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?

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

Your learning curve


Why is your performance worse?

Poor model performance
Why is your performance worse?

- Poor model performance
- Implementation bugs
Most DL bugs are invisible

```
1 features = glob.glob('path/to/features/*')
2 labels  = glob.glob('path/to/labels/*')
3 train(features, labels)
```
Most DL bugs are invisible

Labels out of order!

(real bug I spent 1 day on early in my PhD)
Why is your performance worse?

- Poor model performance
- Implementation bugs

0. Why is troubleshooting hard?
Why is your performance worse?

- Implementation bugs
- Hyperparameter choices

Poor model performance
Models are sensitive to hyperparameters

Andrej Karpathy, CS231n course notes
Models are sensitive to hyperparameters

Andrej Karpathy, CS231n course notes


Why is your performance worse?

- Implementation bugs
- Hyperparameter choices

Poor model performance
Why is your performance worse?

- Implementation bugs
- Data/model fit
- Hyperparameter choices

Poor model performance
0. Why is troubleshooting hard?

Data / model fit

Data from the paper: ImageNet

Yours: self-driving car images
Why is your performance worse?

- Implementation bugs
- Data/model fit
- Poor model performance
- Hyperparameter choices
Why is your performance worse?

- Implementation bugs
- Data/model fit
- Hyperparameter choices
- Dataset construction

Poor model performance
Constructing good datasets is hard

Amount of lost sleep over...

PhD

Tesla

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

Start simple → Implement & debug → Evaluate → Improve model/data → Tune hyper-parameters → Meets requirements

Quick summary

Overview

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

Overview

- Choose the simplest model & data possible (e.g., LeNet on a subset of your data)
- Once model runs, overfit a single batch & reproduce a known result
Quick summary

Overview

- Choose the simplest model & data possible (e.g., LeNet on a subset of your data)
- Once model runs, overfit a single batch & reproduce a known result
- Apply the bias-variance decomposition to decide what to do next
Quick summary

Overview

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

Start simple

Implement & debug

Evaluate

Tune hyperparameters
Quick summary

Overview

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

1. Start simple
2. Implement & debug
3. Evaluate
4. Improve model/data
5. Tune hyper-parameters
6. Meets requirements
Starting simple

Steps

1. Choose a simple architecture
2. Use sensible defaults
3. Normalize inputs
4. Simplify the problem
Demystifying neural network architecture selection

<table>
<thead>
<tr>
<th>Your input data is...</th>
<th>Start here</th>
<th>Consider using this later</th>
</tr>
</thead>
<tbody>
<tr>
<td>Images</td>
<td>LeNet-like architecture</td>
<td>ResNet</td>
</tr>
<tr>
<td>Sequences</td>
<td>LSTM with one hidden layer</td>
<td>Attention model or WaveNet-like model</td>
</tr>
<tr>
<td>Other</td>
<td>Fully connected neural net with one hidden layer</td>
<td>Problem-dependent</td>
</tr>
</tbody>
</table>
Dealing with multiple input modalities

1. Start simple

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

Input 1
- Image of a cat
- ConvNet
- Flatten (64-dim)

Input 2
- Image of a cat
- ConvNet
- Flatten (72-dim)

Input 3
- Text: “This”
- Text: “is”
- Text: “a”
- Text: “cat”
- LSTM
- Flatten (48-dim)
Dealing with multiple input modalities

2. Concatenate

Input 1

Input 2

Input 3

ConvNet

Flatten

Con“cat”

(64-dim)

(72-dim)

(48-dim)

(184-dim)

“This”

“is”

“a”

“cat”


josh-tobin.com/troubleshooting-deep-neural-networks
Dealing with multiple input modalities

3. Pass through fully connected layers to output

Input 1

Input 2

Input 3

ConvNet → Flatten (64-dim) → Concat → FC → FC → Output

ConvNet → Flatten (72-dim) → Concat → FC → FC → Output

ConvNet → Flatten (48-dim) → Concat → FC → FC → Output

LSTM → (48-dim) → Concat → FC → FC → Output

“This” → (184-dim) → FC → FC → Output

“is” → (128-dim) → FC → FC → Output

“a” → (128-dim) → FC → FC → Output

“cat” → (128-dim) → FC → FC → Output
Starting simple

Steps

1. Start simple

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

1. Choose a simple architecture
2. Use sensible defaults
3. Normalize inputs
4. Simplify the problem
Consider simplifying the problem as well

