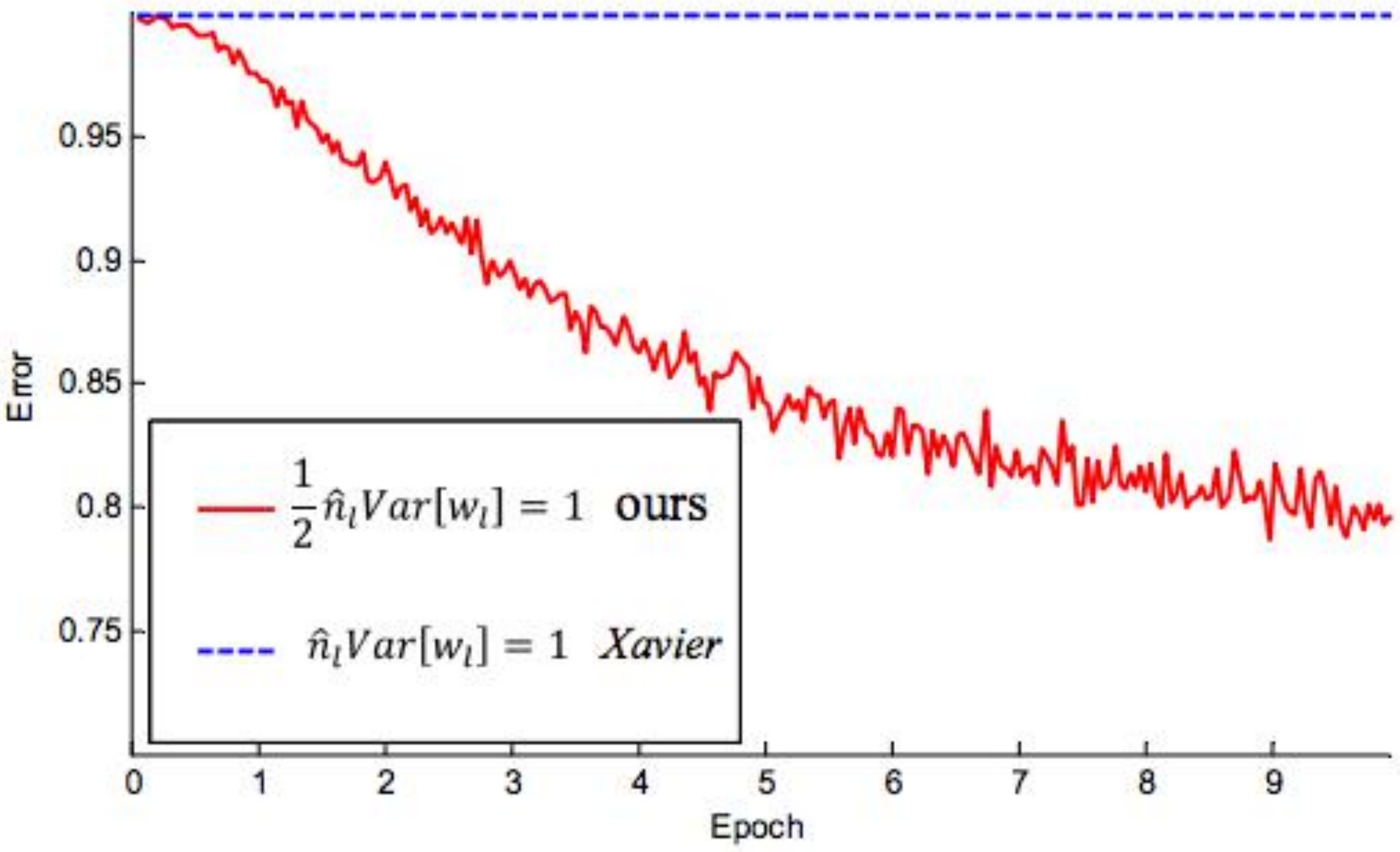
Andrej Karpathy, CS231n course notes

Models are sensitive to hyperparameters



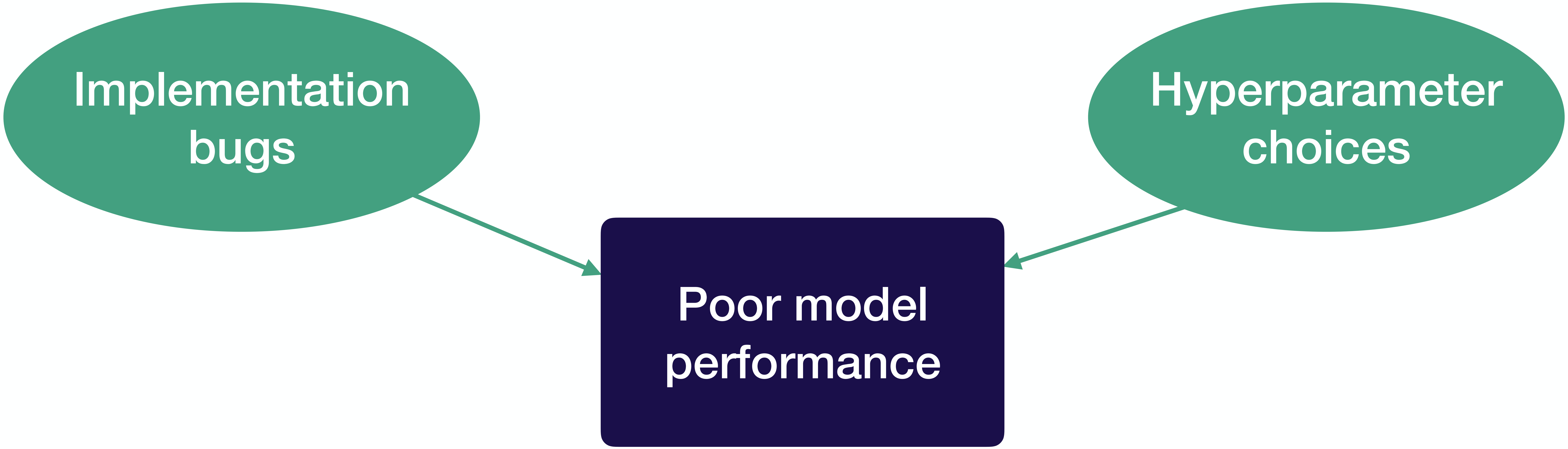
Andrej Karpathy, CS231n course notes

Performance of a 30-layer ResNet with different weight initializations

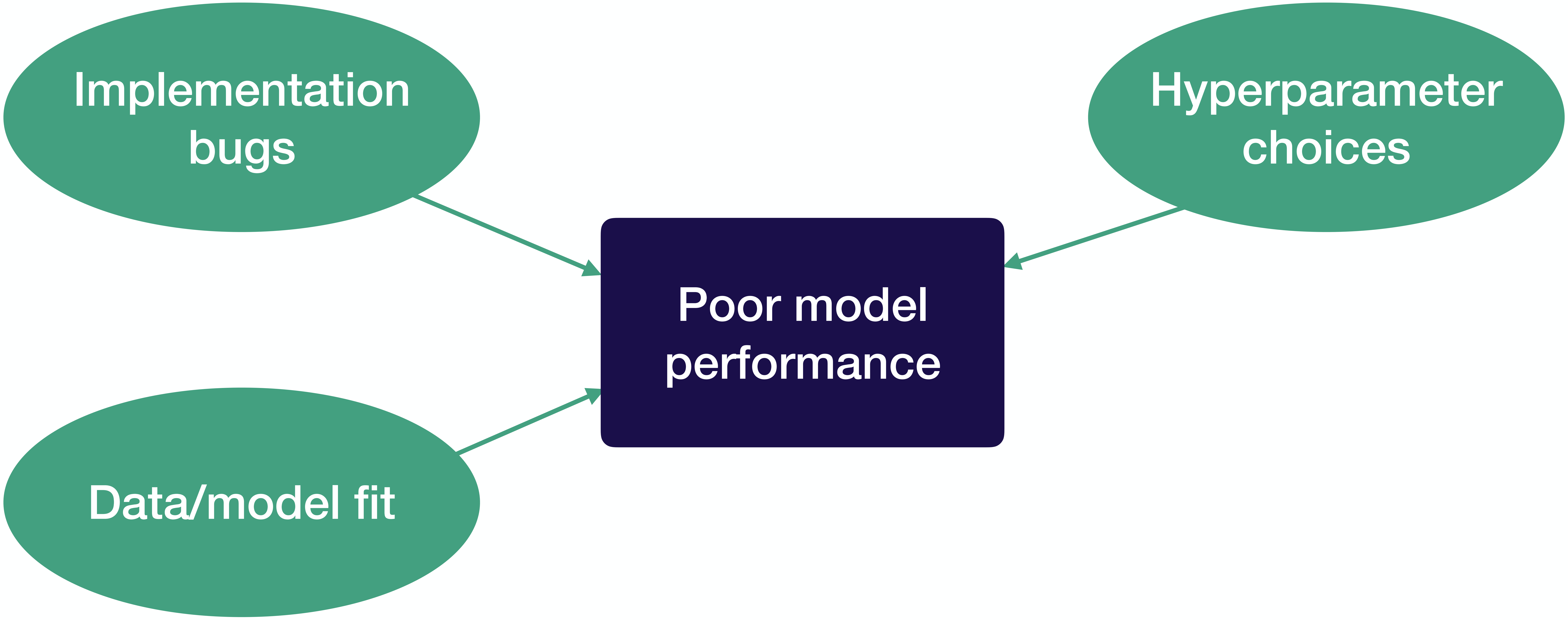


He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." Proceedings of the IEEE international conference on computer vision. 2015.

Why is your performance worse?

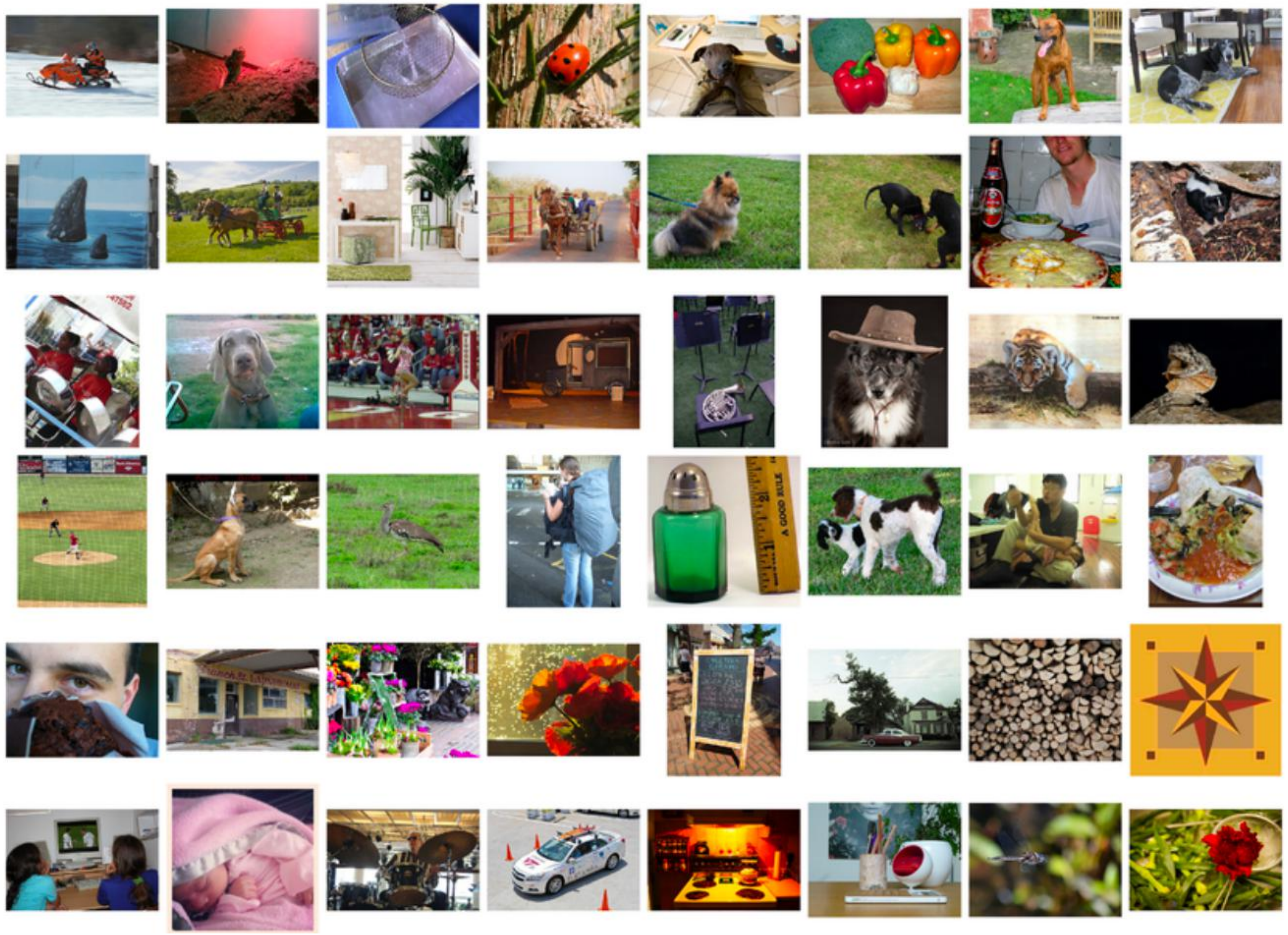


Why is your performance worse?



