Deep Learning

4. Debugging

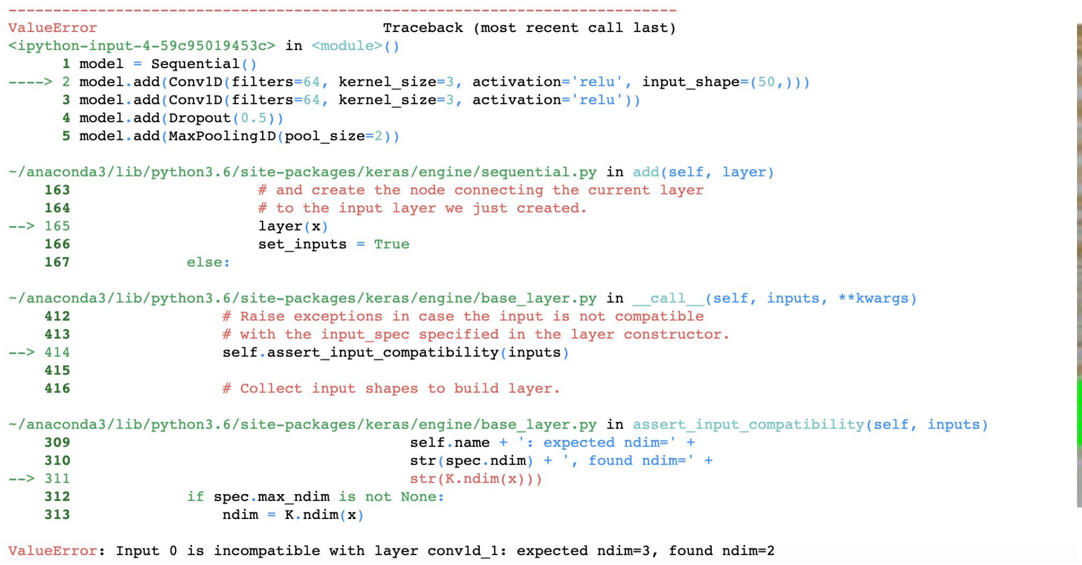
Test 1

- 5 minute
- 2 variante de test:
  - Varianta 1 (numar impar de litere in numele complet)
  - Varianta 2 (numar par de litere in numele complet)
- Exemple:
  - Ion Dan-Ion => 9 litere => Varianta 1
  - Ion Dan Ion => 9 litere => Varianta 1
  - Ana Ion => 6 litere => Varianta 2

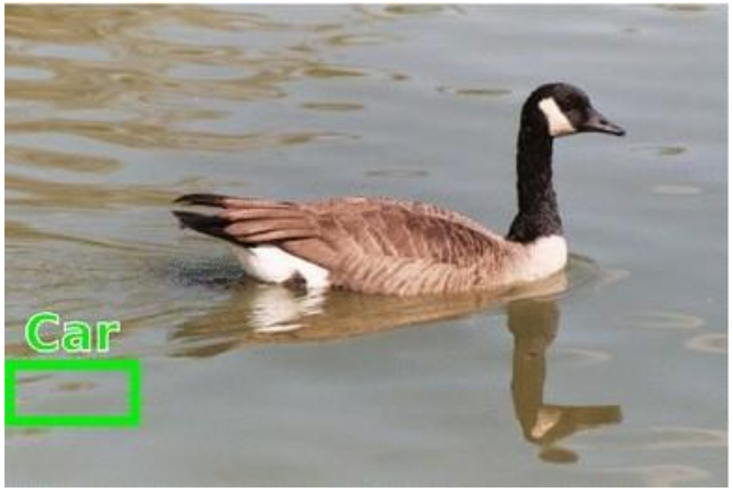
Machine Learning bugs

syntactic

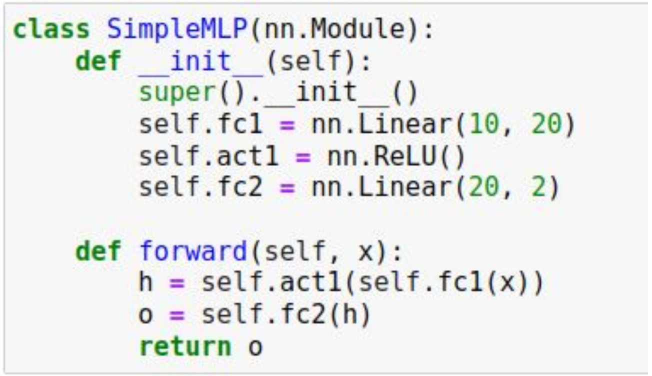
functional



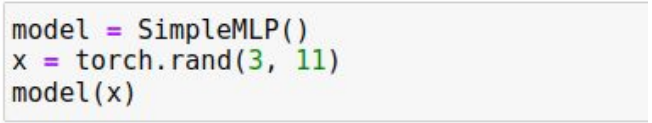
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-4-59c95019453c> in <module>()
      1 model = Sequential()
----> 2 model.add(Conv1D(filters=64, kernel_size=3, activation='relu',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



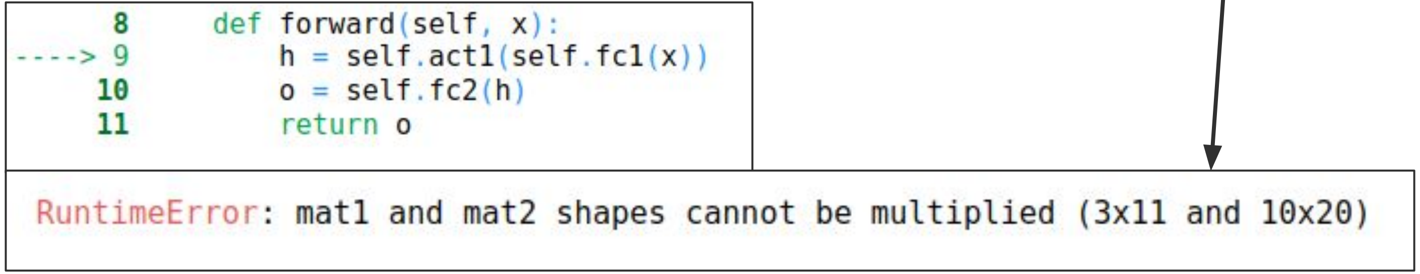
Wrong tensor shape



```
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
```



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

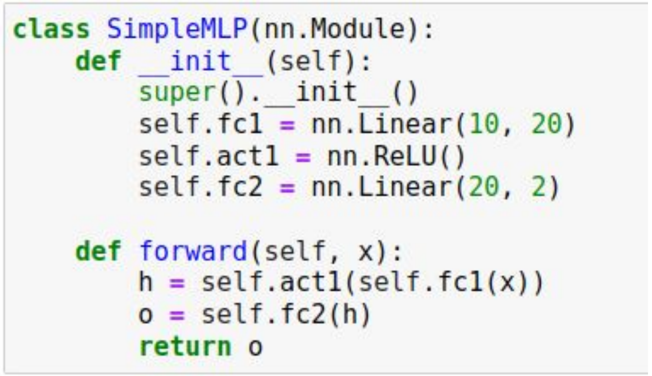


```
      8     def forward(self, x):
----> 9         h = self.act1(self.fc1(x))
     10         o = self.fc2(h)
     11         return o
```

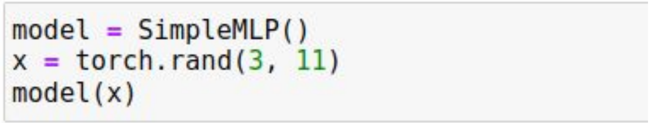
```
RuntimeError: mat1 and mat2 shapes cannot be multiplied (3x11 and 10x20)
```

- one of the most common errors is when the tensor shape is incompatible with the current operation
- can be easy to figure out when the *input data* has the wrong shape

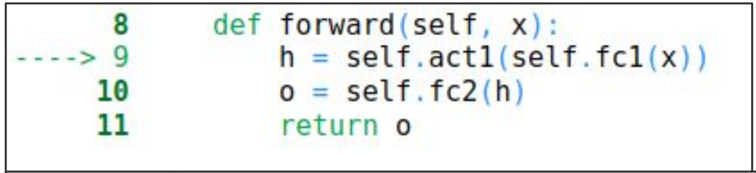
Wrong tensor shape



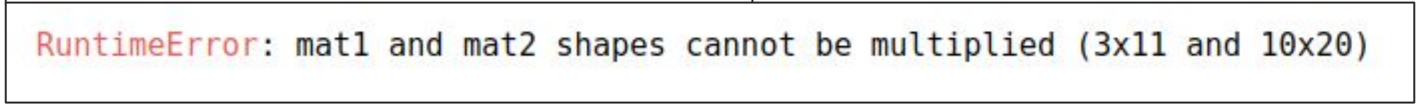
```python
class SimpleMLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(10, 20)
        self.act1 = nn.ReLU()
        self.fc2 = nn.Linear(20, 2)

    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
```

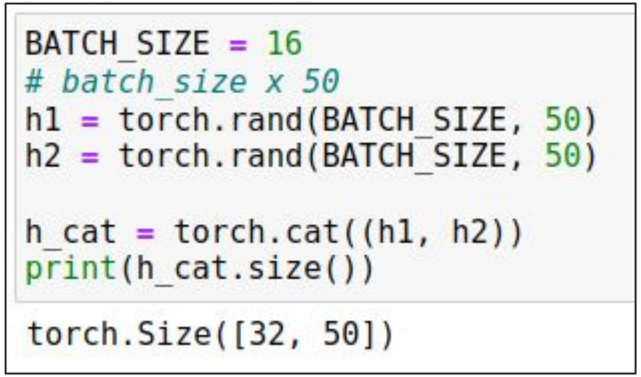


```
RuntimeError: mat1 and mat2 shapes cannot be multiplied (3x11 and 10x20)
```

- one of the most common errors is when the tensor shape is incompatible with the current operation
- can be easy to figure out when the *input data* has the wrong shape
- typical scenarios:
  - *images*: input Tensor has shape B x 3 x W x H instead of B x (3 x W x H)
  - *time series*: input Tensor has shape **HiddenSize** x **timesteps** instead of **timesteps** x **HiddenSize**

Wrong tensor shape (2)

- other shape mismatches may happen during feedforward
  - tensors are manipulated using different operating (slice, concat, max, reshape etc.)
  - the resulting tensor shape no longer matches with the next layer
- scenario: concatenating two tensors on the wrong dimension:



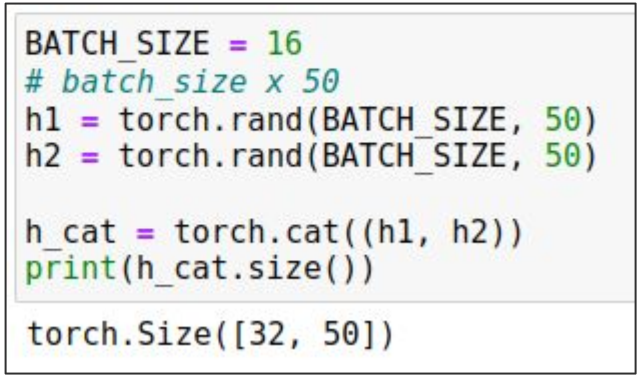
```
BATCH_SIZE = 16
# batch_size x 50
h1 = torch.rand(BATCH_SIZE, 50)
h2 = torch.rand(BATCH_SIZE, 50)

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

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

cat function concatenates on the 1st dimension by default!

Wrong tensor shape (2)

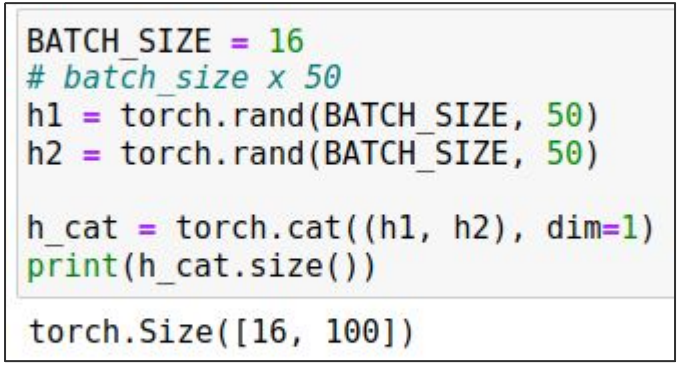
- other shape mismatches may happen during feedforward
  - tensors are manipulated using different operating (slice, concat, max, reshape etc.)
  - the resulting tensor shape no longer matches with the next layer
- scenario: concatenating two tensors on the wrong dimension:
  - good practice: specify 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 the 1st dimension 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])
```

