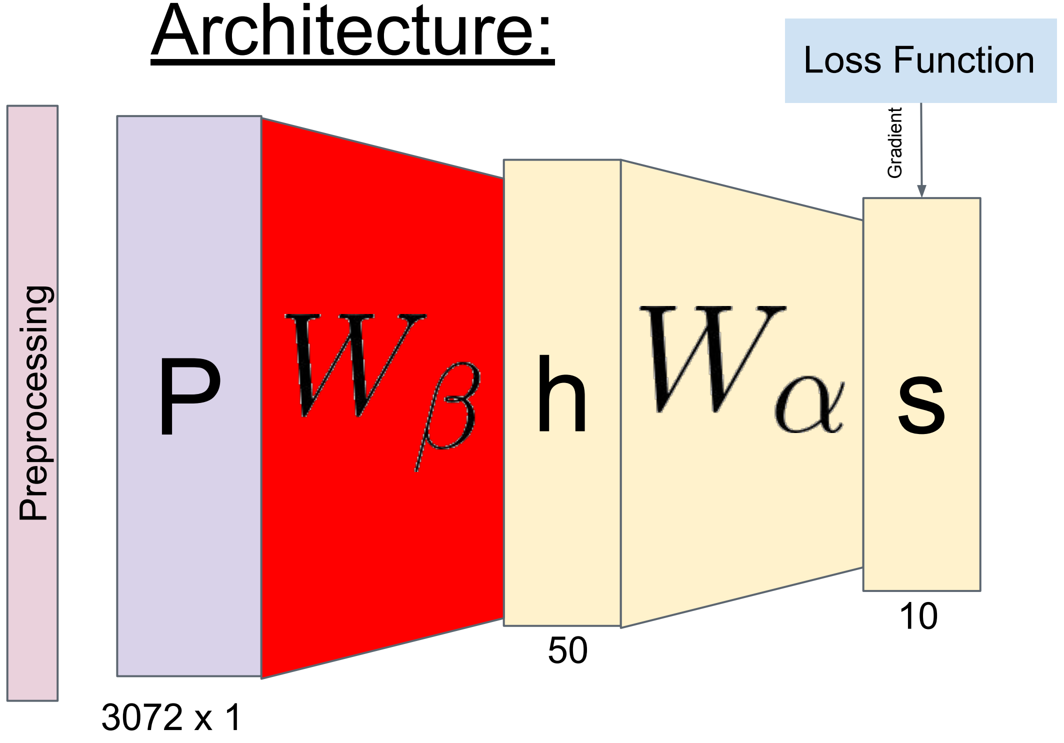
DATA 442: Neural Networks & Deep Learning

Dan Runfola – danr@wm.edu

icss.wm.edu/data442/

Preprocessing: Zero Centered Data
Weights Initialization: He
Activations: ReLU

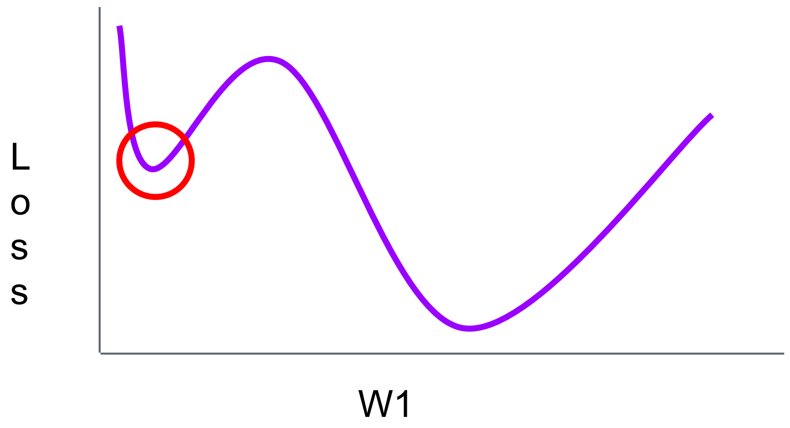
<u>Architecture:</u>



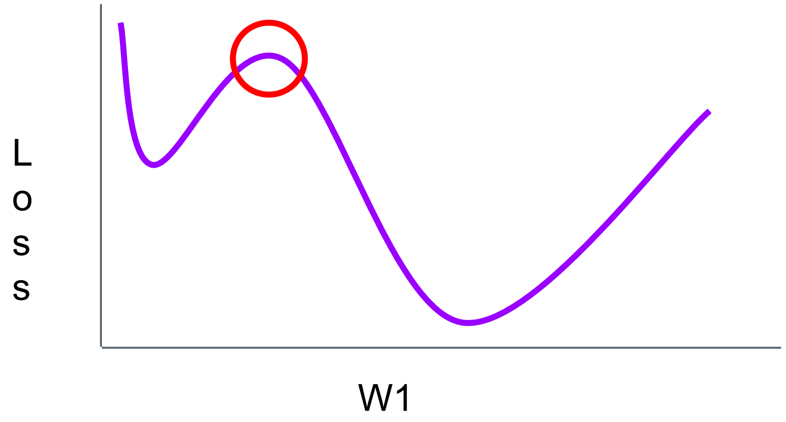

```python
def affineBackward(dUpstream, cache):
    X, W, B = cache

    #Same steps as the forward pass:
    N = X.shape[0]
    D = np.prod(X.shape[1:])
    xReshape = np.reshape(X, (N, D))

    #Gradient calculations for the affine case - nothing you haven't
    #seen before!
    dx = np.reshape(np.dot(dUpstream, W.T), X.shape)
    dw = np.dot(xReshape.T, dUpstream)
    db = np.dot(dUpstream.T, np.ones(N))

    return(dx, dw, db)
```

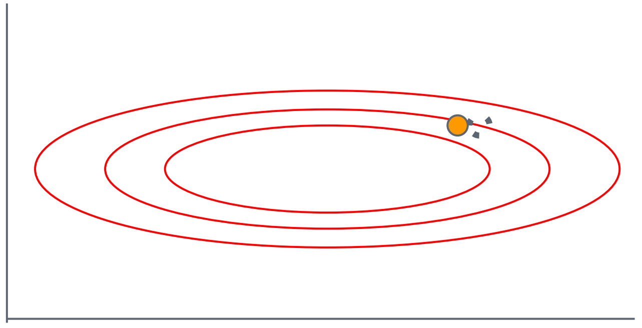
SGD: $W_{iteration+1} = W_{iteration} - \alpha \Delta f(W_{iteration})$

Local Minima



Saddle Points





Poor Conditioning

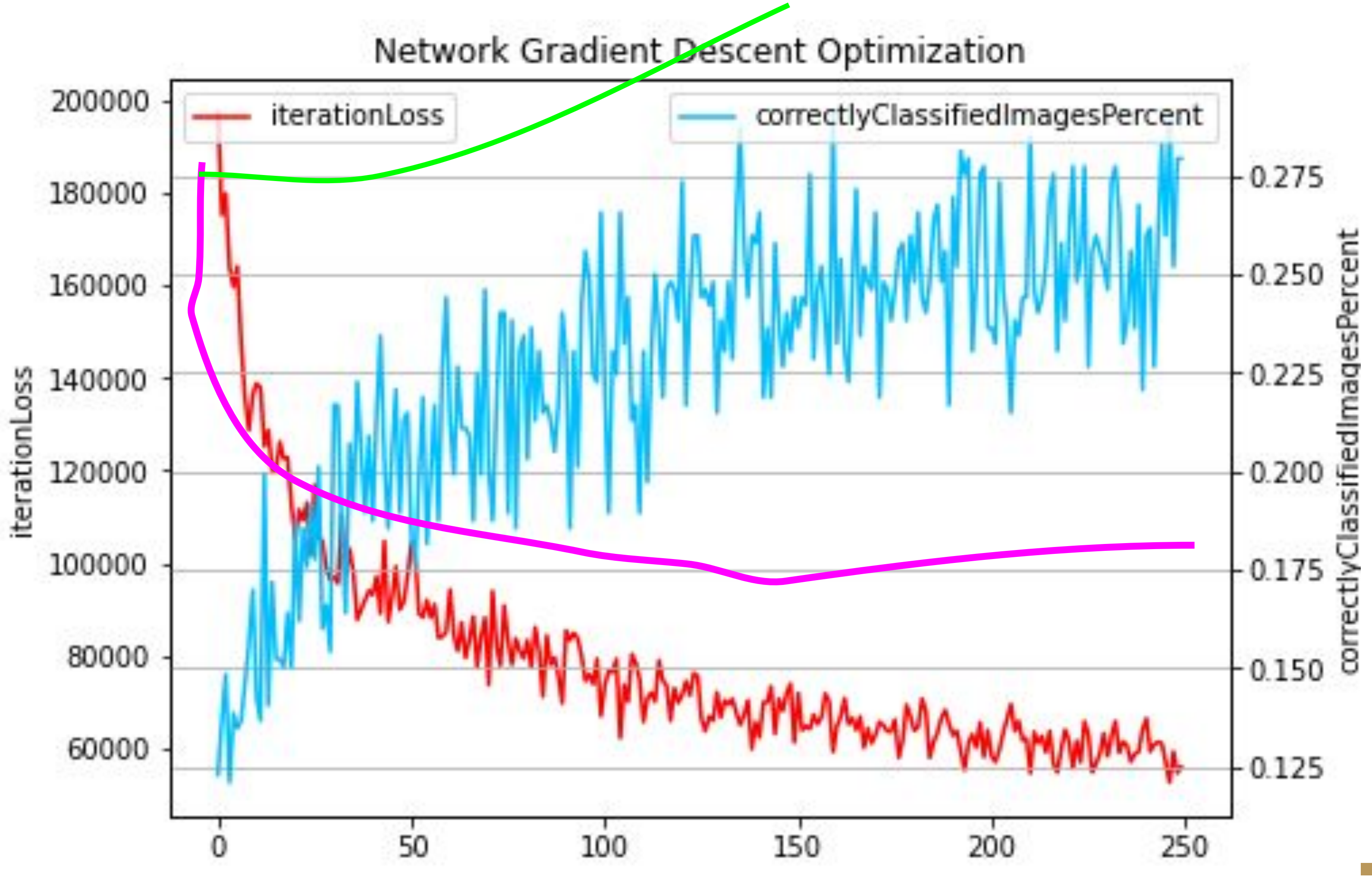
ADAM (Kingma and Ba)