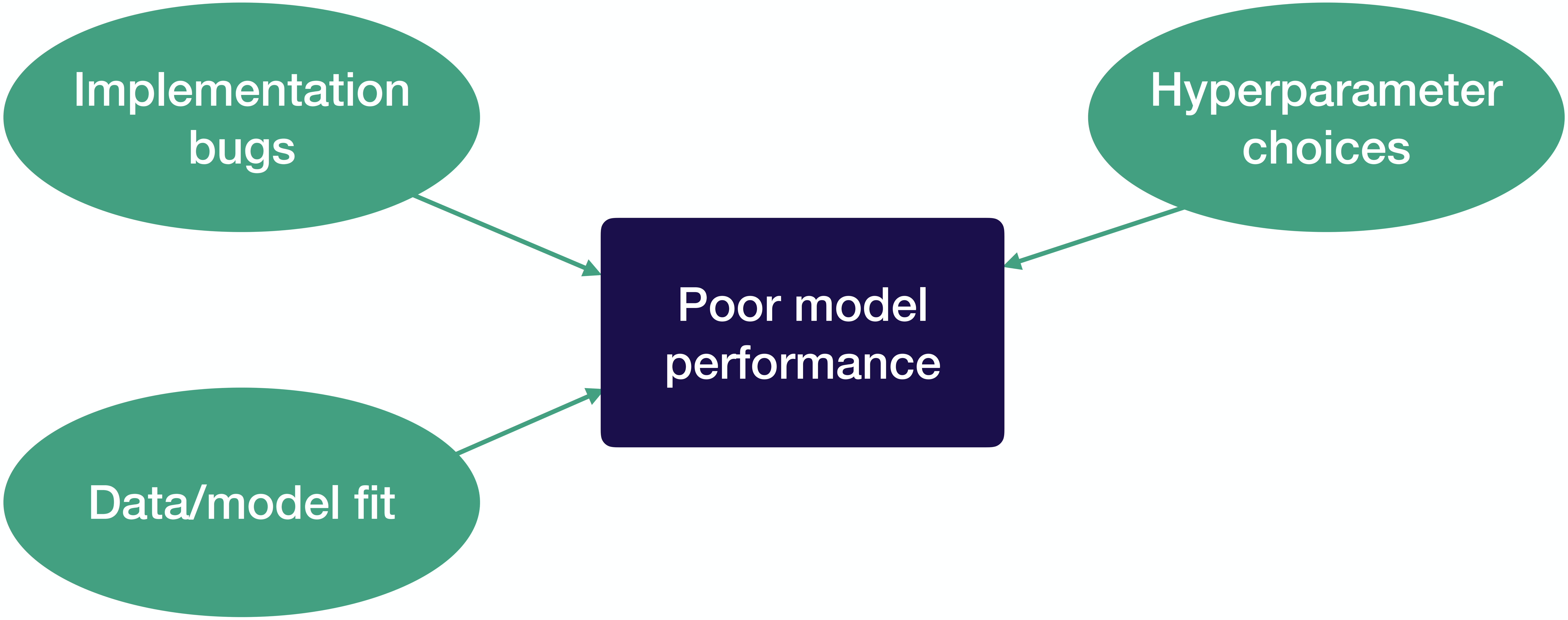
Data / model fit

Data from the paper: ImageNet

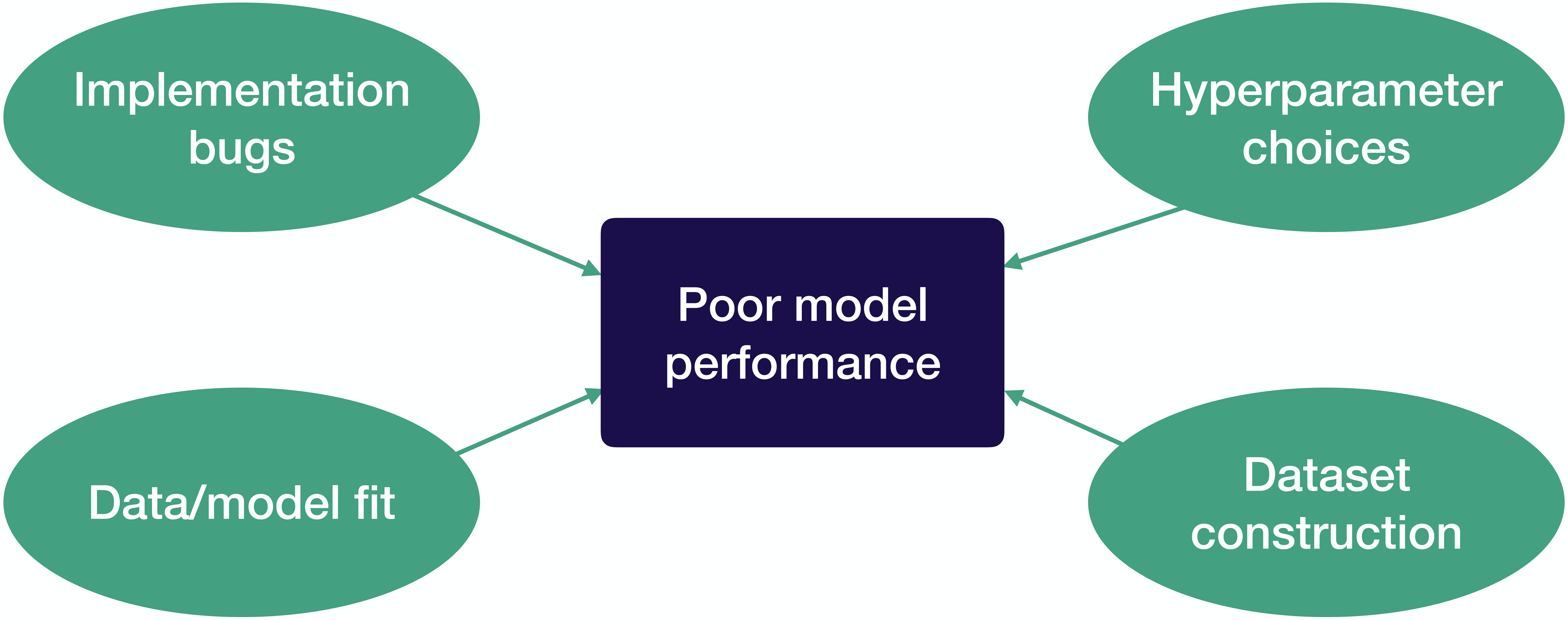


Yours: self-driving car images

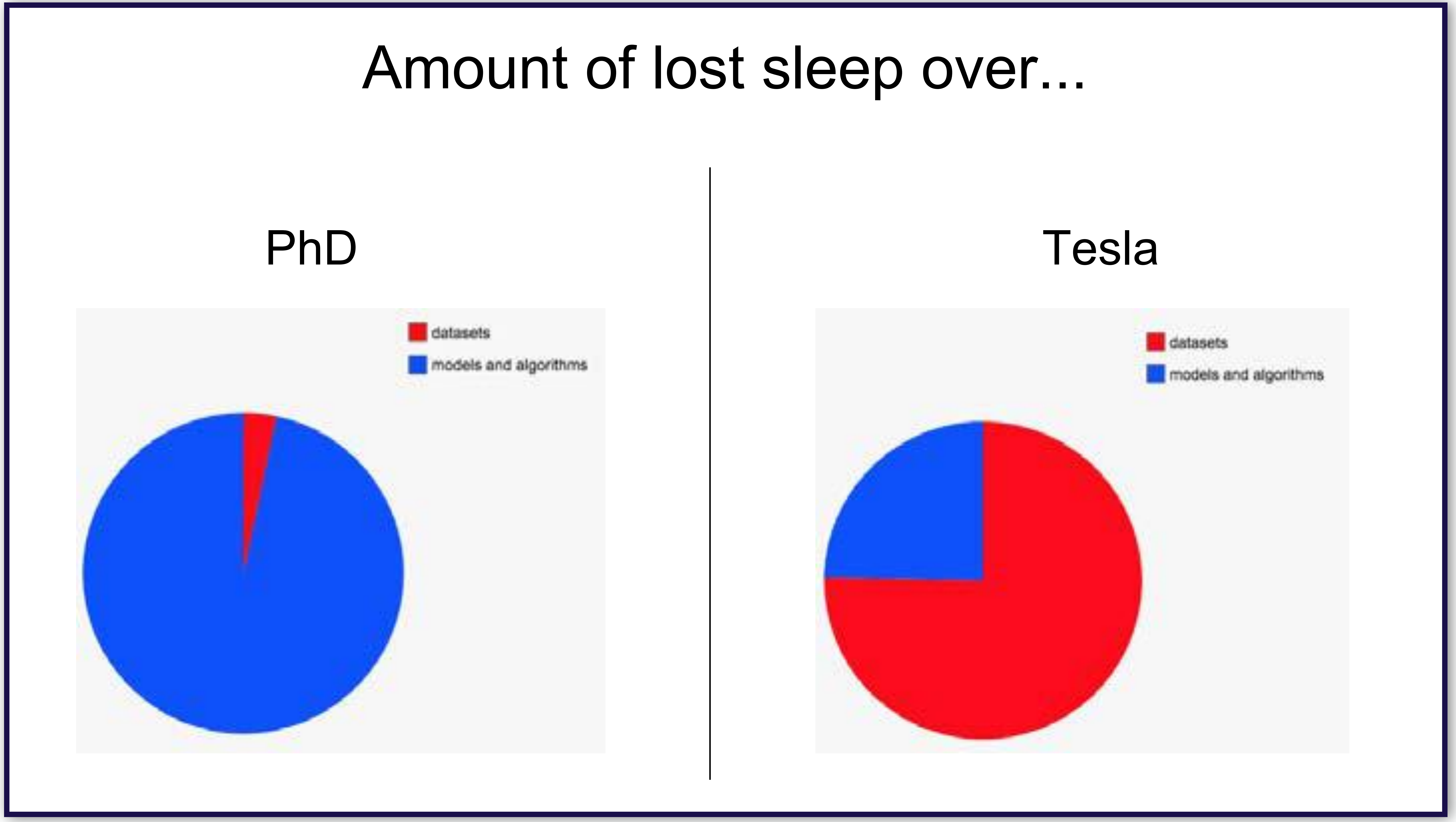
Why is your performance worse?



Why is your performance worse?



Constructing good datasets is hard



Slide from Andrej Karpathy's talk "Building the Software 2.0 Stack" at TrainAI 2018, 5/10/2018

Common dataset construction issues

- Not enough data
- Class imbalances
- Noisy labels
- Train / test from different distributions
- (Not the main focus of this guide)

Takeaways: why is troubleshooting hard?

- Hard to tell if you have a bug
- Lots of possible sources for the same degradation in performance
- Results can be sensitive to small changes in hyperparameters and dataset makeup

Strategy for DL troubleshooting

Key mindset for DL troubleshooting

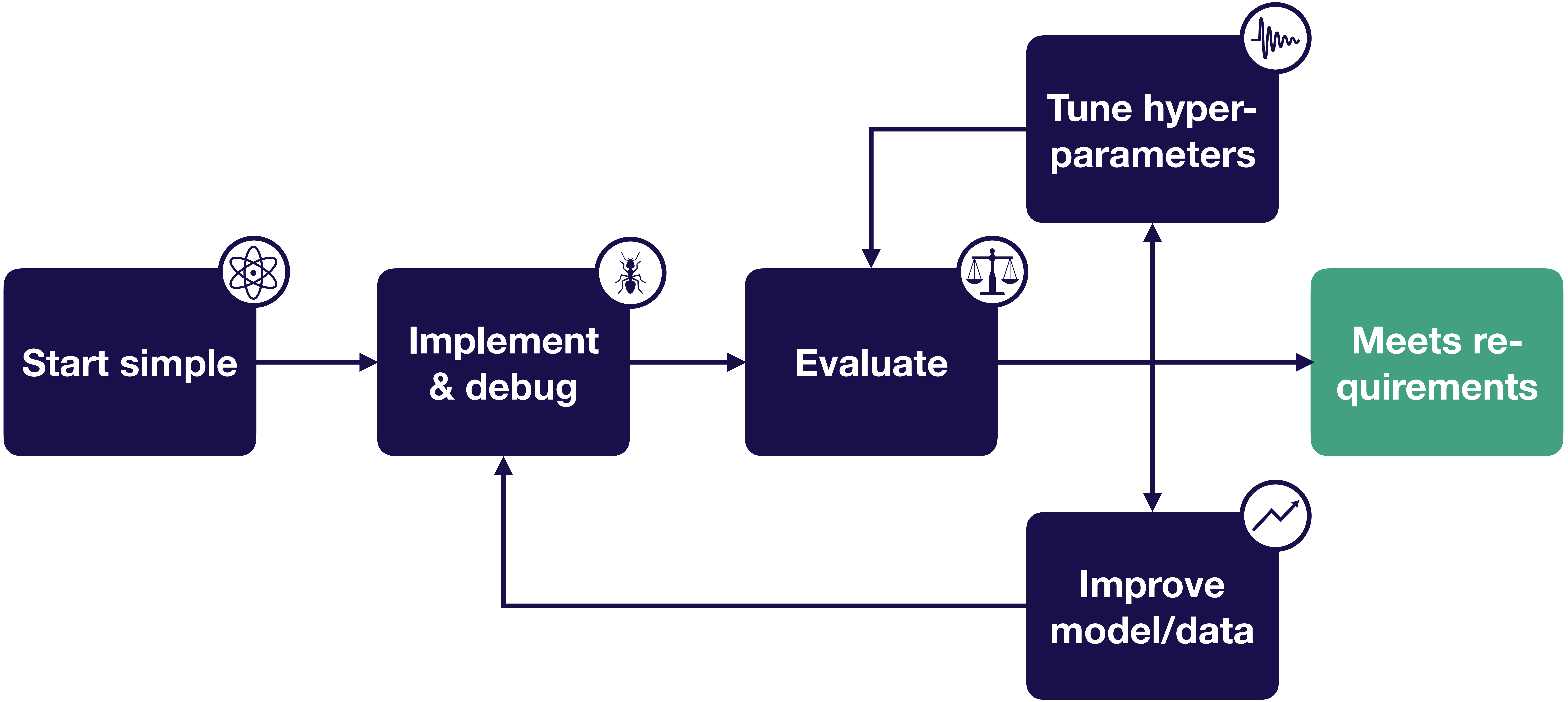
Pessimism.

Key idea of DL troubleshooting

Since it's hard to disambiguate errors…

…Start simple and gradually ramp up complexity

Strategy for DL troubleshooting



Quick summary

Start simple

Overview

- **Choose the simplest model & data possible (e.g., LeNet on a subset of your data)**

Quick summary

Overview

Start simple

- **Choose the simplest model & data possible (e.g., LeNet on a subset of your data)**

Implement & debug

- **Once model runs, overfit a single batch & reproduce a known result**

Quick summary

Overview

Start simple

- **Choose the simplest model & data possible (e.g., LeNet on a subset of your data)**

Implement & debug

- **Once model runs, overfit a single batch & reproduce a known result**

Evaluate

- **Apply the bias-variance decomposition to decide what to do next**

Quick summary

	Overview
Start simple	• Choose the simplest model & data possible (e.g., LeNet on a subset of your data)
Implement & debug	• Once model runs, overfit a single batch & reproduce a known result
Evaluate	• Apply the bias-variance decomposition to decide what to do next
Tune hyp-eparams	• Use coarse-to-fine random searches

Quick summary

Overview

Start simple	• Choose the simplest model & data possible (e.g., LeNet on a subset of your data)
Implement & debug	• Once model runs, overfit a single batch & reproduce a known result
Evaluate	• Apply the bias-variance decomposition to decide what to do next
Tune hyp-eparams	• Use coarse-to-fine random searches
Improve model/data	• Make your model bigger if you underfit; add data or regularize if you overfit

We'll assume you already have…

- Initial test set
- A single metric to improve
- Target performance based on human-level performance, published results, previous baselines, etc

We'll assume you already have…

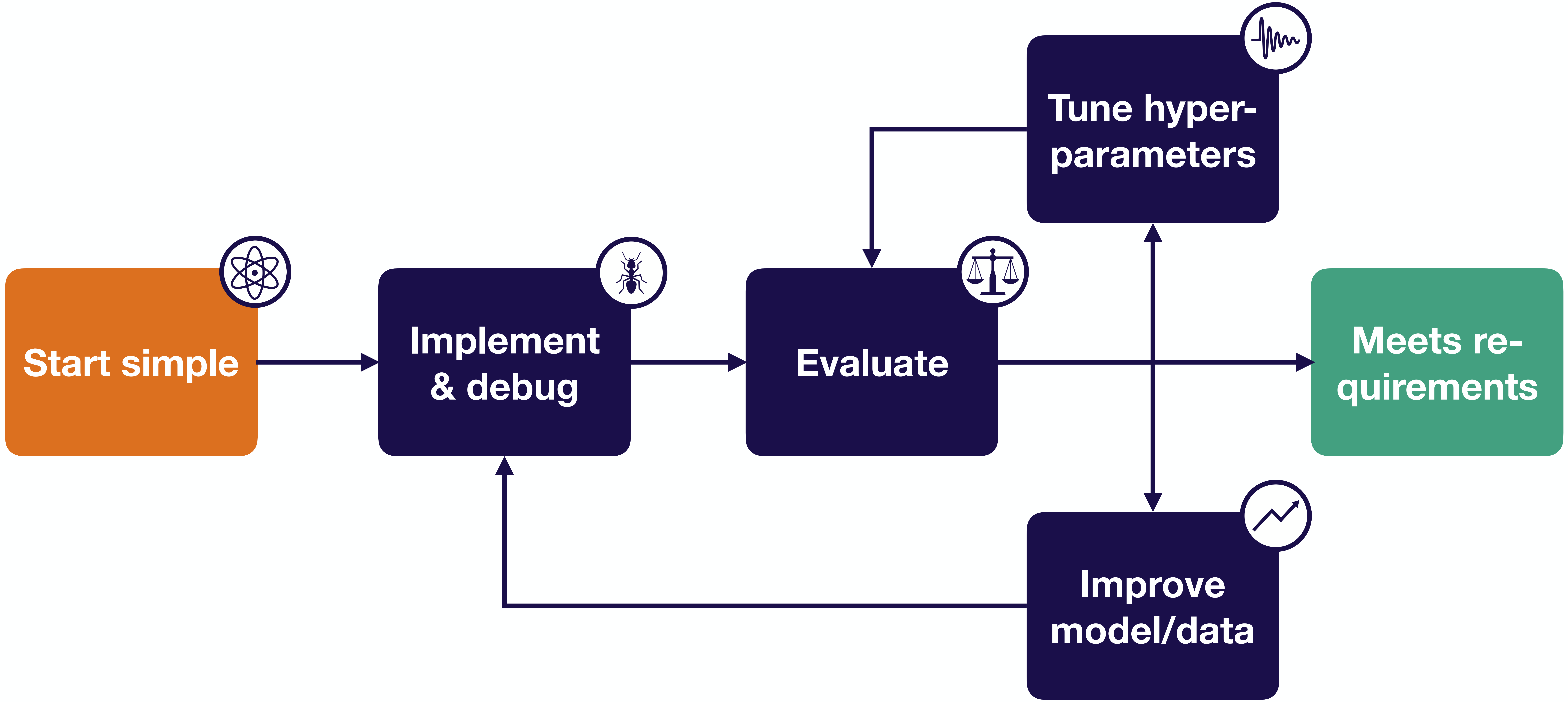
- Initial test set
- A single metric to improve
- Target performance based on human-level performance, published results, previous baselines, etc

Running example

0 (no pedestrian) **1 (yes pedestrian)**

Goal: 99% classification accuracy

Strategy for DL troubleshooting



Starting simple

Steps

- a. Choose a simple architecture
- b. Use sensible defaults
- c. Normalize inputs
- d. Simplify the problem

Demystifying neural network architecture selection

Your input data is…	Start here	Consider using this later
Images	LeNet-like architecture	ResNet
Sequences	LSTM with one hidden layer	Attention model or WaveNet-like model
Other	Fully connected neural net with one hidden layer	Problem-dependent

Dealing with multiple input modalities

Input 1

Input 2

Input 3

"This"

"is"

"a"

"cat"

Dealing with multiple input modalities

1. Map each input into a (lower-dimensional) feature space

Input 1

Input 2

Input 3

“This”

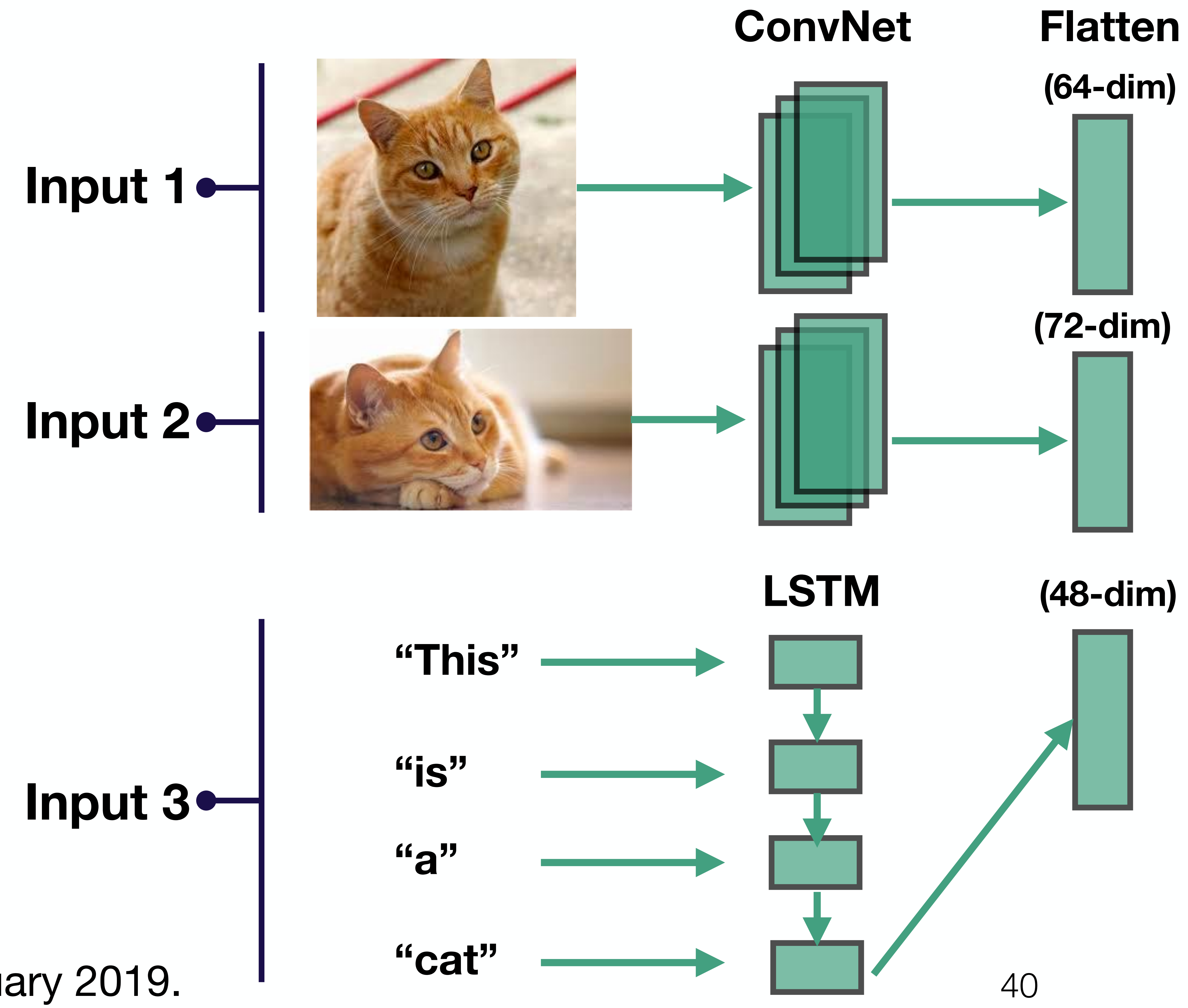
“is”

