We will learn about optimization algorithms to train NN much faster.

deeplearning.ai

Optimization Algorithms

Mini-batch gradient descent

Batch vs. mini-batch gradient descent

X, Y $\quad$ $X^{\{t\}}, Y^{\{t\}}$.

Vectorization allows you to efficiently compute on m examples.

$$X = [\underbrace{x^{(1)}\ x^{(2)}\ x^{(3)} \dots x^{(1000)}}_{X^{\{1\}}\ (n_x,1000)} \mid \underbrace{x^{(1001)} \dots x^{(2000)}}_{X^{\{2\}}\ (n_x,1000)} \mid \dots \mid \underbrace{\dots x^{(m)}}_{X^{\{5,000\}}\ (n_x,1000)}]$$

(n_x, m)

$$Y = [\underbrace{y^{(1)}\ y^{(2)}\ y^{(3)} \dots y^{(1000)}}_{Y^{\{1\}}\ (1,1000)} \mid \underbrace{y^{(1001)} \dots y^{(2000)}}_{Y^{\{2\}}\ (1,1000)} \mid \dots \mid \underbrace{\dots y^{(m)}}_{Y^{\{5,000\}}\ (1,1000)}]$$

$(1, m)$

Even with vectorization m can be very large and make training slow.

it has to process 5 milion for grad decent to take a step, every time it takes a step. You can impove this by letting grad descent make some progress before processing the entire training set.

What if $m = 5,000,000$?

5,000 mini-batches of 1,000 each

Mini-batch t: $X^{\{t\}}, Y^{\{t\}}$.

we split training set into mini batches.

we using curly braces for denotation of mini batch trainin set.

$x^{(i)}$ $\quad$ we use this () brackets to denote training set

$z^{[l]}$ $\quad$ we use [] brackets to denote the layer of NN

$X^{\{t\}}, Y^{\{t\}}$. $\quad$ now we use {} to denote mini batch

Andrew Ng

Mini-batch gradient descent

repeat {
for $t = 1, \ldots, 5000$ {

1 step of grad desc using $X^{\{t\}}, Y^{\{t\}}$ (as if m=1000)

Forward prop on $X^{\{t\}}$.

$$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$\vdots$$
$$A^{[L]} = g^{[L]}(Z^{[L]})$$

Vectorized implementation (1000 examples)

X, Y

WHEN YOU HAVE A LARGE TRAINING SET MINI BATCH RUNS MUCH FASTER, AND ITS WHAT EVERYONE WOULD USE. NEXT WHE SEE DEEPER WHAT I DOES AND WHY IT WORKS.

from $X^{\{t\}}, Y^{\{t\}}$.

Compute cost $J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{l} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_{l} \|W^{[l]}\|_F^2$.

Backprop to compute gradients wrt $J^{\{t\}}$ (using $(X^{\{t\}}, Y^{\{t\}})$)

$W^{[l]} := W^{[l]} - \alpha dW^{[l]}$, $b^{[l]} := b^{[l]} - \alpha db^{[l]}$

}

"1 epoch" — pass through training set.

"1 epoch" which means one pass through the training set.

}

you need this loop to converge

Andrew Ng

deeplearning.ai

Optimization Algorithms

Understanding mini-batch gradient descent

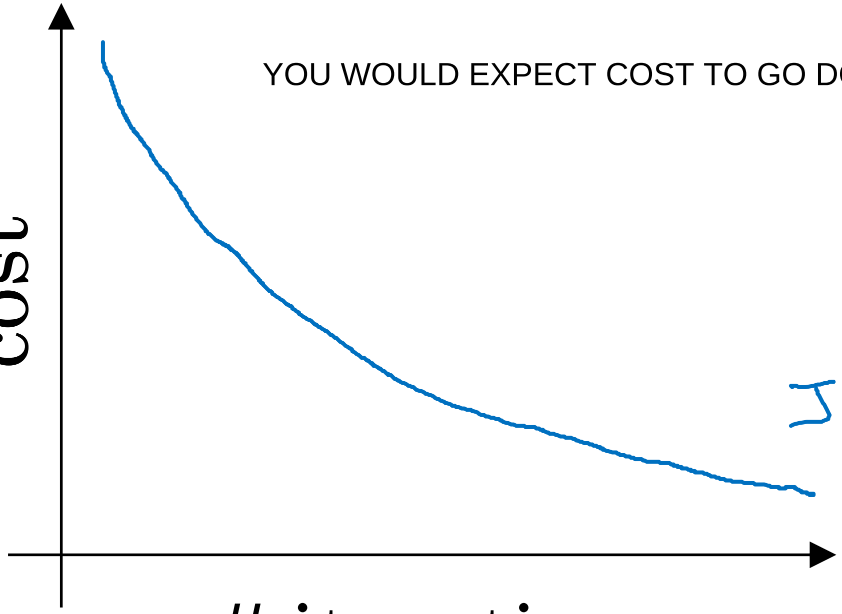
Training with mini batch gradient descent

IN MINI BATCH DOES NOT GO JUST GO DOWN ON EVERY ITERATION
IT SHOULD TREND DOWN.

Batch gradient descent

YOU WOULD EXPECT COST TO GO DOWN ON EVERY ITERATION

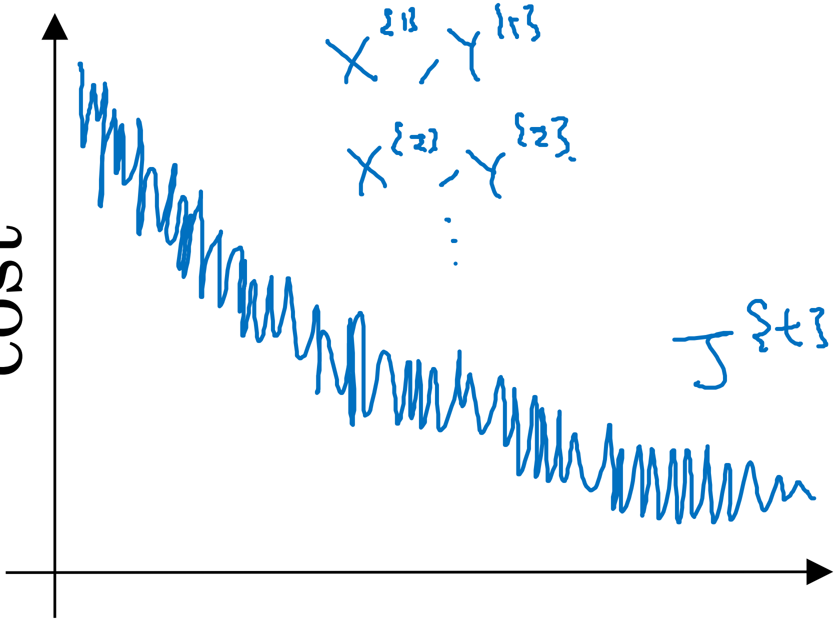
cost



iterations

Mini-batch gradient descent

cost



mini batch # (t)