1. Start simple

- 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

Goal: 99% classification accuracy
### Starting simple

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

---

Strategy for DL troubleshooting

1. Start simple
2. Implement & debug
3. Evaluate
4. Tune hyper-parameters
5. Improve model/data
6. Meets requirements
Implementing bug-free DL models

Steps

1. Get your model to run
2. Overfit a single batch
3. 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

Get your model to run

Common issues

Shape mismatch

Casting issue

OOM

Other

Recommended resolution

Step through model creation and inference in a debugger

Scale back memory intensive operations one-by-one

Standard debugging toolkit (Stack Overflow + interactive debugger)
Implementing bug-free DL models

1. Get your model to run

- 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)

2. Implement & debug
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)
```

```bash
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)
```

Debuggers for DL code

- Pytorch: easy, use ipdb
- tensorflow: trickier

Option 2: step into training loop

Evaluate tensors using `sess.run(...)`

```
9  # Option 2: step into training loop
10  sess = tf.Session()
11  for i in range(num_epochs):
12      import ipdb; ipdb.set_trace()
13      loss_, _ = sess.run([loss, train_op])
14```

Debuggers for DL code

- Pytorch: easy, use ipdb
- tensorflow: trickier

Option 3: use tfdb

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

Stops execution at each `sess.run(...)` and lets you inspect
Implementing bug-free DL models

1. Get your model to run

   a. Common issues
      - Shape mismatch
      - Casting issue
      - OOM
      - Other

2. Recommended resolution
   - Step through model creation and inference in a debugger
   - Scale back memory intensive operations one-by-one
   - Standard debugging toolkit (Stack Overflow + interactive debugger)
Implementing bug-free DL models

Shape mismatch

Common issues

Undefined shapes

Incorrect shapes

Most common causes

- 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)
- 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

**Casting issue**

**Common issues**
- Data not in float32

**Most common causes**
- Forgot to cast images from uint8 to float32
- Generated data using numpy in float64, forgot to cast to float32
Implementing bug-free DL models

OOM

Common issues

- Too big a tensor
- Too much data
- Duplicating operations
- Other processes

Most common causes

- Too large a batch size for your model (e.g., during evaluation)
- Too large fully connected layers
- Loading too large a dataset into memory, rather than using an input queue
- Allocating too large a buffer for dataset creation
- 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 running on your GPU
Implementing bug-free DL models

Other common errors

Common issues

Other bugs

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

1. Get your model to run
2. Overfit a single batch
3. Compare to a known result
Implementing bug-free DL models

Overfit a single batch

Common issues
- Error goes up
- Error explodes
- Error oscillates
- Error plateaus

Most common causes
Implementing bug-free DL models

Overfit a single batch

Common issues
- Error goes up
- Error explodes
- Error oscillates
- Error plateaus

Most common causes
- Flipped the sign of the loss function / gradient
- Learning rate too high
- Softmax taken over wrong dimension
Implementing bug-free DL models

Overfit a single batch

Common issues
- Error goes up
- Error explodes
- Error oscillates
- Error plateaus

Most common causes
- Numerical issue. Check all exp, log, and div operations
- Learning rate too high
Implementing bug-free DL models

Overfit a single batch

Common issues

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

Most common causes

- Data or labels corrupted (e.g., zeroed, incorrectly shuffled, or preprocessed incorrectly)
- Learning rate too high
Implementing bug-free DL models

Overfit a single batch

Common issues

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

Most common causes

- 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

Overfit a single batch

Common issues

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

Most common causes

- Flipped the sign of the loss function / gradient
- Learning rate too high
- Softmax taken over wrong dimension
- Numerical issue. Check all exp, log, and div operations
- Learning rate too high
- Data or labels corrupted (e.g., zeroed or incorrectly shuffled)
- Learning rate too high
- 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

1. Get your model to run
2. Overfit a single batch
3. Compare to a known result
Hierarchy of known results

More useful
• Official model implementation evaluated on similar dataset to yours

Less useful

You can:
• Walk through code line-by-line and ensure you have the same output
• Ensure your performance is up to par with expectations
Hierarchy of known results

More useful

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

Less useful

You can:

• Walk through code line-by-line and ensure you have the same output
Hierarchy of known results

