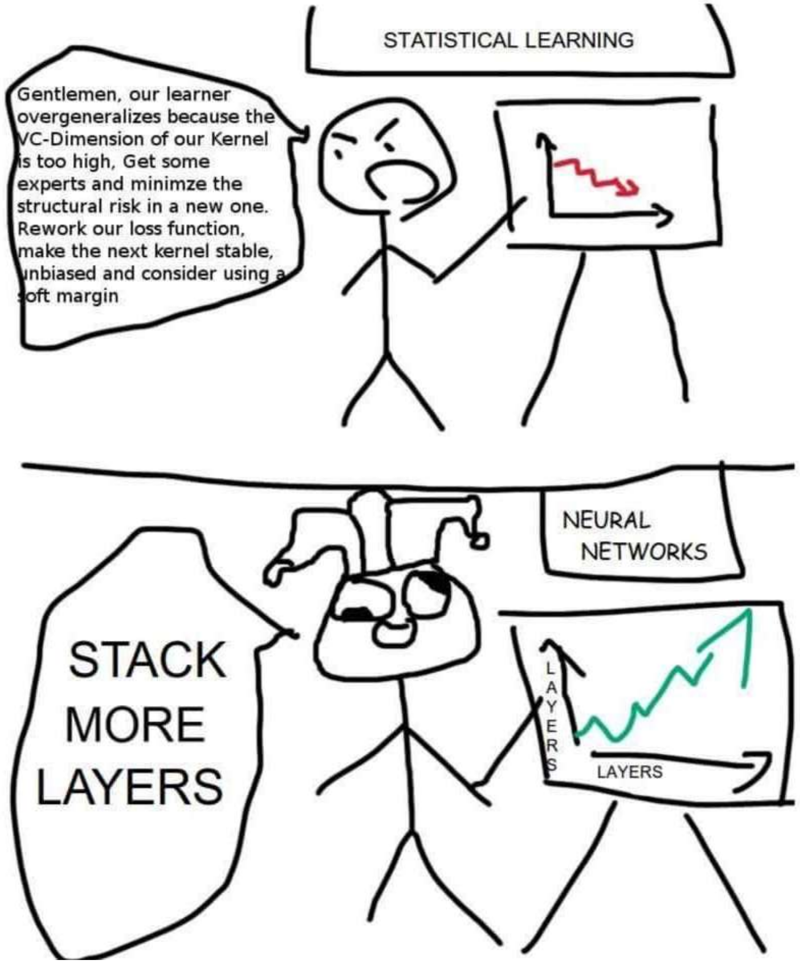
Tips and tricks for tuning NNs

Apr 20

Fei & Nish

Tuning NNs can be a dark art



When NNs don’t work, you have many choices

Your choices: (from Andrew Ng’s talk at Bay Learn’17)

- *Fetch more data*
- *Add more layers to Neural Network*
- *Try some new approach in Neural Network*
- *Train longer (increase the number of iterations)*
- *Change batch size*
- *Try Regularization*
- *Check Bias Variance trade-off to avoid under and overfitting*
- *Use more GPUs for faster computation*
- *...*

How to systematically attempt to fix a NN?

Abstraction of hyperparameter tuning

Your input : Hyper parameters

- Architecture (#layers, #kernels, stride, kernel size)
- Learning rate, optimizer (momentum)
- Regularizations (weight decay rate, dropout probability)
- Batchnorm / no batchnorm

Your output: Some diagnostic statistics:

- Loss curves
- Gradient norms
- Accuracy / Visual output (generative models)
- Performance on training vs validation set
- Other abnormal behaviors

Architecture

Using or adapt from established networks (will see more later this quarter)

