Machine Intelligence & Deep Learning Workshop

Raymond Ptucha, Majid Rabbani, Mark Smith

The Kate Gleason COLLEGE OF ENGINEERING

Introduction to Deep Learning

Raymond Ptucha
June 27-29, 2018
Rochester Institute of Technology
www.rit.edu/kgcoe/cqas/machinelearning

 1

Fair Use Agreement

This agreement covers the use of all slides in this document, please read carefully.

- You may freely use these slides for personal use, if:
  - My name (R. Ptucha) appears on each slide.
- You may freely use these slides externally, if:
  - You send me an email telling me the conference/venue/company name in advance, and which slides you wish to use.
  - You receive a positive confirmation email back from me.
  - My name (R. Ptucha) appears on each slide you use.

(c) Raymond Ptucha, rwpeec@rit.edu

 2

Agenda

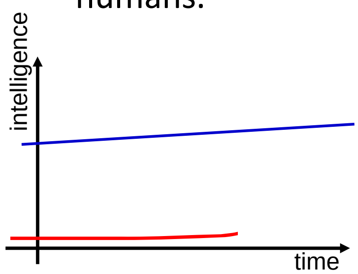
- Wed, June 27
  - 9-10:30am	Regression and Classification
  - 10:30-10:45pm	Break
  - 10:45-12:15pm	Boosting and SVM
  - 12:15-1:30pm	Lunch
  - 1:30-3:30pm	Neural Networks and Dimensionality Reduction
  - 3:30-5pm	Hands-on Python and Machine Learning
- Thur, June 28
  - 9-10:30am	**Introduction to deep learning**
  - 10:30-10:45pm	Break
  - 10:45-12:15pm	Convolutional Neural Networks
  - 12:15-1:30pm	Lunch
  - 1:30-3:30pm	Region and pixel-level convolutions
  - 3:30-5pm	Hands-on CNNs
- Fri, June 29
  - 9-10:30am	Recurrent neural networks
  - 10:30-10:45pm	Break
  - 10:45-12:15pm	Language and Vision
  - 12:15-1:30pm	Lunch
  - 1:30-3:30pm	Graph convolutional neural networks; Generative adversarial networks
  - 3:30-5pm	Hands-on regional CNNs, RNNs

Machine Learning

- Machine learning is giving computers the ability to analyze, generalize, think/reason/behave like humans.
- Machine learning is transforming medical research, financial markets, international security, and generally making humans more efficient and improving quality of life.
- Inspired by the mammalian brain, deep learning is machine learning on steroids- bigger, faster, better!

The point of Singularity

- The point of singularity is when computers become smarter than humans.



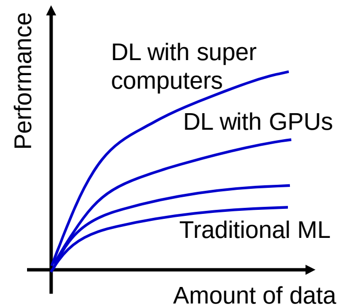
Evolution of biology
Advancement of technology

Unleashing of Intelligence

- Machines will slowly match, then quickly surpass human capabilities.
- Today it is exciting/scary/fun to drive next to an autonomous car.
- Tomorrow it may be considered irresponsible for a human to relinquish control from a car that has faster reaction times, doesn't drink/text/get distracted/tired, and is communicating with surrounding vehicles and objects.

Why is AI (Deep Learning) Just Now Becoming Practical in Many Day-to-Day Situations?

- Availability of data;
- Sustained advances in hardware capabilities (including GPUs running machine learning workloads);
- Omnipresent connectivity;
- Lower cost and power consumption;
- Sustained advances in algorithmic learning techniques.



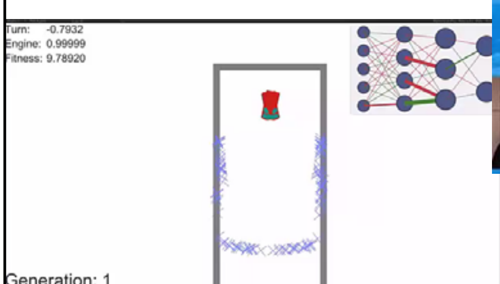
Hot trend:
High performance architecture experts teaming up with deep learning experts

2017: The Year of AI:

The Wall Street Journal, Forbes, and Fortune

NEC Face Recognition

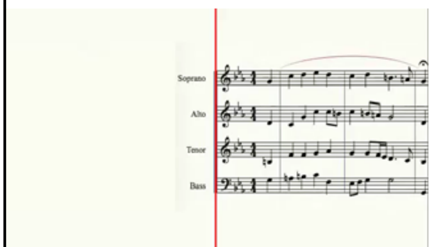
SONY Playstation Virtual Reality



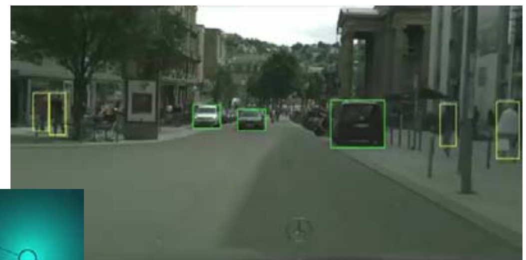
Evolutionary Reinforcement Learning

2017: The Year of AI:

The Wall Street Journal, Forbes, and Fortune



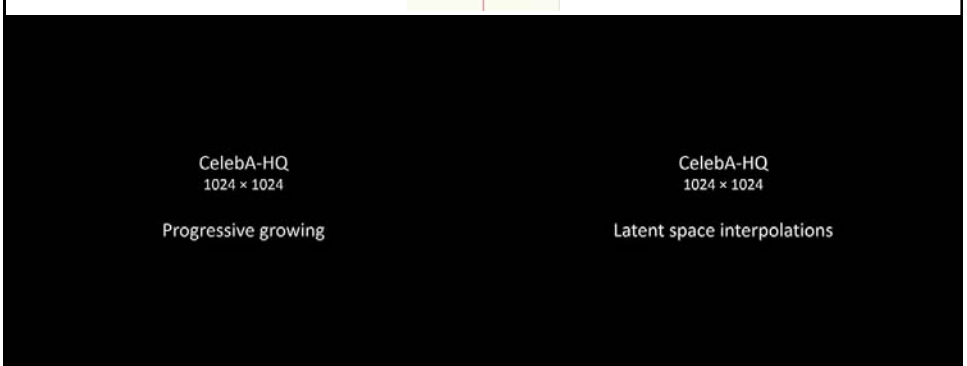
DeepBach



NVIDIA Autonomous Car
Detection & Segmentation

YOLO v2 Object Detection

Some Things to Look for in 2018

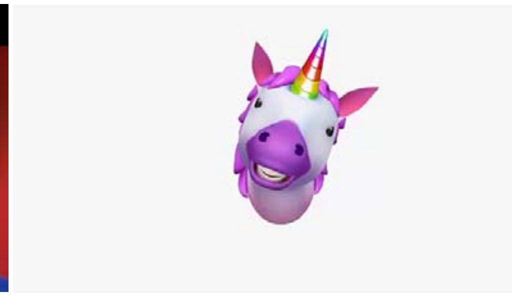


http://research.nvidia.com/sites/default/files/pubs/2017-10_Progressive-Growing-of/karras2017gan-paper.pdf

Some Things to Look for in 2018

Faceshift GDC

Apple iPhone X, Animoji Yourself



Some Things to Look for in 2018

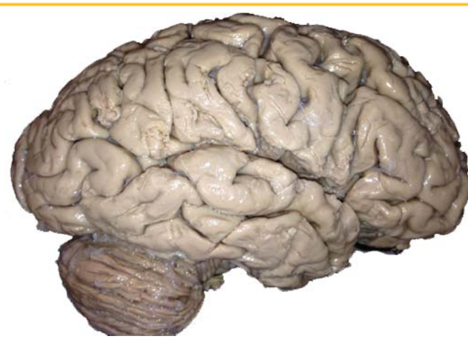
NVIDIA Drive

Prerequisites

- Basic probability and statistics
- Some programming experience (we will be using Python)
- Simple linear algebra
- Simple multivariate calculus

- Don't worry if you don't understand all concepts, over time things will make more sense.
- Feel free to stick around from 5-7pm in the evenings where the instructor and TAs can help you setup your own personal laptop, reinforce concepts, or answer personal applications of deep learning for your particular need.

