Training

Arthur J. Redfern
arthur.redfern@utdallas.edu
Mar 06, 2019
Mar 11, 2019

Outline

- Motivation
- Data
- Initialization
- Forward pass
- Error calculation
- Backward pass
- Weight update
- Evaluation
- Hyper parameter selection
- Case study
- References

The 1 learning hypothesis of the brain …

C. Metin and D. Frost, “Visual responses of neurons in somatosensory cortex of hamsters with experimentally induced retinal projections to somatosensory thalamus,” PNAS Neurobiology, p. 357-361, 1986.
-> Taught the somatosensory cortex to see in hamsters

A. Roe et. al., “Visual projections routed to the auditory pathway in ferrets: receptive fields of visual neurons in primary auditory cortex,” Journal of Neuroscience, p. 3651-3664, 1992.
-> Taught the auditory cortex to see in ferrets

Motivation

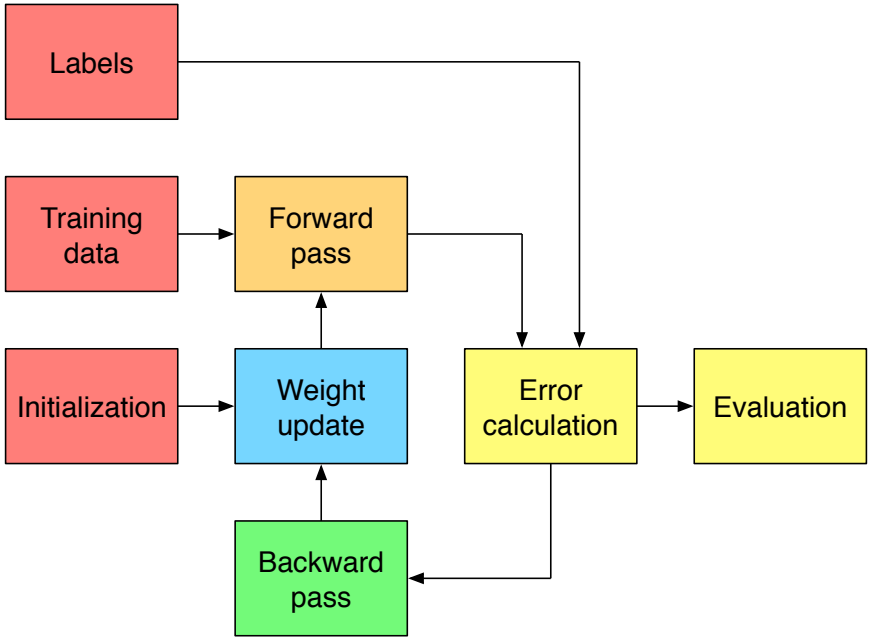
Parameter Estimation To Maximize Accuracy

- The previous lectures looked at a lot of xNN designs
- The goal of training is to find the parameters that control the input output mapping of the (typically) linear operations in a xNN design to maximize the accuracy of the full network on testing data
- Conceptually, extracting information from training data to enable the extraction of information from testing data

Example of doing this as a human in this class
- We look at a lot of different networks, I label the different parts and describe how they work together to allow the network to do what it does
- From this you start to build knowledge in network design (training)
- You read a new paper with a new network design
- In the new paper you recognize the different parts and recognize new innovations and how they interact to allow the network to do what it does (testing)

Supervised Learning Framework

- Hyper parameter selection
- Initialization
- Training
  - Update (serial or parallel)
    - Training data selection
    - Forward pass
    - Error calculation
    - Backward pass
    - Weight update
    - Repeat (~ batch)
  - Evaluation / validation
    - Validation data accuracy
    - Break if appropriate
  - Repeat (~ epoch)
- Testing
  - Evaluation / testing
    - Testing data accuracy

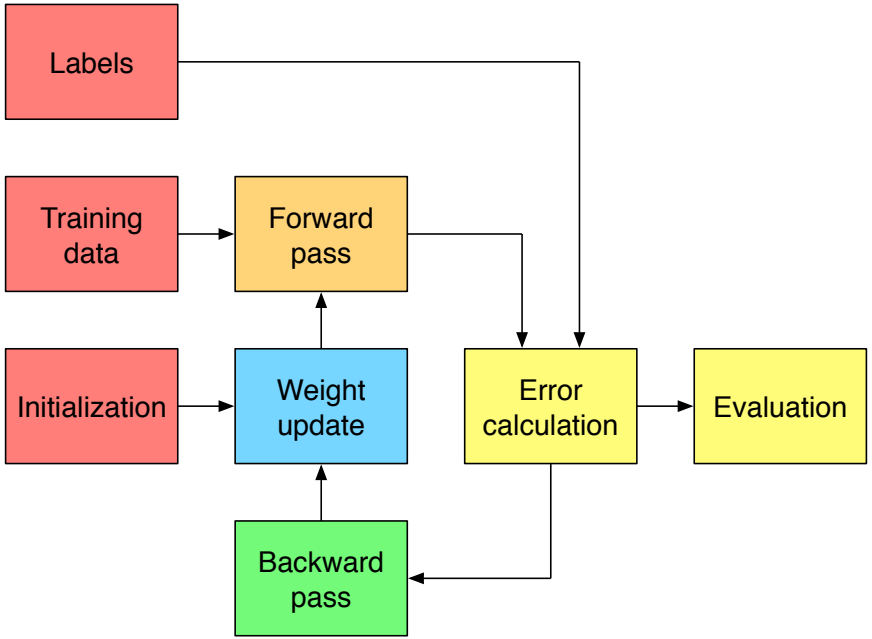


We've seen much of this already

xNN Training Vs Function Optimization

Training builds on the optimization framework discussed during the calculus lectures but there are meaningful conceptual and practical differences

- Differences
  - Different data sets
  - Different initialization strategies
  - Forward pass modifications
  - Different error calculations
  - Backward pass modifications
  - Different weight updates
  - …
- Will look at each of these in subsequent slides



Training a 3-node neural network is np-complete
https://papers.nips.cc/paper/125-training-a-3-node-neural-network-is-np-complete.pdf

Overfitting Regularization And Generalization

All definitions are informal; see https://developers.google.com/machine-learning/glossary/ for a general glossary of terms

- Generalization is the ability of a model (network) to make accurate predictions on testing data given training on training data
  - Generalization gap is the difference in accuracy of the network on training data vs testing data
  - Differences between training and optimization lead to a potential issue with generalization
- Overfitting refers to a model optimized on training data in a way that fails to nicely generalize to testing data
  - xNNs are highly parameterized models that while excellent in their universal approximation capabilities have the downside of potentially overfitting training data
  - Understanding deep learning requires rethinking generalization
    - https://arxiv.org/abs/1611.03530
- Regularization modifies training to improve generalization
  - A number of different regularization strategies will be described in subsequent sections

Overfitting Regularization And Generalization

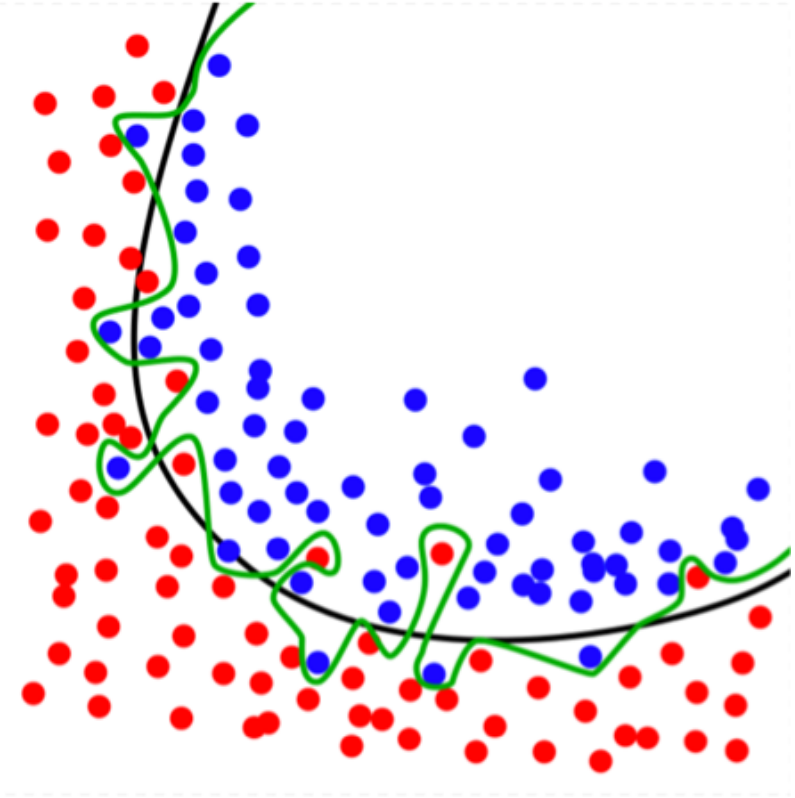


Figure from https://en.wikipedia.org/wiki/Overfitting#/media/File:Overfitting.svg

Overfitting Regularization And Generalization

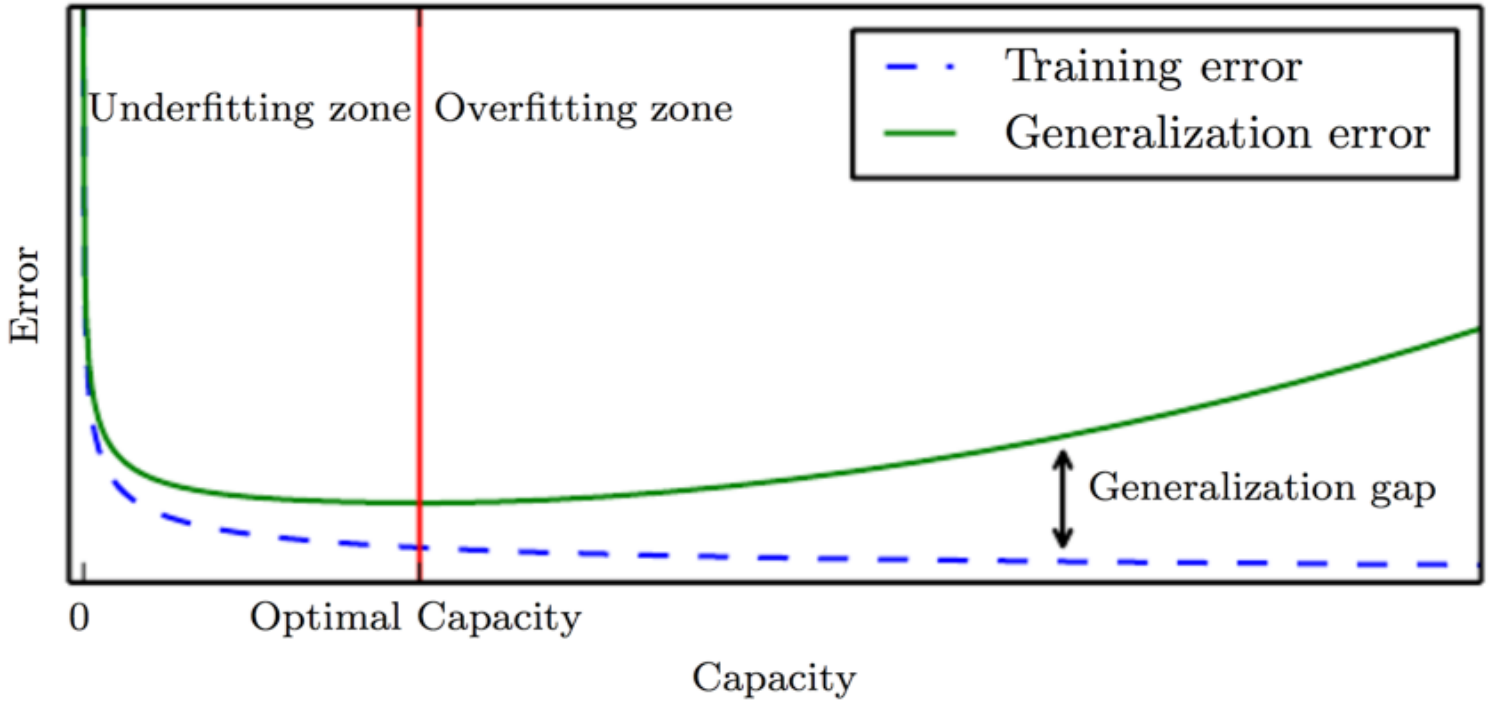


Figure from https://www.deeplearningbook.org

Convergence And Generalization

- Given the previous slides, on subsequent slides you will notice methods serve 1 or more purposes
  - Improve convergence
  - Improve generalization

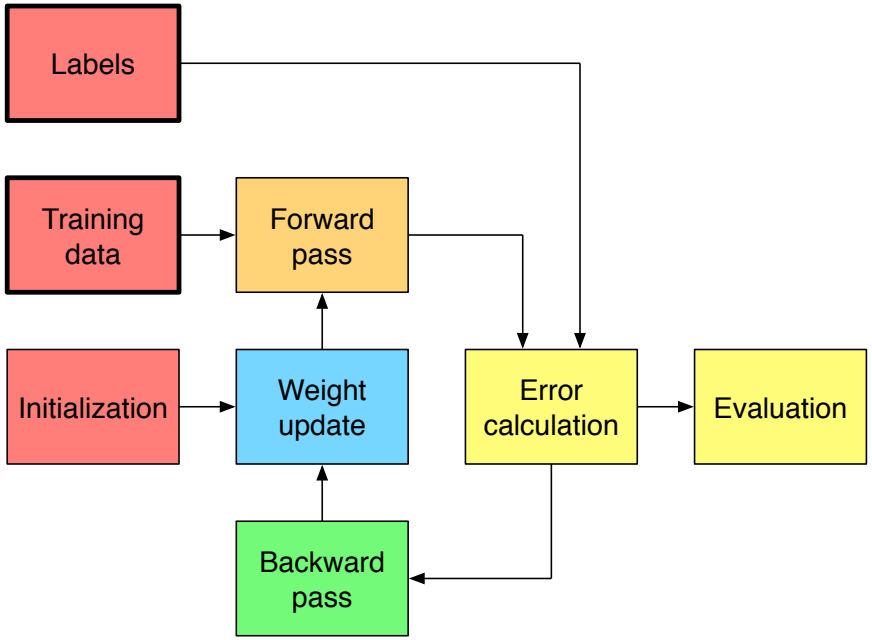
- Example: to improve convergence in very deep networks
  - Use optimal distribution for weight initialization
  - Add batch norms after convolutions
  - Use residual connections

- Keep this in the back of your mind as you continue reading

Data

xNN Training Vs Function Optimization

- xNN training
  - 1 data set used for training
  - Typically a different data set used for validation
  - Definitely a different data set used for testing
- Function optimization
  - Same data set used for training and optimization
  - Generalization is less of / not an issue

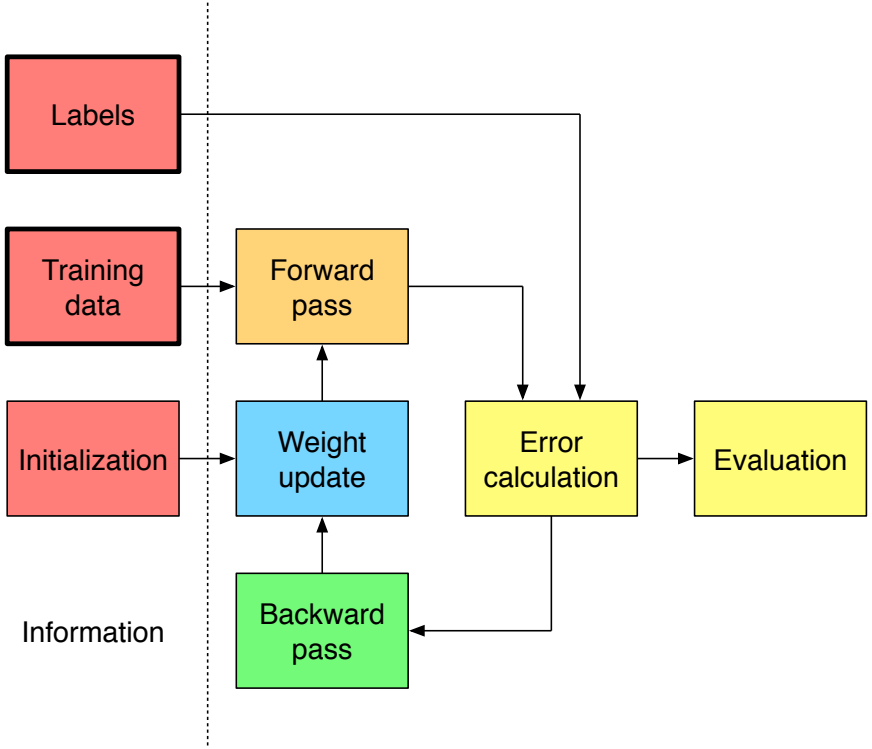


Different Training And Testing Data

- Mental picture
  - Take a function f(**x**) with a huge domain **x**
  - Think of training samples as a few points in the domain and the associated function evaluations
    - $\{\mathbf{x}_0, f(\mathbf{x}_0)\}$, ..., $\{\mathbf{x}_{T-1}, f(\mathbf{x}_{T-1})\}$
  - Ultimately, we want to find a network that approximates f(**x**) for all **x** in the domain from the samples in the domain
- So the samples matter a lot
- As does the ability to take the samples and create an appropriate mapping

Data Contains Information

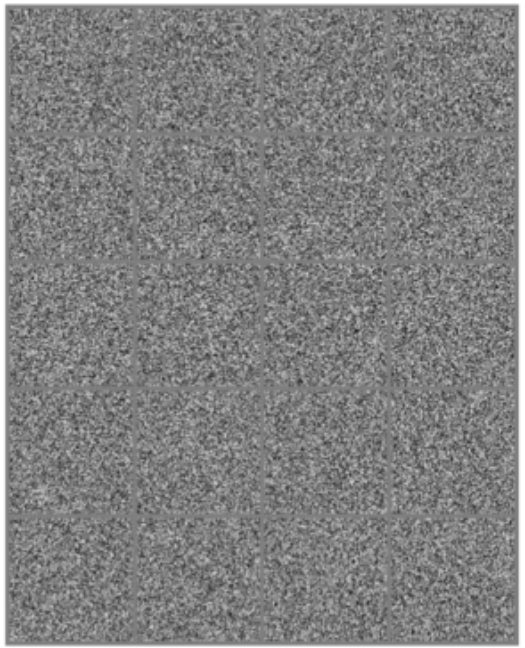
- Impossible to overstate the importance of data (really information) to learning
- For training
  - The more data the better
  - Want good representations of all possible classes / types
  - Want in as many settings as possible
  - Want as similar to the testing data as possible
- Note that initialization weights potentially also contain information (they will be discussed in the next section)



The Curse Of Dimensionality

- The curse of dimensionality: available data for training is sparse relative to all possible data realizations
  - Consider images
  - Consider sounds
- But we can train networks using sparse training data to work on testing data for many cases that we care about: how is this possible?
  - Natural <images, sounds, ...> live on a much smaller dimensional subspace than all possible
  - Successful applications of machine learning are likely possible because of this and it's frequent exploitation via some time, space, spatial frequency, ... in the data

20 random images
(that look nothing like natural images)



Question: how many different possible 8 bit images of size 3 x 1024 x 2048 are there?
Answer: $256^{6000000}$ (a big number)

Figure from https://www.deeplearningbook.org

Training | Validation | Testing Data Split

- Reminder: training data ∩ testing data = ∅
  - Train on training data
  - Test on testing data
- Training and validation data
  - Training data is used for weight updates
  - Validation data is used to periodically monitor progress during training
  - Typical strategy: split training data into training data and validation data
  - Variations on a theme (amount of split, what data to include where, ...)
- Testing data
  - Use to estimate final performance
  - Hidden danger: repeated passes through the flow make effectively make testing data part of training data
  - This is a very real problem for xNNs with lots of parameters and testing many different network and / or training configurations

xNNs vs other ML methods

- It's common in many machine learning training methods to have a small amount of training data and do cross validation (over repeated trials choose different partitions of the training data for training and validation)
- 1 problem with this for xNNs is that xNNs typically don't train well with a small amount of data in the 1st place
- Another problem is that the high capacity of xNNs makes memorization possible which hurts the use of validation data for determining when to stop training
- This is related to the hidden danger mentioned under testing data

2 Classes Of Training Data

Natural

Synthetic

Figures from https://arxiv.org/abs/1605.06457

Natural Data

- Natural data takes advantage of nature's data generation process
  - Doesn't require information on our part as nature supplies the intelligence
  - Natural as used here applies to data generated or measured from a physical environment by a person
  - Ex: images, sounds, radar, lidar, ultrasound, EEG, ...
- It's nice if there's relative uniformity of data generated by diff sensors of the same type
  - This allows for training data taken from 1 sensor to be used to train weights for a network that processes a different sensor
  - It typically implies that the data can it be transformed to a common representation (e.g., image sensor and an ISP)
  - Different sensor types can have different common representation (e.g., RGB image, point cloud, ...)
- A challenge is labeling
  - The number of samples needed for training large network is large
  - Potentially the complexity of labeling even a single sample is high