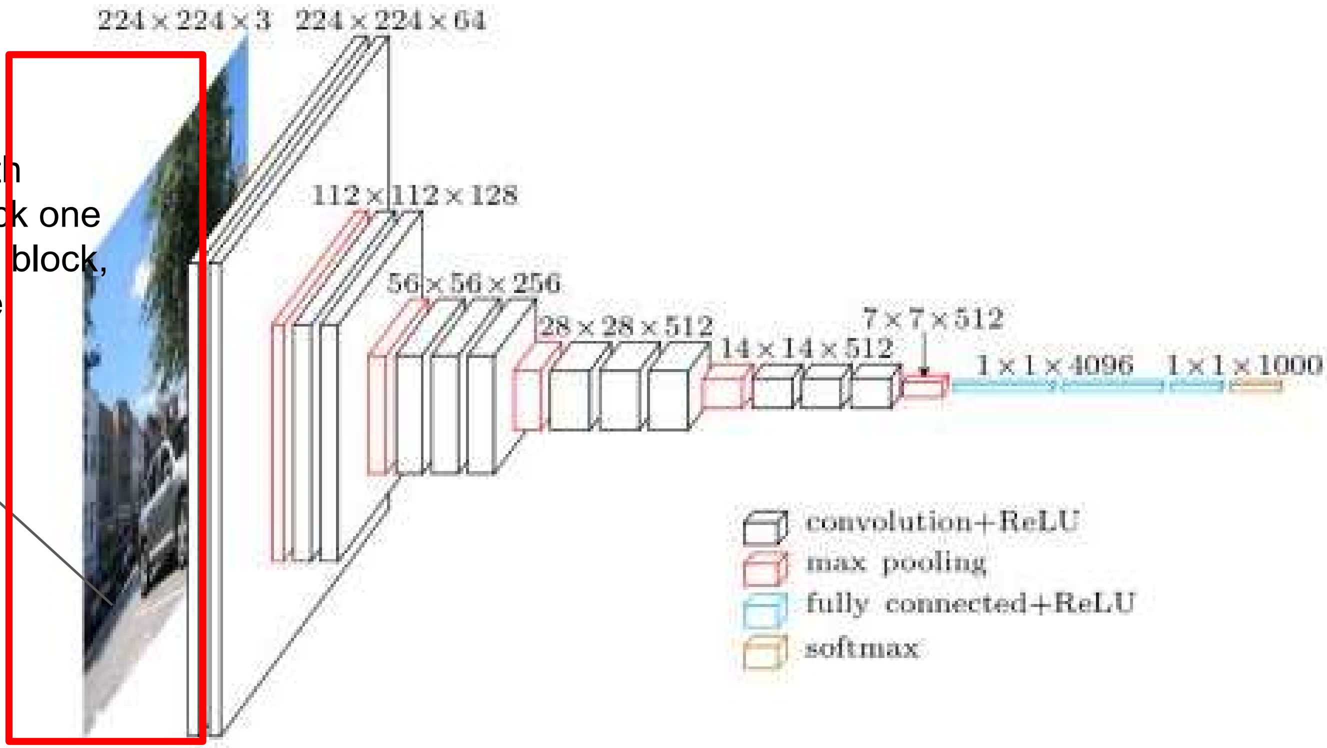
- Classification: AlexNet, VGG, ResNet, DenseNet, …
- Segmentation: FCN, Dilated Convolution, Mask RCNN
- Detection: Faster-RCNN, YOLO, SSD
- Image Generation: UNet, Dilated Convolution, DCGAN, WGAN
- …

How to adapt:

- Change number of kernels to ideal numbers
- Remove / add layers
- Change the structure of last few layers for your task

