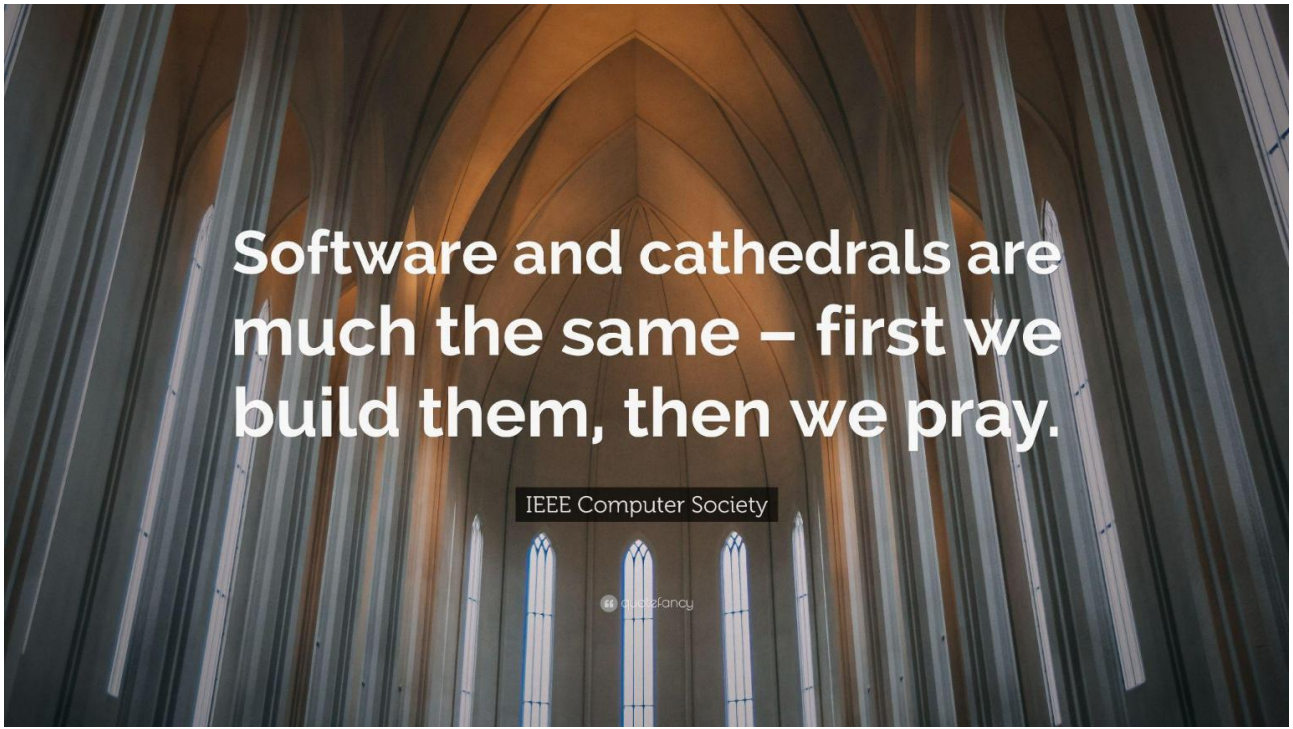
4. Debugging

Debugging software can be painful



https://quotefancy.com/quote/3753413/IEEE-Computer-Society-Software-and-cathedrals-are-much-the-same-first-we-build-them-then

Machine Learning debugging can be even more tedious

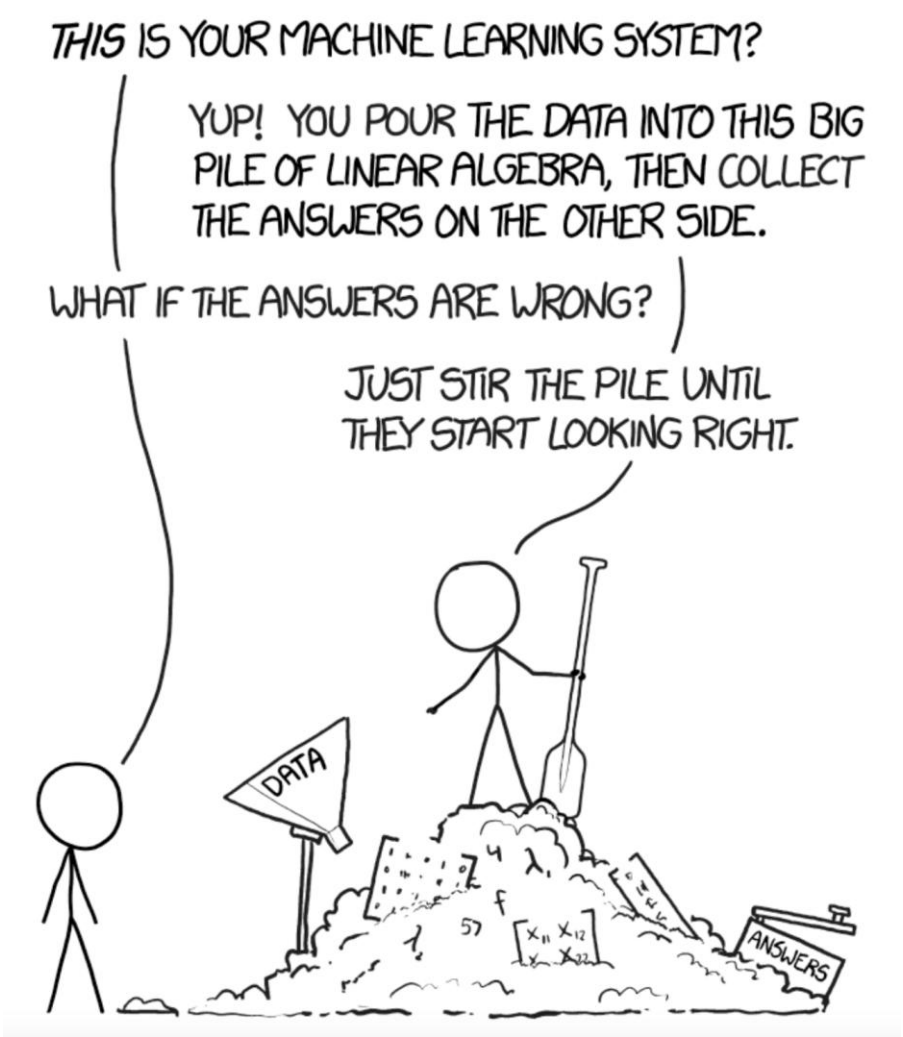
Laura Ruis
@LauraRuis

Follow

when you debug your NN for 2 days until you realise the lr is 1e3 instead of 1e-3 🙂 COOL

5:21 PM · Feb 9, 2023 · **91.3K** Views

37 18 905 12



https://xkcd.com/1838/

Machine Learning development has the reputation of a *trial-and-error, messy* process

If we are principled about it, we can make **debugging ML projects more predictable**

Software Engineering

- **more predictable:** logic is defined by code

Machine Learning

- **less predictable:** ‘logic’ is defined by code + data + training dynamics

Software Engineering

- **more predictable:** logic is defined by code
- **fewer sources of errors:** typically logic bugs or integration issues

Machine Learning

- **less predictable:** ‘logic’ is defined by code + data + training dynamics
- **more sources of error:** data quality, wrong architecture or hyperparameters

Software Engineering

- **more predictable:** logic is defined by code
- **fewer sources of errors:** typically logic bugs or integration issues
- **explainability:** easier to trace back why system behaved a certain way

Machine Learning

- **less predictable:** ‘logic’ is defined by code + data + training dynamics
- **more sources of error:** data quality, wrong architecture or hyperparameters
- **explainability**: ML systems are more opaque => need to look at training dynamics, activations, feature importance etc.

Plan for today

- learn how to tackle the most common bugs in ML lifecycle
  - **syntactic bugs**
  - **functional bugs**
- inspect learning curves:
  - look for signs of good fit/overfitting/underfitting
  - recognize potential ***hyperparameters misconfigurations***

hopefully, by the end of this lecture you will be more confident in when debugging ML projects 

Machine Learning bugs

syntactic

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-4-59c95019453c> in <module>()
      1 model = Sequential()
----> 2 model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(50,)))
      3 model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
      4 model.add(Dropout(0.5))
      5 model.add(MaxPooling1D(pool_size=2))

~/anaconda3/lib/python3.6/site-packages/keras/engine/sequential.py in add(self, layer)
    163                     # and create the node connecting the current layer
    164                     # to the input layer we just created.
--> 165                     layer(x)
    166                     set_inputs = True
    167             else:

~/anaconda3/lib/python3.6/site-packages/keras/engine/base_layer.py in __call__(self, inputs, **kwargs)
    412                 # Raise exceptions in case the input is not compatible
    413                 # with the input_spec specified in the layer constructor.
--> 414                 self.assert_input_compatibility(inputs)
    415
    416                 # Collect input shapes to build layer.

~/anaconda3/lib/python3.6/site-packages/keras/engine/base_layer.py in assert_input_compatibility(self, inputs)
    309                                     self.name + ': expected ndim=' +
    310                                     str(spec.ndim) + ', found ndim=' +
--> 311                                     str(K.ndim(x)))
    312             if spec.max_ndim is not None:
    313                 ndim = K.ndim(x)

ValueError: Input 0 is incompatible with layer conv1d_1: expected ndim=3, found ndim=2
```

functional

Wrong tensor shape

- one of the most common Deep Learning errors
- usually verbose, but can lead to silent bugs

Wrong tensor shape - wrong input shape

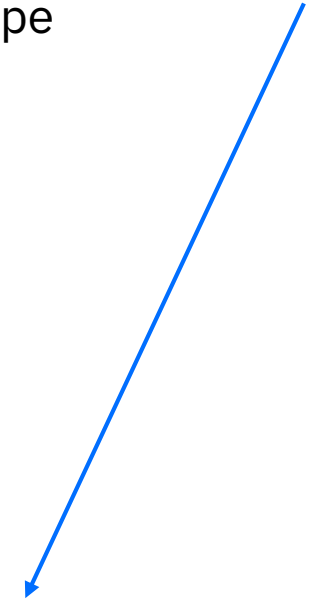
```python
class SimpleMLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(10, 20)
        self.act1 = nn.ReLU()
        self.fc2 = nn.Linear(20, 2)

    def forward(self, x):
        h = self.act1(self.fc1(x))
        o = self.fc2(h)
        return o
```

```python
model = SimpleMLP()
x = torch.rand(3, 11)
model(x)
```

```
      8     def forward(self, x):
----> 9         h = self.act1(self.fc1(x))
     10         o = self.fc2(h)
     11         return o
```

```
RuntimeError: mat1 and mat2 shapes cannot be multiplied (3x11 and 10x20)
```

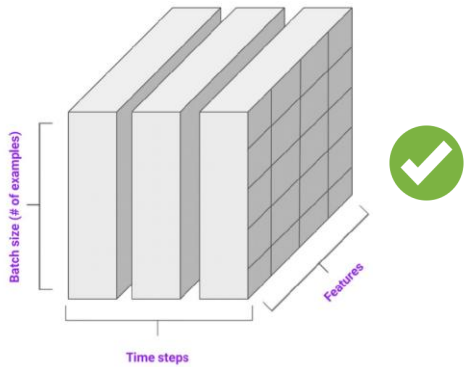
- a lot of the times, the *input data* has the wrong shape



Wrong tensor shape - wrong input shape

typical scenarios:

- *images*:
  - input Tensor has shape batch x W x H x channels instead of **batch x channels x W x H**
  - inputs are greyscale and have batch x W x H
- *sequential data*: input Tensor has shape batch x H x timesteps instead of **batch x timesteps x H**



Wrong tensor shape - wrong input shape

scenario: last batch in a dataset may have 1st dimension != **batch_size:**

- why? let's say **batch_size=32**
- ***DataLoader*** loads 32 elements until it exhausts all the dataset elements
- if dataset has 3216 elements => ***last batch*** has 16 elements

Wrong tensor shape - wrong input shape

scenario: last batch in a dataset may have 1st dimension != **batch_size:**

- if code assumes output size is **batch_size**, shape errors may appear:

```
# resize images: 32x3x32x32 => 32x(3x32x32)
print("[train_epoch] batch_img before resize = ", batch_img.size())
batch_img = batch_img.view(32, -1)
```

```
[train_epoch] batch_img before resize =  torch.Size([16, 3, 32, 32])
[train_epoch] batch_img after resize =  torch.Size([32, 1536])
```

new input size (1536) no longer matches layer size (3x3x32=3072)

```
RuntimeError: mat1 and mat2 shapes cannot be multiplied (32x1536 and 3072x256)
```

Wrong tensor shape - wrong label shape

labels are one-hot but loss function expects scalar ***[0, num_classes-1]***

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

→

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

source: https://www.thetalkingmachines.com/article/beginners-guide-debugging-tensorflow-models

Wrong tensor shape - wrong output shape

output doesn’t conform to *shape expected by loss function*

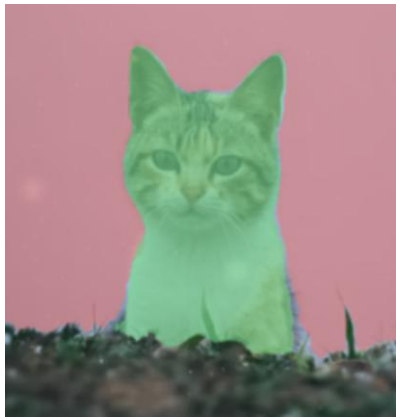
The *input* is expected to contain raw, unnormalized scores for each class. *input* has to be a Tensor of size either $(minibatch, C)$ or $(minibatch, C, d_1, d_2, ..., d_K)$ with $K \geq 1$ for the *K*-dimensional case. The latter is useful

documentation snippet from CrossEntropyLoss in Pytorch

Wrong tensor shape - wrong output shape

typical scenarios:

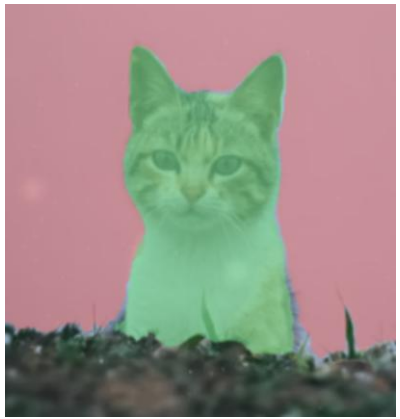
- sequential data:
  - if each timestep is classified into C classes
  - output for `CrossEntropyLoss` should be **batch x C x timesteps**, not batch x timesteps x C
- semantic segmentation:
  - C classes
  - output for `CrossEntropyLoss` should be **?**



Wrong tensor shape - wrong output shape

typical scenarios:

- sequential data:
  - if each timestep is classified into C classes
  - output for `CrossEntropyLoss` should be **batch x C x timesteps**, not batch x timesteps x C
- semantic segmentation:
  - C classes
  - output for `CrossEntropyLoss` should be **batch x C x H x W**



Wrong tensor shape - intermediate ops

- during feedforward, tensors are manipulated using different ops (**slice, concat, max, reshape** etc.)
- common scenario:
  - calling functions with the **wrong arguments** *alters tensor shapes unexpectedly*
  - the resulting **tensor shape no longer matches with the next layer**

Wrong tensor shape - intermediate ops

Wrong tensor shape - intermediate ops

scenario: **concatenating** two tensors on the *wrong dimension*

```
BATCH_SIZE = 16
# batch_size x 50
h1 = torch.rand(BATCH_SIZE, 50)
h2 = torch.rand(BATCH_SIZE, 50)

h_cat = torch.cat((h1, h2))
print(h_cat.size())
```

```
torch.Size([32, 50])
```

cat function concatenates on **dim=0** by default!

Wrong tensor shape - intermediate ops

scenario: **concatenating** two tensors on the *wrong dimension*

good practice: specify dimensions explicitly whenever possible (***dim*** argument)

```
BATCH_SIZE = 16
# batch_size x 50
h1 = torch.rand(BATCH_SIZE, 50)
h2 = torch.rand(BATCH_SIZE, 50)

h_cat = torch.cat((h1, h2))
print(h_cat.size())
```

```
torch.Size([32, 50])
```

cat function concatenates on **dim=0** by default!

