CORNELL TECH

Deep Learning Clinic
(DLC)

Lecture 6
Tricks and Tips on Training Neural Networks

Jin Sun

10/22/2019

Today - Tricks and Tips in DL Training

- **Overview**
- Data Preprocessing
- Network Details
- Training Dynamics

References:
http://cs231n.stanford.edu/syllabus.html

https://towardsdatascience.com/a-bunch-of-tips-and-tricks-for-training-deep-neural-networks-3ca24c31ddc8

http://karpathy.github.io/2019/04/25/recipe/

Overview

Training a neural network is an art!

But there are tricks people found useful.

Why Training Neural Networks Are Hard

“

It is allegedly easy to get started with training neural nets. Numerous libraries and frameworks take pride in displaying 30-line miracle snippets that solve your data problems, giving the (false) impression that this stuff is plug and play. It’s common see things like:

```
>>> your_data = # plug your awesome dataset here

>>> model = SuperCrossValidator(SuperDuper.fit, your_data, ResNet50, SGDOptimizer)

  # conquer world here
```

”

But it is never this easy!

http://karpathy.github.io/2019/04/25/recipe/

Why Training Neural Networks Are Hard

Neural net training fails silently:

Everything could be correct syntactically but the whole thing isn't arranged properly, and it's really hard to tell.

For example:

Forgot to flip the labels when you flipped the images for input

Temporal prediction takes the target as input

Loaded a pretrained model but didn't use the original mean

...

http://karpathy.github.io/2019/04/25/recipe/

A Recipe for Training Neural Networks

1. Inspect and understand your data
2. Set up training/evaluation + some simple baselines
   a. Get some simple model (e.g., a linear classifier or a tiny ConvNet)
   b. Train it, visualize results, and evaluate metrics
   c. Ablation studies
3. Get a model that is large enough that it can overfit (to training loss)
4. Regularize the model to gain more validation accuracy
5. Tune and get more from your model
   a. Hyperparameter search
   b. Ensemble models
   c. Train longer!

http://karpathy.github.io/2019/04/25/recipe/

Recap: Training A Network

Training Loop:

1. Sample a **batch** of data.
2. **Forward** pass the data through the network and calculate the loss.
3. **Backprop** the loss to calculate gradients of the network.
4. **Update** parameters using gradient based optimization method.

Overview

Network Details

Activation, preprocessing, initialization, regularization

Training Dynamics

Monitor the changing of train/val loss, parameter updates, hyperparameters

Today - Tricks and Tips in DL Training

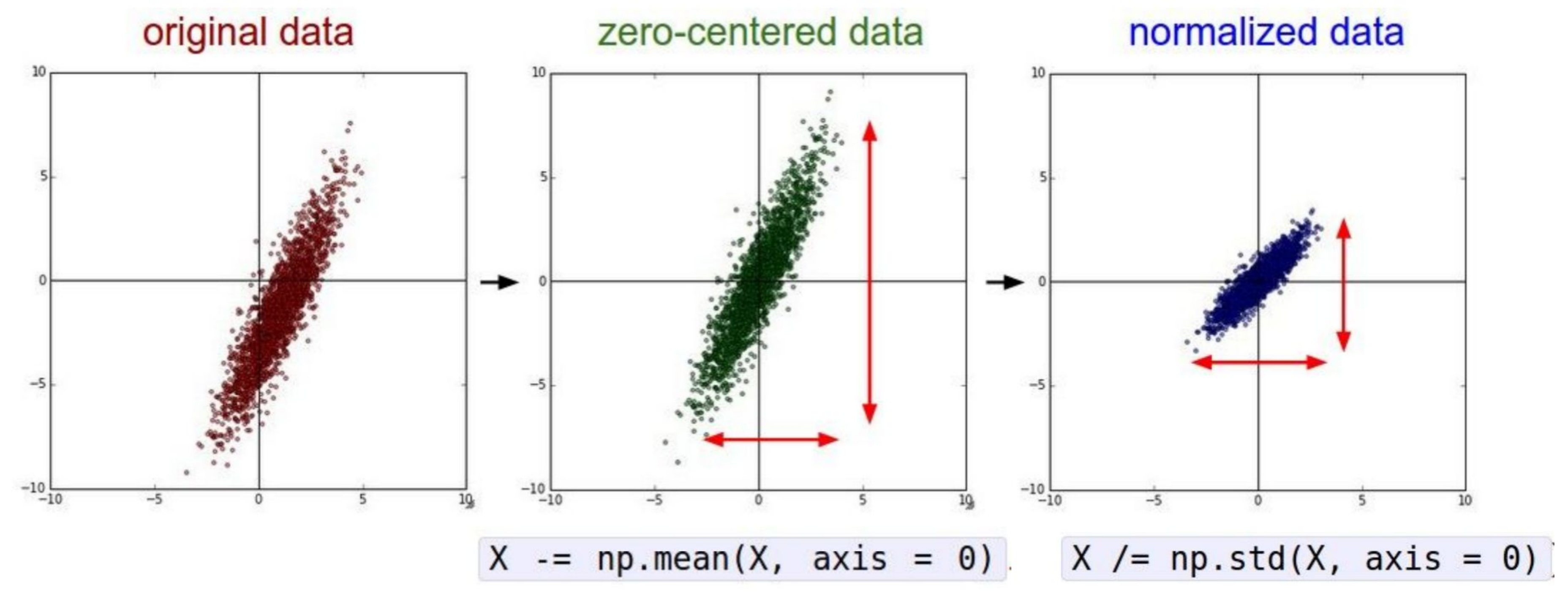
- Overview
- **Data Preprocessing**
- Network Details
- Training Dynamics

References:
http://cs231n.stanford.edu/syllabus.html

https://towardsdatascience.com/a-bunch-of-tips-and-tricks-for-training-deep-neural-networks-3ca24c31ddc8

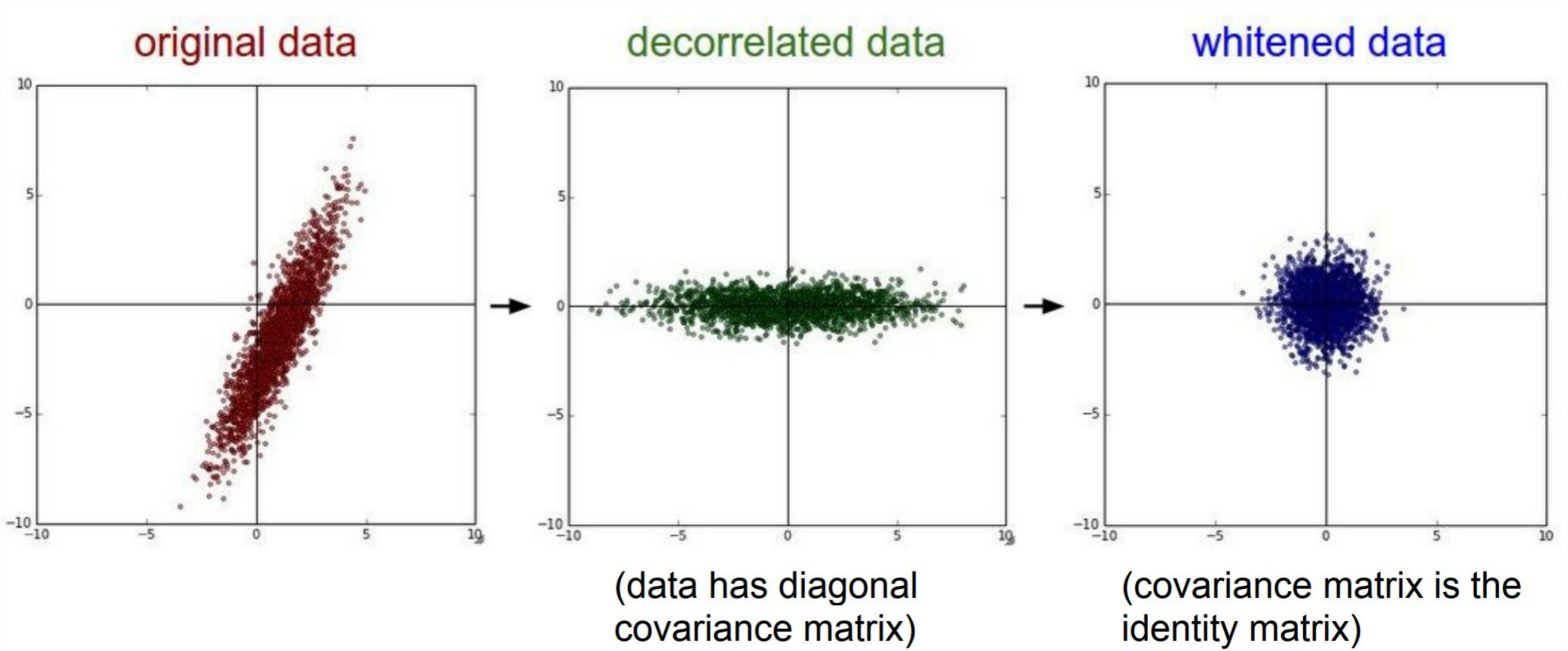
http://karpathy.github.io/2019/04/25/recipe/

Data Preprocessing



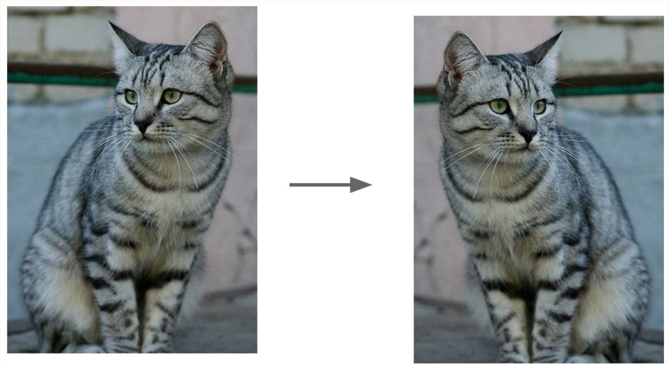
In practice, only do zero-centering for images.