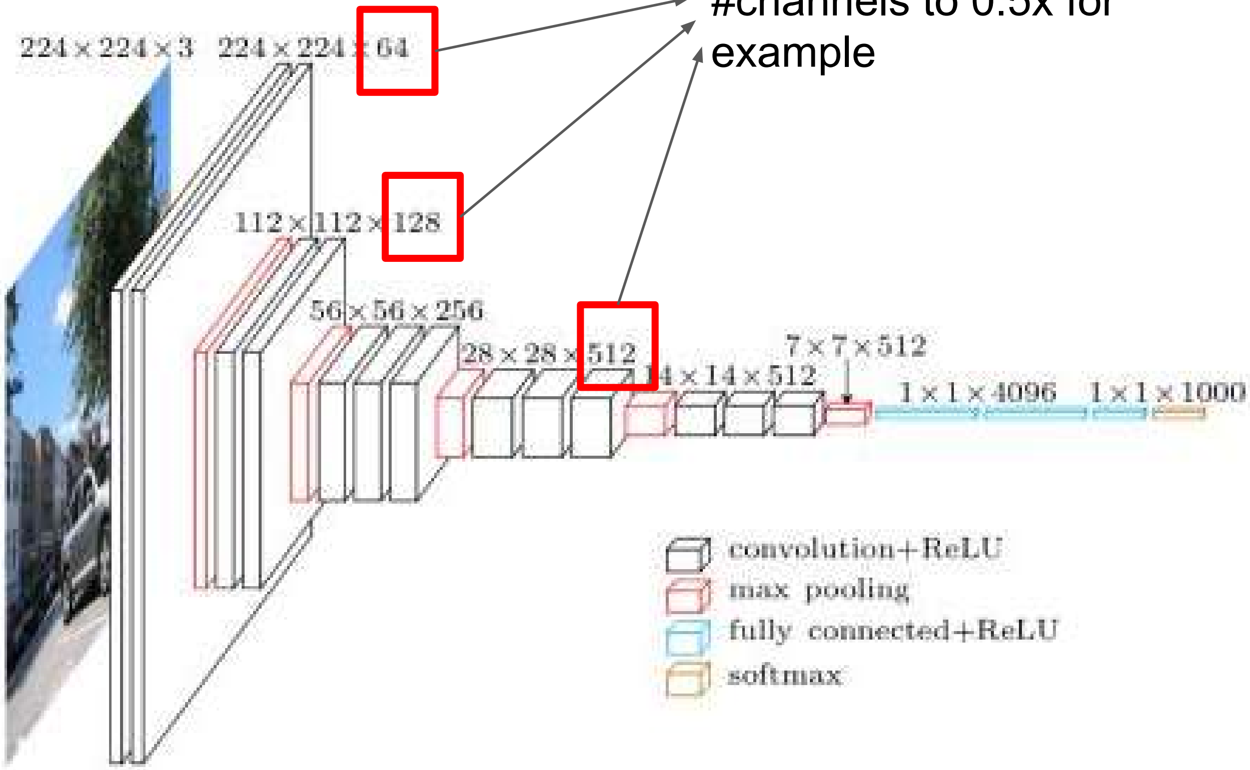
Architecture: adapt to different input

Need to deal with larger input, stack one more conv+pool block, or change kernel size/stride



Let’s say you saw this awesome network on some related work, but the input is not the same.