- Unofficial model implementation

You can:
- Same as before, but with lower confidence
Hierarchy of known results

- Official model implementation evaluated on similar dataset to yours
- Official model implementation evaluated on benchmark (e.g., MNIST)
- Official model implementation
- Results from a paper (with no code)

More useful

You can:

- Ensure your performance is up to par with expectations

Less useful
Hierarchy of known results

More useful

You can:

• Make sure your model performs well in a simpler setting

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

Less useful
Hierarchy of known results

More useful

• Official model implementation evaluated on similar dataset to yours
• Official model implementation evaluated on benchmark (e.g., MNIST)
• Official 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

Less useful

You can:
• Get a general sense of what kind of performance can be expected
• Results from a similar model on a similar dataset

Hierarchy of known results

More useful

• Official model implementation evaluated on similar dataset to yours
• Official model implementation evaluated on benchmark (e.g., MNIST)
• Official 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

You can:
• Make sure your model is learning anything at all


josh-tobin.com/troubleshooting-deep-neural-networks
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)

Less useful

• Results from a similar model on a similar dataset
• Super simple baselines (e.g., average of outputs or linear regression)
Summary: how to implement & debug

Steps

- Get your model to run
- Overfit a single batch
- Compare to a known result

Summary

- Step through in debugger & watch out for shape, casting, and OOM errors
- Look for corrupted data, over-regularization, broadcasting errors
- Keep iterating until model performs up to expectations
Strategy for DL troubleshooting

Start simple → Implement & debug → Evaluate → Tune hyper-parameters → Meets requirements

Improve model/data → Evaluate

Tune hyper-parameters → Evaluate

Evaluate → Improve model/data

Implement & debug → Evaluate
Bias-variance decomposition

3. Evaluation
Bias-variance decomposition
Bias-variance decomposition

3. Evaluation
Bias-variance decomposition

Irreducible error
Avoidable bias (i.e., underfitting)
Train error (i.e., overfitting)
Val error
Val set overfitting
Test error

Breakdown of test error by source

3. Evaluation

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

Use two val sets: one sampled from training distribution and one from test distribution
The bias-variance tradeoff

![Diagram showing the bias-variance tradeoff with lines for test error, validation error, training error, and human level performance over iterations.](image-url)
Bias-variance with distribution shift
Bias-variance with distribution shift

Irreducible error
Avoidable bias (i.e., underfitting)
Train error
Test error
Variance
Train val error
Val overfitting
Distribution shift
Test val error
Test error

Breakdown of test error by source

3. Evaluation
Train, val, and test error for pedestrian detection

<table>
<thead>
<tr>
<th>Error source</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal performance</td>
<td>1%</td>
</tr>
<tr>
<td>Train error</td>
<td>20%</td>
</tr>
<tr>
<td>Validation error</td>
<td>27%</td>
</tr>
<tr>
<td>Test error</td>
<td>28%</td>
</tr>
</tbody>
</table>

**Goal:** 99% classification accuracy

**Train - goal = 19% (under-fitting)**

<table>
<thead>
<tr>
<th>Running example</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="https://josh-tobin.com/troubleshooting-deep-neural-networks" alt="Images" /></td>
</tr>
</tbody>
</table>
Train, val, and test error for pedestrian detection

<table>
<thead>
<tr>
<th>Error source</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal performance</td>
<td>1%</td>
</tr>
<tr>
<td>Train error</td>
<td>20%</td>
</tr>
<tr>
<td>Validation error</td>
<td>27%</td>
</tr>
<tr>
<td>Test error</td>
<td>28%</td>
</tr>
</tbody>
</table>

Val - train = 7% (over-fitting)

Goal: 99% classification accuracy
## Train, val, and test error for pedestrian detection

<table>
<thead>
<tr>
<th>Error source</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal performance</td>
<td>1%</td>
</tr>
<tr>
<td>Train error</td>
<td>20%</td>
</tr>
<tr>
<td>Validation error</td>
<td>27%</td>
</tr>
<tr>
<td>Test error</td>
<td>28%</td>
</tr>
</tbody>
</table>

### Running example

- **Goal:** 99% classification accuracy

<table>
<thead>
<tr>
<th>0 (no pedestrian)</th>
<th>1 (yes pedestrian)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test - val = 1% (looks good!)</td>
<td>Goal: 99% classification accuracy</td>
</tr>
</tbody>
</table>
Summary: evaluating model performance