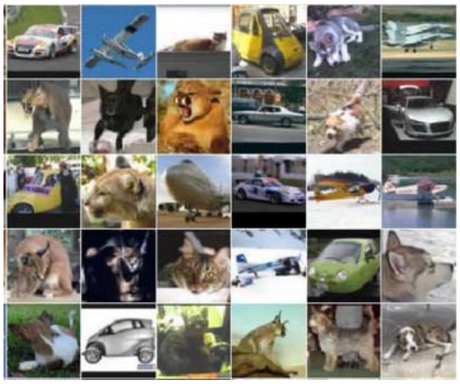
The Human Brain



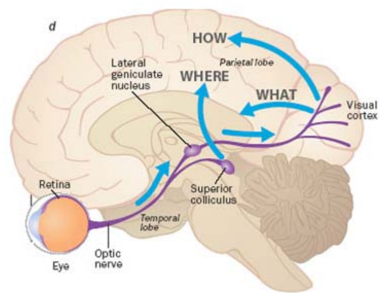
- We've learned more about the brain in the last 5 years than we have learned in the last 5000 years!
- It controls every aspect of our lives, but we still don't understand exactly how it works.

The Brain on Pattern Recognition

- Airplane, Cat, Car, Dog



STL-10 dataset



http://thebraingeek.blogspot.com/2012/08/blindsight.html

The Brain on Pattern Recognition

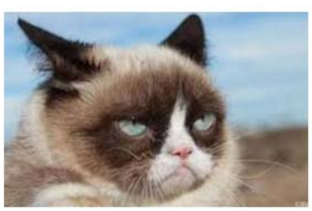
Despite Changes in Deformation:

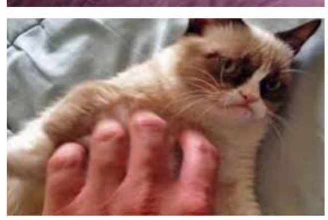
The Brain on Pattern Recognition

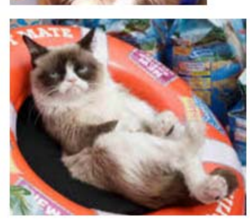
Despite Changes in Occlusion:

The Brain on Pattern Recognition

Despite Changes in Size, Pose, Angle:







Tardar Sauce "Grumpy Cat"

The Brain on Pattern Recognition

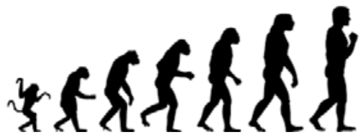
Despite Changes in Background Clutter:

The Brain on Pattern Recognition

Despite Changes in Class Variation…

Teaching Computers to See

- It took evolution 540M years to develop the marvel of the eye-brain.



- Lets say a child collects a new image every 200msec.

- By age 3, this child has processed ~250M images.

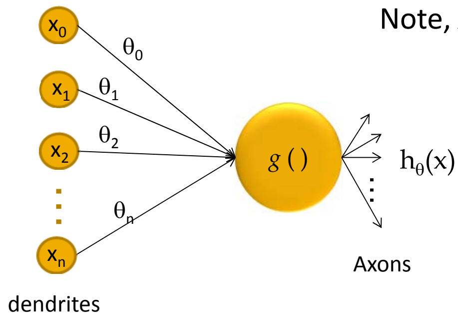
$$(5\ images/sec)(60sec/min)(60min/hr)(12hr/day)(365days/yr)(3yrs) = 236M$$

- Today's computers can do this in a few days…

Neural Nets on Pattern Recognition

- Instead of trying to code simple intuitions/rules on what makes an airplane, car, cat, and dog…
- We feed neural networks a large number of training samples, and it will automatically learn the rules!
- We will learn the magic behind this today!

Artificial Neuron



Note, x_0 is the bias unit, x_0=1

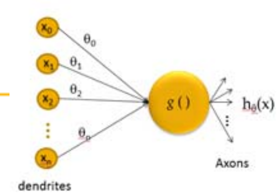
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$$

Activation function

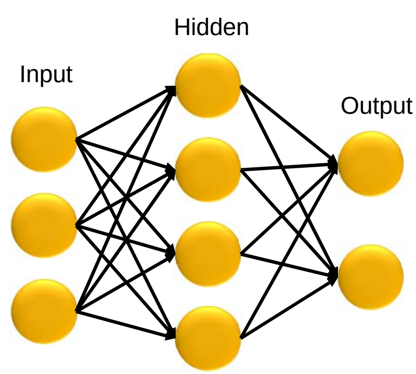
$$h_\theta(x) = g(x_0\theta_0 + x_1\theta_1 + \ \dots + x_n\theta_n) = g\left(\sum_{i=0}^{n} x_i\theta_i\right)$$

$$h_\theta(x) = g(\theta^T x)$$

Artificial Neural Networks

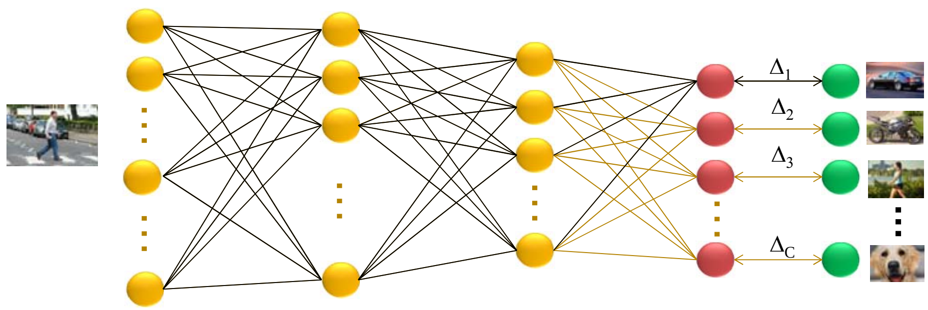


- Artificial Neural Network (ANN) – A network of interconnected nodes that "mimic" the properties of a biological network of neurons



4-Layer ANN Fully Connected Topology

Input Layer | Hidden Layer 1 | Hidden Layer 2 | Output Layer



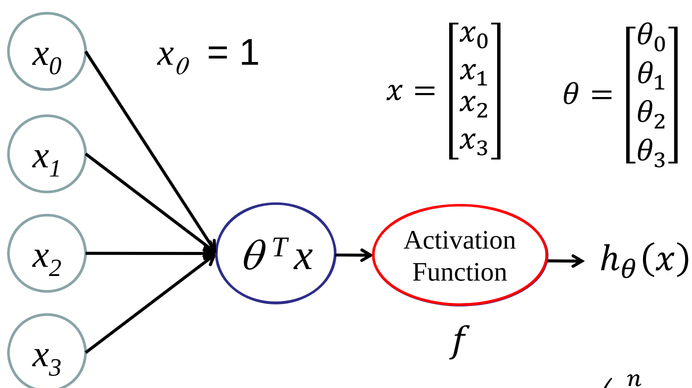
A 20x20 image would have 400 input nodes | *a* nodes | *b* nodes | *C* nodes, where *C* is the number of classes

Backpropagation (~1985) uses $\frac{\partial \Delta}{\partial w}$ for learning

Learning happens in the weights- each line is a weight.