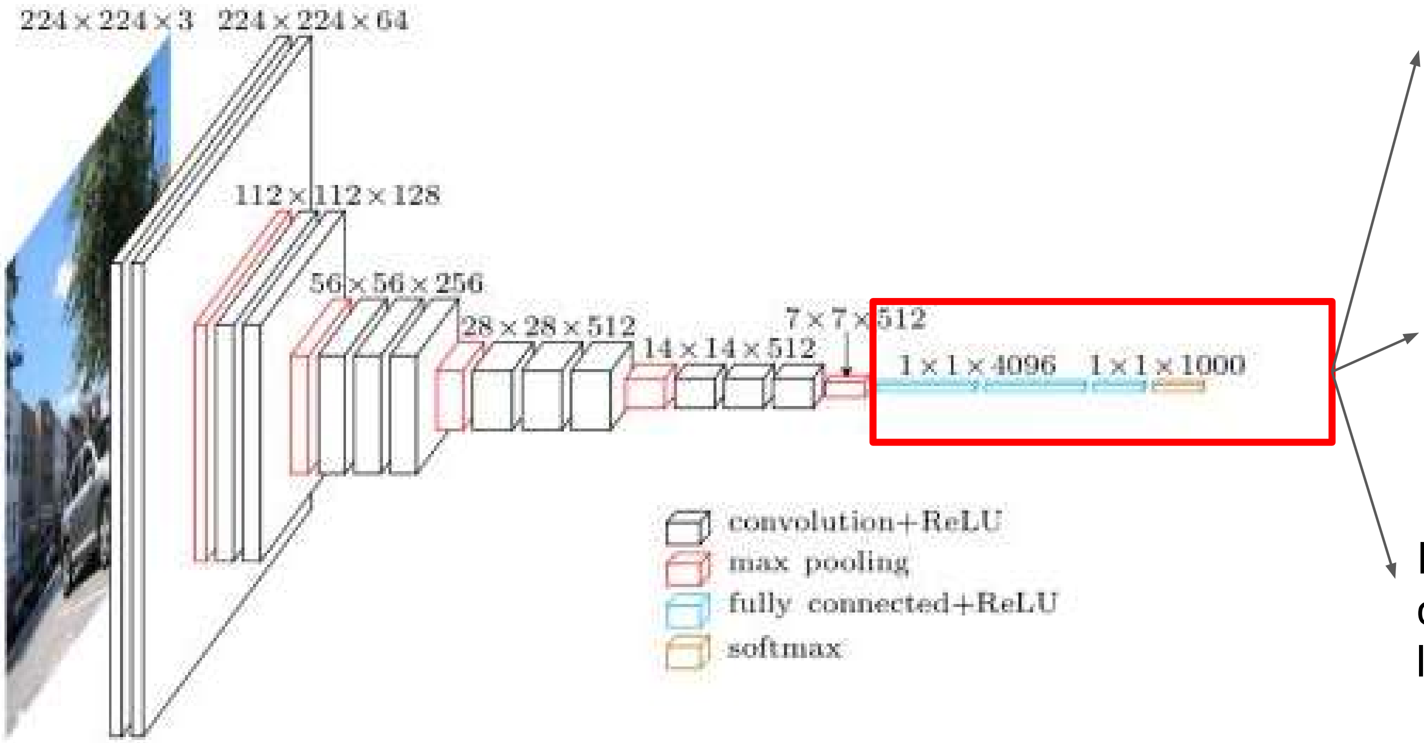
Architecture: for different size of dataset



Small dataset, change #channels to 0.5x for example

Let’s say you saw this awesome network on some related work, but the dataset you have is much smaller

Architecture: for tasks



Regression task:
Change to FC +
linear activation

Captioning task:
change to LSTM
decoder

Image generation task:
change to deconvolution
layers

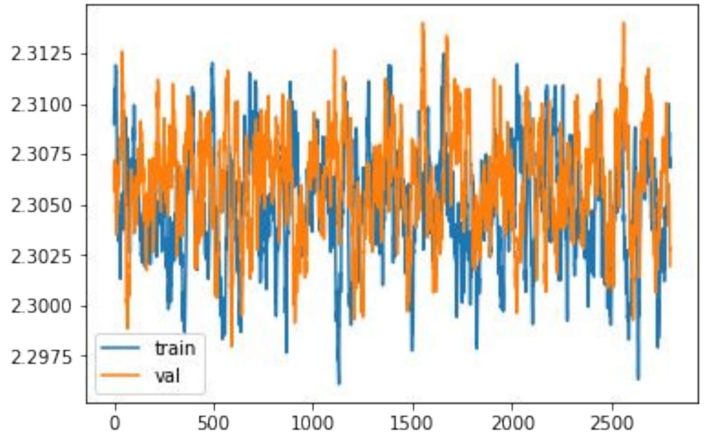
Let’s say you saw this awesome network on some related work, but the task is not the same.

Fast iteration

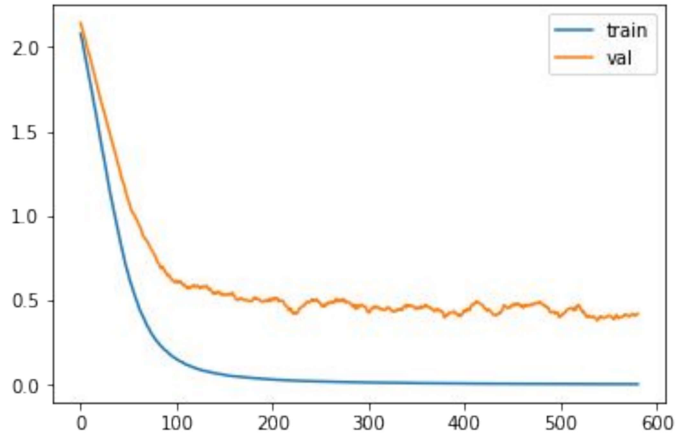
1. When tuning parameters, only use a small portion of the dataset for fast iteration.
2. Start from a larger learning rate for fast prototyping.

```
Class YourAwesomeDataset(torch.data.Dataset):

    def __init__(root_dir, debug=False):

        ...

         if debug:

             train_files = train_files[:len(train_files)//10]

             val_files = val_files[:len(val_files)//10]
```