Test error = irreducible error + bias + variance
+ distribution shift + val overfitting
Strategy for DL troubleshooting

Start simple → Implement & debug → Evaluate → Tune hyper-parameters → Meets requirements

Improve model/data
Prioritizing improvements
(i.e., applying the bias-variance tradeoff)

Steps

1. Address under-fitting
2. Address over-fitting
3. Address distribution shift
4. 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)

**Try later**
- F. Add features
## 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%

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

- 0 (no pedestrian)
- 1 (yes pedestrian)
Train, val, and test error for pedestrian detection

Switch to ResNet-101

<table>
<thead>
<tr>
<th>Error source</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal performance</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>Train error</td>
<td>20%</td>
<td>10%</td>
<td>3%</td>
</tr>
<tr>
<td>Validation error</td>
<td>27%</td>
<td>19%</td>
<td>10%</td>
</tr>
<tr>
<td>Test error</td>
<td>28%</td>
<td>20%</td>
<td>10%</td>
</tr>
</tbody>
</table>

Goal: 99% classification accuracy (i.e., 1% error)
Train, val, and test error for pedestrian detection

<table>
<thead>
<tr>
<th>Error source</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal performance</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>Train error</td>
<td>20%</td>
<td>10%</td>
<td>3%</td>
<td>0.8%</td>
</tr>
<tr>
<td>Validation error</td>
<td>27%</td>
<td>19%</td>
<td>10%</td>
<td>12%</td>
</tr>
<tr>
<td>Test error</td>
<td>28%</td>
<td>20%</td>
<td>10%</td>
<td>12%</td>
</tr>
</tbody>
</table>

Goal: 99% classification accuracy (i.e., 1% error)
Prioritizing improvements (i.e., applying the bias-variance tradeoff)

Steps

1. Address under-fitting
2. Address over-fitting
3. Address distribution shift
4. 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

Try later

Not recommended!

Train, val, and test error for pedestrian detection

<table>
<thead>
<tr>
<th>Error source</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal performance</td>
<td>1%</td>
</tr>
<tr>
<td>Train error</td>
<td>0.8%</td>
</tr>
<tr>
<td>Validation error</td>
<td>12%</td>
</tr>
<tr>
<td>Test error</td>
<td>12%</td>
</tr>
</tbody>
</table>

Goal: 99% classification accuracy

Running example

0 (no pedestrian) 1 (yes pedestrian)
Train, val, and test error for pedestrian detection

Increase dataset size to 250,000

<table>
<thead>
<tr>
<th>Error source</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal performance</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>Train error</td>
<td>0.8%</td>
<td>1.5%</td>
</tr>
<tr>
<td>Validation error</td>
<td>12%</td>
<td>5%</td>
</tr>
<tr>
<td>Test error</td>
<td>12%</td>
<td>6%</td>
</tr>
</tbody>
</table>

Prioritize improvements

Goal: 99% classification accuracy

Running example:

- 0 (no pedestrian)
- 1 (yes pedestrian)
## Train, val, and test error for pedestrian detection

<table>
<thead>
<tr>
<th>Error source</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal performance</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>Train error</td>
<td>0.8%</td>
<td>1.5%</td>
<td>1.7%</td>
</tr>
<tr>
<td>Validation error</td>
<td>12%</td>
<td>5%</td>
<td>4%</td>
</tr>
<tr>
<td>Test error</td>
<td>12%</td>
<td>6%</td>
<td>4%</td>
</tr>
</tbody>
</table>

### Add weight decay

#### Running example

0 (no pedestrian)  1 (yes pedestrian)

**Goal:** 99% classification accuracy

# Train, val, and test error for pedestrian detection

<table>
<thead>
<tr>
<th>Error source</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal performance</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>Train error</td>
<td>0.8%</td>
<td>1.5%</td>
<td>1.7%</td>
<td>2%</td>
</tr>
<tr>
<td>Validation error</td>
<td>12%</td>
<td>5%</td>
<td>4%</td>
<td>2.5%</td>
</tr>
<tr>
<td>Test error</td>
<td>12%</td>
<td>6%</td>
<td>4%</td>
<td>2.6%</td>
</tr>
</tbody>
</table>

Prioritize improvements:

- Add data augmentation

## Running example

<table>
<thead>
<tr>
<th>0 (no pedestrian)</th>
<th>1 (yes pedestrian)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal: 99% classification accuracy</td>
<td></td>
</tr>
</tbody>
</table>

