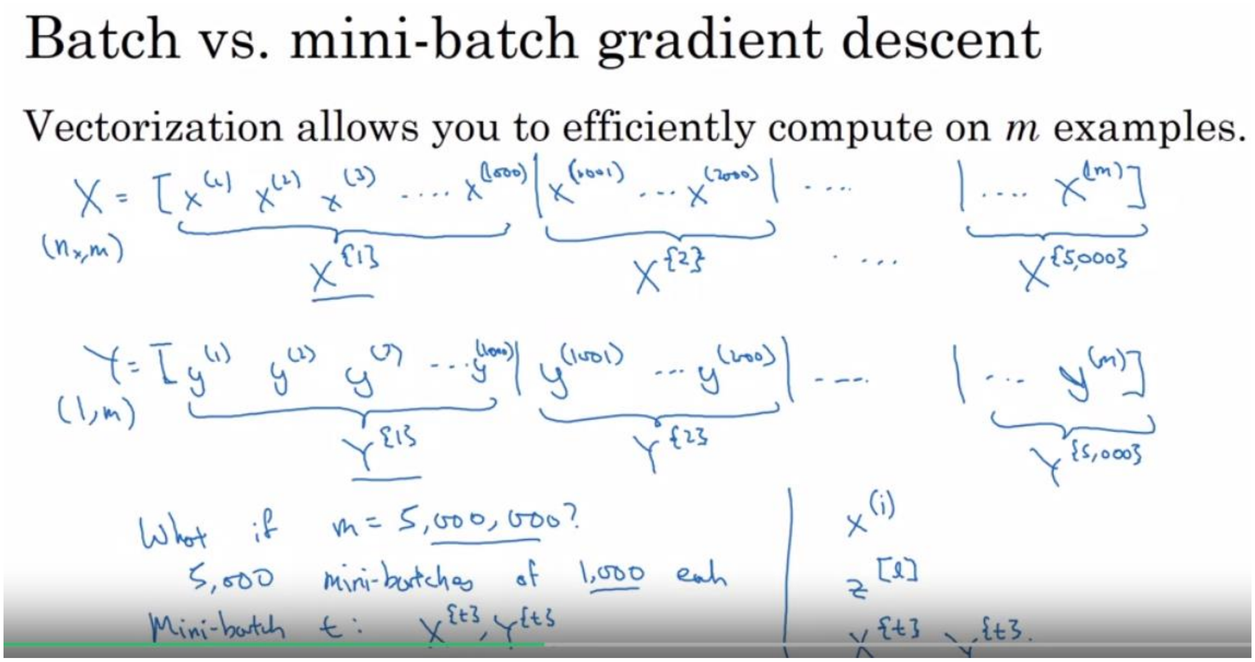
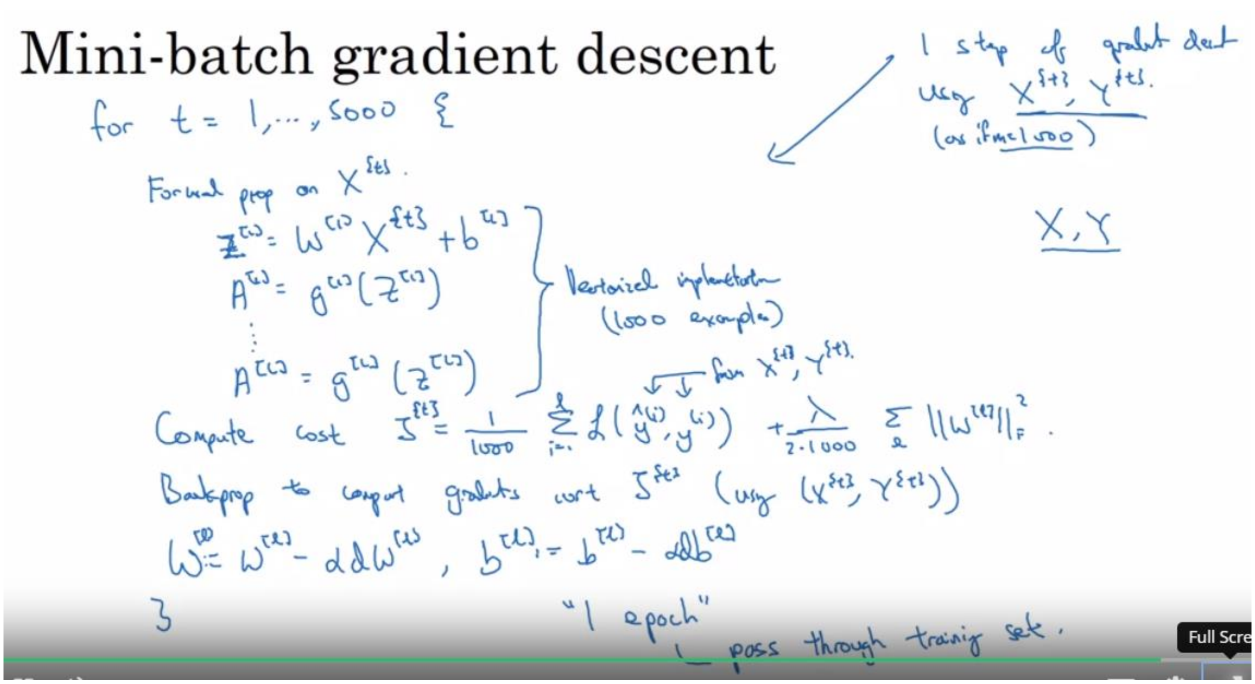