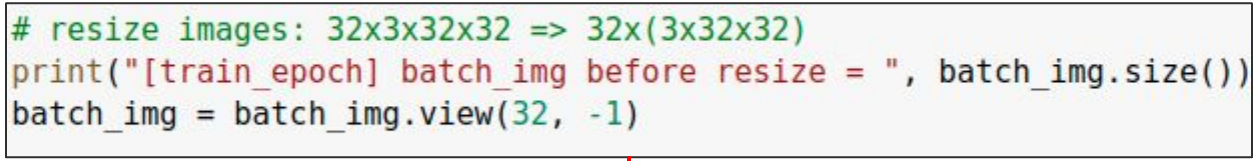
the two input tensors are correctly concatenated

Wrong tensor shape (3)

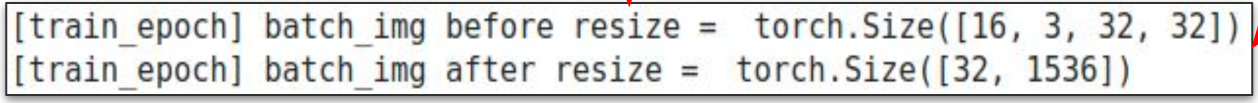
- scenario: last batch in a dataset may have 1st dimension != **batch_size:**
  - why? let’s say **batch_size=32**
  - ***DataLoader*** loads 32 elements until it exhausts all the dataset elements
  - if dataset has 3216 elements => ***last batch*** has 16 elements

Wrong tensor shape (3)

- 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
- 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])
```

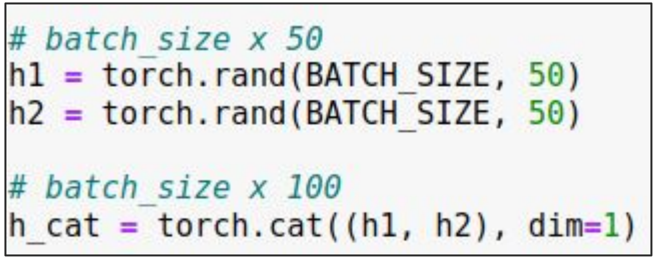
input size no longer matches first linear layer!

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

Wrong tensor shape (4)

good practices:

- write down tensor shapes after each operation, this will make writing and debugging shape issues more easily

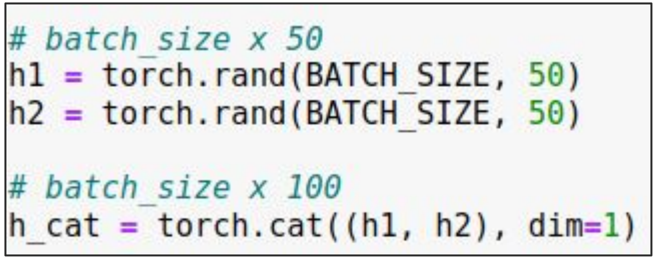


```
# batch_size x 50
h1 = torch.rand(BATCH_SIZE, 50)
h2 = torch.rand(BATCH_SIZE, 50)

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

Wrong tensor shape (4)

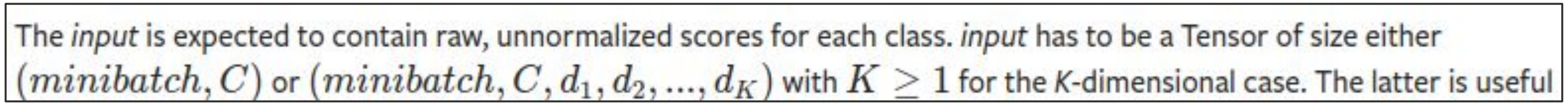
good practices:

- write down tensor shapes after each operation, this will make writing and debugging shape issues more easily



```
# 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)
```

- carefully read Pytorch documentation for layers (linear, conv2d etc), operations (concat, reshape, etc.) and loss functions

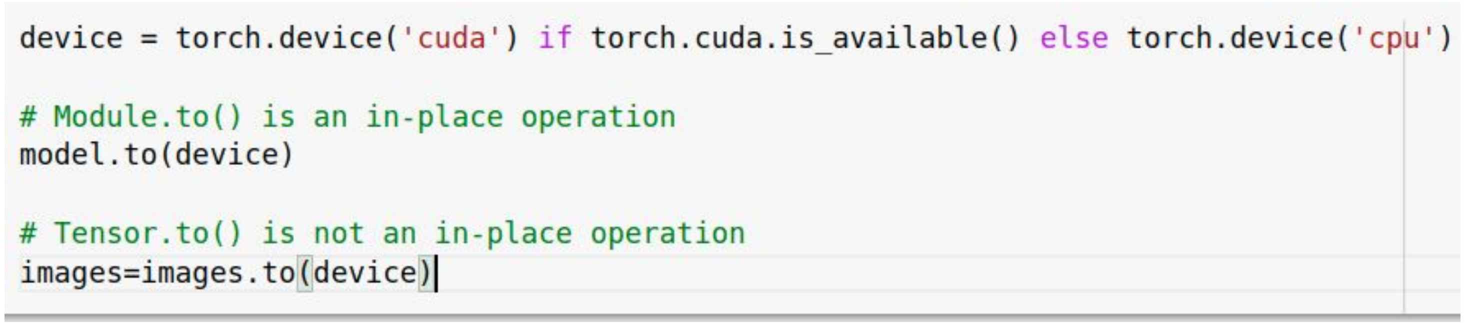


The *input* is expected to contain raw, unnormalized scores for each class. *input* has to be a Tensor of size either $(minibatch, C)$ or $(minibatch, C, d_1, d_2, ..., d_K)$ with $K \geq 1$ for the *K*-dimensional case. The latter is useful

documentation snippet from CrossEntropyLoss in Pytorch

Wrong tensor device

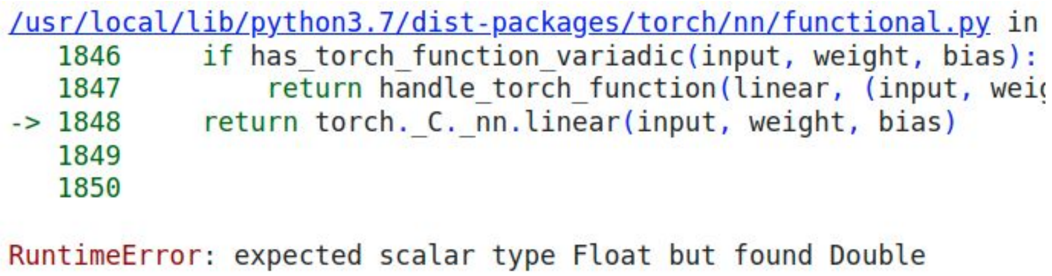
- tensors must be on the same device to perform most operations (no cpu-cuda or cuda1-cuda2 allowed)
- typical errors:
  - model was moved to GPU but input data is still on CPU due to **.to()** operation
  - model was loaded from a checkpoint but we forgot to move it to GPU
- tip: write device agnostic code to easily switch between the two



```
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)
```

Wrong tensor type

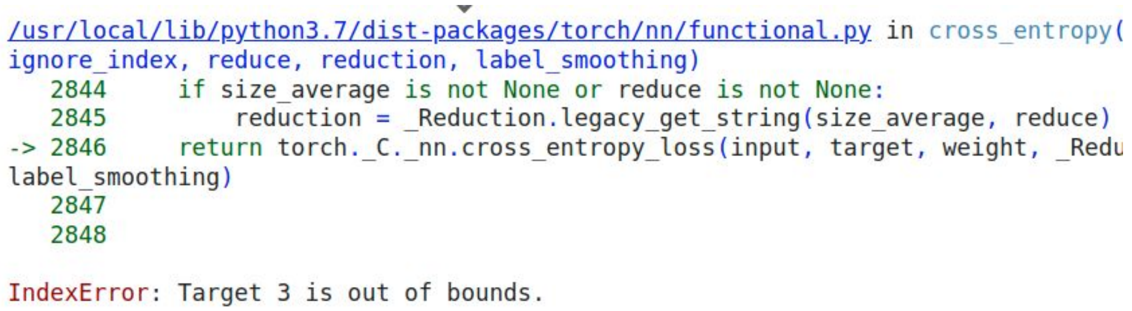
- typical errors:
  - Parameters in **torch.nn** layers have type float32 (Float) by default. If the dataset is stored as a NumPy array, it gets converted to Tensors of type float64 (Double), which are incompatible with the layers
  - labels are read as Float Tensors and we forget to convert them to Long. This usually throws an error when slicing or when computing loss (which usually requires integer label indices)



```
/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 range

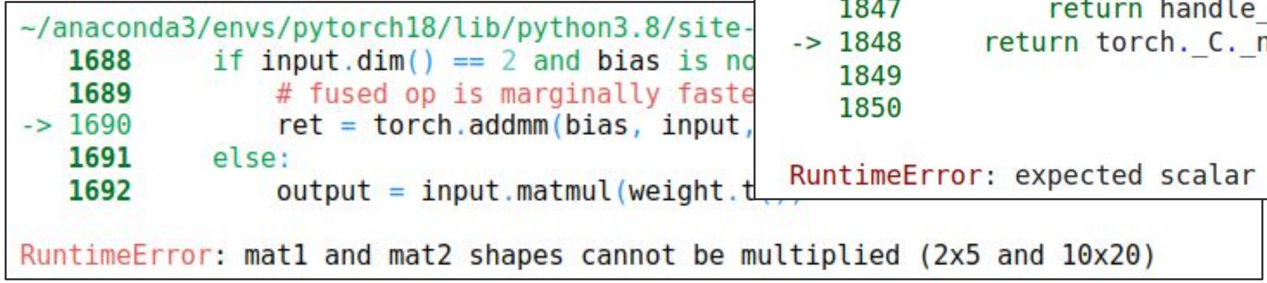
- typical errors:
    - label indices are stored as {1,2,...,C} instead of {0,1,...,C-1}. This throws an error in the cost function when accessing the score or label with index C (out of range)
    - when working with text data, each word in the vocabulary V gets assigned an unique index (0..|V|-1) and corresponding vector. If we feed an index that is out of range, the model has no word vector to represent it



```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py in cross_entropy(
ignore_index, reduce, reduction, label_smoothing)
   2844     if size_average is not None or reduce is not None:
   2845         reduction = _Reduction.legacy_get_string(size_average, reduce)
-> 2846     return torch._C._nn.cross_entropy_loss(input, target, weight, _Redu
label_smoothing)
   2847
   2848