## Train, val, and test error for pedestrian detection

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

<table>
<thead>
<tr>
<th>Error source</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal performance</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>Train error</td>
<td>0.8%</td>
<td>1.5%</td>
<td>1.7%</td>
<td>2%</td>
<td>0.6%</td>
</tr>
<tr>
<td>Validation error</td>
<td>12%</td>
<td>5%</td>
<td>4%</td>
<td>2.5%</td>
<td>0.9%</td>
</tr>
<tr>
<td>Test error</td>
<td>12%</td>
<td>6%</td>
<td>4%</td>
<td>2.6%</td>
<td>1.0%</td>
</tr>
</tbody>
</table>

### Running example

- **Goal:** 99% classification accuracy
- 0 (no pedestrian)
- 1 (yes pedestrian)

---

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
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

Try later
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

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Use case</th>
<th>Example techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Supervised</strong></td>
<td>You have limited data from target domain</td>
<td>• Fine-tuning a pre-trained model&lt;br&gt;• Adding target data to train set</td>
</tr>
<tr>
<td><strong>Un-supervised</strong></td>
<td>You have lots of unlabeled data from target domain</td>
<td>• Correlation Alignment (CORAL)&lt;br&gt;• Domain confusion&lt;br&gt;• CycleGAN</td>
</tr>
</tbody>
</table>
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

Start simple → Implement & debug → Evaluate → Improve model/data → Tune hyperparameters → Meets requirements

Tune hyperparameters

Evaluate

Improve model/data

Start simple

Implement & debug

Evaluate
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

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)

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Approximate sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning rate</td>
<td>High</td>
</tr>
<tr>
<td>Optimizer choice</td>
<td>Low</td>
</tr>
<tr>
<td>Other optimizer params (e.g., Adam beta1)</td>
<td>Low</td>
</tr>
<tr>
<td>Batch size</td>
<td>Low</td>
</tr>
<tr>
<td>Weight initialization</td>
<td>Medium</td>
</tr>
<tr>
<td>Loss function</td>
<td>High</td>
</tr>
<tr>
<td>Model depth</td>
<td>Medium</td>
</tr>
<tr>
<td>Layer size</td>
<td>High</td>
</tr>
<tr>
<td>Layer params (e.g., kernel size)</td>
<td>Medium</td>
</tr>
<tr>
<td>Weight of regularization</td>
<td>Medium</td>
</tr>
<tr>
<td>Nonlinearity</td>
<td>Low</td>
</tr>
</tbody>
</table>
Method 1: manual hyperparam optimization

**How it works**
- Understand the algorithm
  - E.g., higher learning rate means faster but 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 optimize 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

<table>
<thead>
<tr>
<th>How it works</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
</table>
| Hyperparameter 1 (e.g., batch size) | • Super simple to implement  
• Can produce good results | • Not very efficient: need to train on all cross-combos of hyper-parameters  
• May require prior knowledge about parameters to get good results |
| Hyperparameter 2 (e.g., learning rate) | | |
Method 3: random search

How it works

Hyperparameter 1 (e.g., batch size) vs. Hyperparameter 2 (e.g., learning rate)

Advantages

Disadvantages
Method 4: coarse-to-fine

**How it works**

| Hyperparameter 1 (e.g., batch size) | Hyperparameter 2 (e.g., learning rate) |

**Advantages**

**Disadvantages**

- Possible overfitting
- Longer training time
- Requires more computational resources
Method 4: coarse-to-fine

Hyperparameter 1 (e.g., batch size)

Hyperparameter 2 (e.g., learning rate)

How it works
Best performers

Advantages

Disadvantages
Method 4: coarse-to-fine

How it works

Hyperparameter 1 (e.g., batch size)

Hyperparameter 2 (e.g., learning rate)

Advantages

Disadvantages
Method 4: coarse-to-fine

How it works

Hyperparameter 1 (e.g., batch size)

Hyperparameter 2 (e.g., learning rate)

Advantages

Disadvantages

5. Hyperparameter optimization
Method 4: coarse-to-fine

**How it works**

| Hyperparameter 1 (e.g., batch size) | Hyperparameter 2 (e.g., learning rate) |

**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:

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

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

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

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

- Use coarse-to-fine random searches

- 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/