Data Preprocessing - PCA and Whitening

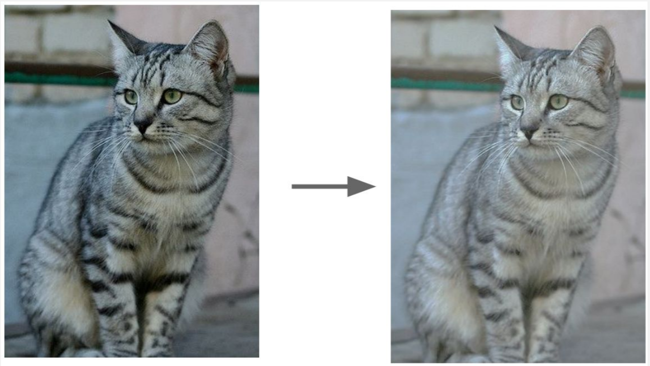


Data Augmentation

Horizontal Flip:

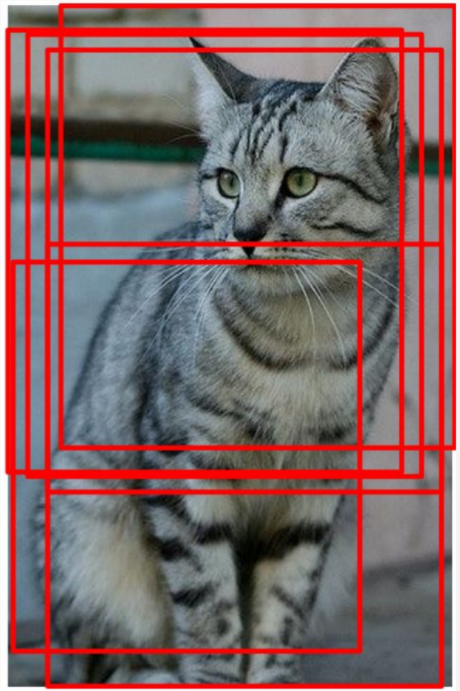


Random Contrast and Brightness

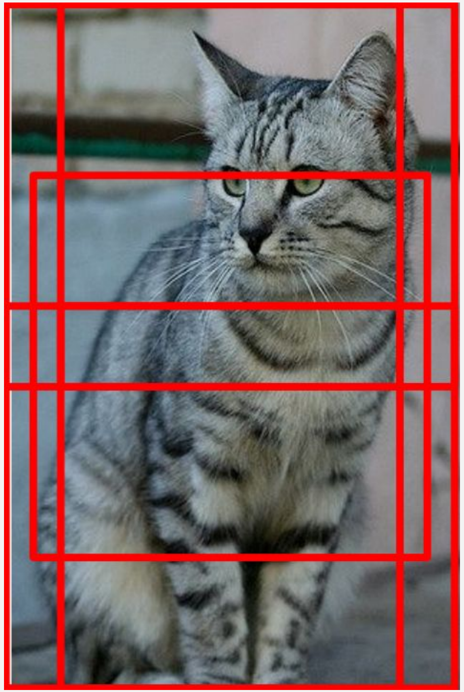


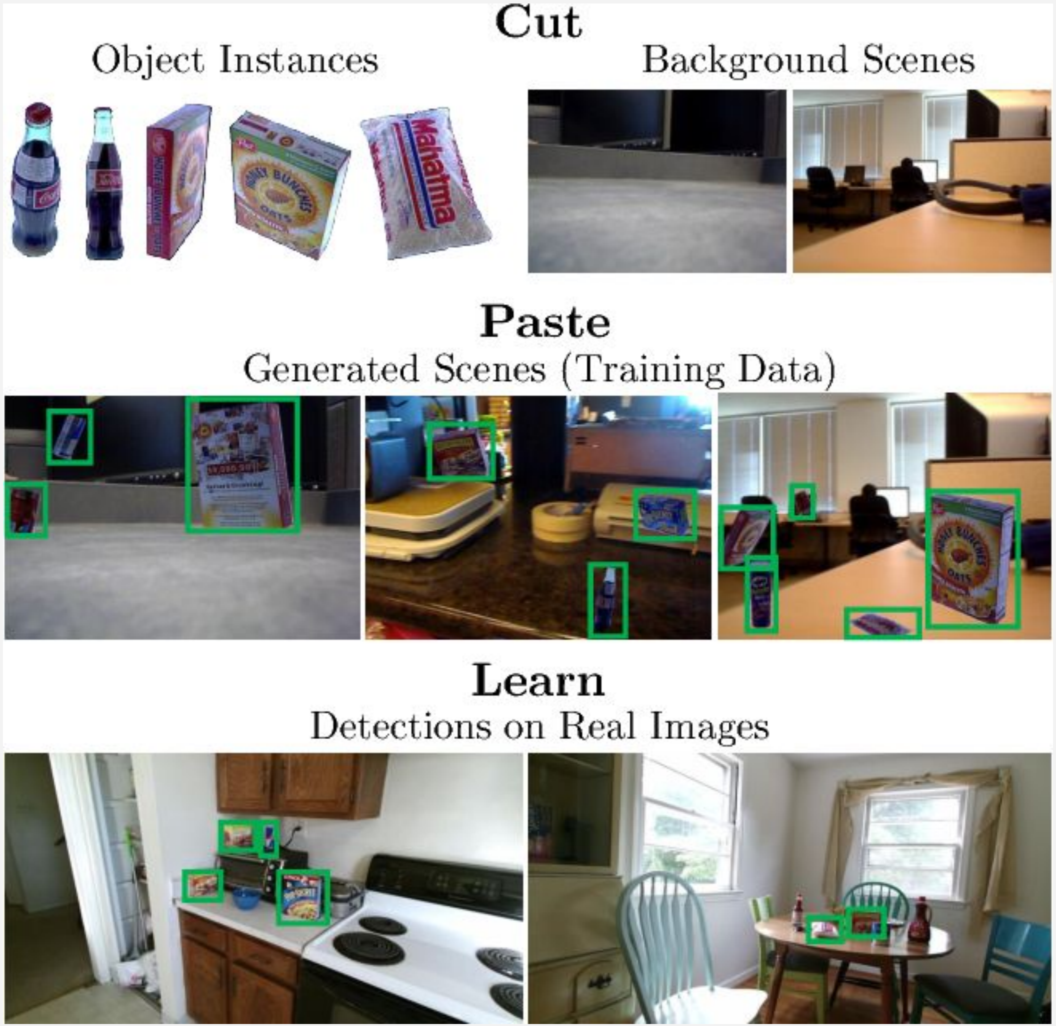
Data Augmentation

Random Crops and Scales (Train)



Fixed Crops and Scales (Test)





https://arxiv.org/abs/1708.01642

Data Shuffling (for non-temporal data)

```
DataLoader(dataset, batch_size=1, shuffle=False, sampler=None,
          batch_sampler=None, num_workers=0, collate_fn=None,
          pin_memory=False, drop_last=False, timeout=0,
          worker_init_fn=None)
```

The purpose is to avoid the network ‘remembers’ correlation of consecutive data samples.

We should be doing **Stochastic** Gradient Descent.

Imbalanced Data

Class weights can be added to MSE or CrossEntropy

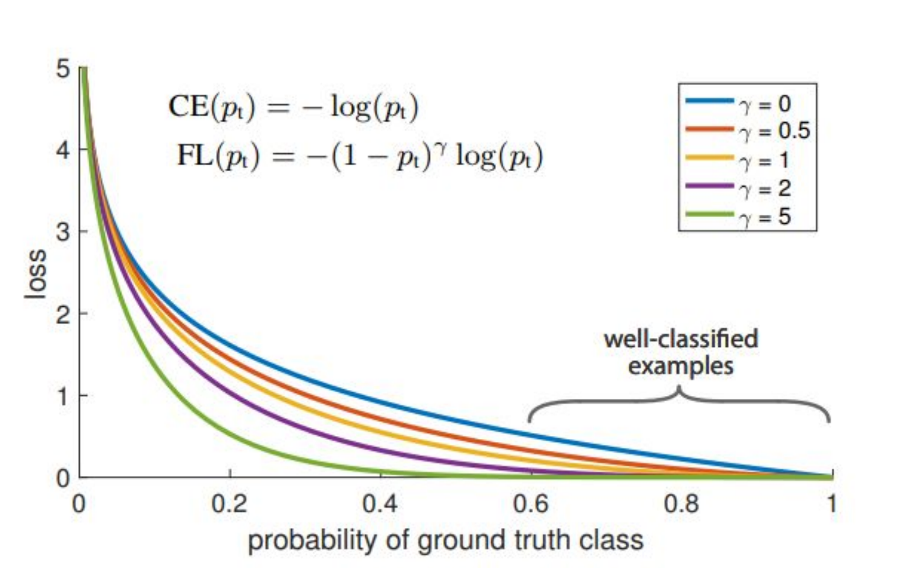
A common method for addressing class imbalance is to introduce a weighting factor $\alpha \in [0, 1]$ for class 1 and $1 - \alpha$ for class -1. In practice α may be set by inverse class frequency or treated as a hyperparameter to set by cross validation. For notational convenience, we define α_t analogously to how we defined p_t. We write the α-balanced CE loss as:

$$\mathrm{CE}(p_t) = -\alpha_t \log(p_t). \tag{3}$$

This loss is a simple extension to CE that we consider as an experimental baseline for our proposed focal loss.

https://arxiv.org/pdf/1708.02002.pdf

Imbalanced Data - Focal Loss



Lin, Tsung-Yi, et al. "Focal loss for dense object detection." *Proceedings of the IEEE international conference on computer vision*. 2017.
https://arxiv.org/pdf/1708.02002.pdf

Imbalanced Data - Resampling

Undersampling

Remove samples from the majority class

Oversampling

‘Get’ more data from the minority class

Synthetic Minority Over-sampling Technique (**SMOTE**):

Create synthetic data points by sampling in the feature space among k-nearest neighbors of a real data point.