IndexError: Target 3 is out of bounds.
```

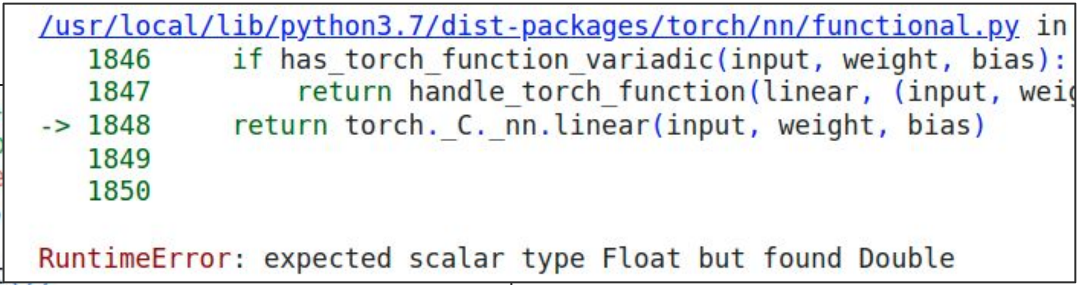
Syntactic Issues - recap

shape errors



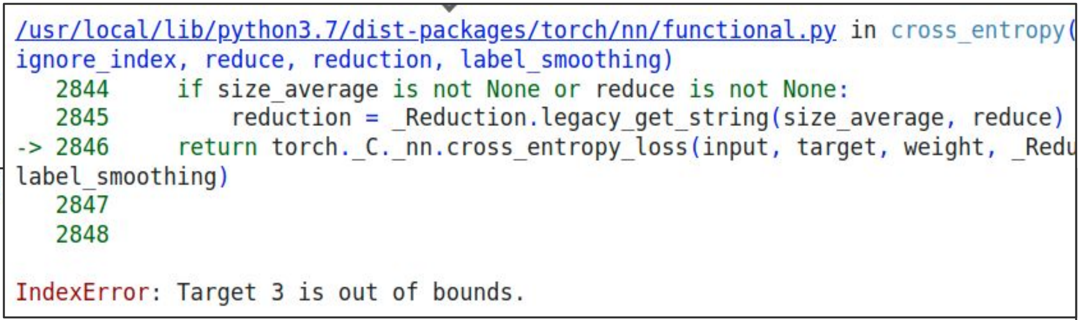
```
~/anaconda3/envs/pytorch18/lib/python3.8/site-
   1688      if input.dim() == 2 and bias is no
   1689          # fused op is marginally faste
-> 1690          ret = torch.addmm(bias, input,
   1691      else:
   1692          output = input.matmul(weight.t

RuntimeError: mat1 and mat2 shapes cannot be multiplied (2x5 and 10x20)
```

type errors



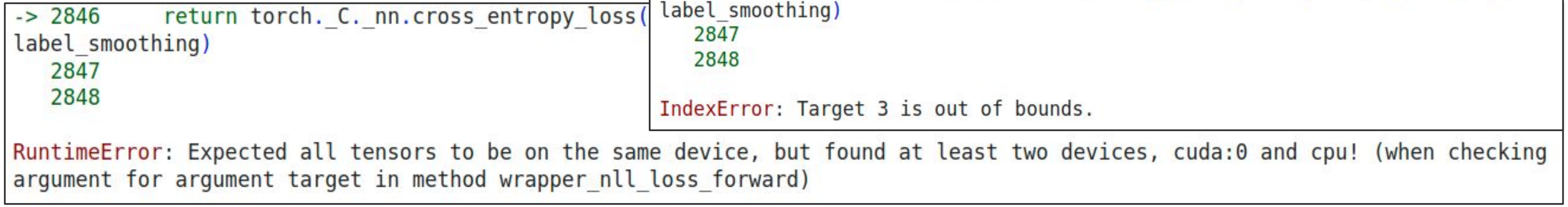
```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py in
   1846      if has_torch_function_variadic(input, weight, bias):
   1847          return handle_torch_function(linear, (input, wei
-> 1848      return torch._C._nn.linear(input, weight, bias)
   1849
   1850

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

range errors



```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py in cross_entropy(
ignore_index, reduce, reduction, label_smoothing)
   2844      if size_average is not None or reduce is not None:
   2845          reduction = _Reduction.legacy_get_string(size_average, reduce)
-> 2846      return torch._C._nn.cross_entropy_loss(input, target, weight, _Redu
label_smoothing)
   2847
   2848

IndexError: Target 3 is out of bounds.
```

device errors



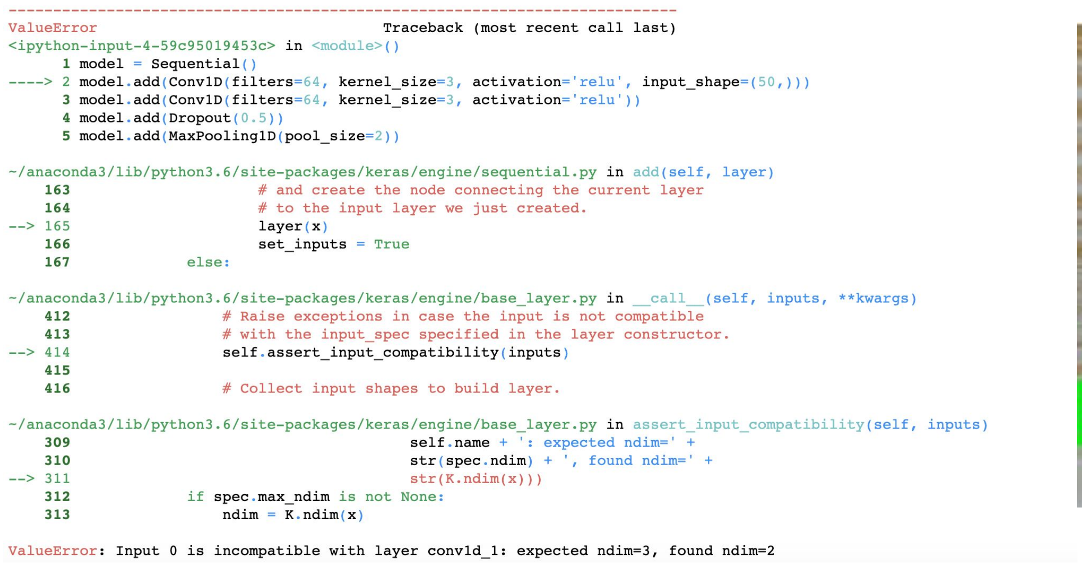
```
-> 2846      return torch._C._nn.cross_entropy_loss(
label_smoothing)
   2847
   2848

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)
```

Machine Learning bugs

Bitdefender

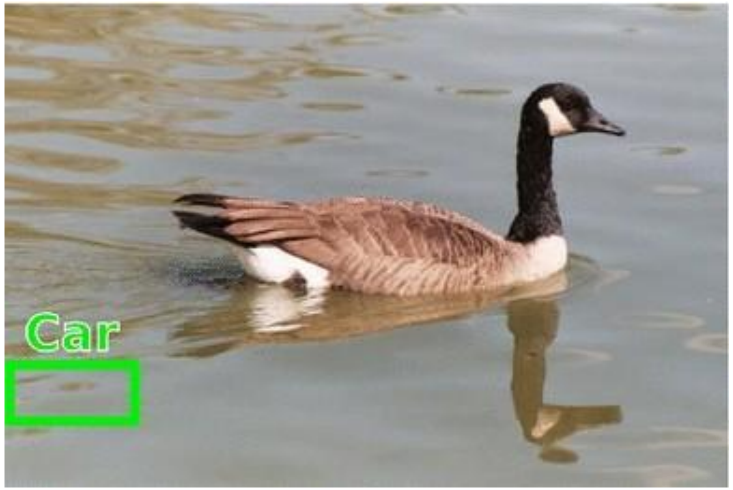
syntactic

functional



```
---------------------------------------------------------------------------
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
```

✓

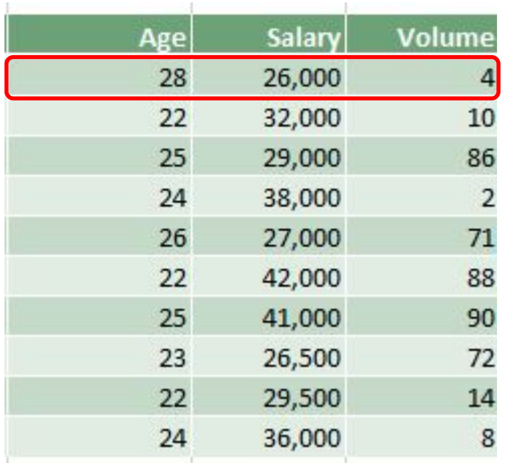


Machine Learning functional bugs

- code runs but model's performance is bad
- there are many sources of errors:
  - logic errors (typos, wrong inputs, wrong labels, incorrect arguments)
  - bad model hyperparameters
  - dataset issues

Numerical issues (1)

- normalization/standardization is:
  - missing
  - computed over the wrong dimension



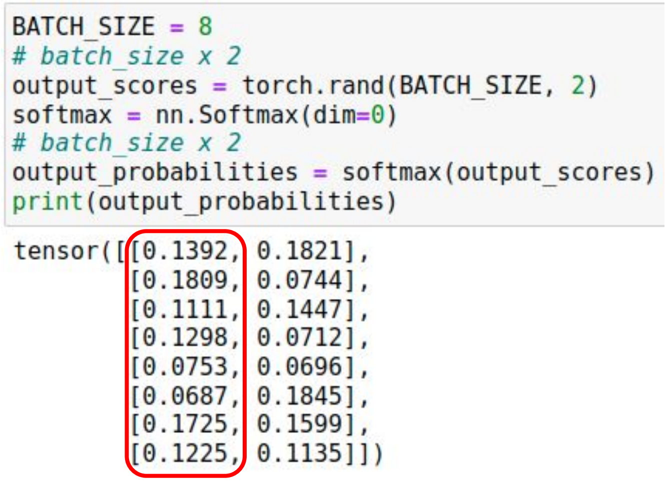
Age	Salary	Volume
28	26,000	4
22	32,000	10
25	29,000	86
24	38,000	2
26	27,000	71
22	42,000	88
25	41,000	90
23	26,500	72
22	29,500	14
24	36,000	8

Numerical issues (2)

Bitdefender

- softmax is:

  applied over the wrong dimension



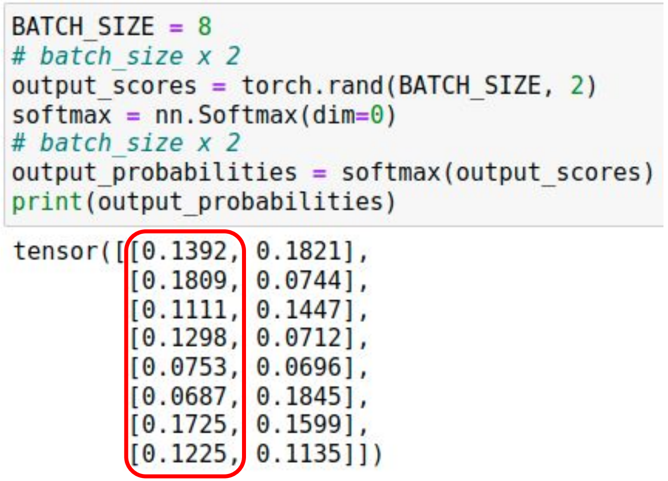
```
BATCH_SIZE = 8
# batch_size x 2
output_scores = torch.rand(BATCH_SIZE, 2)
softmax = nn.Softmax(dim=0)
# batch_size x 2
output_probabilities = softmax(output_scores)
print(output_probabilities)
```

```
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]])
```

Numerical issues (2)

- softmax is:

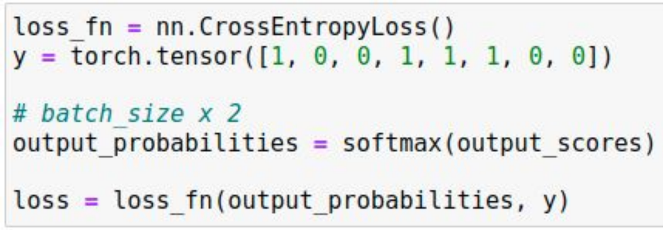
applied over the wrong dimension



```
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]])
```

unnecessary



```
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)
```

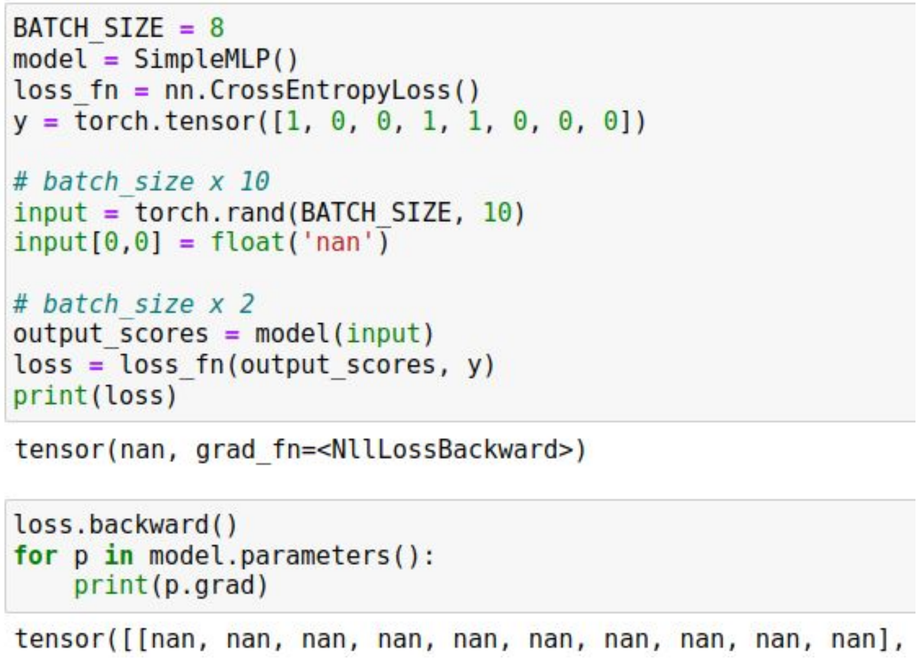
nn.CrossEntropyLoss() requires *raw unnormalized* scores as input