“a”

“cat”

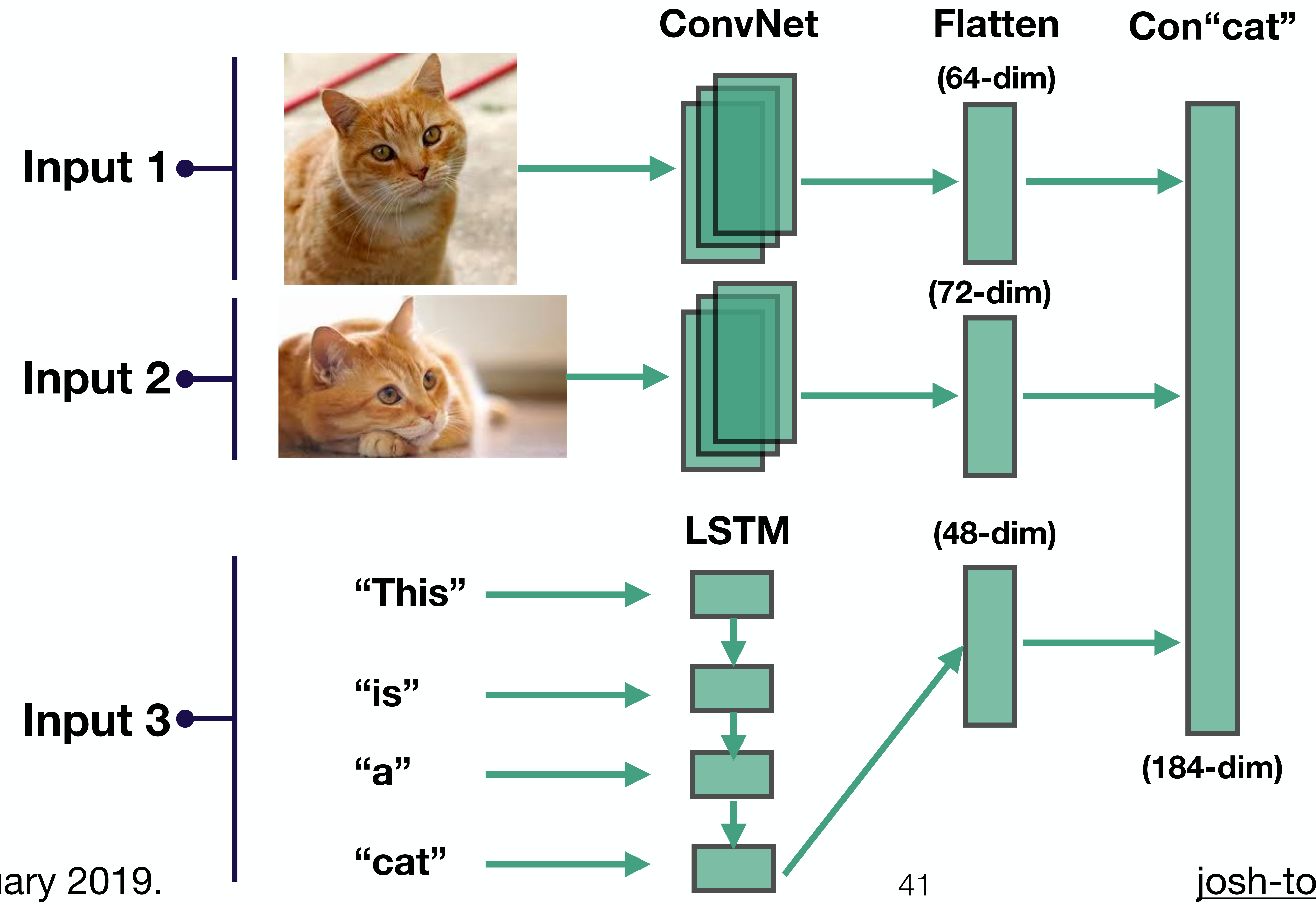
Dealing with multiple input modalities

1. Map each input into a (lower-dimensional) feature space



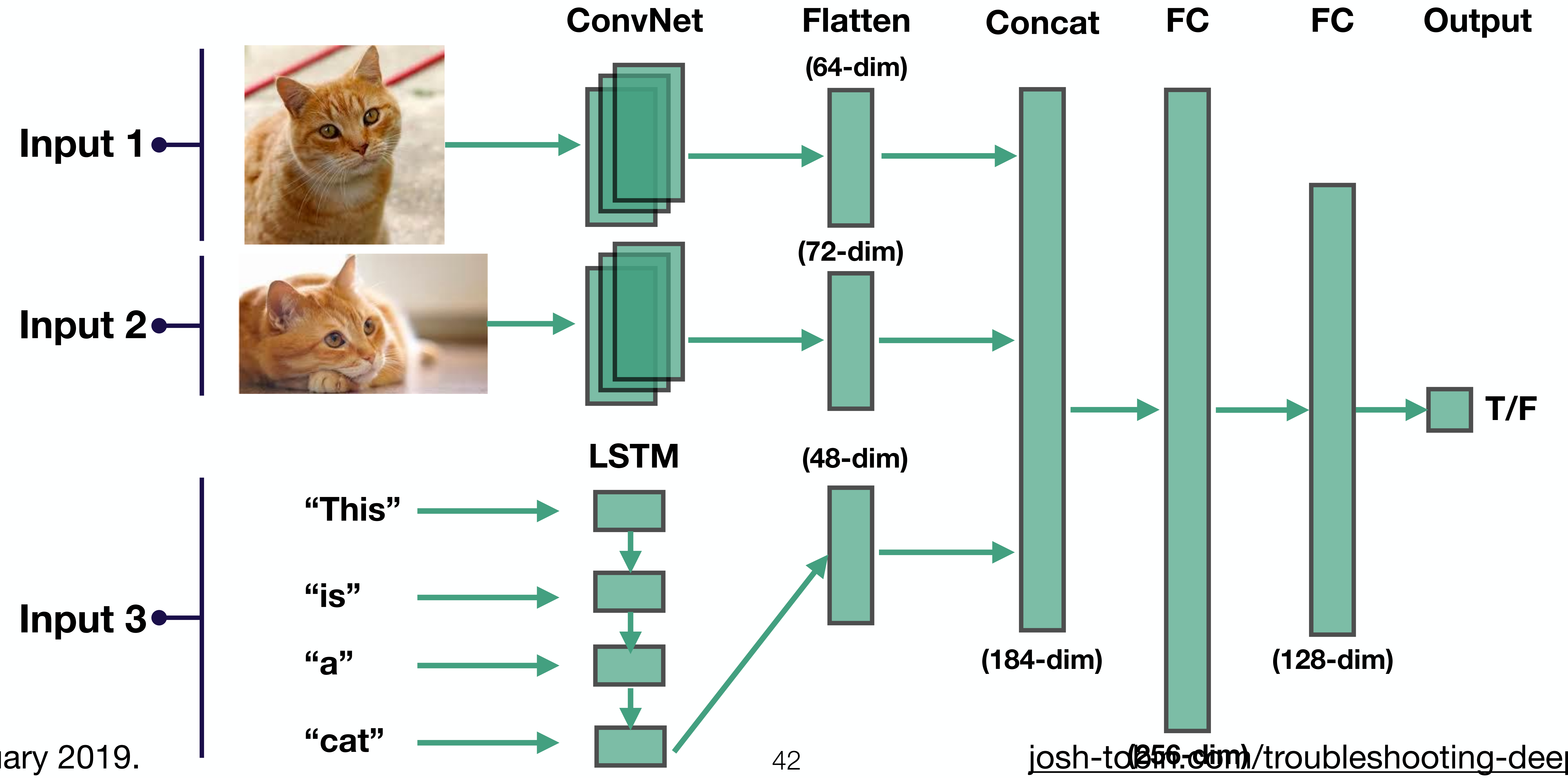
Dealing with multiple input modalities

2. Concatenate



Dealing with multiple input modalities

3. Pass through fully connected layers to output



Starting simple

Steps

- **a** Choose a simple architecture
- **b** Use sensible defaults
- **c** Normalize inputs
- **d** Simplify the problem

Recommended network / optimizer defaults

- **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
- **Data normalization**: None

Definitions of recommended initializers

- (n is the number of inputs, m is the number of outputs)

- He et al. normal (used for ReLU)

$$\mathcal{N}\left(0, \sqrt{\frac{2}{n}}\right)$$

- Glorot normal (used for tanh)

$$\mathcal{N}\left(0, \sqrt{\frac{2}{n+m}}\right)$$

Starting simple

Steps

a. Choose a simple architecture

b. Use sensible defaults

c. Normalize inputs

d. Simplify the problem

Important to normalize scale of input data

- Subtract mean and divide by variance
- For images, fine to scale values to [0, 1]
  (e.g., by dividing by 255)
  [Careful, make sure your library doesn't do it for you!]

Starting simple

Steps

- **a** Choose a simple architecture
- **b** Use sensible defaults
- **c** Normalize inputs
- **d** Simplify the problem

Consider simplifying the problem as well

- Start with a small training set (~10,000 examples)
- Use a fixed number of objects, classes, smaller image size, etc.
- Create a simpler synthetic training set

Simplest model for pedestrian detection

- Start with a subset of 10,000 images for training, 1,000 for val, and 500 for test
- Use a LeNet architecture with sigmoid cross-entropy loss
- Adam optimizer with LR 3e-4
- No regularization

Running example

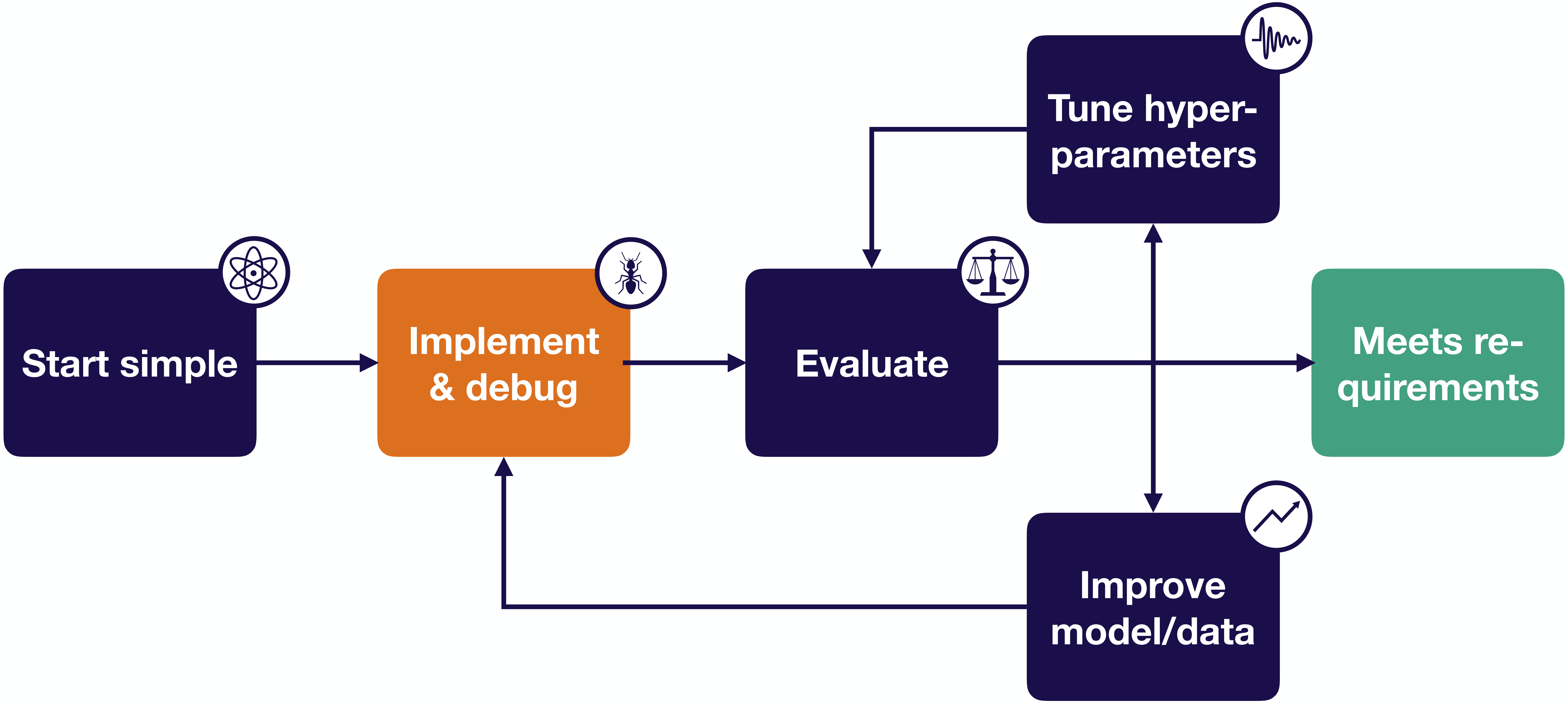
0 (no pedestrian) **1 (yes pedestrian)**

Goal: 99% classification accuracy

Starting simple

Steps	Summary
a. Choose a simple architecture	• LeNet, LSTM, or Fully Connected
b. Use sensible defaults	• Adam optimizer & no regularization
c. Normalize inputs	• Subtract mean and divide by std, or just divide by 255 (ims)
d. Simplify the problem	• Start with a simpler version of your problem (e.g., smaller dataset)

Strategy for DL troubleshooting



Implementing bug-free DL models

Steps

a. Get your model to run

b. Overfit a single batch

c. Compare to a known result

Preview: the five most common DL bugs

- **Incorrect shapes for your tensors**
  Can fail silently! E.g., accidental broadcasting: x.shape = (None,), y.shape = (None, 1), (x+y).shape = (None, None)
- **Pre-processing inputs incorrectly**
  E.g., Forgetting to normalize, or too much pre-processing
- **Incorrect input to your loss function**
  E.g., softmaxed outputs to a loss that expects logits
- **Forgot to set up train mode for the net correctly**
  E.g., toggling train/eval, controlling batch norm dependencies
- **Numerical instability - inf/NaN**
  Often stems from using an exp, log, or div operation

General advice for implementing your model

Lightweight implementation

- Minimum possible new lines of code for v1
- Rule of thumb: <200 lines
- (Tested infrastructure components are fine)

General advice for implementing your model

Lightweight implementation

- Minimum possible new lines of code for v1
- Rule of thumb: <200 lines
- (Tested infrastructure components are fine)

Use off-the-shelf components, e.g.,

- Keras
- `tf.layers.dense(…)` instead of `tf.nn.relu(tf.matmul(W, x))`
- `tf.losses.cross_entropy(…)` instead of writing out the exp

General advice for implementing your model

Lightweight implementation

- Minimum possible new lines of code for v1
- Rule of thumb: <200 lines
- (Tested infrastructure components are fine)

Use off-the-shelf components, e.g.,

- Keras
- `tf.layers.dense(…)` instead of `tf.nn.relu(tf.matmul(W, x))`
- `tf.losses.cross_entropy(…)` instead of writing out the exp

Build complicated data pipelines later

- Start with a dataset you can load into memory

Implementing bug-free DL models

Steps

a. Get your model to run

b. Overfit a single batch

c. Compare to a known result

Implementing bug-free DL models

a Get your model to run

Common issues	Recommended resolution
Shape mismatch	Step through model creation and inference in a debugger
Casting issue	Step through model creation and inference in a debugger
OOM	Scale back memory intensive operations one-by-one
Other	Standard debugging toolkit (Stack Overflow + interactive debugger)

Implementing bug-free DL models

Common issues	Recommended resolution
Shape mismatch	Step through model creation and inference in a debugger
Casting issue	
OOM	Scale back memory intensive operations one-by-one
Other	Standard debugging toolkit (Stack Overflow + interactive debugger)

a Get your model to run

Debuggers for DL code

- Pytorch: easy, use ipdb
- tensorflow: trickier

Option 1: step through graph creation

```python
# Option 1: step through graph creation
import ipdb; ipdb.set_trace()

for i in range(num_layers):
    out = layers.fully_connected(out, 50)

```

```
josh at MacBook-Pro-9 in ~/projects
$ python test.py
> /Users/josh/projects/test.py(5)<module>()
      3 h = tf.placeholder(tf.float32, (None, 100))
      4 import ipdb; ipdb.set_trace()
----> 5 w = tf.layers.dense(h)

ipdb>
```

Debuggers for DL code

- Pytorch: easy, use ipdb
- tensorflow: trickier

Option 2: step into training loop

```
# Option 2: step into training loop
sess = tf.Session()
for i in range(num_epochs):
    import ipdb; ipdb.set_trace()
    loss_, _ = sess.run([loss, train_op])

```

Evaluate tensors using `sess.run(…)`

Debuggers for DL code

- Pytorch: easy, use ipdb
- tensorflow: trickier

Option 3: use tfdb

```
python -m tensorflow.python.debug.examples.debug_mnist --debug
```

```
--- run-start: run #1: 1 fetch (accuracy/accuracy/Mean:0); 2 feeds ------------
| <-- --> | run_info
| run | invoke_stepper | exit |
                                                                          UP
TTTTTT FFFF DDD  BBBB   GGG
  TT   F    D  D B   B G
  TT   FFF  D  D BBBB  G  GG
  TT   F    D  D B   B G   G
  TT   F    DDD  BBBB   GGG

======================================
Session.run() call #1:

Fetch(es):
  accuracy/accuracy/Mean:0

Feed dict(s):
  input/x-input:0
  input/y-input:0
======================================

Select one of the following commands to proceed ---->
  run:
    Execute the run() call with debug tensor-watching
  run -n:
    Execute the run() call without debug tensor-watching                  DN
--- Scroll (PgDn): 0.00% -------------------------------------- Mouse: ON --
tfdbg>
```