Beta 1 - Similar to Friction in SGD + Momentum

Beta 2 - Similar to Decay Rate in RMSProp

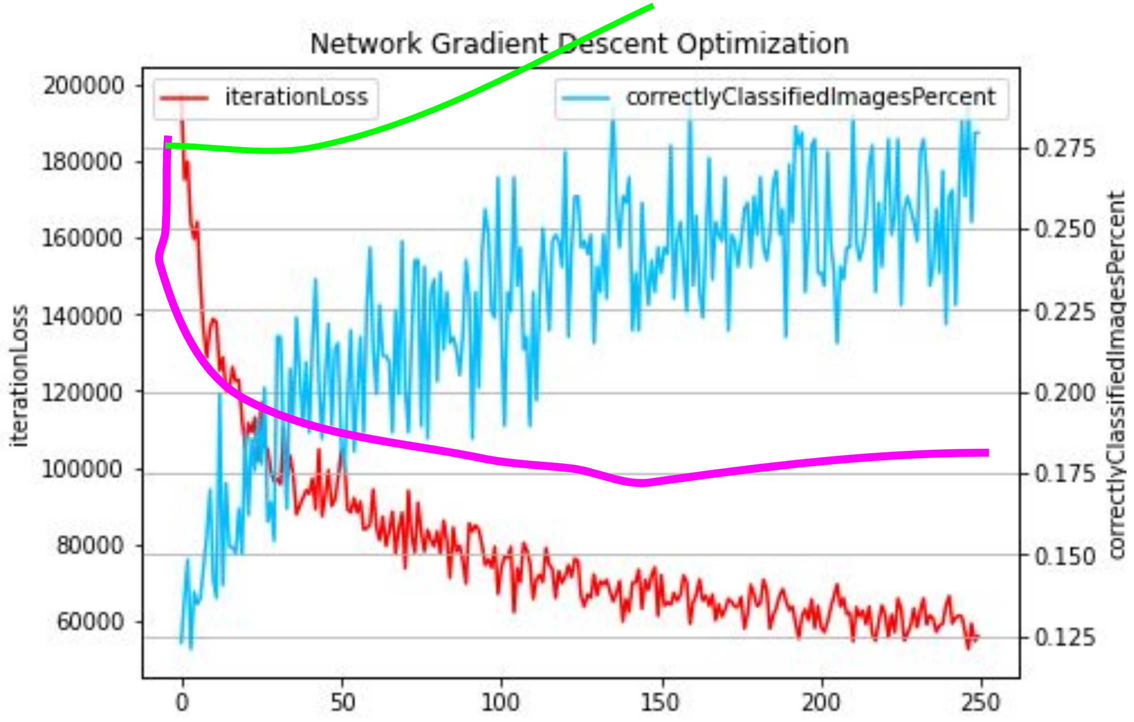
Practical Tip: Beta1 = 0.9, beta2=0.99, LR = 1e-3 can provide a strong starting condition for tests with Adam.

What is a “Good” Learning Rate?

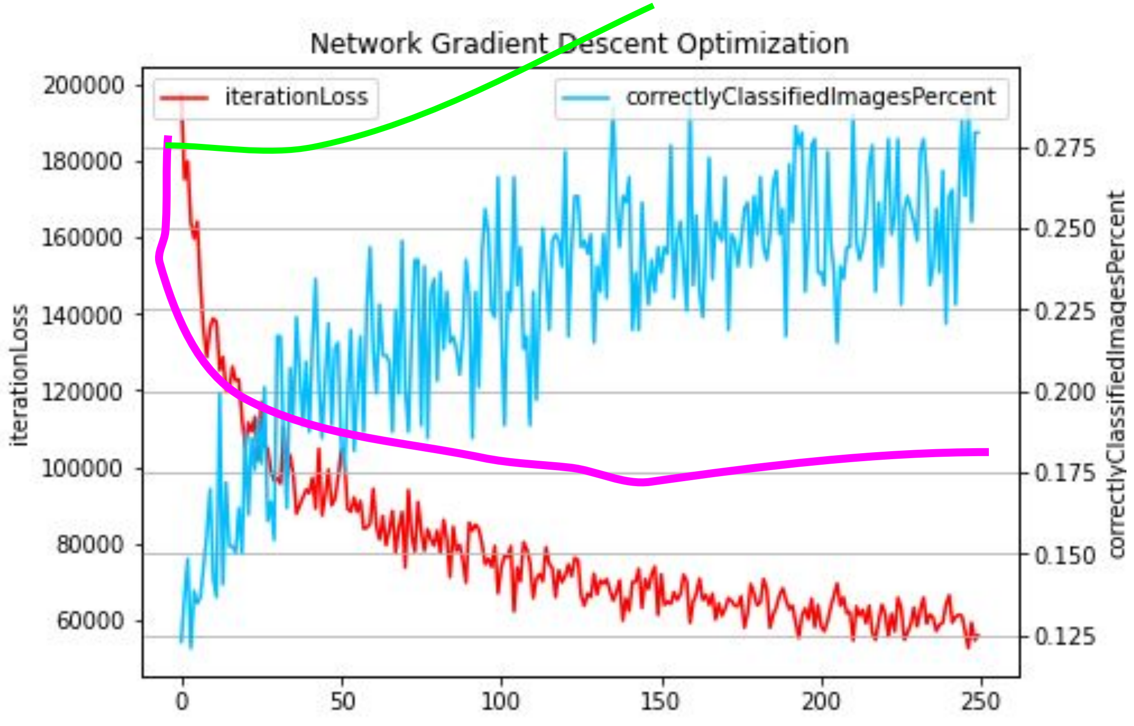


What is a “Good” Learning rate?

- Learning rate can change!
- Can take the good properties of multiple curves. For example, starting with a high learning rate, and then slowing it down.

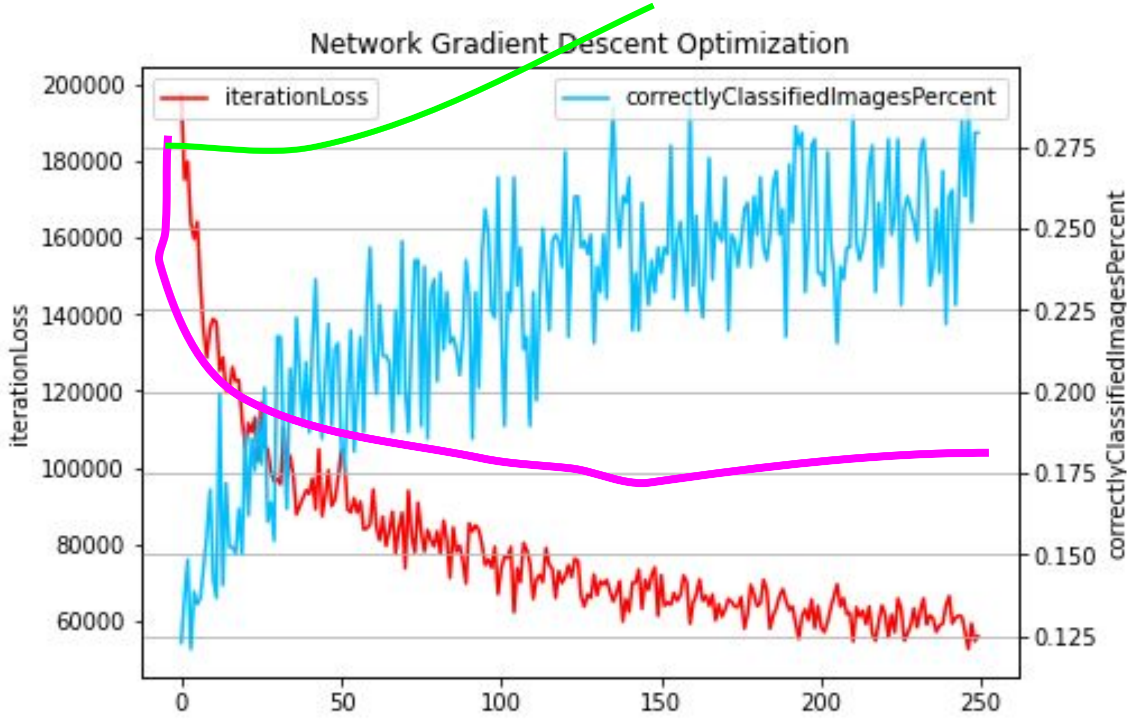


What is a “Good” Learning rate?



What is a “Good” Learning rate?

Step Decay - Every *k* iterations, the learning rate is cut by half.

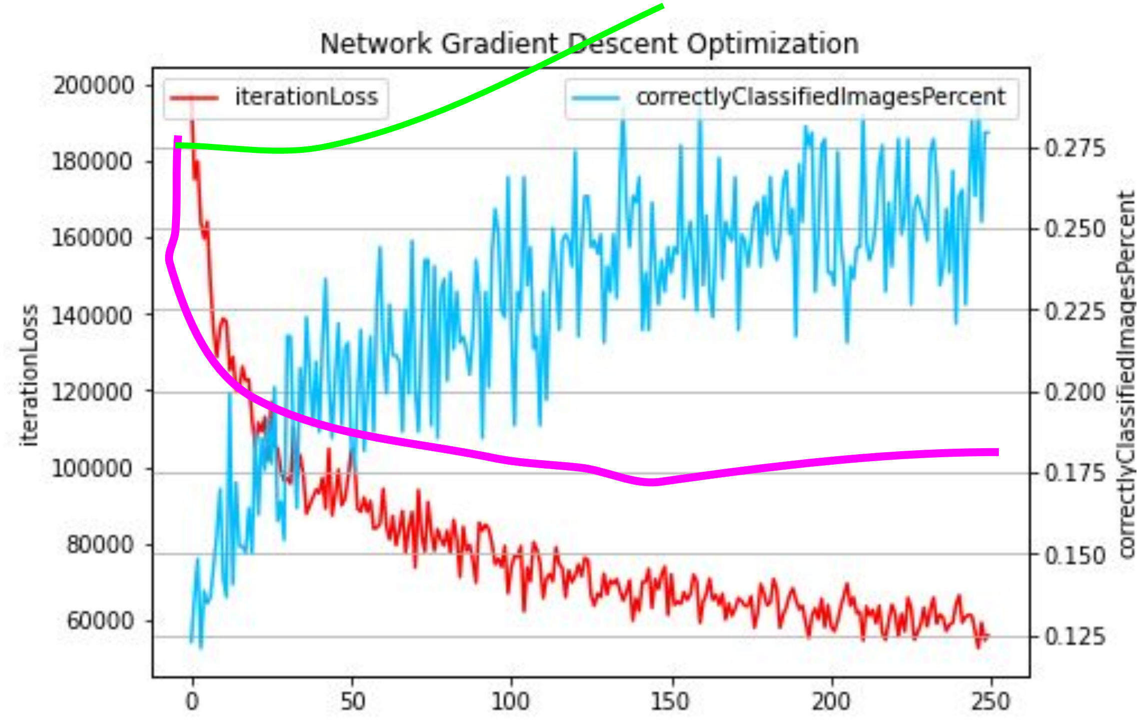


What is a "Good" Learning rate?