Numerical issues (3)

- train/evaluation mode mismatch:
  - when model is set in *eval()* mode, some operations behave differently (*Dropout*, *Batch Normalization* etc.)
  - **common mistake**: at the end of an epoch, after evaluation, model is NOT set back to *train mode*, which results in unwanted behaviour:
    - dropout is *disabled* during training
    - batch normalization uses *running estimates of mean and variance* during training instead of computing current batch statistics

Numerical issues (4)

- **NaN** (not a number) value appears in input data by mistake (missing/corrupted data, preprocessing error etc.)
- no errors raised during:
  - feedforward
  - loss computation
  - backpropagation
- tips: use **torch.isnan(x).any()** to check input tensor **x**



```
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)
```

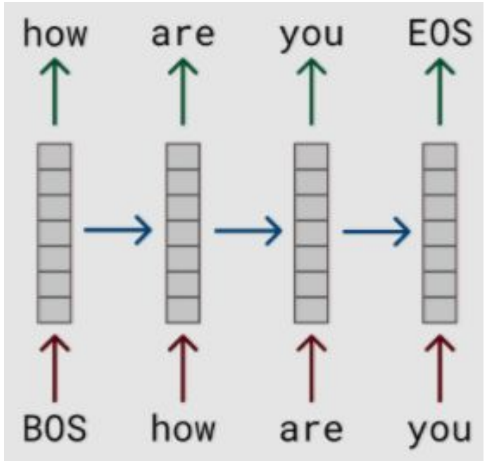
```
tensor([[nan, nan, nan, nan, nan, nan, nan, nan, nan, nan],
```

Numerical issues (5)

- **±Inf** (infinity) value shows up in loss or in some intermediate output
- possible causes:
  - learning rate too high => loss blows up
  - log(small value) = -inf
  - exp(large value) = +inf
- tips:
  - decrease learning rate
  - gradient clipping (rescale gradient if norm > threshold)
  - log(x+eps)

Input issues

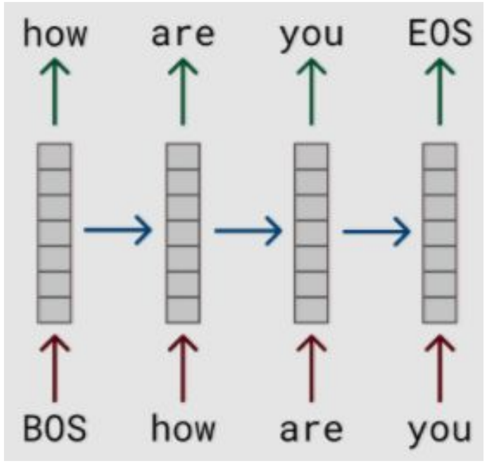
- in language models, for each input word you have to predict the *next* word
- labels = inputs shifted by one to the left



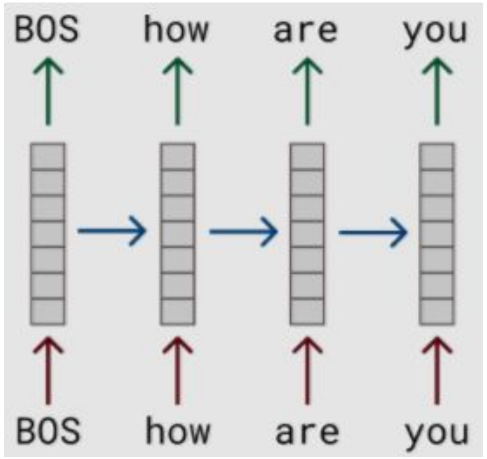
BOS = beginning of sentence, EOS = end of sentence

Input issues

- 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)

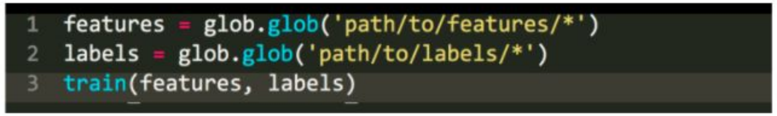


BOS = beginning of sentence, EOS = end of sentence

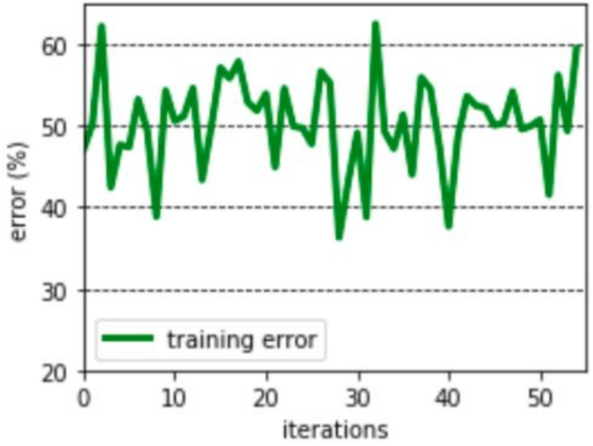


Input issues (2)

- 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)
```

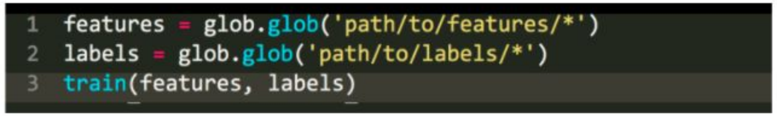


source: https://fullstackdeeplearning.com/spring2021/lecture-7/

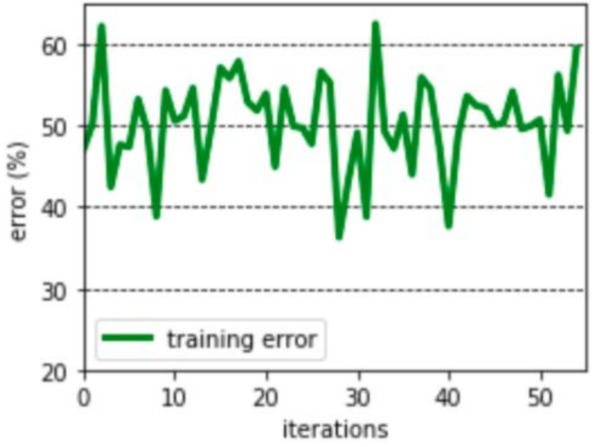
Bitdefender

Input issues (2)

- 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



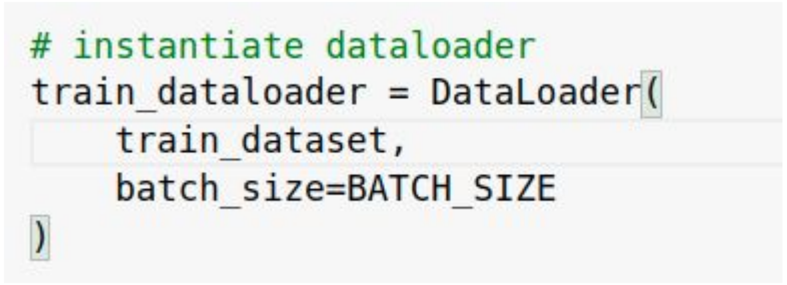
```
features = glob.glob('path/to/features/*')
labels = glob.glob('path/to/labels/*')
train(features, labels)
```



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

Input issues (3)

what is the issue?

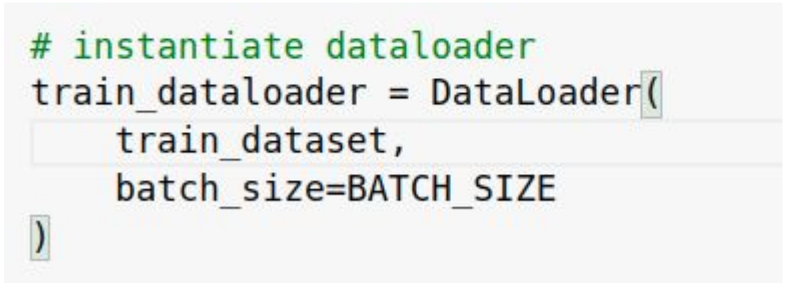


```
# instantiate dataloader
train_dataloader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE
)
```

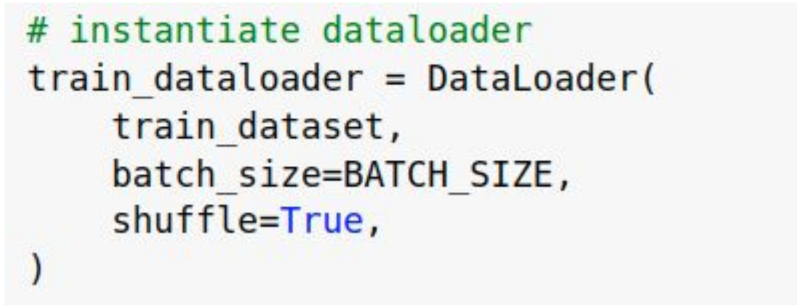
Input issues (3)

what is the issue?

- training examples are not shuffled



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

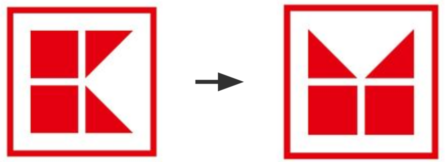


```
# instantiate dataloader
train_dataloader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True,
)
```

✓

Dataset issues

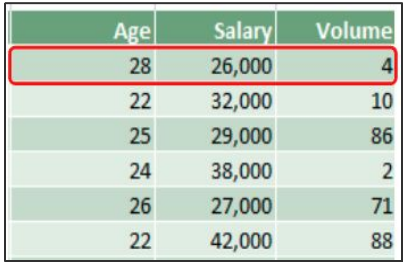
- dataset augmentation is commonly employed to improve performance
- however, some augmentations are not meaningful and actually hurt the ability of the model to generalize:
    - flipping/rotating logos may not improve logo recognition



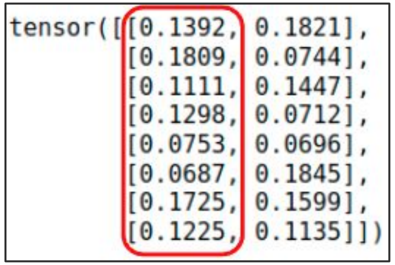
    - shifting or flipping EKG signal changes its meaning

Functional Issues - recap

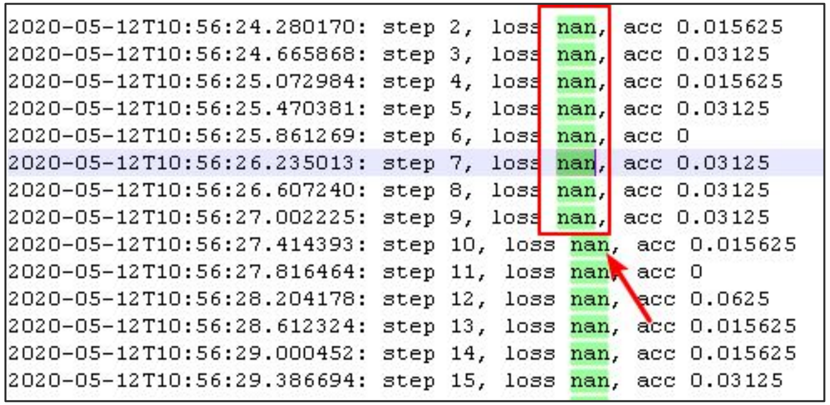
numerical errors (*bad normalization, softmax over the wrong dimension, train/eval model mismatch* etc.)



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

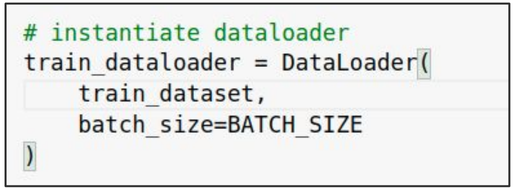