```
BATCH_SIZE = 16
# batch_size x 50
h1 = torch.rand(BATCH_SIZE, 50)
h2 = torch.rand(BATCH_SIZE, 50)

h_cat = torch.cat((h1, h2), dim=1)
print(h_cat.size())
```

```
torch.Size([16, 100])
```

the two inputs are correctly concatenated

Wrong tensor shape - intermediate ops

scenario:

- tensor has batch_size=1 and some other dimension=1
- `torch.squeeze()` removes all dimensions of 1

good practice: explicitly specify squeeze dimensions (***dim*** argument)

```
[ ]  1  def forward(self, x):
     2      # x shape: 1 x 100 x 1
     3      y = x.squeeze()
     4
     5      # y shape: 100 (instead of 1 x 100)
```

Wrong tensor shape - intermediate ops

```
[27]  1  class SimpleNet(nn.Module):
      2      def __init__(self):
      3          super().__init__()
      4          self.conv_layer = nn.Conv2d(
      5              in_channels=3,
      6              out_channels=3,
      7              kernel_size=1
      8          )
      9          self.lin = nn.Linear(in_features=3072, out_features=64)
     10
     11      def forward(self, x):
     12          # x shape: batch_size x 3 x 32 x 32
     13          # o1 shape: batch_size x 3 x 32 x 32
     14          o1 = self.conv_layer(x)
     15
     16          # o2 shape: batch_size x 64
     17          o2 = self.lin(o1)
     18
     19          return o2
```

```
[29]  1  net = SimpleNet()
      2  batch_size = 10
      3  x = torch.rand(batch_size, 3, 32, 32)
      4  out = net(x)
      5  print(out.shape)
```

```
    115      def forward(self, input: Tensor) -> Tensor:
--> 116          return F.linear(input, self.weight, self.bias)
    117
    118      def extra_repr(self) -> str:

RuntimeError: mat1 and mat2 shapes cannot be multiplied (960x32 and 3072x64)
```

scenario:

- *linear layer* after *conv2d layer*
- linear expects input=3x32x32=3072
- linear layer 😢

Wrong tensor shape - intermediate ops

```python
class SimpleNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv_layer = nn.Conv2d(
            in_channels=3,
            out_channels=3,
            kernel_size=1
        )
        self.lin = nn.Linear(in_features=3072, out_features=64)

    def forward(self, x):
        # x shape: batch_size x 3 x 32 x 32
        # o1 shape: batch_size x 3 x 32 x 32
        batch_size = x.size(0)
        o1 = self.conv_layer(x)

        # o1 shape: batch_size x 3072
        o1 = o1.reshape(batch_size, -1)

        # o2 shape: batch_size x 64
        o2 = self.lin(o1)

        return o2
```

```
[33]
net = SimpleNet()
batch_size = 10
x = torch.rand(batch_size, 3, 32, 32)
out = net(x)
print(out.shape)
```

```
torch.Size([10, 64])
```

scenario:

- *linear layer* after *conv2d layer*
- linear expects input=3x32x32=3072
- reshape input

Model summary

- tools such as torchinfo show expected input/output shapes for each layer

```
from torchinfo import summary

model = ConvNet()
batch_size = 16
summary(model, input_size=(batch_size, 1, 28, 28))
```

```
==========================================================================================
Layer (type:depth-idx)                   Input Shape               Output Shape              Param #
==========================================================================================
SingleInputNet                           [7, 1, 28, 28]            [7, 10]                   --
├─Conv2d: 1-1                            [7, 1, 28, 28]            [7, 10, 24, 24]           260
├─Conv2d: 1-2                            [7, 10, 12, 12]           [7, 20, 8, 8]             5,020
├─Dropout2d: 1-3                         [7, 20, 8, 8]             [7, 20, 8, 8]             --
├─Linear: 1-4                            [7, 320]                  [7, 50]                   16,050
├─Linear: 1-5                            [7, 50]                   [7, 10]                   510
==========================================================================================
Total params: 21,840
Trainable params: 21,840
Non-trainable params: 0
Total mult-adds (M): 3.41
==========================================================================================
```

Wrong tensor shape - good practices

- write down tensor shapes after each operation

```
# batch_size x 50
h1 = torch.rand(BATCH_SIZE, 50)
h2 = torch.rand(BATCH_SIZE, 50)

# batch_size x 100
h_cat = torch.cat((h1, h2), dim=1)
```

- use tools such as **torchinfo** to detect *potential layer mismatches*
- when debugging, *log all the tensor shapes* leading up to the shape error
- carefully read Pytorch documentation for layers (linear, conv2d etc), operations (concat, reshape, etc.) and loss functions

Wrong tensor device

- key idea: most operations require Tensors **to be on the same device**
- this is also true for distributed training (multi-GPU training)
  - the distributed framework takes care of syncing gradients and parameters
  - local computation still has to be performed on the same GPU
- mixing devices (**CPU**->**GPU**, or different **GPUs**) results in runtime errors

```
-> 2846     return torch._C._nn.cross_entropy_loss(input, target, weight, _Reduction.get_enum(reduction), ignore_index,
label_smoothing)
   2847
   2848

RuntimeError: Expected all tensors to be on the same device, but found at least two devices, cuda:0 and cpu! (when checking
argument for argument target in method wrapper_nll_loss_forward)
```

Wrong tensor device

common scenarios:

- model was moved to GPU but input data is still on CPU due to **.to()** operation

```
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

# Module.to() is an in-place operation
model.to(device)

# Tensor.to() is not an in-place operation
images=images.to(device)
```

- model was loaded from a checkpoint but we forgot to move it to GPU
- forgot to move labels to GPU

Wrong tensor device - best practices

- write device agnostic code to easily switch between the two

```
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
```

- make sure model, inputs and labels **are on the same device**
  - remember DataLoader usually loads data on CPU
  - pay attention to .to() operations
  - log model/tensor device info

Wrong tensor type

- PyTorch ops expect certain Tensor types (**float32** for operations, **long** for indices/labels)
- Mismatched types usually lead to runtime errors

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py in
   1846     if has_torch_function_variadic(input, weight, bias):
   1847         return handle_torch_function(linear, (input, wei
-> 1848     return torch._C._nn.linear(input, weight, bias)
   1849
   1850

RuntimeError: expected scalar type Float but found Double
```

Wrong tensor type

input data and network mismatch:

- parameters in **torch.nn** layers have type ***float32*** (Float) by default
- NumPy arrays are converted to ***float64*** (Double) Tensors => incompatible with layers

```
1   x = torch.rand(3, 18)
2   print(x.dtype)
```

```
torch.float32
```

```
1   y = torch.tensor(np.array([1.07, 3.87, 4.32]))
2   print(y.dtype)
```

```
torch.float64
```

Wrong tensor type

input data and network mismatch:

- pretrained model parameters have half-precision (***float16***)
- input is ***float32***

```
1  model = MLP(input_size=3072, hidden_size=1024, activation_fn=nn.ReLU())
2  model.half() # convert parameters to float16
3  x = torch.rand(3072) # float32 input
4  out = model(x)
```

3 frames

```
/usr/local/lib/python3.9/dist-packages/torch/nn/modules/linear.py in forward(self, input)
    112
    113     def forward(self, input: Tensor) -> Tensor:
--> 114         return F.linear(input, self.weight, self.bias)
    115
    116     def extra_repr(self) -> str:

RuntimeError: mat1 and mat2 must have the same dtype
```

Wrong tensor type

labels and loss mismatch:

- labels are read as ***Float*** Tensors
- we forget to convert them to ***Long*** Tensors, required by **nn.CrossEntropyLoss**

5 frames

```
/usr/local/lib/python3.6/dist-packages/torch/nn/functional.py in nll_loss(input, target, weight, size_average, ignore_index,
   2113                          .format(input.size(0), target.size(0)))
   2114     if dim == 2:
-> 2115         ret = torch._C._nn.nll_loss(input, target, weight, _Reduction.get_enum(reduction), ignore_index)
   2116     elif dim == 4:
   2117         ret = torch._C._nn.nll_loss2d(input, target, weight, _Reduction.get_enum(reduction), ignore_index)

RuntimeError: expected scalar type Long but found Double
```

Wrong tensor type - best practices

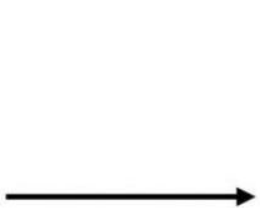
- inspect Tensor types:
  - use `tensor.dtype` to check types before key operations
- check documentation:
  - Loss Functions:
    - nn.CrossEntropyLoss expects **Float** outputs and **Long** labels
  - Indexing and Embedding layers:
    - nn.Embedding expects **Long** inputs (indices of words)
    - torch.gather() expects **Long** indices

Wrong tensor range

- labels are stored as **{1,2,...,C}** instead of **{0,1,...,C-1}**
- this throws an error in the cost function when accessing the score or label with index C (out of range)

Original Data

Team	Points
1	25
1	12
2	15
2	14
2	19
2	23
3	25
3	29

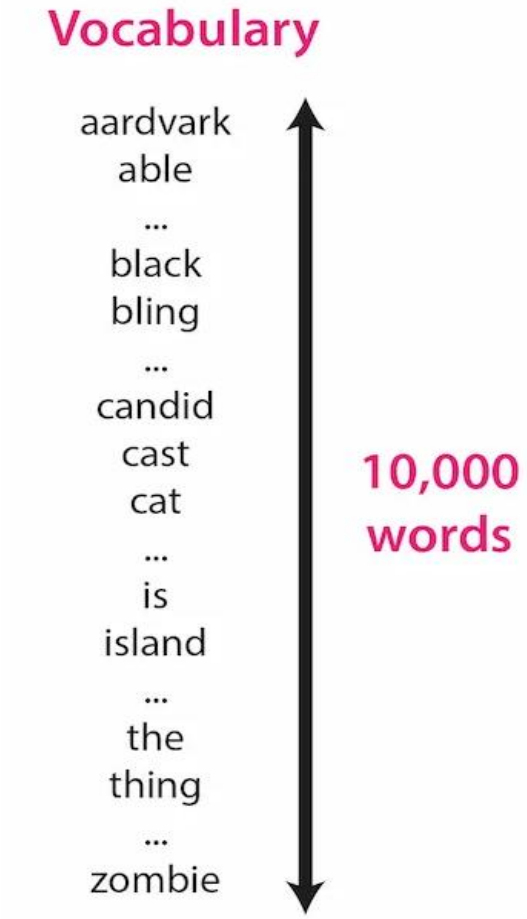


Label Encoded Data

Team	Points
0	25
0	12
1	15
1	14
1	19
1	23
2	25
2	29

Wrong tensor range

- when working with text data, each word has a unique index **{0..V-1}**
- for **V** = 10000, `x[10000]` results in an ‘index out of range’ error



source

Out of memory errors

Suhail
@Suhail

This is currently the bane of my existence:
"RuntimeError: CUDA out of memory. Tried to allocate 220.00 MiB (GPU 0; 15.78 GiB total capacity; 13.16 GiB already allocated; 201.44 MiB free; 14.44 GiB reserved in total by PyTorch)"

This must be an AI rite of passage.

3:41 AM · Jul 20, 2022

Out of memory errors - causes

- memory inefficiency:
  - tensors are not detached (either outputs or losses)

Out of memory errors - causes

- memory inefficiency:
  - tensors are not detached (either outputs or losses)
- large batch size
  - increases memory requirements

Out of memory errors - causes

- memory inefficiency:
  - tensors are not detached (either outputs or losses)
- large batch size
  - increases memory requirements
- large model:
  - model size:
    - 7B model needs 14GB VRAM memory just to store its FP16 weights
  - training overhead:
    - need to store activations/gradients/optimizer states => a 7B model requires ~90-100GB VRAM memory

Out of memory errors - test time

- check for memory inefficiencies:
  - wrap forward code in `torch.no_grad()`, which disables gradient calculation and reduces memory consumption
  - if you save outputs (segmentation maps, probabilities) for later inspection, move Tensors to CPU

Out of memory errors - test time

- check for memory inefficiencies:
  - wrap forward code in `torch.no_grad()`, which disables gradient calculation and reduces memory consumption
  - if you save outputs (segmentation maps, probabilities) for later inspection, move them to CPU
- reduce precision:
  - convert model to FP16/BF16 if supported
  - quantize model (INT8)

Out of memory errors - test time

- check for memory inefficiencies:
  - wrap forward code in `torch.no_grad()`, which disables gradient calculation and reduces memory consumption
  - if you save outputs (segmentation maps, probabilities) for later inspection, move them to CPU
- reduce precision:
  - convert model to FP16/BF16 if supported
  - quantize model (INT8)
- reduce input:
  - reduce batch size
  - feed shorter sequences (if working with text/time series etc.)