Step Decay - Every *k* iterations, the learning rate is cut by half.

Exponential Decay:

$$\alpha_{i+1} = \alpha_i e^{-ki}$$



What is a “Good” Learning rate?

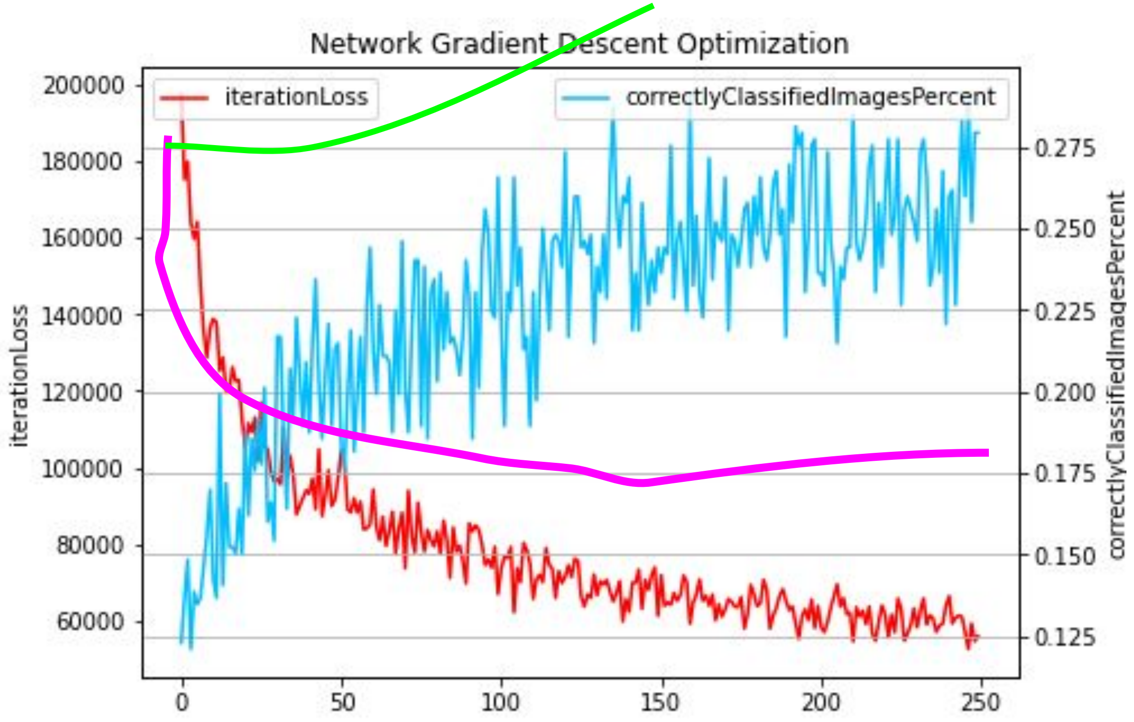
Step Decay - Every *k* iterations, the learning rate is cut by half.

Exponential Decay:

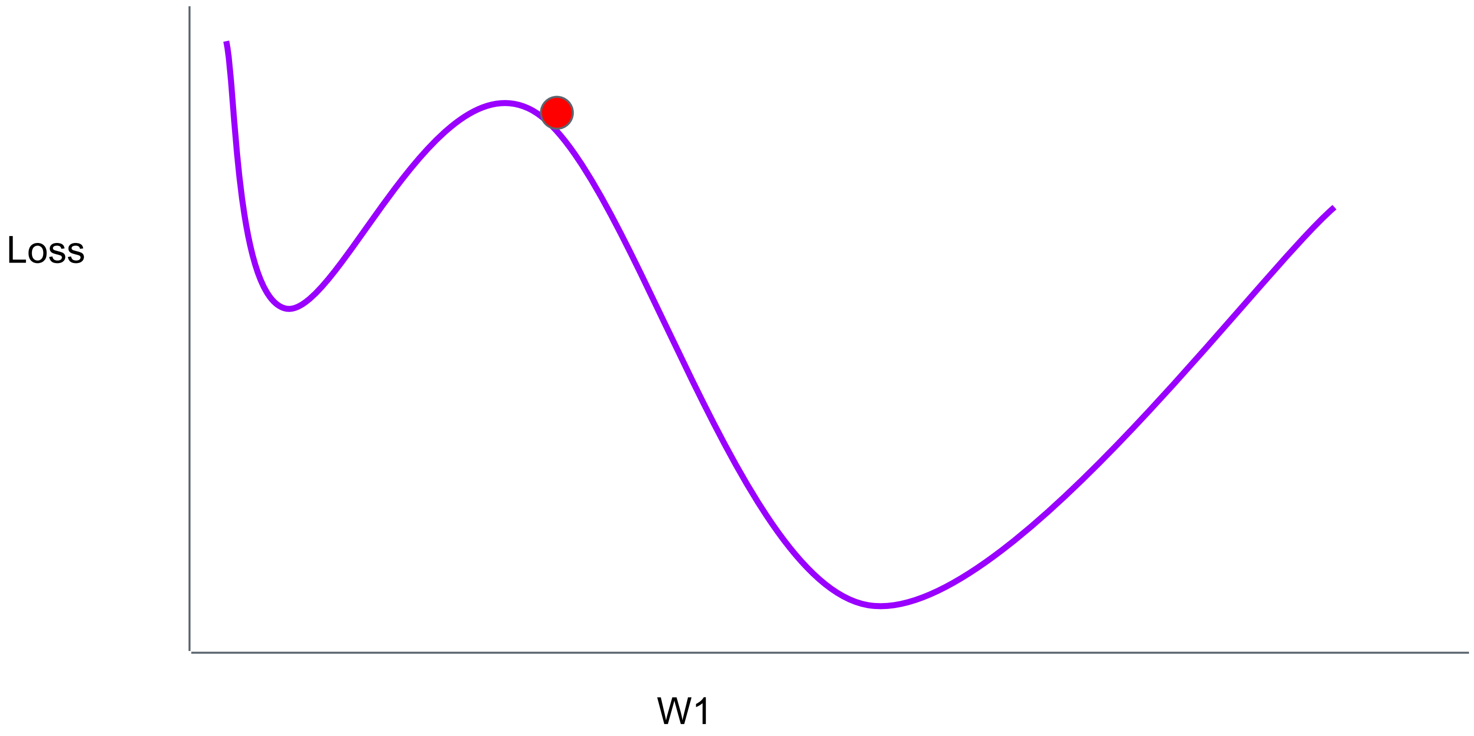
$$\alpha_{i+1} = \alpha_i e^{-ki}$$

Inverse Decay:

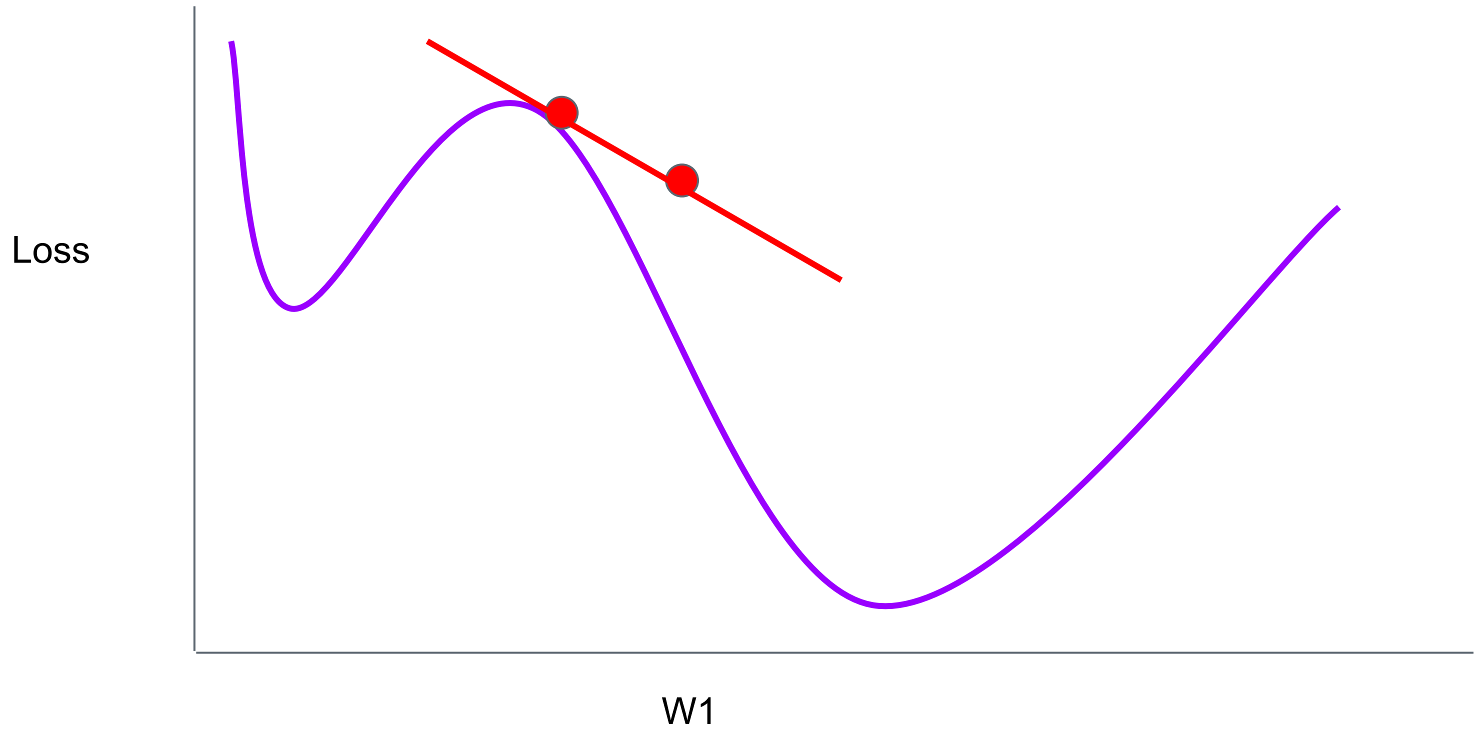
$$\alpha_{i+1} = \alpha_i / (1 + ki)$$



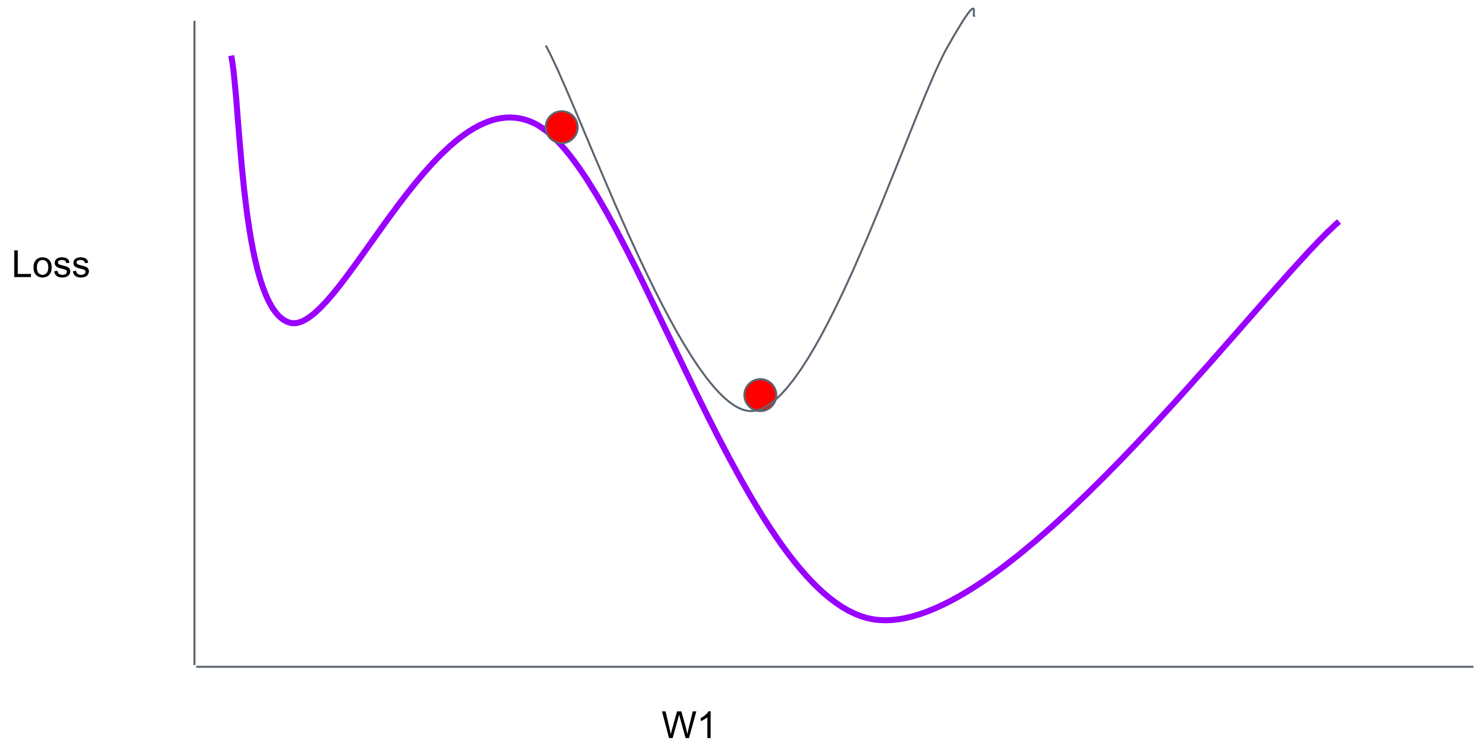
Why not just skip learning rates?



Why not just skip learning rates?



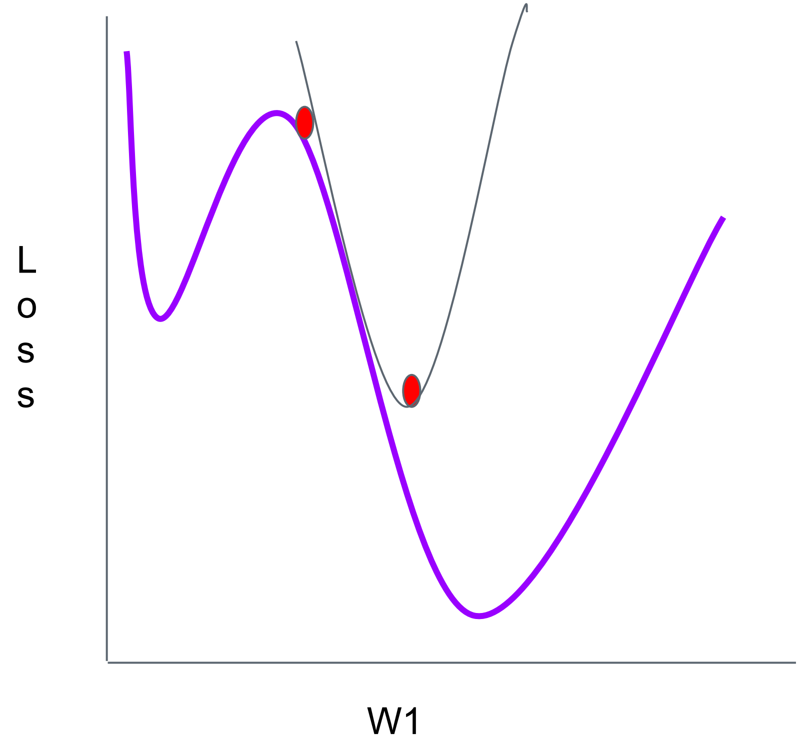
Why not just skip learning rates?



Why not just skip learning rates?

Hessian Matrix: O(N^2)
Inversion of Hessian: O(N^3)

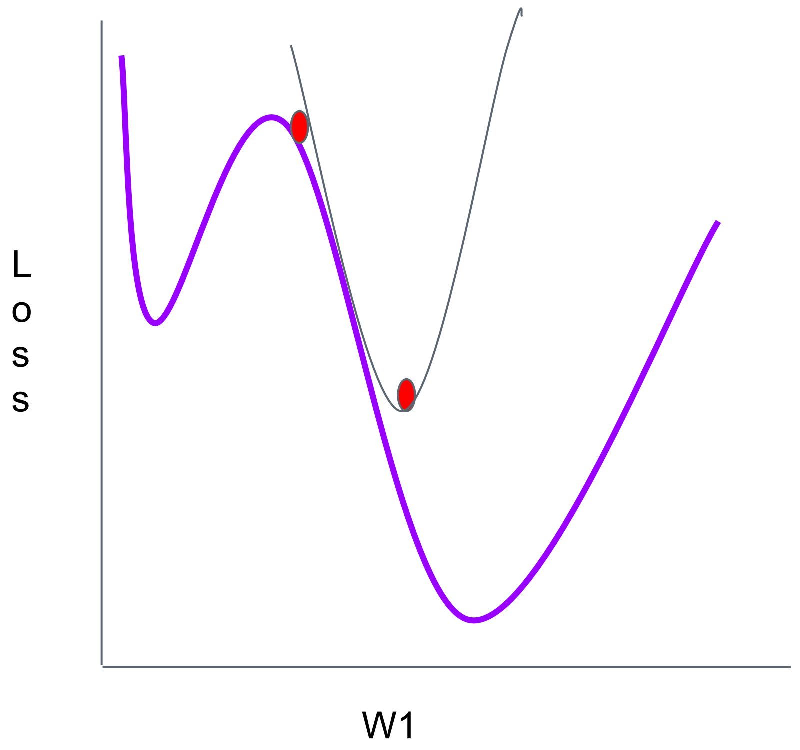
A small neural net has 100,000? parameters. So, around a petabyte of memory would be required to invert a hessian matrix in a small case. This is obviously not well suited to (most) available computation!