Stops execution at each `sess.run(…)` **and lets you inspect**

Implementing bug-free DL models

a Get your model to run

Common issues	Recommended resolution
Shape mismatch	Step through model creation and inference in a debugger
Casting issue	Step through model creation and inference in a debugger
OOM	Scale back memory intensive operations one-by-one
Other	Standard debugging toolkit (Stack Overflow + interactive debugger)

Implementing bug-free DL models

Shape mismatch

Common issues	Most common causes
Undefined shapes	• Confusing tensor.shape, tf.shape(tensor), tensor.get_shape() • Reshaping things to a shape of type Tensor (e.g., when loading data from a file)
Incorrect shapes	• Flipped dimensions when using tf.reshape(…) • Took sum, average, or softmax over wrong dimension • Forgot to flatten after conv layers • Forgot to get rid of extra "1" dimensions (e.g., if shape is (None, 1, 1, 4) • Data stored on disk in a different dtype than loaded (e.g., stored a float64 numpy array, and loaded it as a float32)

Implementing bug-free DL models

Common issues	Most common causes
Data not in float32	• Forgot to cast images from uint8 to float32 • Generated data using numpy in float64, forgot to cast to float32

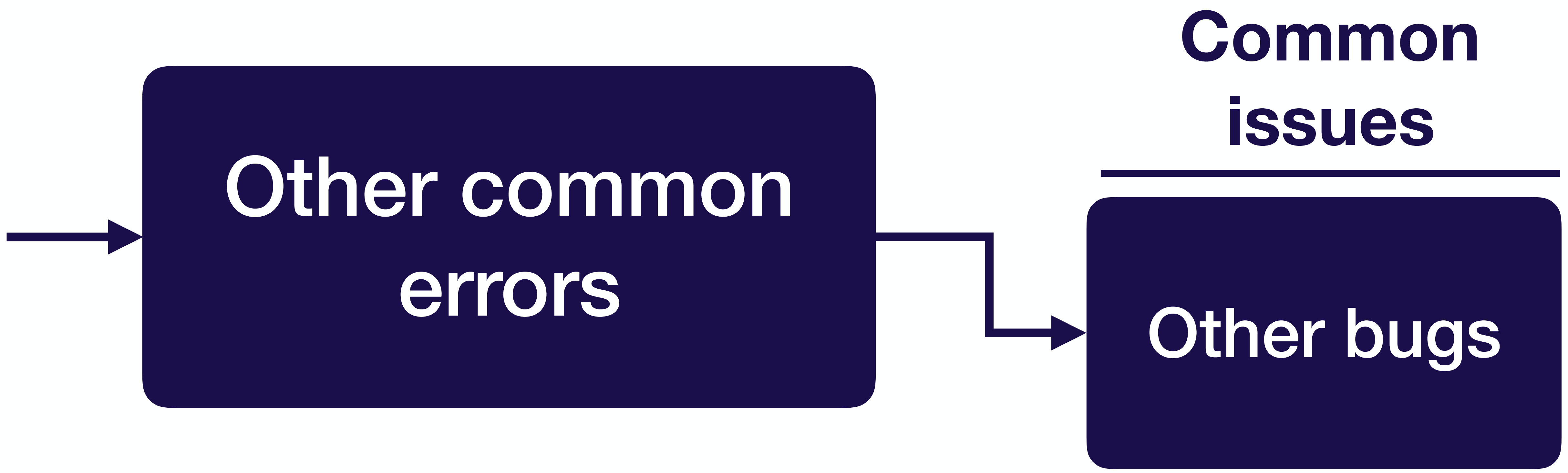
Casting issue → Data not in float32

Implementing bug-free DL models

OOM

Common issues	Most common causes
Too big a tensor	• Too large a batch size for your model (e.g., during evaluation) • Too large fully connected layers
Too much data	• Loading too large a dataset into memory, rather than using an input queue • Allocating too large a buffer for dataset creation
Duplicating operations	• Memory leak due to creating multiple models in the same session • Repeatedly creating an operation (e.g., in a function that gets called over and over again)
Other processes	• Other processes running on your GPU

Implementing bug-free DL models



Most common causes

- Forgot to initialize variables
- Forgot to turn off bias when using batch norm
- "Fetch argument has invalid type" - usually you overwrote one of your ops with an output during training

Implementing bug-free DL models

Steps

a. Get your model to run

b. Overfit a single batch

c. Compare to a known result

Implementing bug-free DL models

b Overfit a single batch

Common issues	Most common causes
Error goes up	
Error explodes	
Error oscillates	
Error plateaus	

Implementing bug-free DL models

b **Overfit a single batch**

Common issues	Most common causes
Error goes up	• Flipped the sign of the loss function / gradient • Learning rate too high • Softmax taken over wrong dimension
Error explodes	
Error oscillates	
Error plateaus	

Implementing bug-free DL models

b

Overfit a single batch

Common issues

- Error goes up
- Error explodes
- Error oscillates
- Error plateaus

Most common causes

Error explodes:

- Numerical issue. Check all exp, log, and div operations
- Learning rate too high

Implementing bug-free DL models

b Overfit a single batch

Common issues	Most common causes
Error goes up	
Error explodes	
Error oscillates	• Data or labels corrupted (e.g., zeroed, incorrectly shuffled, or preprocessed incorrectly) • Learning rate too high
Error plateaus	

Implementing bug-free DL models

b **Overfit a single batch**

Common issues	Most common causes
Error goes up	
Error explodes	
Error oscillates	
Error plateaus	• Learning rate too low • Gradients not flowing through the whole model • Too much regularization • Incorrect input to loss function (e.g., softmax instead of logits) • Data or labels corrupted

Implementing bug-free DL models

b Overfit a single batch

Common issues	Most common causes
Error goes up	• Flipped the sign of the loss function / gradient • Learning rate too high • Softmax taken over wrong dimension
Error explodes	• Numerical issue. Check all exp, log, and div operations • Learning rate too high
Error oscillates	• Data or labels corrupted (e.g., zeroed or incorrectly shuffled) • Learning rate too high
Error plateaus	• Learning rate too low • Gradients not flowing through the whole model • Too much regularization • Incorrect input to loss function (e.g., softmax instead of logits) • Data or labels corrupted

Implementing bug-free DL models

Steps

a. Get your model to run

b. Overfit a single batch

c. Compare to a known result

Hierarchy of known results

More useful

- Official model implementation evaluated on similar dataset to yours

You can:

- **Walk through code line-by-line and ensure you have the same output**
- **Ensure your performance is up to par with expectations**

Less useful

Hierarchy of known results

More useful

- Official model implementation evaluated on benchmark (e.g., MNIST)

You can:

- **Walk through code line-by-line and ensure you have the same output**

Less useful

Hierarchy of known results

More useful

- Unofficial model implementation

You can:

- **Same as before, but with lower confidence**

Less useful

Hierarchy of known results

More useful

- Results from a paper (with no code)

You can:

- **Ensure your performance is up to par with expectations**

Less useful

Hierarchy of known results

More useful

Less useful

You can:

- Make sure your model performs well in a simpler setting

- Results from your model on a benchmark dataset (e.g., MNIST)

Hierarchy of known results

More useful

You can:

- Get a general sense of what kind of performance can be expected

- Results from a similar model on a similar dataset

Less useful

Hierarchy of known results

More useful

You can:

- Make sure your model is learning anything at all

Less useful

- Super simple baselines (e.g., average of outputs or linear regression)

Hierarchy of known results

More useful

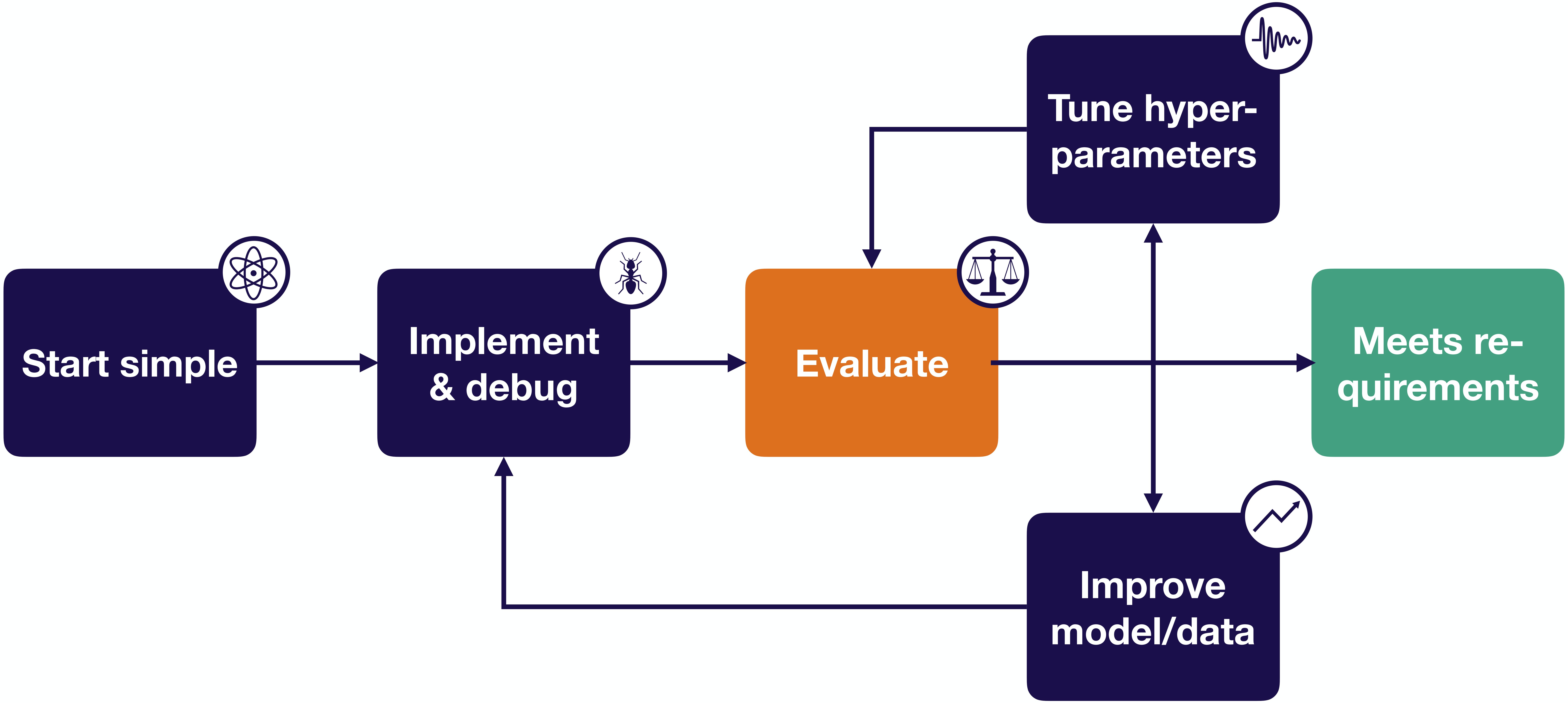
- Official model implementation evaluated on similar dataset to yours
- Official model implementation evaluated on benchmark (e.g., MNIST)
- Unofficial model implementation
- Results from the paper (with no code)
- Results from your model on a benchmark dataset (e.g., MNIST)
- Results from a similar model on a similar dataset
- Super simple baselines (e.g., average of outputs or linear regression)

Less useful

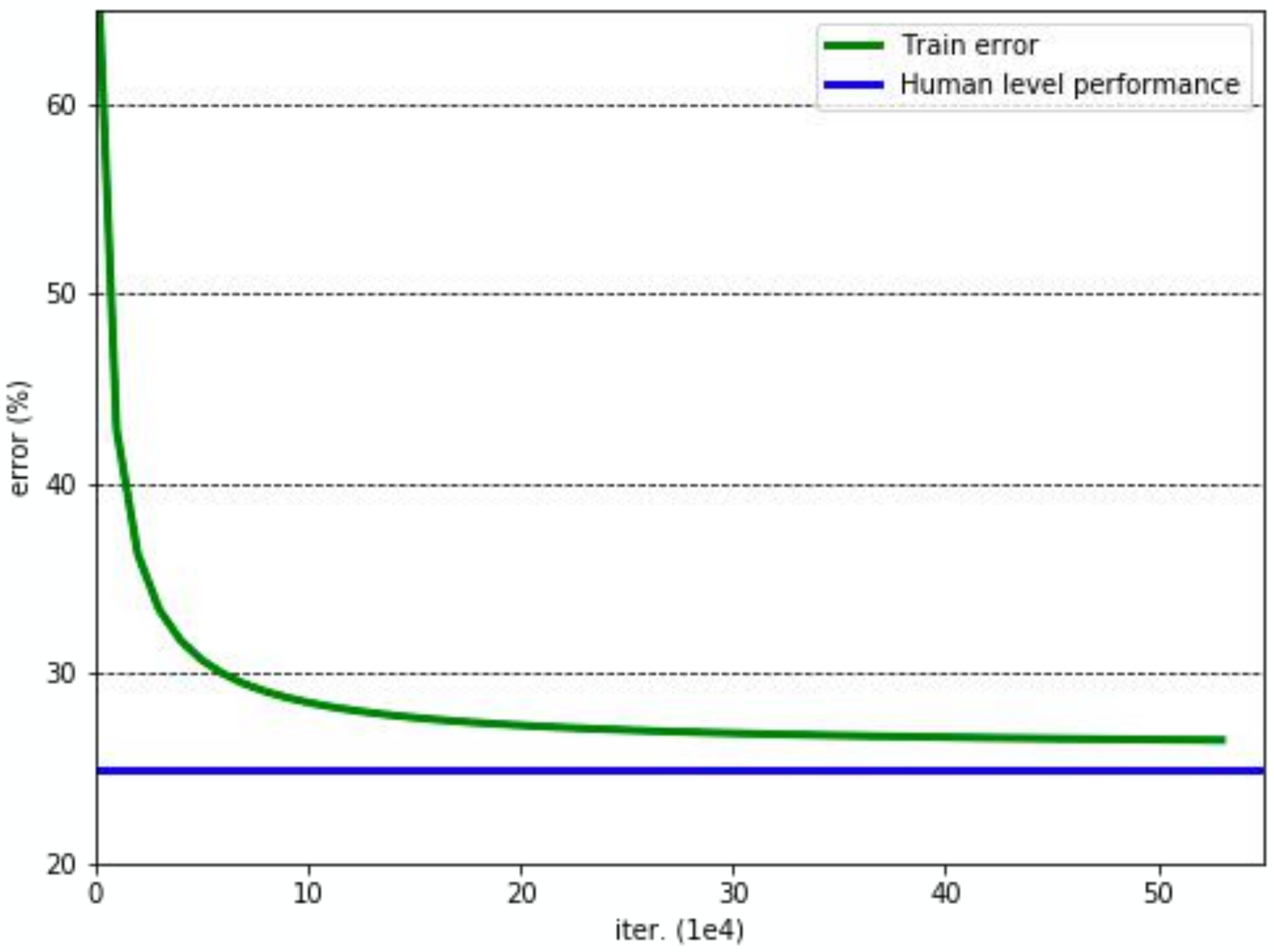
Summary: how to implement & debug

Steps	Summary
a. Get your model to run	• Step through in debugger & watch out for shape, casting, and OOM errors
b. Overfit a single batch	• Look for corrupted data, over-regularization, broadcasting errors
c. Compare to a known result	• Keep iterating until model performs up to expectations