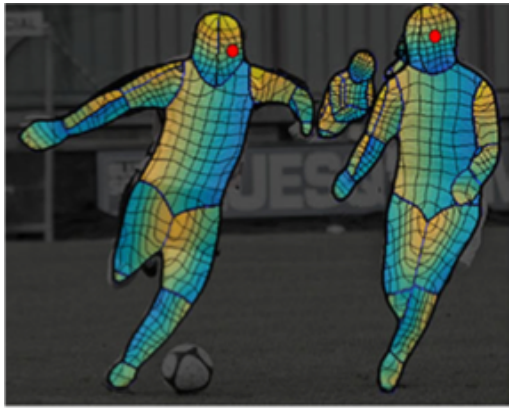
Labeling Data (Is Miserable)

- xNNs like lots of data for training
- The complexity of labeling a single sample can vary a lot
  - Easy
    - 1 input 1 label a few classes
  - Moderate
    - 1 input 1 label many classes
  - Hard
    - 1 input many labels
    - Common to build tools to help
    - Streamline the labeling process
  - Impossible (-ish)
    - Depth after the fact

Figures from https://arxiv.org/abs/1604.01685

Labeling Data (Is Still Miserable)

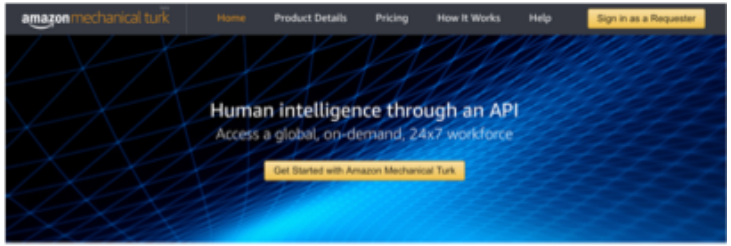
- xNNs like lots of data for training
- The complexity of labeling a single sample can vary a lot
  - Easy
    - 1 input 1 label a few classes
  - Moderate
    - 1 input 1 label many classes
  - Hard
    - 1 input many labels
    - Common to build tools to help
    - Streamline the labeling process
  - Impossible (-ish)
    - Depth after the fact



Figures from https://research.fb.com/facebook-open-sources-densepose/

Tools Money Deception And Coercion

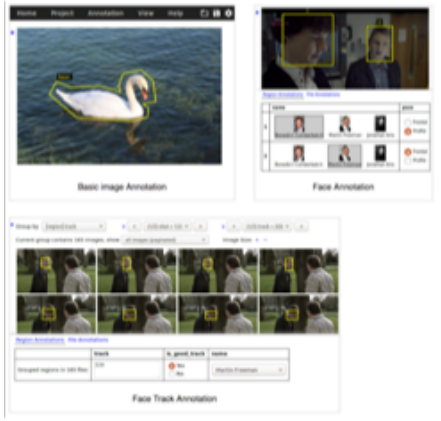
- In practice, build tools then use tools + humans to label
- How to get humans to label
  - Pay people (Mech Turk)
  - Coerce people (grad students, relatives)
  - Figure out a way of getting people to do it even though they don't realize they are (games, apps)
  - Realize it's not getting done as fast as you'd like and do it yourself



Amazon Mechanical Turk (MTurk) operates a marketplace for work that requires human intelligence. The MTurk web service enables companies to programmatically access this marketplace and a diverse, on-demand workforce. Developers can leverage this service to build human intelligence directly into their applications.

While computing technology continues to improve, there are still many things that human beings can do much more effectively than computers, such as identifying objects in a photo or video, performing data de-duplication, transcribing audio recordings or researching data details. Traditionally, tasks like this have been accomplished by hiring a large temporary workforce (which is time consuming, expensive and difficult to scale) or have gone undone.

MTurk aims to make accessing human intelligence simple, scalable, and cost-effective. Businesses or developers needing tasks done (called Human Intelligence Tasks or "HITs") can use the robust MTurk API to access thousands of high quality, global, on-demand Workers—and then programmatically integrate the results of that work directly into their business processes and systems. MTurk enables developers and businesses to achieve their goals more quickly and at a lower cost than was previously possible.



Figures from https://www.mturk.com, https://alpslabel.wordpress.com and http://www.robots.ox.ac.uk/~vgg/software/via/

The Dream

- The dream: training xNNs with unlabeled data (unsupervised learning)
  - There's usually orders of magnitude more unlabeled data than labeled data
- The reality: training xNNs tends to work a lot better with labeled data (supervised learning)
- Likely some tradeoff / information balance between unsupervised and supervised learning
  - Less data (info) + labels (info)
  - More data (info) but no labels

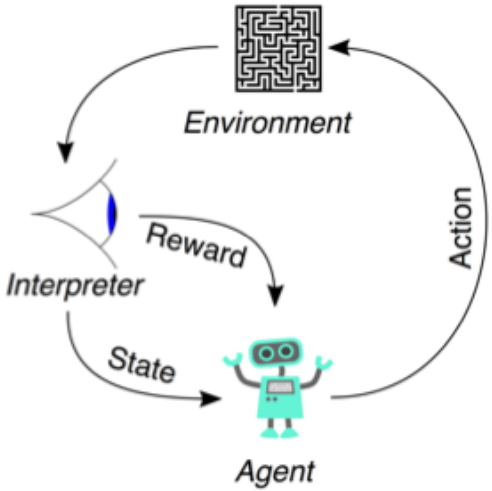
Figures from https://www.mturk.com, https://alpslabel.wordpress.com and http://www.robots.ox.ac.uk/~vgg/software/via/

Somewhere In Between …

- … unsupervised and supervised learning lies …

- Semi supervised learning (some labeled data, some unlabeled data)
  - Example: train multiple xNNs with labeled training data using supervised learning
  - Use unlabeled data as an input and treat the ensemble output label as the correct label
  - Use the ensemble output label with supervised learning to update the individual xNNs

- Reinforcement learning
  - From the current state an agent chooses an action and the environment provides a reward and new state; the combination of input {current state, action} and output {reward, new state} can be used to generate an error signal
  - But the output is frequently sort of 1 step removed from a label (e.g., what really is the value of the new position); so the information content in the output is typically not as strong as the supervised learning cases
  - As such, most reinforcement successes are linked to cases where huge amounts of simulated input output pairs are possible to test and smart strategies are used to approximate labels



Note: will discuss reinforcement learning more in the context of games

Figure from https://upload.wikimedia.org/wikipedia/commons/1/1b/Reinforcement_learning_diagram.svg

Cleaning

- If data is incorrectly labeled then training on it can negatively impact testing accuracy
- Cleaning: remove bad data
- Examples
  - Multiple people labeling the same data
  - 0 lag tick filtering (reminder: tell story)

$$E = mc^3$$

(if you were a physics student and learned this it would likely negatively impact your testing accuracy, i.e., grade; xNNs are no different)

Examples

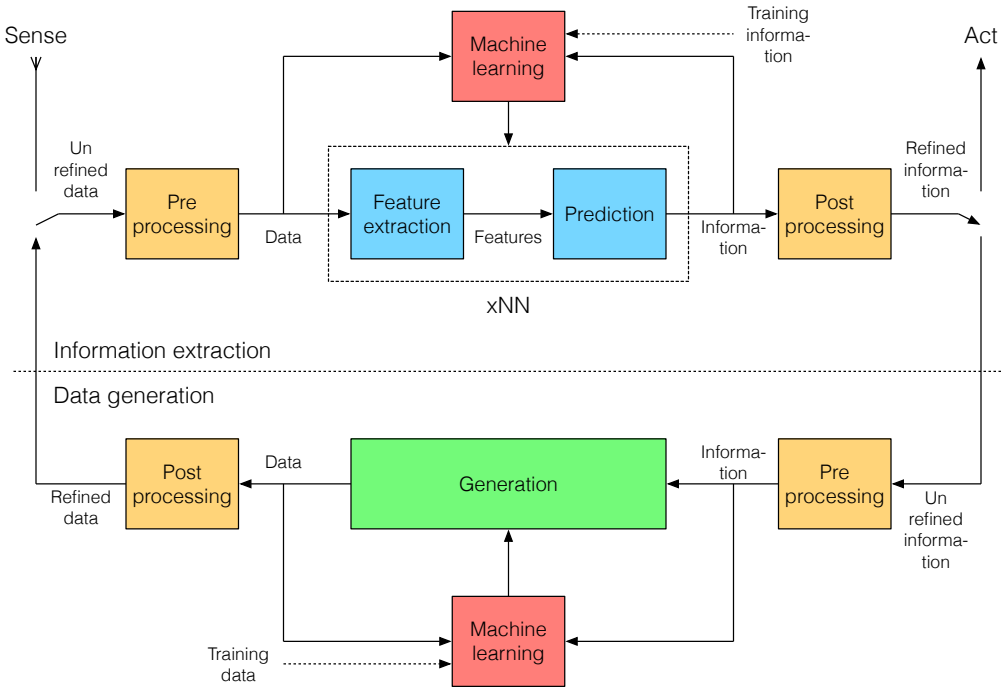
- Labeled classification datasets were created 1st for vision
  - Easiest task to outsource (e.g. Amazon Mechanical Turk)
  - Largest public dataset: ImageNet (1.2M images, 1000 classes)
  - Largest private dataset (that I know about): JFT (100M images, 15k classes)
- Pixel wise labeling can be very expensive
  - Segmentation example: Daimler Cityscapes
    - Fine annotations (5,000 images @ 90+ min each)
    - Coarse annotations (20,000 images @ 7 min each)
    - Total time: 9833 hours (24.5 weeks w / 10 full time people)
  - Other difficult examples: depth, motion, ...
- Example datasets for vision: MNIST, CIFAR 10 / 100, ImageNet, Pascal VOC, KITTI, COCO, Cityscapes, ...

Images from Microsoft COCO
http://cocodataset.org/

Synthetic Data

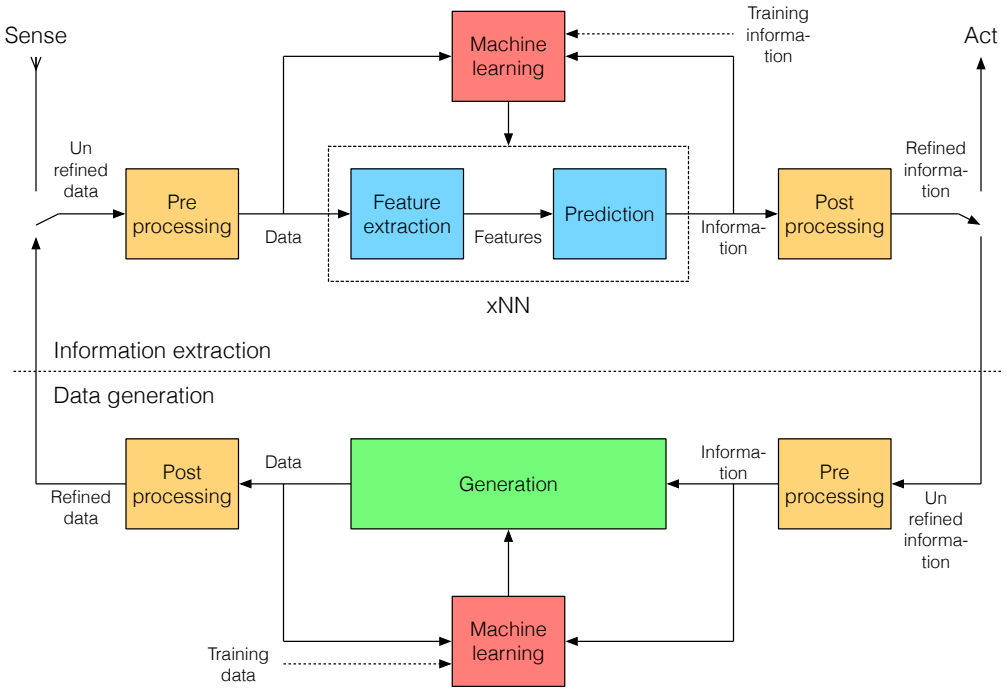
- Reminder
  - Introduction lecture mentioned that there are 2 types of problems
  - Data to information (basically everything we've talked about so far)
  - Information to data (relevant here among other places)

- Thinking
  - Labeling data is miserable
  - So instead invert the problem
    - Which may or may not be easier
  - Synthetic data: start from a label and use an algorithm to generate associated data
  - The data / label combination can then be used for supervised training



2 Ways To Generate Synthetic Data

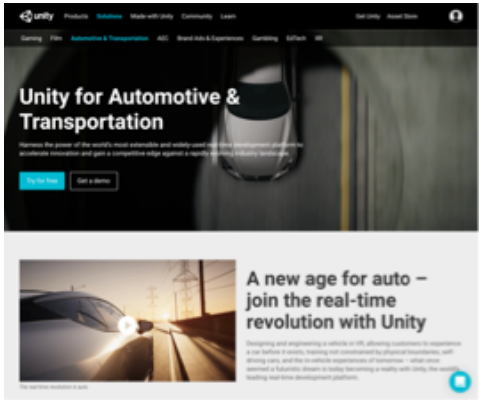
- Reminder
  - When we talked about methods for going from data to information there were 2 basic categories: hand engineered and learned
  - The same applies to going the other direction from information to data
- Hand engineered
  - Requires intelligence on our part
  - Ex: computer vision (thank you to the video game and movie special effects industries)
- Learned
  - Extracts knowledge from nature's information to data generation process to train an algorithm to generate data from information

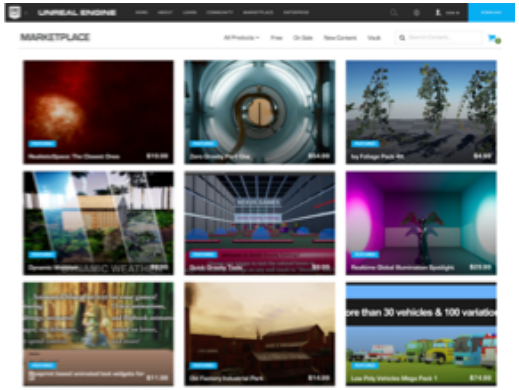


Hand Engineered

- The information content of the generated synthetic data is limited by the information content of the hand engineered model used to generate the data (but for some cases this is a lot)

- Examples
  - Computer graphics



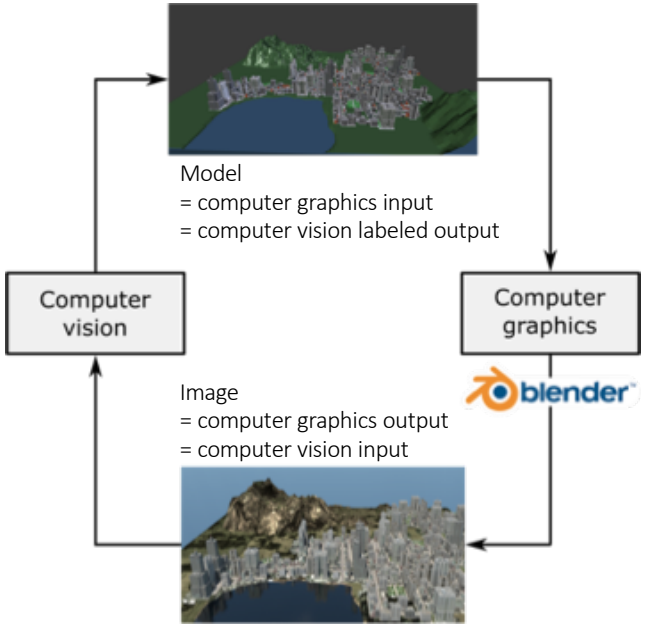


Figures from Unreal Engine https://www.unrealengine.com/ and Unity https://unity3d.com

Computer Graphics

- Computer graphics and computer vision are the opposites (cousins?) of each other
  - We can take advantage of advances in computer graphics from gaming, … to generate realistic synthetic data

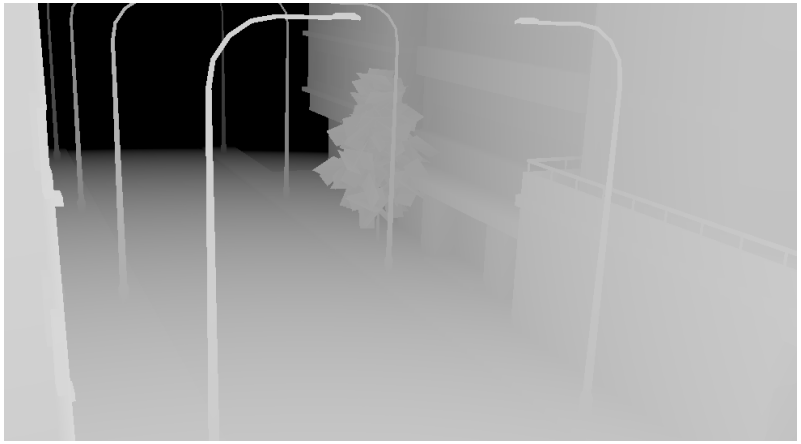
- Example uses
  - RGB images with lenses (industry standard, fisheye warped, …) and orientations (surround, stereo, …)
  - Depth and motion
  - Semantic and instance segmentation
  - Events in different conditions and unlikely events



For a recently developed plugin for Unreal Engine see the Nvidia deep learning dataset synthesizer (NDDS) available at https://github.com/NVIDIA/Dataset_Synthesizer

Synthetic Depth Data

Rendered image



Labeled depth

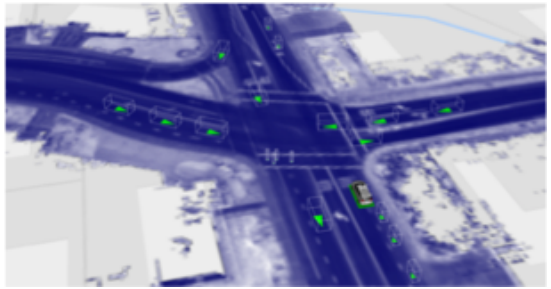
Example RGB and depth renderings
Dense depth labels are difficult to obtain in practice (e.g., KITTI is ~30% labeled)

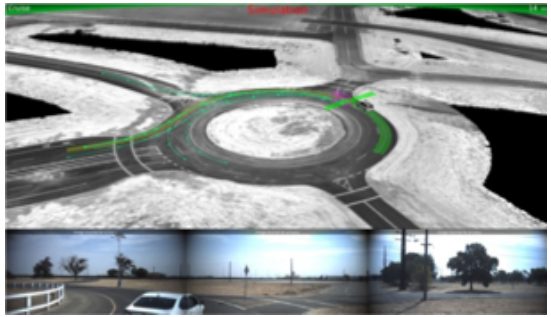
Synthetic Virtual KITTI Data

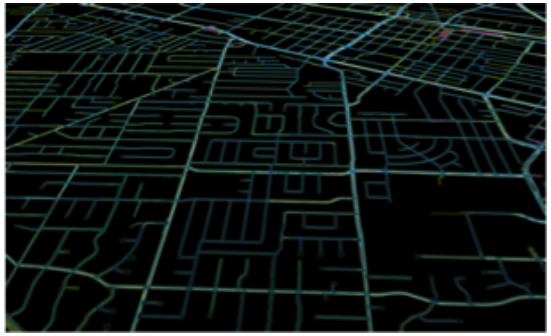
Virtual worlds as a proxy for multiple-object tracking analysis (https://arxiv.org/abs/1605.06457)

Synthetic Carcraft Data

- Google / Waymo Carcraft
  - > 8 million virtual miles driven per day
  - > 2.5 billion virtual miles diven total





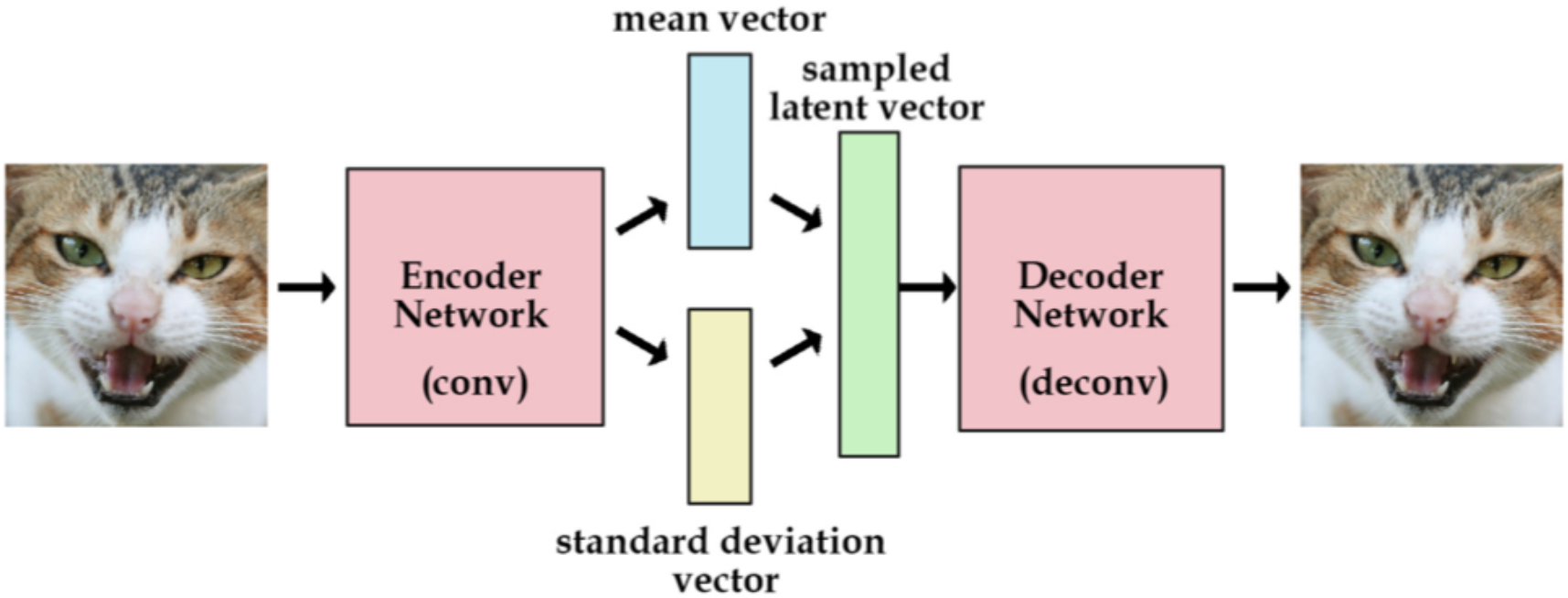


Images from https://www.theatlantic.com/technology/archive/2017/08/inside-waymos-secret-testing-and-simulation-facilities/537648/