Neuron Model

Bias unit



$$h_\theta(x) = g(x_0\theta_0 + x_1\theta_1 + \ \dots + x_n\theta_n) = g\left(\sum_{i=0}^{n} x_i\theta_i\right)$$

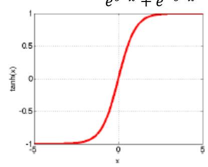
$$h_\theta(x) = g(\theta^T x)$$

Activation Function Comparison

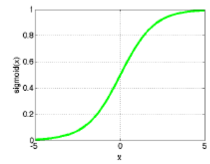
- Tanh
- Sigmoid

Gradient of both saturates at zero. Sigmoid also non-zero centered, so in practice tanh performs better.

$$h_\theta(x) = \frac{e^{\theta^T x} - e^{-\theta^T x}}{e^{\theta^T x} + e^{-\theta^T x}}$$

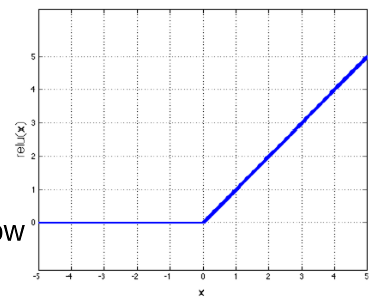


$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

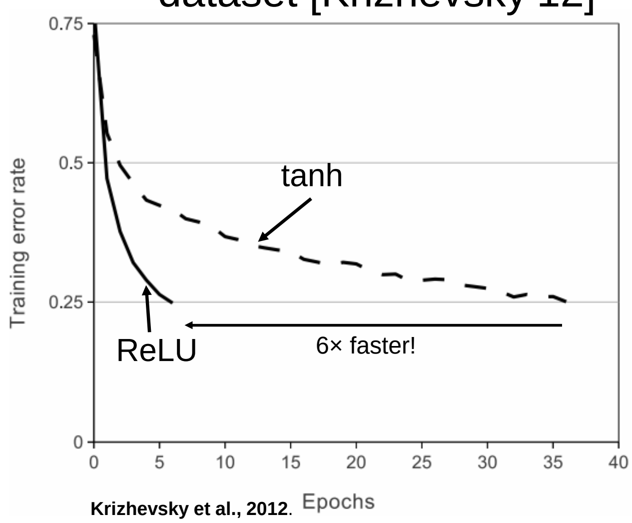


- Rectified Linear Units (ReLU)
  - Better for high dynamic range
  - Faster learning
  - Overall better result
  - Neurons can “die” if allowed to grow unconstrained

$$h_\theta(x) = max(0, x)$$



Tanh vs. ReLU on CIFAR-10 dataset [Krizhevsky’12]



Krizhevsky et al., 2012.

ReLU reaches 25% error 6× faster!
Note: Learning rates optimized for each, no regularization, four layer CNN.

Where Do Weights Come From?

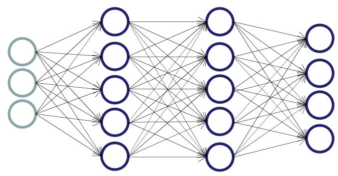
- The weights in a neural network need to be learned such that the errors are minimized.
- Just like logistic regression, we can write a cost function.
- Similar to gradient descent, we can write an iterative procedure to update weights, with each iteration decreasing our cost.
- These iterative methods may be less efficient than a direct analytical solution, but are easier to generalize.

Backpropagation

- We need to solve weights of a network so that the error is minimized.
- Weights can be refined by changing each weight by an amount proportional to the partial derivative of the error with respect to each weight.
- Partial derivatives can be calculated by iteratively changing each weight and measuring the corresponding change in error.
- Hard to do with millions of weights!
- In 1986, a technique called back-propagation was introduced (D. E. Rumelhart, G. E. Hinton, and R. J. Williams "Learning representations by back-propagating errors," J. *Nature* 323, 533-536, 1986).

Hyperparameters vs. parameters

- Hyperparameters are the tuning parameters for a nnet- say number of layers, nodes per layer, learning rate, momentum, regularization, etc.
- Parameters are the weights that are being learned. Ignoring bias terms, the below network has 3x5+5x5+5x4 = 60 parameters.
  - If we include bias terms, we have 4x5+6x5+6x4=74 learnable parameters.



Note: deep nets may contain 100M parameters with 100 layers!

Initialization of Weights

- We initialize weights to small ± values centered on 0.
- If they were all initialized to 0, the network wouldn't learn anything.
  - ie: the output of each node would be equal, therefore the update would update each term identically.
- If they were all initialized to large values (values > +1 or < -1), the inputs to each node would be saturated.
- If they were all initialized to small values around 0, we would be in the linear portion of the activation function.
  - W1 = 0.001*rand(length(h1),length(h2))

Initialization of Weights

- Although the uniform distribution is good, the more inputs to a node, the greater its variance.
- To set output distribution of all nodes equal (this empirically improves convergence), use
  “X

Transfer Learning

h1=# input nodes

- Other solutions restrict weights: $-\epsilon_{init} < \theta_{ji}^{(l)} < \epsilon_{init}$
  $\epsilon_{init} = \sqrt{6}/\sqrt{s_l + s_{l+1}}$ s_l , s_{l+1} are the No. nodes in layers around $\theta^{(l)}$
- For deep ReLU networks, He2015 showed:
  - **W1 = 0.001*rand(length(h1),length(h2))./sqrt(2*length(h1))**
- Works best and is recommended for them

Multiclass Loss Functions

input

Nnet model (w/ 3 outputs)

-0.97	Bunny score
4.38	Cat score
0.62	Dog score

- The input image scores highest against cat, but is also somewhat similar to dog.
- How do we assign a loss function?

Activation Function of Output Layer

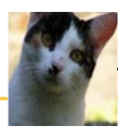
- Sigmoid returns 0 or 1 for each output node.
- What if you wanted a confidence interval?
- Use a linear activation function for regression: $a^{(l)}=z^{(l)}$
- Softmax often used for classification:

$$a_c^{(L)} = h_\theta(x)_c = g\left(z_c^{(L)}\right) = \frac{exp\left(z_c^{(L)}\right)}{\sum_{c=1:C} exp\left(z_c^{(L)}\right)}$$

exp() of each output node

Sum of all output nodes

- Note: Only the output layer activation function changes- all hidden layer nodes activation functions would be the sigmoid/tanh/ReLU function.

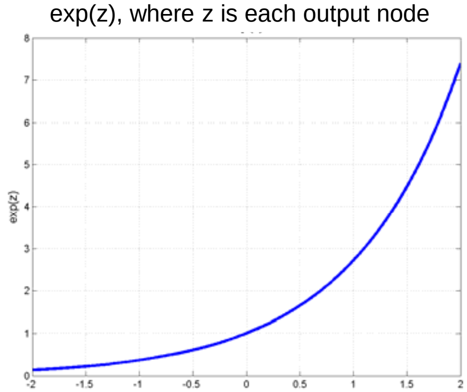


input

Nnet model (w/ 3 outputs)

		$exp\left(z_c^{(L)}\right)$	$\frac{exp\left(z_c^{(L)}\right)}{\sum_{c=1:C} exp\left(z_c^{(L)}\right)}$
-0.97	Bunny score	0.38	0.46
4.38	Cat score	79.8	97.3
0.62	Dog score	1.86	2.26

exp(z), where z is each output node



exp() take in values +/- values and returns all positive values.