Today - Tricks and Tips in DL Training

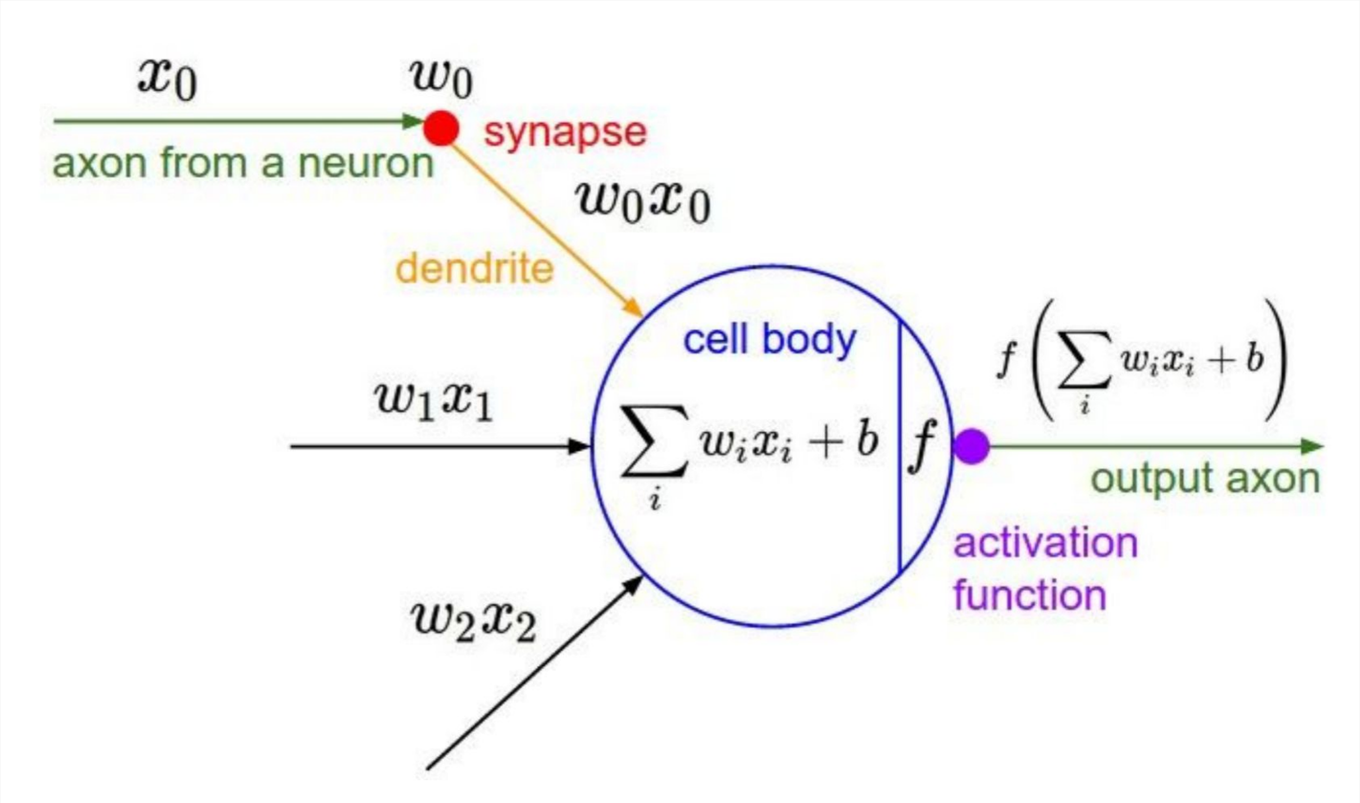
- Overview
- Data Preprocessing
- **Network Details**
- Training Dynamics

References:
http://cs231n.stanford.edu/syllabus.html

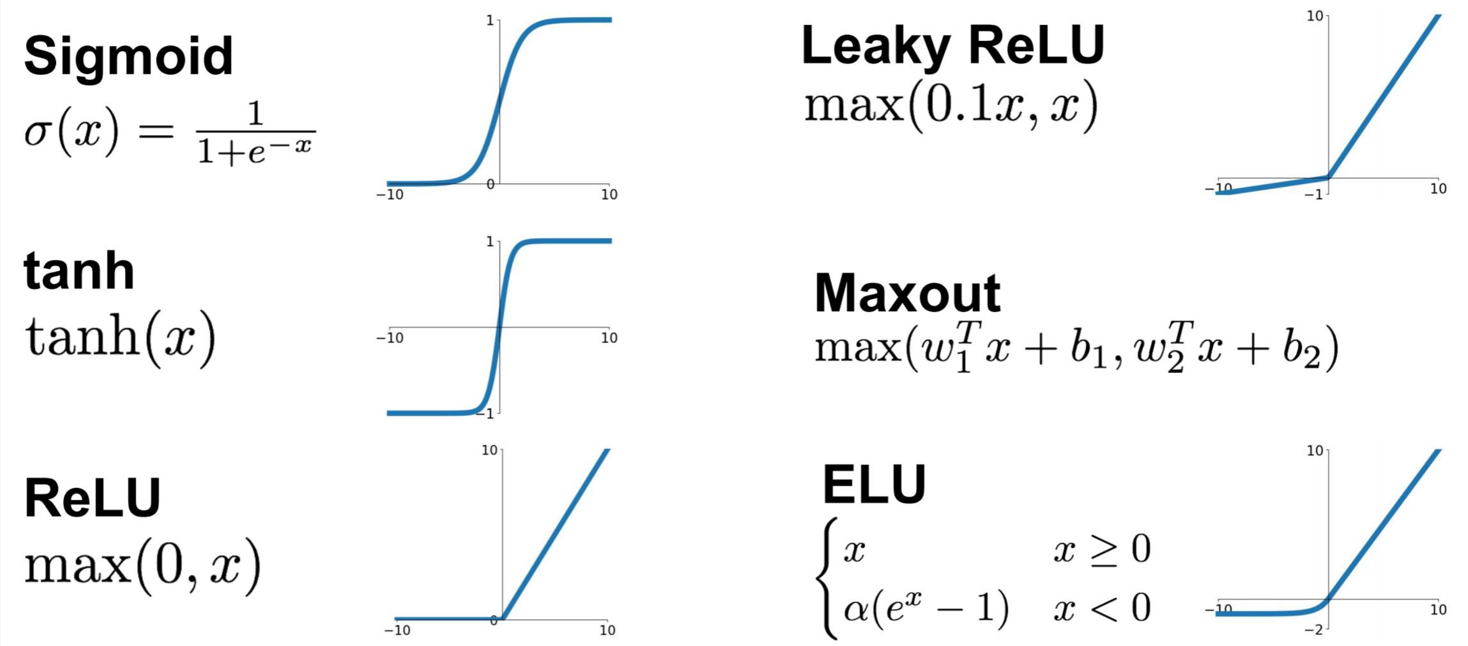
https://towardsdatascience.com/a-bunch-of-tips-and-tricks-for-training-deep-neural-networks-3ca24c31ddc8

http://karpathy.github.io/2019/04/25/recipe/

Activation Functions



Activation Functions - the Gallery



Activation Functions - Summary

Use **ReLU** as the **default**

Try Leaky ReLU / Maxout / ELU for alternatives

Try tanh

Sigmoid is usually not recommended.

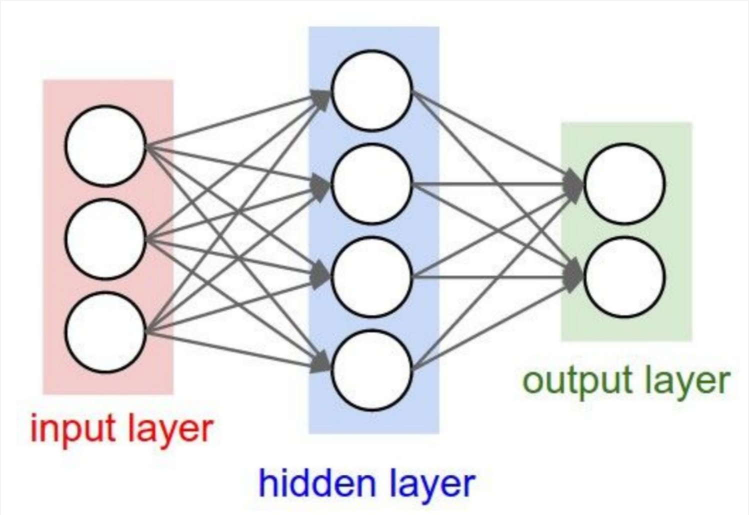
Weight Initialization

Do not use constant weights.

Small Gaussian random numbers not working well for deep networks..

Recommended:

Xavier initialization, and its variants.



Still an active research area.

Xavier initialization

Uniform Xavier initialization: draw each weight, w, from a random uniform distribution

in [-x,x] for $x = \sqrt{\frac{6}{inputs + outputs}}$

Normal Xavier initialization: draw each weight, w, from a normal distribution with a mean

of 0, and a standard deviation $\sigma = \sqrt{\frac{2}{inputs + outputs}}$

https://www.youtube.com/watch?v=OAb_p-SXSeM

365√DataScience

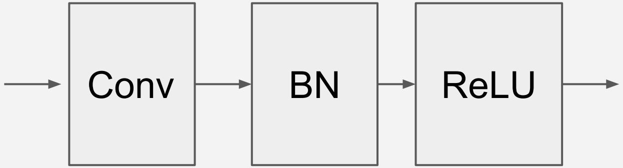
Batch Normalization

Make a batch of activations to have zero-mean and unit-variance.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Works as a layer in neural networks.

Put it after fully connected or convolutional layers, before nonlinearity.



Batch Normalization

Benefit:

Improve gradient flow in the network.

Allows higher learning rate.