```
tensor([[0.1392, 0.1821],
        [0.1809, 0.0744],
        [0.1111, 0.1447],
        [0.1298, 0.0712],
        [0.0753, 0.0696],
        [0.0687, 0.1845],
        [0.1725, 0.1599],
        [0.1225, 0.1135]])
```

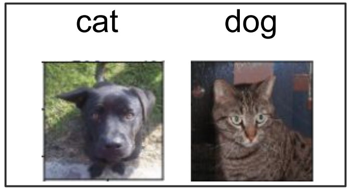


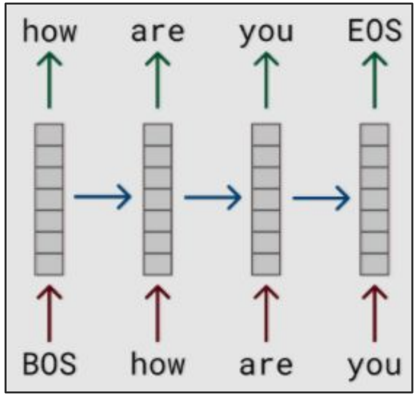
```
2020-05-12T10:56:24.280170: step 2, loss nan, acc 0.015625
2020-05-12T10:56:24.665868: step 3, loss nan, acc 0.03125
2020-05-12T10:56:25.072984: step 4, loss nan, acc 0.015625
2020-05-12T10:56:25.470381: step 5, loss nan, acc 0.03125
2020-05-12T10:56:25.861269: step 6, loss nan, acc 0
2020-05-12T10:56:26.235013: step 7, loss nan, acc 0.03125
2020-05-12T10:56:26.607240: step 8, loss nan, acc 0.03125
2020-05-12T10:56:27.002225: step 9, loss nan, acc 0.03125
2020-05-12T10:56:27.414393: step 10, loss nan, acc 0.015625
2020-05-12T10:56:27.816464: step 11, loss nan, acc 0
2020-05-12T10:56:28.204178: step 12, loss nan, acc 0.0625
2020-05-12T10:56:28.612324: step 13, loss nan, acc 0.015625
2020-05-12T10:56:29.000452: step 14, loss nan, acc 0.015625
2020-05-12T10:56:29.386694: step 15, loss nan, acc 0.03125
```

input issues (*data not shuffled, out of order labels, input is not fed correctly to model* etc.)

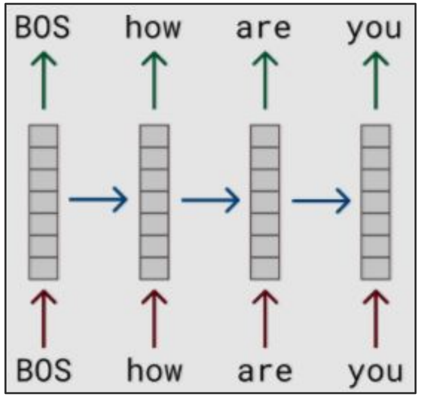