Plot $J^{\{t\}}$ computed using $\underline{X^{\{t\}}, Y^{\{t\}}}$

Andrew Ng

Choosing your mini-batch size

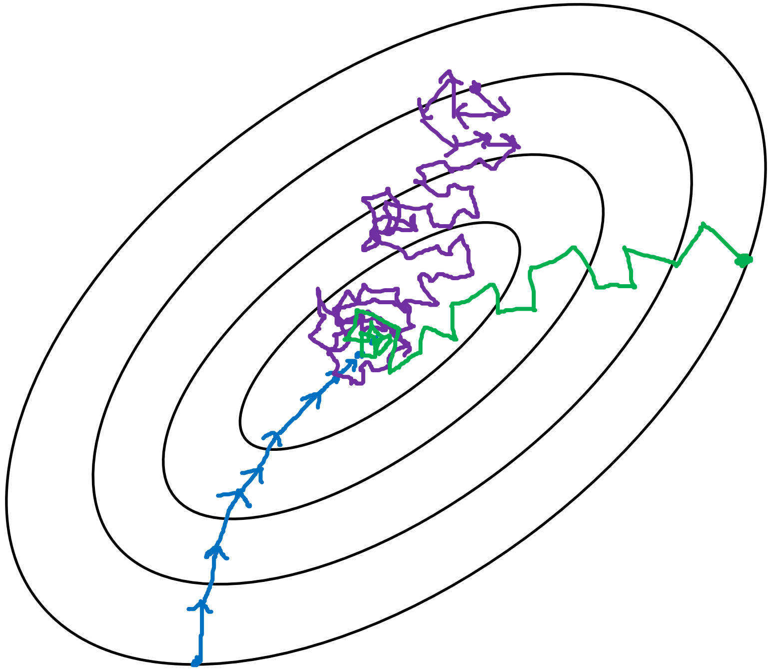
→ If mini-batch size = m : Batch gradient descent. $(X^{\{1\}}, Y^{\{1\}}) = (X, Y)$.

→ If mini-batch size = 1 : Stochastic gradient descent. Every example is it own mini-batch.
$(X^{\{1\}}, Y^{\{1\}}) = (x^{(1)}, y^{(1)})$ ... $(x^{(2)}, y^{(1)})$ mini-batch.

In practice: Somewhere in-between 1 and m

iN PRACTICE U USE SOMETHINK IN BETWEEN 1 AND M

WHAT WE DO IS TO USE STH IN BETWEEN



Stochastic gradient descent

Lose speedup from vectorization

LOOSE SPEED FROM VECTORIZATION

In-between (mini-batch size not too big/small)

Fastest learning.

- Vectorization. (~1000)
- Make progress without processing entire training set.

Batch gradient descent (mini-batch size = m)

Too long per iteration

MAKE PROGRESS WITHOUT NEEDING TO END ENTIRE TRAINING SET

Andrew Ng

Choosing your mini-batch size

SO HOW DO YOU CHOOSE M IT SHOULD NOT BE M OR 1 ??????

If small trainig set: Use batch gradient descent.
($m \leq 2000$)

IF M LESS THEN 2000 ITS FINE TO JUST USE BATCH GRAD DESCENT

IF WE HAVE A SMALL TRAINING SE WE JUST USE BATCH GRAD DESCENT.

Typical mini-batch sizes:

IT RUNS FASTER IF U CHOOSE M AS POWER OF 2, HERE SOME EXAMPLES

$\rightarrow \underbrace{\underset{2^6}{64}, \underset{2^7}{128}, \underset{2^8}{256}, \underset{2^9}{512}}$ $\quad \underset{2^{10}}{1024}$

Make sure mini-batch fit in CPU/GPU memory.

$X^{\{t\}}, Y^{\{t\}}$

THIS EPOCH HAS TO FIT TO THE CPU

IT TURNS OUT THERE ARE EFFICIENT ALGOS THAN BATCH GRAD DESCENT OR MINI BATCH GRAD DESCENT. THATS CHECK THEM OUT. NEST VIDEO

Andrew Ng

WE WILL SEE SOME ALGOS THAT ARE FASTER THEN GRAD DESCENT. TO UNDESTAND THEM U NEED TO UNDERSTAND EXPONENTIALLY WEIGHTED AVG

deeplearning.ai

Optimization Algorithms

Exponentially weighted averages

Temperature in London

$\theta_1 = 40°F$ 4°C ←

$\theta_2 = 49°F$ 9°C

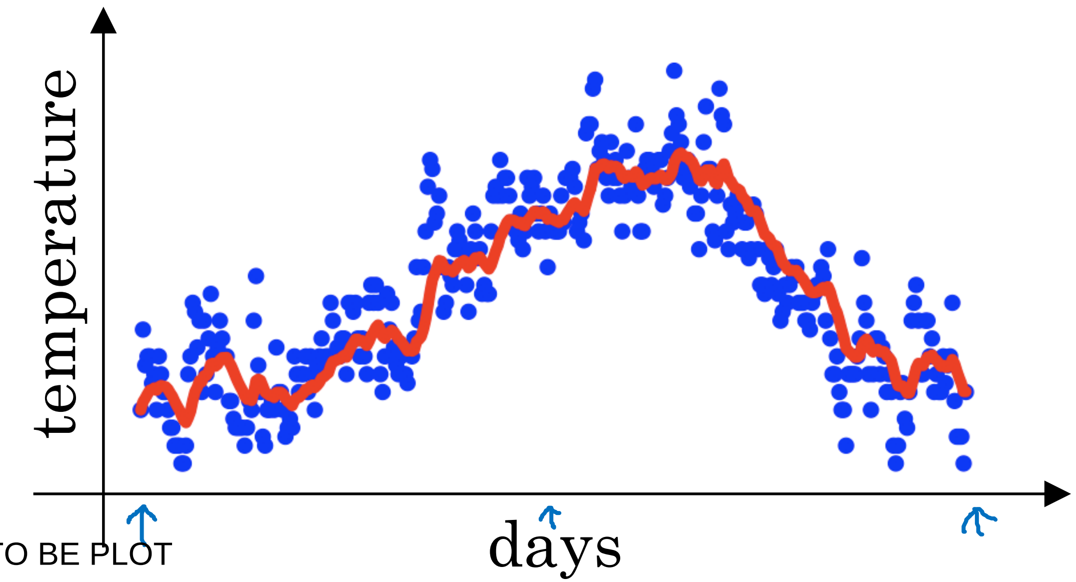
$\theta_3 = 45°F$ ⋮

⋮

$\theta_{180} = 60°F$ 15°C

$\theta_{181} = 56°F$ ⋮

⋮



THIS V IS FOR CALC THE
EXPONENTIAL MOVING AVG TO BE PLOT

$V_0 = 0$

$V_1 = 0.9\, V_0 + 0.1\, \theta_1$

$V_2 = 0.9\, V_1 + 0.1\, \theta_2$

$V_3 = 0.9\, V_2 + 0.1\, \theta_3$

⋮

$$\boxed{V_t = 0.9\, V_{t-1} + 0.1\, \theta_t}$$

Andrew Ng

Exponentially weighted averages
moving

$V_t = \beta V_{t-1} + (1-\beta)\theta_t$

$\beta = 0.9$: ≈ 10 days' temperature.