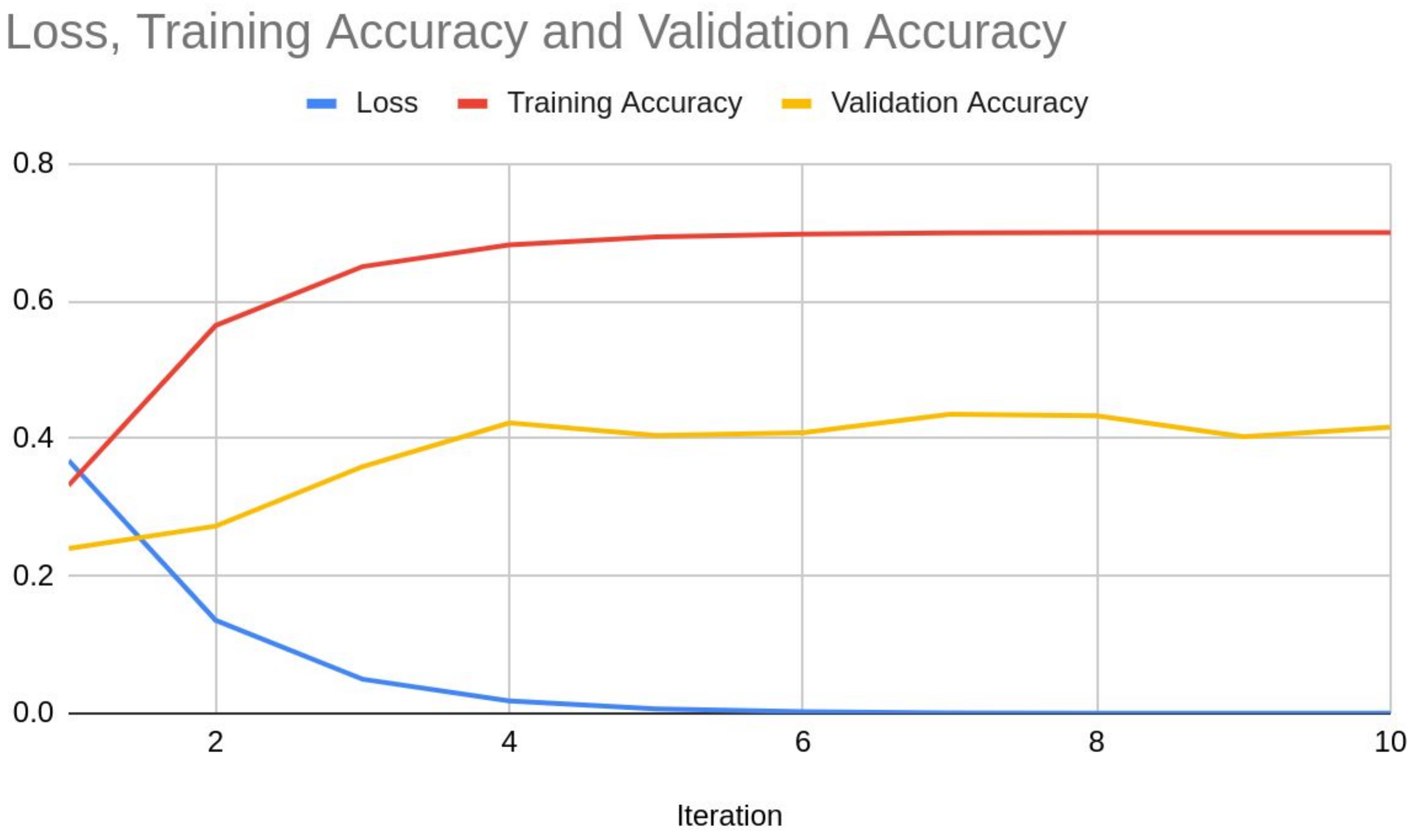


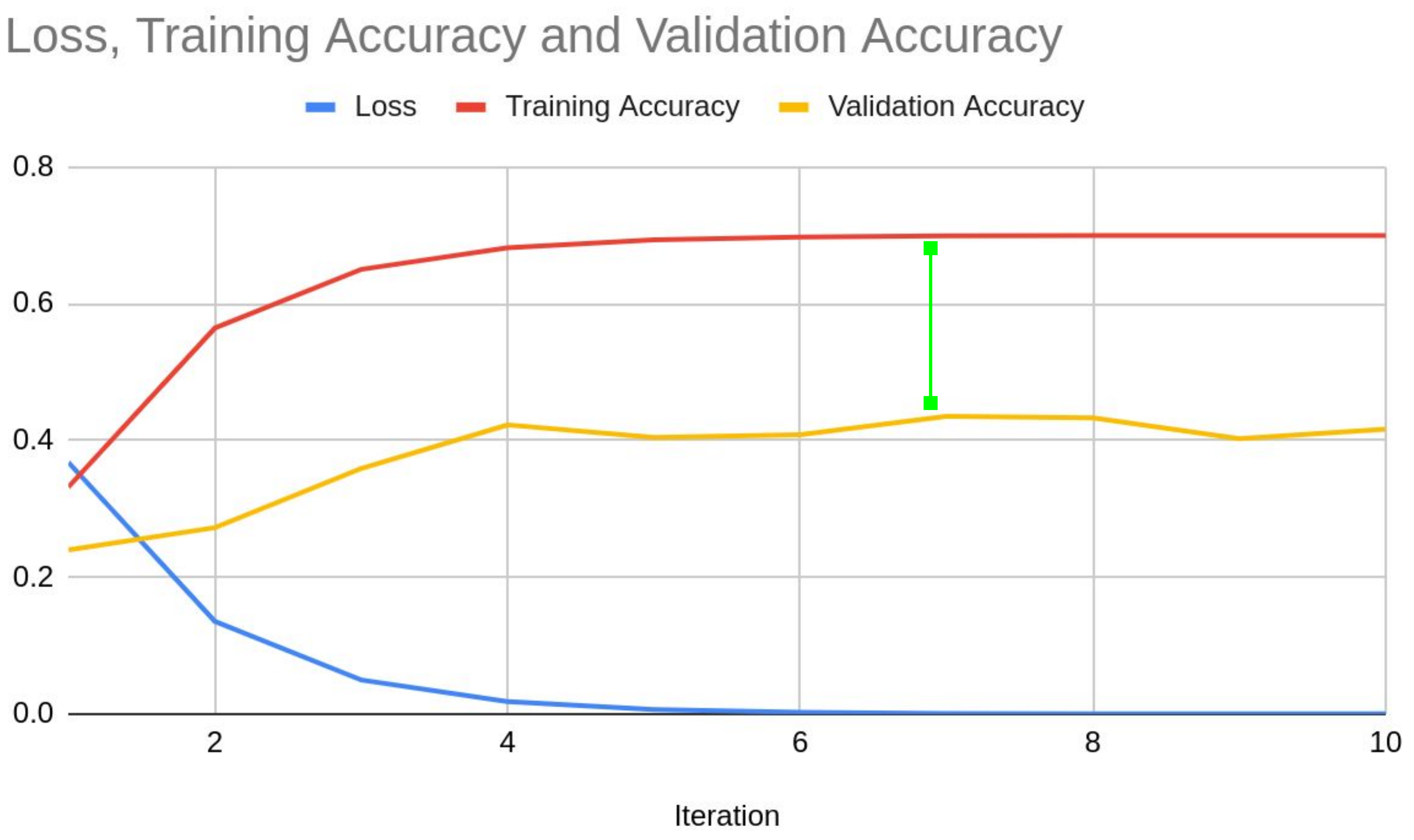
Quasi-Newton Alternatives

BFGS - Broyden Fletcher Goldfarb Shanno algorithm.
Approximates Hessian with low-rank updates.

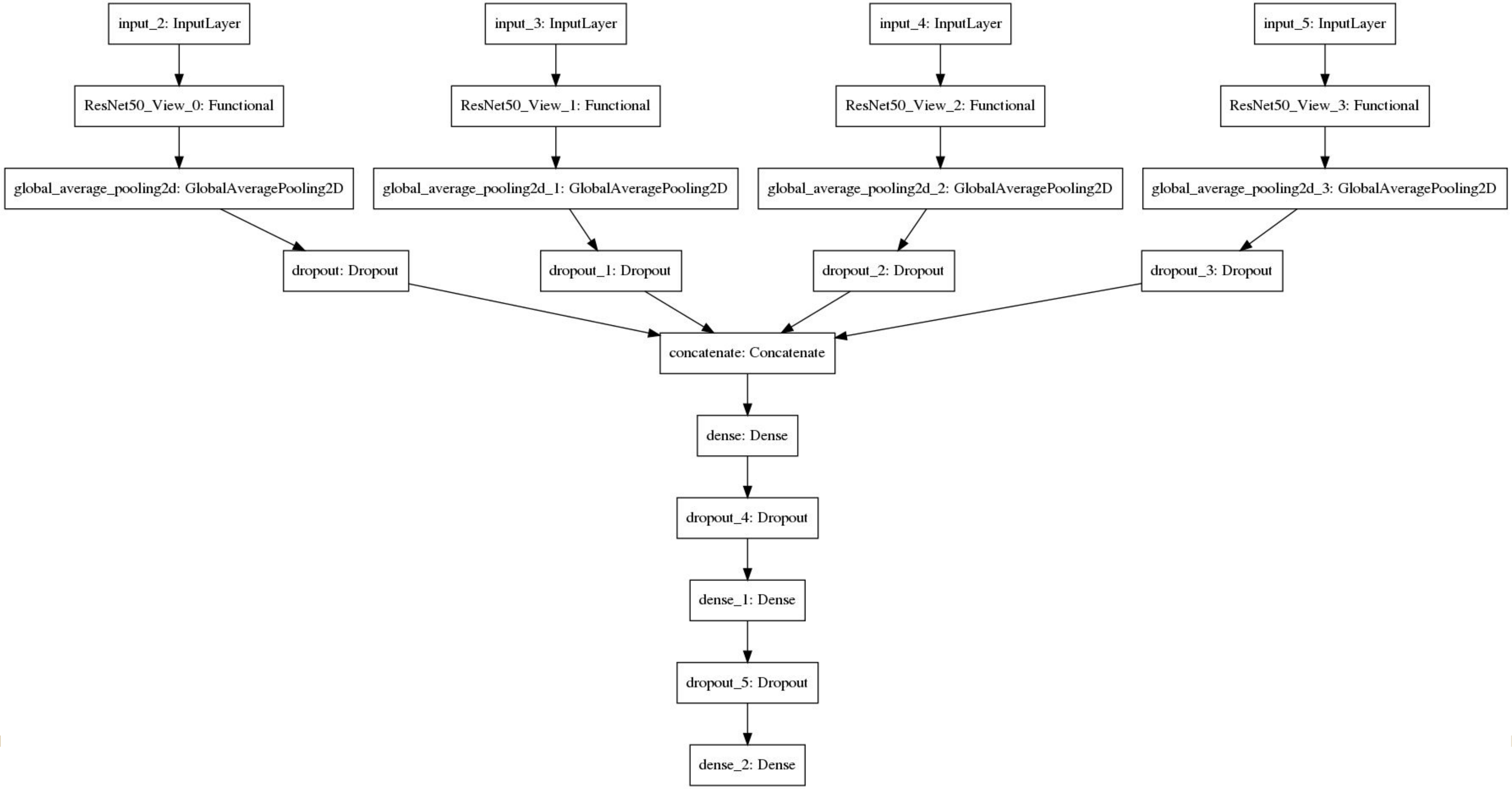
L-BFGS - Limited Memory BFGS.
Avoids storing the full Hessian during approximation, but does poorly with stochastic updates (i.e., if you are batching it struggles).




Loss, Training Accuracy and Validation Accuracy
Loss
Training Accuracy
Validation Accuracy
0.8
0.6
0.4
0.2
0.0
2
4
6
8
10
Iteration


Loss, Training Accuracy and Validation Accuracy
Loss
Training Accuracy
Validation Accuracy
0.8
0.6
0.4
0.2
0.0
2
4
6
8
10
Iteration

Model Ensembles



Repeat:

-> Forward Pass

-> Backward Pass

-> Update Weights with Gradient

Final weights after all iterations complete, or convergence.

Repeat:
-> Forward Pass
-> Backward Pass
-> Update Weights with Gradient

Dramatically Increase Learning Rate

Save weights after convergence (become ensemble member).