Out of memory errors - train time

- check for memory leaks (detach loss from computational graph before storing it)

Out of memory errors - train time

- check for memory leaks (detach loss from computational graph before storing it)
- reduce model training footprint:
  - train with mixed precision (FP16 + FP32)
  - train with parameter-efficient methods (LoRA)
  - reduce model size (less layers etc.) if training from scratch

Out of memory errors - train time

- check for memory leaks (detach loss from computational graph before storing it)
- reduce model training footprint:
  - train with mixed precision (FP16 + FP32)
  - train with parameter-efficient methods (LoRA)
  - reduce model size (less layers etc.) if training from scratch
- reduce batch size:
  - use smaller BS with gradient accumulation to simulate larger batches

Out of memory errors - train time

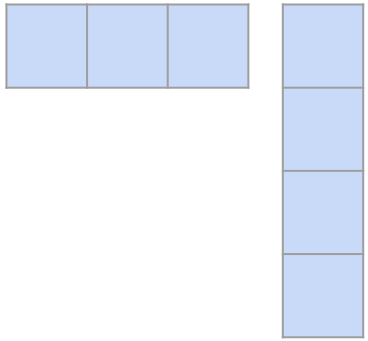
- check for memory leaks (detach loss from computational graph before storing it)
- reduce model training footprint:
  - train with mixed precision (FP16 + FP32)
  - train with parameter-efficient methods (LoRA)
  - reduce model size (less layers etc.) if training from scratch
- reduce batch size:
  - use smaller BS with gradient accumulation to simulate larger batches
- activation checkpointing:
  - recompute activations during backprop to save memory

Out of memory errors - train time

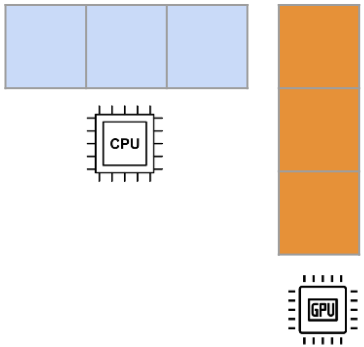
- check for memory leaks (detach loss from computational graph before storing it)
- reduce model training footprint:
  - train with mixed precision (FP16 + FP32)
  - train with parameter-efficient methods (LoRA)
  - reduce model size (less layers etc.) if training from scratch
- reduce batch size:
  - use smaller BS with gradient accumulation to simulate larger batches
- activation checkpointing:
  - recompute activations during backprop to save memory
- if more GPUs available, model parallelism:
  - split model to several GPUs

Syntactic Issues - recap

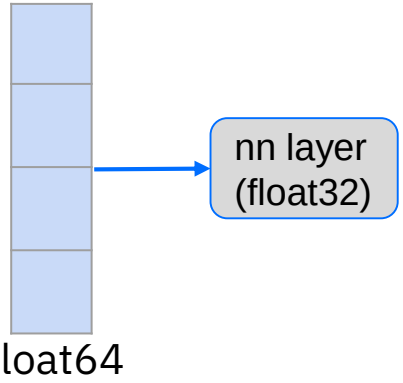
shape errors



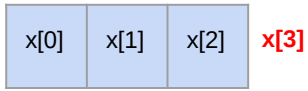
device errors



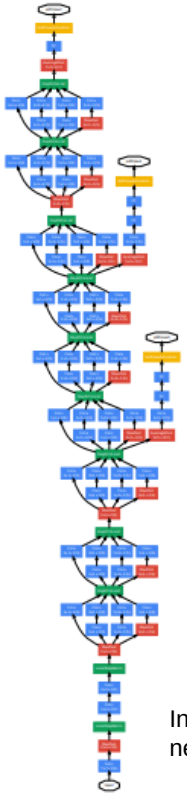
type errors



range errors



OOM errors



Inception network

Machine Learning bugs

syntactic

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-4-59c95019453c> in <module>()
      1 model = Sequential()
----> 2 model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(50,)))
      3 model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
      4 model.add(Dropout(0.5))
      5 model.add(MaxPooling1D(pool_size=2))

~/anaconda3/lib/python3.6/site-packages/keras/engine/sequential.py in add(self, layer)
    163                     # and create the node connecting the current layer
    164                     # to the input layer we just created.
--> 165                     layer(x)
    166                     set_inputs = True
    167             else:

~/anaconda3/lib/python3.6/site-packages/keras/engine/base_layer.py in __call__(self, inputs, **kwargs)
    412                 # Raise exceptions in case the input is not compatible
    413                 # with the input_spec specified in the layer constructor.
--> 414                 self.assert_input_compatibility(inputs)
    415
    416                 # Collect input shapes to build layer.

~/anaconda3/lib/python3.6/site-packages/keras/engine/base_layer.py in assert_input_compatibility(self, inputs)
    309                                     self.name + ': expected ndim=' +
    310                                     str(spec.ndim) + ', found ndim=' +
--> 311                                     str(K.ndim(x)))
    312             if spec.max_ndim is not None:
    313                 ndim = K.ndim(x)

ValueError: Input 0 is incompatible with layer conv1d_1: expected ndim=3, found ndim=2
```

functional

Machine Learning functional bugs

- no syntax errors present
- however, model either:
  - fails to learn the train data
  - doesn’t generalize to new data

Functional bugs - sources of error

- **logic errors** (incorrect arguments, wrong labels)
- **dataset issues** (data preprocessing, augmentation, train/test mismatch)
- **hyperparameter misconfigurations** (learning rates, regularization issues)

Logic errors

normalization/standardization is:

- missing
- computed over the wrong dimension

Age	Salary	Volume
28	26,000	4
22	32,000	10
25	29,000	86
24	38,000	2
26	27,000	71
22	42,000	88
25	41,000	90
23	26,500	72
22	29,500	14
24	36,000	8

Logic errors

normalization/standardization is:

- missing
- computed over the wrong dimension

Age	Salary	Volume
28	26,000	4
22	32,000	10
25	29,000	86
24	38,000	2
26	27,000	71
22	42,000	88
25	41,000	90
23	26,500	72
22	29,500	14
24	36,000	8

```
[[0.001 1.    0.   ]
 [0.    1.    0.   ]
 [0.    1.    0.002]
 [0.001 1.    0.   ]
 [0.    1.    0.002]
 [0.    1.    0.002]
 [0.    1.    0.002]
 [0.    1.    0.002]
 [0.    1.    0.   ]
 [0.    1.    0.   ]]
```

Logic errors

normalization/standardization is:

- missing
- computed over the wrong dimension

Age	Salary	Volume
28	26,000	4
22	32,000	10
25	29,000	86
24	38,000	2
26	27,000	71
22	42,000	88
25	41,000	90
23	26,500	72
22	29,500	14
24	36,000	8

```
[[1.   , 0.   , 0.023],
 [0.   , 0.375, 0.091],
 [0.5  , 0.188, 0.955],
 [0.333, 0.75 , 0.   ],
 [0.667, 0.062, 0.784],
 [0.   , 1.   , 0.977],
 [0.5  , 0.938, 1.   ],
 [0.167, 0.031, 0.795],
 [0.   , 0.219, 0.136],
 [0.333, 0.625, 0.068]])
```

Logic errors

softmax is:

applied over the wrong dimension

```
BATCH_SIZE = 8
# batch_size x 2
output_scores = torch.rand(BATCH_SIZE, 2)
softmax = nn.Softmax(dim=0)
# batch_size x 2
output_probabilities = softmax(output_scores)
print(output_probabilities)
```

```
tensor([[0.1392, 0.1821],
        [0.1809, 0.0744],
        [0.1111, 0.1447],
        [0.1298, 0.0712],
        [0.0753, 0.0696],
        [0.0687, 0.1845],
        [0.1725, 0.1599],
        [0.1225, 0.1135]])
```

Logic errors

softmax is:

applied over the wrong dimension

unnecessary

```
BATCH_SIZE = 8
# batch_size x 2
output_scores = torch.rand(BATCH_SIZE, 2)
softmax = nn.Softmax(dim=0)
# batch_size x 2
output_probabilities = softmax(output_scores)
print(output_probabilities)
```

```
tensor([[0.1392, 0.1821],
        [0.1809, 0.0744],
        [0.1111, 0.1447],
        [0.1298, 0.0712],
        [0.0753, 0.0696],
        [0.0687, 0.1845],
        [0.1725, 0.1599],
        [0.1225, 0.1135]])
```

```
loss_fn = nn.CrossEntropyLoss()
y = torch.tensor([1, 0, 0, 1, 1, 1, 0, 0])

# batch_size x 2
output_probabilities = softmax(output_scores)

loss = loss_fn(output_probabilities, y)
```


nn.CrossEntropyLoss() requires *raw unnormalized* scores as input

B

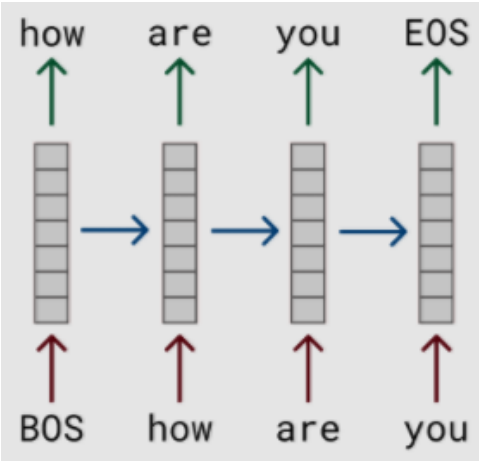
Logic errors

train/evaluation mode mismatch:

- when model is set in *eval()* mode, some operations behave differently (*Dropout*, *Batch Normalization* etc.)
- **common mistake**: after evaluation, model is **NOT** set back to *train mode*, which results in unwanted behaviour:
  - dropout remains disabled during training => no more regularization
  - batch normalization uses wrong estimates of mean and variance => bad updates

Logic errors

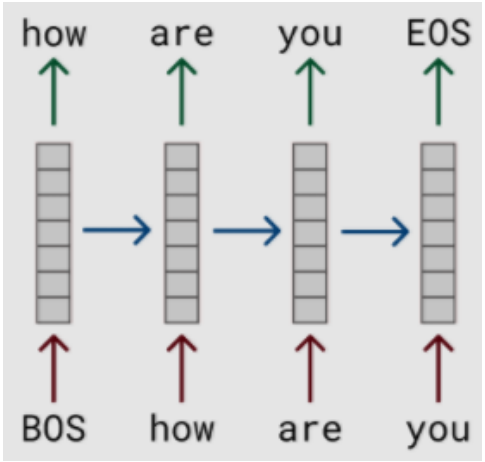
- in language models, for each input word you have to predict the *next* word
- labels = inputs shifted by one to the left



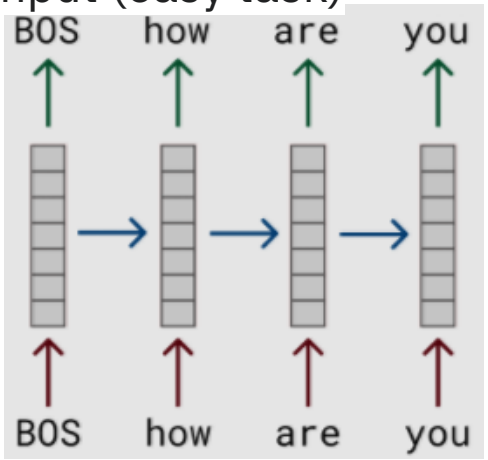
BOS = beginning of sentence, EOS = end of sentence

Logic errors

- in language models, for each input word you have to predict the *next* word
- labels = inputs shifted by one to the left



- if labels are not shifted properly, model solves different task by mistake
- in this example, model predicts its own input (easy task)



BOS = beginning of sentence, EOS = end of sentence

Dataset issues

check if data is faulty: mislabeled, unshuffled, improperly augmented etc.

Dataset issues

- **NaN** (not a number) value appears in input data by mistake (missing/corrupted data, preprocessing error etc.)
- **no syntax errors** raised during:
  - feedforward
  - loss computation
  - backpropagation

```
BATCH_SIZE = 8
model = SimpleMLP()
loss_fn = nn.CrossEntropyLoss()
y = torch.tensor([1, 0, 0, 1, 1, 0, 0, 0])

# batch_size x 10
input = torch.rand(BATCH_SIZE, 10)
input[0,0] = float('nan')

# batch_size x 2
output_scores = model(input)
loss = loss_fn(output_scores, y)
print(loss)
```

```
tensor(nan, grad_fn=<NllLossBackward>)
```

```
loss.backward()
for p in model.parameters():
    print(p.grad)
```

```
tensor([[nan, nan, nan, nan, nan, nan, nan, nan, nan, nan],
```

Dataset issues

- use `torch.isnan(x).any()` to manually check tensors
- use `torch.autograd.detect_anomaly()`

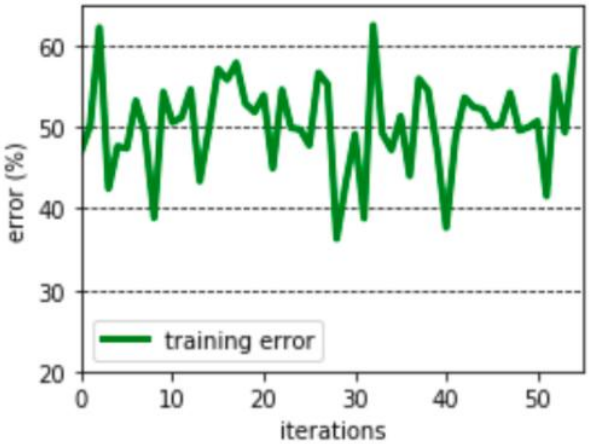
```
1  with autograd.detect_anomaly():
2      x = torch.tensor([[1.0, 2.0],[3.0, float('nan')]], requires_grad=True)
3      y = torch.tensor([[1.0, 2.0],[3.0, 4.0]], requires_grad=True)
4      z = x * y
5      out = torch.sum(z)
6      out.backward()
```

```
      4     z = x * y
      5     out = torch.sum(z)
----> 6     out.backward()

                                 1 frames
/usr/local/lib/python3.10/dist-packages/torch/autograd/__init__.py in backward(tensors, grad_tensors, retai
    264     # some Python versions print out the first line of a multi-line function
    265     # calls in the traceback and some print out the last line
--> 266     Variable._execution_engine.run_backward(  # Calls into the C++ engine to run the backward pass
    267         tensors,
    268         grad_tensors_,

RuntimeError: Function 'MulBackward0' returned nan values in its 1th output.
```

Dataset issues

- examples and labels are read from individual files
- what is the issue?

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

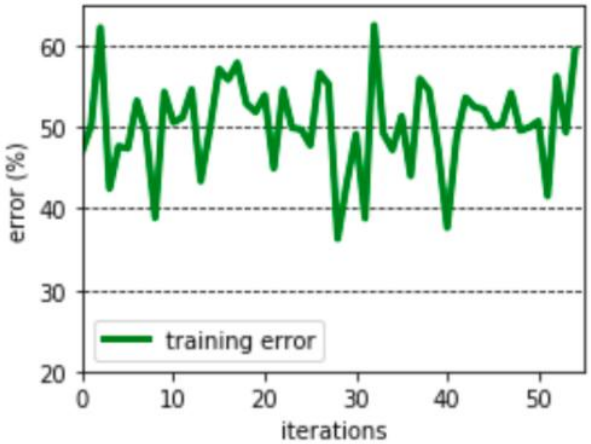


source: https://fullstackdeeplearning.com/spring2021/lecture-7/

Dataset issues

- examples and labels are read from individual files
- what is the issue?
  - files are read in random order, features[i] and labels[i] are not related

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



source: https://fullstackdeeplearning.com/spring2021/lecture-7/

Dataset issues

what is the issue?