WE CAN THINK OF THIS AS AVG 10 DAYS TIME

$\beta = 0.98$: ≈ 50 days

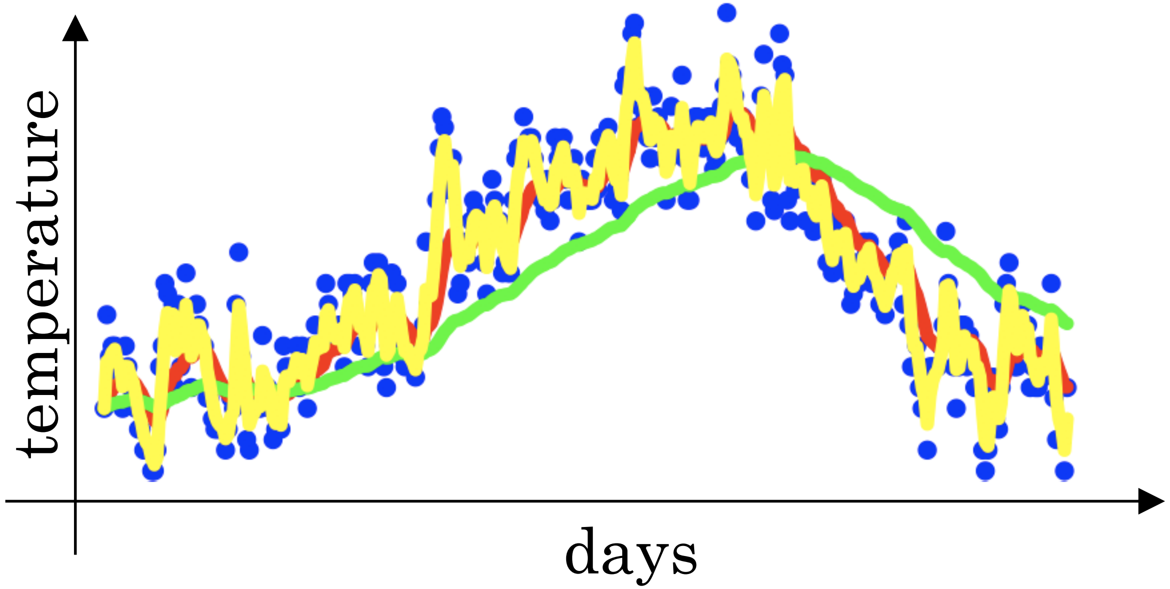
$\beta = 0.5$: ≈ 2 days

VERY DEPENDENT TO TEPERATURE CHANGES

U CAN THINK OF Vt AS AVERAGING OVER 1/1-beta DAYS TEMPERATURE.

V_t as approximately average over $\approx \frac{1}{1-\beta}$ days' temperature.

$\frac{1}{1-0.98} = 50$



Andrew Ng

deeplearning.ai

Optimization Algorithms

Understanding exponentially weighted averages

Exponentially weighted averages

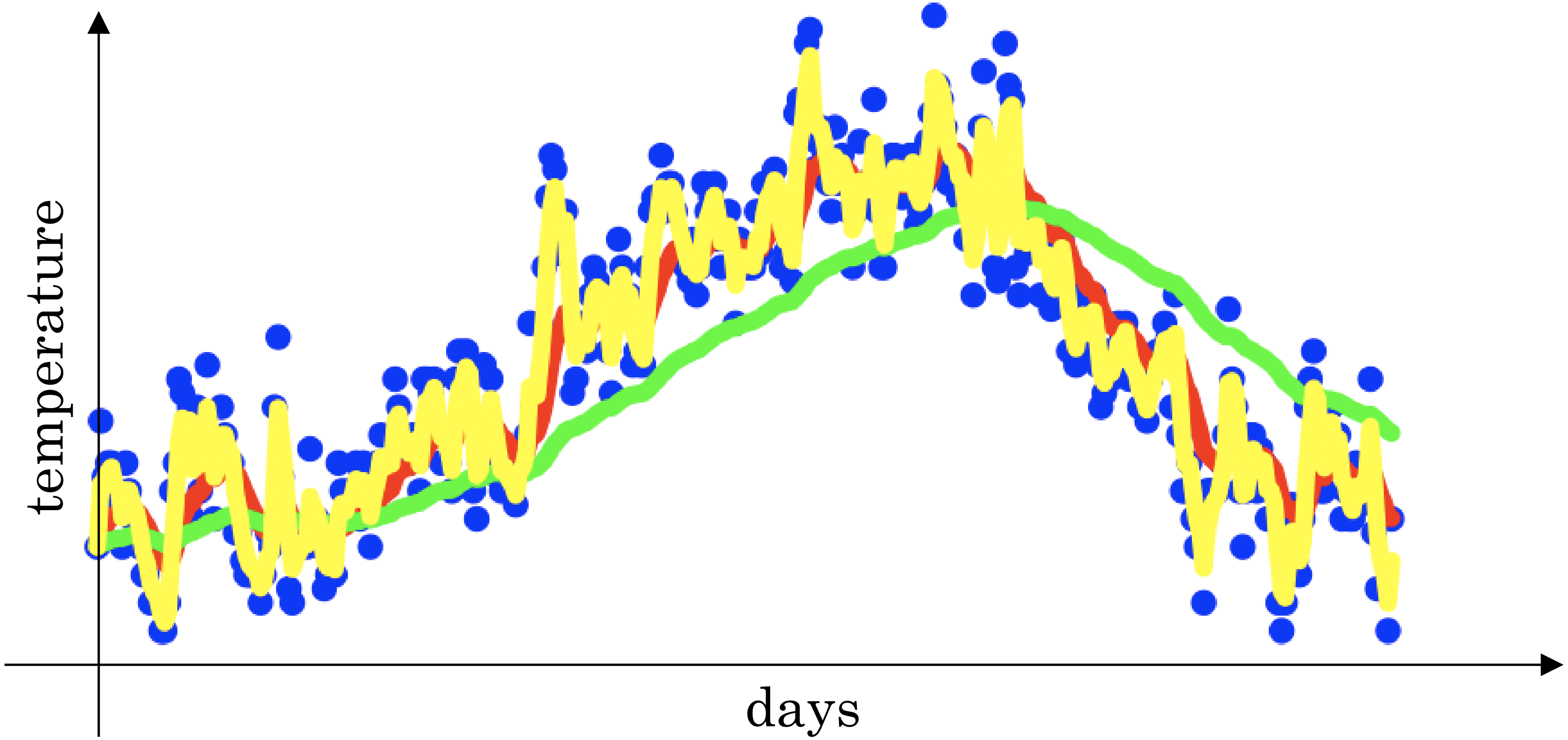
THIS WILL TURN OUT TO BE A KEY COMPONENT OF SEVERAL OPTIMISATION ALGOS TO TRAIN NN

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

$\beta = 0.9$ RED LINE

0.98 GREEN

0.5 YELLOW



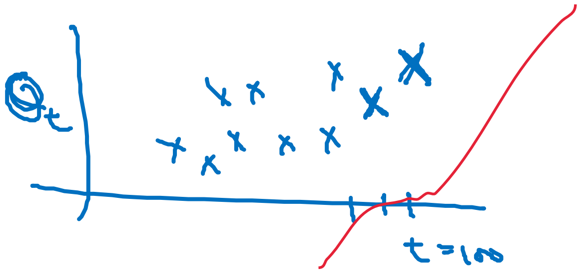
Andrew Ng

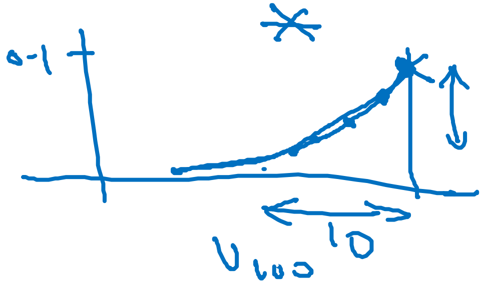
Exponentially weighted averages

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

THETA T
(TEMPERATURE
OF DAY T)

ELEMENTWISE MULTIPLICATION
TO GET V100





$\approx \frac{1}{1-\beta}$

$\varepsilon = 1-\beta$

$$v_{100} = 0.9v_{99} + 0.1\theta_{100}$$

$$\boxed{v_{99} = 0.9v_{98} + 0.1\theta_{99}}$$

$$\boxed{v_{98} = 0.9v_{97} + 0.1\theta_{98}}$$

...