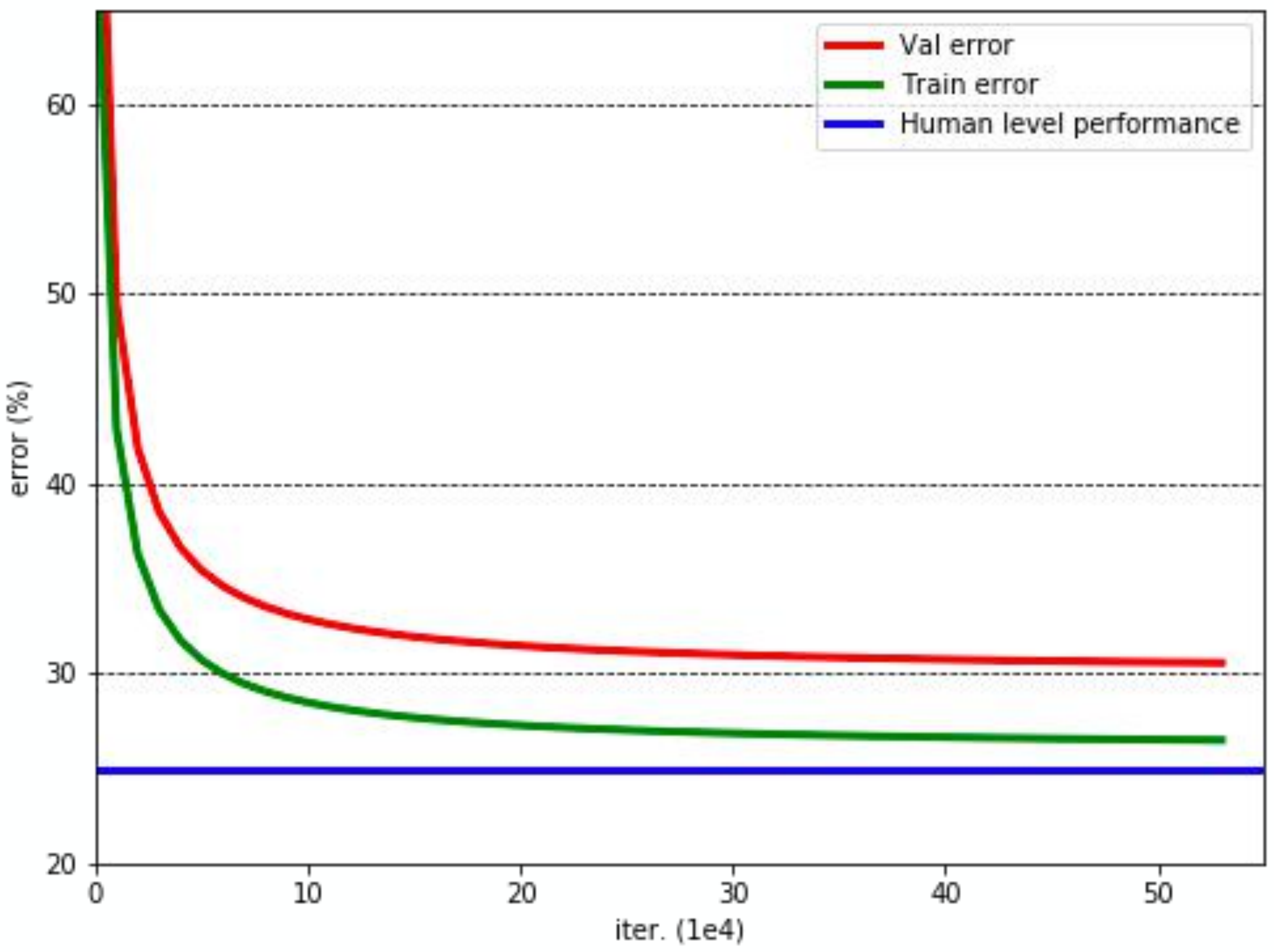
Strategy for DL troubleshooting



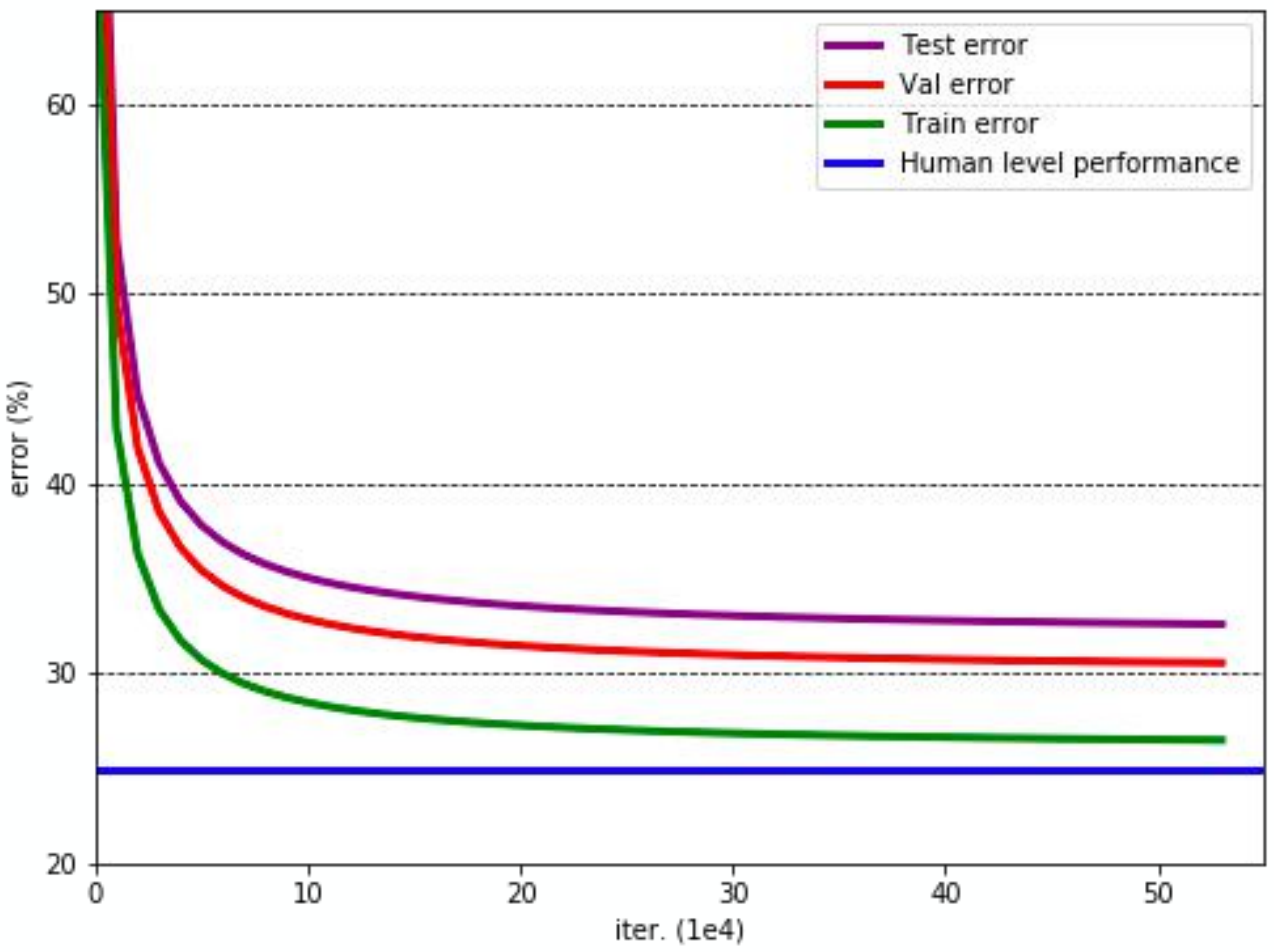
Bias-variance decomposition



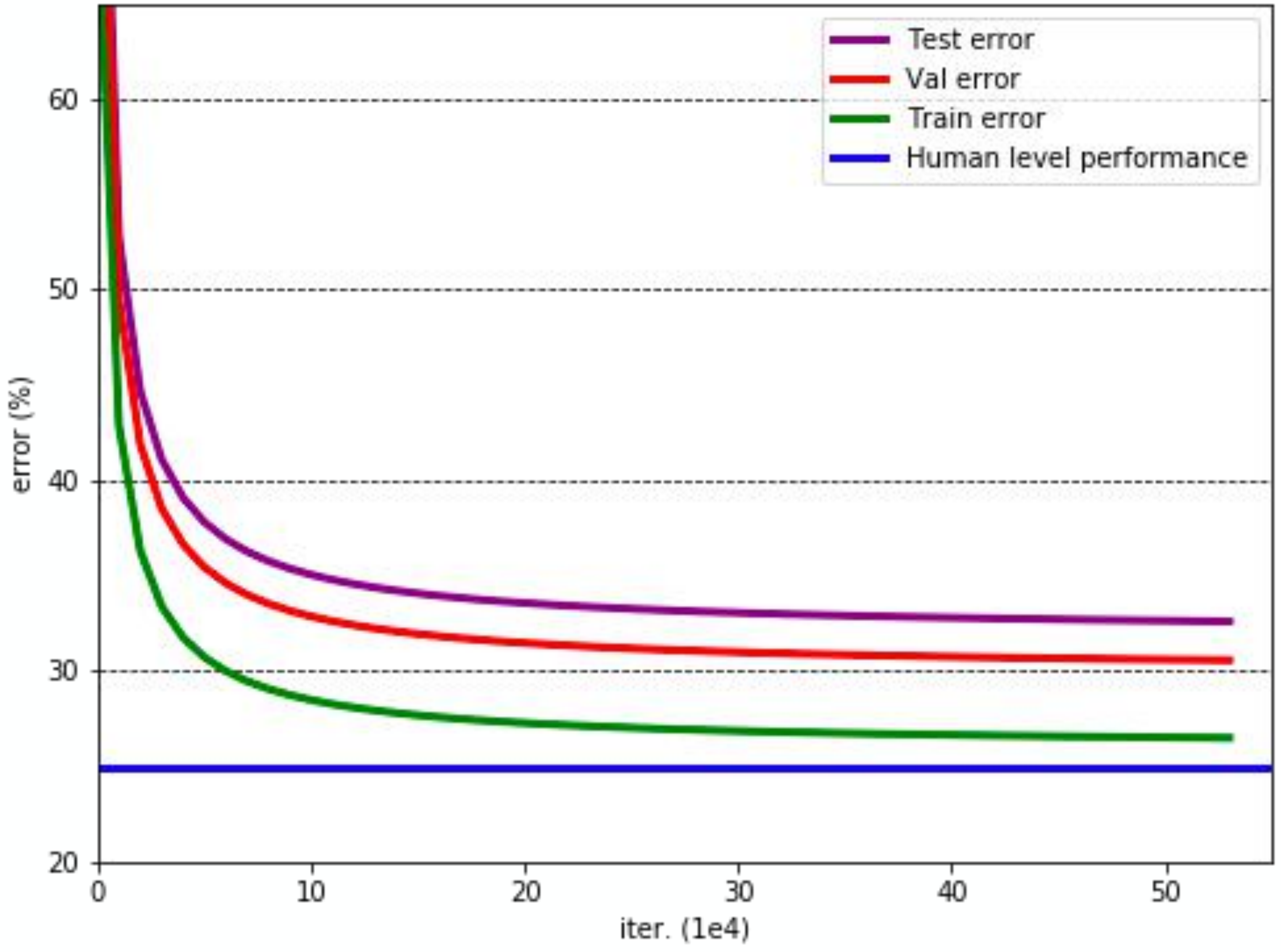
Bias-variance decomposition

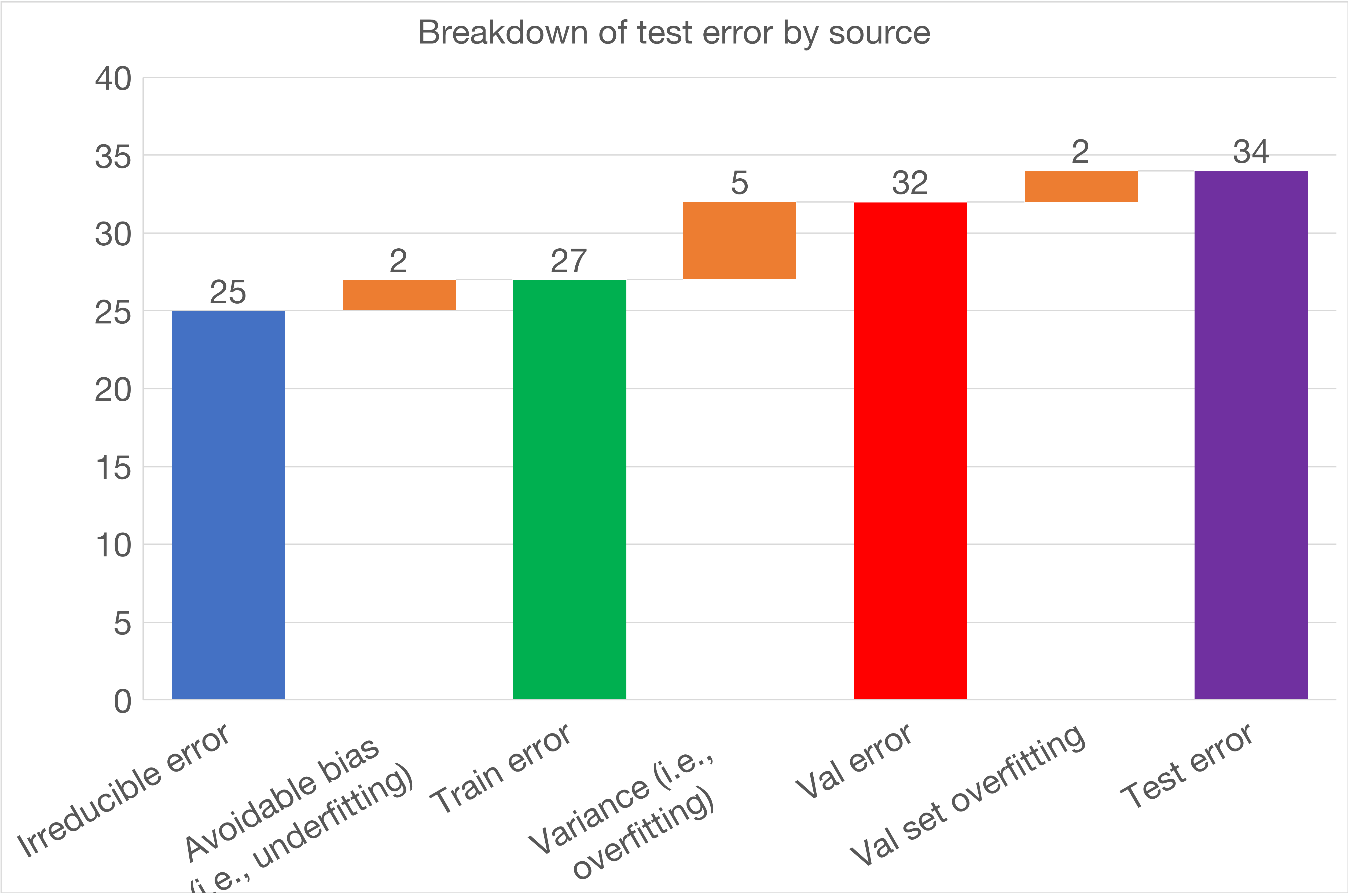


Bias-variance decomposition



Bias-variance decomposition





Bias-variance decomposition

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

This assumes train, val, and test all come from the same distribution. What if not?

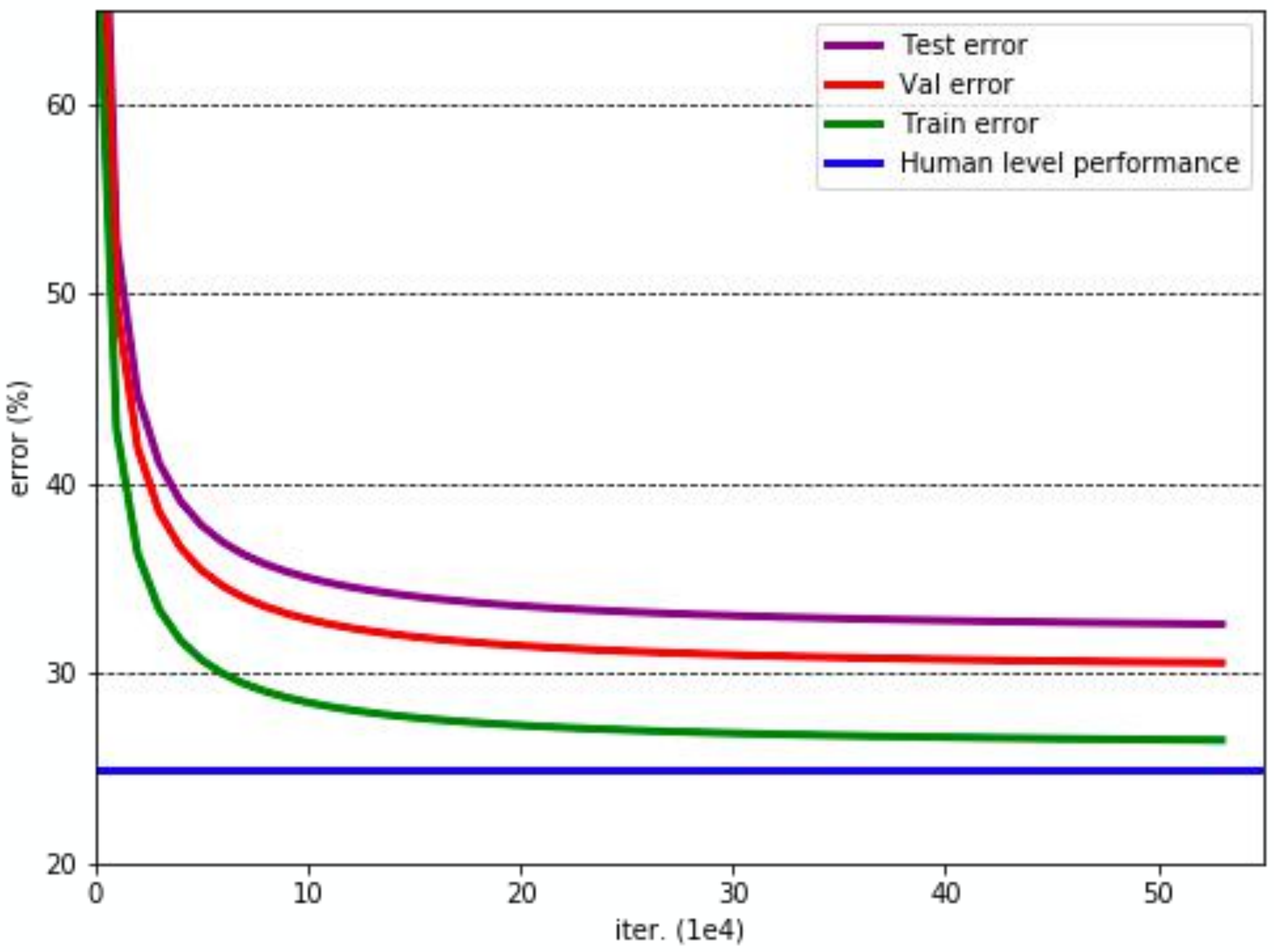
Handling distribution shift

Train data

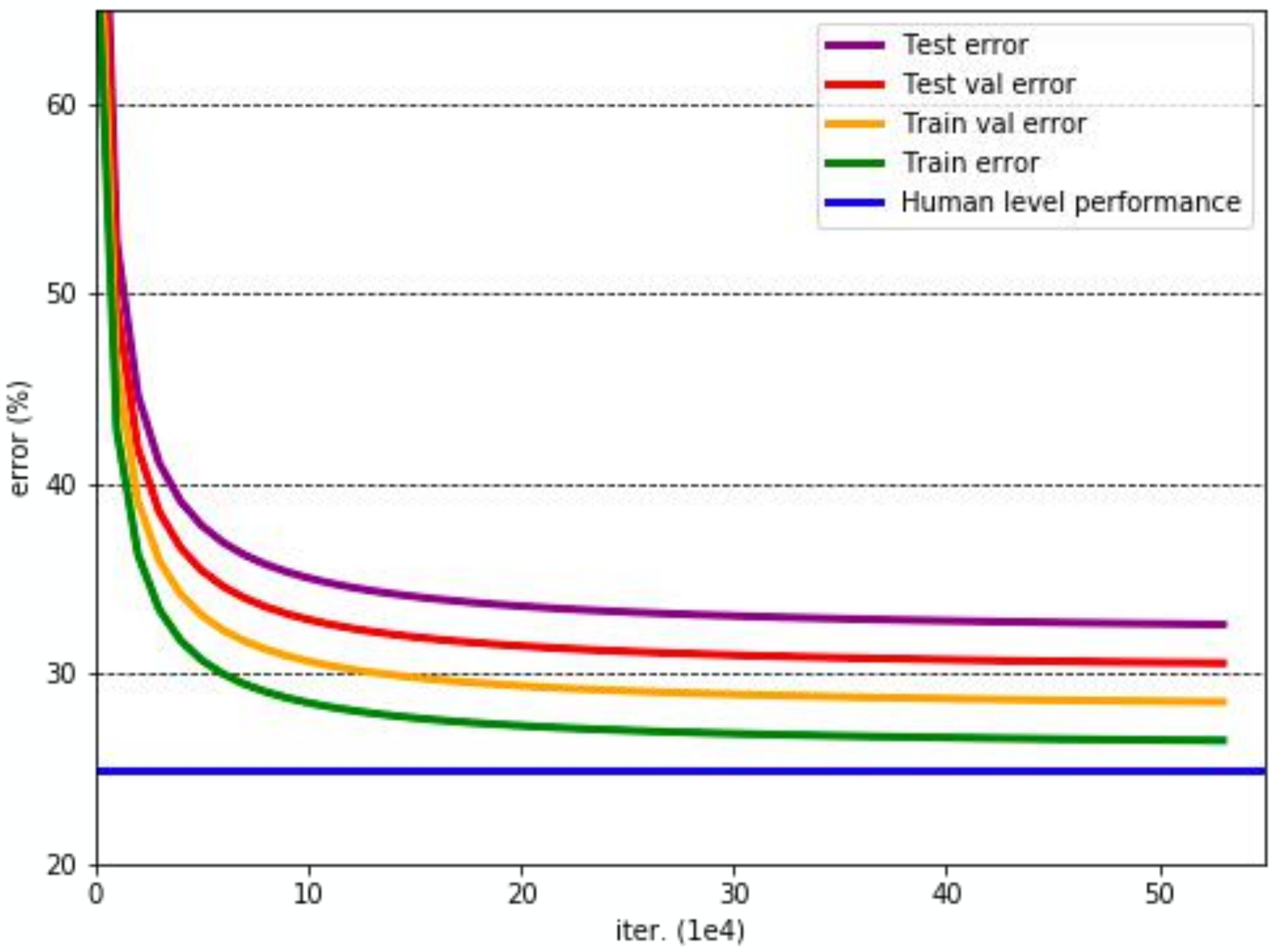
Test data

Use two val sets: one sampled from training distribution and one from test distribution