Most Common Loss Functions

- The cos … from
  logistic … tion.
- For mu … ds:

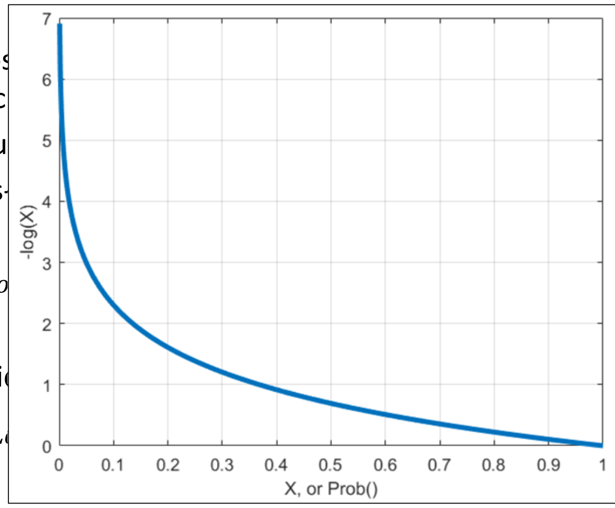
1. Cross-

$Loss^{(i)} = -lo$ … GT node)

… ut) of all nodes

2. Multi

L … rect –
… es



Most Common Loss Functions

- The cost function we previously used was a direct copy from logistic regression and works great for binary classification.
- For multi-class, there are two popular data loss methods:

1. Cross-entropy loss, which uses softmax:

$$Loss^{(i)} = -log\left(\frac{exp\left(out_{yi}^{(i)}\right)}{\sum_{c=1:C} exp\left(out_c^{(i)}\right)}\right)$$

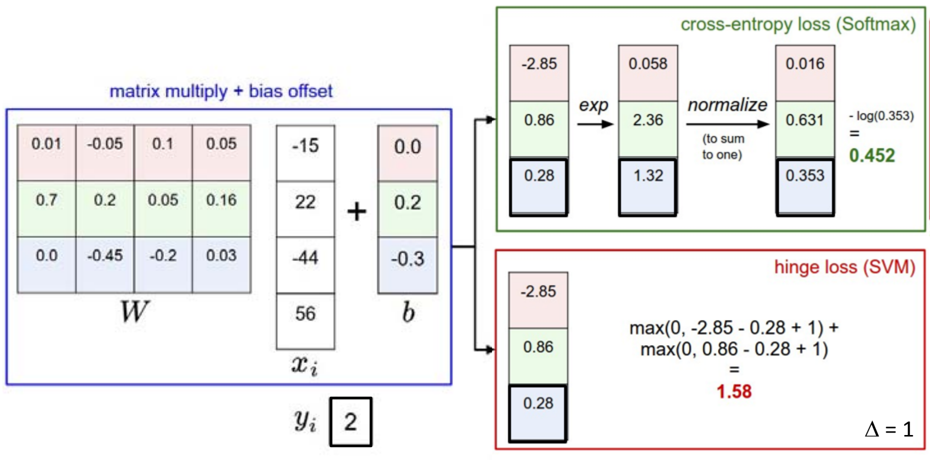
Loss for sample *i* = $\dfrac{\text{exp(output of GT node)}}{\text{Sum of exp(output) of all nodes}}$

2. Multiclass SVM Loss (Weston Watkins formulation):

$$Loss^{(i)} = \sum_{j \neq yi} max\left(0, out_j - out_{yi} + \Delta\right)$$

Sum of incorrect – correct classes

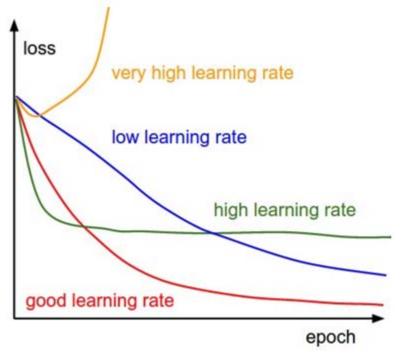
Loss Example
(based on cs231n, Li/Karpathy 2016)

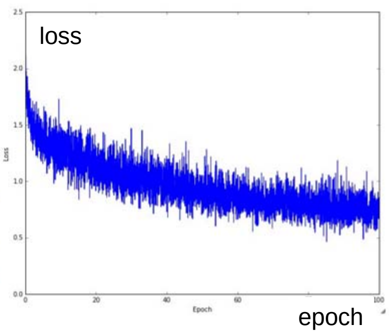


Sequential vs. Batch Training

- Back propagation can be done:
  - Sequential: one sample at a time,
    - Weights are shifted back and forth quite a bit
  - Group (minibatch): a group of samples at a time, or
  - Batch: all training samples at once
    - Weights are shifted in direction that makes most input samples better
    - Generally converges the fastest
- Recommended to use largest minibatch possible and stay within memory constraints of hardware.

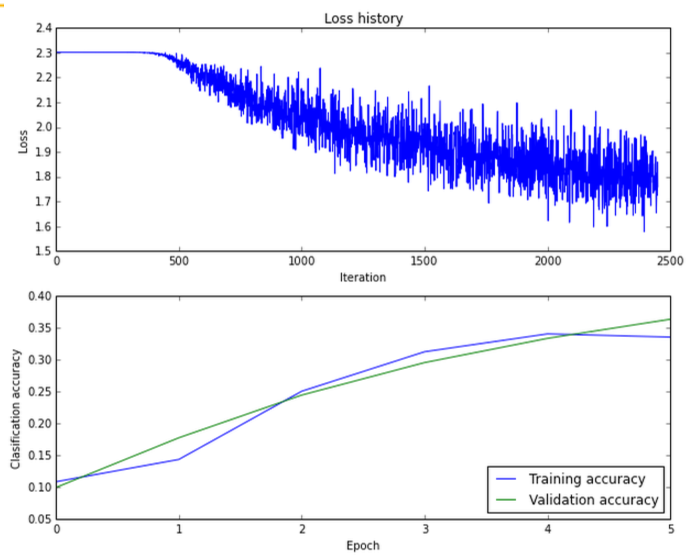
Examples of Learning Rate and Batch Size





This batch size could be made a little larger to shrink the variance

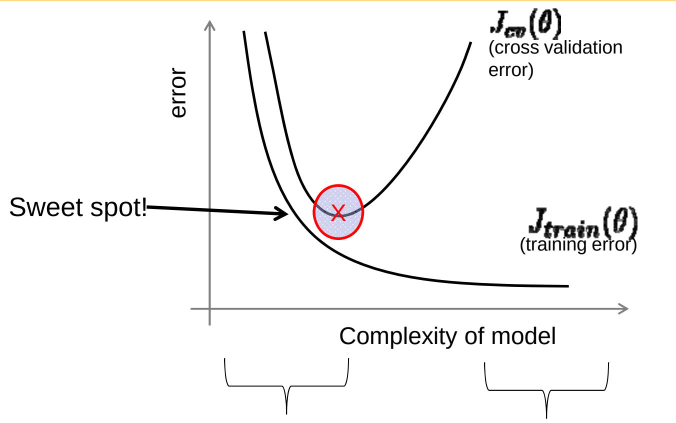
Tuning Parameters



Curve too linear- increase learning rate.

Train and Test sets too similar- increase model complexity.
Be careful, larger models susceptible to overfitting.
Perhaps increase regularization

Bias (underfit) vs. Variance (overfit) errors



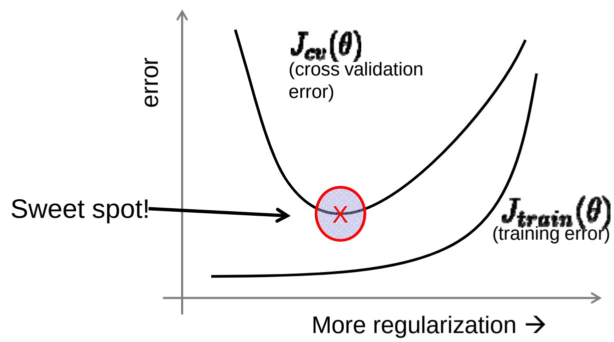
Model too simple. Too high error on train and test