$0.1\,\theta_{98} + 0.9\,V_{97}$

$V_{100} = 0.1\,\theta_{100} + 0.9\,\cancel{V_{99}}\,(0.1\,\theta_{99} + 0.9\,\cancel{V_{98}})$

THATS UNDERSTAND
WHAT V 100 IS

$= 0.1\,\theta_{100} + 0.1\times 0.9\cdot\theta_{99} + 0.1\,(0.9)^2\theta_{98} + 0.1\,(0.9)^3\theta_{97} + 0.1\,(0.9)^4\theta_{96}$
$+ \dots$

$0.9^{10} \approx 0.35 \approx \frac{1}{e}$

$\underbrace{(1-\varepsilon)}_{0.9}{}^{1/\varepsilon} = \frac{1}{e}$

0.98 ?

$\varepsilon = 0.02 \rightarrow 0.98^{50} \approx \frac{1}{e}$

Andrew Ng

Implementing exponentially weighted averages

$$v_0 = 0$$
$$v_1 = \beta v_0 + (1-\beta)\,\theta_1$$
$$v_2 = \beta v_1 + (1-\beta)\,\theta_2$$
$$v_3 = \beta v_2 + (1-\beta)\,\theta_3$$
$$\ldots$$

$V_\theta := 0$

$V_\theta := \beta v + (1-\beta)\,\theta_1$

$V_\theta := \beta v + (1-\beta)\,\theta_2$

$\vdots$

$\rightarrow V_\theta = 0$

Repeat {

Get next θ_t

$V_\theta := \beta V_\theta + (1-\beta)\,\theta_t \leftarrow$

}

Andrew Ng

deeplearning.ai

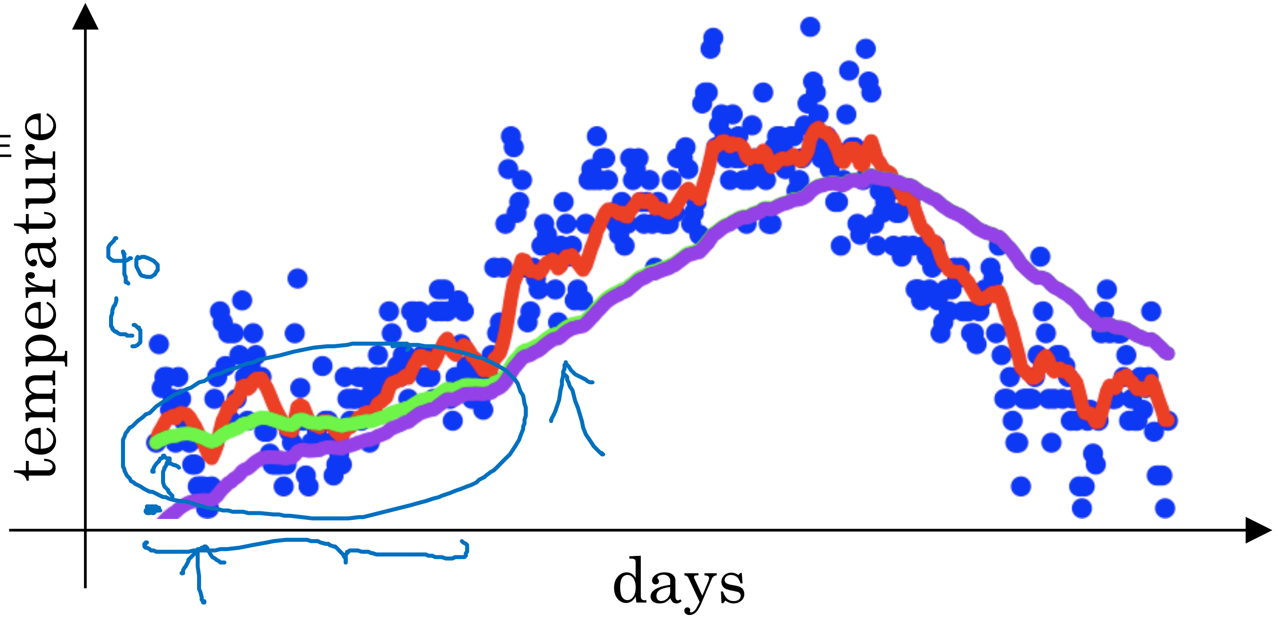
Optimization Algorithms

THIS CAN MAKE THE COMPUTATIONS OF THE AVERAGES MORE ACCURATE

Bias correction in exponentially weighted average

Bias correction

WE DONT GET THE GREEN CURVE WE GET THE PURPLE CURVE. WE NOTICE IT STARTS VERY LOW. THE PROBLE IS THAT WE NEED A BIAS TERM FOR INITIAL VALUS OF V



$\beta = 0.98$

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

$V_0 = 0$

$V_1 = \cancel{0.98 V_0 +} \; 0.02\,\theta_1$

NOT GOOD ESTIMATE

$V_2 = 0.98\, V_1 + 0.02\,\theta_2$

$= 0.98 \times 0.02 \times \theta_1 + 0.02\,\theta_2$

$= 0.0196\,\theta_1 + 0.02\,\theta_2$

V2 NOT GOOD ESTIMATE OF FIRST TWO DAY OF YEAR

$$\frac{V_t}{1-\beta^t}$$

SO THERE IS A WAY TO MODIFY THIS ESTIMATE THAT MAKES IT MUCH BETTER. INSEAD OF TAKING VT WE TAKE THIS ON LEFT. AS T BECOMES LARGE BETA TO THE T BECOMES 0