Learned

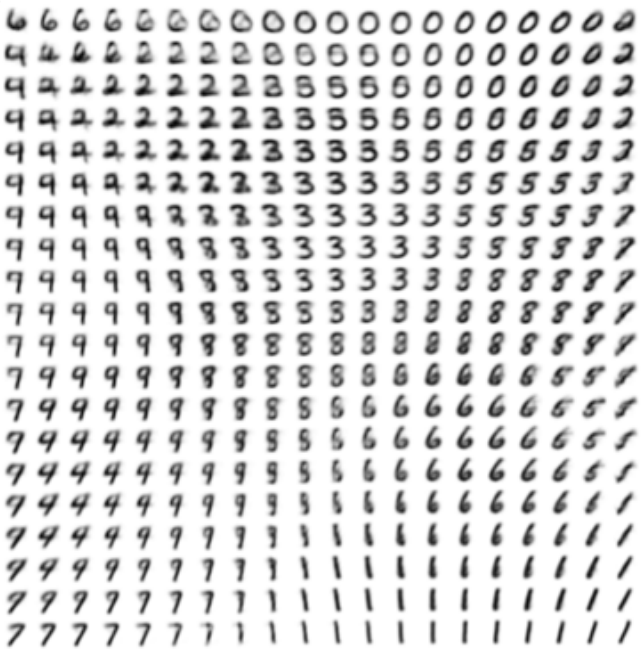
- Thought chain
  - Neural networks provide a structure for learning to map from data to information
  - So instead of a hand engineered mapping, is it possible to train a neural network to learn to map from information to data (i.e., the other direction)?
  - Subtlety: The information content of the generated synthetic data is limited by the information content of data used for training the algorithm for generating the data
- An auto encoder is an example structure that learns to recreate it's input after it’s input is pushed through a bottleneck
  - The bottleneck forces a representation that contains the key underlying features of the data
  - Can then start at the bottleneck and generate new data (e.g., see variational auto encoder)
- A generative adversarial network is an example that learns a generative model that generates examples similar to the characteristics of natural samples
  - Typically learned in conjunction with another algorithm for information extraction that determines if data is natural or synthetic

Ex: Variational Auto Encoder



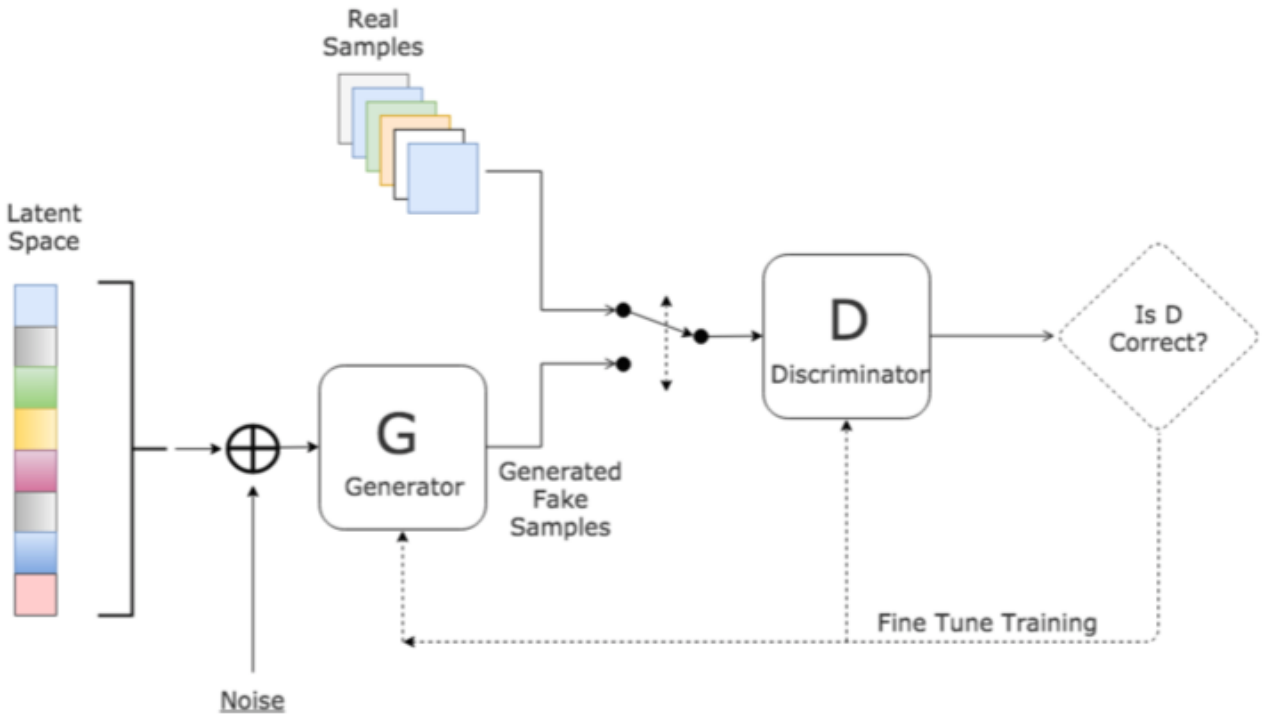
Variational autoencoders explained (http://kvfrans.com/variational-autoencoders-explained/)

Ex: Variational Auto Encoder



Auto-encoding variational Bayes (https://arxiv.org/abs/1312.6114)

Ex: Generative Adversarial Networks



Note: there's a lot of activity in this area and this topic may get it's own focused lecture later in the semester

Generative adversarial networks (https://www.kdnuggets.com/2017/01/generative-adversarial-networks-hot-topic-machine-learning.html)

Data Augmentation

- Basically always want more data to improve training
  - Generate new data by augmenting existing data via modifications
  - Target places where training data is insufficient for learning to achieve a desired level of accuracy on testing data
- Starting point can be real or synthetic data
  - Augmentation can be abstract
    - Warping, flips, color distortions, additive noise, …
  - Augmentation can be meaningful
    - Changing time of day or weather
- The amount of information added is based on the augmentation process
  - Even if 0 (e.g., noise or a deterministic invertible mapping) it's still potentially useful to training if it helps regularize the learning process

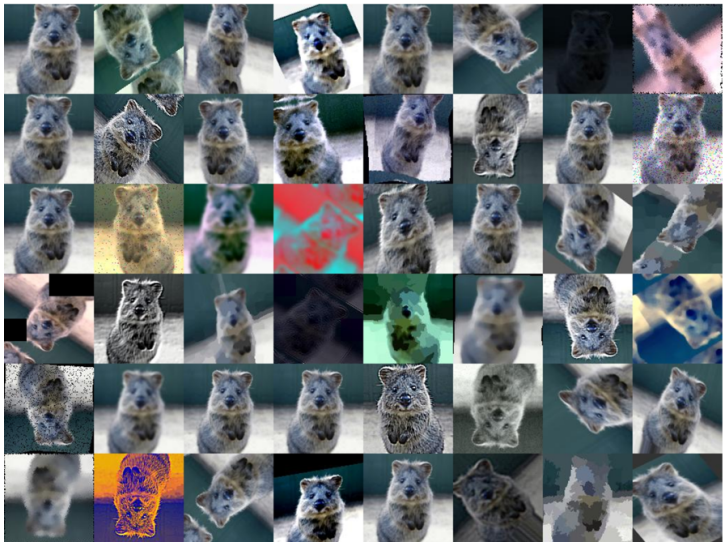
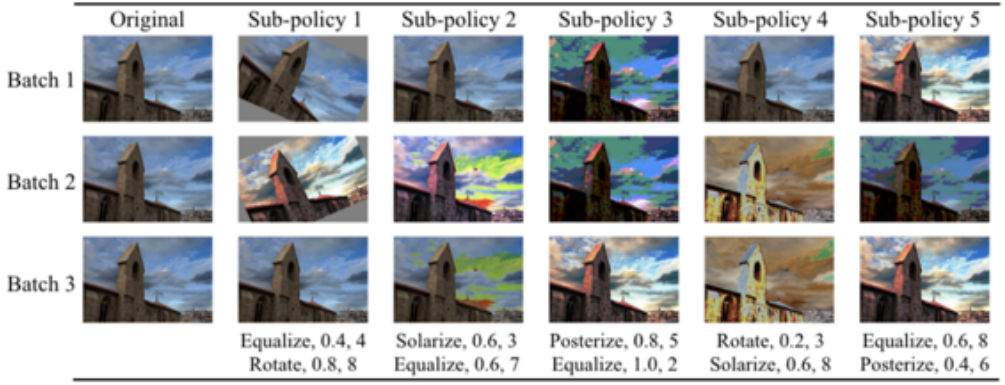


Figure from https://towardsdatascience.com/image-augmentation-for-deep-learning-using-keras-and-histogram-equalization-9329f6ae5085

Data Augmentation

- Auto augmentation
  - The design slides looked automated network design (neural architecture search)
  - The strategy of replacing human design with search / optimization can be applied to many parts of xNN design and training
  - Auto augmentation is this idea applied to data augmentation
- Discrete search problem to find the best augmentation policy
  - Each policy has 5 sub policies
  - Each sub policy has 2 operations
  - Each operation has a probability of application and a magnitude parameter
- Achieved state of the art results on a few different classification benchmarks



AutoAugment: learning augmentation policies from data
https://arxiv.org/abs/1805.09501
https://github.com/tensorflow/models/tree/master/research/autoaugment

Available operations: shearX/Y, translateX/Y, rotate, auto contrast, invert, equalize, solarize, posterize, contrast, color, brightness, sharpness, cutout and sample pairing

Figure from https://arxiv.org/abs/1805.09501

Synthetic Virtual KITTI Data

Clone

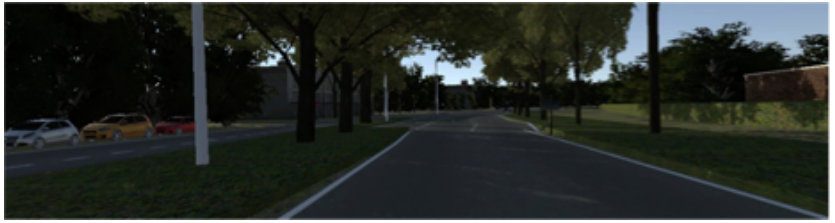
Overcast

Fog

Rain

Morning

Sunset



Adversarial Examples

- High capacity networks that map from data to 1 of many classes can be easily fooled
  - An input that humans are confident in correctly classifying that looks like a typical member of a class
  - But a network is confident in incorrectly classifying to a totally different class
- These types of inputs are called adversarial
- Why create adversarial examples?
  - The good: to use as a data augmentation method for generating additional training samples to make a network more robust
  - The bad: to fool a network into making an incorrect decision for a negative purpose

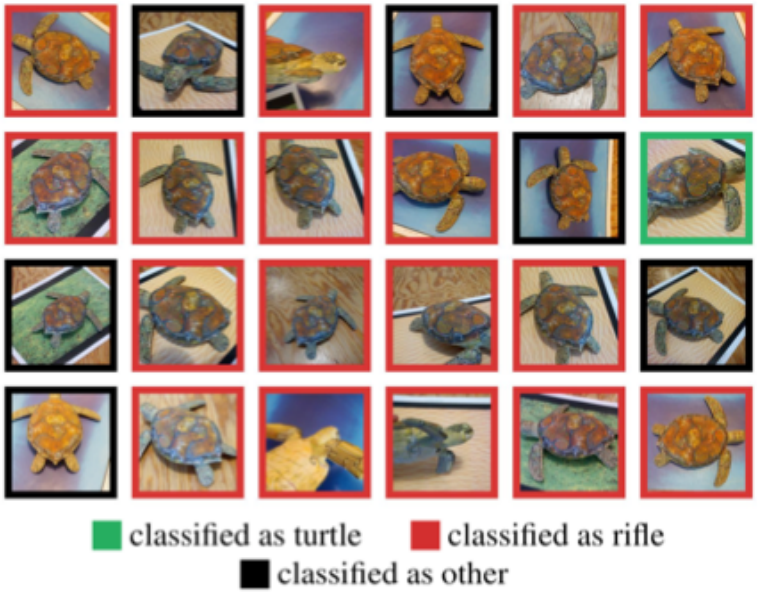


Figure from https://arxiv.org/abs/1707.07397

Generating Adversarial Examples

- Goal: generate examples with the following characteristics
  - Human strongly believes generated data belongs to class A
  - Information extraction algorithm strongly believes generated data belongs to class B
- Strategy
  - Input natural image from class A
  - Input information extraction algorithm
  - Input human perception sensitivity information
  - Use optimization algorithm to compute perturbation of natural image from class A based on the information extraction algorithm and human perception sensitivity that achieves the goals

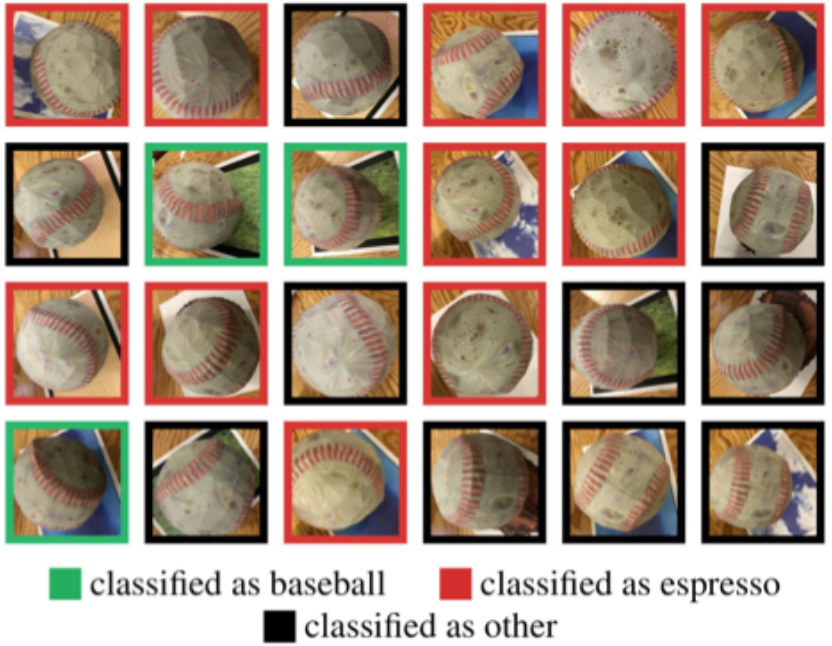


Figure from https://arxiv.org/abs/1707.07397

Selecting Samples Of Training Data

Note: batch size will be discussed more during the weight update section

- Batch / iteration / epoch
  - Batch size: a group of inputs (a batch) used to make a single weight update
  - Number of iterations: number of weight updates
  - Number of epochs: batch size * number of iterations / number of training samples

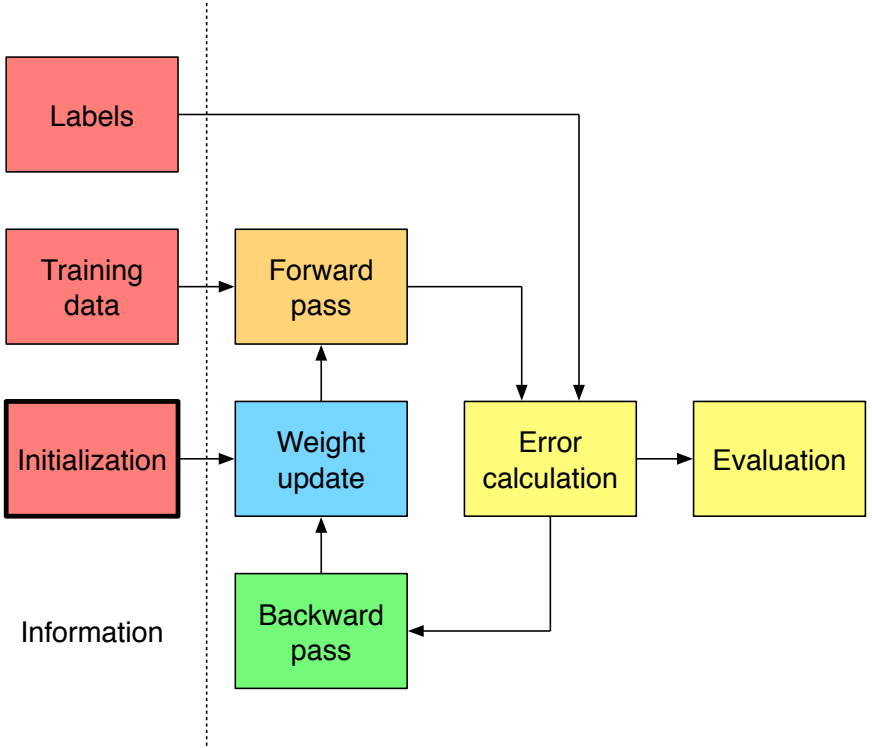
- Options for selecting training samples
  - Walk through a random ordering of all training data a batch at a time (maybe randomize each epoch)
  - Start with easier samples then move to harder samples (curriculum learning)
  - Force some level of balance in different samples
  - Force some bias in different samples
  - More difficult cases vs easy cases (online hard example mining, importance sampling)
  - For subsequent epochs can keep the same ordering or switch to a new random ordering (sometimes a hassle due to practical data preparation constraints)

For some new results on importance sampling as applied to xNN training see: Not all samples are created equal: deep learning with importance sampling (https://arxiv.org/abs/1803.00942)

Initialization

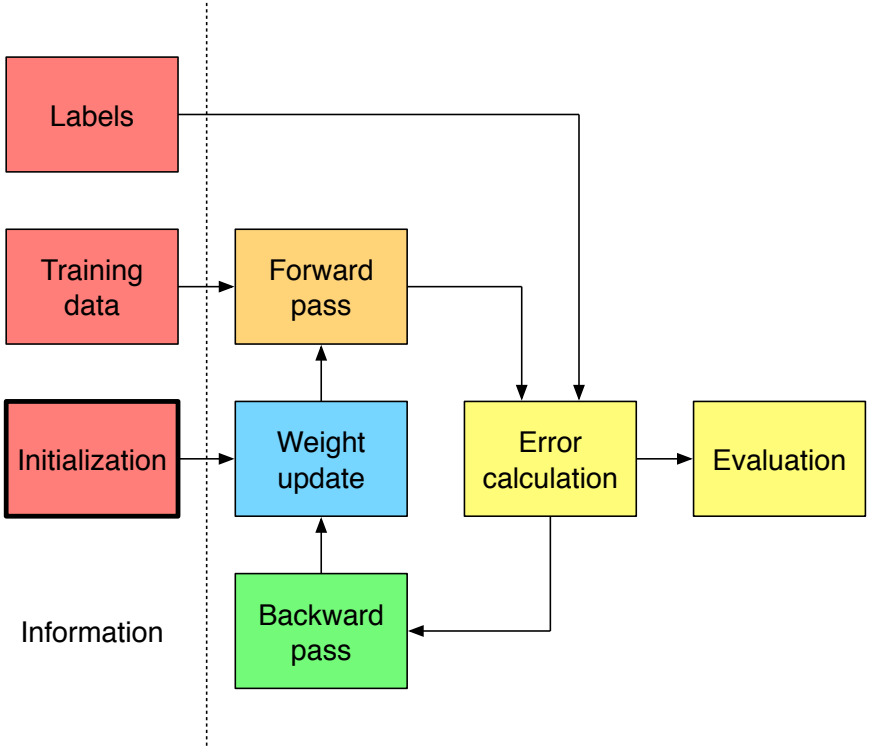
A Little Bit Of Historical Context

- Weight initialization and nonlinearity choice used to be a very active focus area for training deep neural networks (achieving good convergence)
  - During this time networks tended to be ~ 20 layers or less
- Then batch norm and residual connections “happened”
  - Basically everyone began using them to train very deep neural network designs
  - Weight initialization and nonlinearity choice fell out of favor from a research perspective
  - Learned to train networks with a practically arbitrary number of layers