Model too complex. Overfitting to training set

Adopted from: Andrew Ng, ML class

58

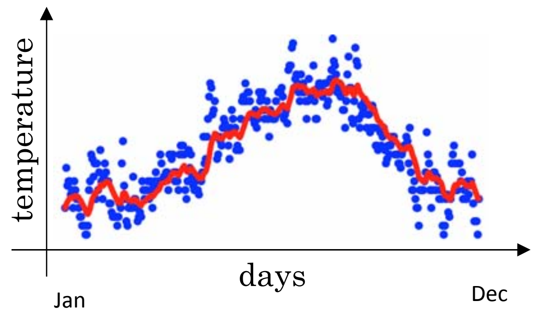
Regularization Tuning



Adopted from: Andrew Ng, ML class

59

Exponentially Weight Average

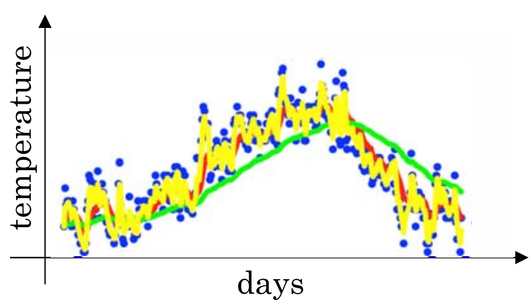


Smoothing algorithm:

$$t_1 = 0.9t_0 + 0.1t_1$$
$$t_2 = 0.9t_1 + 0.1t_2$$
$$t_3 = 0.9t_2 + 0.1t_3$$
$$\ldots$$

https://www.coursera.org/learn/deep-neural-network/lecture/Ud7t0/understanding-exponentially-weighted-averages

Exponentially Weight Average



Typically use v for velocity:

$$v_t = \beta v_{t-1} + (1-\beta)v_t$$

Where β controls smoothness

$$avg \approx \frac{1}{(1-\beta)} readings$$

β = 0.9 (~10 readings average)

β = 0.98 (~50 readings average) Big latency, but smooth

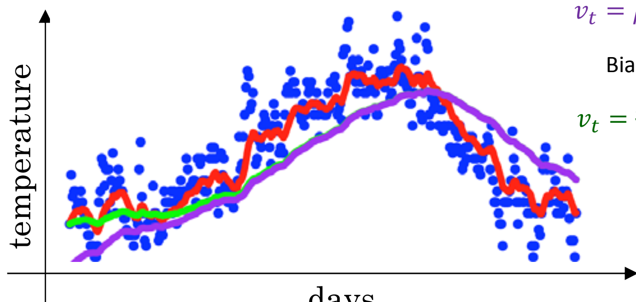
β = 0.5 (~2 readings average) Small latency, but overfit

- One big advantage of this method is that it takes up minimal memory

https://www.coursera.org/learn/deep-neural-network/lecture/Ud7t0/understanding-exponentially-weighted-averages

Exponentially Weight Average with Bias Correction

When β=0.98, we don't actually get green curve, we get purple curve if v_0=0



$$v_t = \beta v_{t-1} + (1-\beta)v_t$$

Bias correction:

$$v_t = \frac{\beta v_{t-1} + (1-\beta)v_t}{1-\beta^t}$$

$v_0 = 0$

$v_1 = 0.98v_0 + 0.02v_1$

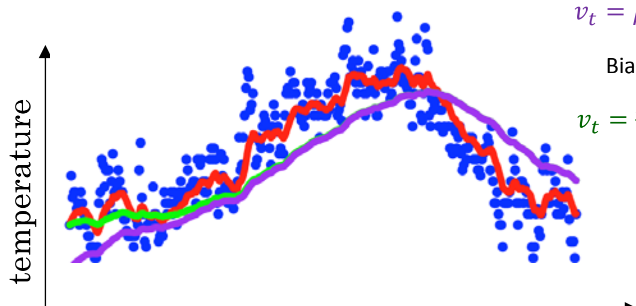
$t = 1\text{:}\ 1-\beta^t = 1 - 0.98^1 = 0.02$

$$\frac{v_1}{1-\beta^t} :: v_1 = \frac{0.98v_0 + 0.02v_1}{0.02} = v_1$$

Exponentially Weight Average with Bias Correction

When β=0.98, we don't actually get green curve, we get purple curve if v_0=0

$$v_t = \beta v_{t-1} + (1-\beta)v_t$$



Bias correction:

$$v_t = \frac{\beta v_{t-1} + (1-\beta)v_t}{1-\beta^t}$$

As t gets larger, β^t goes towards 0 and the green and purple lines coincide.

$v_0 = 0$

$v_1 = 0.98v_0 + 0.02v_1$

$v_2 = 0.98v_1 + 0.02v_2$

$v_2 = 0.98(0.02v_1) + 0.02v_2$

$v_2 = 0.0196v_1 + 0.02v_2$

$t = 1\text{:}\ 1-\beta^t = 1 - 0.98^1 = 0.02$

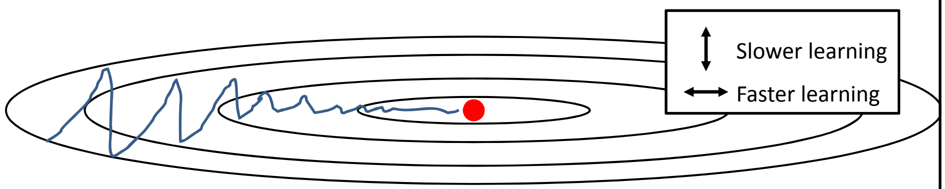
$t = 2\text{:}\ 1-\beta^t = 1 - 0.98^2 = 0.0396$

$$\frac{v_2}{1-\beta^t} :: v_2 = \frac{0.0196v_1 + 0.02v_2}{0.0396}$$

Since 0.0196+0.02=0.0396, v2 is now a weighted average of our two terms, removing our bias.

$v_2 = 0.495v_1 + 0.505v_2$

Gradient Descent with Momentum



- Typical gradient descent- oscillations slow down progress
- Further, if learning rate too large, initial step would cause significant overshooting, and perhaps divergence.

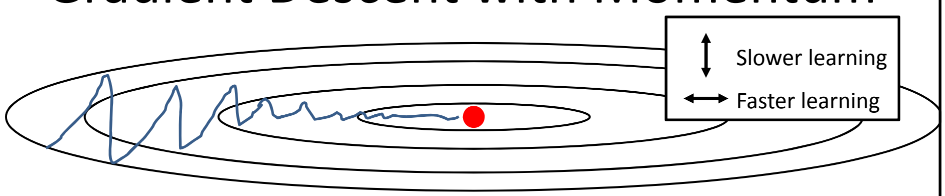
For each iteration, *t*:

Compute *dW*, *db* on current mini-batch

$$\left.\begin{aligned} V_{dW} &= \beta V_{dW} + (1-\beta)dW \\ V_{db} &= \beta V_{db} + (1-\beta)db \end{aligned}\right\}$$ Replace each partial derivative with a smoothed version

$$\left.\begin{aligned} W &= W - \eta V_{dW} \\ b &= b - \eta V_{db} \end{aligned}\right\}$$ Update using smoothed derivatives

Gradient Descent with Momentum



For each iteration, *t*:

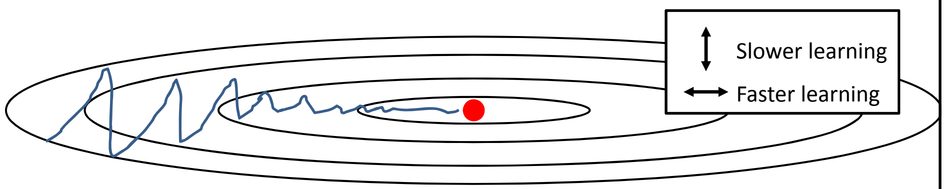
Compute *dW*, *db* on current mini-batch

$$\left.\begin{aligned} V_{dW} &= \beta V_{dW} + (1-\beta)dW \\ V_{db} &= \beta V_{db} + (1-\beta)db \end{aligned}\right\}$$ Replace each partial derivative with a smoothed version

$$\left.\begin{aligned} W &= W - \eta V_{dW} \\ b &= b - \eta V_{db} \end{aligned}\right\}$$ Update using smoothed derivatives

We end up with two hyperparameters, η and β (learning rate and momentum)
Very common to set β=0.9
Because moving average moves up quickly, bias correction not commonly used