$t=2: \quad 1-\beta^t = 1-(0.98)^2 = 0.0396$

$$\frac{V_2}{0.0396} = \frac{0.0196\,\theta_1 + 0.02\,\theta_2}{0.0396}$$

NOW WE TRY BUILD BETTER OPTIMIZATION ALGORITHMS

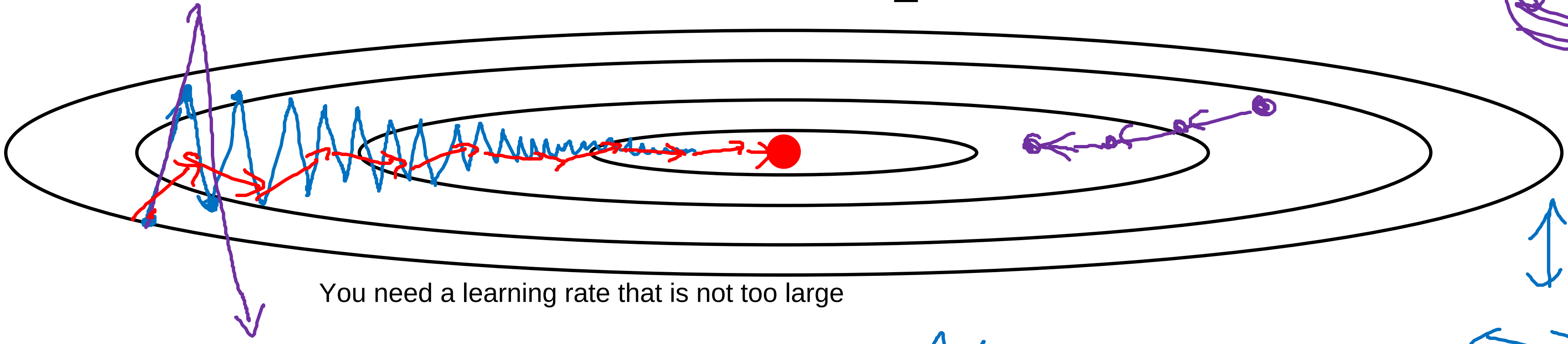
deeplearning.ai

Optimization Algorithms

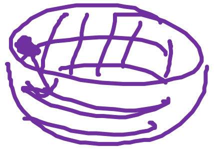
Gradient descent with momentum

ALMOST ALLWAYS WORKS BETTER THEN THE CLASSIC GRADIENT DESCENT ALGO
THE BASIC IDEA IS TO COMPUTE AN EXPONENTIAL WEIGHTED AVG OF YOUR GRADIENTS
AND THEN USE THAT GRADIENT TO UPDATE YOUR WEIGHTS

Gradient descent example



You need a learning rate that is not too large



↕ slower learning

↔ faster learning.

another way to view the learning rate problem is that you want the learning to be low on vertical and fast in horizontal

Momentum:

On iteration t:

Compute dW, db on current mini-batch.

on current mini-batch
works also for batch

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

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

friction ← velocity ← acceleration

"$V_\theta = \beta V_\theta + (1-\beta)\,\theta_t$"

IT ACCELERATE DOWN THE BOWL WITH A MOMENTUM

$$W := W - \alpha V_{dW}, \quad b := b - \alpha V_{db}$$

deeplearning.ai

SO WHAT THIS DOES IS SMOOTHS OUT STEPS OF GRADIENT DESCENT. FOR EX

Andrew Ng

Implementation details

SO THATS SEE SOME DETTAILS ON HOW TO IMPLEMENT IT

$V_{dw} = 0$, $V_{db} = 0$

On iteration t:

Compute dW, db on the current mini-batch

$$\to v_{dW} = \beta v_{dW} + \cancel{(1-\beta)}\, dW$$

$$\to v_{db} = \beta v_{db} + \cancel{(1-\beta)}\, db$$

$$W = W - \alpha v_{dW}, \;\; b = b - \alpha v_{db}$$

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

often you see it with the term of 1- beta ommited

he does not prefer this formulation. Formula on left is much better

$\cancel{\dfrac{V_{dw}}{1-\beta^t}}$

in practice people dont do bias correction becouse after just 10 iteration your moving avg will have warmed up and its no longer a biased estimator.

Hyperparameters: α, β

$\beta = 0.9$

average over last ≈ 10 gradients

the most common beta is 0.9 averagin over tha last 10 iteration gradients It works really well.

Andrew Ng

THIS IS ANOTHER ALGO THAT SPEEDS UP GRAD DESCENT
ROOT MEAN SQUARE PROP THAT CAN SPEED UP GRAD DESCENT

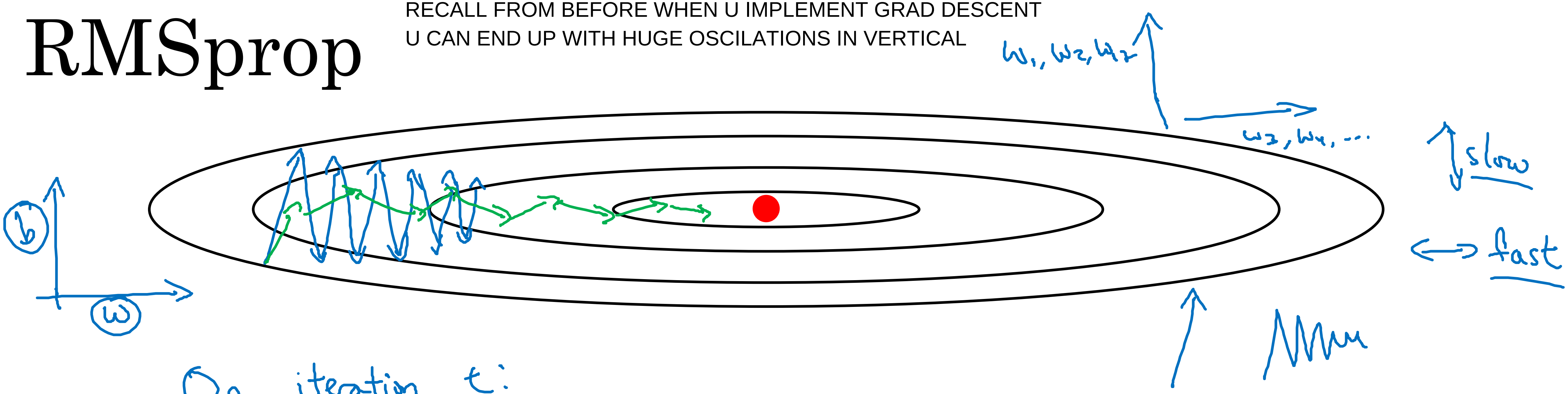
deeplearning.ai

Optimization Algorithms

RMSprop

RMSprop

RECALL FROM BEFORE WHEN U IMPLEMENT GRAD DESCENT
U CAN END UP WITH HUGE OSCILATIONS IN VERTICAL



On iteration t:

Compute dW, db on current mini-batch

WE JUST GONE USE Sdw instead of Vdw

element-wise

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2)\, dW^2 \leftarrow \text{small}$$

$$\rightarrow S_{db} = \beta_2 S_{db} + (1-\beta_2)\, db^2 \leftarrow \text{large}$$

SEPARATION OF W AND b IS JUST A ILLUSTRATION AS IN PRACTICE dW IS A HIGH DIMENTIONAL PARAMETER VECTOR AND dB ALSO.