Snapshot Ensemble

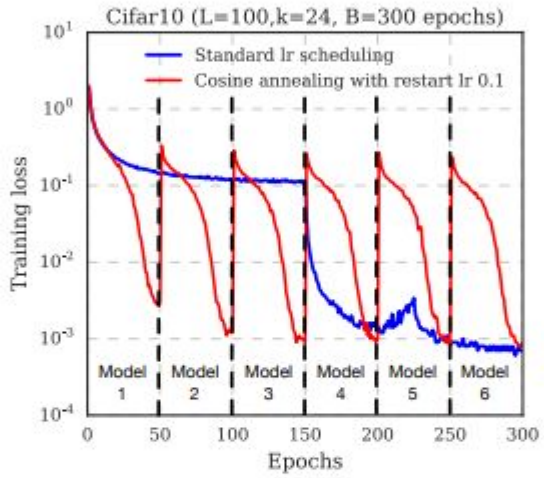


Figure 2: Training loss of 100-layer DenseNet on CIFAR10 using standard learning rate (blue) and $M = 6$ cosine annealing cycles (red). The intermediate models, denoted by the dotted lines, form an ensemble at the end of training.

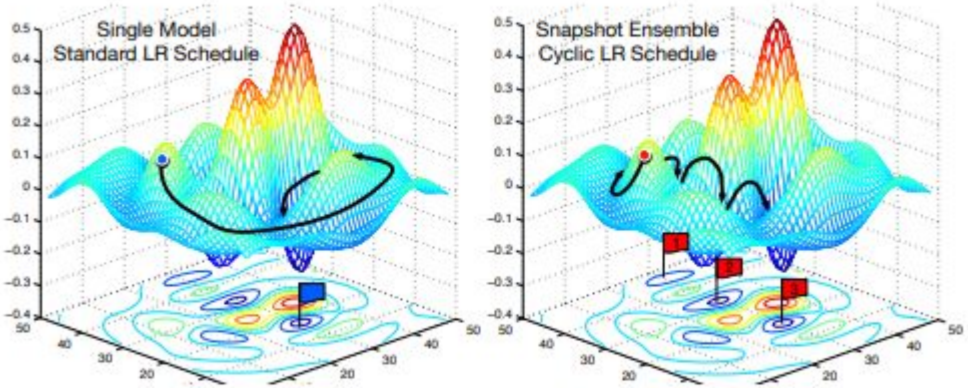
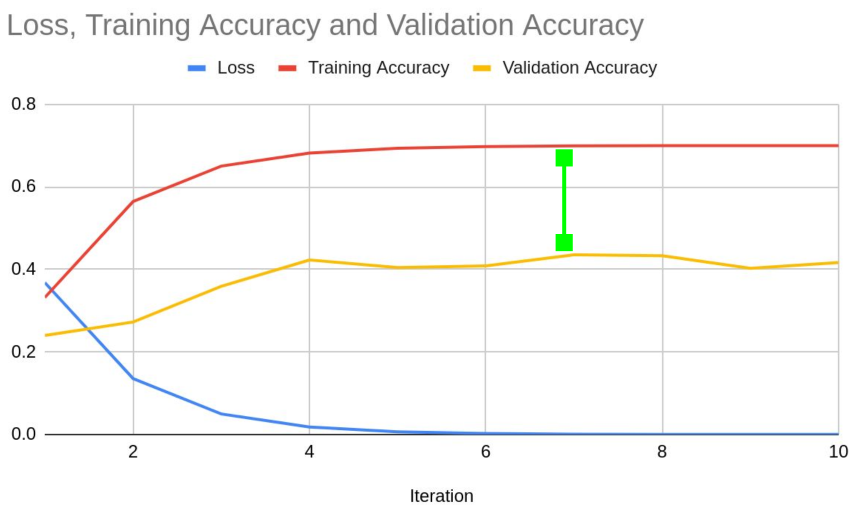


Figure 1: **Left:** Illustration of SGD optimization with a typical learning rate schedule. The model converges to a minimum at the end of training. **Right:** Illustration of Snapshot Ensembling. The model undergoes several learning rate annealing cycles, converging to and escaping from multiple local minima. We take a snapshot at each minimum for test-time ensembling.

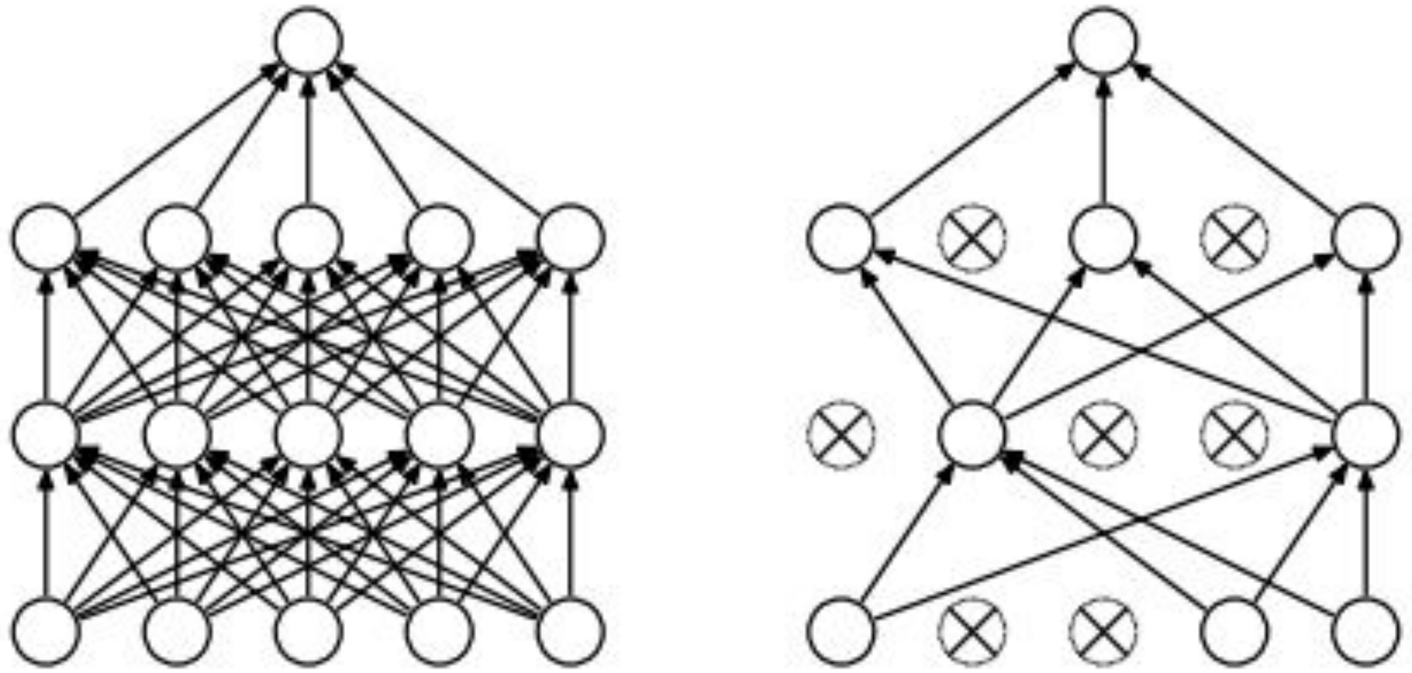
https://arxiv.org/pdf/1704.00109.pdf

$$\underbrace{\frac{1}{N}\sum_{i}^{N} Loss_i(f(x_i, W), y_i)}_{\text{Data Loss}} + \underbrace{\boxed{\lambda} R(W)}_{\text{Regularization Loss}}$$

$$R(W) = \sum_{k=1}^{K} W_k^2$$



Dropout



(a) Standard Neural Net (b) After applying dropout.

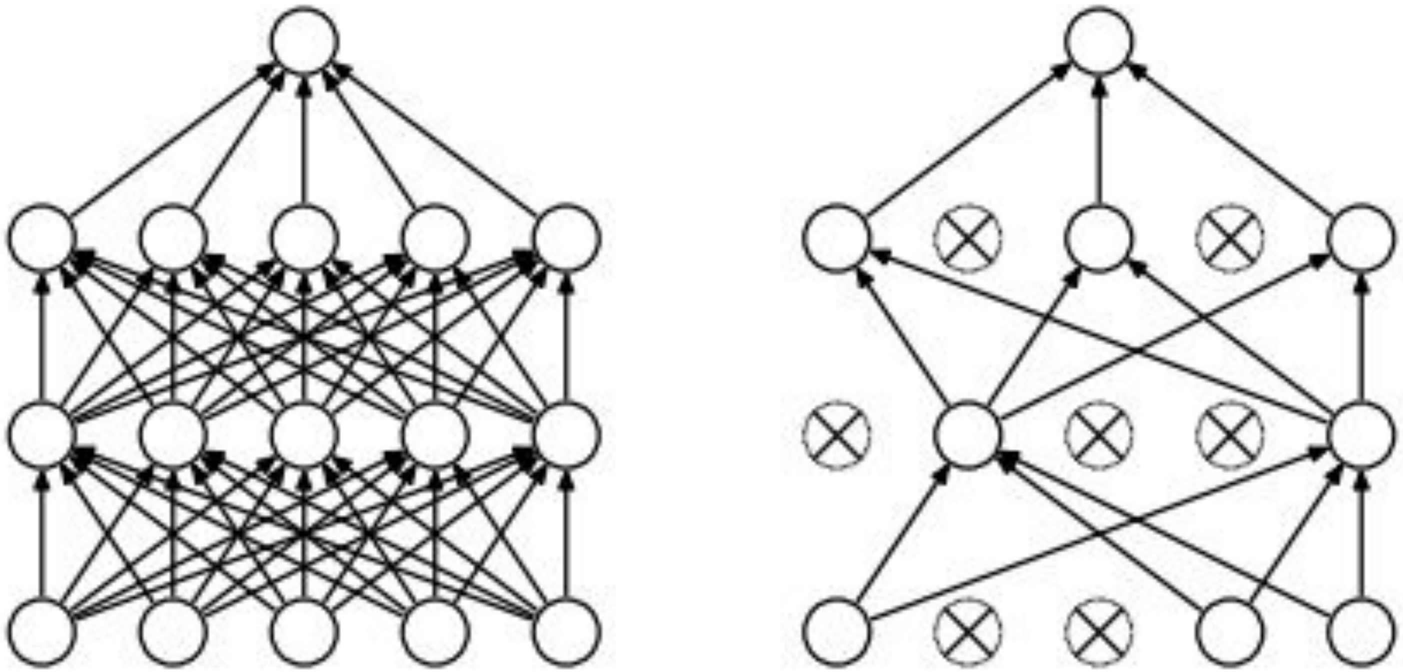
Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