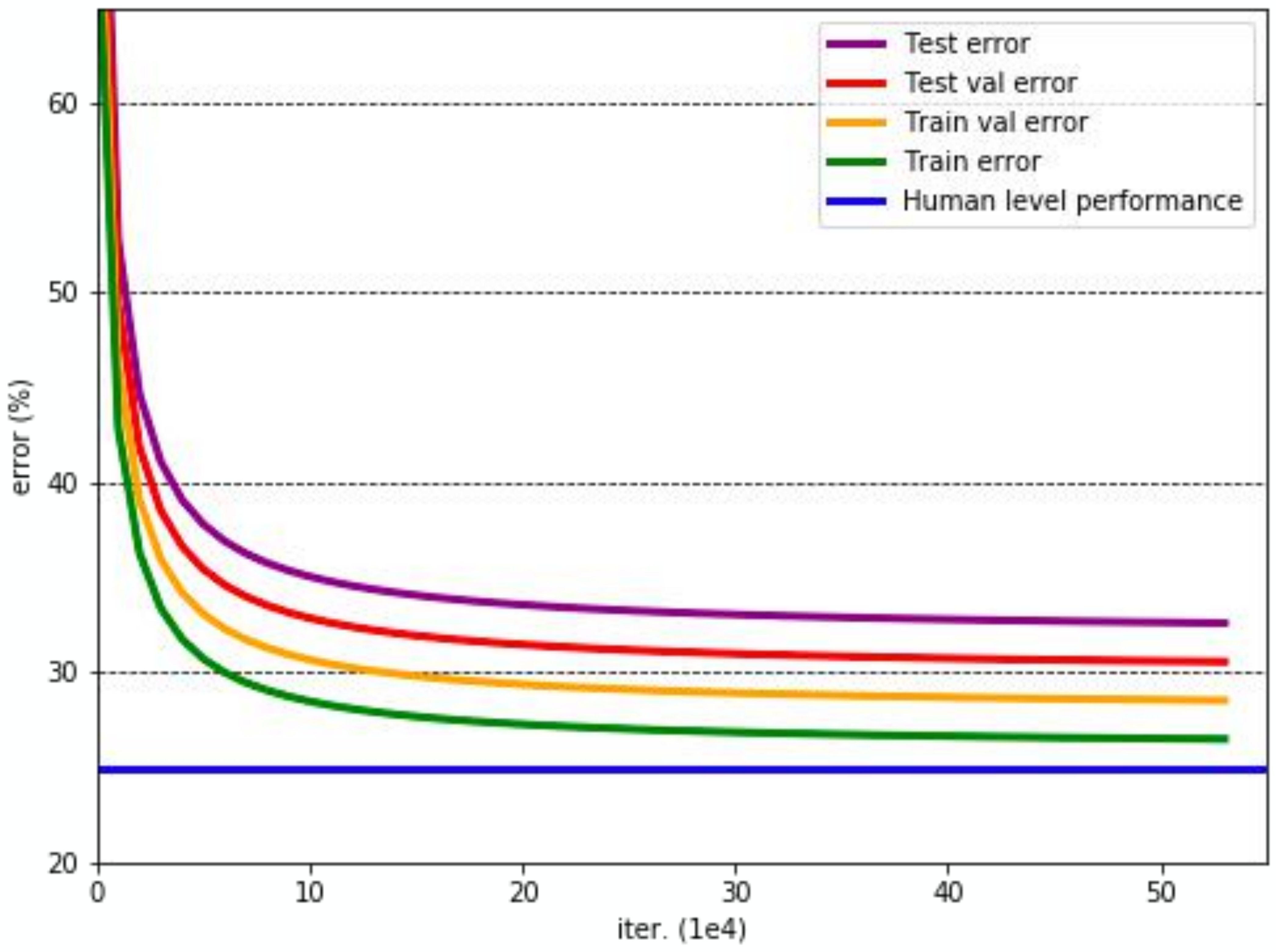
The bias-variance tradeoff

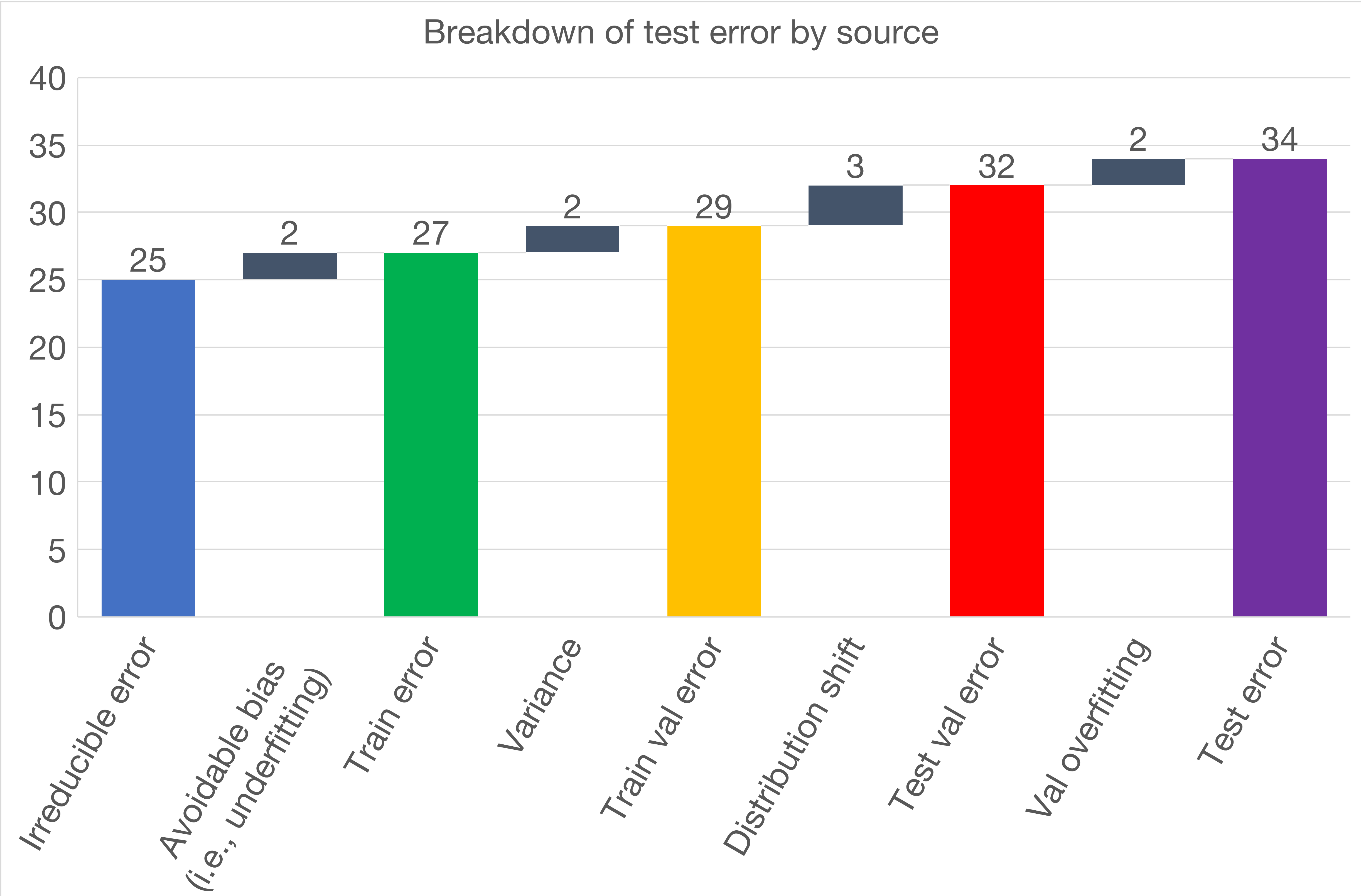


Bias-variance with distribution shift



Bias-variance with distribution shift





Train, val, and test error for pedestrian detection

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

Train - goal = 19% (under-fitting)

Running example

0 (no pedestrian) **1 (yes pedestrian)**

Goal: 99% classification accuracy

Train, val, and test error for pedestrian detection

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

Val - train = 7% (over-fitting)

Running example

0 (no pedestrian) 1 (yes pedestrian)

Goal: 99% classification accuracy

Train, val, and test error for pedestrian detection

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

Test - val = 1%
(looks good!)

Running example

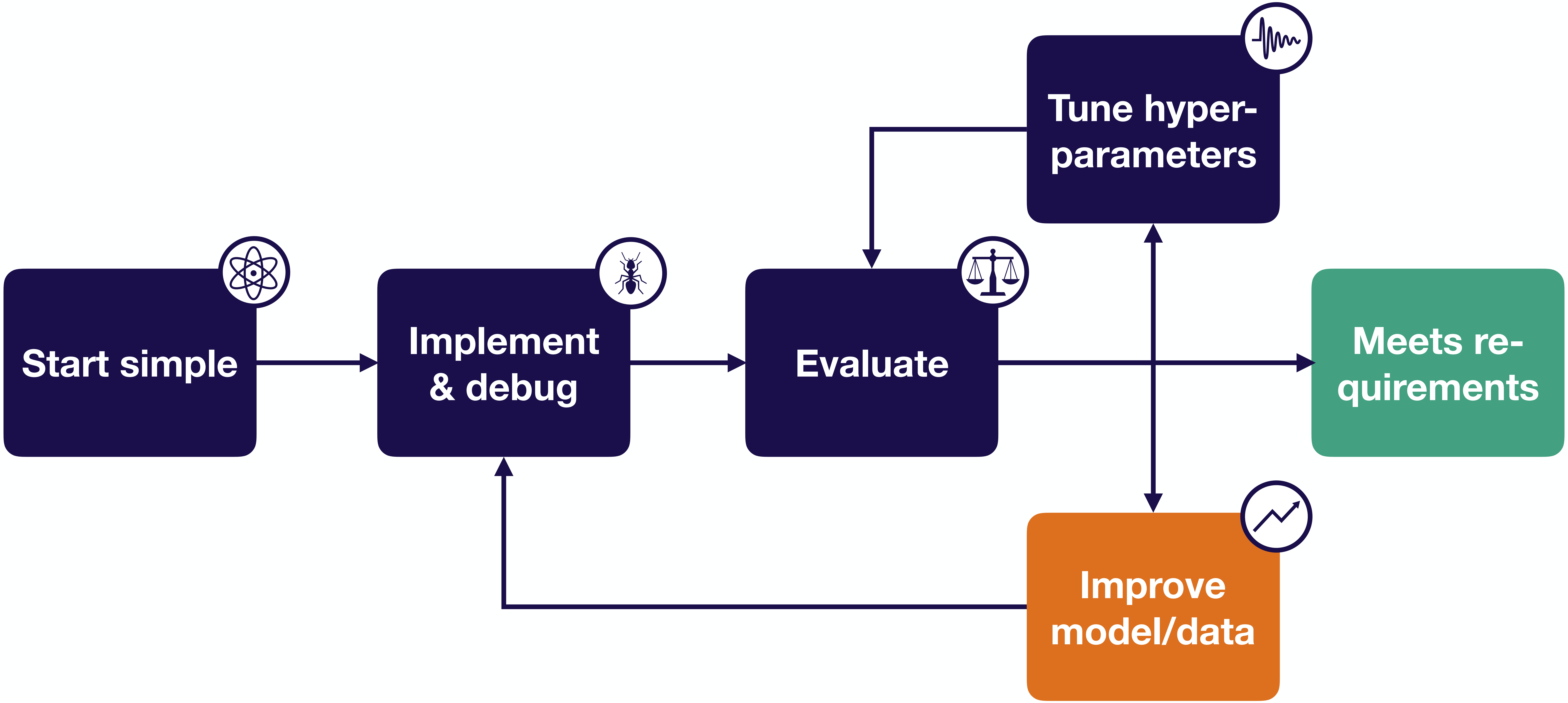
0 (no pedestrian) **1 (yes pedestrian)**

Goal: 99% classification accuracy

Summary: evaluating model performance

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

Strategy for DL troubleshooting



Prioritizing improvements (i.e., applying the bias-variance tradeoff)

Steps

- a. Address under-fitting
- b. Address over-fitting
- c. Address distribution shift
- d. Re-balance datasets (if applicable)

Addressing under-fitting (i.e., reducing bias)

Try first

A. Make your model bigger (i.e., add layers or use more units per layer)

B. Reduce regularization

C. Error analysis

D. Choose a different (closer to state-of-the art) model architecture (e.g., move from LeNet to ResNet)

E. Tune hyper-parameters (e.g., learning rate)

F. Add features

Try later

Train, val, and test error for pedestrian detection

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%

0 (no pedestrian) **1 (yes pedestrian)**

Goal: 99% classification accuracy (i.e., 1% error)

Train, val, and test error for pedestrian detection

Switch to ResNet-101

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

0 (no pedestrian) **1 (yes pedestrian)**

Goal: 99% classification accuracy (i.e., 1% error)

Train, val, and test error for pedestrian detection

Tune learning rate

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

0 (no pedestrian) **1 (yes pedestrian)**

Goal: 99% classification accuracy
(i.e., 1% error)

Prioritizing improvements (i.e., applying the bias-variance tradeoff)

Steps

a. Address under-fitting
b. Address over-fitting
c. Address distribution shift
d. Re-balance datasets (if applicable)

Addressing over-fitting (i.e., reducing variance)

Try first

A. Add more training data (if possible!)

B. Add normalization (e.g., batch norm, layer norm)

C. Add data augmentation

D. Increase regularization (e.g., dropout, L2, weight decay)

E. Error analysis

F. Choose a different (closer to state-of-the-art) model architecture

G. Tune hyperparameters

H. Early stopping

I. Remove features

J. Reduce model size

Try later

Addressing over-fitting (i.e., reducing variance)

Try first

A. Add more training data (if possible!)

B. Add normalization (e.g., batch norm, layer norm)

C. Add data augmentation

D. Increase regularization (e.g., dropout, L2, weight decay)

E. Error analysis

F. Choose a different (closer to state-of-the-art) model architecture

G. Tune hyperparameters

H. Early stopping

I. Remove features

J. Reduce model size

Not recommended!

Try later

Train, val, and test error for pedestrian detection

Error source	Value
Goal performance	1%
Train error	0.8%
Validation error	12%
Test error	12%

Running example

0 (no pedestrian) **1 (yes pedestrian)**

Goal: 99% classification accuracy

Train, val, and test error for pedestrian detection

Increase dataset size to 250,000

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

Running example

0 (no pedestrian) **1 (yes pedestrian)**

Goal: 99% classification accuracy

Train, val, and test error for pedestrian detection

Add weight decay

Error source	~~Value~~	~~Value~~	Value
Goal performance	~~1%~~	~~1%~~	1%
Train error	~~0.8%~~	~~1.5%~~	1.7%
Validation error	~~12%~~	~~5%~~	4%
Test error	~~12%~~	~~6%~~	4%

Running example

0 (no pedestrian)

1 (yes pedestrian)

Goal: 99% classification accuracy

Train, val, and test error for pedestrian detection

Add data augmentation

Error source	~~Value~~	~~Value~~	~~Value~~	Value
Goal performance	~~1%~~	~~1%~~	~~1%~~	1%
Train error	~~0.8%~~	~~1.5%~~	~~1.7%~~	2%
Validation error	~~12%~~	~~5%~~	~~4%~~	2.5%
Test error	~~12%~~	~~6%~~	~~4%~~	2.6%

Running example

0 (no pedestrian) **1 (yes pedestrian)**

Goal: 99% classification accuracy

Train, val, and test error for pedestrian detection

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	~~0.8%~~	~~1.5%~~	~~1.7%~~	~~2%~~	0.6%
Validation error	~~12%~~	~~5%~~	~~4%~~	~~2.5%~~	0.9%
Test error	~~12%~~	~~6%~~	~~4%~~	~~2.6%~~	1.0%

Running example

0 (no pedestrian)

1 (yes pedestrian)

Goal: 99% classification accuracy

Prioritizing improvements (i.e., applying the bias-variance tradeoff)

Steps

a. Address under-fitting
b. Address over-fitting
c. Address distribution shift
d. Re-balance datasets (if applicable)

Addressing distribution shift

Try first

Try later

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

Error analysis

Test-val set errors (no pedestrian detected)

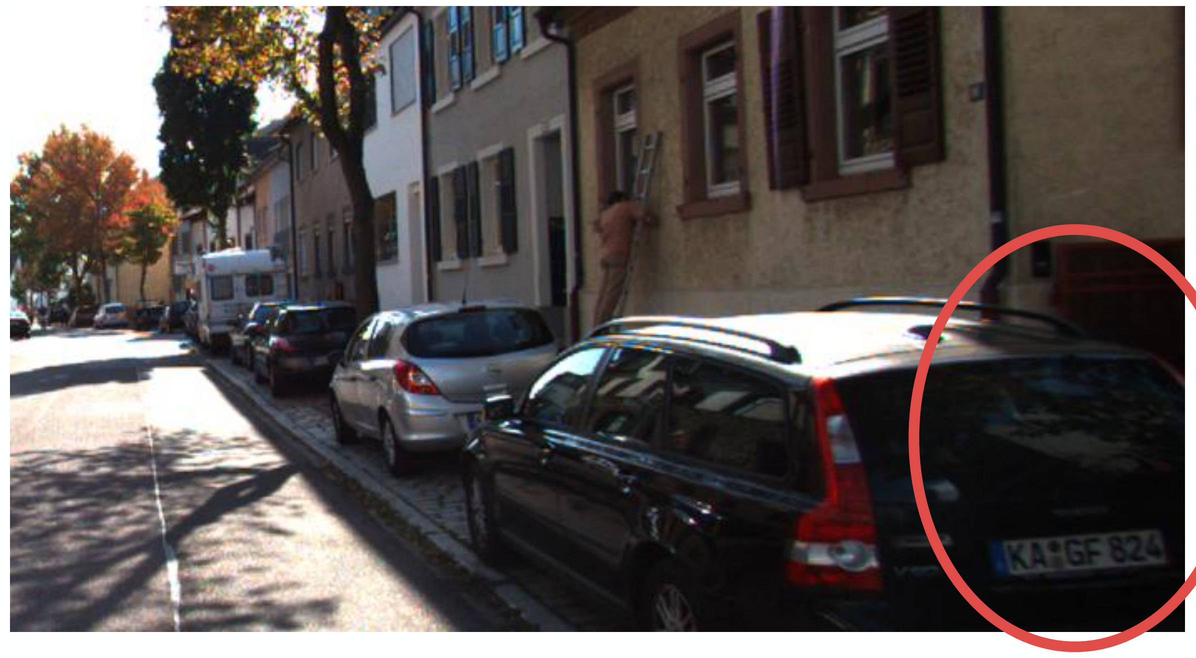




Train-val set errors (no pedestrian detected)


Error analysis

Test-val set errors (no pedestrian detected)



Train-val set errors (no pedestrian detected)

Error type 1: hard-to-see pedestrians

Error analysis

Test-val set errors (no pedestrian detected)

Train-val set errors (no pedestrian detected)

Error type 2: reflections

Error analysis

Test-val set errors (no pedestrian detected)





Train-val set errors (no pedestrian detected)


Error type 3 (test-val only):
night scenes

Error analysis

Error type	Error % (train-val)	Error % (test-val)	Potential solutions	Priority
1. Hard-to-see pedestrians	0.1%	0.1%	• Better sensors	Low
2. Reflections	0.3%	0.3%	• Collect more data with reflections • Add synthetic reflections to train set • Try to remove with pre-processing • Better sensors	Medium
3. Nighttime scenes	0.1%	1%	• Collect more data at night • Synthetically darken training images • Simulate night-time data • Use domain adaptation	High

Domain adaptation

What is it?