```
# instantiate dataloader
train_dataloader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE
)
```





✓



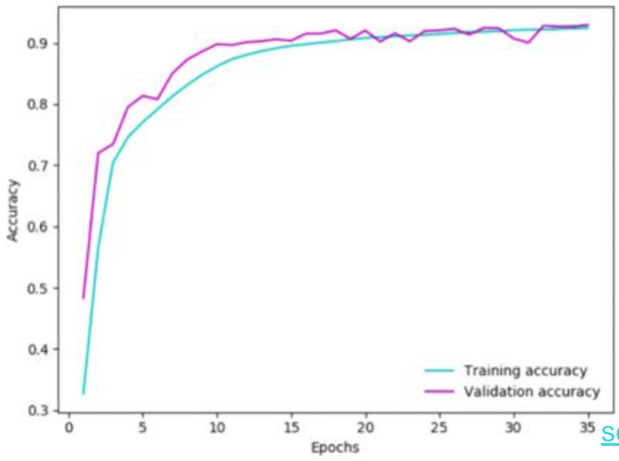
✗

Debugging plan

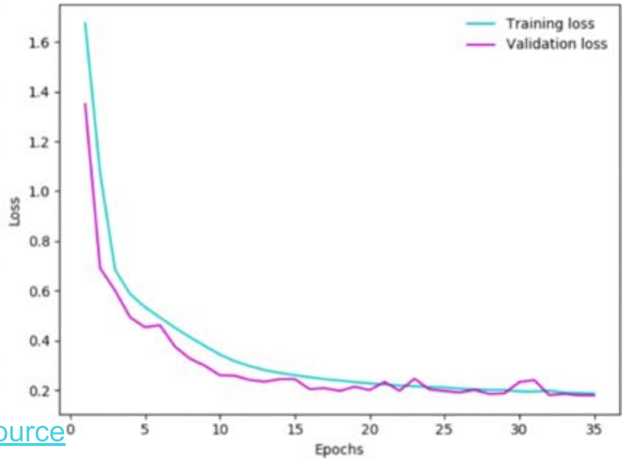
- run code and fix syntactic bugs (*shape/device/type mismatches*, *OOM issues*)
- overfit model on a few (1-2) examples
  - if training loss ~0, start training on full dataset
  - if not, check for invisible bugs:
    - manually check input:
      - lots of zeros or NaN (*preprocessing error, corrupted 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*)
- start training on the full dataset
  - inspect loss/metric learning curves
  - we may still have invisible bugs at this point! (*train data leaks into validation*, *bug when measuring validation error* etc.)

Inspecting learning curves

- We plot:
  - training loss and validation loss curves
  - training metric and validation metric curves (where metric = accuracy, F1, BLEU etc.)
- Based on the shape and values of these learning curves:
  - we inspect whether the model is training correctly
  - we choose the best model



source

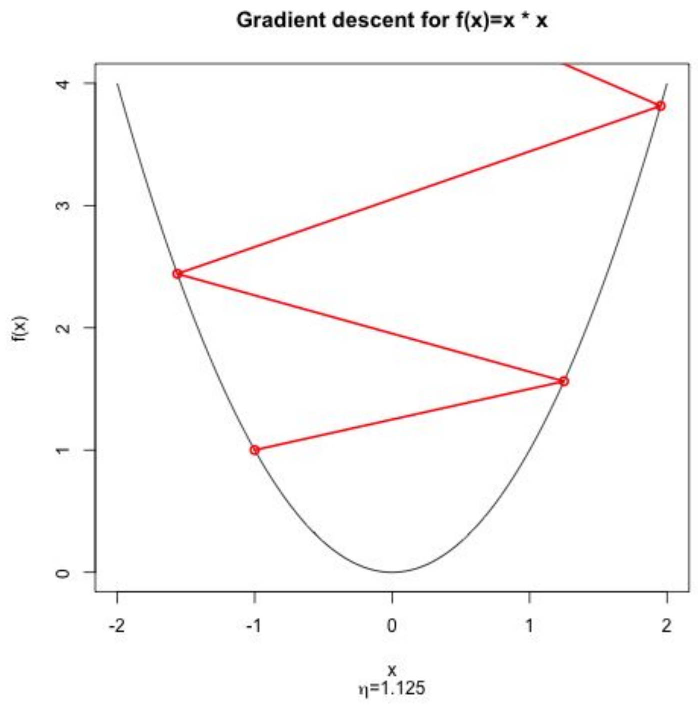


Learning curves scenarios

- training loss increases
- training loss stops decreasing
- training loss decreases slowly
- training loss decreases

Training loss increases

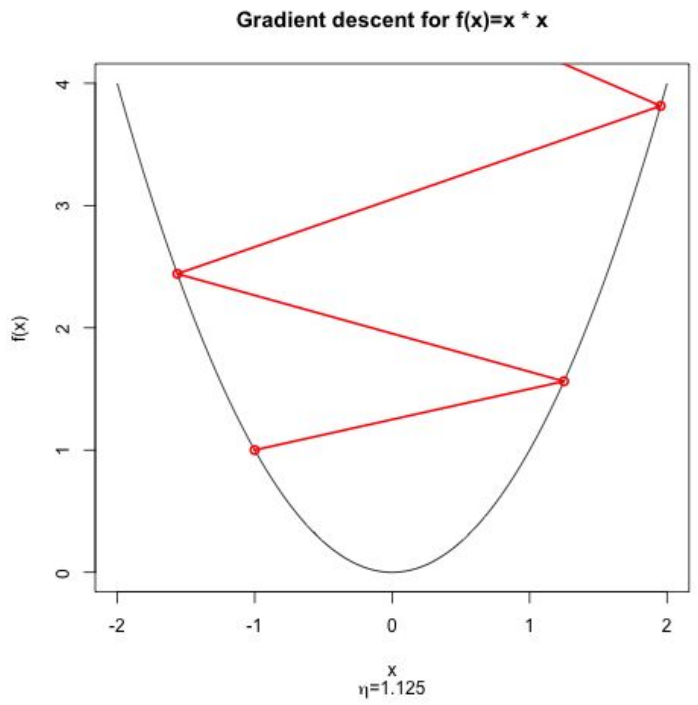
- Q: what does it usually indicate?



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

Training loss increases

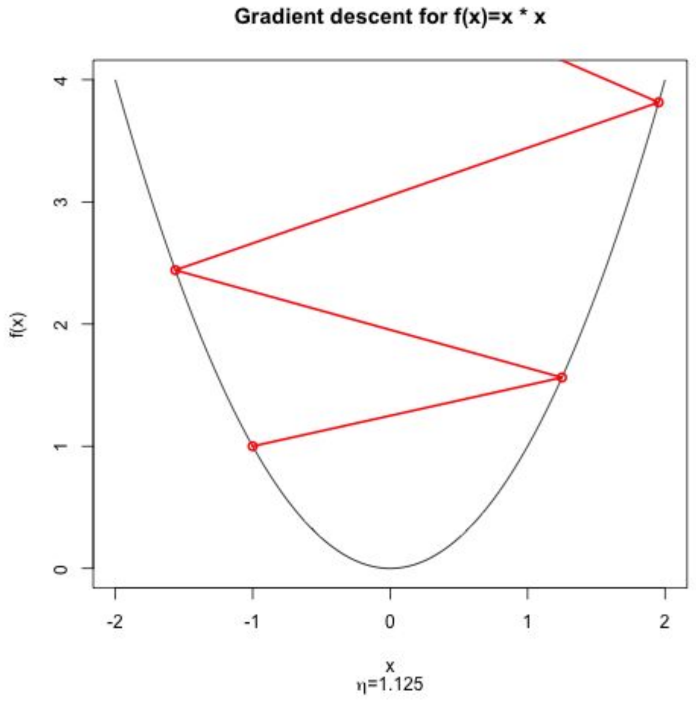
- Q: what does it usually indicate?
  A: the learning rate may be *too high*, moving the parameters away from the local minimum



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

Training loss increases

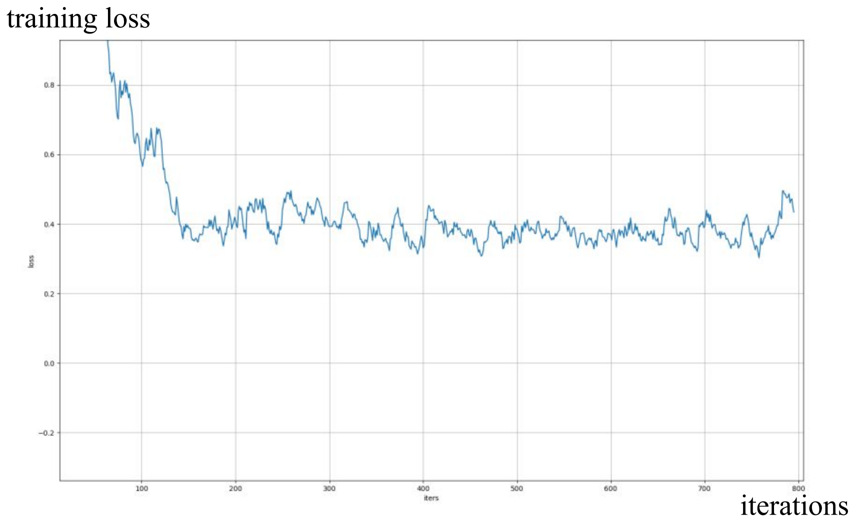
- Q: what does it usually indicate?
  A: the learning rate may be *too high*, moving the parameters away from the local minimum
- **solution**: decrease learning rate



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

Training loss stops decreasing

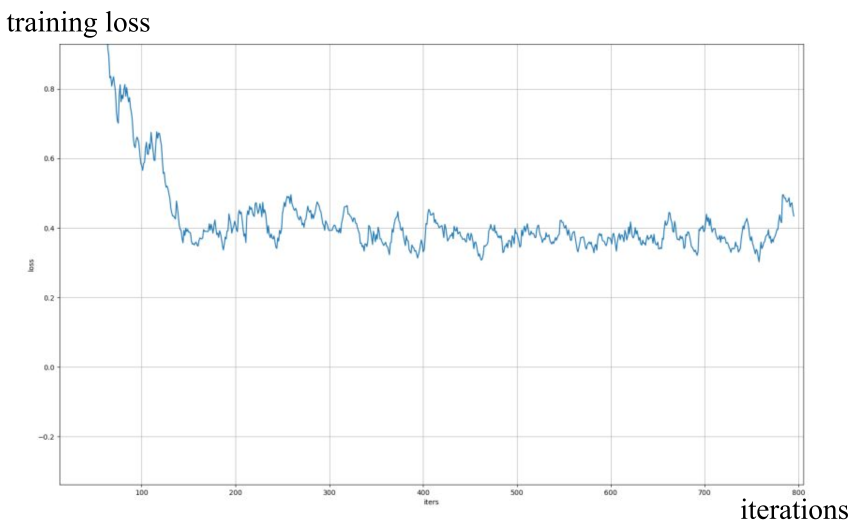
- Q: what could it mean?



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

Training loss stops decreasing

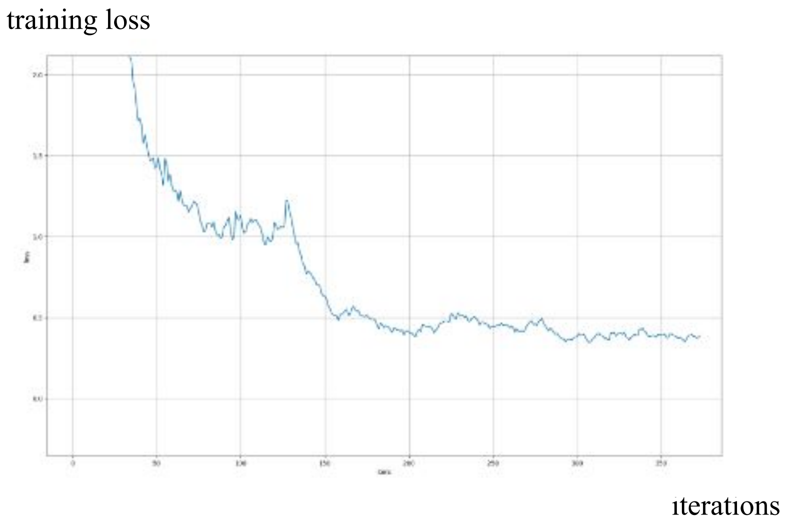
- Q: what could it mean?
- A: it may still indicate a learning rate that is too high



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

Training loss stops decreasing

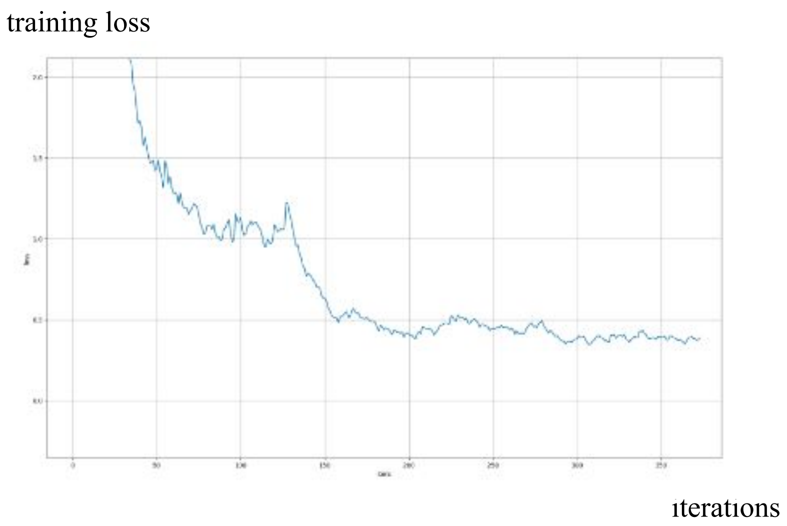
- Q: what could it mean?
- A: it may still indicate a learning rate that is *too high*
- solutions:
    - if it happens in the early stages of training, start with a smaller learning rate (halve it)



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

Training loss stops decreasing

- Q: what could it mean?
- A: it may still indicate a learning rate that is *too high*
- solutions:
    - if it happens in the early stages of training, start with a smaller learning rate (halve it)
    - if training loss reaches plateau after a few epochs, use learning rate scheduler (ReduceLROnPlateau - reduces lr when metric stops improving)

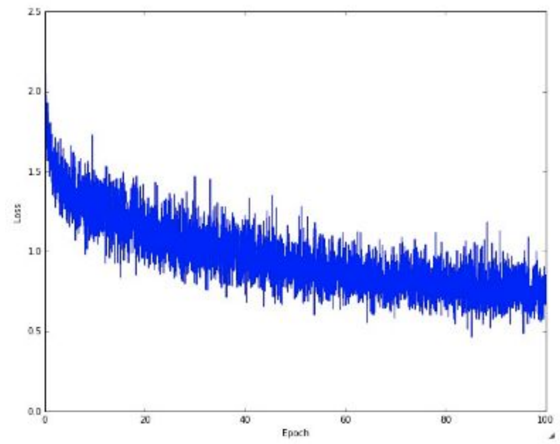


source: https://jumabek.wordpress.com/2017/03/21/781/

Training loss decreases slowly

