Introduction To Deep Learning

Workshop 3

NetBrain.ml

Last Edit: 11/11/19

Optimizers

Stochastic gradient descent: Uses ***standard gradient descent*** after every example (Updates after every example)

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta).$$

Mini-batching: Groups a '***mini-batch***' of examples in the dataset and uses optimizer after every minibatch

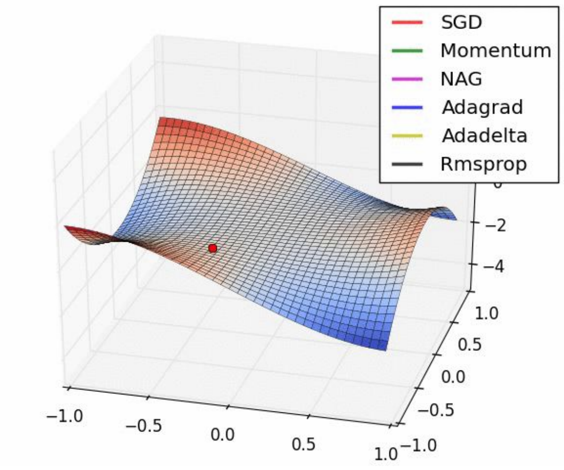
Batch gradient descent: Updates once a epoch

Momentum: ***Weighs new gradients and the past***

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

Nesterov accelerated gradient: A 'smarter' momentum that takes ***the gradient of after updated***

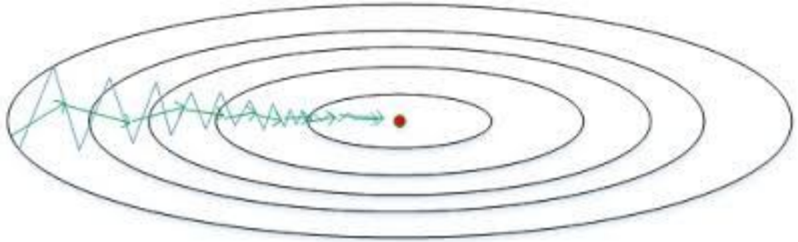
$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$



Optimizers

RMSProp: Similar to momentum with squared gradient but is geared to ***prevent oscillations***, and ***penalties*** for it

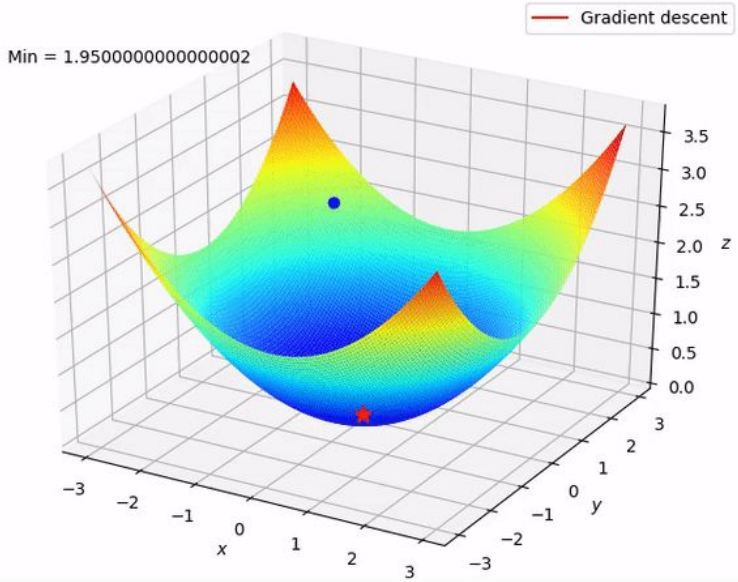
$$v_{dw} = \beta \cdot v_{dw} + (1-\beta) \cdot dw^2 \qquad W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$



Adam: ***Combines Momentum(or AdaGrad) and RMSProp***

$$\nu_t = \beta_1 * \nu_{t-1} - (1-\beta_1) * g_t$$
$$s_t = \beta_2 * s_{t-1} - (1-\beta_2) * g_t^2$$

$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$
$$\omega_{t+1} = \omega_t + \Delta\omega_t$$



What to choose?

SGD or Adam is common to use as a start, but other may work better. (Note: https://ruder.io/optimizing-gradient-descent/)

More information can be found : *https://medium.com/datadriveninvestor/overview-of-different-optimizers-for-neural-networks-e0ed119440c3*

Parameter Initializations

Goal: ***Start off with good values*** that can lead to optimal convergence.