```
# instantiate dataloader
train_dataloader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE
)
```

Dataset issues

what is the issue?

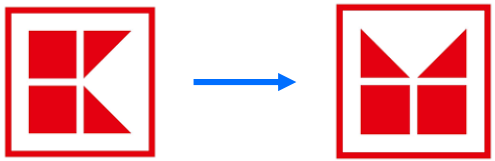
- training examples are not shuffled

```
# instantiate dataloader
train_dataloader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE
)
```

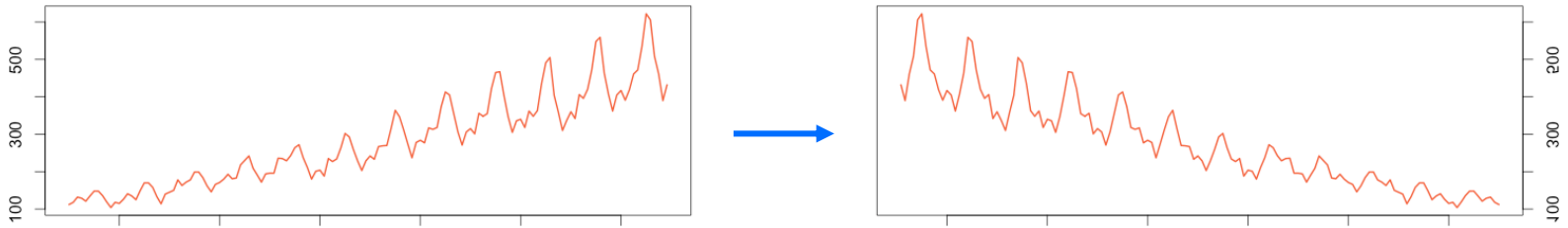
```
# instantiate dataloader
train_dataloader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True,
)
```

Dataset issues

- dataset augmentation commonly improves performance
- however, some augmentations are not meaningful and actually hurt model's performance:
  - flipping/rotating logos may not improve logo recognition



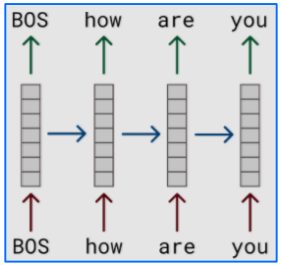
  - flipping time series changes trends



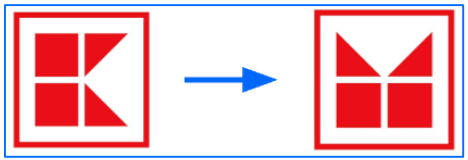
Functional Issues - recap

1.fix **logical errors** (*bad normalization, softmax over the wrong dimension, wrong labels fed to the model* etc.)

Age	Salary	Volume
28	26,000	4
22	32,000	10
25	29,000	86
24	38,000	2
26	27,000	71
22	42,000	88



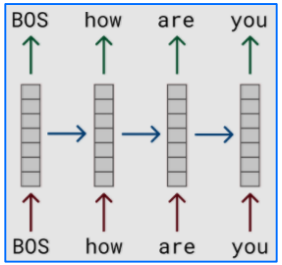
2. fix **dataset issues** (*data not shuffled, mislabeled data, improperly augmented* etc.)



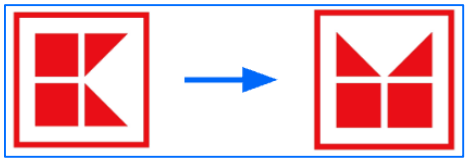
Functional Issues - recap

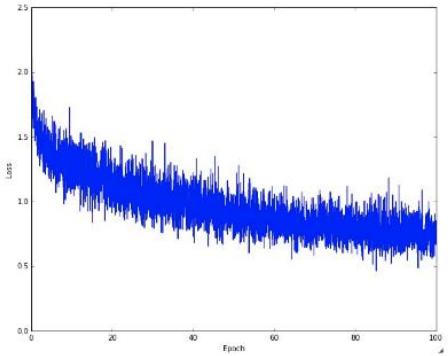
1.fix **logical errors** (*bad normalization, softmax over the wrong dimension, wrong labels fed to the model* etc.)

Age	Salary	Volume
28	26,000	4
22	32,000	10
25	29,000	86
24	38,000	2
26	27,000	71
22	42,000	88



2. fix **dataset issues** (*data not shuffled, mislabeled data, improperly augmented* etc.)





3. we can now start looking at **hyperparameter misconfigurations**

Debugging plan

1. run code and fix **syntactic bugs** (*shape/device/type mismatches*, *OOM issues*)

Debugging plan

1. run code and fix **syntactic bugs** (*shape/device/type mismatches*, *OOM issues*)
2. overfit model on a few (1-2) examples
    - if training loss ~0, start training on full dataset

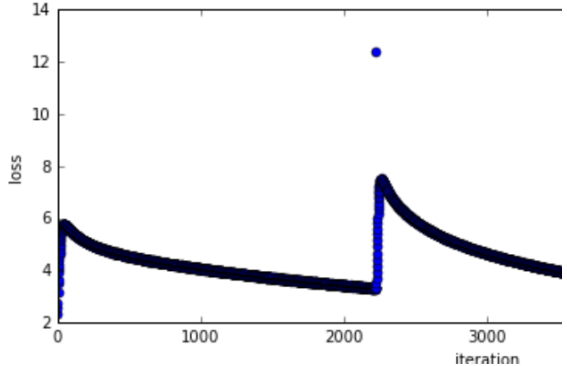
Debugging plan

1. run code and fix **syntactic bugs** (*shape/device/type mismatches*, *OOM issues*)
2. overfit model on a few (1-2) examples
    - if training loss ~0, start training on full dataset
    - if not, check for **logical and dataset errors**:
        - manually check input:
            - lots of zeros or NaN (*preprocessing error, corrupted data*)
            - duplicated rows (*dataset contains duplicates*, *augmentation bug*)
            - all batch examples have the same label (*no shuffling*)
        - check the arguments:
            - softmax taken over the correct **dimensions**
            - loss function receives the correct output (*unnormalized or normalized scores*)

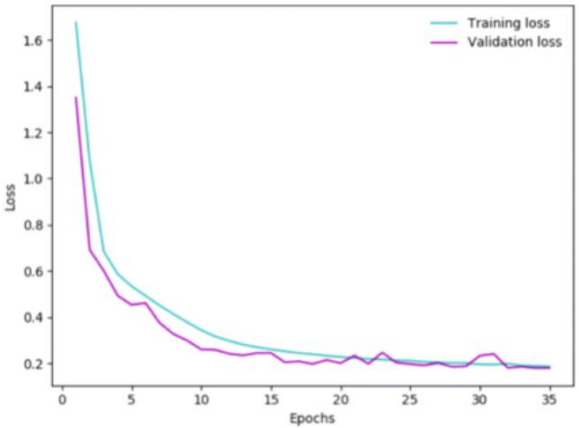
Debugging plan

1. run code and fix **syntactic bugs** (*shape/device/type mismatches*, *OOM issues*)
2. overfit model on a few (1-2) examples
    - if training loss ~0, start training on full dataset
    - if not, check for **logical and dataset errors**:
        - manually check input:
            - lots of zeros or NaN (*preprocessing error, corrupted data*)
            - duplicated rows (*dataset contains duplicates*, *augmentation bug*)
            - all batch examples have the same label (*no shuffling*)
        - check the arguments:
            - softmax taken over the correct **dimensions**
            - loss function receives the correct output (*unnormalized or normalized scores*)