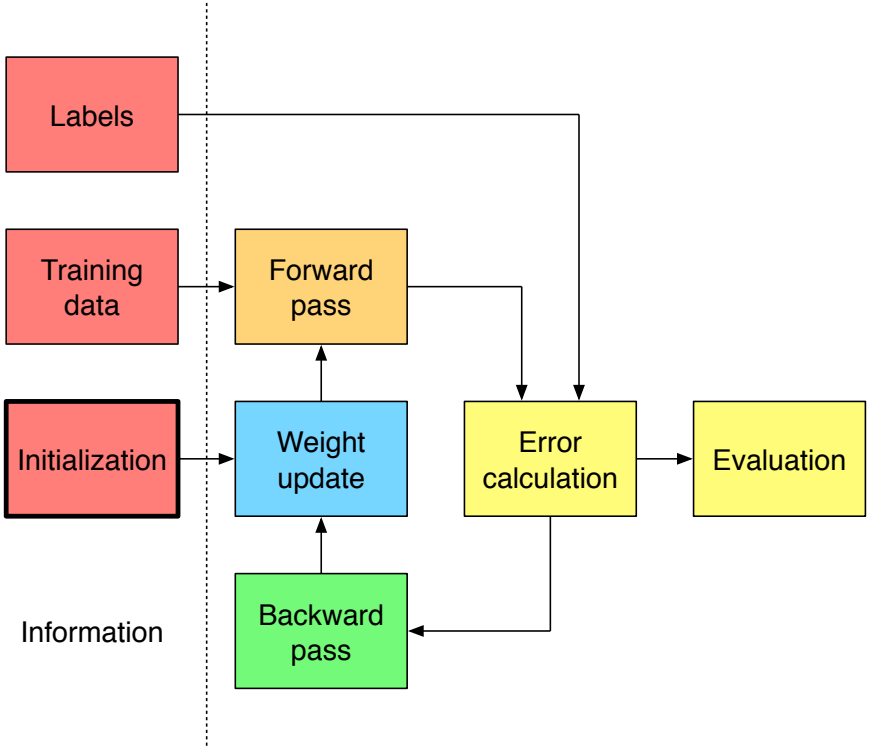
A Little Bit Of Historical Context

- Recent work is revisiting weight initialization and nonlinearity choice for training very deep neural networks without the need for normalization or residual connections
  - The emergence of spectral universality in deep networks
    - https://arxiv.org/abs/1802.09979
    - Develops a framework for understanding weight initialization and network nonlinearities that leads to fast training
    - Points to orthogonal weight initialization vs Gaussian
    - Points to smoothed variants of ReLU vs ReLU
  - Dynamical isometry and a mean field theory of CNNs: how to train 10,000-layer vanilla convolutional neural networks
    - https://arxiv.org/abs/1806.05393
    - Trained extremely deep CNNs without residual connections or batch normalization
    - Using orthogonal weight initialization and tanh nonlinearity



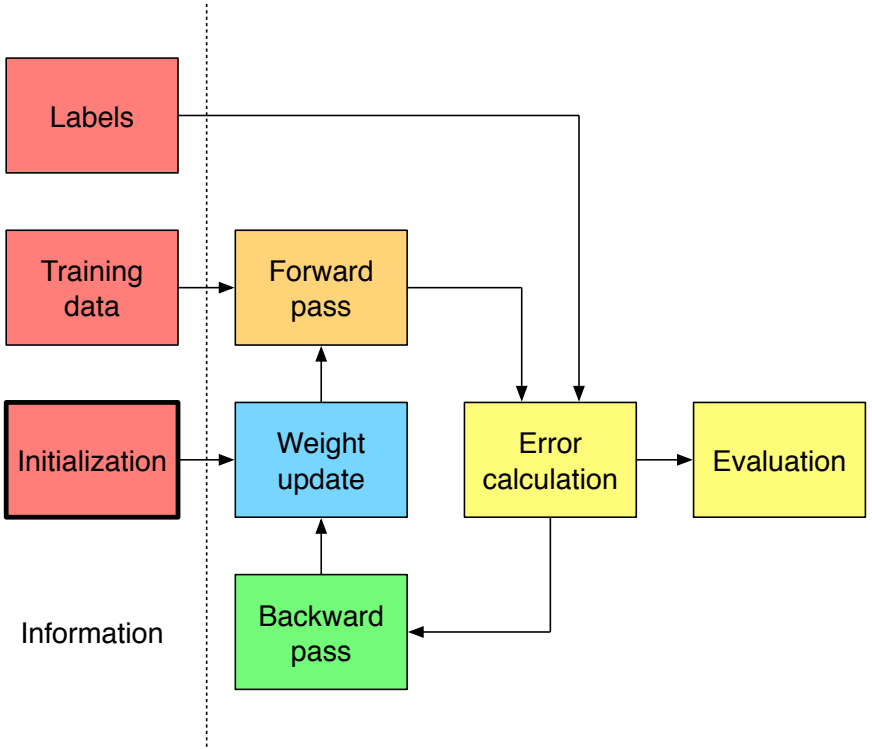
A Little Bit Of Historical Context

- Recent work is revisiting weight initialization and nonlinearity choice for training very deep neural networks without the need for normalization or residual connections
  - Fixup initialization: residual learning without normalization
    - https://arxiv.org/abs/1901.09321
    - Ok … so this paper put residual connections back in :)
    - Rescales standard initialization to prevent exploding and vanishing gradients at the start of training
    - Shown to work on networks with 10,000 layers
    - Convenient as batch norm is a bit of a pain to deal with from training to testing, even more so when quantization is considered



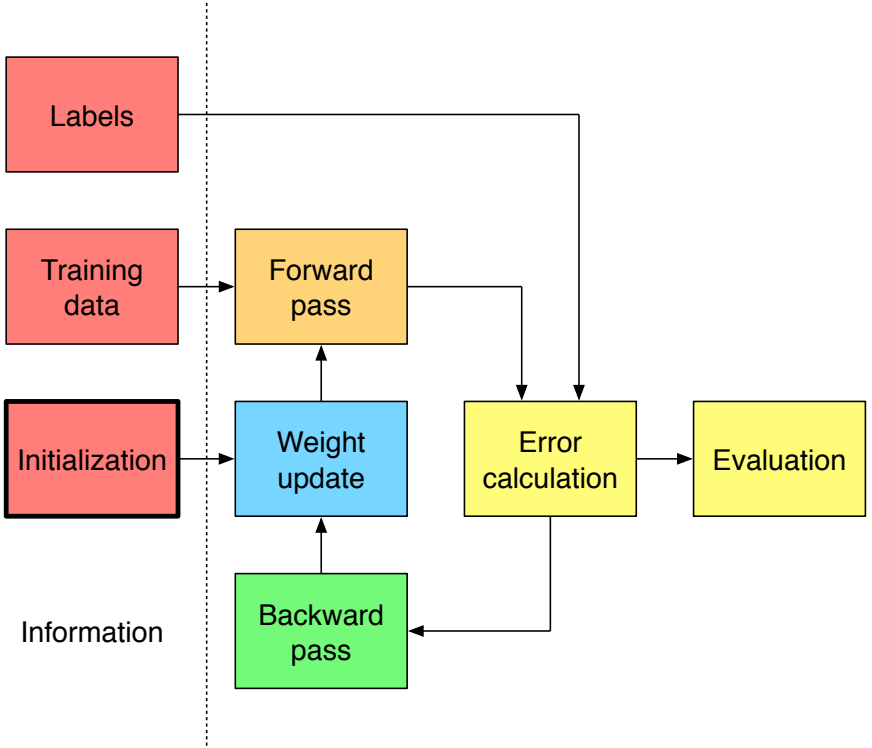
A Little Bit Of Historical Context

- So where do things stand now?
  - The starting point for designing and training deep networks is still residual connections, normalization and ReLU nonlinearities (at least for vision)
- Residual connections
  - Will probably continue to be used as recent research suggests that they have a smoothing affect on the loss surface making convergence easier
- Normalization
  - Will probably continue to be used
  - But batch normalization may be replaced with a different normalization strategy (more on this in a later section)
- Nonlinearity choice
  - Recent work in NAS points to accuracy improvements using smoothed nonlinearities
  - This was also observed historically with nonlinearities like the PReLU which could vary from linear to nonlinear
  - It's possible that it becomes popular to see more variants in a chase for accuracy, though ReLU is nice from a 0 inducing perspective for implementations



Strategies

- Random initialization
  - When you only know things probabilistically
- Transfer learning
  - From a different problem
  - From a simpler version of the same problem (curriculum learning)



End of abbreviated history lesson and back to initialization material

Random Initialization

- Training a model “from scratch”
- Typically use random initialization
  - Don’t choose all zeros
  - Do use multiple probably independent realizations of a random variable
  - Maybe handle weights different than biases
- Need to determine
  - Distribution type
  - Mean, variance, …

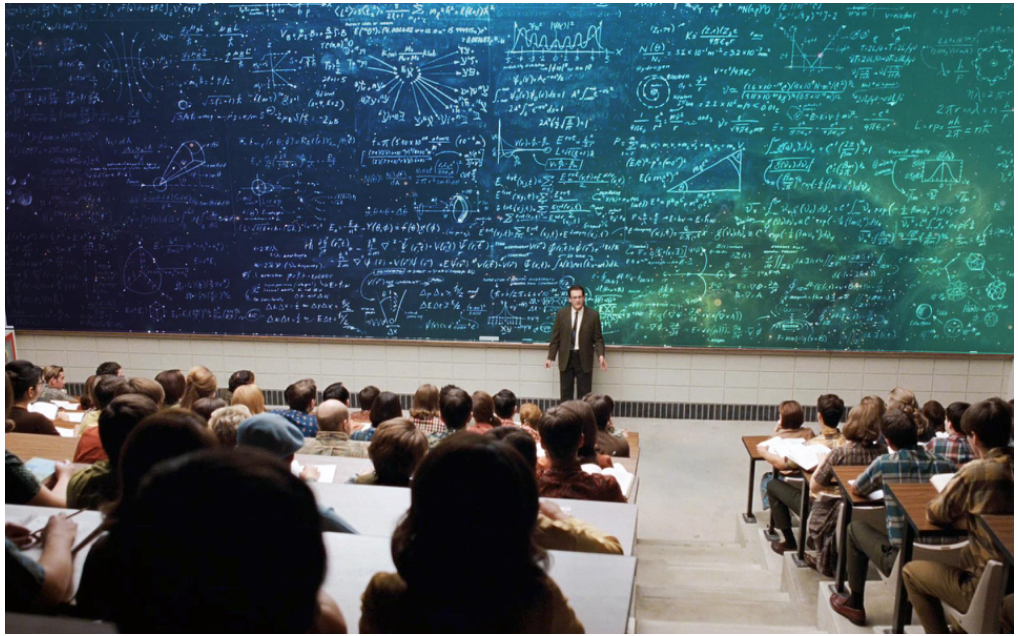


Figure from http://radscreens.com/i/9709/

How To Compute $2^{1/2}$ In 5th Grade

- Algorithm
  - $x = 2^{1/2}$
  - $x^2 = 2$
  - $0 = x^2 - 2$
  - $0 = -\alpha(x^2 - 2)$
  - $x \leftarrow x - \alpha(x^2 - 2)$

- Comment
  - Ideally, you'd like to choose an initialization that's close and in a contractive basin around the true value
  - Practically, when in a million dimensions with a non convex cost function, you're happy to choose a starting point that doesn't lead to divergence

Let $\alpha = 0.1$ and $x_0 = 1.50$ (convergence)

$x_1 = 1.48$
$x_2 = 1.46$
$x_3 = 1.45$
$x_4 = 1.44$
…

Let $\alpha = 0.1$ and $x_0 = 15$ (divergence)

$x_1 = -7$
$x_2 = -12$
$x_3 = -28$
$x_4 = -104$
…

Info Theory And Distribution Choice

- What does selecting a Gaussian distribution mean from an information theoretic perspective?
  - Max entropy distribution when only the mean and variance are known
- What does selecting a uniform distribution mean from an information theoretic perspective?
  - Max entropy distribution when only the domain of support is known
- Question: which is the correct / best choice?
  - What distribution do trained models tend to follow?

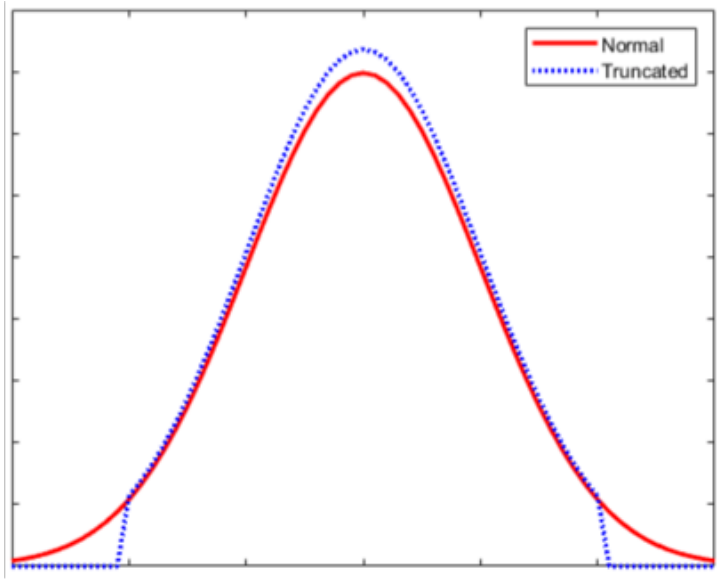


Figure from https://www.mathworks.com/help/stats/prob.normaldistribution.truncate.html

Mean = 0, Standard Deviation = ?

- Understanding the difficulty of training deep feedforward neural networks
  - http://proceedings.mlr.press/v9/glorot10a.html
  - Derived a distribution referred based on an assumption of linear activations and no exploding or vanishing data
  - Glorot / Xavier uniform: [-limit, limit] where limit = sqrt(6/(fan_in + fan_out)) and fan_in and fan_out are the number of input and output units in the weight tensor (paper is for NN, for CNN unclear if fan_in = N_i*F_r*F_c or N_i and fan_out = 1 or N_o)
  - Glorot / Xavier Gaussian: 0 mean and stddev = sqrt(2/(fan_in + fan_out))

- Delving deep into rectifiers: surpassing human-level performance on ImageNet classification
  - https://arxiv.org/abs/1502.01852
  - Follow a similar derivation strategy as Xavier but take the nonlinearity into account and target keeping magnitudes of inputs constant (no exponential growth or shrink of signals)
  - He uniform: [-limit, limit] where limit = sqrt(6/(Fr*Fc*Ni))
  - He Gaussian: 0 mean and standard deviation sqrt(2/(Fr*Fc*Ni)) and biases initialized to 0 (per paper)

Transfer Learning

- Strategy
  - Train the network on a related problem that typically has a lot of data
  - Use those trained parameters as the starting point for the network applied to the problem of interest which typically has less data
  - May need to modify the network head to account for the differences in the problem
- How transferable are features in deep neural networks?
  - https://arxiv.org/pdf/1411.1792.pdf
- Can also use a smaller network to initialize a larger network or a larger network to initialize a smaller network
  - Very deep convolutional networks for large-scale image recognition
    - https://arxiv.org/abs/1409.1556
  - Used a strategy to train a smaller model to initialize a larger model
  - While it helps in some cases, it requires extra training and potentially has issues with getting stuck in sub optimal local minima

How This Impacts You This Semester

- Training a ~ large model like ResNet 50 on ImageNet using a fast gaming system with 2x Nvidia GTX 1080 Ti GPUs can easily take a couple of weeks to a month
  - And it's very easy to mess up and have to experiment a few times with hyper parameters to get it to work
- Transfer learning let's you take advantage of the work someone else did training the network to give you a jump start as you apply it to a different but related enough problem

Figures from https://github.com/tensorflow/models/tree/master/official and https://pytorch.org/docs/stable/torchvision/models.html

But Is ImageNet Pre Training Necessary?

- Consider the task of object based image segmentation on the MS COCO dataset
  - Does ImageNet pre training help?
- With respect to accuracy: surprisingly the answer seems to be **no**
  - Good training on the target task alone is sufficient, going against what has been conventional wisdom of using ImageNet pre training for basically all vision tasks
  - Note that MS COCO has a reasonable amount of data (vs Pascal VOC) and pre training can still help with speed
- Rethinking ImageNet pre-training
  - https://arxiv.org/abs/1811.08883

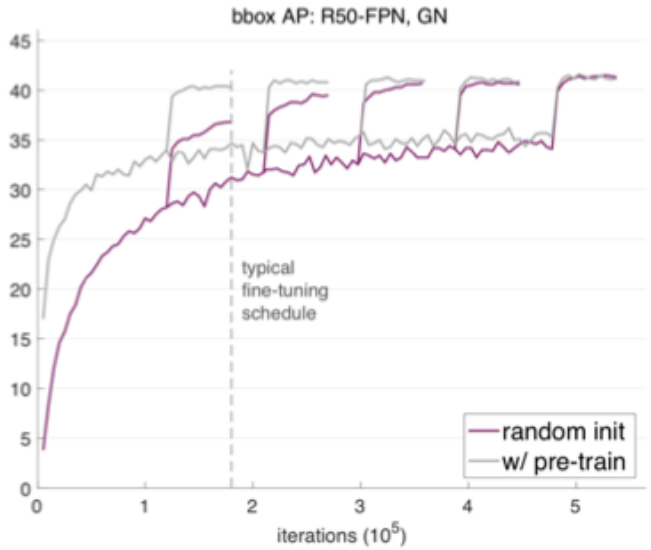
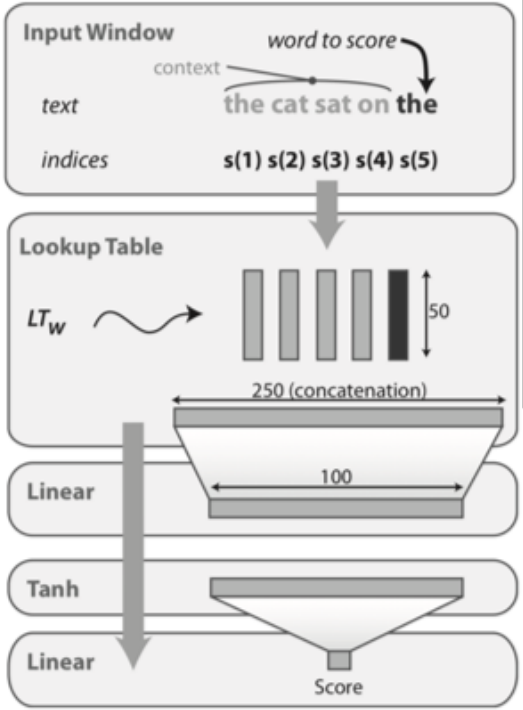
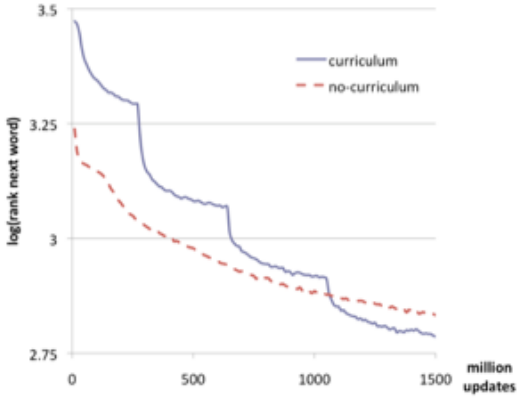


Figure from https://arxiv.org/abs/1811.08883

Curriculum Learning

- This sits 1/2 way between the data and initialization sections
- Strategy
  - Mimic how humans learn and orders training samples typically from easier to more difficult
  - Potentially improves accuracy and generalization
  - So it's a data selection method
  - But training on easier samples can be thought of as an initialization for subsequent training on more complex samples



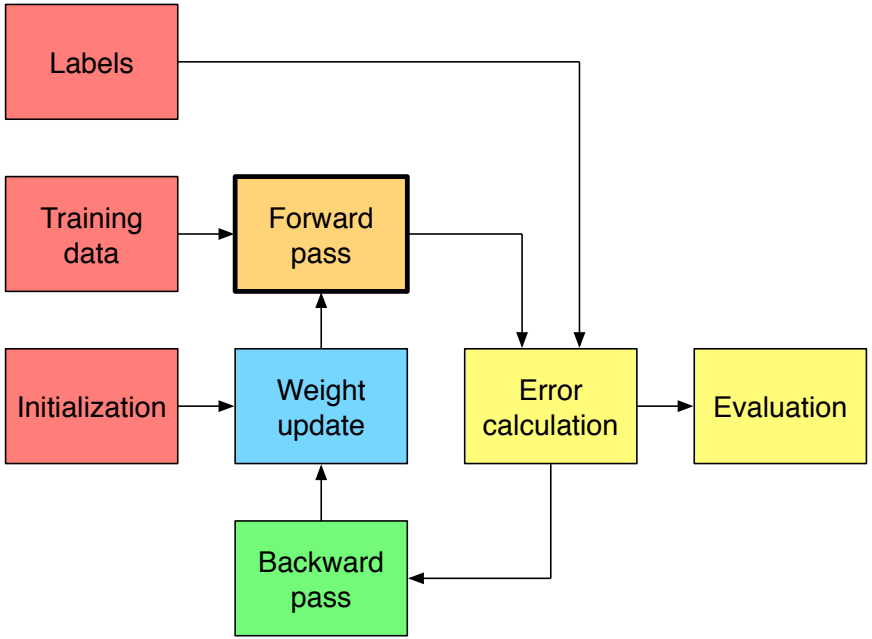


Curriculum learning (https://ronan.collobert.com/pub/matos/2009_curriculum_icml.pdf)