Optimal convergence: [Deeplearning.ai Visual]

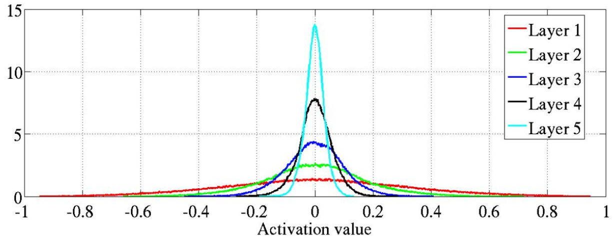
Initialization Goal: Want to have unit variances with zero mean in activation values throughout model.

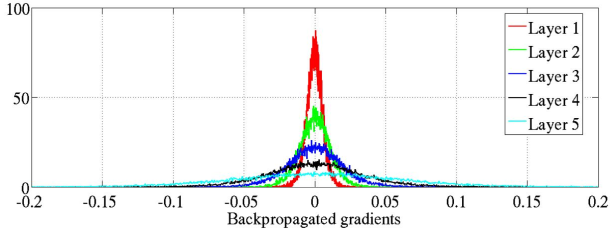
Xavier: Experimented with sigmoid activation

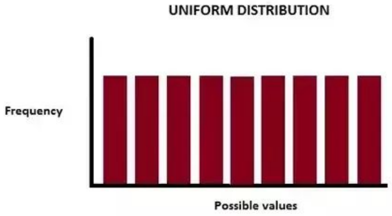
Kaiming/He: Experimented with relu activation

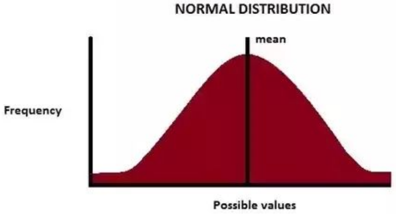
Normal: Uses bell-shaped curve to describe probability of all allowable values

Uniform: Allows probability of all allowable values to be equal









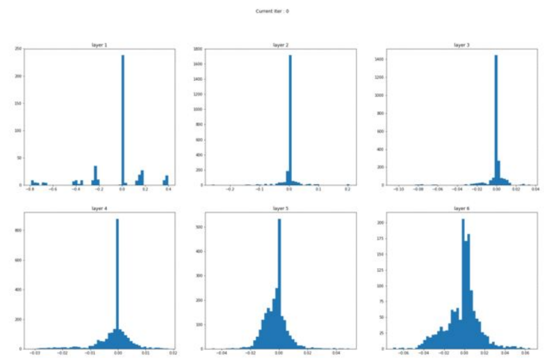
Normalization Layers

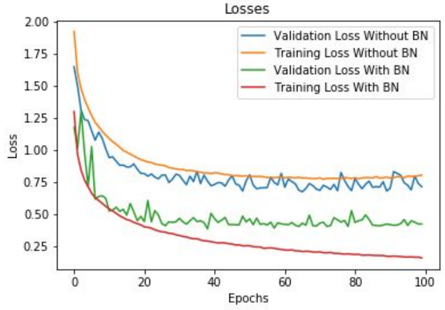
Goal: ***Speed up and optimize learning*** by putting all the features into the same scale by normalizing within hidden layers

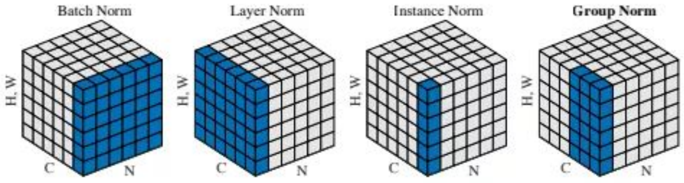
Normalization: Subtract by the mean and divide by the standard deviation, to have a unit variance and mean of zero.

Batch Norm: Does normalization ***by batch*** and i***ntroduces two learnable parameters*** for denormalization during backpropagation.

There are many other normalization layers, but they are less common and attempt to tackle various potential issues with Batch Norm, mainly its inability to learn on unit batch size and its difficulties with recurrent networks.