Works also like a regularizer.

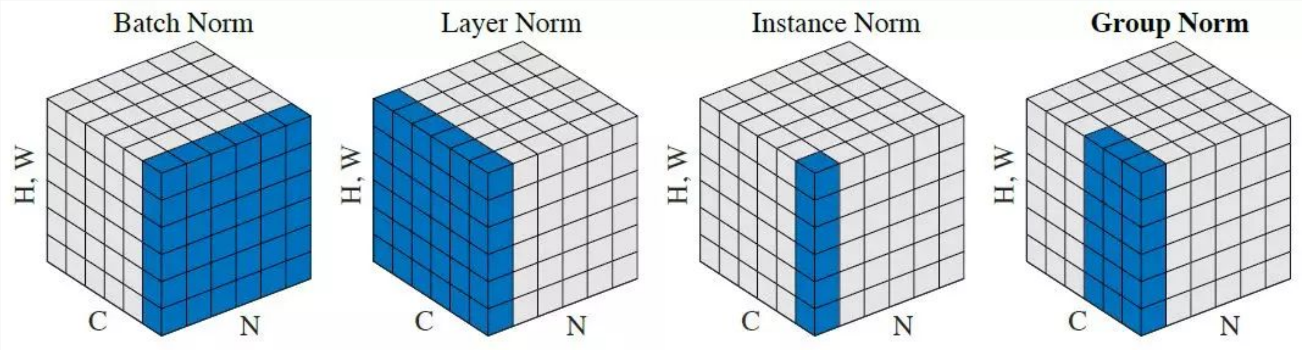
During Testing:

Use a single fixed mean/std obtained from training.

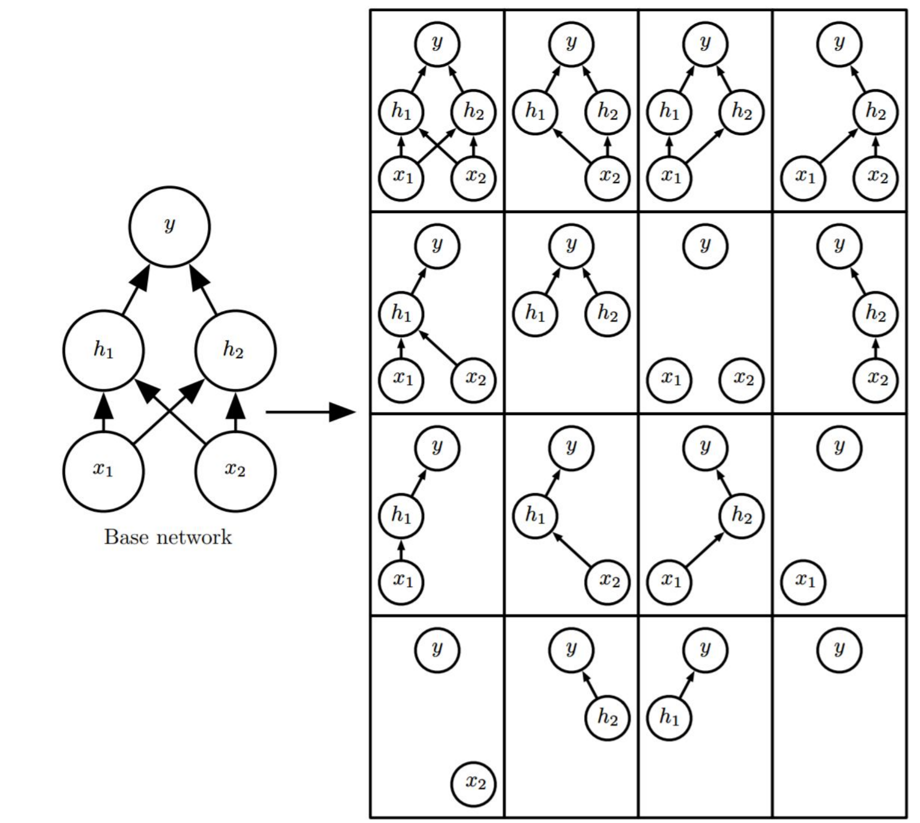
Group Normalization and Instance Normalization

BatchNorm works poorly when batch size is small.

https://arxiv.org/abs/1803.08494

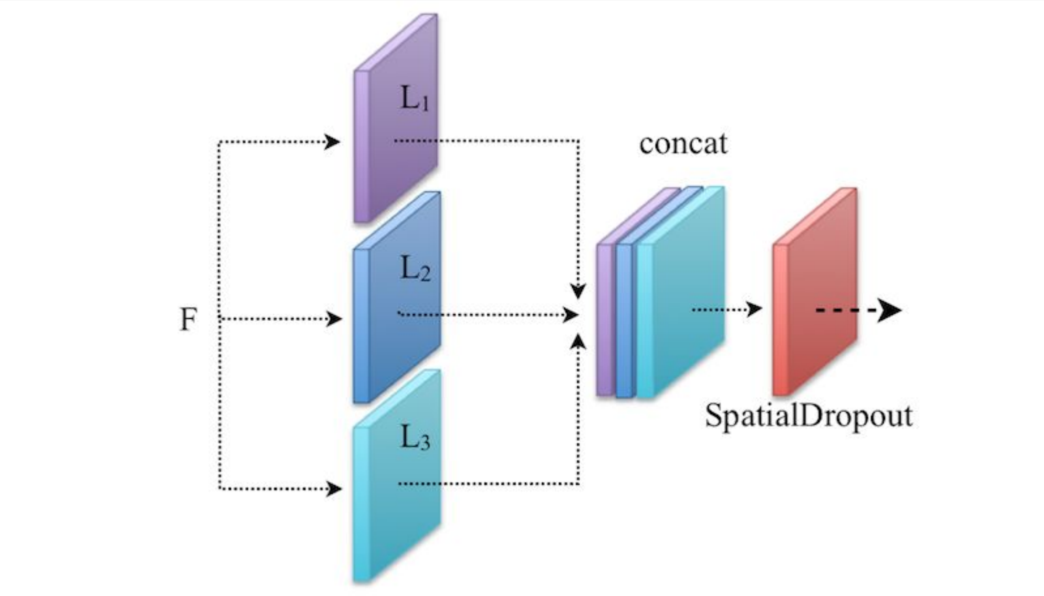


Dropout

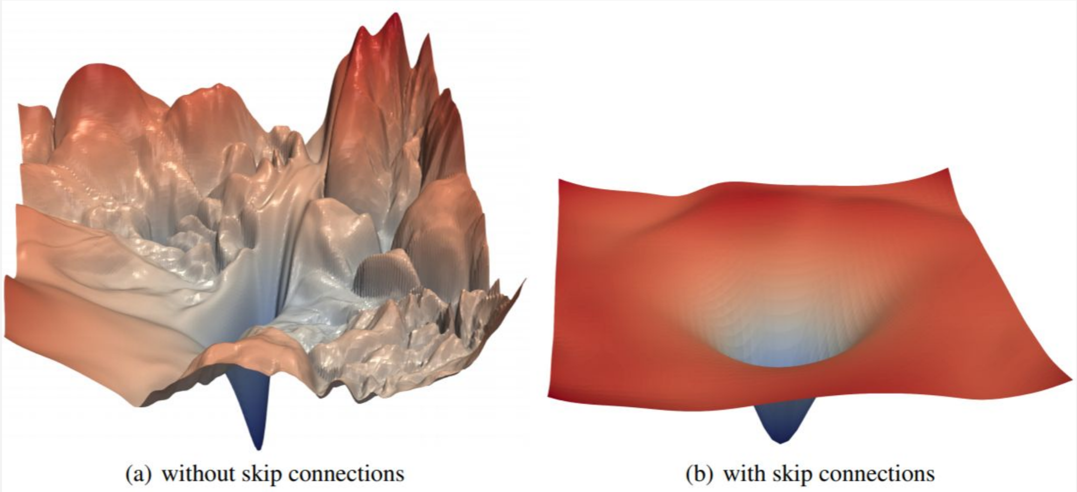


Spatial Dropout

For images, remove the whole channel



Network Structure Affects Optimization



Visualizing the Loss Landscape of Neural Nets
https://arxiv.org/pdf/1712.09913.pdf

Choosing architecture with various layers

Do hyper-parameter search on network with less layers

For example ResNet-50

Then choose the deeper version once you have a good set of parameters

For example ResNet-152

Order of computation matters

Use Max-pooling before ReLU to save some computations. Since ReLU thresholds the values with zero: `f(x)=max(0,x)` and Max-pooling pools only max activations: `f(x)=max(x1,x2,...,xi)`, use `Conv > MaxPool > ReLU` rather than `Conv > ReLU > MaxPool`.