- Q: what could it mean?

training loss



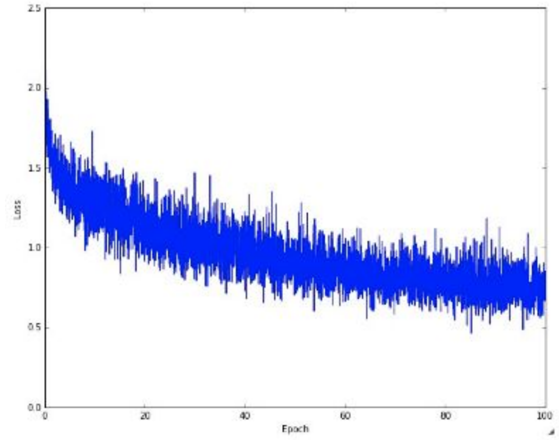
epochs

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)
- solution: higher learning rate

training loss

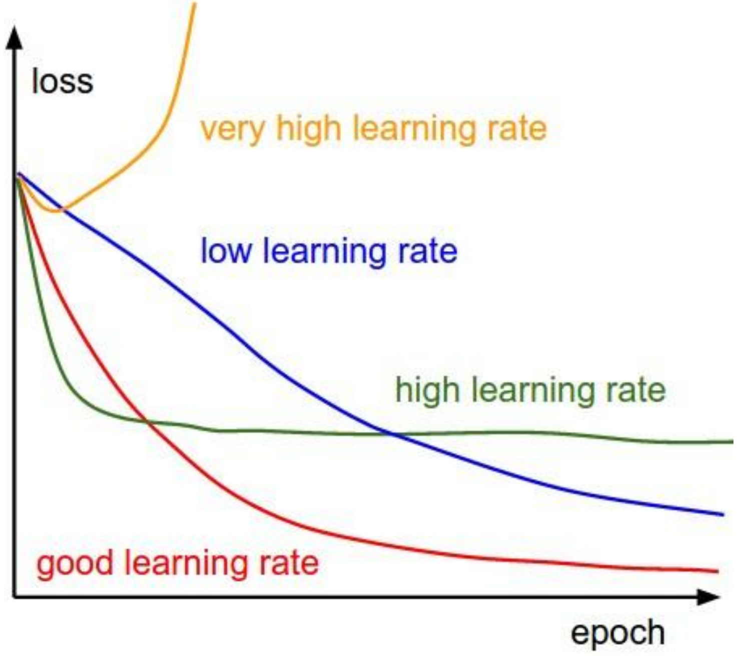


epochs

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

Shape of training loss curves

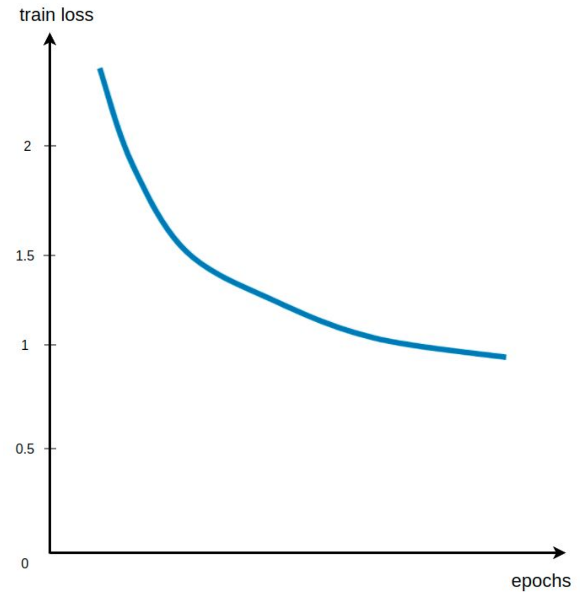
- the shape of the loss curve provides clue about the learning rate:
  - loss increases early in the training (*very high learning rate*)
  - loss decreases quickly then reaches a plateau (*high learning rate*)
  - loss decreases linearly (potentially *low learning rate*)
- however, we care more than the shape



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

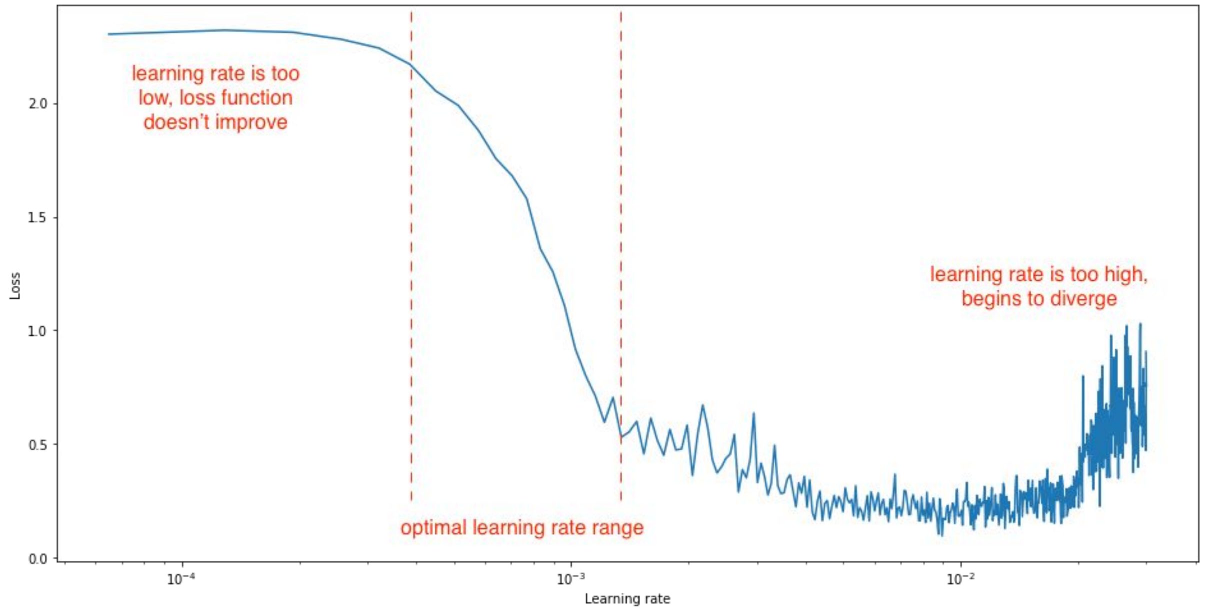
Is this a good training curve?

- if training loss stabilizes around a value that is still high, the model is *underfitting*
- solutions:
    - the model is too small to learn the training dataset => *increase model capacity* (more layers, more neurons per layer)
    - the model has enough capacity, but:
        - it is heavily regularized => *reduce regularization* (smaller Dropout rate, smaller L2 coefficient)
        - it has a lot of frozen parameters => unfreeze some of them



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 stops decreasing
- training loss decreases slowly
- training loss decreases

we have looked at possible issues with training curves!

Learning curves scenarios

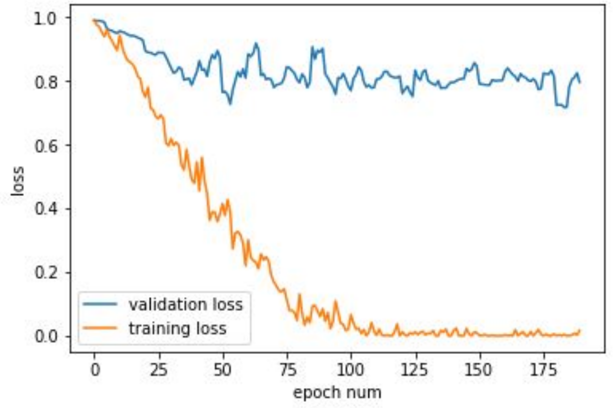
- training loss increases
- training loss stops decreasing
- training loss decreases slowly
- training loss decreases, but:
    - validation loss >> training loss
    - validation loss ≈ training loss, then diverges
    - validation loss ≈ training loss
    - validation loss < training loss
    - validation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

we have looked at possible issues with training curves!

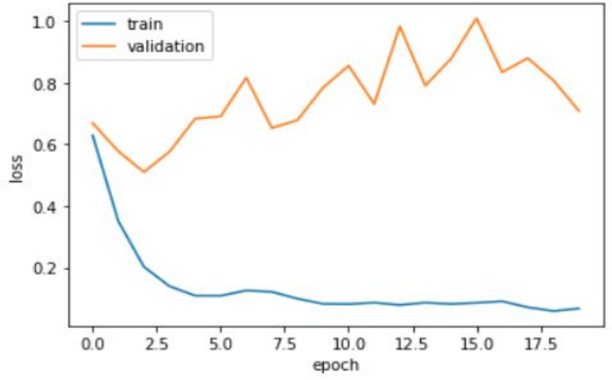
however, a good training curve says nothing about the generalization capability of the model => inspect validation curves!

Validation loss >> training loss

- Q: overfitting or underfitting?



source

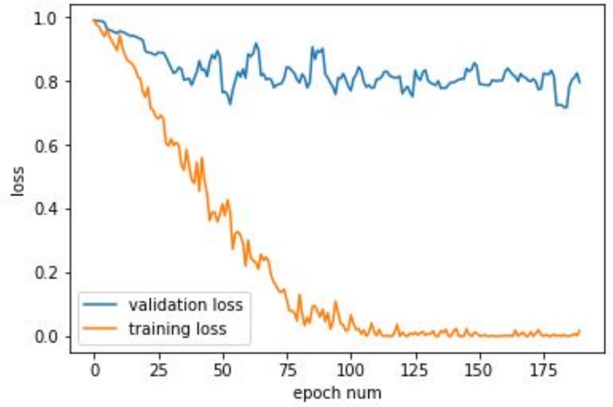


source

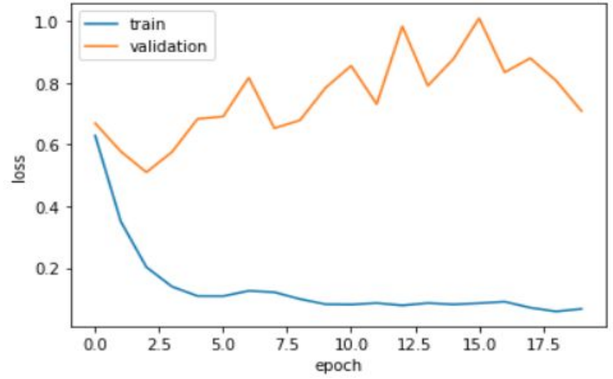
Bitdefender

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)
  - reduce model size (fewer layers/neurons per layer)



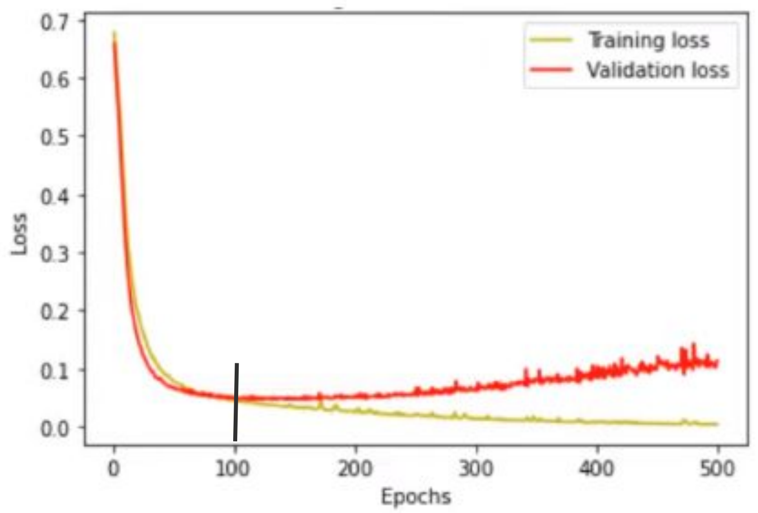
source



source

Validation loss tracks train loss then diverges

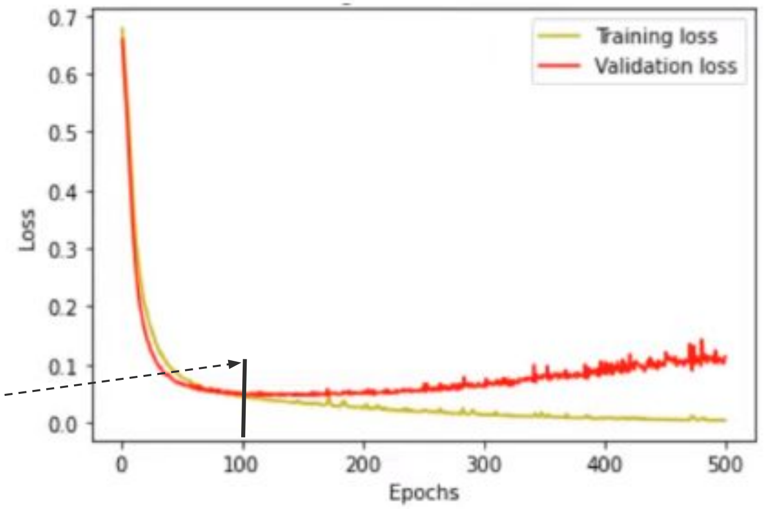
- Q: overfitting or underfitting?



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

Validation loss tracks train loss then diverges

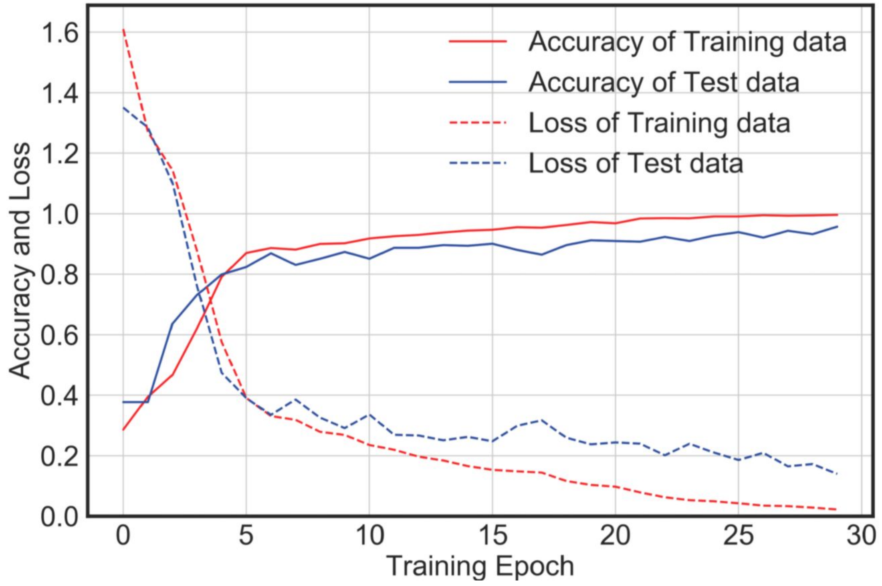
- Q: overfitting or underfitting?
- A: overfitting from epoch 100 onwards
  - validation loss tracks training loss, but diverges from epoch 100
- solutions:
  - early stop at epoch 100
  - add regularization (Dropout)



source: https://www.youtube.com/watch?v=p3CcfljycBA

Validation loss ≈ training loss

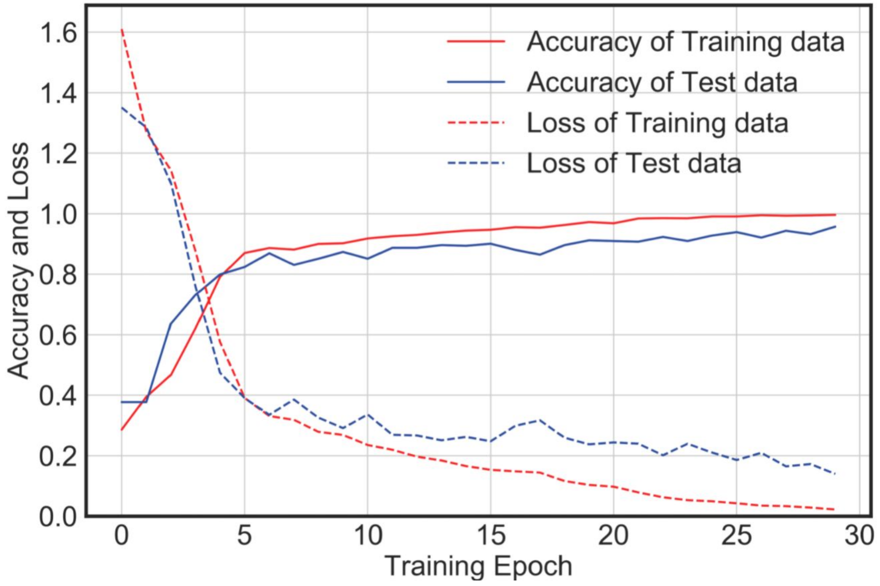
- Q: overfitting or underfitting?



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

Validation loss ≈ training loss