Blocks

Goal: Create an ***abstraction*** with layers

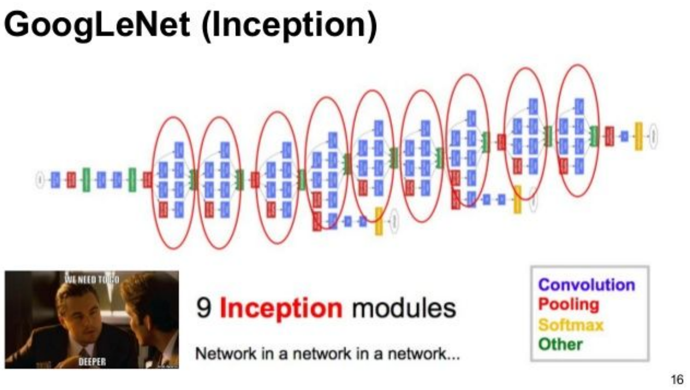
Ex: Inception modules

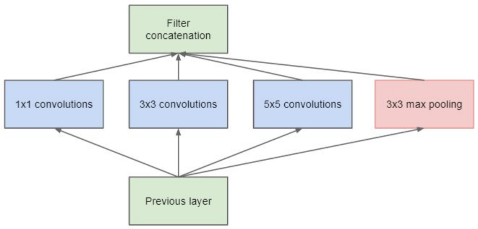
Important ideas:

Kernel sizes of 1: Control the number of channels without changes in other dims

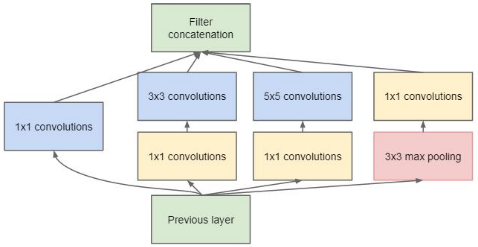
Pooling: Control the lower dims without changes in number of channels

This is essentially refactoring our code, but it still performs well.





(a) Inception module, naïve version



(b) Inception module with dimension reductions

Skip Connections

Goal: As models get deeper, due to its complexity, ***saturates and degrades in performance***. Thus, having an identity connection will allow the deeper model to perform only as well or better than its counterparts.

Residual connection: Adds with identity

Dense skip connection: Concatenates with identity

Note: https://arxiv.org/pdf/1712.09913.pdf

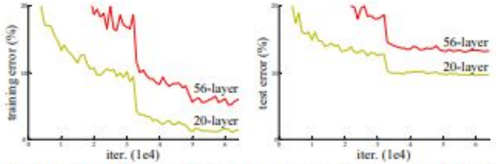
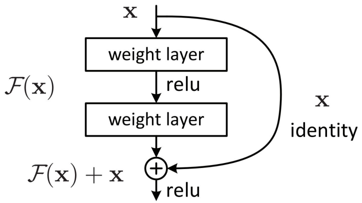
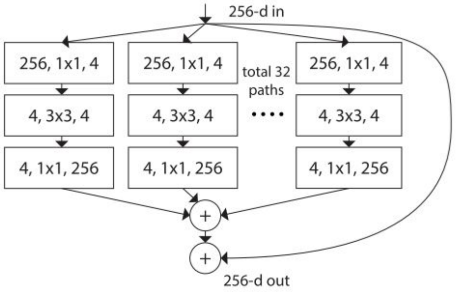
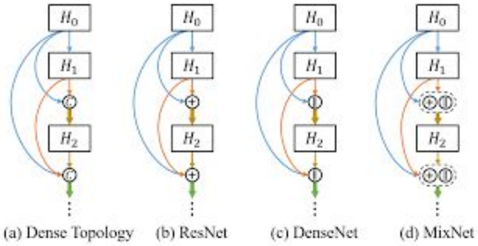
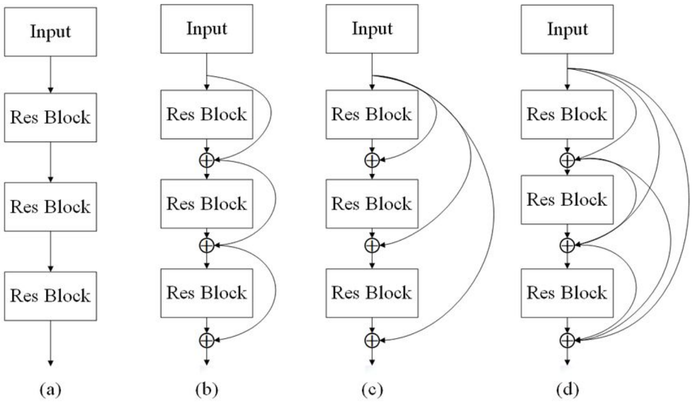


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.