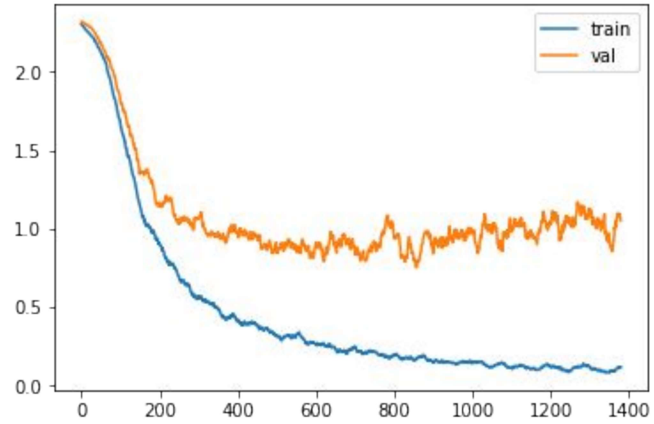
Loss curves: what are the problems?



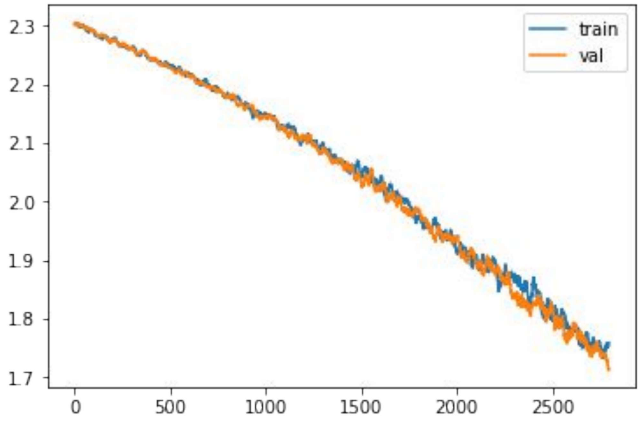
Not learning: gradients not applied to weights