https://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Today: Toddler

Tomorrow: Candy Cane

Dropout



(a) Standard Neural Net

(b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

https://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

DropConnect

Regularization of Neural Networks using DropConnect

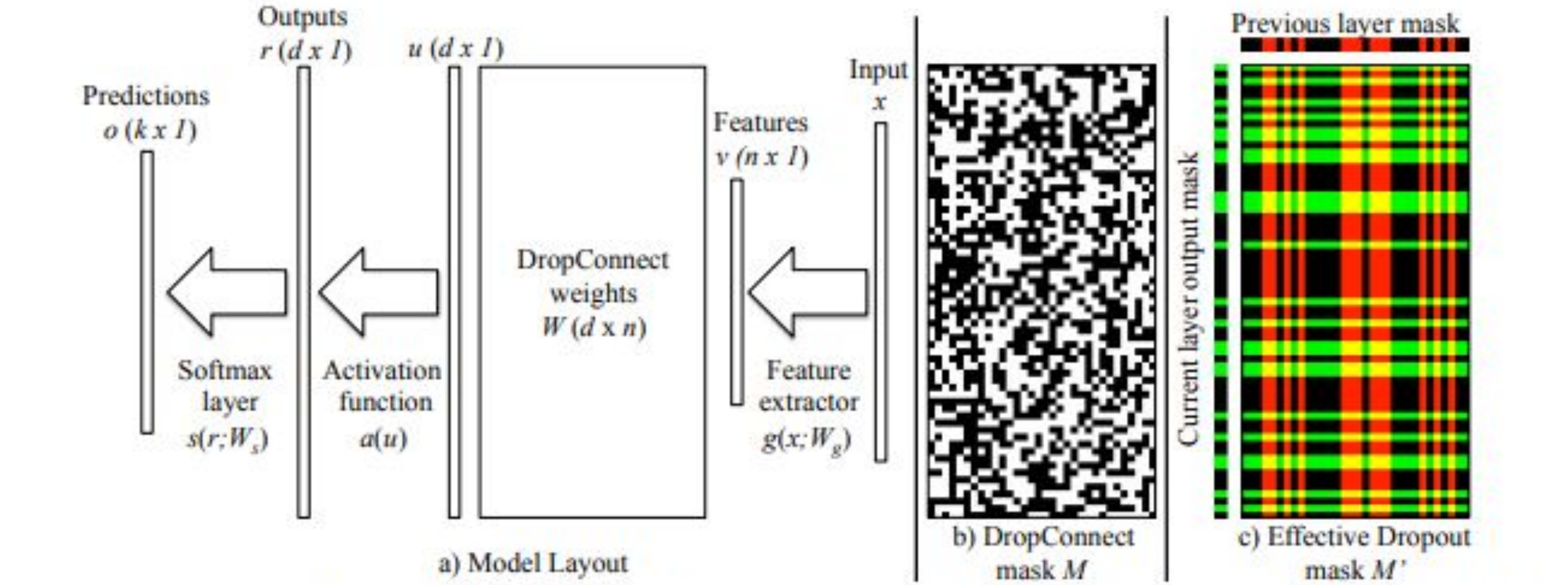
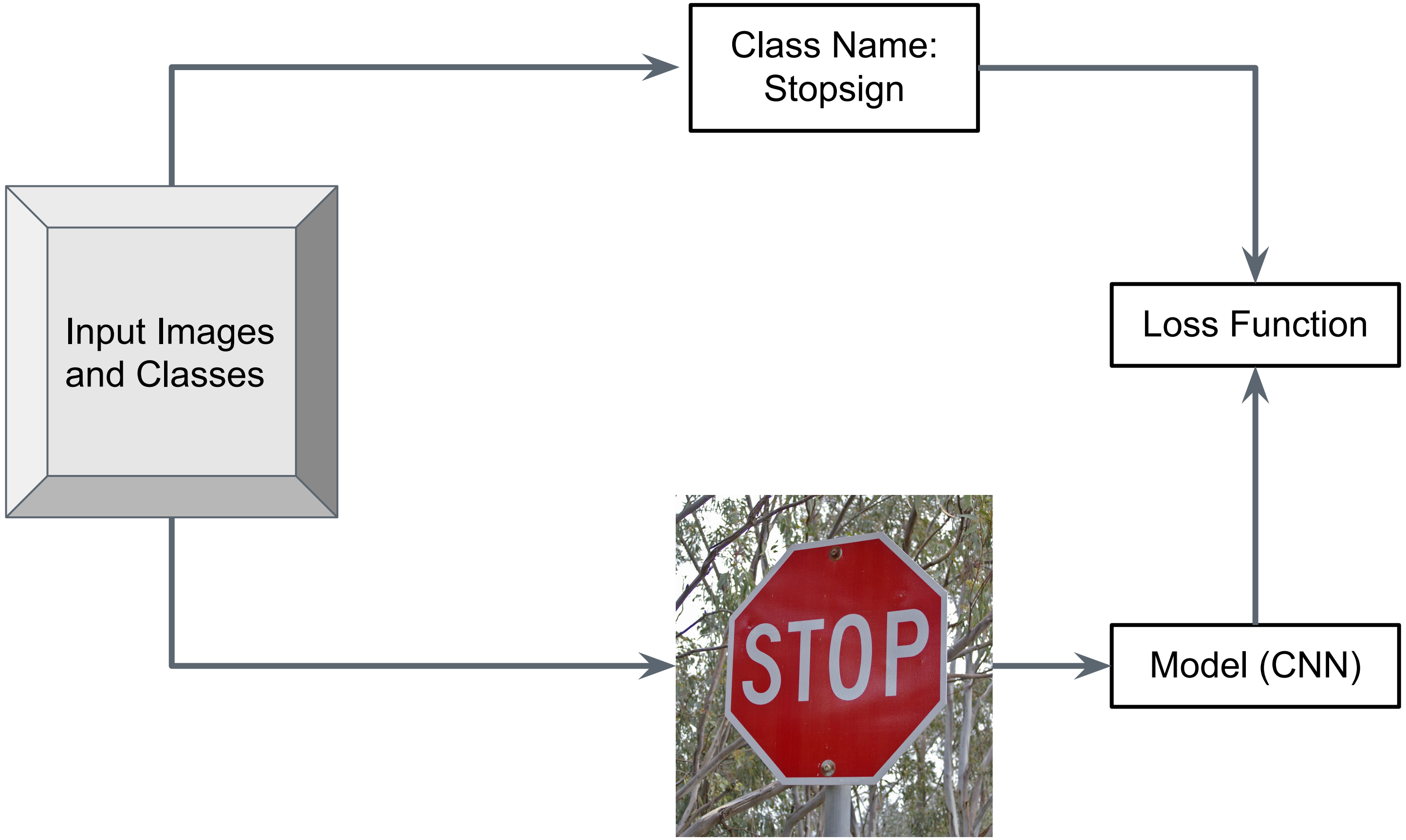


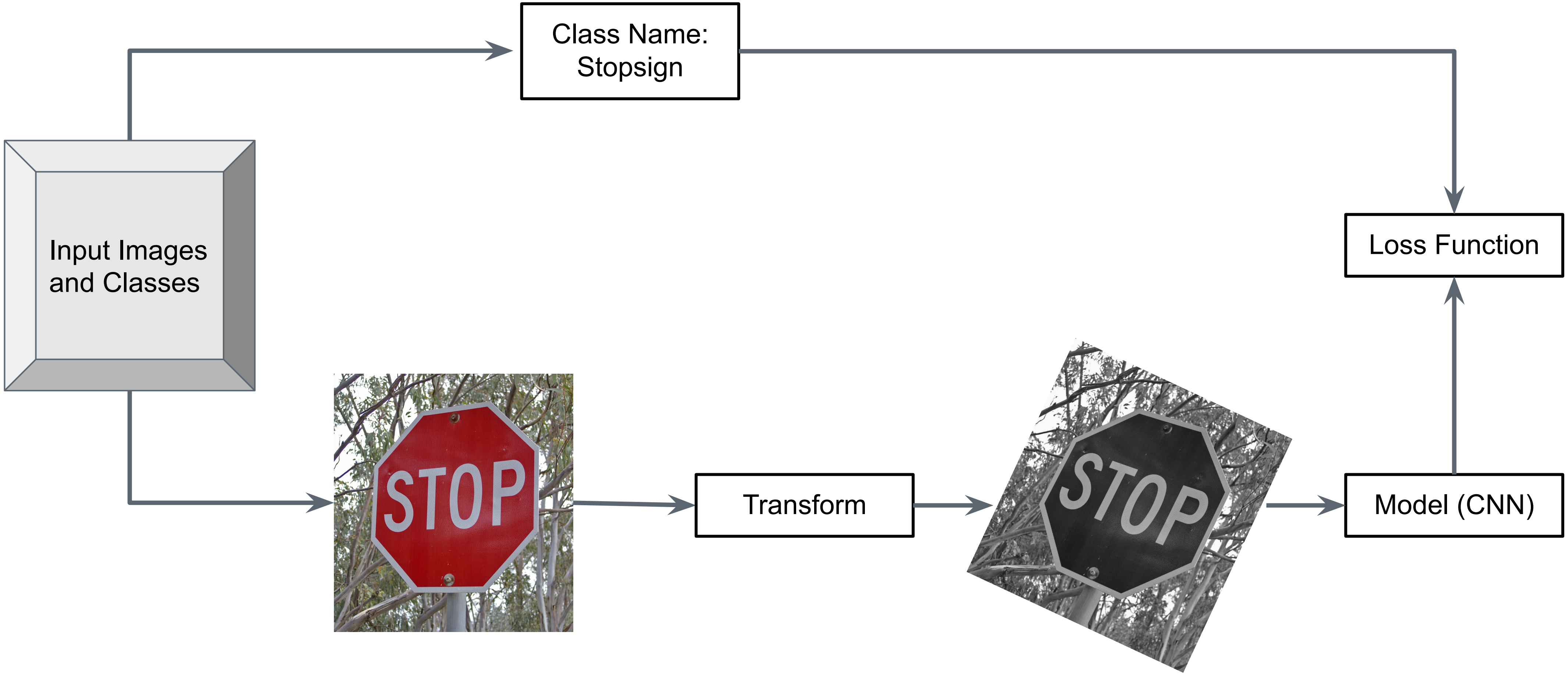
Figure 1. (a): An example model layout for a single DropConnect layer. After running feature extractor $g()$ on input x, a random instantiation of the mask M (e.g. (b)), masks out the weight matrix W. The masked weights are multiplied with this feature vector to produce u which is the input to an activation function a and a softmax layer s. For comparison, (c) shows an effective weight mask for elements that Dropout uses when applied to the previous layer's output (red columns) and this layer's output (green rows). Note the lack of structure in (b) compared to (c).