3. start training on the full dataset
    - inspect **loss/metric learning curves** and look for **hyperparameters misconfigurations**
    - we may still have invisible bugs at this point! (*train data leaks into validation*, *bug when measuring validation error* etc.)

Loss curves

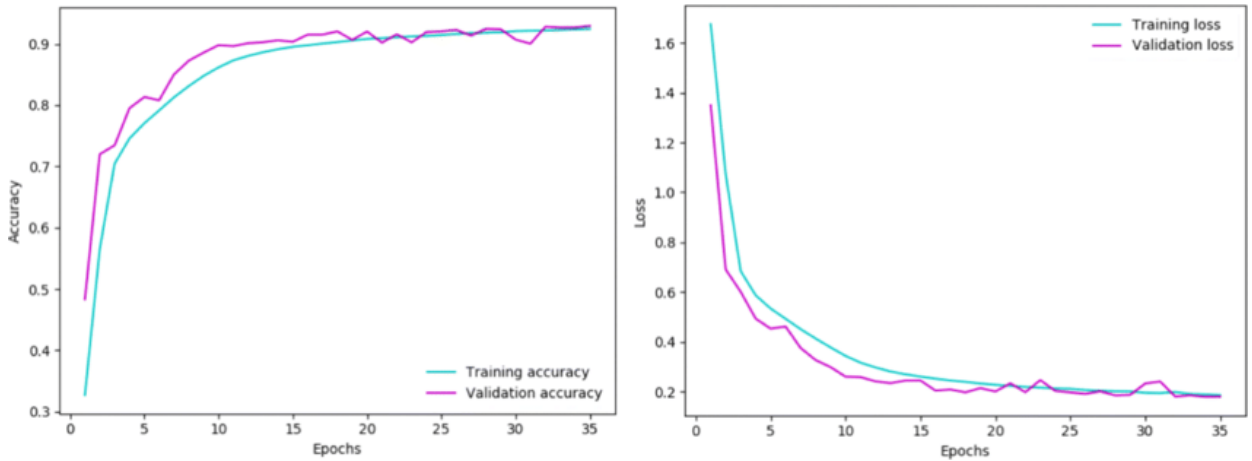


ons.tumblr.com



Learning curves

- plot values for train/validation set over epochs/optimization steps:
  - **loss curves**: captures the optimization progress
  - **metric curves**: reflects our business/research goal (accuracy/F1/AUROC etc.)
- ideally, loss inversely correlated with metric



source

Why look at learning curves?

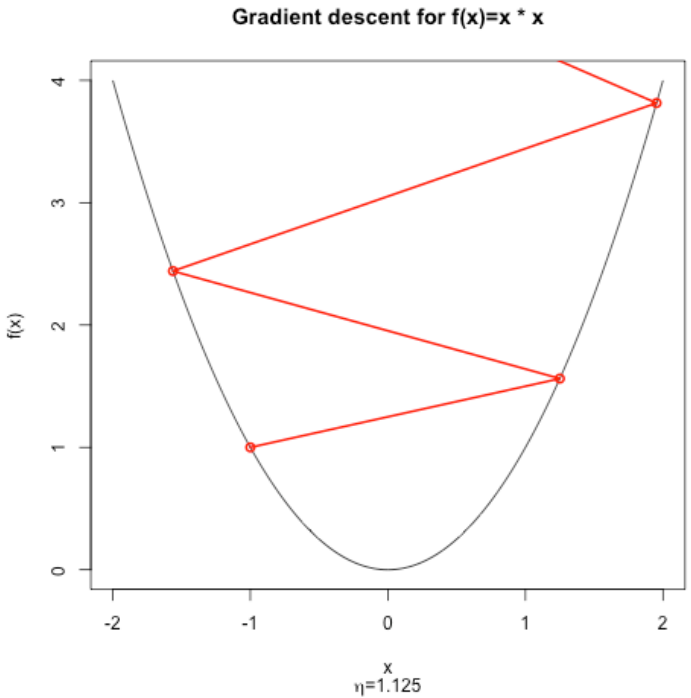
- detect if the model **learns the training data and/or generalizes to new data**
  - train and validation losses decrease at similar rate or show very small gap
  - if not, check for **hyperparameter misconfiguration**
- pick the best model based on the validation loss
- check if loss improvements **translate** to better metrics

Does the model learn the train data?

- training loss increases
- training loss decreases slowly
- training loss decreases then stagnates
- training loss decreases

Training loss increases

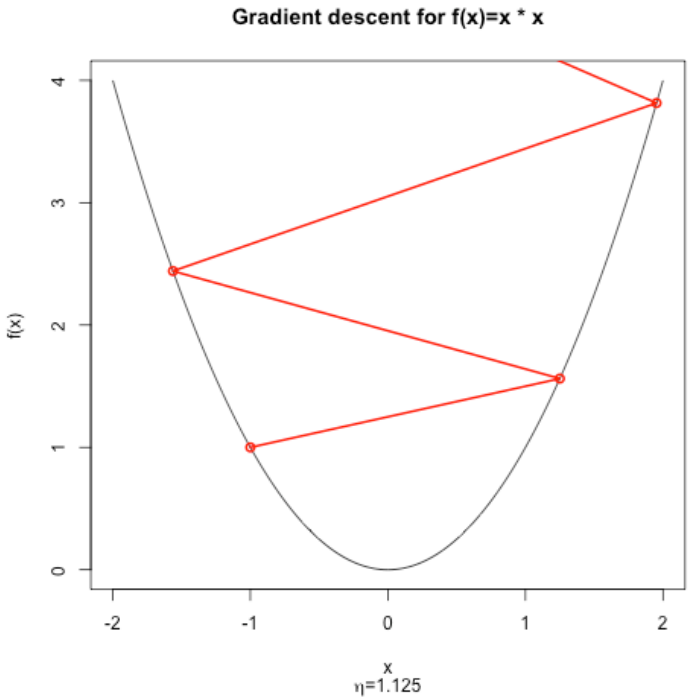
- Q: what does it usually indicate?



https://stats.stackexchange.com/questions/364360/how-can-change-in-cost-function-be-positive

Training loss increases

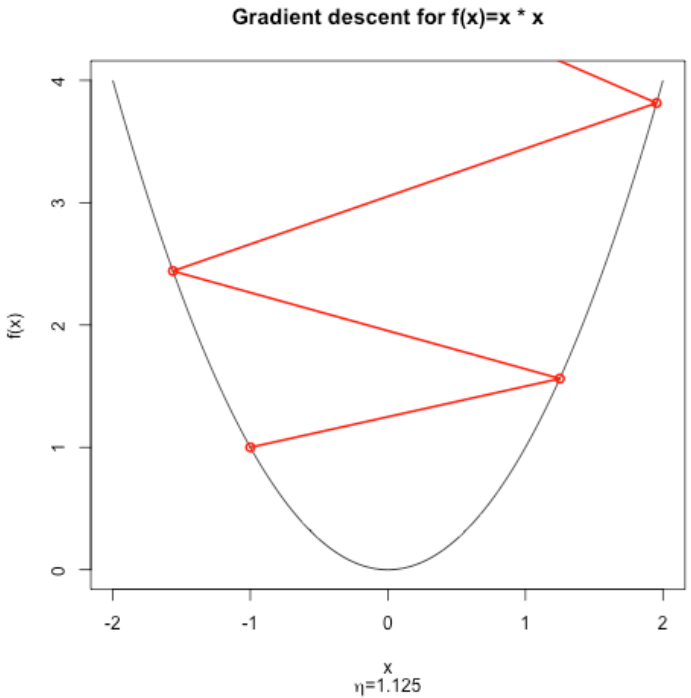
- Q: what does it usually indicate?
  A: the learning rate may be *too high*, moving the parameters away from the local minimum



https://stats.stackexchange.com/questions/364360/how-can-change-in-cost-function-be-positive

Training loss increases

- Q: what does it usually indicate?
  A: the learning rate may be *too high*, moving the parameters away from the local minimum
- **solution**: decrease learning rate

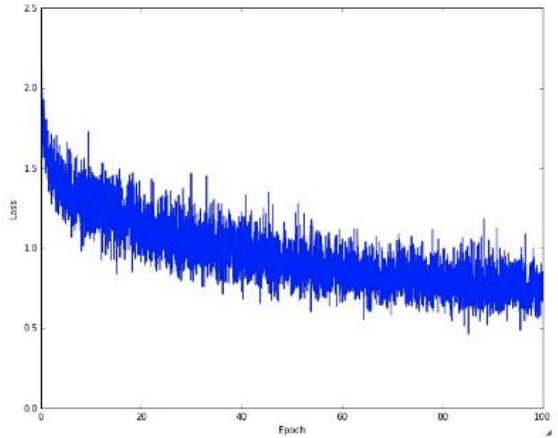


https://stats.stackexchange.com/questions/364360/how-can-change-in-cost-function-be-positive

Training loss decreases slowly

- Q: what could it mean?

training loss



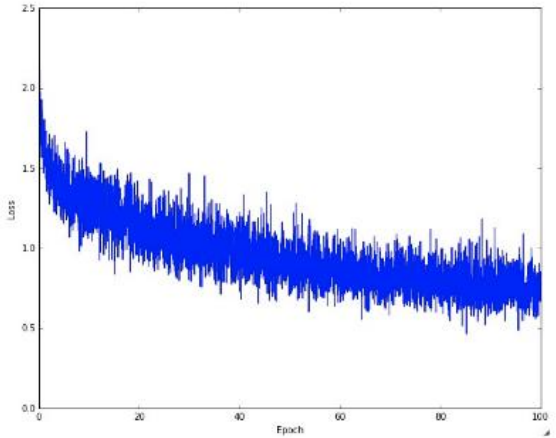
epochs

source: https://cs231n.github.io/neural-networks-3/

Training loss decreases slowly

- Q: what could it mean?
- A: it could indicate a learning rate that is *too low* (loss decreases linearly)

training loss

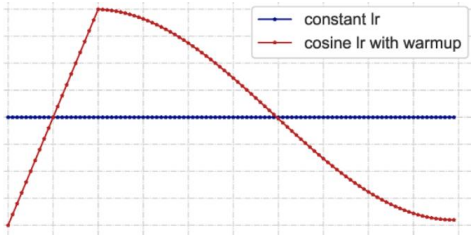


epochs

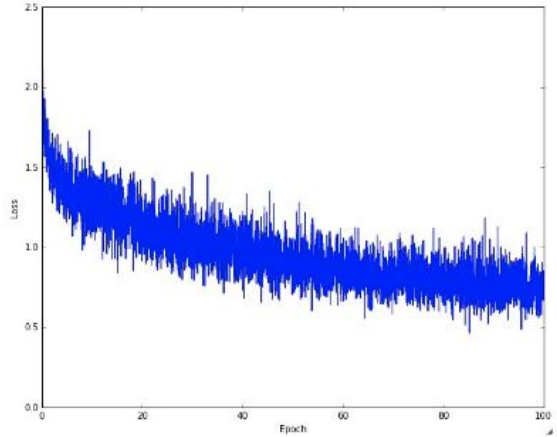
source: https://cs231n.github.io/neural-networks-3/

Training loss decreases slowly

- Q: what could it mean?
- A: it could indicate a learning rate that is *too low* (loss decreases linearly)
- solution:
  - higher learning rate
  - use learning rate scheduler with warmup (LR increases during warmup then decreases)
    - cosineLRWithWarmup: increase LR linearly then decrease with cosine



training loss

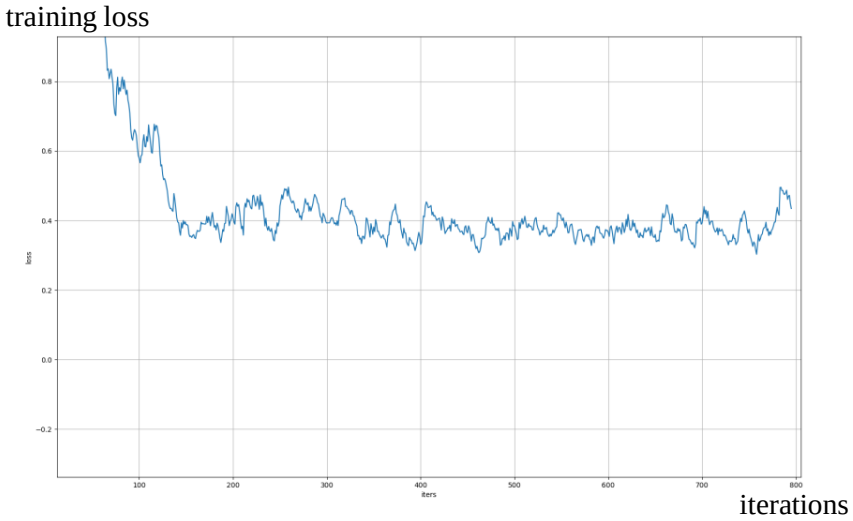


epochs

source: https://cs231n.github.io/neural-networks-3/

Training loss decreases then stagnates

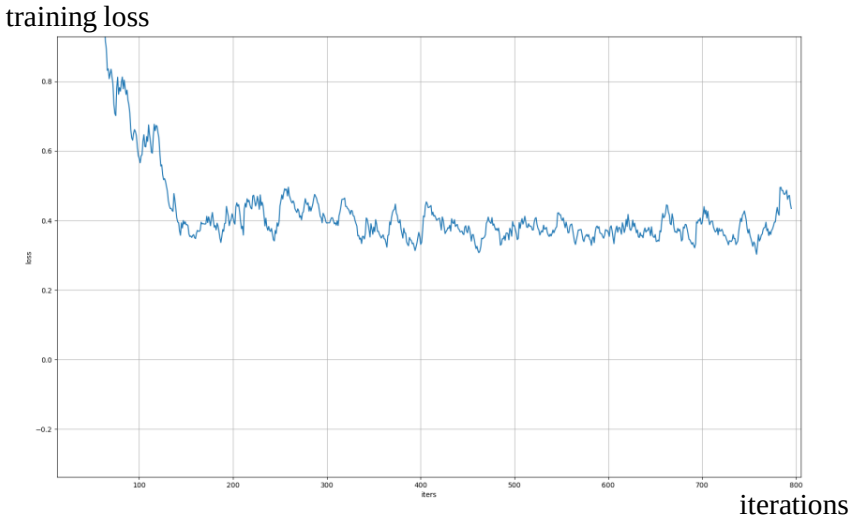
- Q: what could it mean?



source: https://jumabek.wordpress.com/2017/03/21/781/

Training loss decreases then stagnates

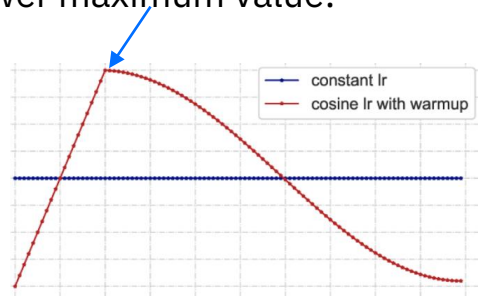
- Q: what could it mean?
- A: it may still indicate a learning rate that is too high (particularly if it oscillates a lot)

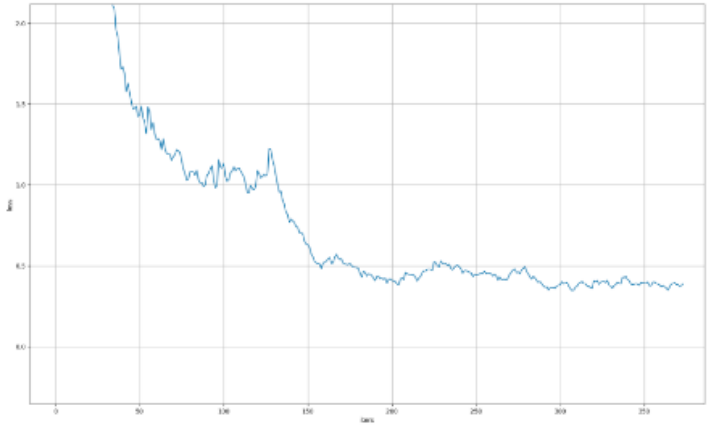


source: https://jumabek.wordpress.com/2017/03/21/781/