Techniques to train on “source” distribution and generalize to another “target” using only unlabeled data or limited labeled data

When should you consider using it?

- Access to labeled data from test distribution is limited
- Access to relatively similar data is plentiful

Types of domain adaptation

Type	Use case	Example techniques
Supervised	You have limited data from target domain	• Fine-tuning a pre-trained model • Adding target data to train set
Un-supervised	You have lots of un-labeled data from target domain	• Correlation Alignment (CORAL) • Domain confusion • CycleGAN

Prioritizing improvements (i.e., applying the bias-variance tradeoff)

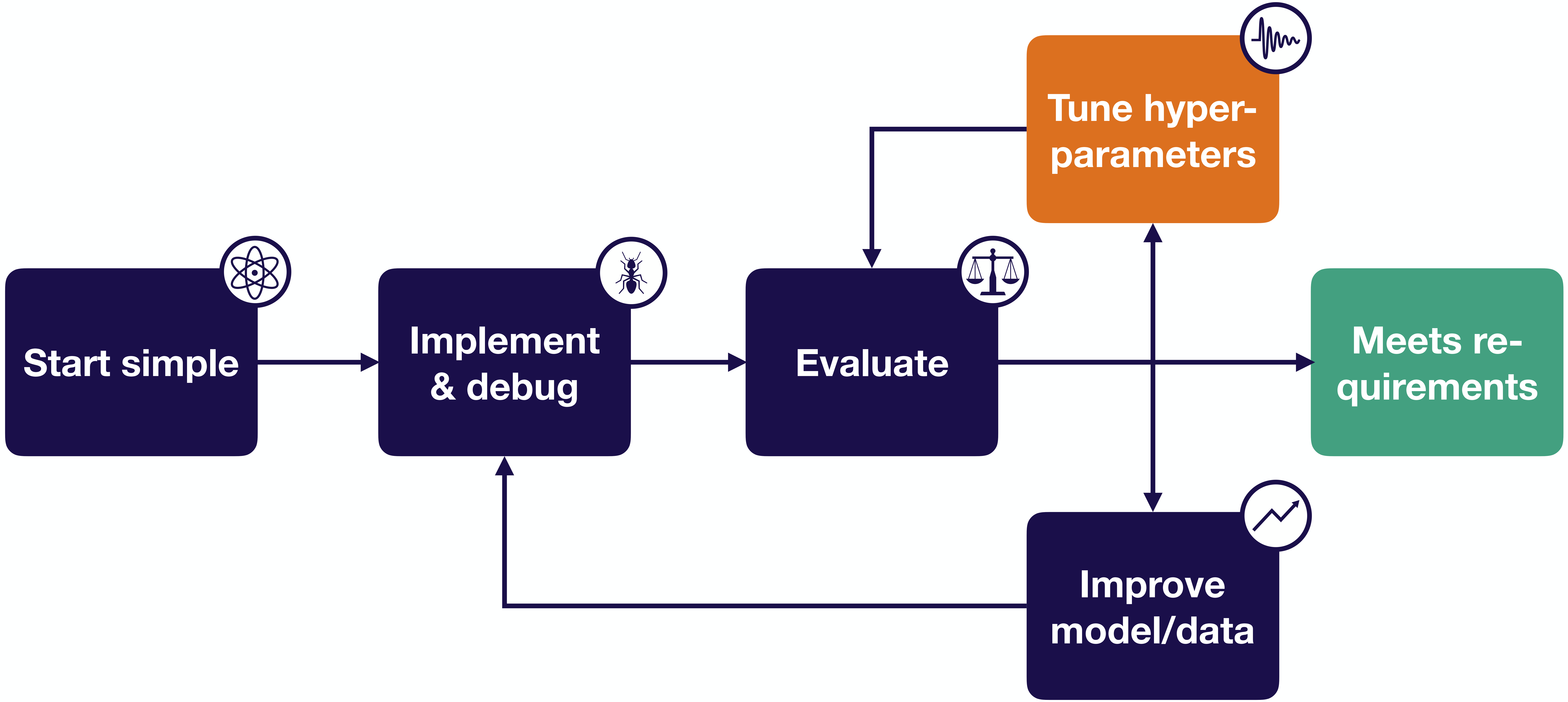
Steps

a. Address under-fitting
b. Address over-fitting
c. Address distribution shift
d. Re-balance datasets (if applicable)

Rebalancing datasets

- If (test)-val looks significantly better than test, you overfit to the val set
- This happens with small val sets or lots of hyper parameter tuning
- When it does, recollect val data

Strategy for DL troubleshooting



Hyperparameter optimization

Model & optimizer choices?

Network: ResNet

- How many layers?
- Weight initialization?
- Kernel size?
- Etc

Optimizer: Adam

- Batch size?
- Learning rate?
- beta1, beta2, epsilon?

Regularization

- ….

Running example

0 (no pedestrian) **1 (yes pedestrian)**

Goal: 99% classification accuracy

Which hyper-parameters to tune?

Choosing hyper-parameters

- More sensitive to some than others
- Depends on choice of model
- Rules of thumb (only) to the right
- Sensitivity is relative to default values! (e.g., if you are using all-zeros weight initialization or vanilla SGD, changing to the defaults will make a big difference)

Hyperparameter	Approximate sensitivity
Learning rate	High
Optimizer choice	Low
Other optimizer params (e.g., Adam beta1)	Low
Batch size	Low
Weight initialization	Medium
Loss function	High
Model depth	Medium
Layer size	High
Layer params (e.g., kernel size)	Medium
Weight of regularization	Medium
Nonlinearity	Low

Method 1: manual hyperparam optimization

How it works

- Understand the algorithm
  - E.g., higher learning rate means faster less stable training
- Train & evaluate model
- Guess a better hyperparam value & re-evaluate
- Can be combined with other methods (e.g., manually select parameter ranges to optimizer over)

Advantages

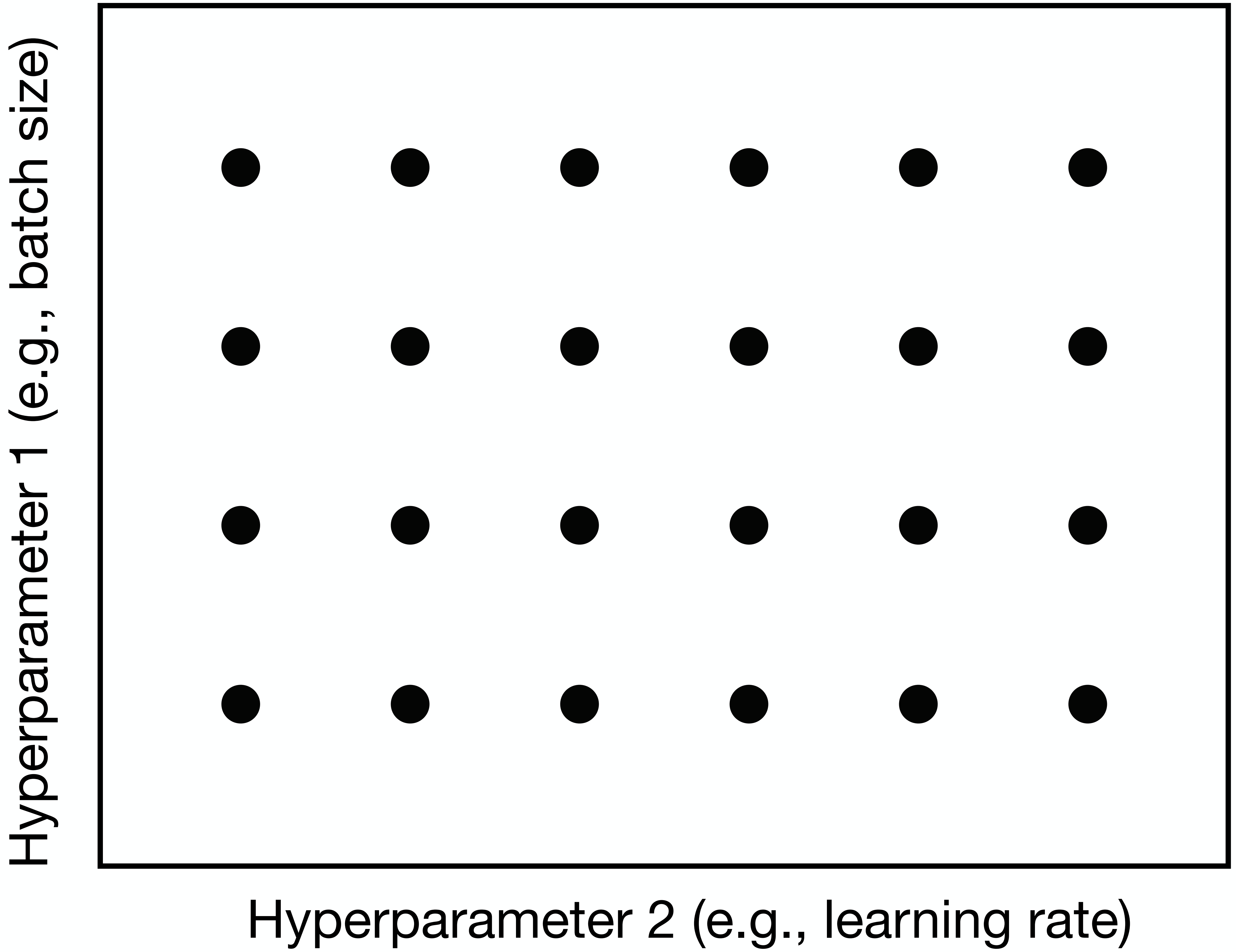
- For a skilled practitioner, may require least computation to get good result

Disadvantages

- Requires detailed understanding of the algorithm
- Time-consuming

Method 2: grid search

How it works



Advantages

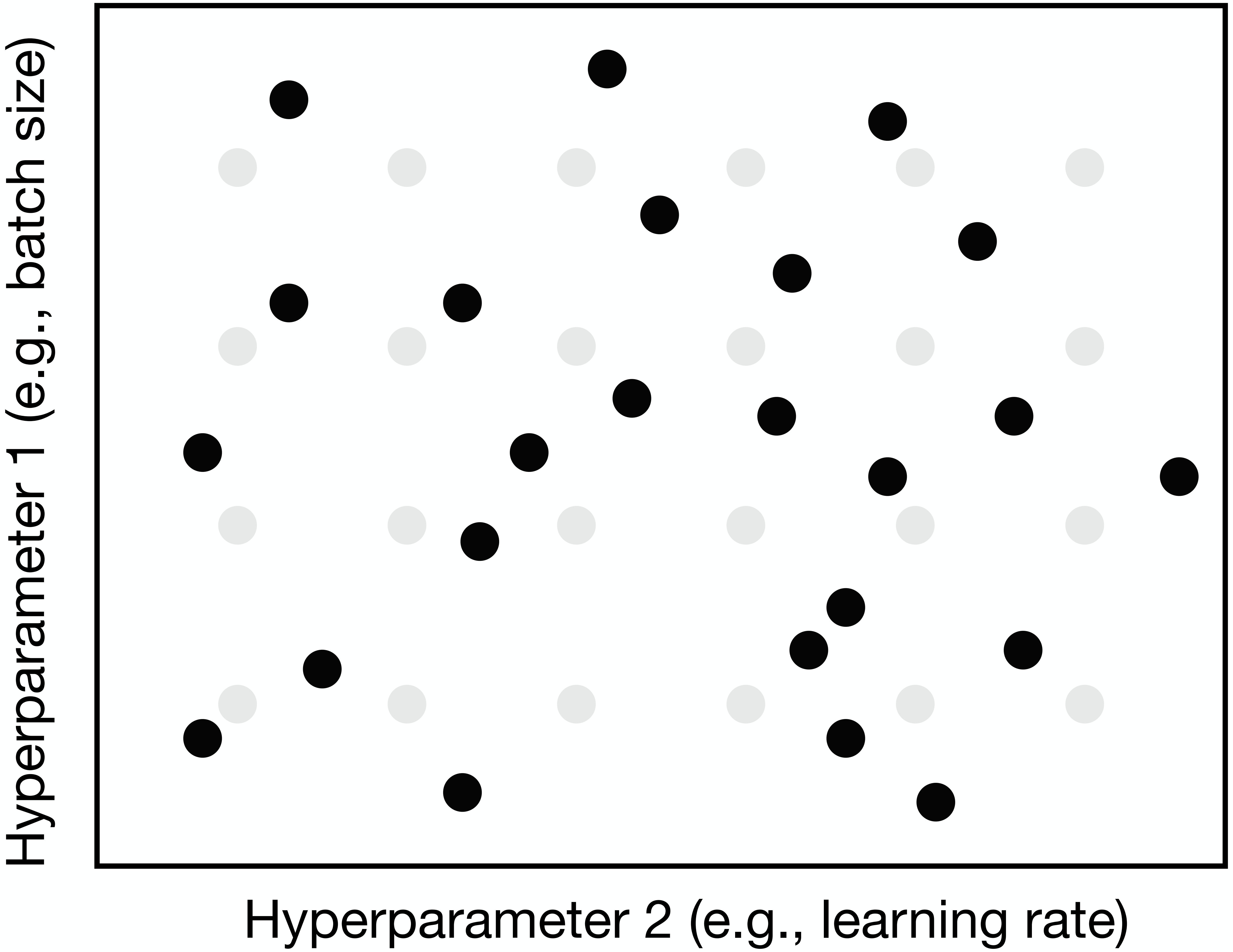
- Super simple to implement
- Can produce good results