Gradient Descent with Momentum



For each iteration, *t*:

Compute *dW*, *db* on current mini-batch

$V_{dw} = \beta V_{dw} + (1-\beta)dW$

$V_{db} = \beta V_{db} + (1-\beta)db$

$W = W - \eta V_{dw}$

$b = b - \eta V_{db}$

Sometimes approximated with:

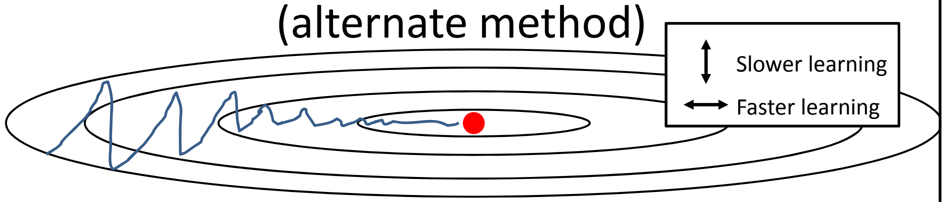
$V_{dw} = \beta V_{dw} + dW$

$V_{db} = \beta V_{db} + db$

We end up with two hyperparameters, η and β (learning rate and momentum)
Very common to set β=0.9
Because moving average moves up quickly, bias correction not commonly used

 66

Gradient Descent with Momentum (alternate method)



For each iteration, *t*:

Compute *dW*, *db* on current mini-batch

$V_{dw} = \beta V_{dw} - \eta dW$

$V_{db} = \beta V_{db} - \eta db$

Smoothed version of dW, but include learning rate to increase momentum more.

$W = W + V_{dw}$

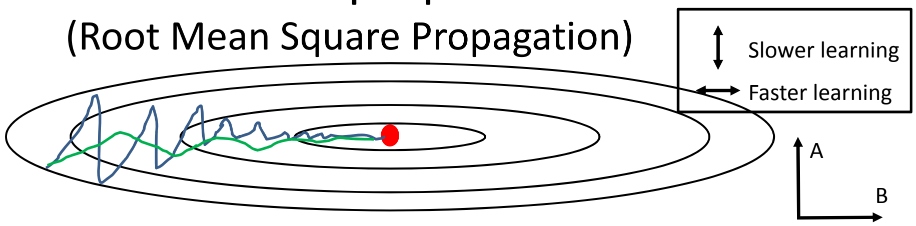
$b = b + V_{db}$

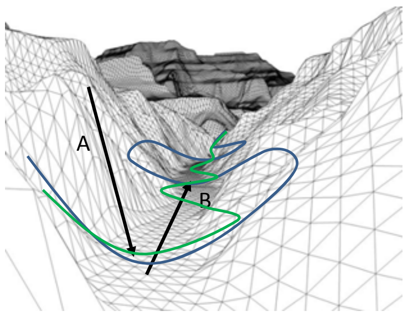
Since we were learning '-' *dW* above, we add, not subtract here.

We end up with two hyperparameters, η and β (learning rate and momentum)
Very common to set β=0.9
Because moving average moves up quickly, bias correction not commonly used

 67

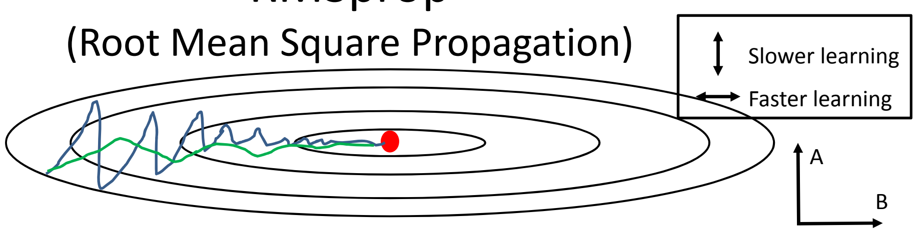
RMSprop
(Root Mean Square Propagation)





- In general the derivative in direction A will be large and B will be small.
- Weighted derivative squares in A/B will be large/small.
- When A/B large, want to dampen
- When A/B small, don't dampen as much

RMSprop
(Root Mean Square Propagation)



For each iteration, *t*:

Compute *dW*, *db* on current mini-batch

$S_{dW} = \beta S_{dW} + (1-\beta) dW .* dW$

$S_{db} = \beta S_{db} + (1-\beta) db .* db$

Keep track of exponentially weighted squares of derivatives

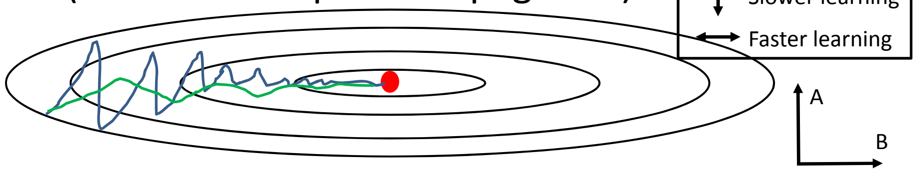
$$W = W - \eta \frac{dW}{\sqrt{S_{dW}}}$$

$$b = b - \eta \frac{db}{\sqrt{S_{db}}}$$

- In general the derivative in direction A will be large and B will be small.
- Weighted squares in A/B will be large/small.
- When A/B large, want to dampen
- When A/B small, don't dampen as much

RMSprop

(Root Mean Square Propagation)



For each iteration, *t*:

Compute *dW*, *db* on current mini-batch

$$S_{dW} = \beta_2 S_{dW} + (1-\beta_2) dW.* dW$$
$$S_{db} = \beta_2 S_{db} + (1-\beta_2) db.* db$$

Keep track of exponentially weighted squares of derivatives

$$W = W - \eta \frac{dW}{\sqrt{S_{dW} + \epsilon}}$$

$$b = b - \eta \frac{db}{\sqrt{S_{db} + \epsilon}}$$

- Next we will combine RMSprop with Momentum so lets rename β to β_2
- Also, lets make sure we don't divide by zero

Adam Optimization

Combine momentum with RMSprop with Bias Correction

V_{dW}=0, S_{dW}=0, V_{db}=0, S_{db}=0

For each iteration, *t*:

Compute *dW*, *db* on current mini-batch

momentum
$$V_{dW} = \beta_1 V_{dW} + (1-\beta_1) dW$$
$$V_{db} = \beta_1 V_{db} + (1-\beta_1) db$$

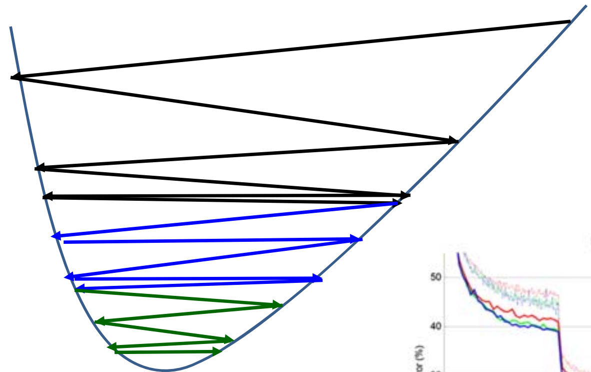
RMSprop
$$S_{dW} = \beta_2 S_{dW} + (1-\beta_2) dW.* dW$$
$$S_{db} = \beta_2 S_{db} + (1-\beta_2) db.* db$$