Training loss decreases then stagnates

- Q: what could it mean?
- A: it may still indicate a learning rate that is too *high* (particularly if it oscillates a lot)
- solution:
  - start with a smaller learning rate (½-⅓ of original)
  - if using LR scheduler with warmup, try to lower maximum value:

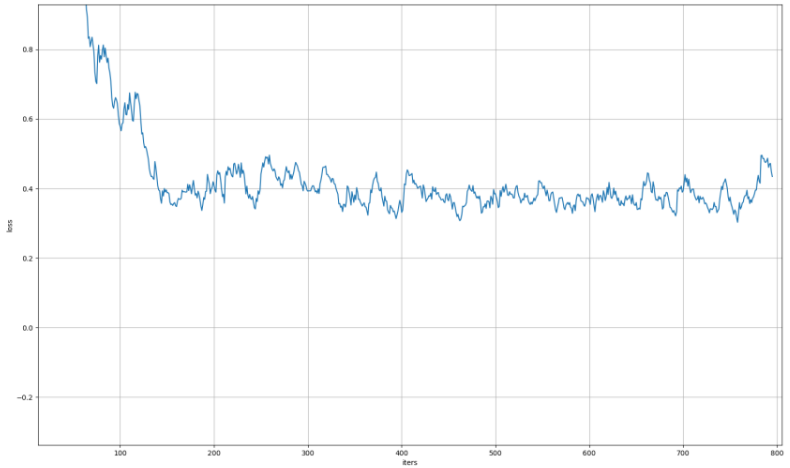




source: https://jumabek.wordpress.com/2017/03/21/781/

Training loss decreases then stagnates

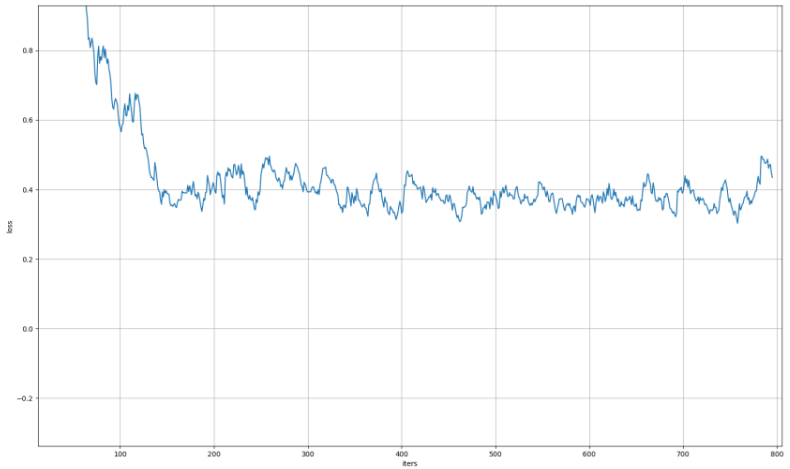
- what if higher LR still doesn't solve the issue?



source: https://jumabek.wordpress.com/2017/03/21/781/

Training loss decreases then stagnates

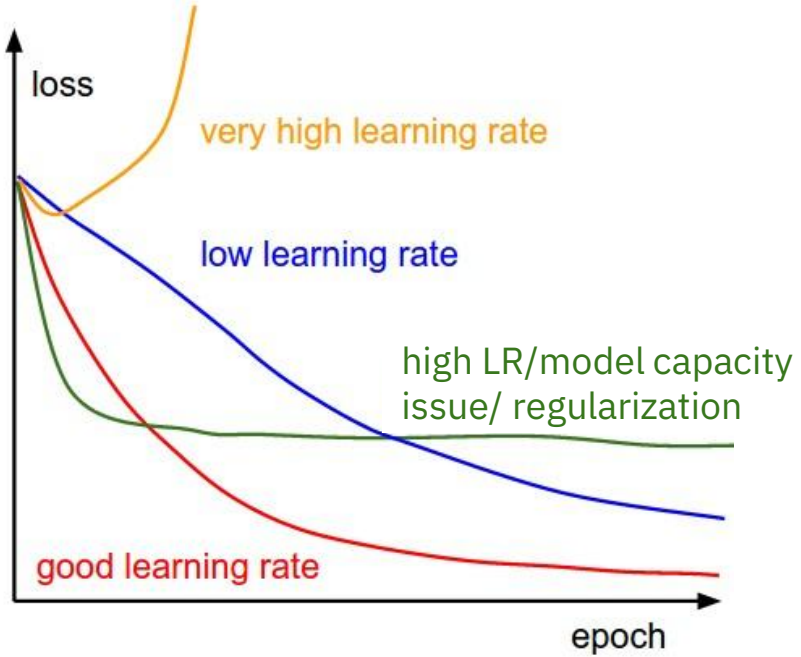
- what if higher LR still doesn't lower the train loss?
- possible issues:
  - the model is too small => *increase model capacity* (more layers, more neurons per layer)
  - it is heavily regularized => *reduce regularization* (smaller Dropout rate, smaller L2 coefficient)
  - lots of frozen parameters => unfreeze some of them



source: https://jumabek.wordpress.com/2017/03/21/781/

training loss curves - recap

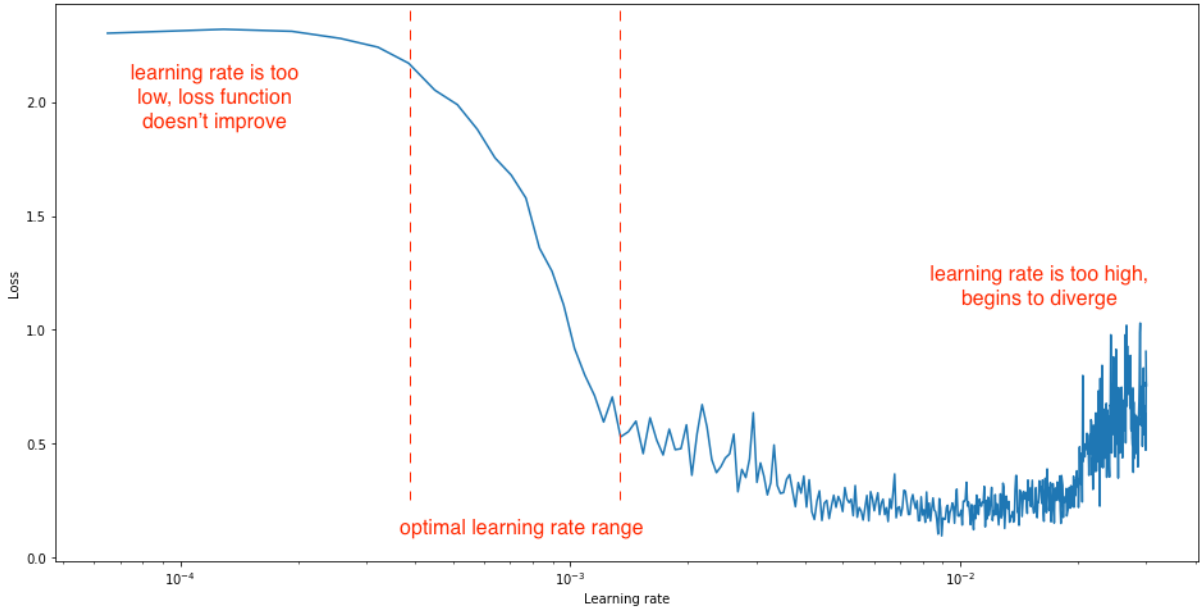
- loss increases early in the training (*high learning rate*)
- loss decreases slowly (potentially *low learning rate*)
- loss decreases quickly then reaches a plateau (*high learning rate or model capacity issues*)



source: https://cs231n.github.io/neural-networks-3/

How to pick a good learning rate range?

- let model train and decrease learning rate for each new batch
- a good learning rate range:
  - starts when loss begins to drop steeply
  - ends when loss slows down, wiggles or starts increasing



source: https://www.jeremyjordan.me/nn-learning-rate/

Learning curves scenarios

- training loss increases
- training loss decreases slowly
- training loss decreases then stagnates
- training loss decreases

we have looked at possible issues with training curves!

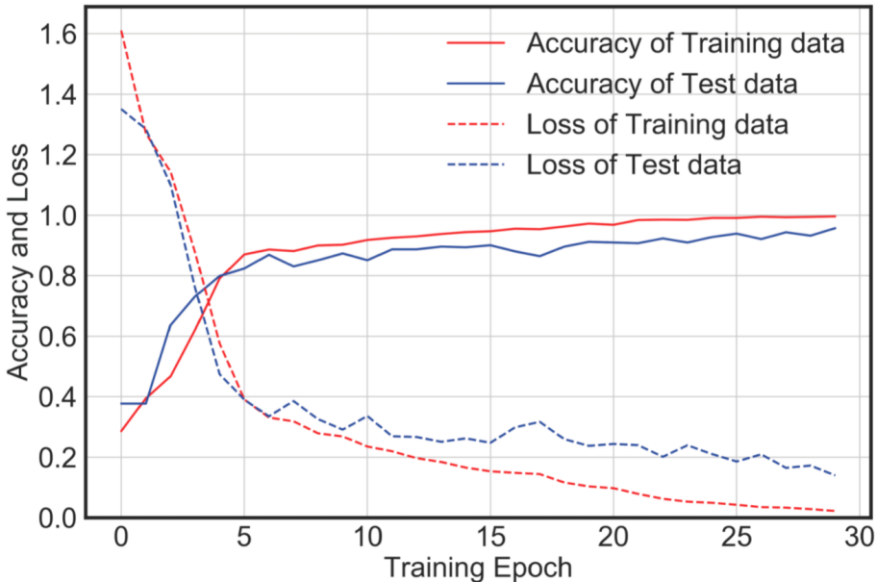
Learning curves scenarios

- training loss increases
- training loss decreases slowly
- training loss decreases then stagnates
- training loss decreases, but:
  - validation loss ≈ training loss
  - validation loss >> training loss
  - validation loss ≈ training loss, then diverges
  - validation loss < training loss
  - validation loss << training loss
  - both validation loss and validation accuracy increase

however, we also want to see the generalization capability of the model
=> inspect validation curves!

Validation loss ≈ training loss

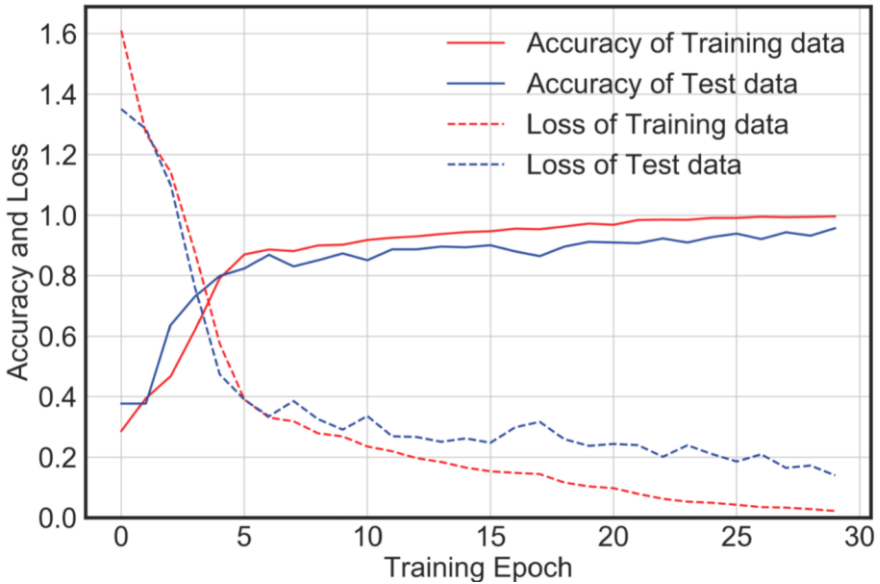
- Q: overfitting or underfitting?



source: https://www.mdpi.com/1424-8220/20/9/2710/htm

Validation loss ≈ training loss

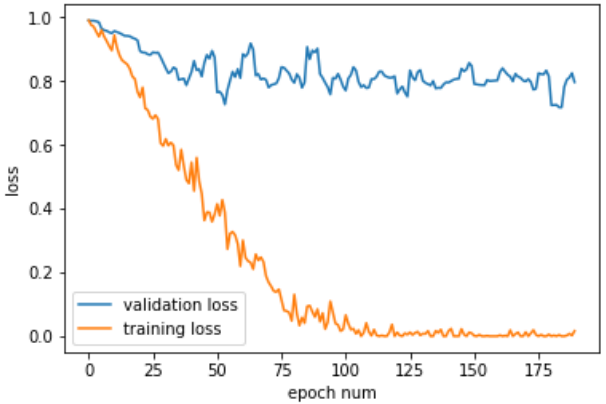
- Q: overfitting or underfitting?
- A: good fit
  - training and validation loss approach 0, with small gap between them
  - validation loss usually (but now always) slightly higher than training loss
  - training and validation accuracy also increase simultaneously (they are not always correlated with loss curves)



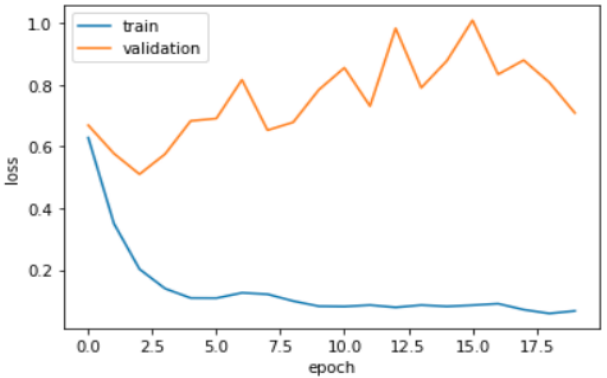
source: https://www.mdpi.com/1424-8220/20/9/2710/htm

Validation loss >> training loss

- Q: overfitting or underfitting?



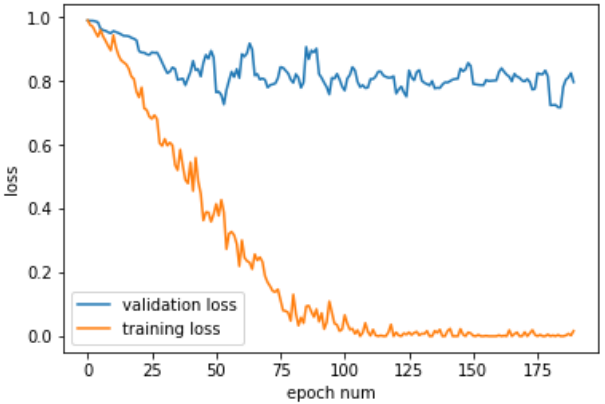
source



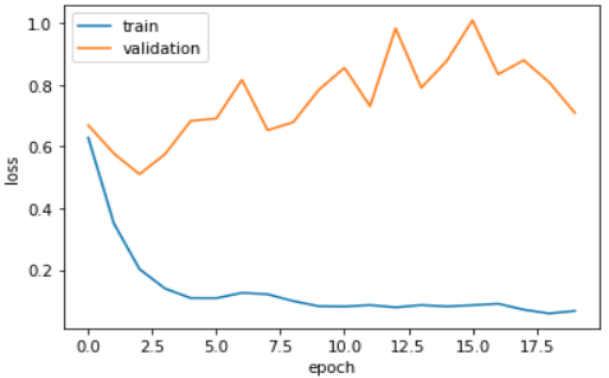
source

Validation loss >> training loss

- Q: overfitting or underfitting?
- A: model is overfitting
  - training loss decreases, while validation loss either gets worse or doesn't improve