Disadvantages

- 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

How it works

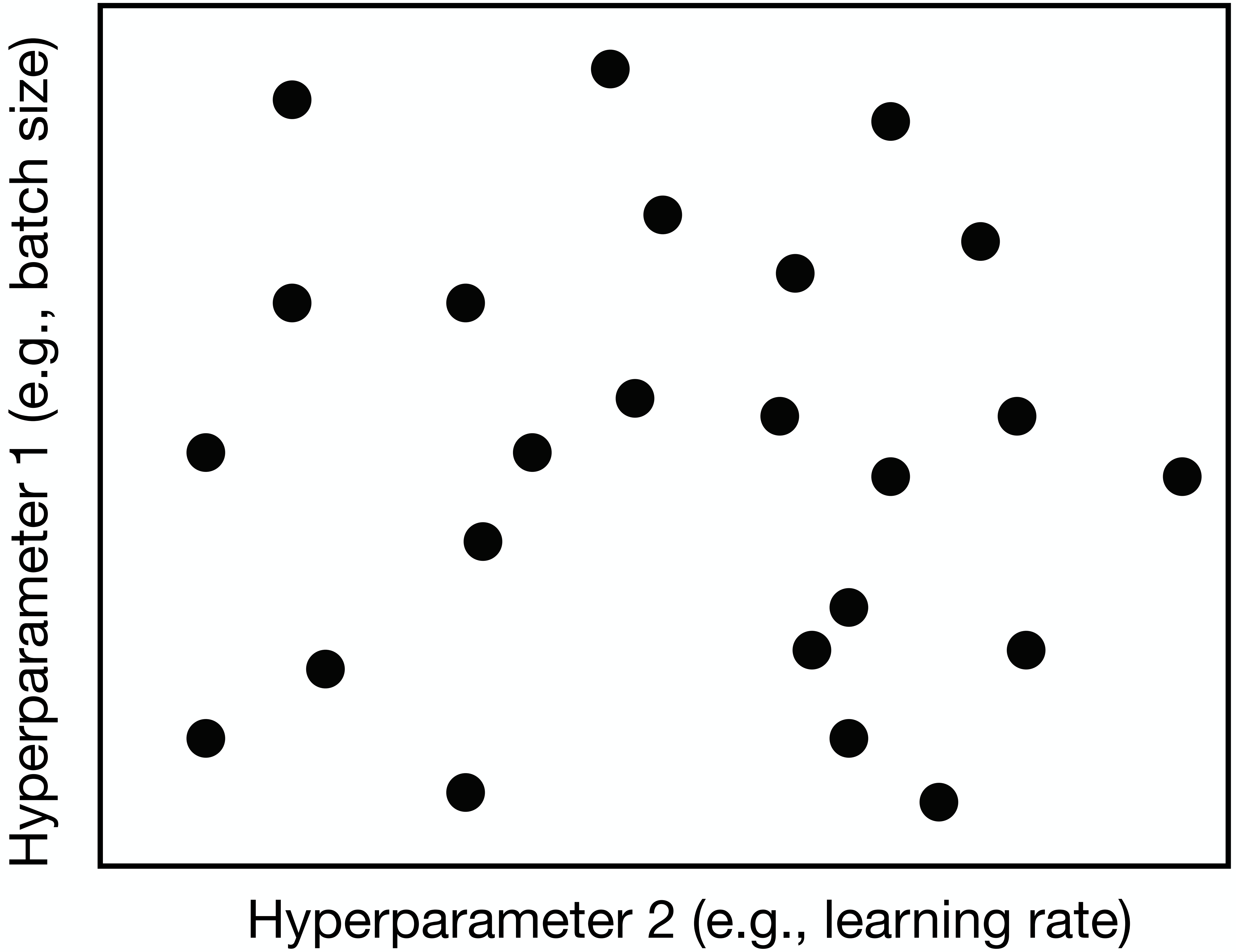


Advantages

Disadvantages

Method 4: coarse-to-fine

How it works

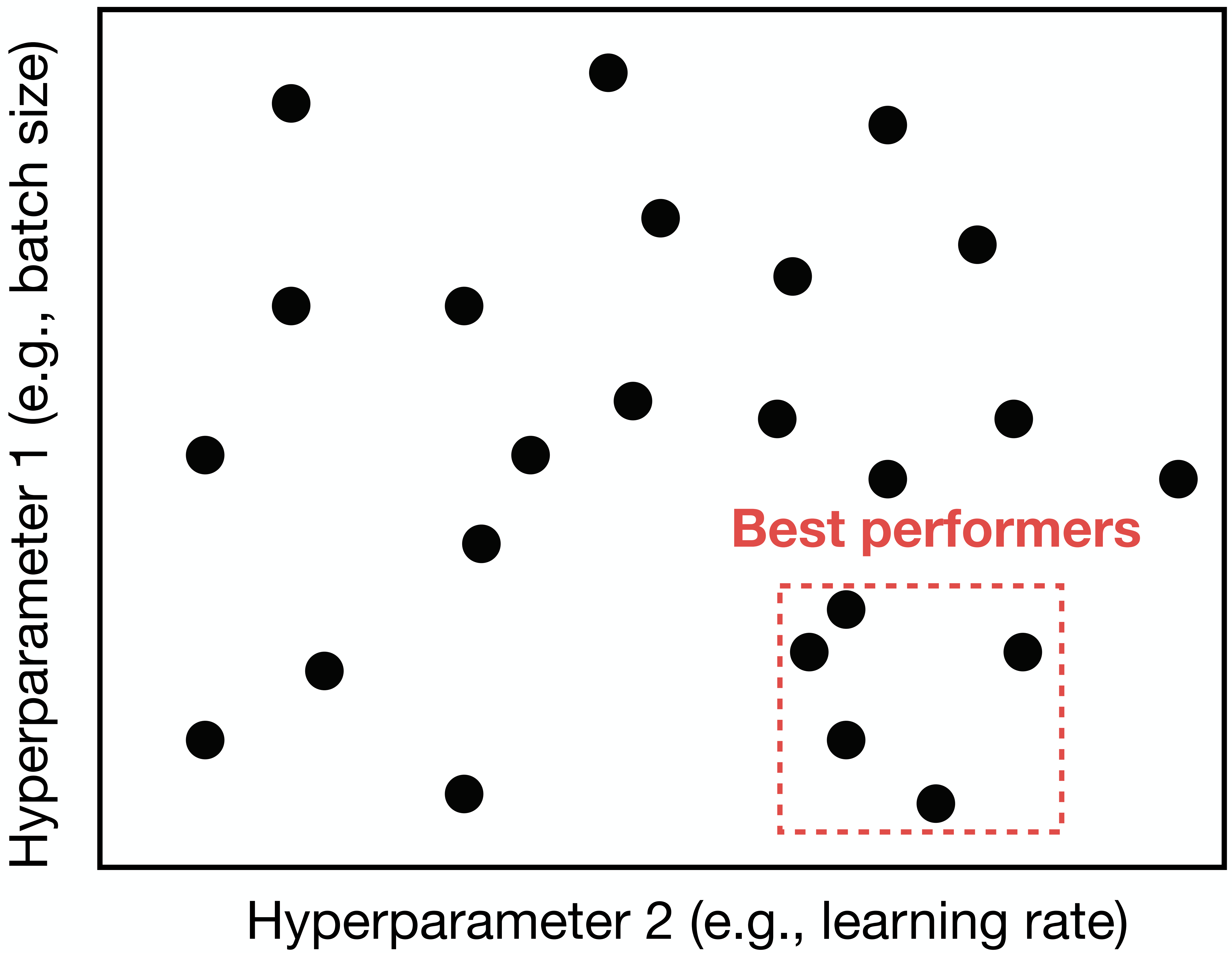


Advantages

Disadvantages

Method 4: coarse-to-fine

How it works

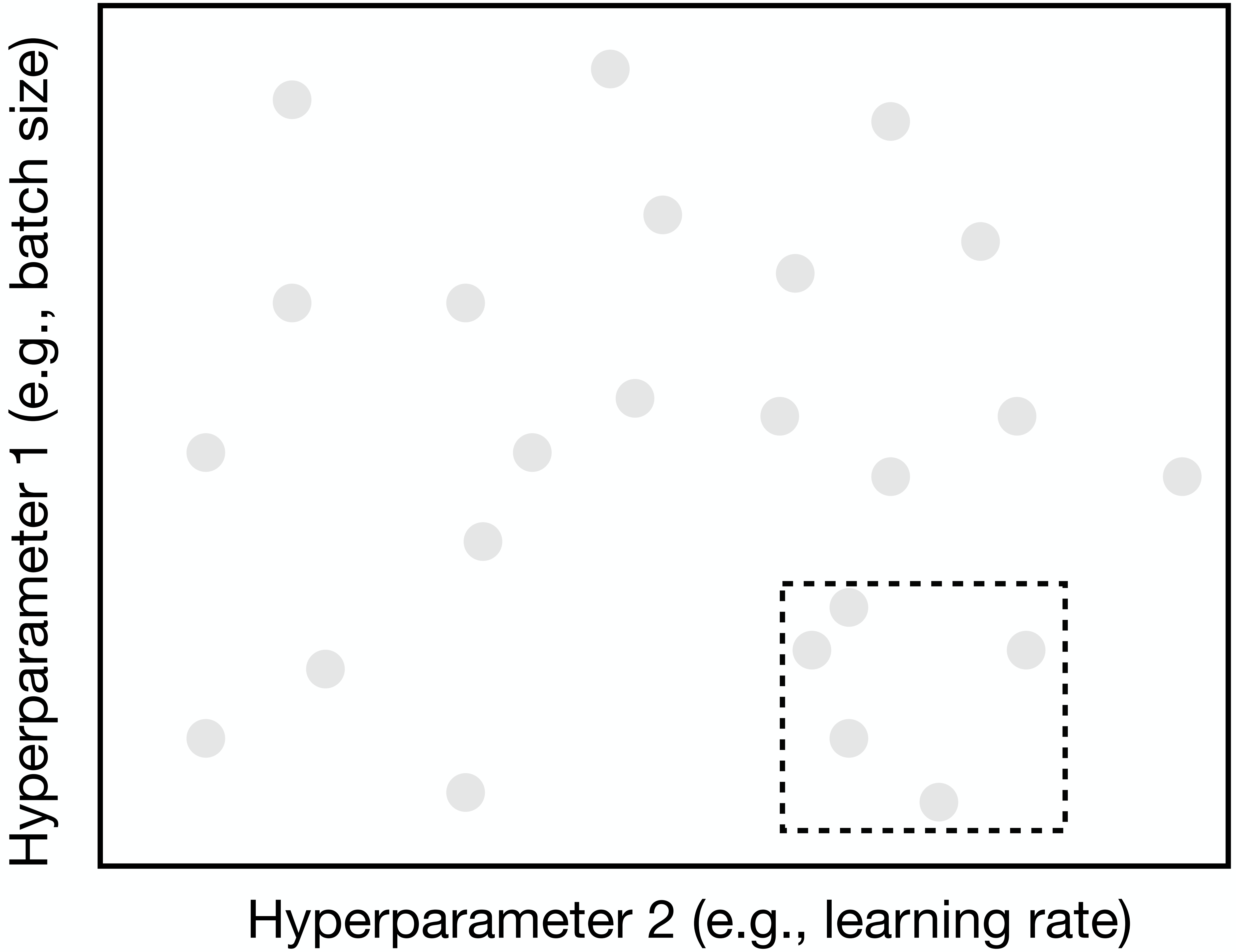


Advantages

Disadvantages

Method 4: coarse-to-fine

How it works

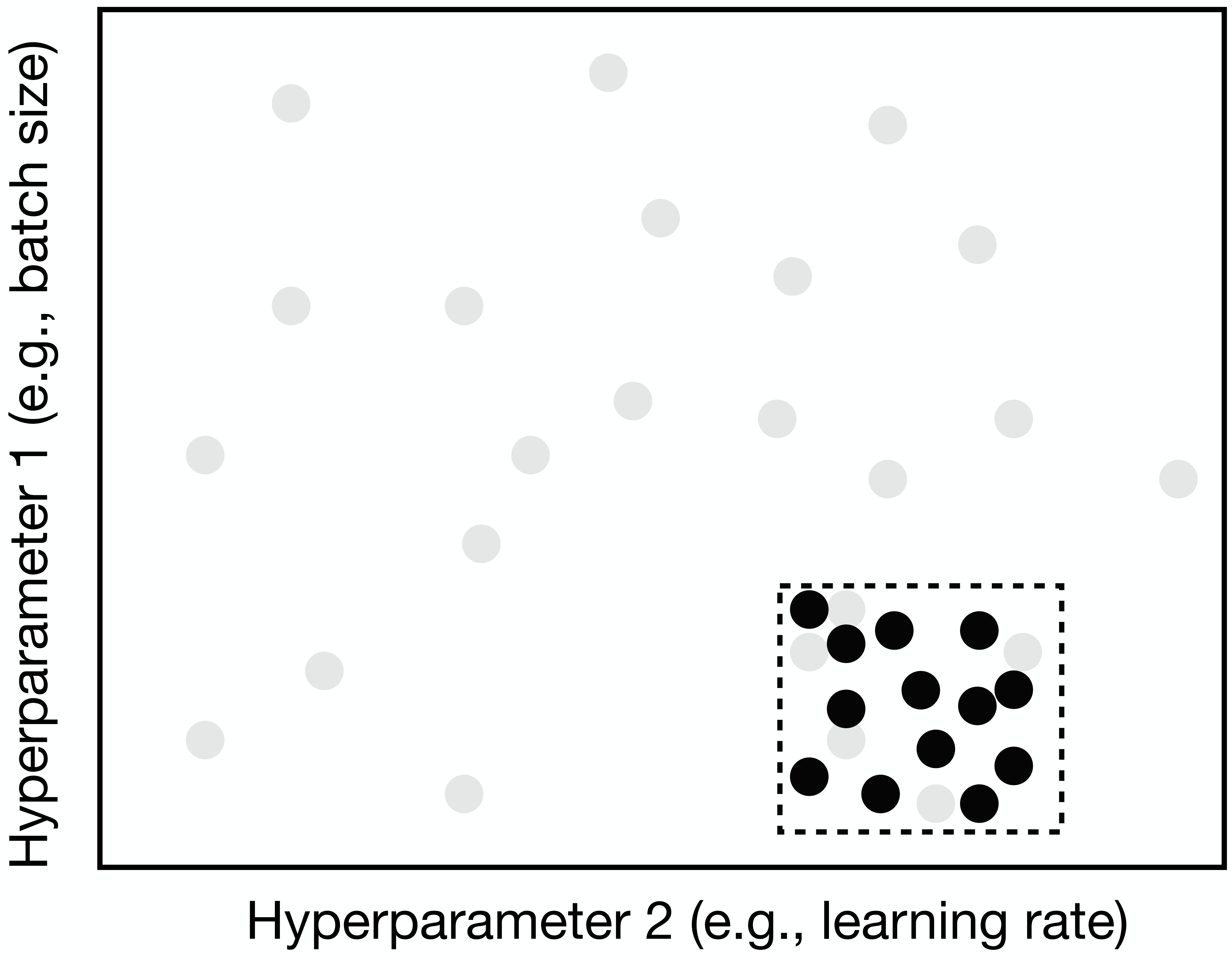


Advantages

Disadvantages

Method 4: coarse-to-fine

How it works

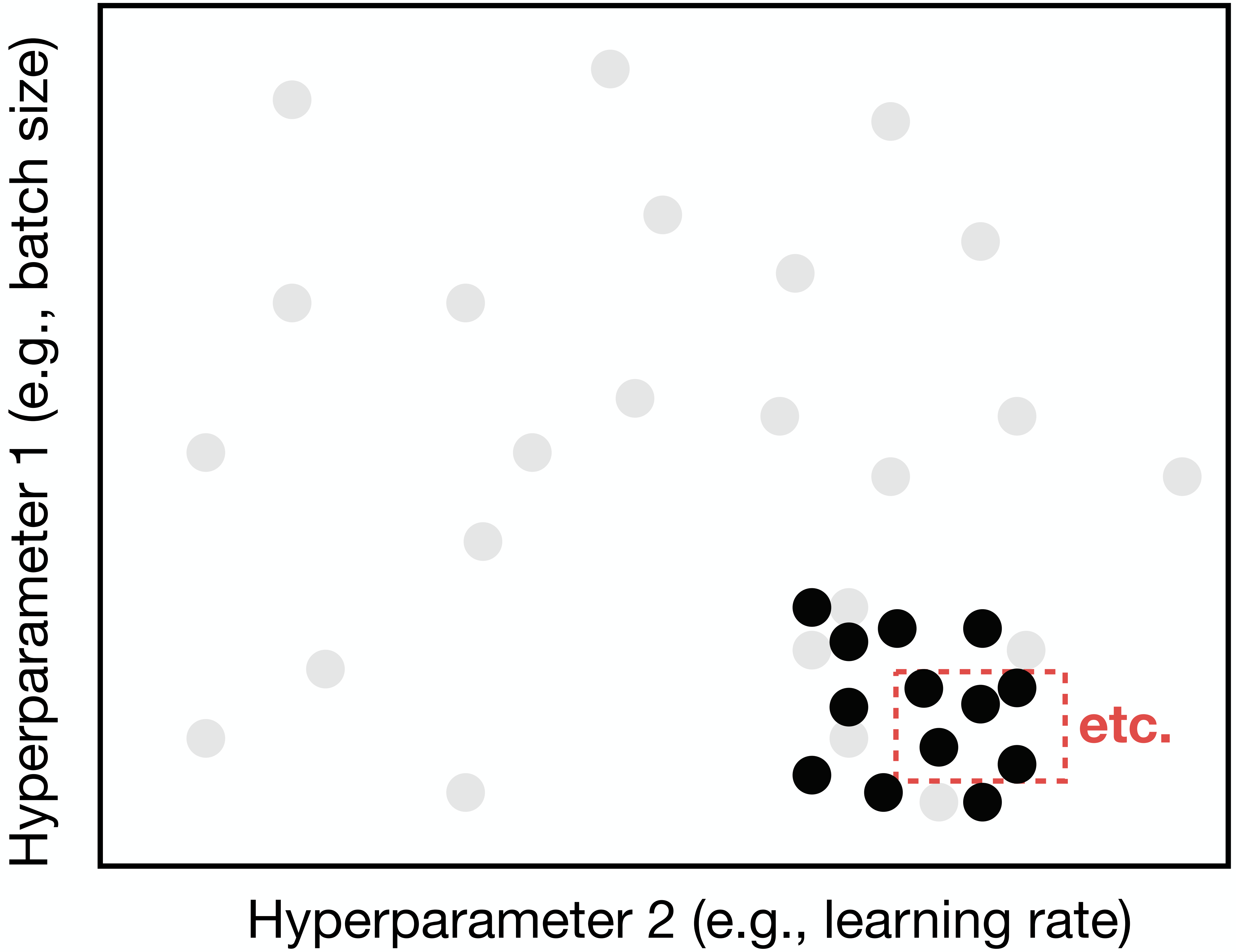


Advantages

Disadvantages

Method 4: coarse-to-fine

How it works



Advantages

- Can narrow in on very high performing hyperparameters
- Most used method in practice

Disadvantages

- Somewhat manual process

Method 5: Bayesian hyperparam opt

How it works (at a high level)

- Start with a prior estimate of parameter distributions
- Maintain a probabilistic model of the relationship between hyper-parameter values and model performance
- Alternate between:
  - Training with the hyper-parameter values that maximize the expected improvement
  - Using training results to update our probabilistic model
- To learn more, see:

https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f

Advantages

- Generally the most efficient hands-off way to choose hyperparameters

Disadvantages

- Difficult to implement from scratch
- Can be hard to integrate with off-the-shelf tools

Summary of how to optimize hyperparams

- Coarse-to-fine random searches
- Consider Bayesian hyper-parameter optimization solutions as your codebase matures

Conclusion

Conclusion

- **DL debugging is hard due to many competing sources of error**
- **To train bug-free DL models, we treat building our model as an iterative process**
- **The following steps can make the process easier and catch errors as early as possible**

How to build bug-free DL models

	Overview
Start simple	• Choose the simplest model & data possible (e.g., LeNet on a subset of your data)
Implement & debug	• Once model runs, overfit a single batch & reproduce a known result
Evaluate	• Apply the bias-variance decomposition to decide what to do next
Tune hyp-eparams	• Use coarse-to-fine random searches
Improve model/data	• Make your model bigger if you underfit; add data or regularize if you overfit

Where to go to learn more

- Andrew Ng's book Machine Learning Yearning (http://www.mlyearning.org/)

- The following Twitter thread: https://twitter.com/karpathy/status/1013244313327681536

- This blog post: https://pcc.cs.byu.edu/2017/10/02/practical-advice-for-building-deep-neural-networks/