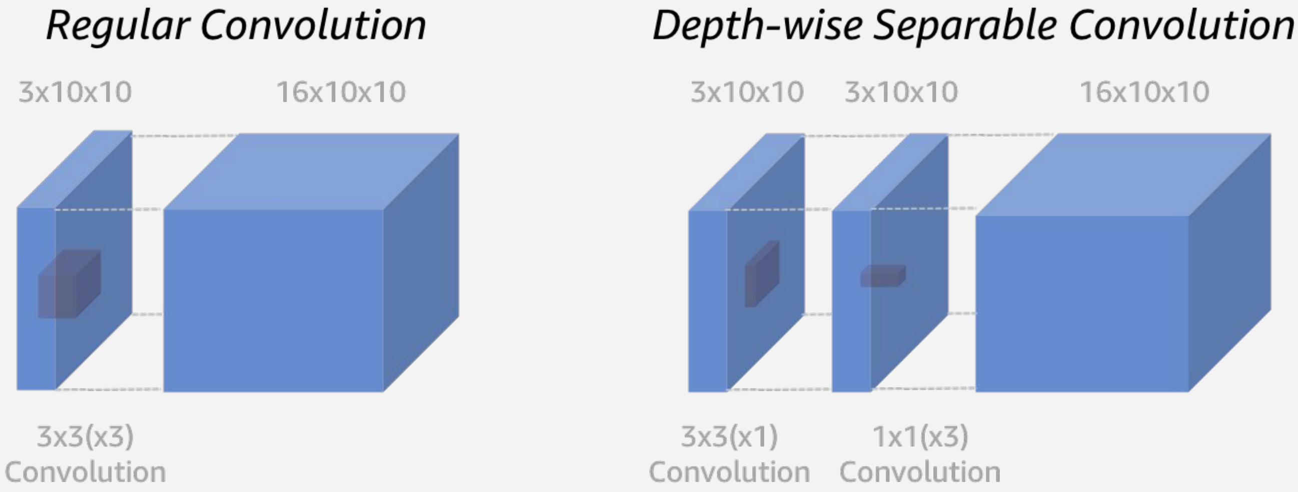
E.g. Assume that we have had two activations from `Conv` (i.e. 0.5 and -0.5):

- so `MaxPool > ReLU = max(0, max(0.5,-0.5)) = 0.5`
- and `ReLU > MaxPool = max(max(0,0.5), max(0,-0.5)) = 0.5`

See? the output from these two operations is still `0.5`. In this case, using `MaxPool > ReLU` can save us one `max` operation.

https://towardsdatascience.com/a-bunch-of-tips-and-tricks-for-training-deep-neural-networks-3ca24c31ddc8

Re-think about common operations



\# of params: 432
\# of computations: 43,200

\# of params: 27+48 = 75
\# of computations: 2,700+4,800 = 7,500

https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108

Today - Tricks and Tips in DL Training

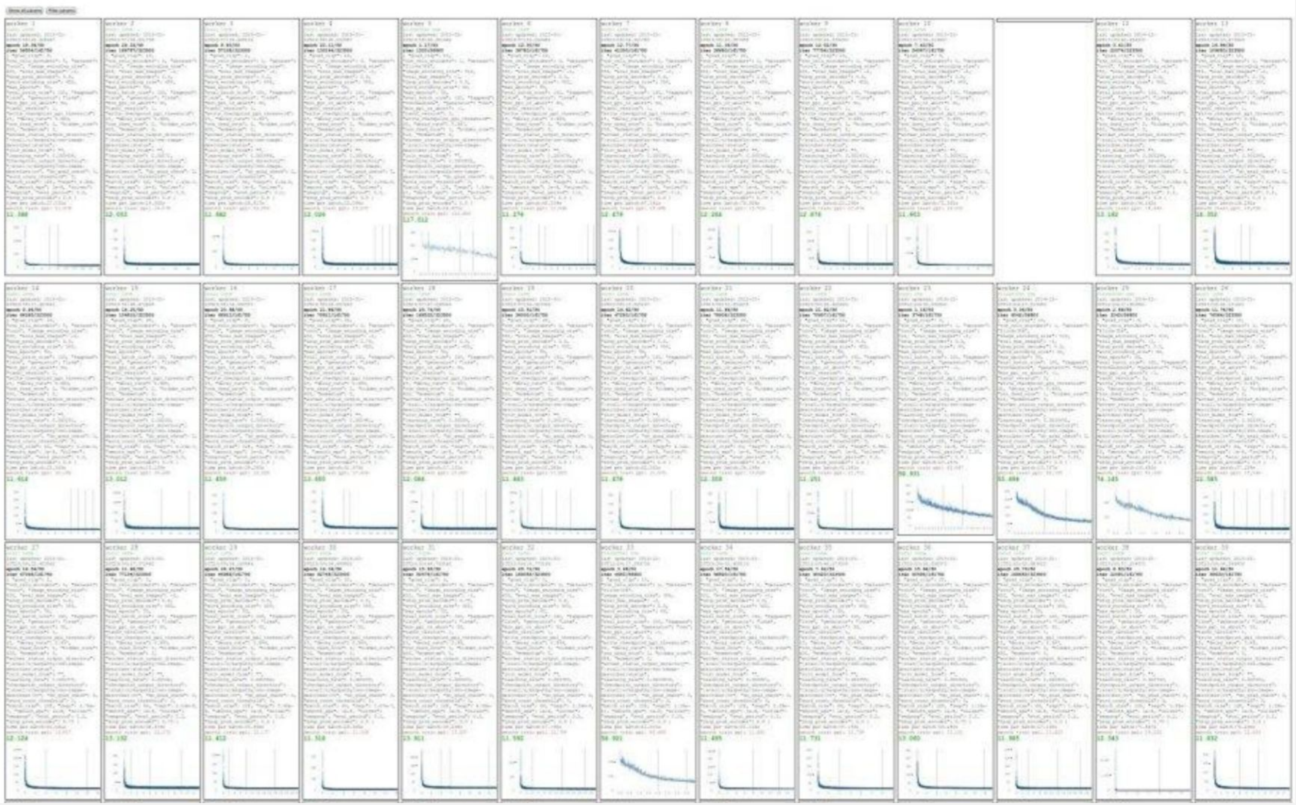
- Overview
- Data Preprocessing
- Network Details
- **Training Dynamics**

References:
http://cs231n.stanford.edu/syllabus.html

https://towardsdatascience.com/a-bunch-of-tips-and-tricks-for-training-deep-neural-networks-3ca24c31ddc8

http://karpathy.github.io/2019/04/25/recipe/

Hyperparameter Search and Training Curves



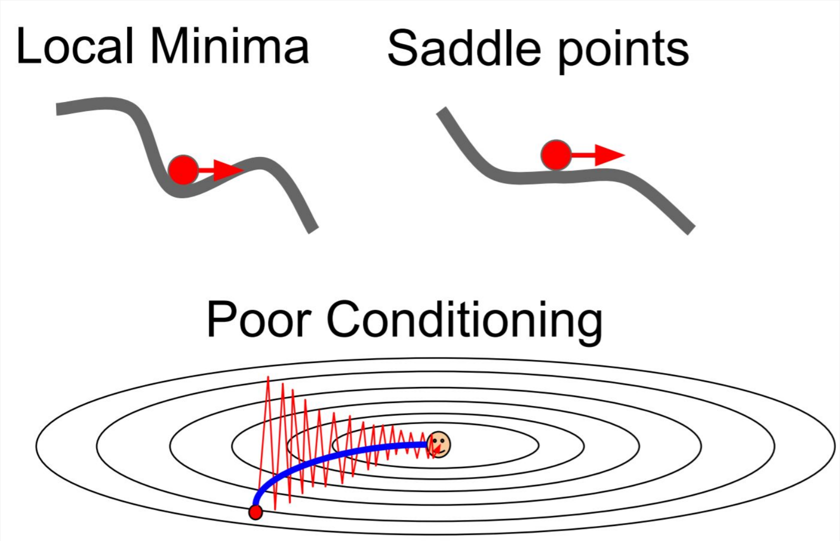
Optimization

Stochastic Gradient Descent (**SGD**)

Usually works with momentum.

Adagrad, **RMSProp**, **Adadelta**

Recommended first try: **Adam**



Overview of Gradient Descent Methods

http://ruder.io/optimizing-gradient-descent/index.html#visualizationofalgorithms

Choosing a different optimization method might make the training much slower/faster.

Sadly you may not know which is which before you try.

Monitoring the Learning Process

Some simple things to try and observe:

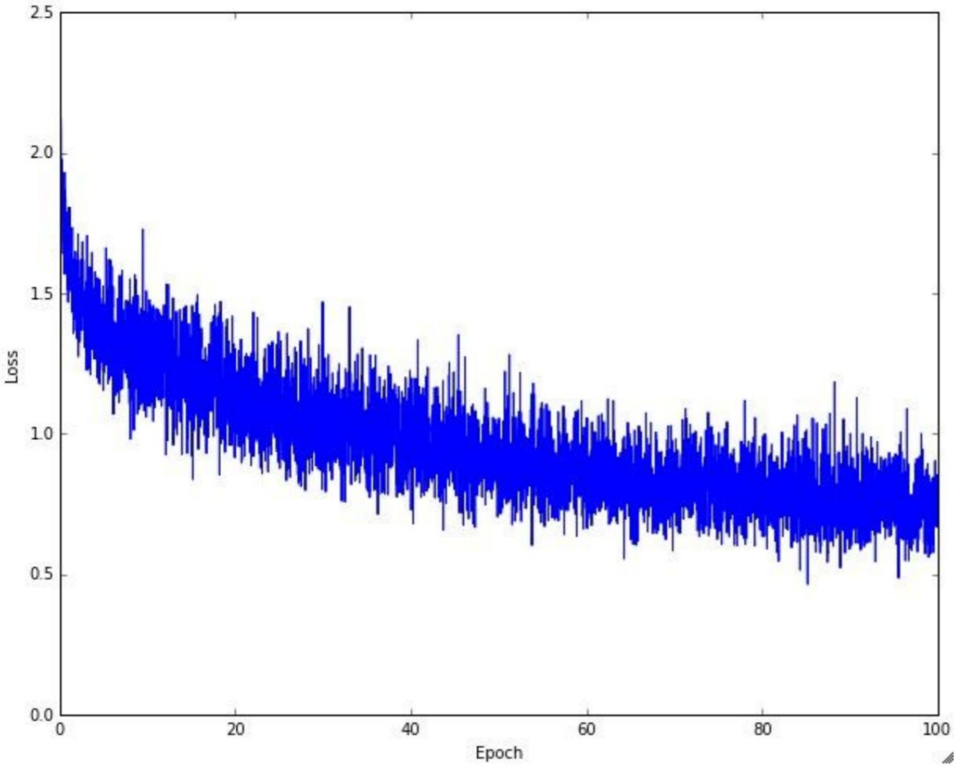
You should be able to overfit your model to a small portion of the data. Do this as a sanity check to make sure the pipeline works.