Forward Pass

xNN Training Vs Function Optimization

- Modifications to the forward pass for 2 reasons
  - Convergence: improve parameter estimation on training data
  - Regularization: improve performance on testing data



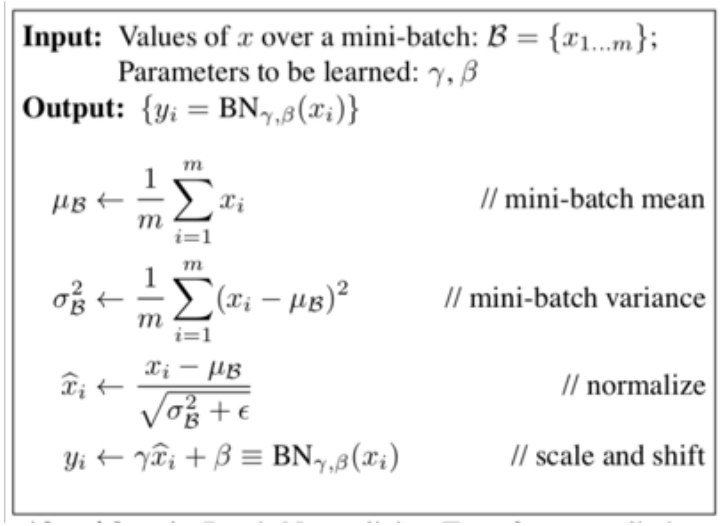
Convergence

- Strategy
  - Modify the network structure (forward pass) to improve convergence during training (backward pass)
  - After training, the network structure modification can frequently be absorbed into the original network
- Examples
  - Batch normalization
  - Batch renormalization
  - Group normalization

Batch Normalization

- Notes
  - Initialization played all sorts of games to prevent exploding or vanishing signals as they propagate through the network
  - Idea: why not add a data dependent layer after convolution that on a per channel basis normalizes the data to 0 mean unit variance?
  - The benefits of this were described in the paper Efficient backprop (http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf)
  - Note that the addition of this layer does not increase the set of functions the network can approximate in the forward path, it only improves training

- Batch normalization
  - For each batch compute a per channel mean and variance
  - Normalize each channel to ~ 0 mean ~ 1 variance
  - Includes learnable scale and shift parameters to maintain expressiveness in transformation
  - If possible absorb the scale and shift parameters into a neighboring layer during testing



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure from original paper

Batch Normalization

- How does batch normalization help optimization?
  - https://arxiv.org/abs/1805.11604
- Convention wisdom from the original paper
  - Reduces internal covariate shift (the change in input distribution to a layer based on an update of the previous layer)
  - Also: prevents exploding and vanishing gradients, improves robustness to learning rate and initialization and keeps the signal away from the saturation region of the nonlinearity
- Conclusion from the above paper
  - The link between batch norm and reducing internal covariate shift is tenuous at best
  - The improvement in training is because batch norm makes the loss surface significantly more smooth
  - There are additional methods for making the loss surface smooth that can be used
  - A principled exploration of these methods likely makes sense

Batch Renormalization

- Issues with batch normalization
  - Different operations during training and testing
  - If small batch size then large changes between batches

- Batch renormalization
  - Start with batch norm
  - Start accumulating running averages for the mean and variance
  - Then gradually transition from using the per sample mean and variance to the running average mean and variance

Input: Values of x over a training mini-batch $\mathcal{B} = \{x_{1...m}\}$; parameters γ, β; current moving mean μ and standard deviation σ; moving average update rate α; maximum allowed correction $r_{\max}, d_{\max}$.
Output: $\{y_i = \text{BatchRenorm}(x_i)\}$; updated μ, σ.

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_{\mathcal{B}} \leftarrow \sqrt{\epsilon + \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2}$$

$$r \leftarrow \texttt{stop\_gradient}\left(\text{clip}_{[1/r_{\max}, r_{\max}]}\left(\frac{\sigma_{\mathcal{B}}}{\sigma}\right)\right)$$

$$d \leftarrow \texttt{stop\_gradient}\left(\text{clip}_{[-d_{\max}, d_{\max}]}\left(\frac{\mu_{\mathcal{B}} - \mu}{\sigma}\right)\right)$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}}} \cdot r + d$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta$$

$$\mu := \mu + \alpha(\mu_{\mathcal{B}} - \mu) \qquad \text{// Update moving averages}$$

$$\sigma := \sigma + \alpha(\sigma_{\mathcal{B}} - \sigma)$$

Inference: $y \leftarrow \gamma \cdot \frac{x - \mu}{\sigma} + \beta$

Figure from original paper

Group Normalization

- Issues with batch normalization
  - Different operations during training and testing
  - If small batch size then large changes between batches
- Group normalization
  - Divide channels into groups
  - Compute mean and variance based on groups of channels

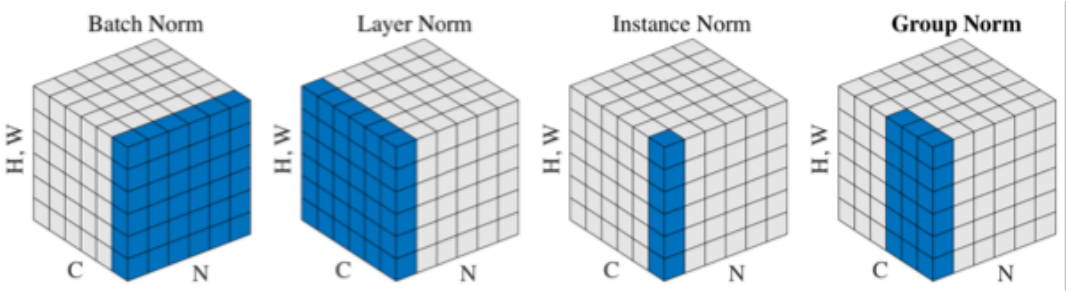
```
def GroupNorm(x, gamma, beta, G, eps=1e−5):
    # x: input features with shape [N,C,H,W]
    # gamma, beta: scale and offset, with shape [1,C,1,1]
    # G: number of groups for GN

    N, C, H, W = x.shape
    x = tf.reshape(x, [N, G, C // G, H, W])

    mean, var = tf.nn.moments(x, [2, 3, 4], keep_dims=True)
    x = (x − mean) / tf.sqrt(var + eps)

    x = tf.reshape(x, [N, C, H, W])

    return x * gamma + beta
```



C = channel dimension
N = batch dimension
H = feature map height
W = feature map width

Figures from original paper

Regularization

- It’s possible to modify the forward path to help with regularization

- Example network modifications to improve regularization
  - Stochastic width
  - Stochastic depth
  - Noise addition in the network

Stochastic Width (Dropout, DropConnect)

- The ideas behind stochastic width were most helpful in improving training with (multiple) large fully connected layers in earlier xNN designs
  - For wide networks this may be desirable
  - For narrow networks where the number of classes is on the order of or greater than the number of features this may be undesirable
  - Common current designs with global average pool and a single relatively smaller fully connected layer typically don't need to use stochastic width; but regardless, it's still a useful technique to know
- Dropout
  - Dropout zeros out a random set of layer outputs per batch
  - Implicitly it forces multiple groups of output features to be able to estimate a class
- DropConnect
  - DropConnect zeros out a random set of layer weights per batch
  - Implicitly it forces multiple groups of input features to be able to generate and output feature

Stochastic Width (Dropout, DropConnect)

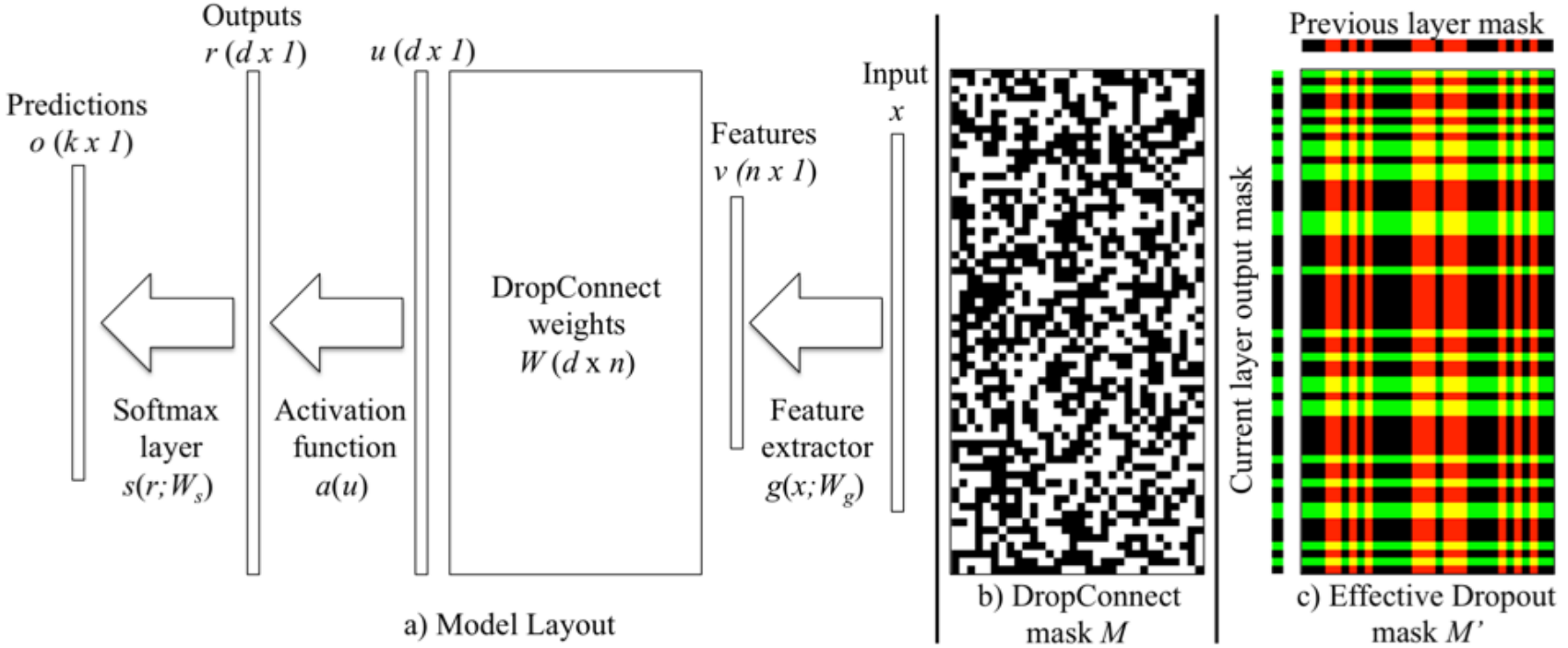


Figure from https://cs.nyu.edu/~wanli/dropc/dropc.pdf

Stochastic Width (DropBlock)

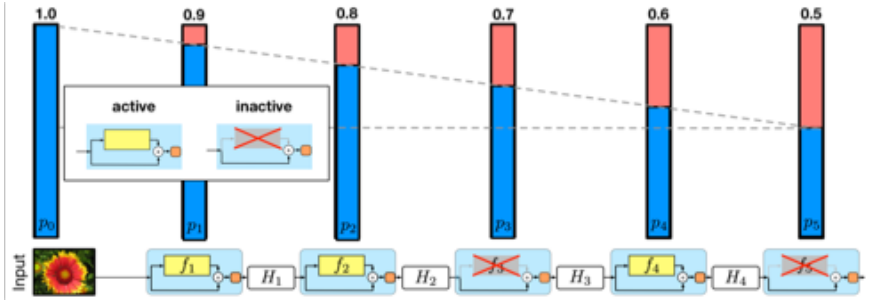
- An effective regularization method for CNN layers inspired by Dropout

- DropBlock: a regularization method for convolutional networks
  - https://arxiv.org/abs/1810.12890

Stochastic Depth (Layer Skipping)

- Stochastic depth (layer / building block skipping)
  - Only really used in very very deep residual networks
  - Implicitly it forces layers to do iterative refinement

- Notes
  - Probability of a layer being on or skipped (identity) is recommended to be layer dependent
  - Earlier in the net a layer is more likely to be on
  - Later in the net a layer is more likely to be bypassed
  - In testing a network output is scaled based on the probability that it was on during training



Note that H_n refers to refers to the feature map at the output of operation n, not the layer itself

Figure from https://arxiv.org/abs/1603.09382

Stochastic Branching (ShakeShake, ShakeDrop)

- ShakeShake was proposed to regularized ResNeXt training
  - https://arxiv.org/abs/1705.07485
- ShakeDrop extends the basic idea to additional ResNet variants and includes a stabailzing mechanism to improve convergence
  - https://arxiv.org/abs/1802.02375

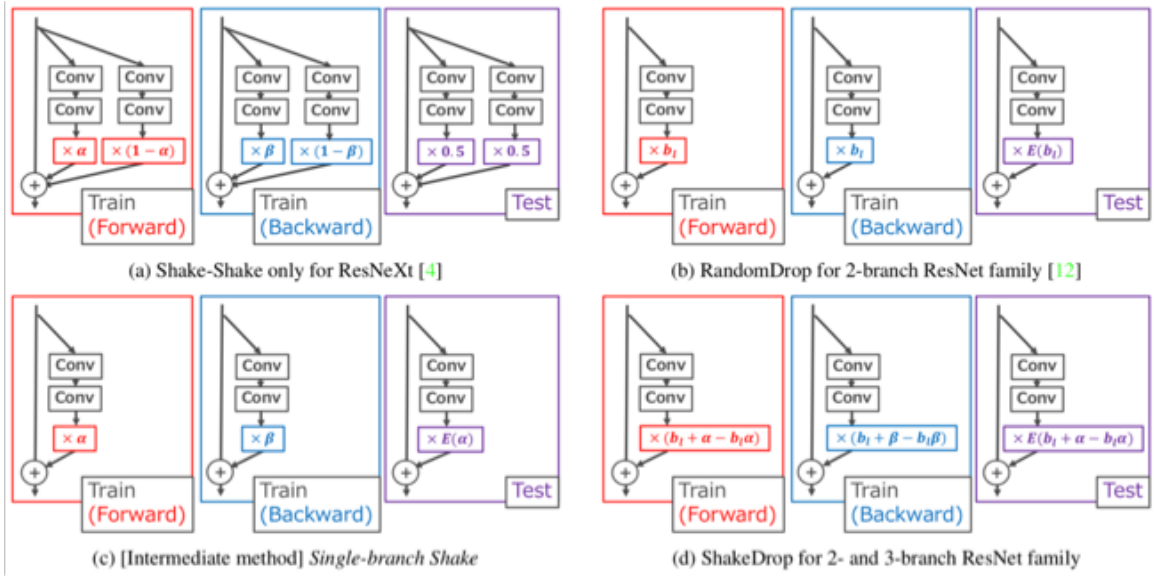


Figure from https://arxiv.org/abs/1802.02375

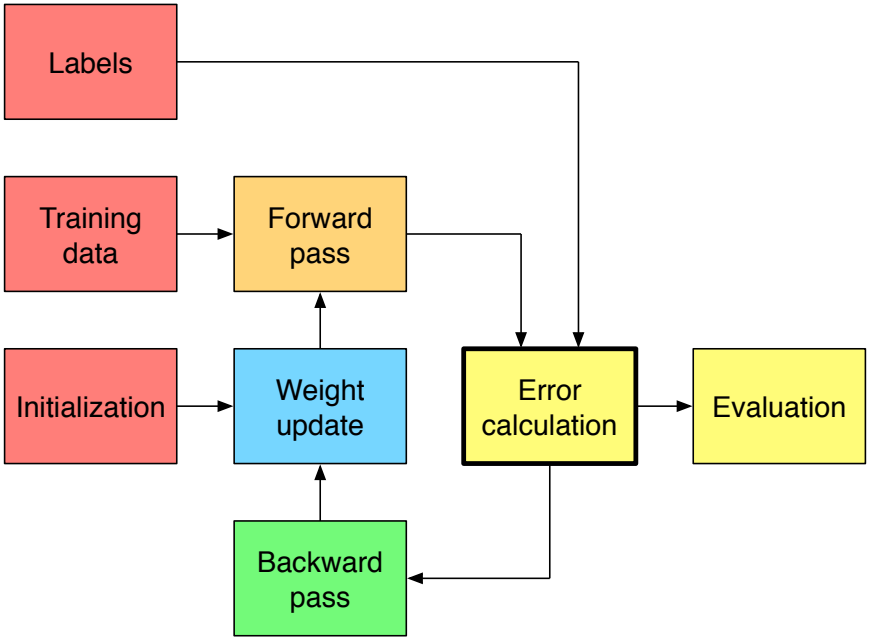
Noise Addition

- Noise addition in the network
  - Feature maps
  - ReLU 0 point
  - …
- Examples
  - Estimating or propagating gradients through stochastic neurons
    - https://arxiv.org/abs/1305.2982
  - Noisy activation functions
    - https://arxiv.org/abs/1603.00391
  - Dataset augmentation in feature space
    - https://arxiv.org/abs/1702.05538

Error Calculation

xNN Training Vs Function Optimization

- In the case of classification
  - Training with 1 loss (e.g., soft max – cross entropy)
  - Testing with a different loss (e.g., arg max)



2 Broad Categories Of Loss Functions

- Classification
  - Prediction in a finite series of classes
  - Typically …
    - Easier than regression
    - Complexity increases with number of classes
    - Complexity increases with similarity of classes

- Regression
  - Prediction in a continuum of values
  - Typically …
    - More difficult than classification
    - Complexity increases with range and required resolution

Classification Loss

- Distinct outputs (classes)
- Softmax cross entropy loss function
  - Soft max effectively converts network outputs to a probability mass function
  - KL divergence for comparing 2 probability mass functions: target and network output
  - For a 1 hot target probability mass function KL divergence reduces to cross entropy
  - The calculus lecture notes derived a nice form for the gradient of the error at the output of soft max cross entropy with the input feature map to soft max
- Other options are possible
  - KL divergence with label smoothing, noise or overconfidence penalization
  - Large margin softmax
  - Optimal transport
  - …

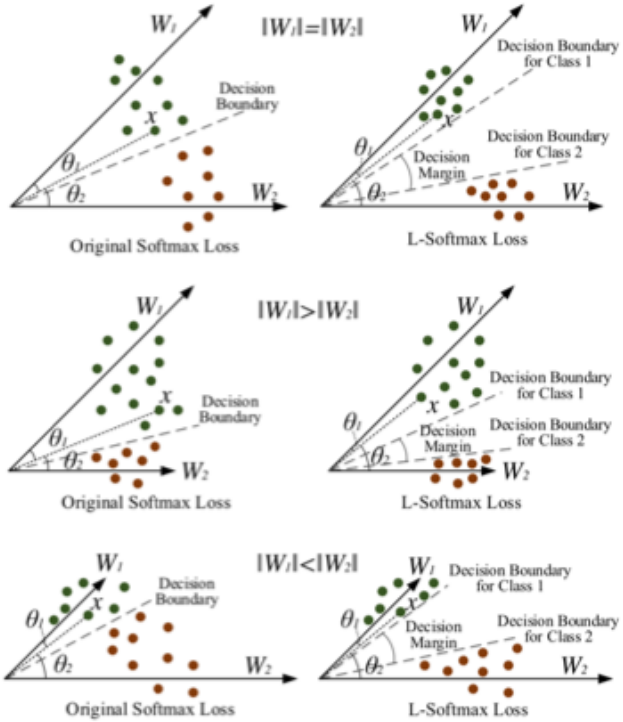


Figure from https://arxiv.org/abs/1612.02295

Regression Loss

- Continuous valued output
  - Strategy: if you can quantize the continuous range and use classification instead of regression it may be better
- Standard l_p norms
  - p = 1
  - p = 2
- Other options are possible
  - Huber loss / smooth L1 loss (like l_2 for small values and l_1 for large values)