Transfer Learning

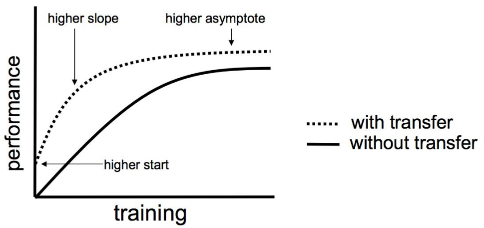
Goal**:** Using ***pre-trained models*** for your application to get a head start.

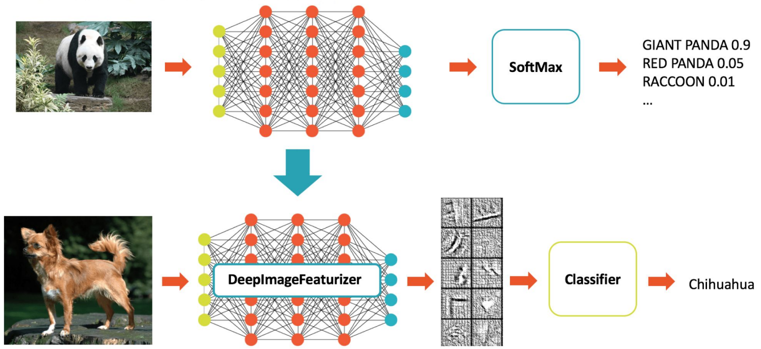
Note: Typically we only use the first part of the model, otherwise known as the "body".

Fine-Tuning the pretrained model:

Freeze the pre-trained body and train the "head", the untrained part (Stage 1)

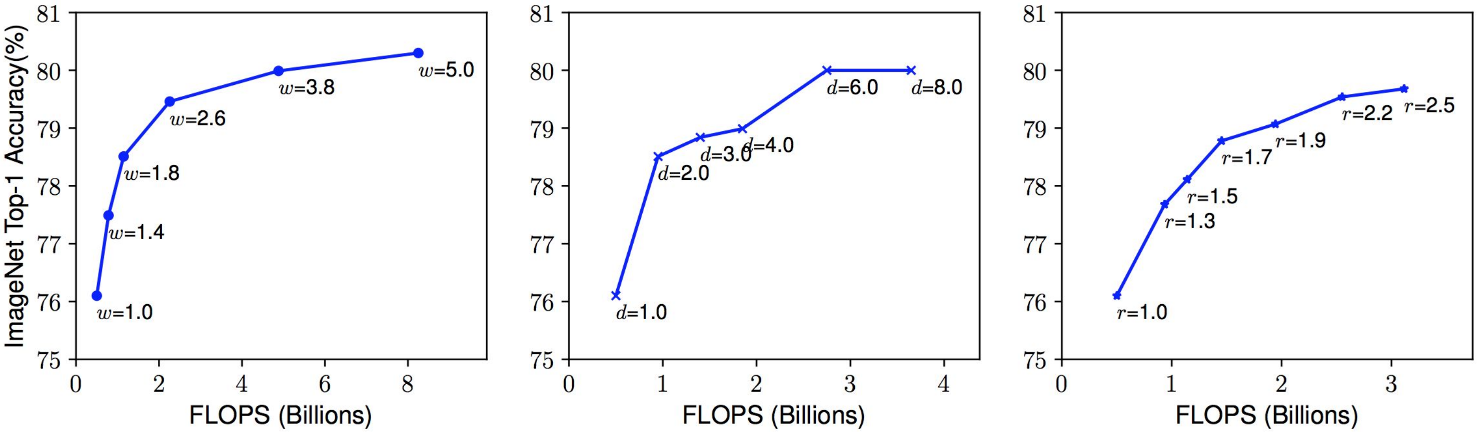
Unfreeze the body and train the entire model (Stage 2)





EfficientNet (Currently SOTA)

Paper: https://arxiv.org/abs/1905.11946



What’s next?

Explore papers

- Often with SOTA performance, or well-received

Try new things

- Research and implement ideas on datasets