Hyperparameters to tune:
β_1: 0.9
β_2: 0.999
ϵ: 10^{-8}
η: need to solve

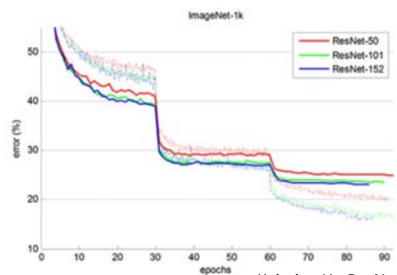
Bias Correction
$$V_{dW}^{Corrected} = V_{dW}/(1-\beta_1^t)\,; \quad V_{db}^{Corrected} = V_{db}/(1-\beta_1^t)$$
$$S_{dW}^{Corrected} = S_{dW}/(1-\beta_2^t)\,; \quad S_{db}^{Corrected} = S_{db}/(1-\beta_2^t)$$

$$W = W - \eta\left(V_{dW}^{Corrected} \Big/ \sqrt{S_{dW}^{Corrected} + \epsilon}\right); \quad b = b - \eta\left(V_{db}^{Corrected} \Big/ \sqrt{S_{db}^{Corrected} + \epsilon}\right)$$

Learning Rate Decay



- If shrink step size, we can find a better solution
- If shrink step size, we can find a better solution



Kaiming He ResNet

Learning Rate Decay

Initial learning rate

$$\eta = \frac{1}{1 + decayRate * epochNum}\eta_0$$

$\eta_0 = 0.2$

$decayRate = 1$

epochNum	η
1	0.1
2	0.067
3	0.05
4	0.04
…	…

Learning Rate Decay- Other formulas

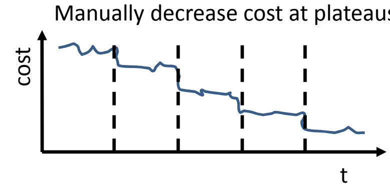
Some hyperparameter <1

$$\eta = 0.95^{epochNum} \cdot \eta_0$$ (Exponential decay)

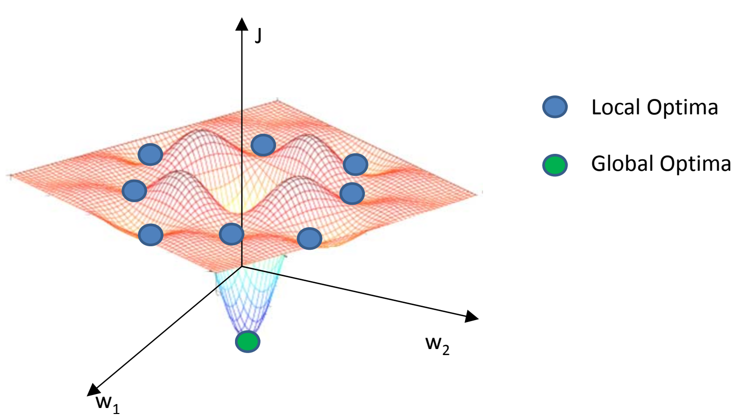
$$\eta = \frac{k}{\sqrt{epochNum}} \cdot \eta_0$$ --OR-- $$\eta = \frac{k}{\sqrt{t}} \cdot \eta_0$$ *t* can be custom to problem

Discrete staircase:
Divide learning rate in half every 100 iterations

Manually decrease cost at plateaus

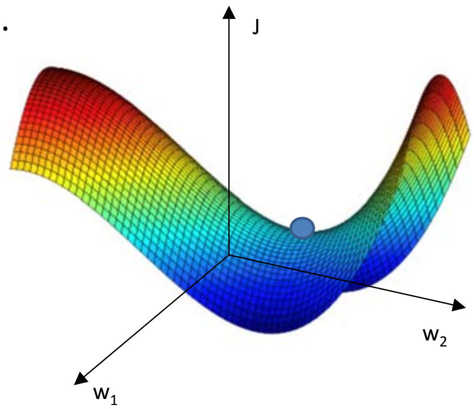


Local Optima in Neural Networks



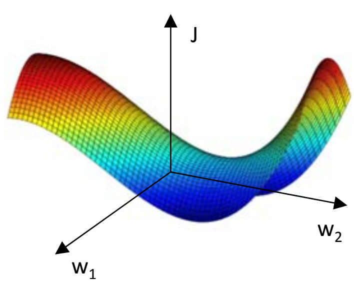
Local Optima in Neural Networks

- As luck would have it, most points with gradient zero are not like previous slide, but occur on saddle points.

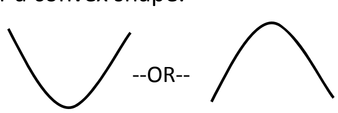


 76

Local Optima in Neural Networks



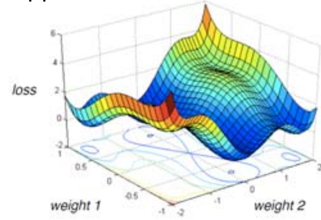
For a gradient to be zero, in each direction we either have a concave or a convex shape.



Now…in a very high dimensional space, say 10,000 dimensions, what are the odds all will be concave?

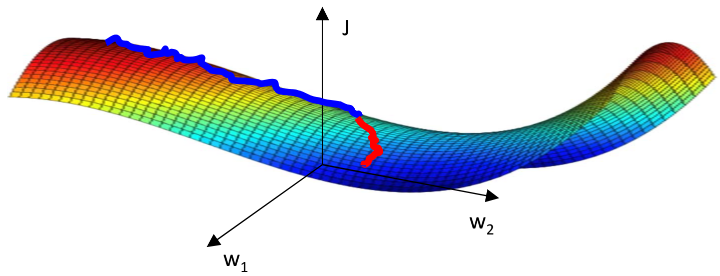
Pretty small!

We are much more likely to have some directions run up, and some run down, like the saddle point on the upper left or chaos below.



 77

Problem of Plateaus



- Given a surface like this, we traverse the cost function for a long time making little progress.
- Eventually, or perhaps by luck, we step to the side and finally begin to learn more aggressively again.
- But…thankfully in high dimensional spaces, unlikely to get stuck in local optima

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe SIOFFE@GOOGLE.COM
Christian Szegedy SZEGEDY@GOOGLE.COM
Google, 1600 Amphitheatre Pkwy, Mountain View, CA 94043

Abstract

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training*

minimize the loss

$$\Theta = \arg\min_{\Theta} \frac{1}{N}\sum_{i=1}^{N} \ell(\mathrm{x}_i, \Theta)$$

where $\mathrm{x}_{1\ldots N}$ is the training data set. With SGD, the training proceeds in steps, at each step considering a *mini-batch* $\mathrm{x}_{1\ldots m}$ of size m. Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch $\frac{1}{m}\sum_{i=1}^{m}\frac{\partial \ell(\mathrm{x}_i,\Theta)}{\partial \Theta}$ is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a mini-batch can be more efficient than m computations for individual exam-

http://jmlr.org/proceedings/papers/v37/ioffe15.pdf

Batch Normalization

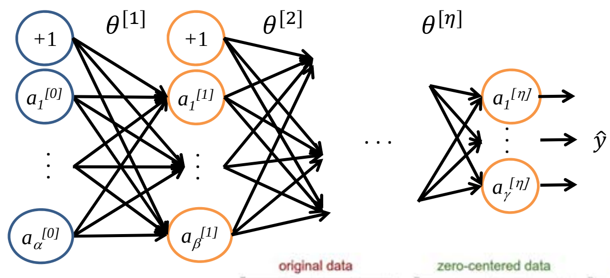
- Makes hyperparameter search easier
- Makes learning more robust to the choice of hyperparameters
- Makes learning work with a larger range of hyperparameters
- Enables easier and often faster training of very deep networks

Normalization of Features

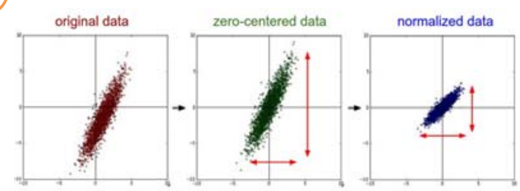
- We have seen how it is beneficial to subtract the mean and divide by the standard deviation for our input vector *X* (note, each dimension has its own μ and σ).
- In a deeper net, it would make the training of $\theta^{[l]}$ more efficient if the input to the layer, $a^{[l-1]}$ was also normalized.
- Note, in batch normalization, we will actually normalize *z*, not *a*. There is some disagreement on this, but most normalize *z*.