  - model learns patterns that are not useful outside training set
- solutions:
  - add regularization (Dropout/L2)
  - add more training data
  - reduce model size (fewer layers/neurons per layer)



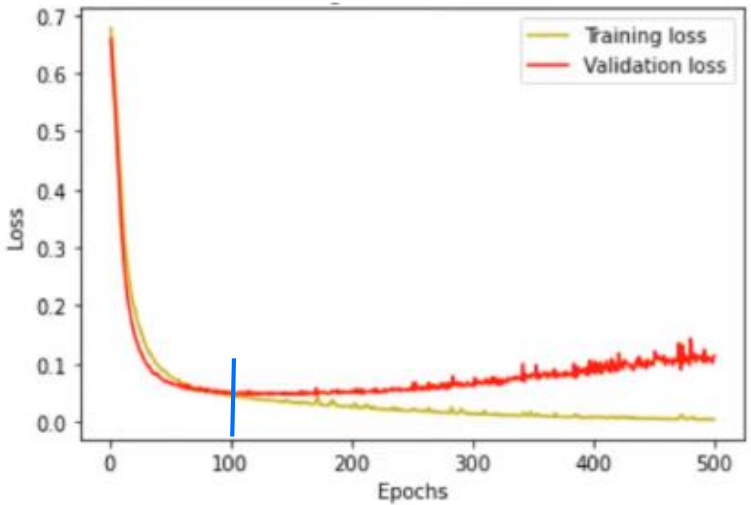
source



source

Validation loss tracks train loss then diverges

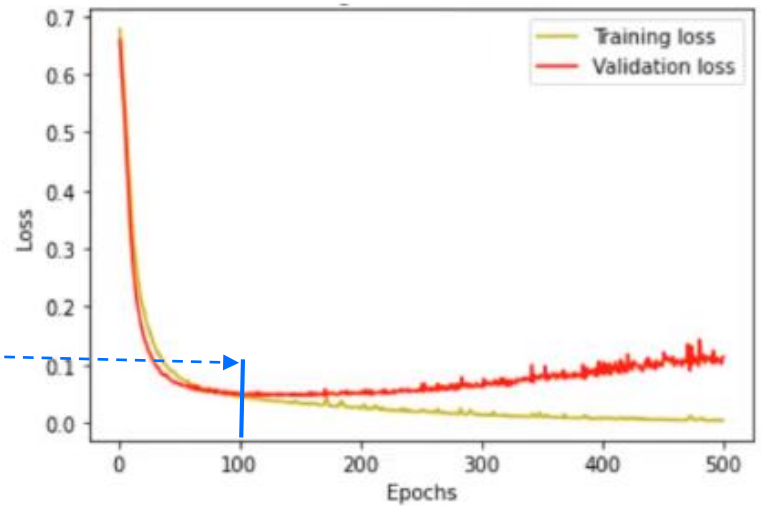
- Q: overfitting or underfitting?



source: https://www.youtube.com/watch?v=p3CcfIjycBA

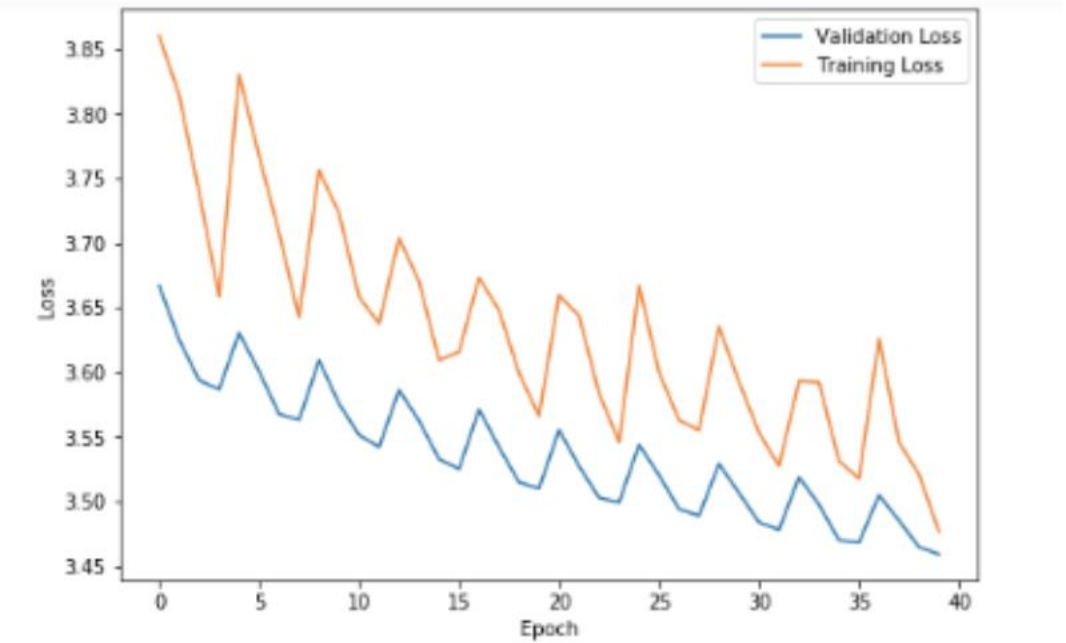
Validation loss tracks train loss then diverges

- Q: overfitting or underfitting?
- A: overfitting from epoch 100 onwards
  - validation loss tracks training loss, but diverges from epoch 100
- solutions:
  - early stop at epoch 100
  - add more regularization to improve validation loss



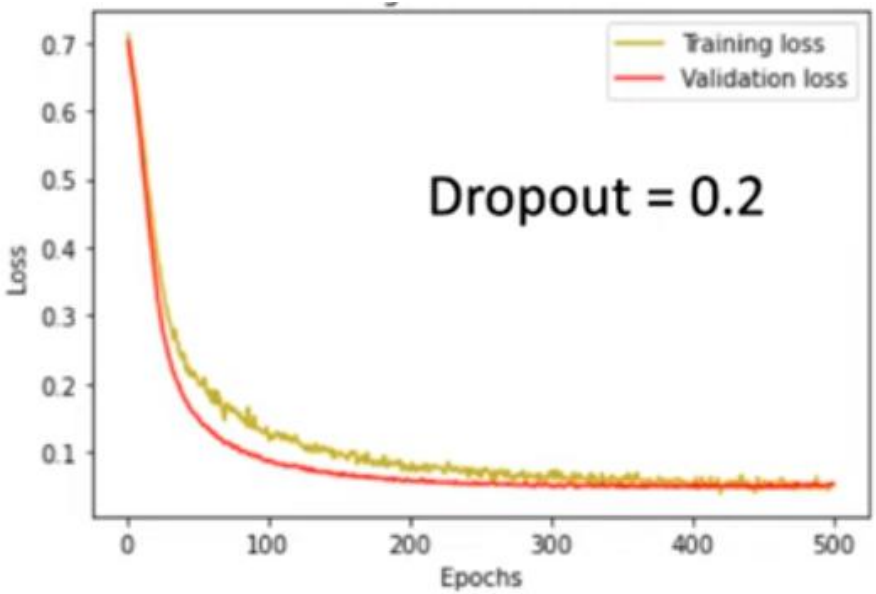
source: https://www.youtube.com/watch?v=p3CcfIjycBA

Unusual train-validation loss curves



Validation loss < training loss

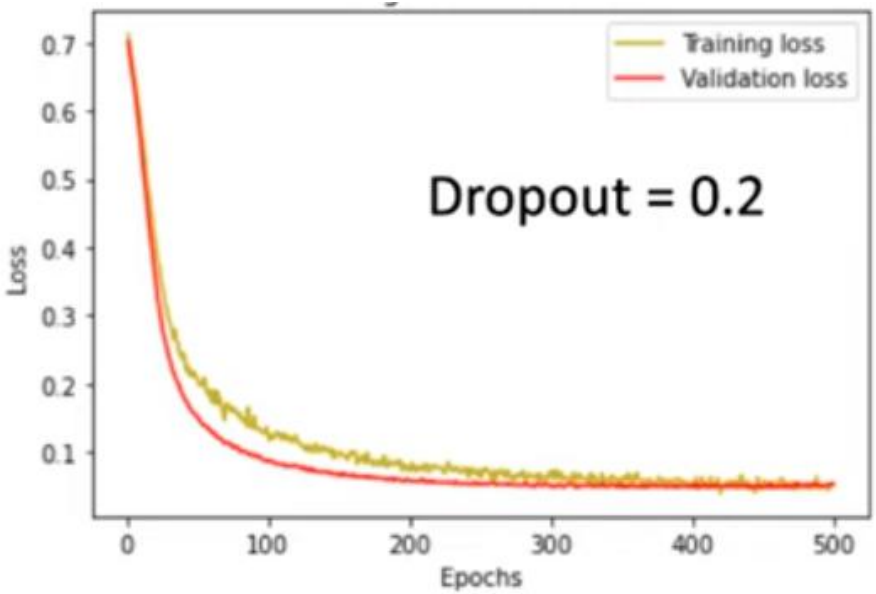
- Q: validation loss should normally be slightly higher than training loss, why is it lower here?



source: https://www.youtube.com/watch?v=p3CcfIjycBA

Validation loss < training loss

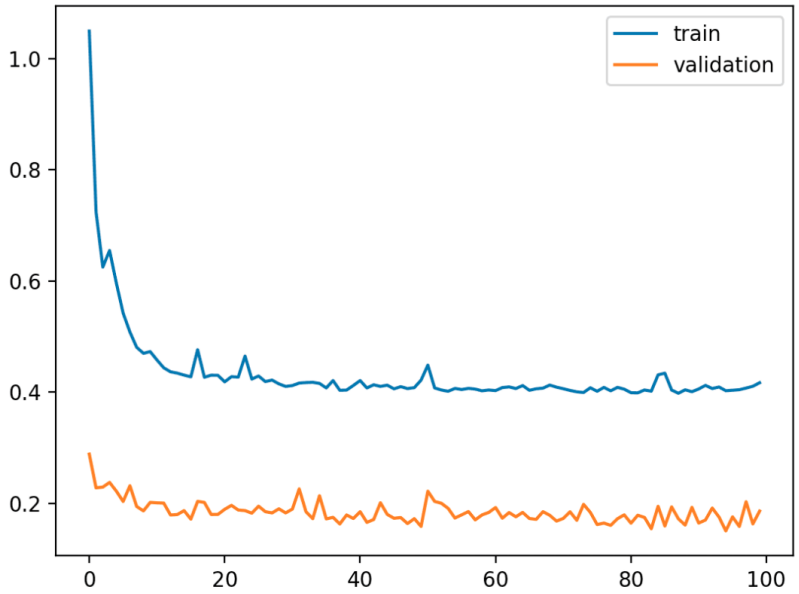
- Q: validation loss should normally be slightly higher than training loss, why is it lower here?
- A: the model is still a good fit if validation loss slightly lower than training loss
  - this effect may be due to Dropout, which disables some of the neurons during training
  - this lowers training performance at the expense of increased generalization during validation (when dropout is disabled)



source: https://www.youtube.com/watch?v=p3CcfIjycBA

Validation loss << training loss

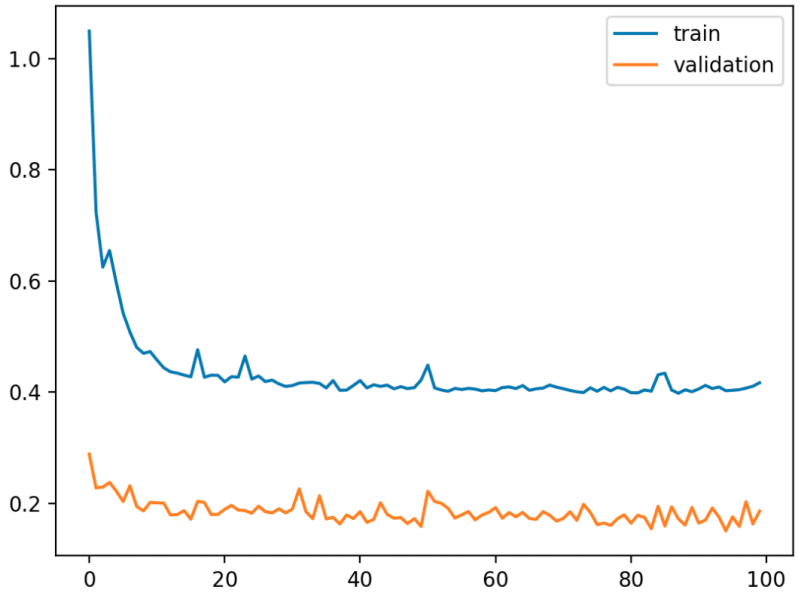
- Q: overfitting or underfitting?



source: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Validation loss << training loss

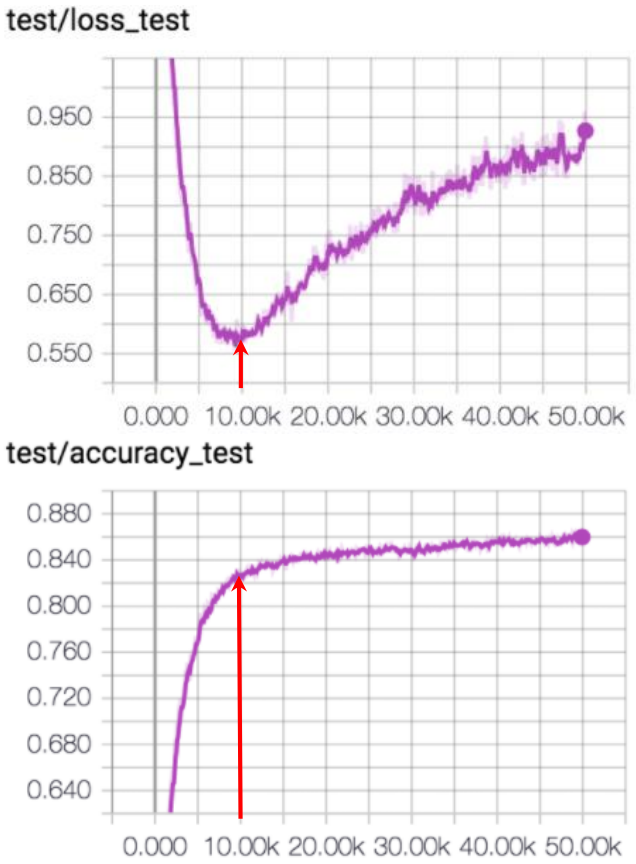
- Q: overfitting or underfitting?
- A: neither
  - validation loss is much lower than training loss, which means the validation set is ‘easier’ for the model than the training set
  - this may point to **dataset issues**:
    - validation examples easier than training examples (due to some bug in the splitting process or augmenting training examples only)



source: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Both validation loss & accuracy increase

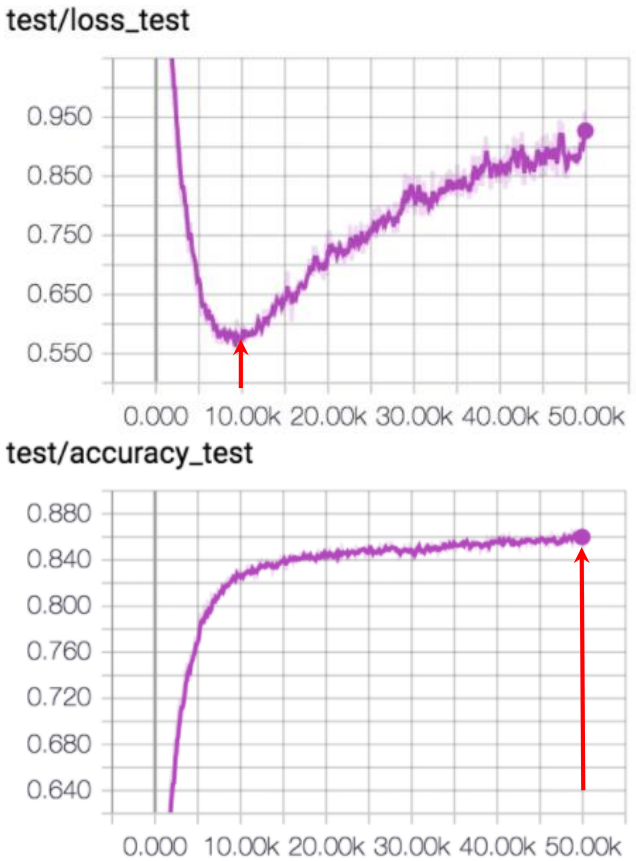
- Q: lowest validation loss is at ~10k steps, but validation accuracy continues to increase after that, where does overfitting begin?



source

Both validation loss & accuracy increase

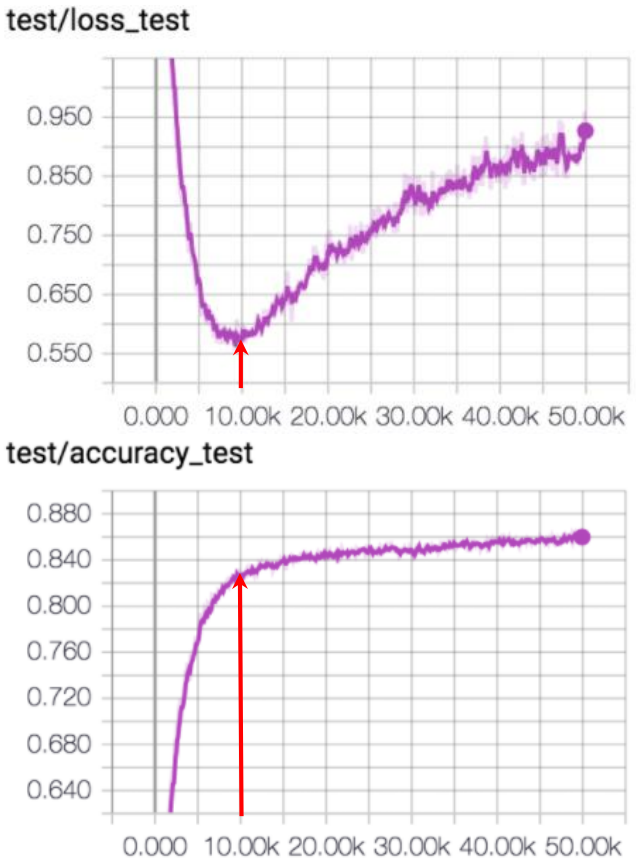
- Q: lowest validation loss is at ~10k steps, but validation accuracy continues to increase after that, where does overfitting begin?
- A: overfitting begins at 10k steps
  - validation loss degrades rapidly, even though validation accuracy slightly improves



source

Both validation loss & accuracy increase - why?

- loss is sensitive to probabilities:
  - it may become overconfident in wrong predictions, leading to higher loss
- thresholded metrics (like accuracy) ignore confidence
  - they usually decide right or wrong based on a fixed threshold (0.5)
  - accuracy may remain the same (or slightly improve) while probabilities deteriorate



source

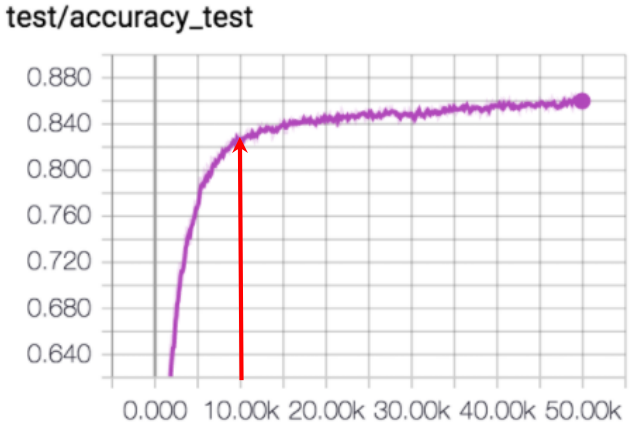
Both validation loss & accuracy increase - why?