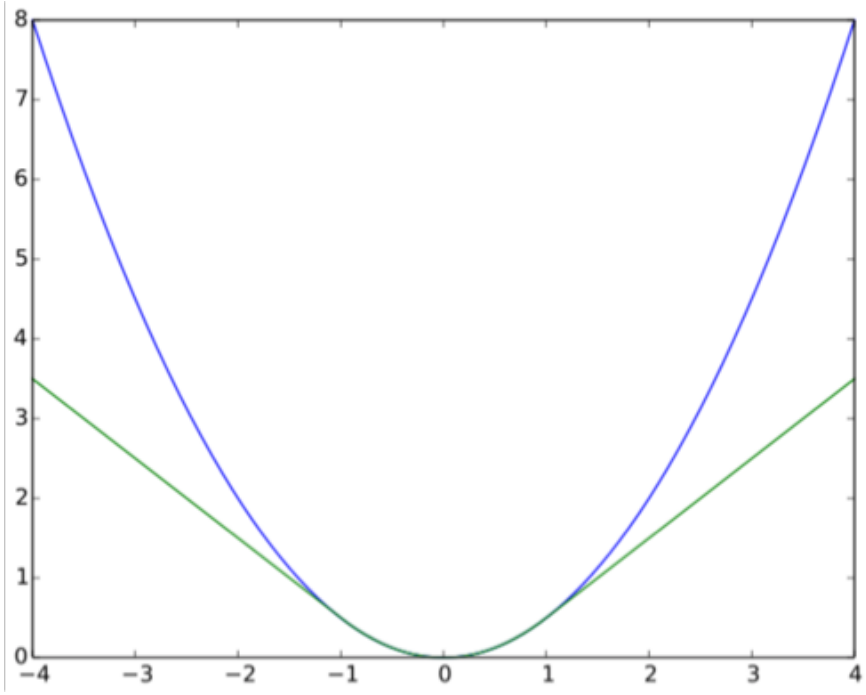
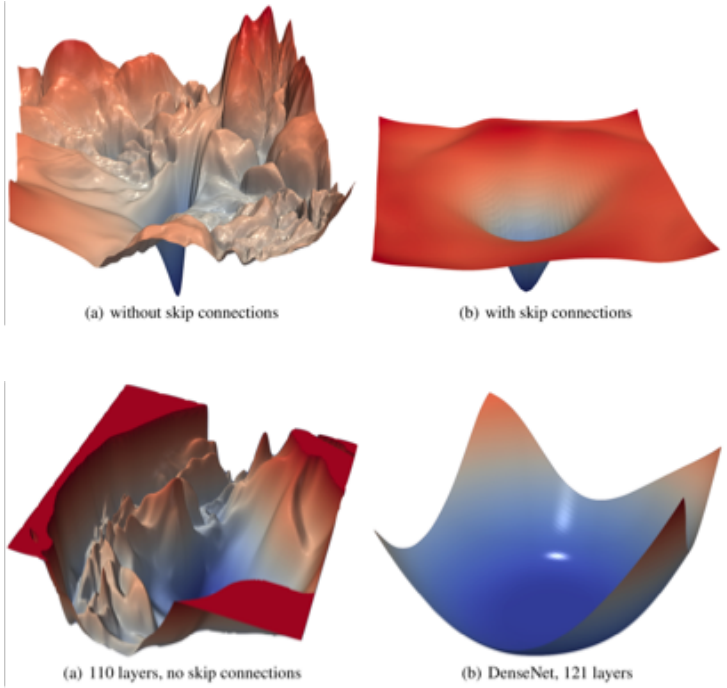


Figure from https://en.wikipedia.org/wiki/Huber_loss#/media/File:Huber_loss.svg

Loss Surface Shapes

- The shape is a consequence of the choice of the data, network design and loss function
  - The weight update is going to attempt to find the lowest value of the function (and assume it also corresponds to the optimal value on the testing data)
  - Unfortunately, the loss surface is not convex
- You can't visualize it
  - You can think of shapes in ~ 1 – 4 dimensions
  - The error (loss) is a function of the number of parameters (easily millions)
  - You can't (easily) think of shapes in millions dimensions
- But it has some characteristics that allow training to work
- Reminder: tell story on how to escape from jail



Figures from https://arxiv.org/abs/1712.09913

Unequal Class Weightings

- Some classes are more difficult than others
- But so far the error calculation has treated all classes the same
- Strategy for modifications to the equal output class weighting of errors
  - Weight classes that are difficult (likely under represented in the training data) for the network more in the error computation
- Focal loss for dense object detection
  - https://arxiv.org/abs/1708.02002
  - Modifies cross entropy based on the inverse of category likelihood and probability when misclassified

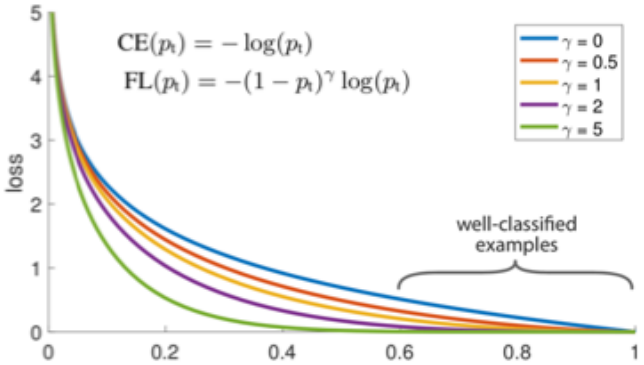
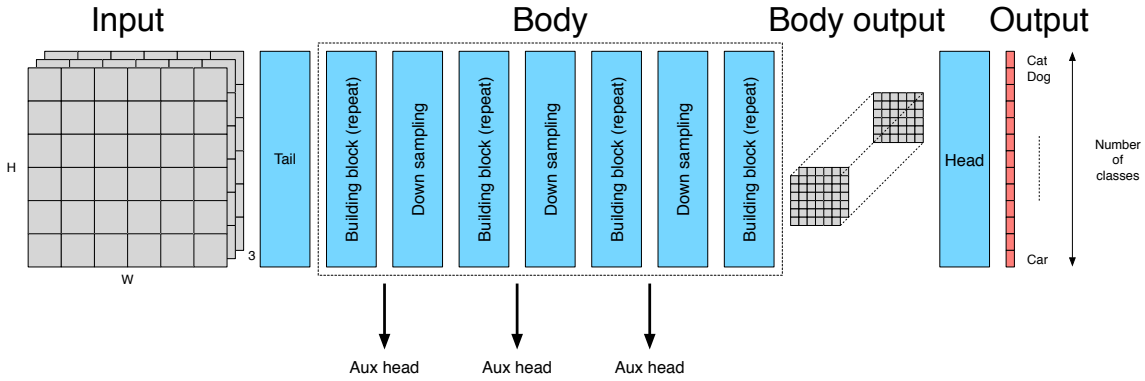


Figure from https://arxiv.org/abs/1708.02002

Auxiliary Network Heads

- Features strong but poorly localized at the output of the body
  - Features are better localized but weaker earlier on in the network
  - While weaker, they're potentially still strong enough to predict classes, just not as accurately



- Idea: use auxiliary network heads during training to provide gradient information directly into the middle of the network

- Note: has possible drawback of getting the parameters stuck too much at a local minima

Multiple Network Heads

- Sometimes a network tail and body have multiple heads, each that accomplishes a different task
  - Want to optimize the performance of the full network
  - So need to create a loss that's combines the losses of all the heads
  - Multi task / combined function

- It's common to do this in multiple object detection networks
  - Classification of boxes as bounding objects and dimension modifications
  - Classification of objects in the bounding boxes

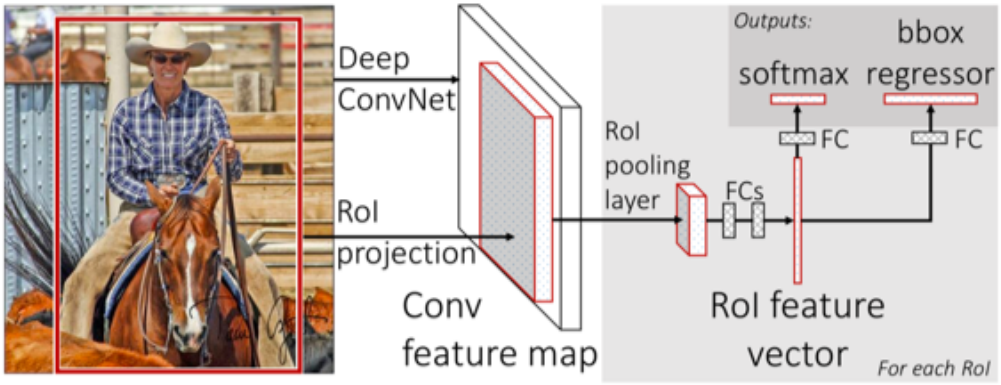
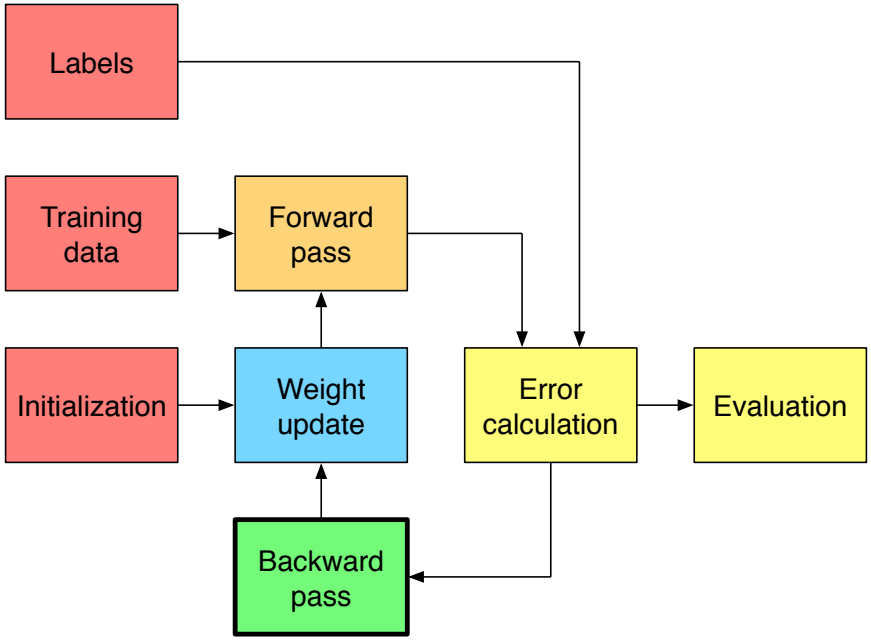


Figure from https://arxiv.org/abs/1504.08083

Backward Pass

Running Out Of Memory Is No Fun

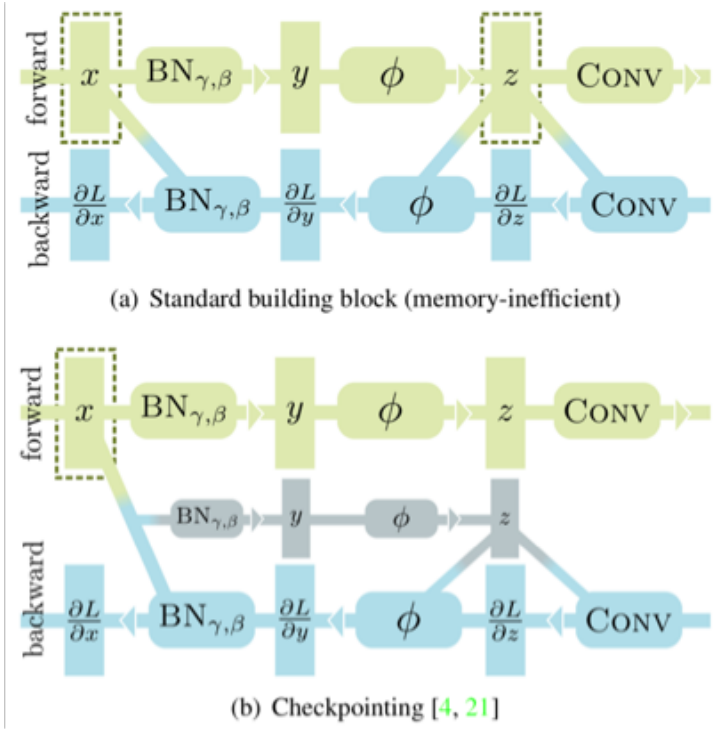
- Reverse mode automatic differentiation
  - Covered in the calculus lecture
  - Basically stays the same
- But possibly make a modifications to address a practical issue: memory (or a lack thereof)
- Situation
  - Typically want larger batch sizes
  - This increases memory
  - Note that gradient computations in the backward pass are dependent on feature maps in the forward pass
  - End up running out of memory
  - Also, moving memory around is typically less efficient then computing



Checkpointing

- A strategy for addressing running out of memory
  - Trade increased computation for reduced memory
  - A collection of techniques that falls under the umbrella of checkpointing
  - Also, change some non invertible operations in the forward pass to similar but invertible versions and use these in a similar way

- The data-flow equations of checkpointing in reverse automatic differentiation
  - https://www-sop.inria.fr/tropics/papers/DauvergneHascoet06.pdf
- Training deep nets with sublinear memory cost
  - https://arxiv.org/pdf/1604.06174.pdf
- Memory-efficient backpropagation through time
  - https://arxiv.org/abs/1606.03401
- For a nice implementation of this and figures see https://github.com/openai/gradient-checkpointing



(a) Standard building block (memory-inefficient)

(b) Checkpointing [4, 21]

Figure from https://arxiv.org/abs/1712.02616

In Place Activated Batch Norm

- Strategy
  - Replaces batch norm -> activation with a single layer and uses leaky ReLU vs ReLU to allow inversion through the nonlinearity to save ~ 50% of memory
  - Remember the word “bijection” from the discussion of functions

- In-place activated batchnorm for memory-optimized training of DNNs
  - https://arxiv.org/abs/1712.02616
  - See also: https://blog.mapillary.com/update/2017/12/08/massive-memory-savings-for-training-modern-deep-learning-architectures.html

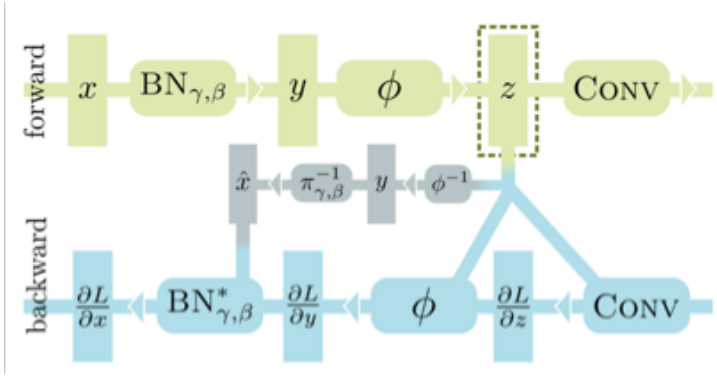


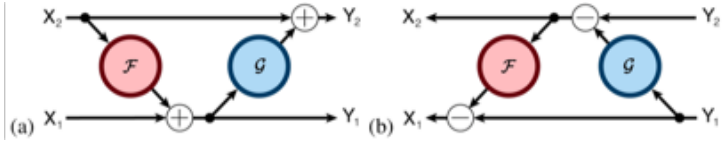
Figure from https://arxiv.org/abs/1712.02616

Reversible Architectures

- Strategy
  - In place activated batch norm made the pointwise nonlinearity invertible
  - Question: is it possible to make the whole network (or at least large blocks of it) invertible

- RevNet
  - Apologies: I know you thought you were done learning new xNet names
  - Defines a different residual network such that each layers output can be derived from the previous layers output
  - The reversible residual network: backpropagation without storing activations
    - https://arxiv.org/pdf/1707.04585.pdf

- Perhaps less popular to do this in practice, but it's worth being aware of as a generally useful idea



Technique	Spatial Complexity (Activations)	Computational Complexity
Naive	$\mathcal{O}(L)$	$\mathcal{O}(L)$
Checkpointing [20]	$\mathcal{O}(\sqrt{L})$	$\mathcal{O}(L)$
Recursive Checkpointing [5]	$\mathcal{O}(\log L)$	$\mathcal{O}(L \log L)$
Reversible Networks (Ours)	$\mathcal{O}(1)$	$\mathcal{O}(L)$

Figures from https://arxiv.org/pdf/1707.04585.pdf

Evolving Back Propagation

- Search / optimization / learning techniques have been used for
  - Data augmentation
  - Network structure
  - Nonlinearity design
  - Weight update (will see in a few slides)
  - …

- Question: Is it possible to modify or augment the standard back propagation algorithm?

- For more information on this path of work see
  - Backprop evolution
  - https://arxiv.org/abs/1808.02822

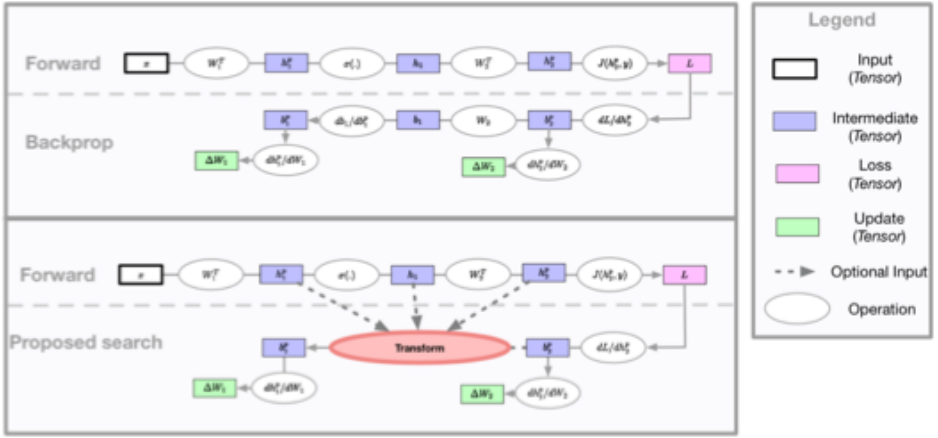


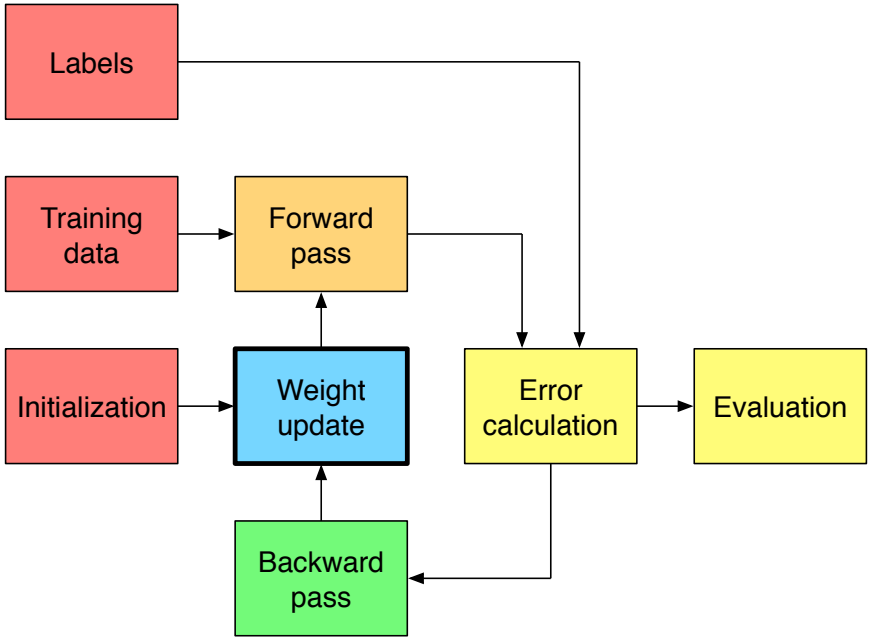
Figure from https://arxiv.org/abs/1808.02822

Weight Update

xNN Training Vs Function Optimization

- Classic gradient descent
  - Runs through all data samples
  - Then makes an update

- Modifications to classical gradient descent considered in this section serve at least 1 of 3 purposes
  - Improve accuracy (convergence)
  - Improve generalization (regularization)
  - Improve performance (speed)



Convergence

- Items described here for improving convergence include
  - Batch size
  - Gradient update method
  - Parameter update schedule

Batch Size

- Some notational things, just be aware of
  - 1 input, 1 update (stochastic gradient descent)
  - Batch of inputs, 1 update (mini batch stochastic gradient descent)
  - All inputs, 1 update (gradient descent)
  - Effectively, how many training samples are we trying to improve the performance on at 1 time

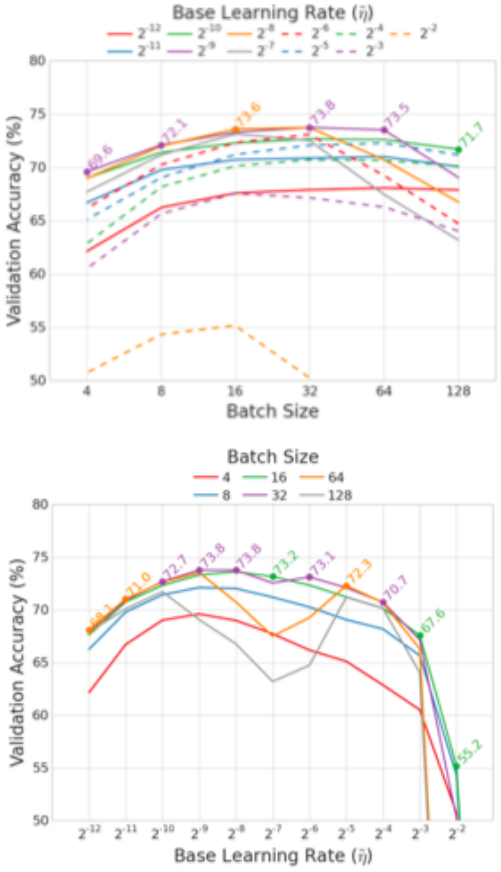
- What does the choice of batch size affect?
  - Memory (smaller == less)
  - Serial updates (larger == faster, to a saturation lim)
  - Parallel updates (larger == faster, more machines)
  - Gradient noise (larger == less)
  - Batch norm effectiveness (larger == better)
  - Accuracy on training data (maybe smaller == better)
  - Accuracy on testing data (maybe smaller == better)

Figure from https://twitter.com/ylecun/status/989610208497360896?lang=en

Batch Size For A Single Machine

- How to select the batch size for a single machine
  - Not a horrible starting point: 32
  - Other not horrible strategy: increase size until the training GPU runs out of memory and use that, unless it's too small and then do some aggregation

- Notes
  - Train longer, generalize better: closing the generalization gap in large batch training of neural networks
    - https://arxiv.org/abs/1705.08741
  - Revisiting small batch training for deep neural networks
    - https://arxiv.org/abs/1804.07612
  - Other
    - https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/
    - https://stats.stackexchange.com/questions/140811/how-large-should-the-batch-size-be-for-stochastic-gradient-descent



Figures from https://arxiv.org/abs/1804.07612

Weight Update Considerations

- Notation
  - Weights $\mathbf{w}$
  - Error $e(\mathbf{w})$
  - Gradient $\mathbf{g}$, $g_i = \partial e / \partial w_i$
  - Hessian $\mathbf{H}$, $H_{i,j} = \partial^2 e / \partial w_i \partial w_j$

- 2nd order approximation of the error about a point $\mathbf{w}_0$
  - Error = current error + 1st order approximation of change + 2nd order correction
  - $e(\mathbf{w}) \approx e(\mathbf{w}_0) + (\mathbf{w} - \mathbf{w}_0)^T \mathbf{g} + 0.5\, (\mathbf{w} - \mathbf{w}_0)^T \mathbf{H} (\mathbf{w} - \mathbf{w}_0)$

- 2nd order Newton's method (reminder: show quick derivation)
  - $\mathbf{H} > \mathbf{0}$ (positive definite)
  - $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{H}^{-1} \mathbf{g}$

Weight Update Considerations

- 1st order gradient descent
  - $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha\, \mathbf{g}$
  - $e(\mathbf{w}_t - \alpha\, \mathbf{g}) \approx \quad e(\mathbf{w}_t) \quad - \alpha\, \mathbf{g}^T \mathbf{g} \quad + 0.5\, \alpha^2\, \mathbf{g}^T \mathbf{H}\, \mathbf{g}$
    Orig Improvement 2nd order correction

- When the 2nd order correction is positive the optimal step size is (show derivation)
  - $\alpha = (\mathbf{g}^T \mathbf{g}) / (\mathbf{g}^T \mathbf{H}\, \mathbf{g})$

- Pertinent note: gradient descent doesn't exploit curvature as captured in the **H** matrix, but different directions have different shapes

- Random note: we also haven't discussed persistent excitation (is the system input / output identifiable?)