Implementing Batch Normalization

- Analogous to the way we normalized the input to $\theta^{[1]}$, normalize the input to all layers:

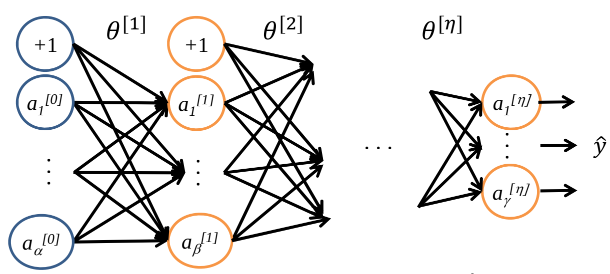


$$a^{[0]} = \frac{(a^{[0]} - \mu)}{\sigma}$$



Karpathy, cs231, NeuralNet_part2

Implementing Batch Normalization

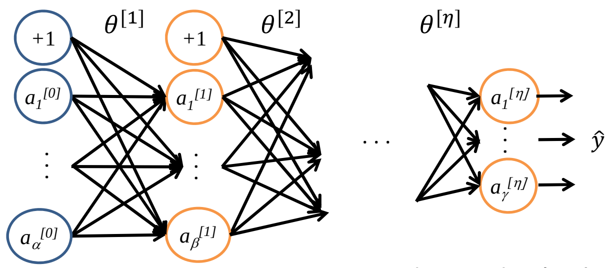


- As we then pass training samples through the net, replace $z^{[i]}$ with $z_{norm}^{[l]}$:

$$z_{norm}^{[l]} = \frac{\left(z^{[l]} - \mu^{[l]}\right)}{\sqrt{\sigma^{[l]2} + \epsilon}}$$

- Normalize *z*, not *a*
- Calculate μ and σ^2 from mini-batch training samples:
- $\mu^{[l]} = \frac{1}{n}\sum_i z^{[l](i)}$
- $\left(\sigma^{[l]}\right)^2 = \frac{1}{n}\sum_i (z^{[l](i)} - \mu^{[l]})^2$

Implementing Batch Normalization



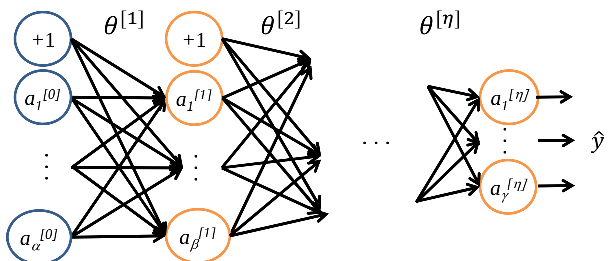
$$z_{norm}^{[l]} = \frac{\left(z^{[l]} - \mu^{[l]}\right)}{\sqrt{\sigma^{[l]2} + \epsilon}}$$

$$\tilde{z}^{[l]} = \gamma z_{norm}^{[l]} + \beta$$

- Then go through activation function

- But, do we don't always want our distribution to have mean=0 and unit variance.
- So we learn two new parameters, γ and β, these will be learned along with our $\theta^{[l]}$'s

Implementing Batch Normalization



$$z_{norm}^{[l]} = \frac{\left(z^{[l]} - \mu^{[l]}\right)}{\sqrt{\sigma^{[l]2} + \epsilon}}$$

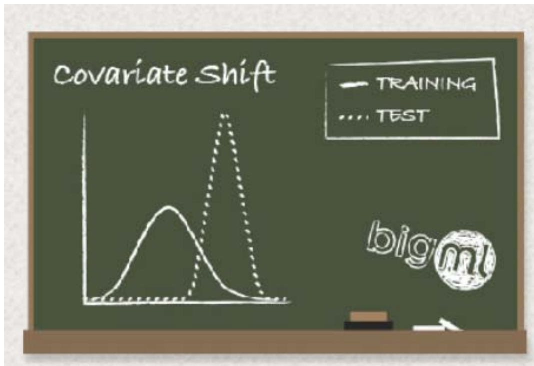
$$\tilde{z}^{[l]} = \gamma^{[l]} z_{norm}^{[l]} + \beta^{[l]}$$

- Then go through activation function

- Note $\gamma^{[l]}$ and $\beta^{[l]}$ are vectors for each layer, and get applied independently to each dimension.
- If using batch normalization with $\beta^{[l]}$, no need to use bias values as they both do the same job.

Batch Normalization Intuition

- When training set statistics differ from test set statistics, we call this covariate shift.



blog.bigml.com

Batch Normalization Intuition

- When training set statistics differ from test set statistics, we call this covariate shift.
- Typically we would have to relearn the model if we have a covariate shift, but batch norm minimizes covariate shift.
- In effect, it forces the input to each layer to have a predictable distribution, $\mu=0$, $\sigma=1$, or whatever distribution the γ and β determine works best for that layer.
- This predictable distribution makes later layers somewhat independent of earlier layers, enabling robust behavior even in deep networks.

Batch Normalization at Test Time

- Using mini-batch training samples, we calculate μ and σ^2 for each layer:
- $\mu^{[l]} = \frac{1}{n}\sum_i z^{[l](i)}$
- $\left(\sigma^{[l]}\right)^2 = \frac{1}{n}\sum_i (z^{[l](i)} - \mu^{[l]})^2$

$$z_{norm}^{[l]} = \frac{\left(z^{[l]} - \mu^{[l]}\right)}{\sqrt{\sigma^{[l]2} + \epsilon}}$$

$$\tilde{z}^{[l]} = \gamma z_{norm}^{[l]} + \beta$$

- During test, if we were testing a mini-batch at once, we could use the test mini-batch to calculate μ and σ.
- Alternately, if only have one or a few samples you need an estimate for μ and σ.
  - Can compute during training using a full epoch.
  - Can use an exponentially weighted average for each:

$$v_t = \beta v_{t-1} + (1 - \beta) v_t$$

Note: this β is from exponentially weighted average and not related to batch norm β!

deeplearning.ai | coursera

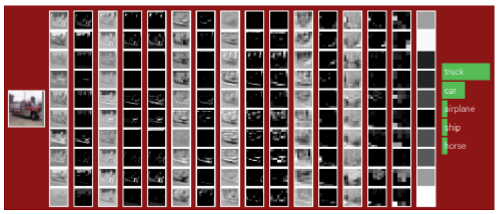
Andrew Ng, 2017

https://www.deeplearning.ai/

Deep Learning Specialization, Five courses:

1. Neural Networks and Deep Learning
2. Improving Deep Neural Networks
3. Structured Machine Learning Projects
4. Convolutional Neural Networks
5. Sequence Models

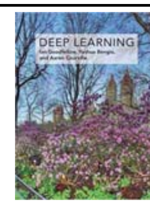
CS231n: Convolutional Neural Networks for Visual Recognition



Li, Johnson, Yeung 2017

http://cs231n.stanford.edu/ l

Books on Deep Learning



- **Grokking Deep Learning** by Andrew Trask. Use Udacity discount code **traskud17** for 40% off. This provides a very gentle introduction to Deep Learning and covers the intuition more than the theory.
  - https://www.manning.com/books/grokking-deep-learning ($49)
- **Neural Networks And Deep Learning** by Michael Nielsen. This book is more rigorous than Grokking Deep Learning and includes a lot of fun, interactive visualizations to play with.
  - http://neuralnetworksanddeeplearning.com/ (free!)
- **The Deep Learning Textbook** from Ian Goodfellow, Yoshua Bengio, and Aaron Courville. This online book contains a lot of material and is the most rigorous of the three books suggested.
  - http://www.deeplearningbook.org/ (html- free!)
  - https://www.amazon.com/Deep-Learning-Adaptive-Computation-Machine/dp/0262035618/ (hard copy $50)

Thank you!!

Ray Ptucha
rwpeec@rit.edu

https://www.rit.edu/mil