Overfit: model too large/dataset too small

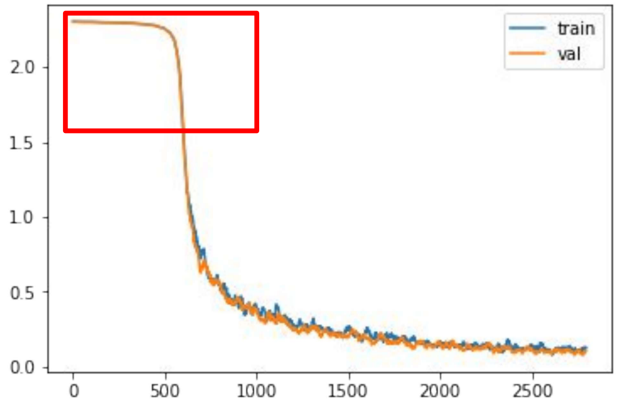


More extreme case of overfitting

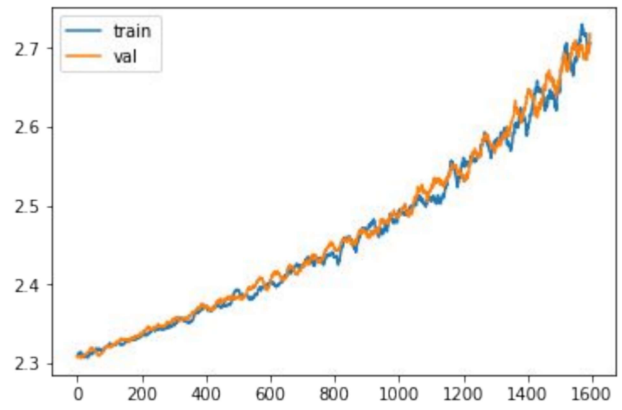


Not converged yet: need longer training

slow start

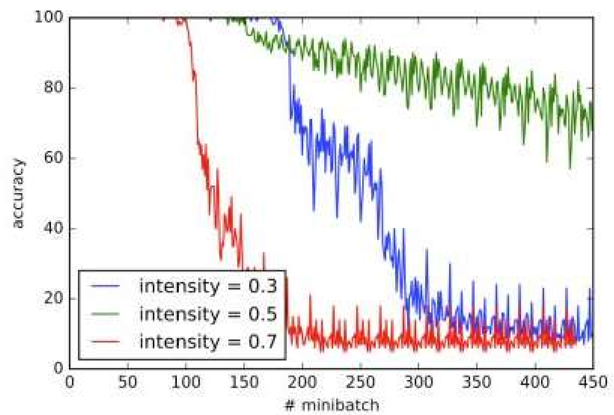


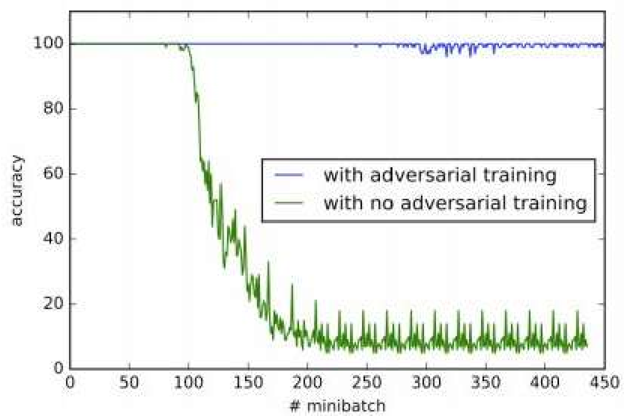
Slow start: initialization weights too small



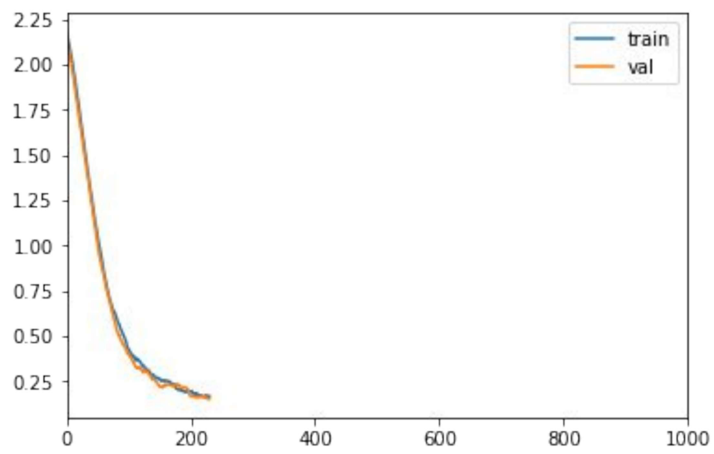
Applied the negative of gradients

Loss curves: more examples (or more epic fails)