THE DIFFERENCE IN THE RMSprop IS IN THE UPDATE WHEN WE DO IT LIKE THIS:

$$W := W - \alpha \frac{dW}{\sqrt{S_{dw}+\varepsilon}} \qquad b := b - \alpha \frac{db}{\sqrt{S_{db}+\varepsilon}}$$

TO MAKE SURE NUMERICAL STABILITY YOU ADD THIS SMALL AMOUNT $\varepsilon = 10^{-8}$

ANOTHER THINK IS THAT WITH THIS U CAN USE A LARGER LEARNING RATE ALFA AND SPEED IT UP EVEN MORE

ITS CALLED ROOT MEAN SQUARE BECOUSE YOU ARE SQUARING THE DERIVATIVES AND THEN YOU ARE TAKING THE ROOT IN THE END

NEXT WE GONE COMBINE MOMENTUM WITH RMSprop

Andrew Ng

deeplearning.ai

Optimization Algorithms

Adam optimization algorithm

SO THIS ALGO IS BASICALLY TAKING MOMENTUM AND RMS AND PUTTING THEM TOGETHER.

Adam optimization algorithm

$V_{dw} = 0, \; S_{dw} = 0. \quad V_{db} = 0, \; S_{db} = 0$

YOU MAKE THIS INITIALIZATION

On iteration t:

Compute dW, db using current mini-batch (U USUALLY DO THIS WITH MINI BATCH)

$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dW, \quad V_{db} = \beta_1 V_{db} + (1-\beta_1) db \leftarrow$ "momentum" β_1

$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dW^2, \quad S_{db} = \beta_2 S_{db} + (1-\beta_2) db \leftarrow$ "RMSprop" β_2

yhat = np.array([.9, 0.2, 0.1, .4, .9])

IN THE ADAM WE DO BIAS CORRECTION

$V_{dw}^{corrected} = V_{dw} / (1-\beta_1^t), \quad V_{db}^{corrected} = V_{db} / (1-\beta_1^t)$

$S_{dw}^{corrected} = S_{dw} / (1-\beta_2^t), \quad S_{db}^{corrected} = S_{db} / (1-\beta_2^t)$

HERE WE DO THE UPDATE

$$W := W - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}} + \varepsilon} \qquad b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \varepsilon}$$

SO THIS ALGO COMBINES GRAD DESCENT WITH MOMENTUM TOGETHER WITH GRAD DESCENT WITH RMSprop

Andrew Ng

Hyperparameters choice:

SO THE ADAM ALGO HAS QUITE A NUMBER OF HYPERPARAMETER

→ α : needs to be tune

TRY RANGE OF VALUES AS WE HAVE SEEN

→ β_1 : 0.9 → $(\underline{dw})$

THIS IS DEFAULT CHOICE

→ β_2 : 0.999 → $(\underline{dw^2})$

ADAM PAPER INVETORS RECOMAND THIS ONE.

→ ε : 10^{-8}

THI IS OK LIKE THIS. YOU REALLY DONT NEED TO SET IT AND DOES NOT AFFECT PERFORMANCE THAT MUCH. NO ONE TUNES EPSILON.

Adam: Adaptive moment estimation

Adam Coates

Andrew Ng

deeplearning.ai

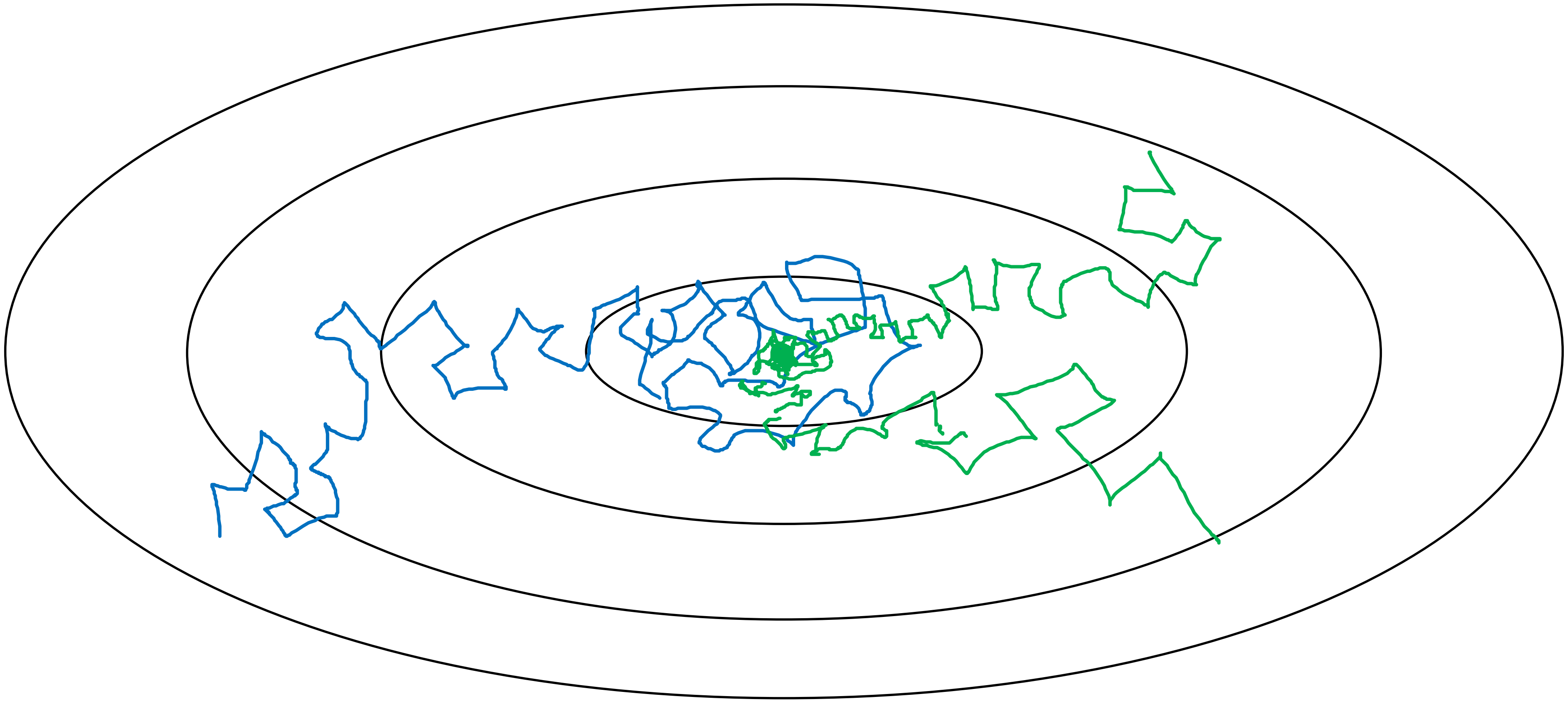
Optimization Algorithms

Learning rate decay

Learning rate decay

ONE OF THE THINGS THAT MIGHT SPEED UP U LERNING ALGO IS LEARNING DECAY TO SLOWLY REDUCE YOUR LEARNING RATE OVER TIME. WE CALL THIS LEARNING RATE DECAY.

Slowly reduce α



SUPPOSE YOU ARE IMPLEMENTING MINI BATCH WITH 64, 128 EXAMPLES, AS U ITERATE THE STEPS WILL BE A BIT NOISY AND IT WILL TEND TO THE MINIMUM BUT IT WONT EXACTLY CONVERGE, IT WILL WONDER AROUND AND WILL NEVER REALLLY CONVERGE BECOUSE YOU ARE USING A FIXED ALFA.
BUT IF YOU SLOWLY REDUCE ALFA THEN YOU ARE GONE HAVE FAST LEARNING AT BEGINING AND AS ALFA GETS SMALLER YOUR STEPS YOU TEKA WILL BE SMALLER AND YOU LEARN SLOWLY SO YOU END UP OSCILATING IN A TIGHTER REAGION.