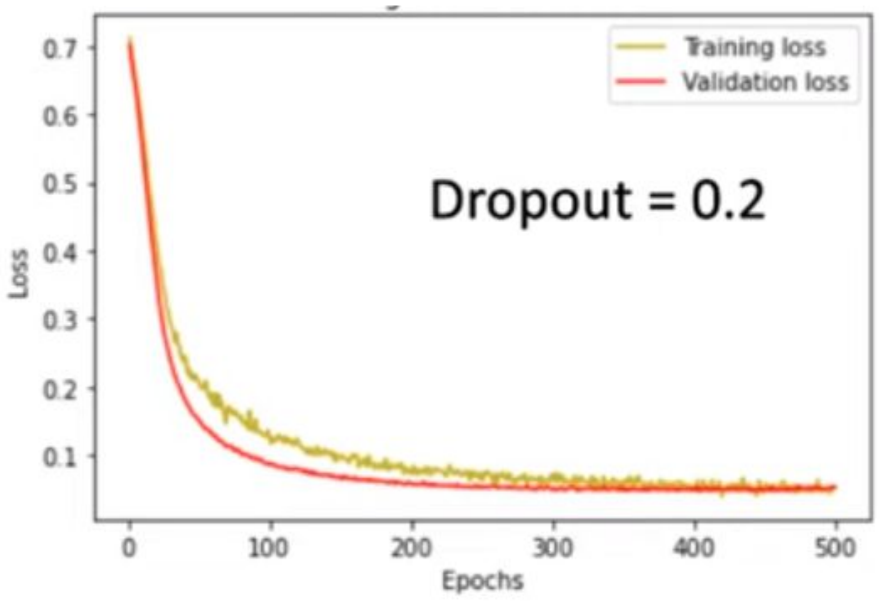
- Q: overfitting or underfitting?
- A: good fit
  - training and validation loss approach 0, with small gap between them
  - validation loss usually (but now always) slightly higher than training loss
  - training and validation accuracy also increase simultaneously (they are not always correlated with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



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

Validation loss < training loss

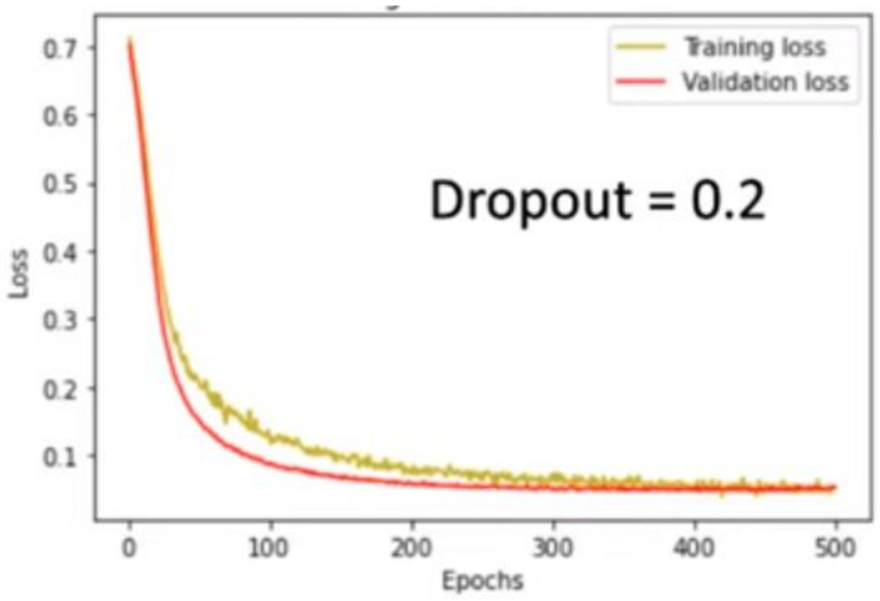
- Q: validation loss should normally be slightly higher than training loss, why is it lower here?



source: https://www.youtube.com/watch?v=p3CcfljycBA

Validation loss < training loss

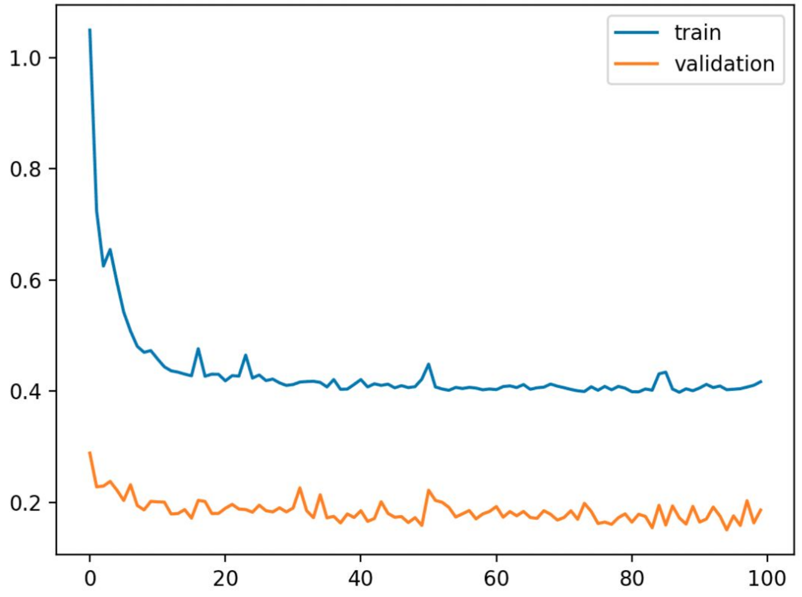
- Q: validation loss should normally be slightly higher than training loss, why is it lower here?
- A: the model is still a good fit if validation loss slightly lower than training loss
  - this effect may be due to Dropout, which disables some of the neurons during training
  - this lowers training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

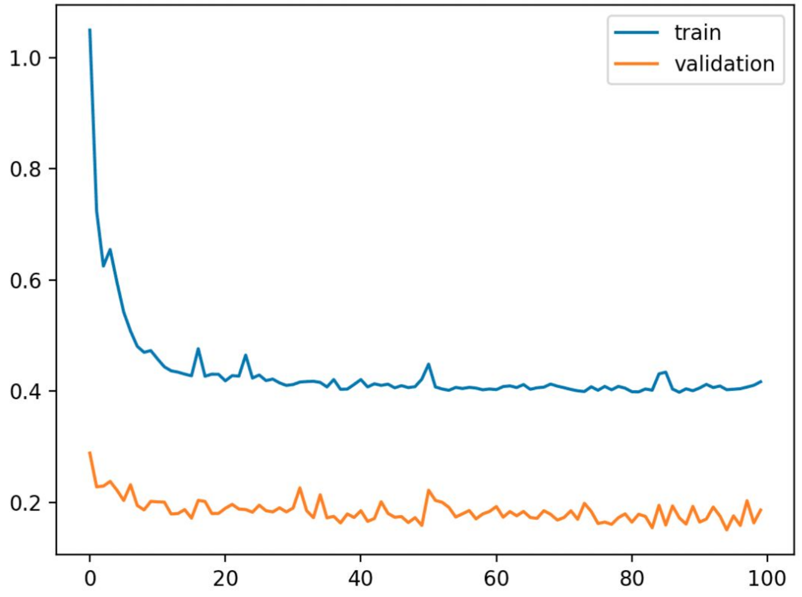
- Q: overfitting or underfitting?



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

Validation loss << training loss

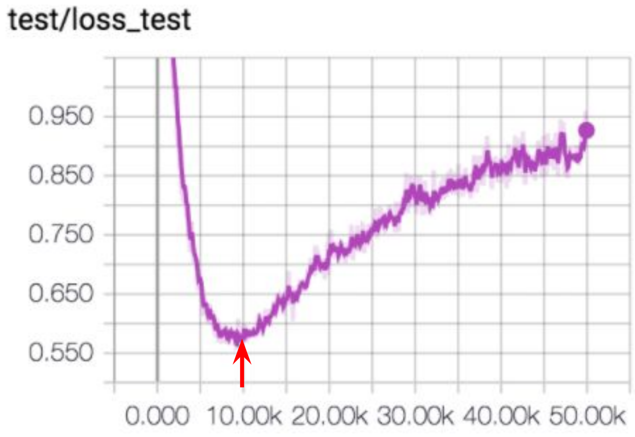
- Q: overfitting or underfitting?
- A: neither
    - validation loss is much lower than training loss, which means the validation set is 'easier' for the model than the training set
    - this may point to potential problems with the validation set:
        - validation examples easier than training examples (due to some bug in the splitting process or augmenting training examples only)
        - some training examples leaked into the validation set

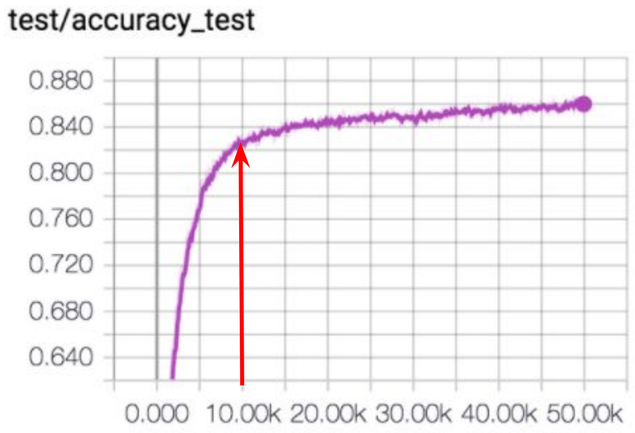


source:
https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Both validation loss & accuracy increase

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



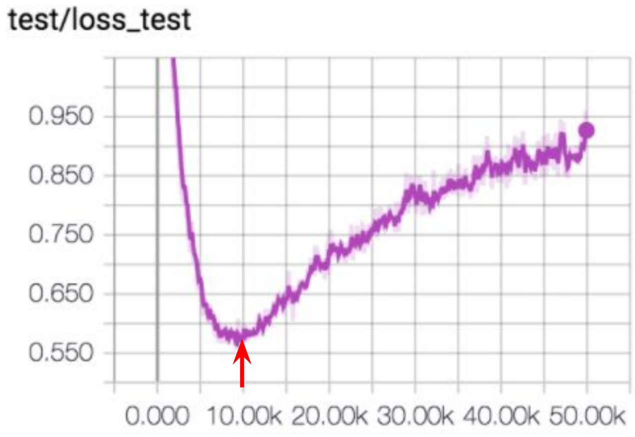


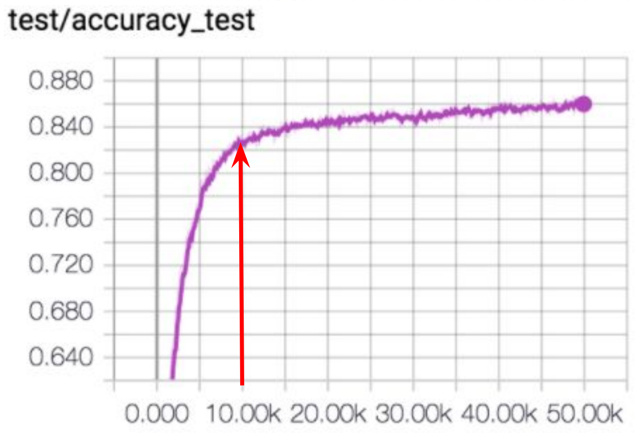
source

Both validation loss & accuracy increase

- Q: lowest validation loss is at ~10k steps, but validation accuracy continues to increase after that, why?
- A: model is making fewer mistakes, but:
    - may be less certain when making correct predictions (correct probability goes from 0.9 to 0.6 => loss increases from 0.1 to 0.5)
    - may be more confident when making incorrect predictions (loss increases from 0.91 to 2.3)

labels	0	**1**
old predictions	0.6	0.4
new predictions	0.9	0.1





source

Learning curves TL;DR

- validate that the model is training correctly:
    - train/validation loss curves show a good fit
    - train/validation loss curves indicate beginning of overfit
- choose the best model:
    - this is a bit subjective
    - we usually care about the *metric on the validation set*, so apply *early stopping* at the epoch where:
        - validation metric (accuracy/F1/etc.) stops increasing
        - validation loss starts increasing
    - if model starts overfitting on validation loss but not on validation metric:
        - choose epoch where validation loss is lowest or where it hasn’t degraded significantly