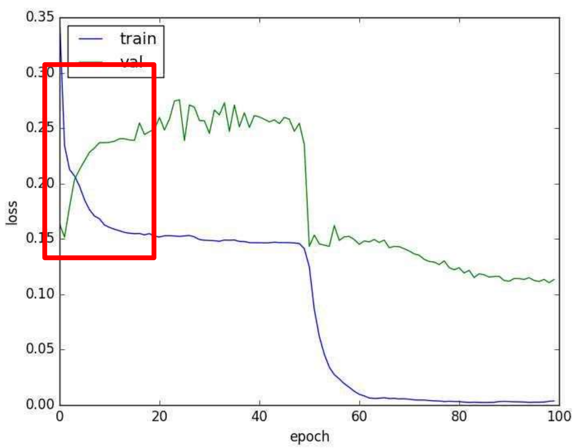




Problem: Not shuffling data, periodical patterns in loss curve

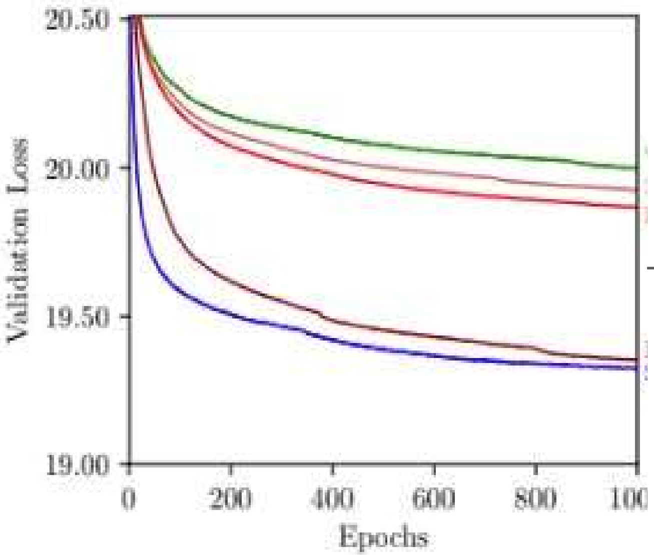


Get nans in the loss after a number of iterations: caused by numerical instability in models

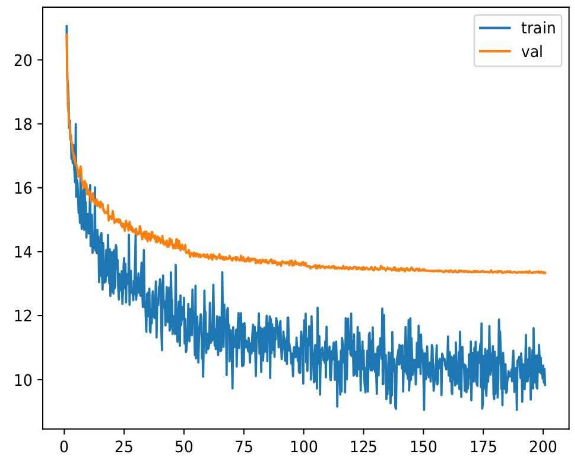


Problem: val set too small, statistics not meaningful

Loss curves: more examples



Problem: applied nonlinear activation before softmax



Make sure to optimize correct loss

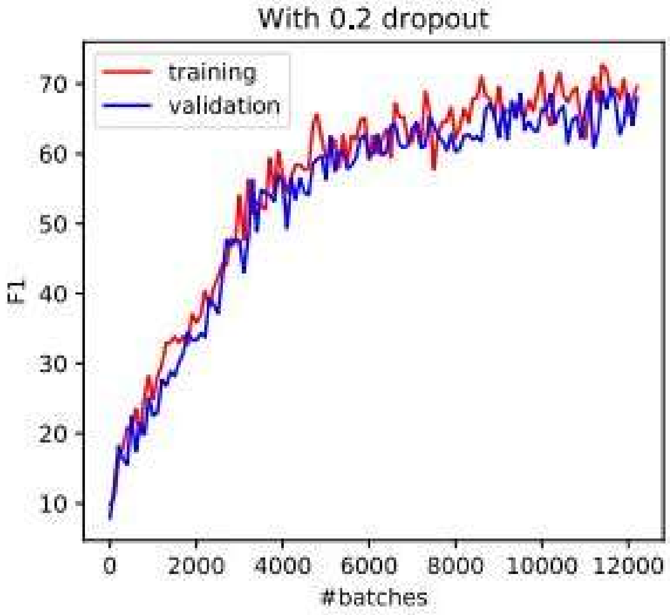
$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{k} e^{z_k}} \; for \; j = 1, \ldots, k$$