- becomes more confident in incorrect predictions (p=0.6 -> p=0.9)

	p(y=0)	**p(y=1)**	loss (-logp)
old predictions	0.6	0.4	0.92
new predictions	0.9	0.1	2.30

- wrong predictions close to threshold 0.5 become correct p=0.48 => p=0.51

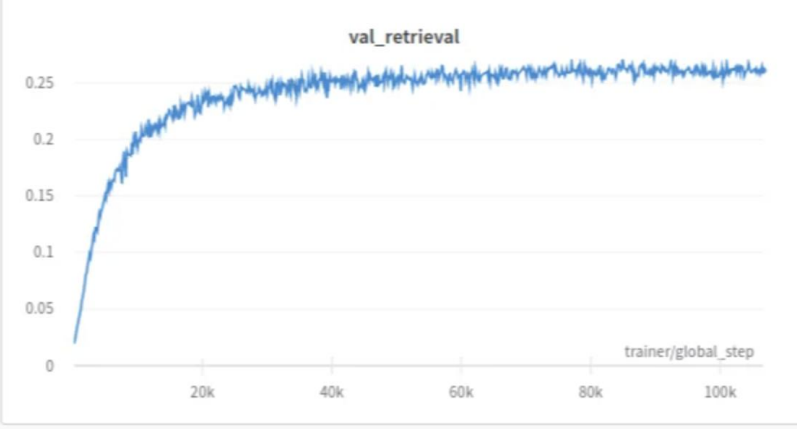
	p(y=0)	**p(y=1)**	loss (-logp)
old predictions	0.52	0.48	0.73
new predictions	0.49	0.51	0.67

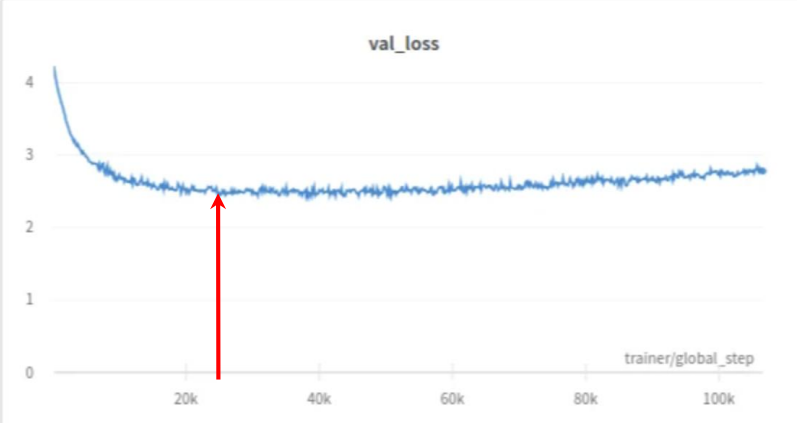


source

Both validation loss & accuracy increase

- model begins to overfit around step 25K, which model to choose?

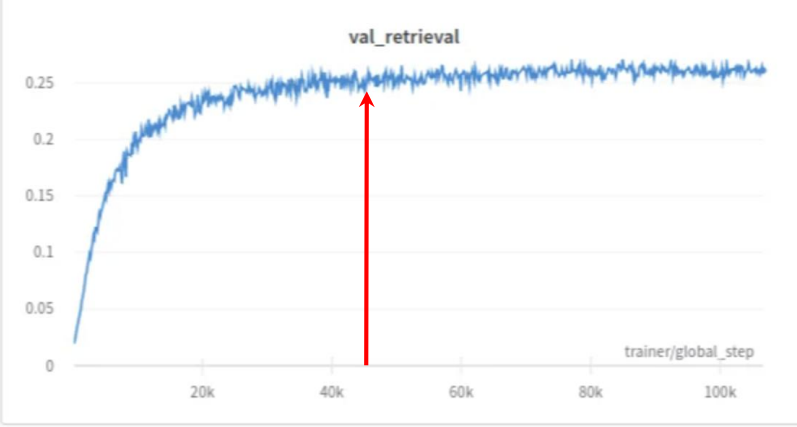


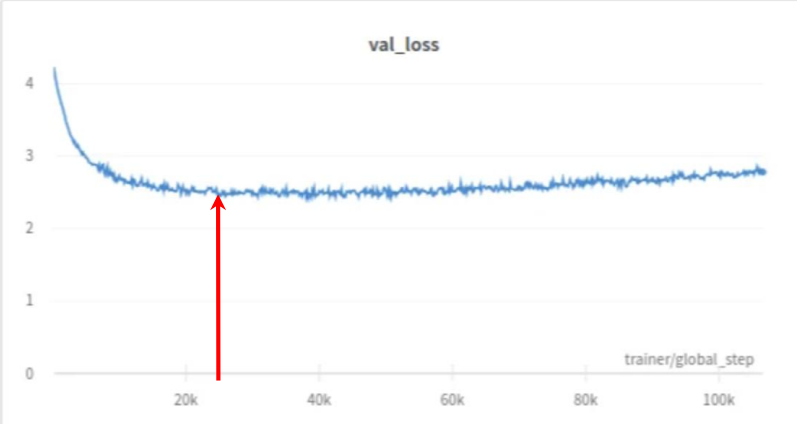


source

Both validation loss & accuracy increase

- model begins to overfit around step 25K, which model to choose?
- we can pick a model with a slightly higher validation metric, as long as validation loss doesn't degrade significantly





source

Learning curves TL;DR

- validate that the model is training correctly and generalizes:
  - **good fit:** train/validation loss decrease and stabilize at similar value
  - **beginning of overfit**: gap between validation and train loss widens

Learning curves TL;DR

- validate that the model is training correctly and generalizes:
  - **good fit:** train/validation loss decrease and stabilize at similar value
  - **beginning of overfit**: gap between validation and train loss widens
- choose the best model:
  - **primary criterion:** choose epoch/step with ***lowest loss on the validation set***

Learning curves TL;DR

- validate that the model is training correctly and generalizes:
  - **good fit:** train/validation loss decrease and stabilize at similar value
  - **beginning of overfit**: gap between validation and train loss widens
- choose the best model:
  - **primary criterion:** choose epoch/step with ***lowest loss on the validation set***
  - **secondary check:** double check with validation metrics:
    - if *validation loss* ***is at a minimum****, or* ***it slightly increases*** and validation metric still increases, pick model where validation metric (acc./F1/etc.) maximum
    - if *validation loss* ***rises significantly*** and validation metric still increases, pick model where validation loss was minimum

Debugging plan - recap

1. run code and fix **syntactic bugs** (*shape/device/type mismatches*, *OOM issues*)

Debugging plan - recap

1. run code and fix **syntactic bugs** (*shape/device/type mismatches*, *OOM issues*)
2. overfit model on a few (1-2) examples
    - if train loss -> 0, start training on the full data
    - otherwise look for **logical and dataset issues**

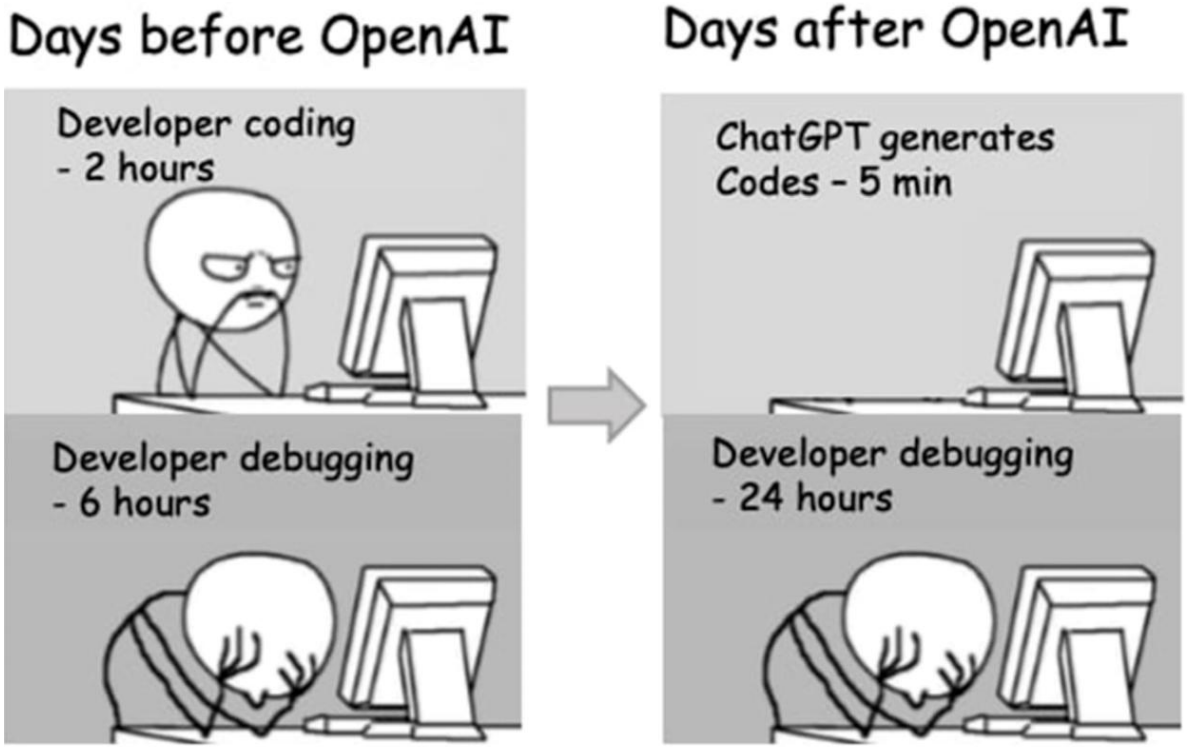
Debugging plan - recap

1. run code and fix **syntactic bugs** (*shape/device/type mismatches*, *OOM issues*)
2. overfit model on a few (1-2) examples
    - if train loss -> 0, start training on the full data
    - otherwise look for **logical and dataset issues**

1. start training on the full dataset
    - inspect loss/metric learning curves
    - if loss curves don't show a good fit:
        - look for **hyperparameter misconfigurations** (learning rate, LR scheduler etc.)
        - reinvestigate **dataset issues** if model still can't generalize

Debugging conclusion and QA

- ML projects projects have many sources of error:
  - start small and tackle one thing at a time 🔍
- embrace the debugging experience 🤔:
  - debugging is still a relevant skill even in the LLM era



https://programmerhumor.io/debugging-memes/ai-revolution/