Andrew Ng

Learning rate decay

HERE ITS HOW YOU CAN IMPLEMENT LEARNING RATE DECAY

TRAINING SET

$X^{\{1\}}$	$X^{\{2\}}$	...

→ epoch 1

→ epoch 2

⋮

1 epoch = 1 pass through data.

1 EPOCH = 1 PASS THROUGH THE DATA

THIS IS THE FORMULA HOW TO IMPLEMENT LEARNING DECAY

$$\alpha = \frac{1}{1 + \underline{\text{decay-rate}} * \text{epoch-num}} \underline{\alpha_0}$$

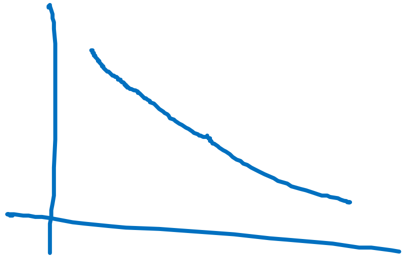
ONE PASS THROUGH THE TRAINING SET IS CALLED ONE EPOCH

YOU SET ONE INITIAL LEARNING RATE

$\alpha_0 = 0.2$

decay.rate = 1

Epoch	α
1	0.1
2	0.67
3	0.5
4	0.4
⋮	⋮



Andrew Ng

Other learning rate decay methods

THESE ARE OTHER METHOND PEOPLE IMPLEMENT LEARNING RATE DECAY

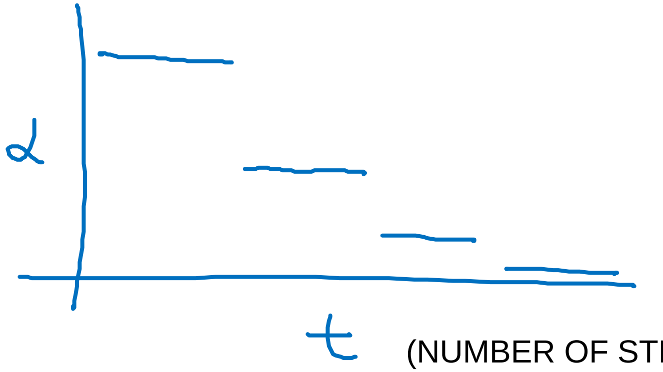
formula

$$\alpha = 0.95^{\text{epoch-num}} \cdot \alpha_0$$ – exponentially decay.

THIS WILL EXPONENTIALLY DECAY THE LEARNING RATE

$$\alpha = \frac{k}{\sqrt{\text{epoch-num}}} \cdot \alpha_0 \quad \text{or} \quad \frac{k}{\sqrt{t}} \cdot \alpha_0$$

THIS IS ANOTHER MEHTOD



(NUMBER OF STEPS)

discrete staircase

THE LEARNING RATE DECREASES AFTER A CERTAIN NUMBER OF STEPS

Manual decay.

- OTHER THEN USING FORMULA PEOPLE DO ALSO A MANUAL DECAY, IF IT TAKES MANY HOURS OR EVEN MANY DAYS TO TRAIN.