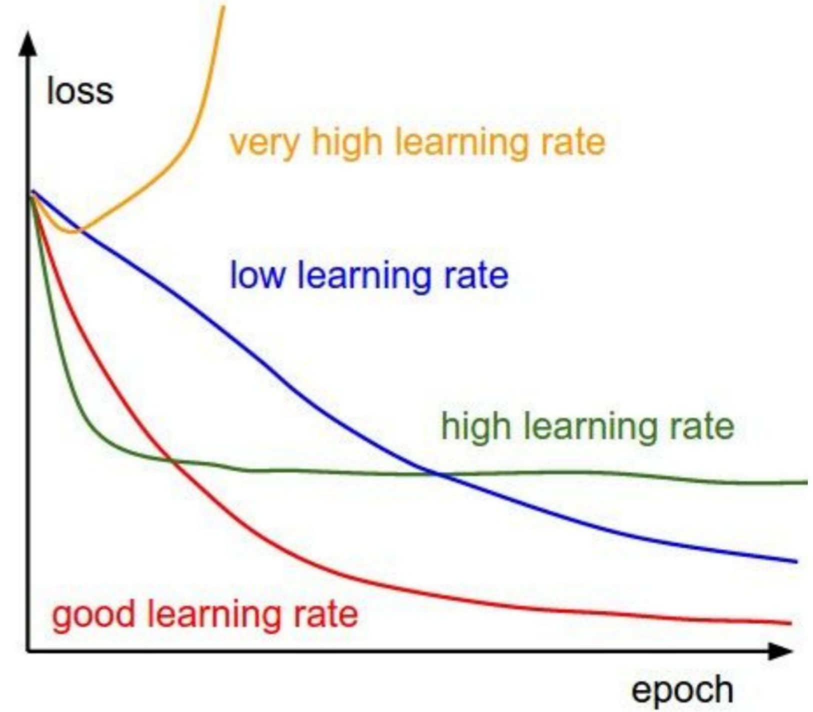
Train loss is not going down: try higher learning rate

Train loss explodes: try lower learning rate

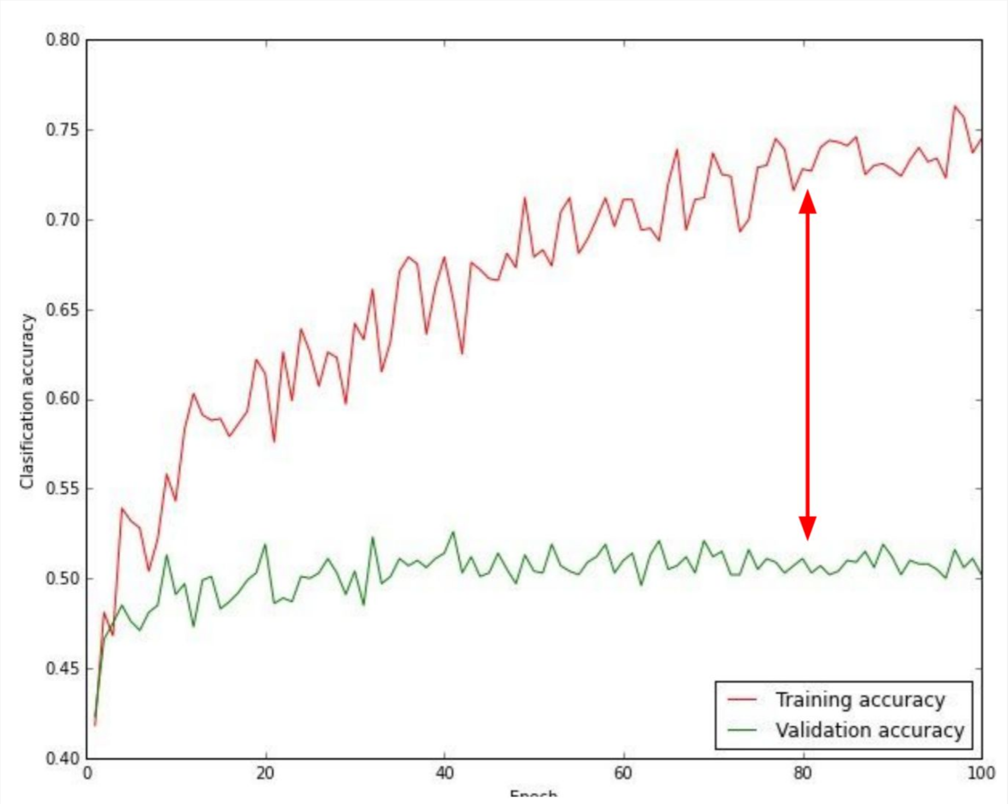
Find a range of learning rate by doing a simple set of experiments.

Learning Rate vs Curves





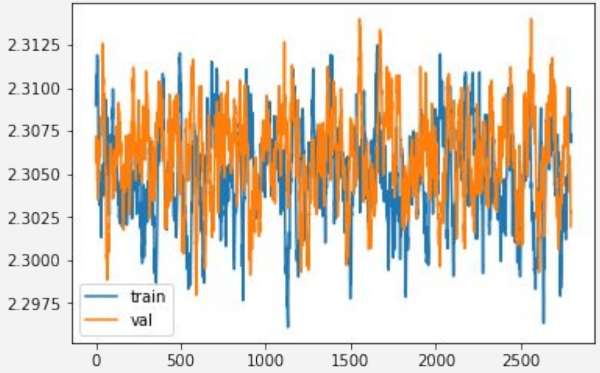
Sign of Overfitting



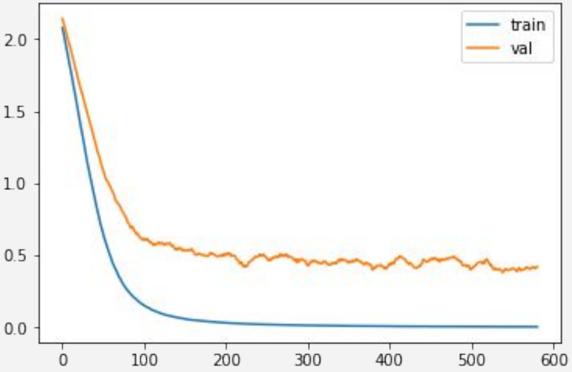
Big gap: Overfitting

No gap: Maybe model can be more complex

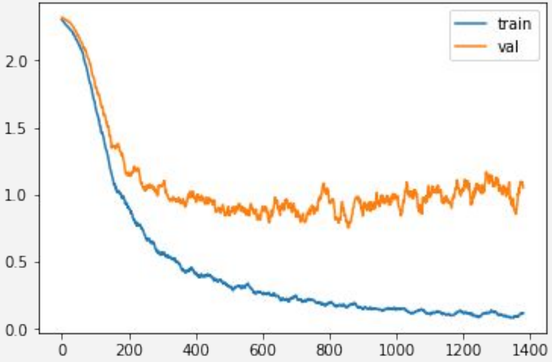
Early Stopping



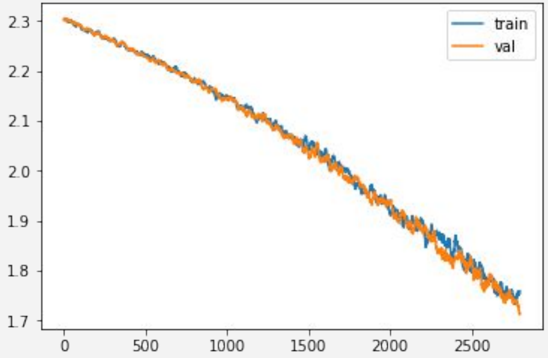
Not learning: gradients not applied to weights



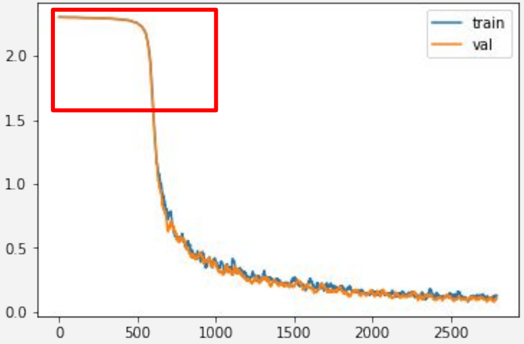
Overfit: model too large/dataset too small