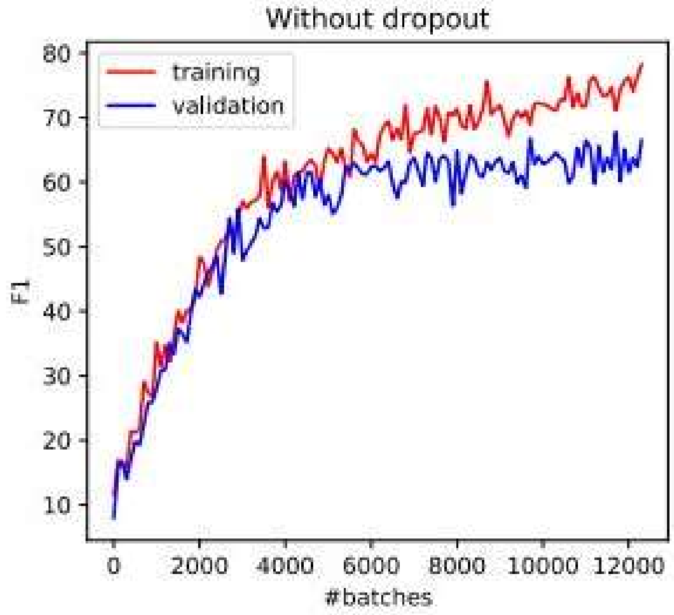
Summary

- Loss curve is one powerful indicator when debugging NNs
- Abnormal loss curves can be caused by
  - Wrong implementation of data loading
  - Wrong implementation/choice of losses
  - Optimizer problems
  - Suboptimal hyper-parameters
- Some back of the envelope calculation can help:
  - What do you expect the loss to start at?
  - What do you expect the loss to converge to?
  - Any ideas of how many iterations are required?

Regularization

Dropout, weight decay, use a smaller model

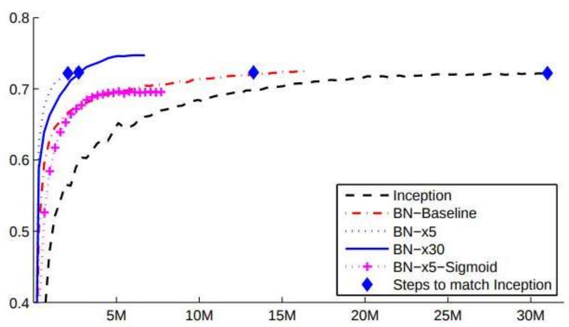




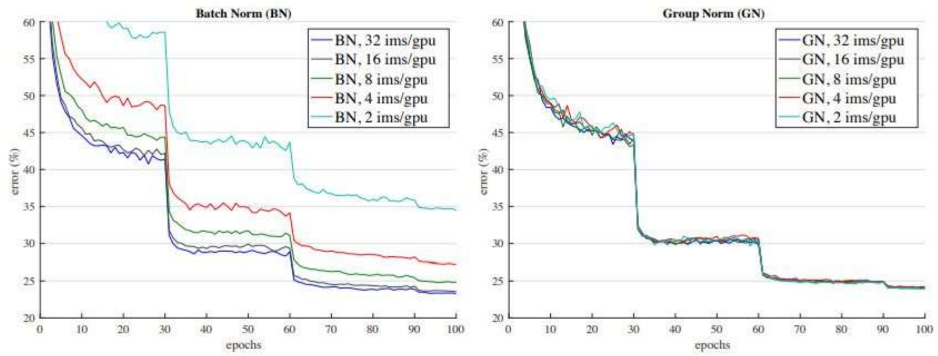
Tip: try overfit first, then try to close the gap between train and val.

Normalization

- You will explore in PS2
- Trains faster
- Robust to initializations
- Remember to do change to evaluation mode at testing time `model.eval()`



Batch normalization



Group normalization

Generative models

- Looking at loss curves is still insightful
- Looking at visual output for pathological issues
  - Grid patterns
  - Blurry output
  - Inductive bias
  - ...

Examples:

Generative models: useful references

https://github.com/soumith/ganhacks

https://distill.pub/2016/deconv-checkerboard/

Live code

Let’s try training a neural network on a chest xray dataset!