Andrew Ng

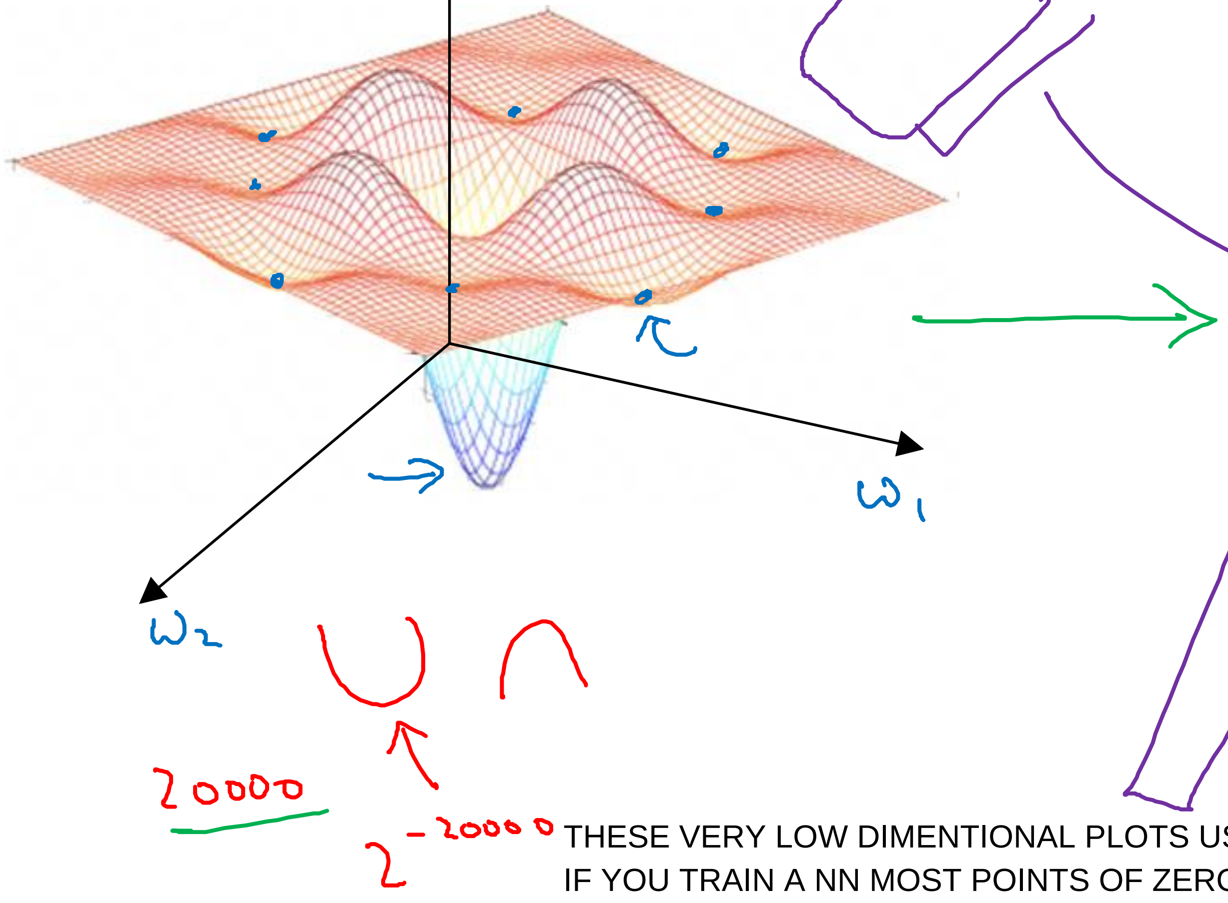
deeplearning.ai

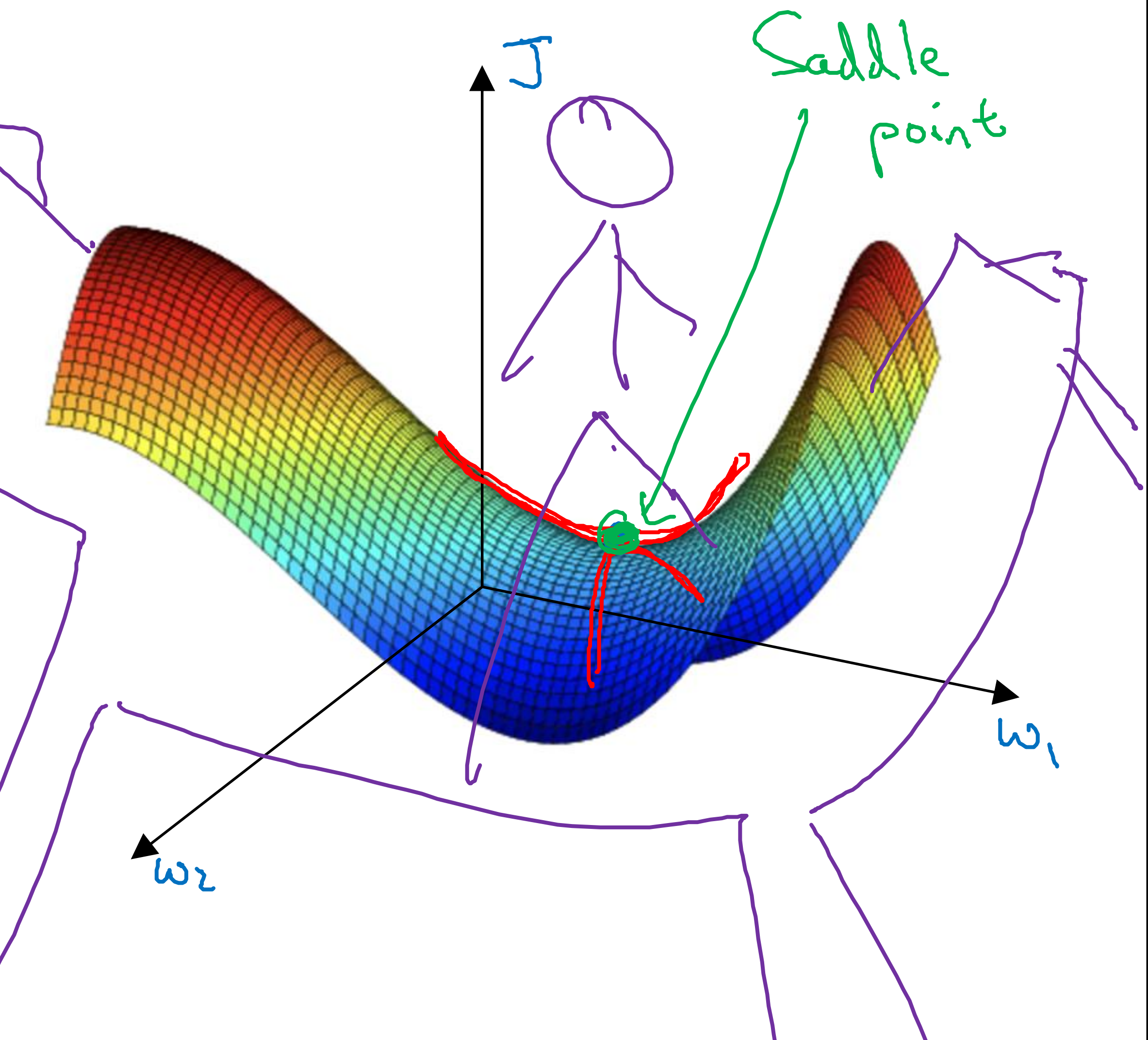
Optimization Algorithms

The problem of local optima

Local optima in neural networks

IN THIS PICTURE IT LOOKS LIKE
THERE ARE A LOT OF LOCAL OPTIMA
AND IT LOOKS EASY THAT OPTIM ALGO TO GET
STOCK IN ONE OF THEM

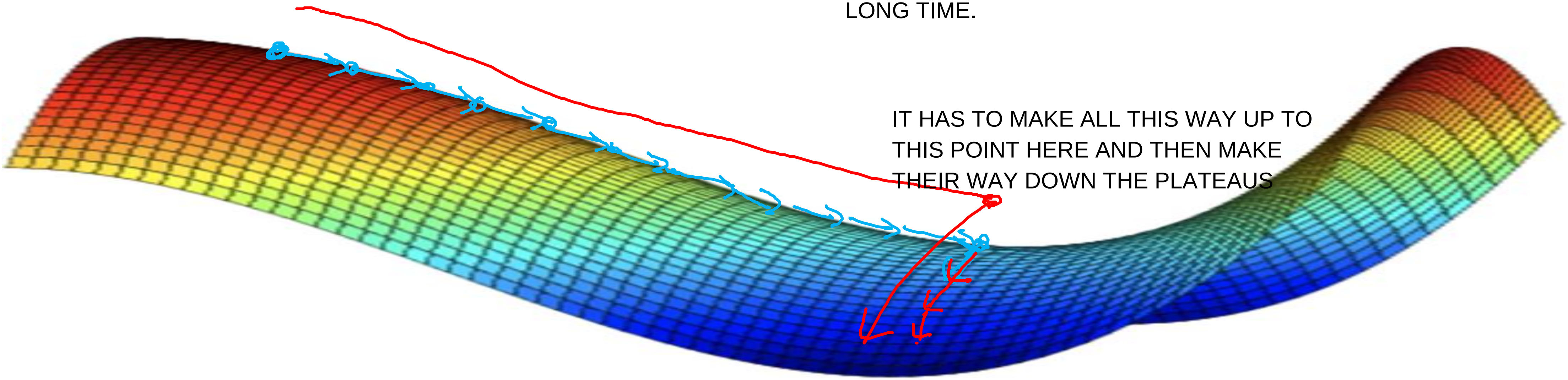




THESE VERY LOW DIMENTIONAL PLOTS USED TO GUIDE THEIR INTUITION, BUT ITS NOT ACTUALLY CORRECT
IF YOU TRAIN A NN MOST POINTS OF ZERO GRADIENTS ARE NOT LOCAL OPTIMA. INSTEAD MOST POINTS OF
ZERO GRADIENT IN A COST FUNCTION ARE SADDLE POINTS, SO THATS A POINT WITH ZERO GRADIE

Problem of plateaus

IF LOCAL OPTIMA ITS NOT A PROBLEM THEN WHATS THE PROBLEM
IT TURNS OUT THAT PLATEAUS CAN REALLY SLOW DOWN THE LEARNING
A PLATEAUS IS A REGION WHERE THE DERIVATIVES IS CLOSE TO ZERO FOR LONG TIME.



IT HAS TO MAKE ALL THIS WAY UP TO THIS POINT HERE AND THEN MAKE THEIR WAY DOWN THE PLATEAUS

- Unlikely to get stuck in a bad local optima
- Plateaus can make learning slow

- THIS IS WHERE ALGOS LIKE MOMENTUM OR RMSprop OR ADAMS CAN HELP

SINCE THEN NN IS SOLVING OPTIMISATION ALGOS IN HIGH DIMENTIONAL SPACES, ANYONE HAS GREAT INTUITION ABOUT WHAT THESE SPACES REALLY LOOK LIKE AND OUR UNDERSTANDING OF THEM IS STILL EVOLVING.