Weight Update Notes

- Weight update methods need to deal with non convex error surfaces with 3 types of critical points
  - Minima
  - Maxima
  - Saddle (some directions minima, some maxima)

- Saddle points are the most common type of point in high dimensions
  - Argument: Imagine that minima and maxima are equally likely
  - A critical point requires 1000s to millions of derivatives all to be 0
  - Many more are a blend of positive and negative vs all positive or all negative

- Saddle points tend to have a moderately high error
  - Newton's method is attracted to critical points, critical points tend to be saddle points and saddle points tend to have moderately high error (1 reason why Newton's method is not commonly used for training xNNs)
  - Gradient descent methods aren't attracted specifically to saddle points and seem to be able to escape them (1 reason why they're used for training xNNs)

Weight Update Notes

- Some error surface shapes that are tricky for gradient descent methods
  - Wide flat regions: tend to be tough to get through as the gradient is close to 0; sometimes momentum helps
  - Cliffs: unstable at the edge as gradients can become very large; sometimes gradient clipping helps
  - Local minima in training of neural networks
    - https://arxiv.org/abs/1611.06310

- A caution when optimizing a system with repeated transformations (like in a RNN)
  - Transformations scales gradients
  - If a transformation is repeatedly applied (as in a RNN) then it scales gradients repeatedly
  - If the scaling is large this causes gradient explosion, if the scaling is small this causes gradient vanishing
  - So ideally want the scale of a transformation that's applied many times to be close to 1

- Everything we've talked about only makes local updates
  - Note that gradient descent moves locally, usually just a small amount
  - Problem: if the region where the error is small is not near the current point
  - Hints at the importance of initialization and / or being able to make bigger jumps

Weight Update Methods (Optimizers, Solvers)

- Algorithms without adaptive gradients
  - Stochastic gradient descent (SGD)
  - Stochastic gradient descent with momentum
    - Heavy ball (HB)
    - Nesterov accelerated gradient (NAG)
- Algorithms with adaptive gradients
  - AdaGrad
  - RMSProp
  - Adam
- Presentation styles
  - Deep Learning Book (chapter 8)
    - https://www.deeplearningbook.org/contents/optimization.html
  - The marginal value of adaptive gradient methods in machine learning
    - https://arxiv.org/abs/1705.08292

	SGD	HB	NAG	AdaGrad	RMSProp	Adam
G_k	I	I	I	$G_{k-1} + D_k$	$\beta_2 G_{k-1} + (1-\beta_2) D_k$	$\frac{\beta_2}{1-\beta_2^k} G_{k-1} + \frac{(1-\beta_2)}{1-\beta_2^k} D_k$
α_k	α	α	α	α	α	$\alpha \frac{1-\beta_1}{1-\beta_1^k}$
β_k	0	β	β	0	0	$\frac{\beta_1(1-\beta_1^{k-1})}{1-\beta_1^k}$
γ	0	0	β	0	0	0

Table 1: Parameter settings of algorithms used in deep learning. Here, $D_k = \mathrm{diag}(g_k \circ g_k)$ and $G_k := H_k \circ H_k$. We omit the additional ϵ added to the adaptive methods, which is only needed to ensure non-singularity of the matrices H_k.

General algorithm form without adaptive gradient

$$w_{k+1} = w_k - \alpha_k \tilde{\nabla} f(w_k + \gamma_k(w_k - w_{k-1})) + \beta_k(w_k - w_{k-1})$$

General algorithm form with adaptive gradient

$$w_{k+1} = w_k - \alpha_k H_k^{-1} \tilde{\nabla} f(w_k + \gamma_k(w_k - w_{k-1})) + \beta_k H_k^{-1} H_{k-1}(w_k - w_{k-1})$$

$$H_k = \mathrm{diag}\left(\left\{\sum_{i=1}^{k} \eta_i g_i \circ g_i\right\}^{1/2}\right)$$

Table and equations from https://arxiv.org/abs/1705.08292

Stochastic Gradient Descent

- Stochastic gradient descent
  - Too much data to use gradient descent so use batches
- From the calculus lectures
  - Starting from the gradient of the error with respect to the filter parameters
  - The gradient points in the direction of the local maximum change of the error with respect to the filter parameters
  - To reduce the error take a step in the opposite (negative) direction
- Similar to Hebbian theory in biology
  - Neurons that fire together wire together
  - This adjusts up the neurons that work together more

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  end while

Hyper parameters that require a choice

- Initialization (previously discussed)
- Batch size (previously discussed)
- Batch selection (previously discussed)
- Step size (will discuss next)
- Stopping criteria (will discuss soon)

Figure from https://www.deeplearningbook.org/contents/optimization.html

Stochastic Gradient Descent With Momentum

- Momentum adds short term memory to the update (heavy ball or Nesterov)
- Comments
  - Accumulates an exponentially decaying moving average of past gradients
  - Improves poor conditioning of the Hessian and variance in the stochastic gradient
  - Moves are now in the direction of gradient + momentum
- Paper and nice discussion
  - Why momentum really works (note: excellent post)
    - https://distill.pub/2017/momentum/
  - On the importance of initialization and momentum in deep learning
    - http://proceedings.mlr.press/v28/sutskever13.pdf

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.
  Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
  Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
  Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

Figures from https://www.deeplearningbook.org/contents/optimization.html

AdaGrad

- Inversely scales individual elements of the gradient based on the square root of the square of all previous updates
  - Parameters with large updates have learning rates that are reduced a lot
  - Parameters with small updates have learning rates that are reduced a little (allowing relatively more progress in areas with small gradients)
  - May choose to start this after the initial updates have stabilized
  - Works well with sparse gradients

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ

Require: Initial parameter $\boldsymbol{\theta}$

Require: Small constant δ, perhaps 10^{-7}, for numerical stability

Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$

Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

- Adaptive subgradient methods for online learning and stochastic optimization
  - http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

Figure from https://www.deeplearningbook.org/contents/optimization.html

RMSProp

- Like AdaGrad but with an exponentially weighted forgetting factor on the gradient scale
  - Gets around some of the initial variance of the gradient scale problems that AdaGrad has
  - Exponential forgetting gives it a bit more of a local flavor which seems right
  - Works well with sparse gradients and non stationarity

- Neural networks for machine learning
  - http://www.cs.toronto.edu/%7Etijmen/csc321/slides/lecture_slides_lec6.pdf

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ.
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers.
Initialize accumulation variables $\boldsymbol{r} = 0$
while stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
  Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
end while

Figure from https://www.deeplearningbook.org/contents/optimization.html

RMSProp With Momentum

- Add Nesterov momentum to RMSProp

- Why did I include this beyond the fact that it works in some cases?
  - SGD started us down a path with a nice intuition
  - We added momentum to that which has some plausibility (new direction = old direction + correction)
  - AdaGrad starts to feel like we're playing some games with heuristics, though motivated by intuition of looking at many results
  - RMSProp feels this way even more as it's a modification to AdaGrad
  - By the time we get to RMSProp with momentum we start to think of both creating modifications around gradient descent and combining different modifications that seem to work well together
  - This combination of different ideas is mentally setting us up for learning solvers in a few slides

Algorithm 8.6 RMSProp algorithm with Nesterov momentum

Require: Global learning rate ϵ, decay rate ρ, momentum coefficient α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
Initialize accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

Figure from https://www.deeplearningbook.org/contents/optimization.html

Adam

- Adaptive moments
- Comments
  - Exponential moving averages of the gradient (mean) and the squared gradient (variance)
  - Corrects for the bias in the 0 initialized estimates
  - Update is the the direction of the gradient mean with individual elements scaled by the square root of the gradient variance
- Adam: a method for stochastic optimization
  - https://arxiv.org/abs/1412.6980

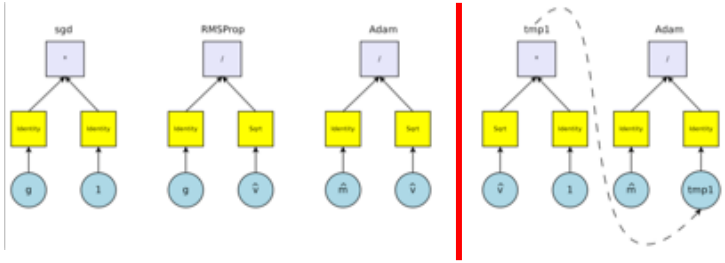
Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

Figure from https://www.deeplearningbook.org/contents/optimization.html

Learning A New Solver

- Flashback to the design lecture
  - We spent a considerable amount of time discussing xNN designs created by humans
  - Then briefly spent time discussing that we could put various configurable network design building blocks in an optimizer and algorithmically design a network
  - We also did this for nonlinearities, using an optimizer to find a replacement for ReLU
- Now think about right now
  - We just spent a considerable amount of time discussing weight update methods created by humans
  - Guess what: we can put various configurable weight update building blocks in an optimizer and algorithmically find a solver (and also a learning rate schedule)
- Neural optimizer search with reinforcement learning
  - https://arxiv.org/abs/1709.07417



Figures from https://arxiv.org/abs/1709.07417

Learning A New Solver

- Flashback to the design lecture
  - We spent a considerable amount of time discussing xNN designs created by humans
  - Then briefly spent time discussing that we could put various configurable network design building blocks in an optimizer and algorithmically design a network
  - We also did this for nonlinearities, using an optimizer to find a replacement for ReLU
- Now think about right now
  - We just spent a considerable amount of time discussing weight update methods created by humans
  - Guess what: we can put various configurable weight update building blocks in an optimizer and algorithmically find a solver (and also a learning rate schedule)
- Neural optimizer search with reinforcement learning
  - https://arxiv.org/abs/1709.07417

Operands: g, g^2, g^3, $\hat{m}$, $\hat{v}$, $\hat{\gamma}$, $\text{sign}(g)$, $\text{sign}(\hat{m})$, 1, 2, $\epsilon \sim N(0, 0.01)$, $10^{-4}w$, $10^{-3}w$, $10^{-2}w$, $10^{-1}w$, Adam and RMSProp.

Unary functions which map input x to: x, $-x$, e^x, $\log|x|$, $\sqrt{|x|}$, $clip(x, 10^{-5})$, $clip(x, 10^{-4})$, $clip(x, 10^{-3})$, $drop(x, 0.1)$, $drop(x, 0.3)$, $drop(x, 0.5)$ and $\text{sign}(x)$.

Binary functions which map (x, y) to $x + y$ (addition), $x - y$ (subtraction), $x * y$ (multiplication), $\frac{x}{y+\delta}$ (division), x^y (exponentiation) or x (keep left).

linear decay: $ld = 1 - \frac{t}{T}$.

cyclical decay: $cd_n = 0.5 * (1 + cos(2\pi n \frac{t}{T}))$.

restart decay: $rd_n = 0.5 * (1 + cos(\pi \frac{(tn)\%T}{T}))$ introduced in Loshchilov & Hutter (2017).

annealed noise: $\epsilon_t \sim N(0, 1/(1+t)^{0.55})$ inspired by Neelakantan et al. (2016).

Figures from https://arxiv.org/abs/1709.07417

Learning A New Solver

- Flashback to the design lecture
  - We spent a considerable amount of time discussing xNN designs created by humans
  - Then briefly spent time discussing that we could put various configurable network design building blocks in an optimizer and algorithmically design a network
  - We also did this for nonlinearities, using an optimizer to find a replacement for ReLU
- Now think about right now
  - We just spent a considerable amount of time discussing weight update methods created by humans
  - Guess what: we can put various configurable weight update building blocks in an optimizer and algorithmically find a solver (and also a learning rate schedule)
- Neural optimizer search with reinforcement learning
  - https://arxiv.org/abs/1709.07417

PowerSign: $\alpha^{f(t)*\text{sign}(g)*\text{sign}(m)} * g$. Some sampled update rules in this family include:

- $e^{\text{sign}(g)*\text{sign}(m)} * g$
- $e^{ld*\text{sign}(g)*\text{sign}(m)} * g$
- $e^{cd*\text{sign}(g)*\text{sign}(m)} * g$
- $2^{\text{sign}(g)*\text{sign}(m)} * g$

AddSign: $(\alpha + f(t) * \text{sign}(g) * \text{sign}(m)) * g$. Some sampled update rules in this family include:

- $(1 + \text{sign}(g) * \text{sign}(m)) * g$
- $(1 + ld * \text{sign}(g) * \text{sign}(m)) * g$
- $(1 + cd * \text{sign}(g) * \text{sign}(m)) * g$
- $(2 + \text{sign}(g) * \text{sign}(m)) * g$

$ld * cd$, which we call *linear cosine decay*.

$(ld+\epsilon_t)*cd+0.001$, which we call *noisy linear decay*.

Figures from https://arxiv.org/abs/1709.07417

Which Solver To Use?

- Think of the below suggestions as a starting point for an initial training when you're still sorting out variations in network design

- Suggestions
  - SGD with momentum
    - Maybe slower
    - Maybe generalizes better
  - Adam
    - Maybe faster
    - Maybe generalizes worse

Figure from http://www.cartoonswallpapers.net/scooby-doo/scooby-doo-where-are-you-original-intro-widescreen-wallpaper-phone/

Learning Rate Update Schedules

- This is typically encoded as a hyper parameter
- Types
  - Fixed step schedule
  - Continual step schedule
  - Adaptive step schedule
  - Hyper gradient descent
- Online learning rate adaptation with hypergradient descent
  - https://arxiv.org/abs/1703.04782
- Step size matters in deep learning
  - https://arxiv.org/abs/1805.08890

Learning Rate Update Schedules

- An interesting arc of papers on using cyclical learning rate schedules that has been used for very fast convergence

- Cyclical learning rates for training neural networks
  - https://arxiv.org/abs/1506.01186
  - Instead of monotonically decreasing the learning rate this cycles the learning rate between 2 limiting values
  - Improved accuracy in some cases and simplifies the tuning process

- Super-convergence: very fast training of neural networks using large learning rates
  - https://arxiv.org/abs/1708.07120
  - Use cyclical learning rates during training
  - Uses a learning rate range test to find the learning bounds
  - Fast.ai further modified the learning rate and momentum strategy of this method to improve accuracy as shown here https://www.fast.ai/2018/04/30/dawnbench-fastai/

Regularization

- Example methods
  - Weight decay
  - Gradient noise
  - Gradient clipping

Weight Decay

- Basic idea
  - Large weights make overfitting more likely
  - Introduce a term in the error calculation to penalize large weights, acts as a regularizer to improve generalization
  - The effect of the introduction of this term in the error calculation is a decay of the weights during the weight update step (hence the name weight decay)

- 2 common penalty terms
  - l_2 norm: $e_{decay}(\mathbf{w}) = e_{original}(\mathbf{w}) + \lambda \Sigma_k (w^2(k))$
  - l_1 norm: $e_{decay}(\mathbf{w}) = e_{original}(\mathbf{w}) + \lambda \Sigma_k |w(k)|$

- The regularization strategy is also called ridge regression or Tikhonov regularization

Weight Decay

- Resulting weight update
  - l_2 norm: $\mathbf{w}_{decay} = \mathbf{w}_{original} - \alpha\, \partial e/\partial \mathbf{w} - \alpha\, \lambda\, \mathbf{w}$
  - l_1 norm: $\mathbf{w}_{decay} = \mathbf{w}_{original} - \alpha\, \partial e/\partial \mathbf{w} - \alpha\, \lambda\, \text{sign}(\mathbf{w})$

- Comments
  - l_2 regularization is more common than l_1 regularization
  - l_1 regularization tends to promote more sparsity
  - It's common to only apply this to multiplicative weights and not additive weights (appropriate due to the scale of additive weights being a function of the feature map and multiplicative weight scales)

- A simple weight decay can improve generalization
  - https://papers.nips.cc/paper/563-a-simple-weight-decay-can-improve-generalization.pdf

Weight Decay

- A couple of additional weight decay results ...