http://proceedings.mlr.press/v28/wan13.pdf

Data Augmentation



Data Augmentation



Data Augmentation

ORIGINAL

FLIP

Data Augmentation

ORIGINAL

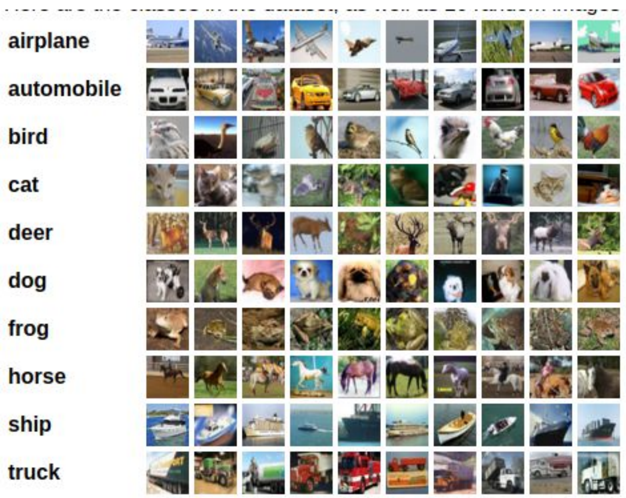
FLIP

Contrast / Brightness

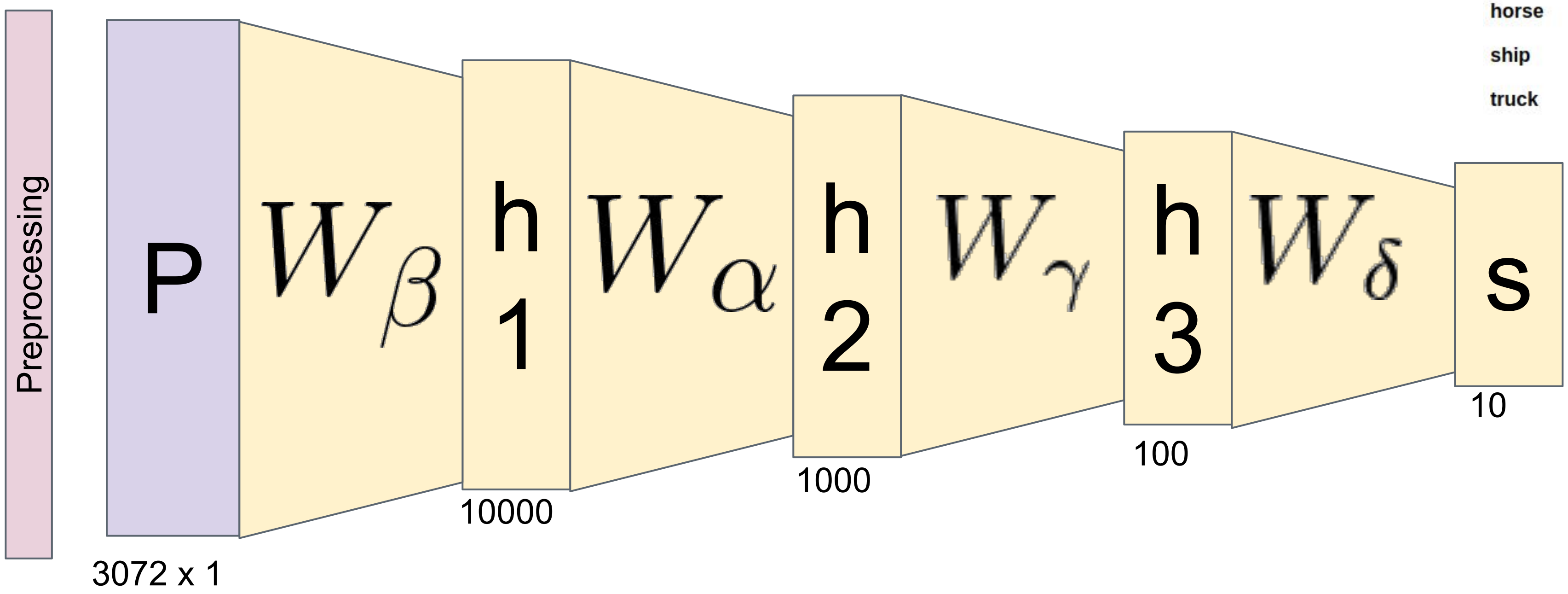
Cropping

Transfer Learning

CNN Architecture → Train on Large Dataset (i.e., ImageNet) → Fine tune based on small Dataset (i.e., CIFAR10).

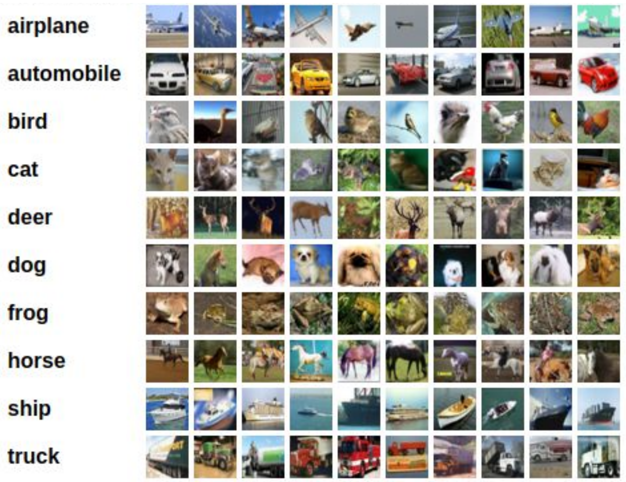


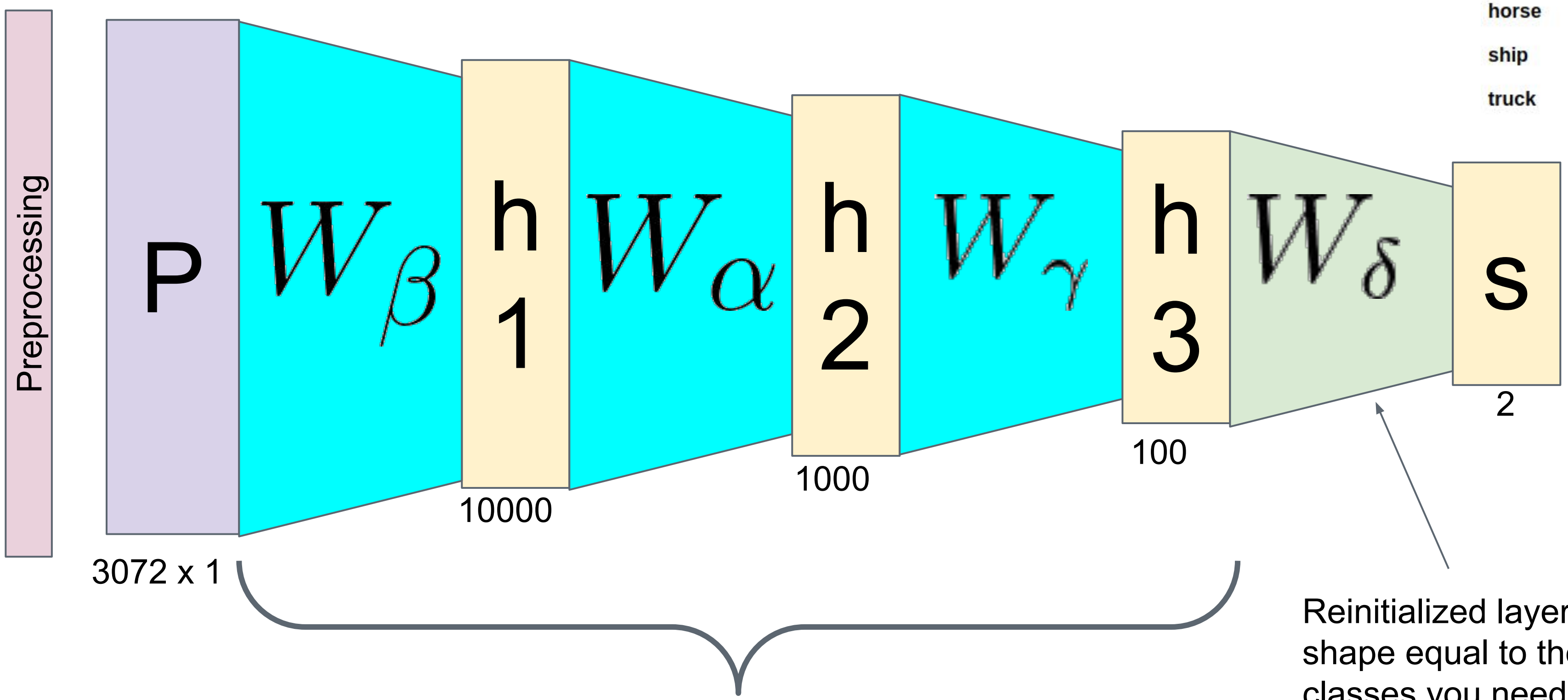
Transfer Learning



Transfer Learning

Transfer Learning





Frozen Weights Layers, Calculated from all of CIFAR

Reinitialized layer with an output shape equal to the number of classes you need (Parameters here = 200)

Practical Pointers

1) If you have less than 1 million(!) images - consider using a large dataset of similar data to train your network on.

2) Apply a transfer learning approach on your own dataset.

Note that most major packages - i.e., Keras, PyTorch - provide easy mechanisms to load pre-trained weights in for a wide range of architectures.

Summary

- Strategies for setting or varying a learning rate
- Newtonian Steps (no learning rate) vs. Linear Approximations
- Strategies for reducing the difference between training accuracy and testing accuracy
  - Model Ensembles
  - Regularization
  - Dropout
  - Data Augmentation
- Transfer Learning