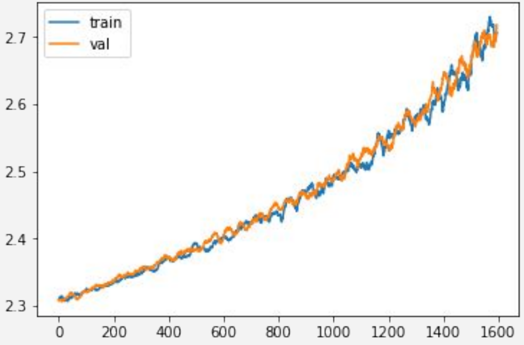
More extreme case of overfitting



Not converged yet: need longer training



Slow start: initialization not ideal



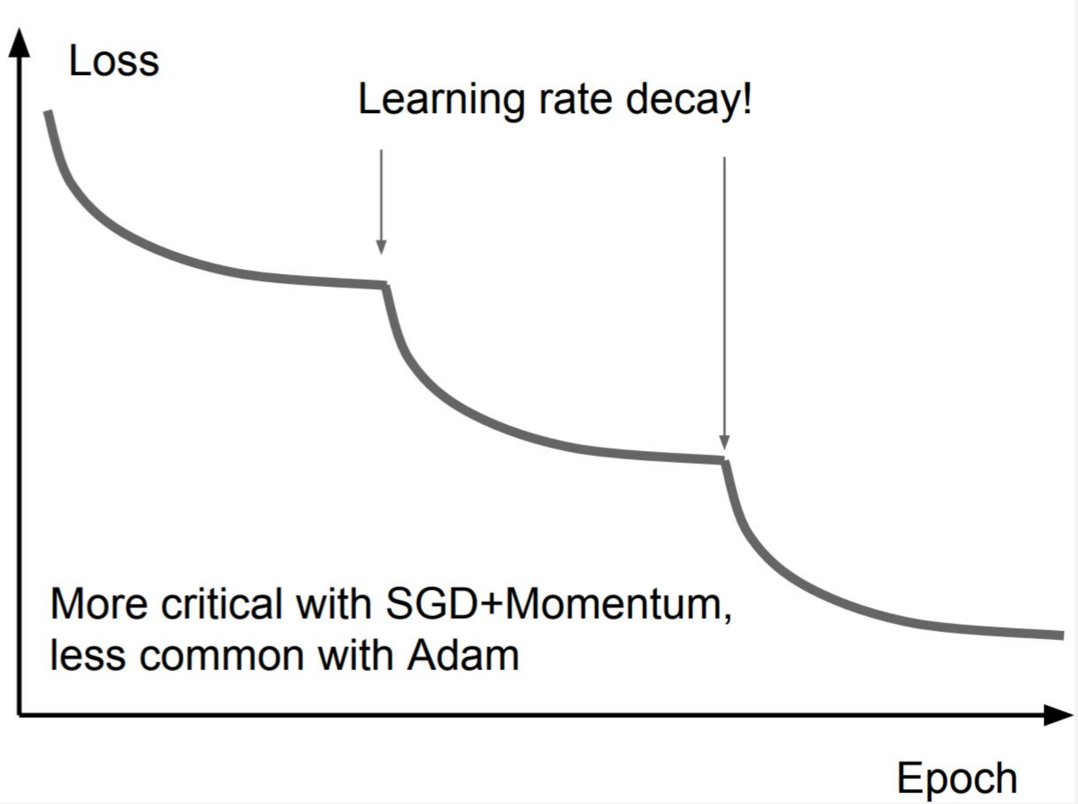
Applied the negative of gradients

Learning Rate Decay Strategy

Step Decay

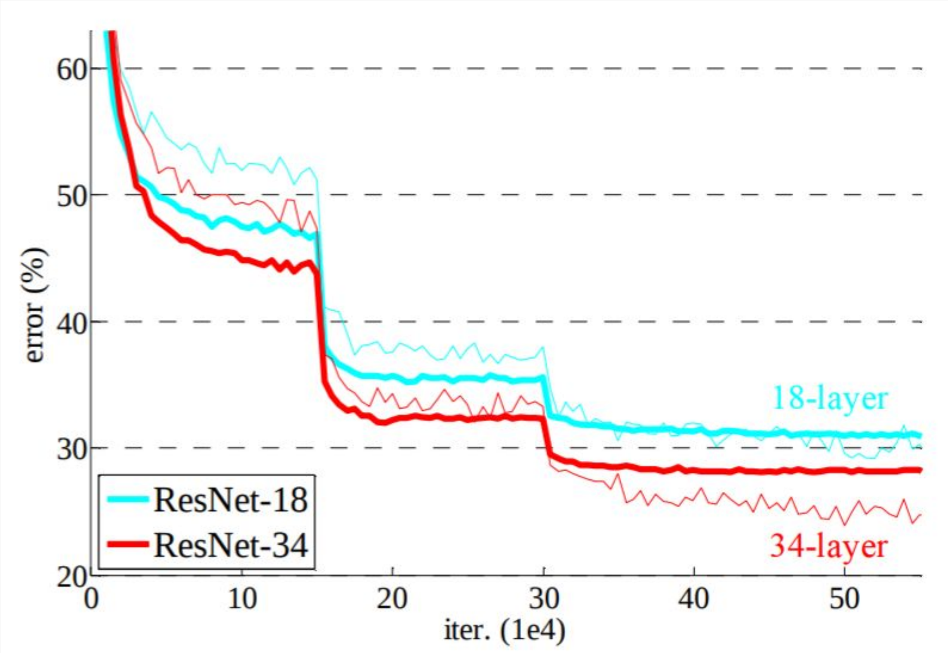
Exponential Decay

1/t Decay



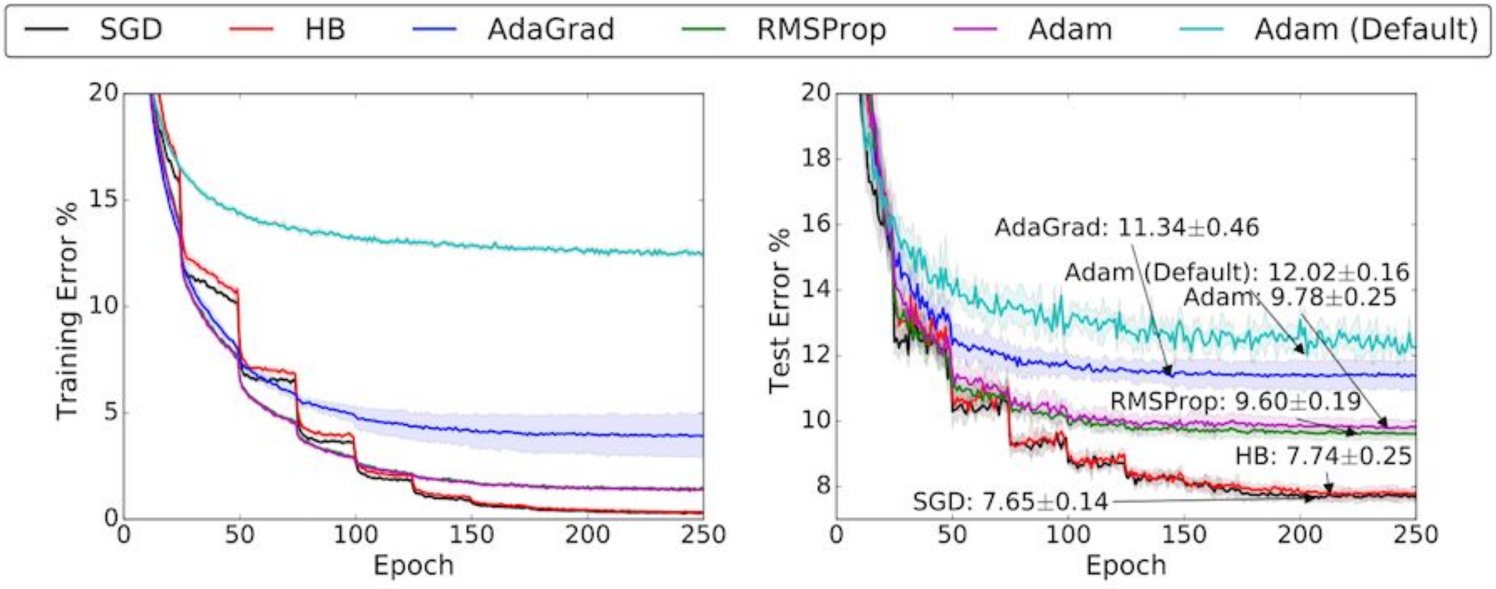
Example: ResNet Learning Rate Schedule

Start from 0.1, divide by 10 when the error plateaus



https://arxiv.org/pdf/1512.03385.pdf

SGD can works better than Adam with a good LR schedule



(a) CIFAR-10 (Train) **(b)** CIFAR-10 (Test)

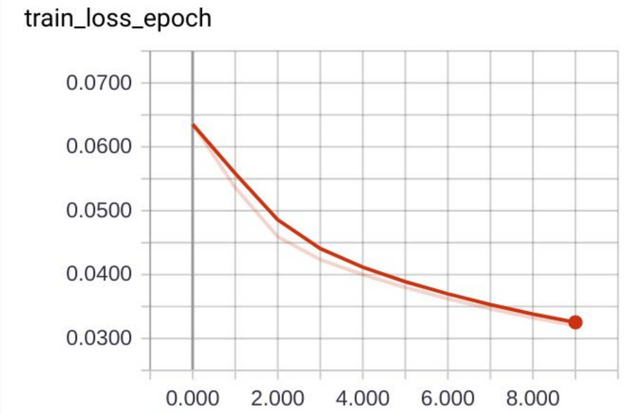
https://arxiv.org/abs/1705.08292

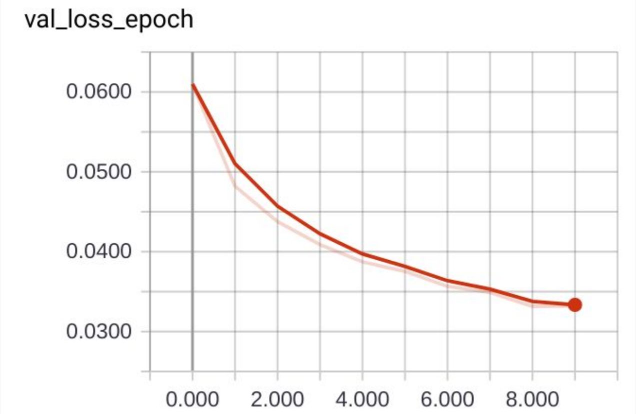
CIFAR10 Example

Based on
https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

Reasonable Training Curves

Note training and validation losses are in roughly same scale.