- Spectral norm regularization for improving the generalizability of deep learning
  - https://arxiv.org/abs/1705.10941
  - Bounds the spectral norm (max singular value) of weight matrices during training
  - Makes the network less sensitive to perturbations of the test data

- Decoupled weight decay regularization
  - https://arxiv.org/abs/1711.05101
  - Modifies weight decay regularization for adaptive gradient methods like Adam to decouple the weight decay from the learning rate
  - Allows for better accuracy (similar to SGD) on ImageNet classification

Gradient Noise

- Basic idea
  - Add noise to the gradient used by the weight update to improve regularization
- On the energy landscape of deep networks
  - https://arxiv.org/abs/1511.06485
  - Adds noise to the gradient via an annealing strategy to control the number of local minima in the error surface, starting from a simple shaped surface and transitioning to the original complex shaped surface
- For variations on this theme see
  - Adding gradient noise improves learning for very deep networks
    - https://arxiv.org/abs/1511.06807
  - The effects of adding noise during backpropagation training on a generalization performance
    - https://ieeexplore.ieee.org/document/6796981

Gradient Clipping

- RNNs, as they use the same transform repeatedly, have a potential problem with exploding or vanishing gradients
  - Learning long-term dependencies with gradient descent is difficult
    - http://www.iro.umontreal.ca/~lisa/pointeurs/ieeetrnn94.pdf
- One method for addressing the exploding gradient problem is to use gradient clipping
  - If the magnitude of the gradient exceeds a threshold clip the magnitude to the threshold
  - On the difficulty of training Recurrent Neural Networks
    - https://arxiv.org/abs/1211.5063
- Note that this may introduce some subtle issues with respect to back prop

Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

Figure from https://arxiv.org/abs/1211.5063

Parallelization

- Performance == training speed
  - Not accuracy, we'll call that accuracy
- Motivation
  - Training large models on large datasets takes a long time
  - Would like to use multiple computers to parallelize training
- Different methods for parallelizing across multiple computers
  - **Data**
  - Model
- Different methods for synchronization across multiple computers
  - **Synchronous**
  - Asynchronous

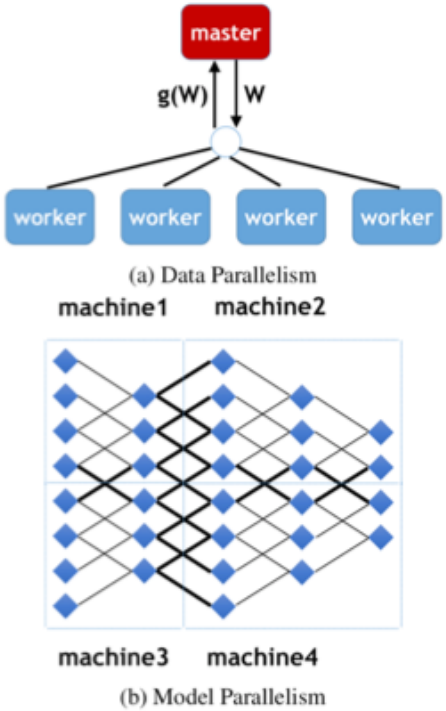


Figure from https://arxiv.org/abs/1709.05011

Synchronous Stochastic Gradient Descent

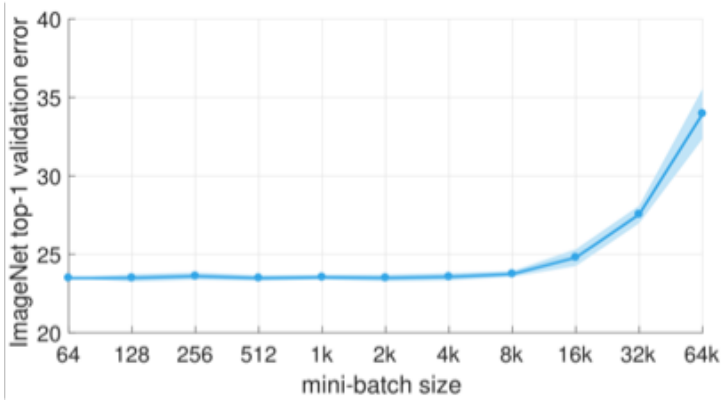
- Synchronous gradient updates
  - Want a large batch size for efficiency
  - Give a part of the larger batch to each worker
  - But a large batch size causes optimization problems
  - Popular benchmark is training ResNet 50 on ImageNet to a specified accuracy

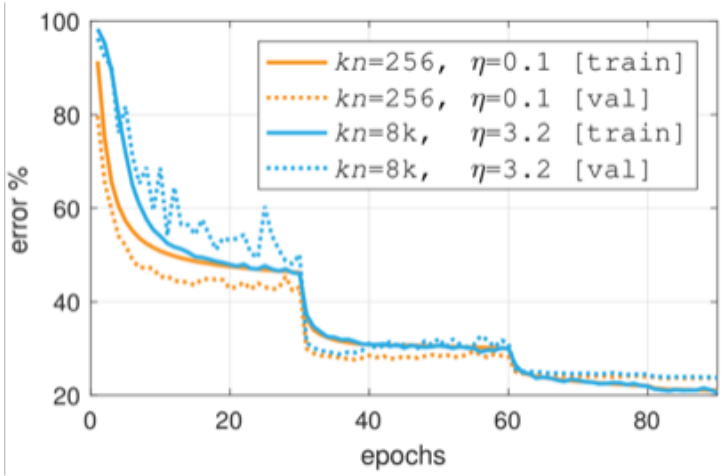
- Accurate, large minibatch SGD: training ImageNet in 1 hour
  - https://arxiv.org/abs/1706.02677
- Large batch training of convolutional networks
  - https://arxiv.org/abs/1708.03888
- ImageNet training in minutes
  - https://arxiv.org/abs/1709.05011
- Extremely large minibatch SGD: training ResNet-50 on ImageNet in 15 minutes
  - https://arxiv.org/abs/1711.04325

Batch Size For Parallel

- How to select the batch size for parallel training
  - Comment: ideally want a huge batch size
  - Why? So an arbitrarily large number of workers can all process a serial machine optimal batch size
  - Say serial machine optimal is 32 and the desire is to have 1024 machines in parallel for training, that implies a batch size of 32768
  - Without modification, SGD tends to fail for that case; this observation has motivated a number of SGD modifications

- Accurate, large minibatch SGD: training ImageNet in 1 hour
  - https://arxiv.org/abs/1706.02677
  - Used a linear scaling rule to adjust the learning rate (multiply the learning rate by the multiple of the batch size) and used a warm up approach that started the learning rate small in the beginning when the weights were varying the most to allow training with batch sizes up to 8192 image



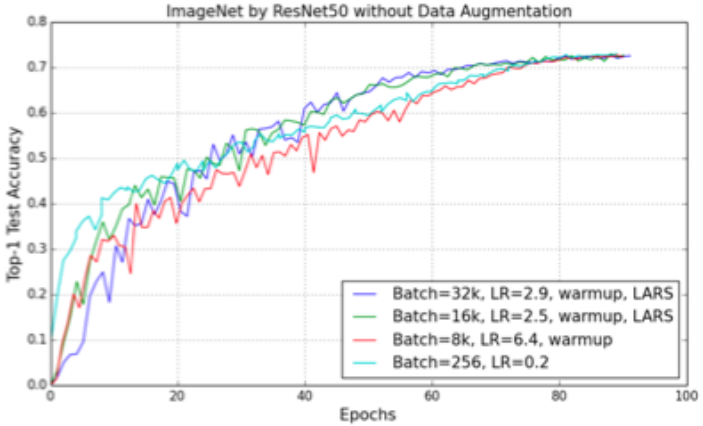


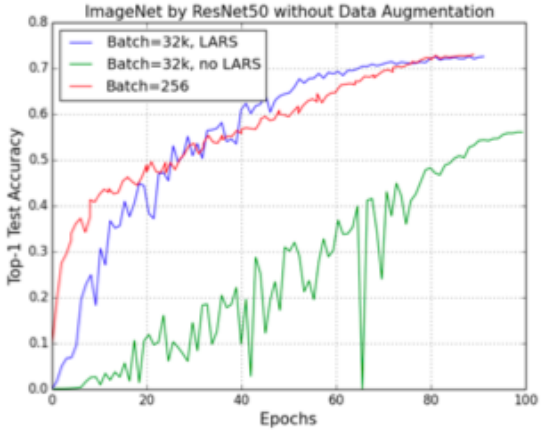
Figures from https://arxiv.org/abs/1706.02677

Batch Size For Parallel

- Large batch training of convolutional networks
  - https://arxiv.org/abs/1708.03888
  - Found that the warm up method didn't work well enough, replaced it with a different learning rate for each layer that allowed training with very large batch sizes

- ImageNet training in minutes
  - https://arxiv.org/abs/1709.05011
  - Used the above large batch training algorithm scaled to 1024 nodes and trained ResNet to published accuracy in ~ 15 minutes





Figures from https://arxiv.org/abs/1708.03888 and https://arxiv.org/abs/1709.05011

Asynchronous Stochastic Gradient Descent

- Motivation
  - Synchronization of gradient updates introduce a computational bottleneck
  - This can be seen as a feedback system with a tight loop
  - Allow asynchronous updates of the gradients to a common parameter server
  - Challenge is that all gradient updates are based on different models

- Large scale distributed deep networks
  - https://storage.googleapis.com/pub-tools-public-publication-data/pdf/40565.pdf
- Asynchronous stochastic gradient descent with delay compensation
  - https://arxiv.org/abs/1609.08326

Evaluation

Flow

- 2 parts
  - Validation (do this during training)
  - Testing (goal is for this to be good; but can't put it in training loop)

Validation

- Training length
  - Fixed number of iterations
  - Early stopping criteria based on validation error
- Early stopping - but when?
  - https://page.mi.fu-berlin.de/prechelt/Biblio/stop_tricks1997.pdf

Figure from https://page.mi.fu-berlin.de/prechelt/Biblio/stop_tricks1997.pdf

Metrics

- We'll talk about evaluation metrics in the context of applications
  - A few examples are shown below
- Vision
  - Top 1 and top 5 errors for classification
  - Precision recall curves in multiple object detection
- Speech
  - Confusion matrices for key word spotting
  - CER, WER, insertions, deletions and substitutions in speech to text transduction
- Language
  - Word similarity and word analogy scores for word embeddings
  - Perplexity for language modeling
  - BLEU scores for language translation

Testing

- Remember: danger of putting final testing in the same loop (frequently implicitly)

Hyper Parameter Selection

How You Configure Training

- This effectively how you configure the training
  - But it's listed here at the end of the training presentation to allow the previous sections to introduce all the different parameters first

- How do you choose values?
  - Start from good settings that someone else found for a similar problem
  - Maybe modify them a little
  - Do some sort of search (grid, random, ...) with a bunch of test trainings
    - Some amount of randomization is likely beneficial
    - Possibly some sort of Bayesian optimization depending on the problem
  - Max out some of the parameters until you hit divergence, then back off a bit
  - ...

Examples

- Data
  - Batch size and selection
  - Pre processing
- Initialization
  - Distribution
- Forward pass

Examples Continued

- Error calculation
  - Loss function
- Weight update method and schedule
  - Stochastic gradient descent variant
  - Learning rate
  - Weight decay
- Evaluation
  - Number of batches for learning before evaluation / validation
  - Stopping criteria

Selecting Samples Of Training Data

Note: batch size will be discussed more during the weight update section

- Batch / iteration / epoch
  - Batch size: a group of inputs (a batch) used to make a single weight update
  - Number of iterations: number of weight updates
  - Number of epochs: batch size * number of iterations / number of training samples

- Options for selecting training samples
  - Walk through a random ordering of all training data a batch at a time (maybe randomize each epoch)
  - Start with easier samples then move to harder samples (curriculum learning)
  - Force some level of balance in different samples
  - Force some bias in different samples
  - More difficult cases vs easy cases (online hard example mining, importance sampling)
  - For subsequent epochs can keep the same ordering or switch to a new random ordering (sometimes a hassle due to practical data preparation constraints)

For some new results on importance sampling as applied to xNN training see: Not all samples are created equal: deep learning with importance sampling (https://arxiv.org/abs/1803.00942)

Case Study

1: Practical ResNet50 Training On ImageNet

- Note: This was done using resources that many people have at their disposal
- To win a contest, you typically need to throw the kitchen sink at a problem ...
- These are some links to fast.ai's approach to training ResNet50 on ImageNet
  - DAWNBench
    - https://dawn.cs.stanford.edu/benchmark/
  - Now anyone can train ImageNet in 18 minutes
    - https://www.fast.ai/2018/08/10/fastai-diu-imagenet/
  - AdamW and super-convergence is now the fastest way to train neural nets
    - https://www.fast.ai/2018/07/02/adam-weight-decay/
  - Training ImageNet in 3 hours for $25; and CIFAR10 for $0.26
    - https://www.fast.ai/2018/04/30/dawnbench-fastai/

1: Practical ResNet50 Training On ImageNet

- Techniques used
  - Progressive resizing for classification
  - Rectangular windows for validation
  - Nvidia NCCL
  - Large batch training of convolutional networks
    - https://arxiv.org/abs/1708.03888
    - LARS
  - PyTorch all reduce
  - Highly scalable deep learning training system with mixed-precision: training ImageNet in four minutes
    - https://arxiv.org/abs/1807.11205
    - Weight decay tuning
  - Google Brain dynamic batch sizes
  - Accurate, large minibatch SGD: training ImageNet in 1 hour
    - https://arxiv.org/abs/1706.02677
    - Gradual learning rate warm up

2: Specialized ResNet50 Training On ImageNet

- Note: This was done at supercomputer scale

- Image classification at supercomputer scale
  - https://arxiv.org/abs/1811.06992
  - 1.8 min to get to 75.2% accuracy
  - 2.2 min to get to 76.3% accuracy

- Techniques (from paper)
  - Mixed precision training using bfloat 16
  - Learning rate scaling, warmup, and decay schedule
  - LARS optimizer to scale to 32768 batch size
  - Distributed batch normalization to control batch normalization batch sizes
  - Input pipeline optimizations to sustain the model throughput
  - Gradient summation with 2D algorithm using torus links
  - 1024 chip TPU v3 Pod using TensorFlow

References

Generalization

- Regularization for deep learning: a taxonomy
  - https://arxiv.org/abs/1710.10686
- Generalization in deep learning
  - https://arxiv.org/abs/1710.05468
- Deep learning and generalization
  - https://www.lpsm.paris/conf_lpsm/SlideBousquetParis2018.pdf
- Statistical learning theory: a hitchhiker's guide
  - https://media.neurips.cc/Conferences/NIPS2018/Slides/stastical_learning_theory.pdf

Data

- ImageNet: a large-scale hierarchical image database
  - https://ieeexplore.ieee.org/document/5206848
- ImageNet large scale visual recognition challenge
  - https://arxiv.org/abs/1409.0575
- The PASCAL visual object classes (VOC) challenge
  - http://host.robots.ox.ac.uk/pascal/VOC/pubs/everingham10.pdf
- Microsoft COCO
  - http://cocodataset.org/
- The Cityscapes dataset for semantic urban scene understanding
  - https://arxiv.org/abs/1604.01685
  - https://www.cityscapes-dataset.com

Data

- Amazon mechanical turk
  - https://www.mturk.com
- VoTT: visual object tagging tool
  - https://github.com/Microsoft/VoTT
- VGG image annotator (VIA)
  - http://www.robots.ox.ac.uk/~vgg/software/via/
- ALP's label tool
  - https://alpslabel.wordpress.com
- LabelMe
  - http://labelme2.csail.mit.edu/Release3.0/index.php

- A closer look at memorization in deep networks
  - https://arxiv.org/abs/1706.05394
- Virtual worlds as a proxy for multiple-object tracking analysis
  - https://arxiv.org/abs/1605.06457

Data

- Auto-encoding variational bayes
  - https://arxiv.org/abs/1312.6114
- Generative adversarial networks
  - https://arxiv.org/abs/1406.2661
- Generative adversarial networks (GANs)
  - https://media.nips.cc/Conferences/2016/Slides/6202-Slides.pdf
- Generative models
  - http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf
- Introduction to GANs
  - http://www.iangoodfellow.com/slides/2018-06-22-gan_tutorial.pdf
- A beginner's guide to generative adversarial networks (GANs)
  - https://skymind.ai/wiki/generative-adversarial-network-gan

- AutoAugment: learning augmentation policies from data
  - https://arxiv.org/abs/1805.09501

Data

- Intriguing properties of neural networks
  - https://arxiv.org/abs/1312.6199
- Deep neural networks are easily fooled: high confidence predictions for unrecognizable images
  - https://arxiv.org/abs/1412.1897
- Explaining and harnessing adversarial examples
  - https://arxiv.org/abs/1412.6572
- Towards deep learning models resistant to adversarial attacks
  - https://arxiv.org/pdf/1706.06083.pdf
- Synthesizing robust adversarial examples
  - https://arxiv.org/abs/1707.07397
- Training region-based object detectors with online hard example mining
  - https://arxiv.org/abs/1604.03540

Initialization

- How transferable are features in deep neural networks?
  - https://arxiv.org/abs/1411.1792
- Curriculum learning
  - https://ronan.collobert.com/pub/matos/2009_curriculum_icml.pdf
- Automated curriculum learning for neural networks
  - https://arxiv.org/abs/1704.03003

Forward Pass

- Batch normalization: accelerating deep network training by reducing internal covariate shift
  - https://arxiv.org/abs/1502.03167
- Batch renormalization: towards reducing minibatch dependence in batch-normalized models
  - https://arxiv.org/abs/1702.03275
- Group normalization
  - https://arxiv.org/abs/1803.08494
- Improving neural networks by preventing co-adaptation of feature detectors
  - https://arxiv.org/abs/1207.0580

- Dropout: a simple way to prevent neural networks from overfitting
  - https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf
- Regularization of neural networks using dropconnect
  - https://cs.nyu.edu/~wanli/dropc/dropc.pdf
- Deep networks with stochastic depth
  - https://arxiv.org/abs/1603.09382

Forward Pass

- Dataset augmentation in feature space
  - https://arxiv.org/abs/1702.05538
- Estimating or propagating gradients through stochastic neurons
  - https://arxiv.org/abs/1305.2982
- Noisy activation functions
  - https://arxiv.org/abs/1603.00391

Error Calculation

- Large-margin softmax loss for convolutional neural networks
  - https://arxiv.org/abs/1612.02295
- Robust estimation of a location parameter
  - https://projecteuclid.org/download/pdf_1/euclid.aoms/1177703732

- Transport-based analysis, modeling, and learning from signal and data distributions
  - https://arxiv.org/abs/1609.04767
- Optimal mass transport: signal processing and machine-learning applications
  - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6024256/
- Applications of optimal transport to machine learning and signal processing
  - https://mathematical-coffees.github.io/slides/mc01-courty.pdf

Error Calculation

- Distilling the knowledge in a neural network
  - https://arxiv.org/abs/1503.02531
- Rethinking the inception architecture for computer vision
  - https://arxiv.org/abs/1512.00567
- DisturbLabel: regularizing CNN on the loss layer
  - https://arxiv.org/abs/1605.00055
- Regularizing neural networks by penalizing confident output distributions
  - https://arxiv.org/abs/1701.06548

Error Calculation

- The loss surfaces of multilayer networks
  - https://arxiv.org/abs/1412.0233
- Identifying and attacking the saddle point problem in high-dimensional non-convex optimization
  - https://arxiv.org/abs/1406.2572
- Visualizing the loss landscape of neural nets
  - https://arxiv.org/abs/1712.09913
- Comparing dynamics: deep neural networks versus glassy systems
  - https://arxiv.org/abs/1803.06969

Error Calculation

- Focal loss for dense object detection
  - https://arxiv.org/abs/1708.02002
- Hierarchical loss for classification
  - https://arxiv.org/abs/1709.01062
- Deeply-supervised nets
  - https://arxiv.org/abs/1409.5185
- Going deeper with convolutions
  - https://arxiv.org/abs/1409.4842

Backward Pass

- Automatic differentiation
  - http://www.robots.ox.ac.uk/~tvg/publications/talks/autodiff.pdf
- Reverse-mode automatic differentiation: a tutorial
  - https://rufflewind.com/2016-12-30/reverse-mode-automatic-differentiation
- Automatic differentiation of algorithms
  - https://www.sciencedirect.com/science/article/pii/S0377042700004222?via%3Dihub
- Automatic reverse-mode differentiation: lecture notes
  - http://www.cs.cmu.edu/~wcohen/10-605/notes/autodiff.pdf
- Training neural networks using features replay
  - https://arxiv.org/abs/1807.04511
- Automatic differentiation in ML: Where we are and where we should be going
  - https://arxiv.org/abs/1810.11530

Weight Update

- Large-scale machine learning with stochastic gradient descent
  - http://leon.bottou.org/publications/pdf/compstat-2010.pdf
- The tradeoffs of large scale learning
  - https://papers.nips.cc/paper/3323-the-tradeoffs-of-large-scale-learning.pdf
- Why momentum really works
  - https://distill.pub/2017/momentum/
- On the importance of initialization and momentum in deep learning
  - http://www.cs.toronto.edu/~fritz/absps/momentum.pdf
- An overview of gradient descent optimization algorithms
  - https://arxiv.org/abs/1609.04747
- Optimization methods for large-scale machine learning
  - https://arxiv.org/abs/1606.04838
- Online learning rate adaptation with hypergradient descent
  - https://arxiv.org/abs/1703.04782

Weight Update

- ADAM: a method for stochastic optimization
  - https://arxiv.org/pdf/1412.6980.pdf
- On the convergence of ADAM and beyond
  - https://openreview.net/pdf?id=ryQu7f-RZ
- Nesterov accelerated gradient and momentum
  - https://jlmelville.github.io/mize/nesterov.html
- Nesterov's optimal gradient methods
  - https://www2.cs.uic.edu/~zhangx/teaching/agm.pdf
- Neural optimizer search with reinforcement learning
  - https://arxiv.org/abs/1709.07417
- The marginal value of adaptive gradient methods in machine learning
  - https://arxiv.org/pdf/1705.08292.pdf

Weight Update

- When is a convolutional filter easy to learn?
  - https://research.fb.com/wp-content/uploads/2018/04/when-is-a-convolutional-filter-easy-to-learn.pdf
- Don't decay the learning rate, increase the batch size
  - https://arxiv.org/abs/1711.00489
- Revisiting small batch training for deep neural networks
  - https://arxiv.org/abs/1804.07612
- Train longer, generalize better: closing the generalization gap in large batch training of neural networks
  - https://arxiv.org/abs/1705.08741

- Sensitivity and generalization in neural networks: an empirical study
  - https://arxiv.org/abs/1802.08760
- Robust large margin deep neural networks
  - https://arxiv.org/abs/1605.08254

Weight Update

- Adding gradient noise improves learning for very deep networks
  - https://arxiv.org/abs/1511.06807
- The effect of gradient noise on the energy landscape of deep networks
  - https://arxiv.org/abs/1511.06485
- The effects of adding noise during backpropagation training on a generalization performance
  - https://ieeexplore.ieee.org/document/6796981

- Accurate, large minibatch SGD: training ImageNet in 1 hour
  - https://arxiv.org/abs/1706.02677
- Extremely large minibatch SGD: training ResNet-50 on ImageNet in 15 minutes
  - https://arxiv.org/abs/1711.04325

Weight Update

- Newton's method
  - http://www.stat.cmu.edu/~ryantibs/convexopt-S15/lectures/14-newton.pdf
- An introduction to optimization chapter 9 Newton's method
  - https://www.cs.ccu.edu.tw/~wtchu/courses/2014s_OPT/Lectures/Chapter%209%20Newton%27s%20Method.pdf

- Large scale distributed deep networks
  - https://storage.googleapis.com/pub-tools-public-publication-data/pdf/40565.pdf
- Asynchronous stochastic gradient descent with delay compensation
  - https://arxiv.org/abs/1609.08326

Hyper Parameter Selection

- Scalable gradient-based tuning of continuous regularization hyperparameters
  - https://arxiv.org/abs/1511.06727