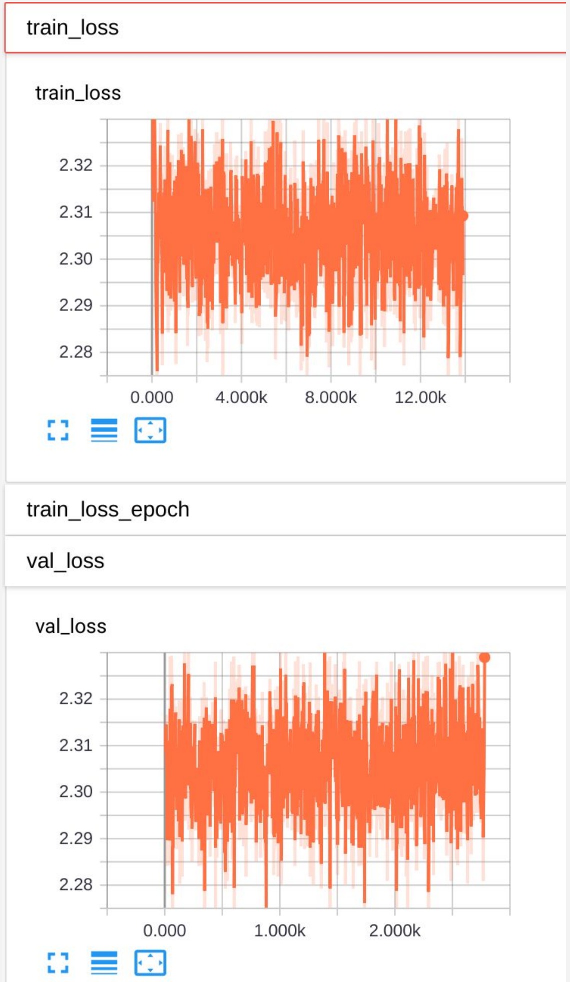




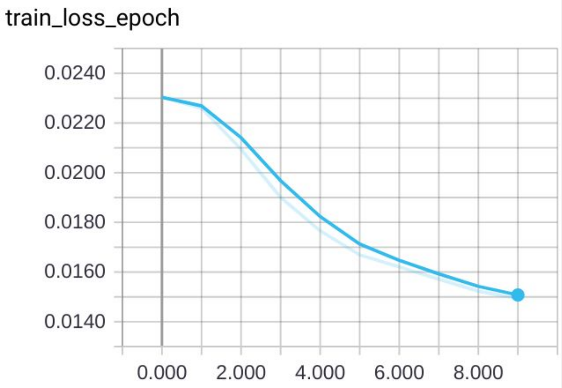
No Gradient Updated

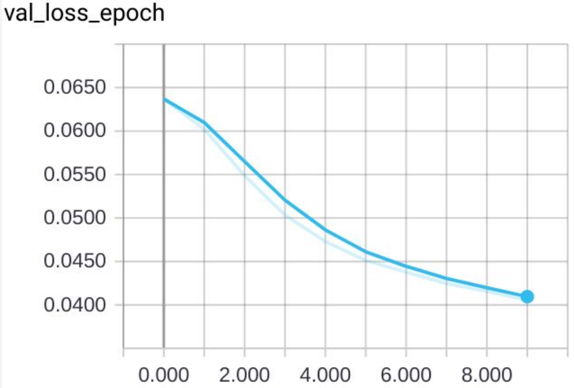
Don't forget to step your optimizer!

```
# get the inputs
inputs, labels = data
inputs = inputs.to(device)
labels = labels.to(device)

# zero the parameter gradients
optimizer.zero_grad()

# forward + backward + optimize
outputs = net(inputs)
loss = criterion(outputs, labels)
loss.backward()
# optimizer.step()
```



Overfitting

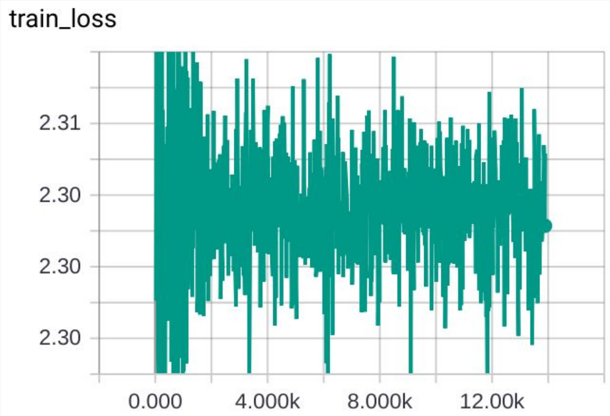


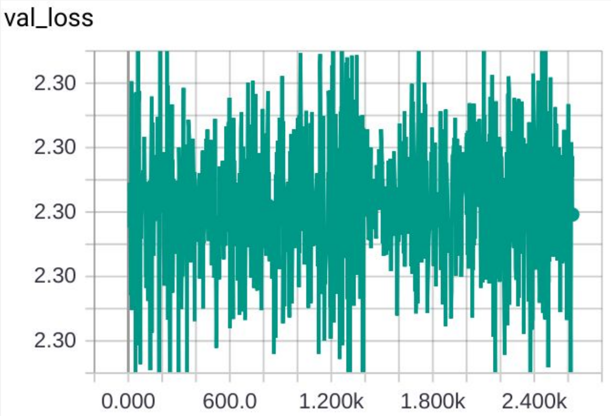



```python
for epoch in range(10):  # loop over the dataset multiple times
    # Train the model
    total_step = len(trainloader)
    net.train()
    train_loss_epoch = 0.0
    for i, data in enumerate(trainloader, 0):
        if i > 500:
            break
```

Using about 50% of training data

Bad Initialization

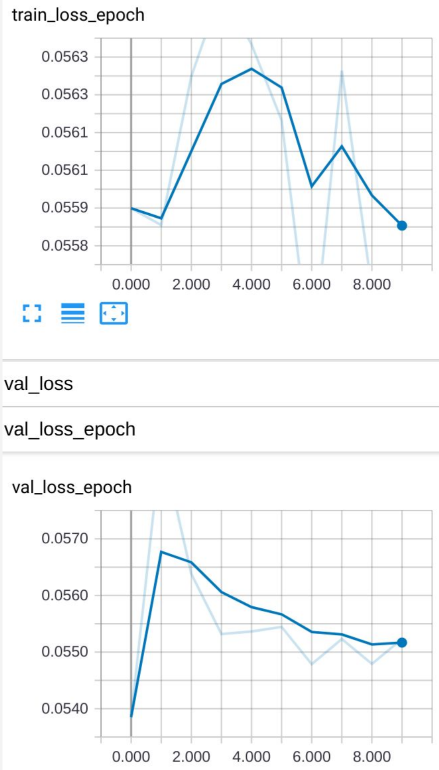




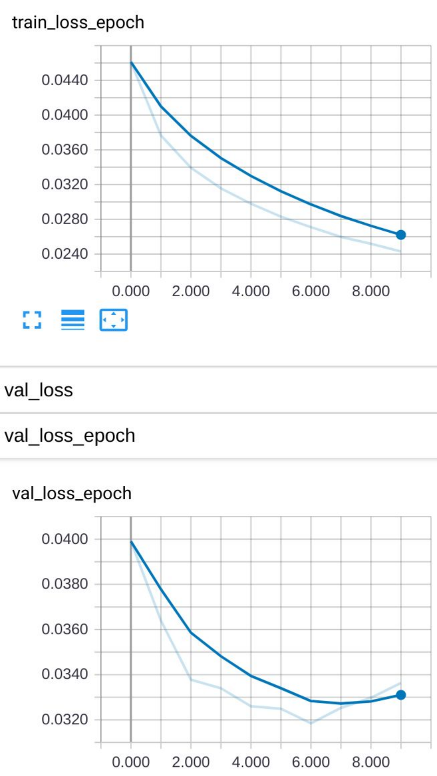

```
net = Net().to(device)

# initialization
for m in net.modules():
    if isinstance(m, nn.Conv2d):
        # nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu'
)
        nn.init.constant_(m.weight, 0.0)
        nn.init.constant_(m.bias, 0.0)
```

Effect of Learning Rate

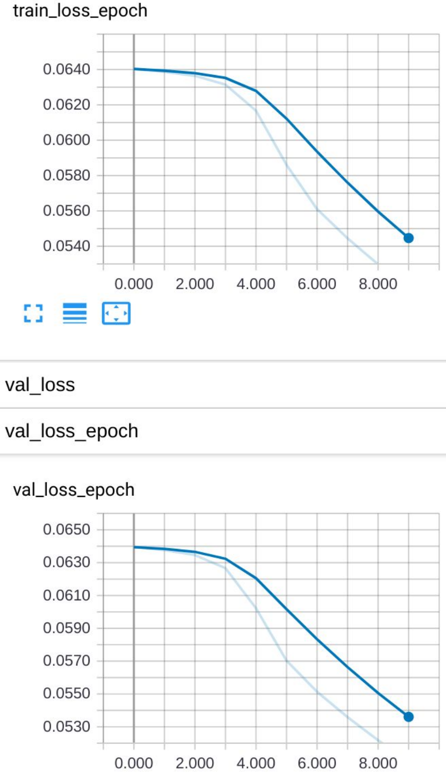
LR = 0.1



LR = 0.01



LR = 0.0001



Learning Rate Scheduler In Pytorch

https://pytorch.org/docs/master/optim.html#how-to-adjust-learning-rate

```
>>> # Assuming optimizer uses lr = 0.05 for all groups
>>> # lr = 0.05     if epoch < 30
>>> # lr = 0.005    if 30 <= epoch < 60
>>> # lr = 0.0005   if 60 <= epoch < 90
>>> # ...
>>> scheduler = StepLR(optimizer, step_size=30, gamma=0.1)
>>> for epoch in range(100):
>>>     scheduler.step()
>>>     train(...)
>>>     validate(...)
```

Try a lot of task-dependent tricks if you really want to push the limit

Example - FastAI: Train ImageNet In 18 Minutes

Link: https://www.fast.ai/2018/08/10/fastai-diu-imagenet/

Tricks:

- Rectangular crops of input images
- Progressive resizing: train on small images first, then increase image size
- Dynamic batch size

Result:

"...we got a training time of 18 minutes on 16 AWS instances, at a total compute cost (including the cost of machine setup time) of around $40. "

Debugging and Monitoring Your Training Process

Tools:

- PyCharm (Get the professional version if you want remote development)
- Tensorboard (Plot train/val curves, Visualize data)

Summary - Tricks and Tips in DL Training

- Overview
- Data Preprocessing
- Network Details
- Training Dynamics

References:
http://cs231n.stanford.edu/syllabus.html

https://towardsdatascience.com/a-bunch-of-tips-and-tricks-for-training-deep-neural-networks-3ca24c31ddc8

http://karpathy.github.io/2019/04/25/recipe/