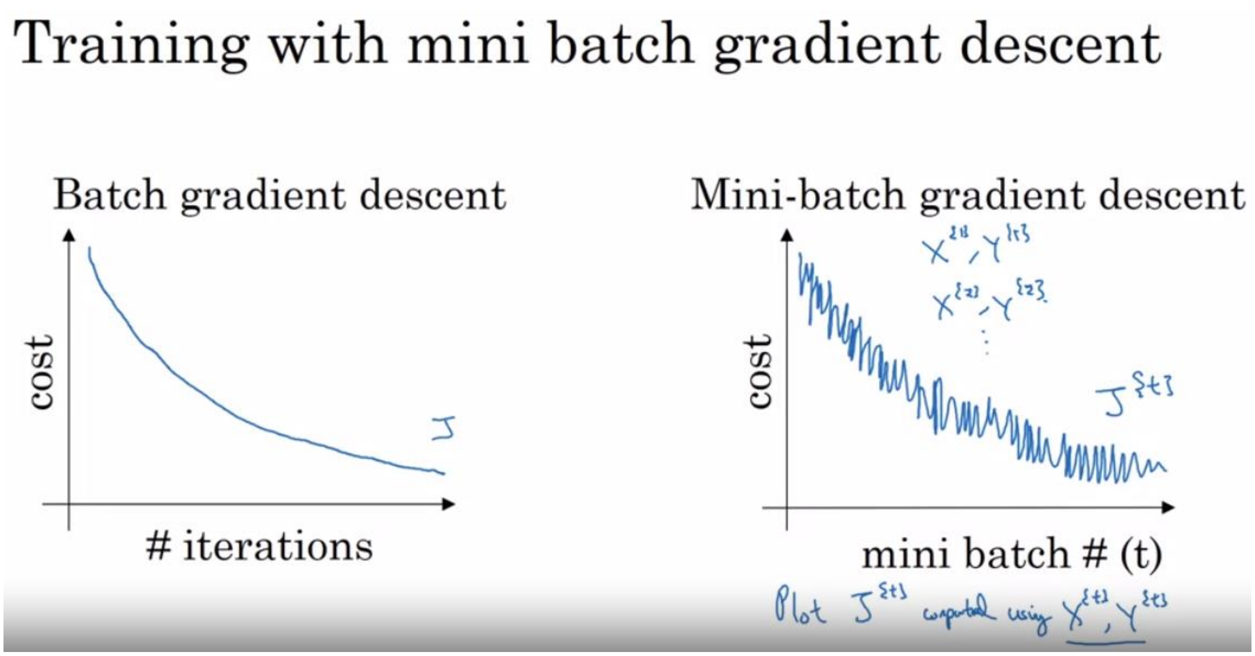
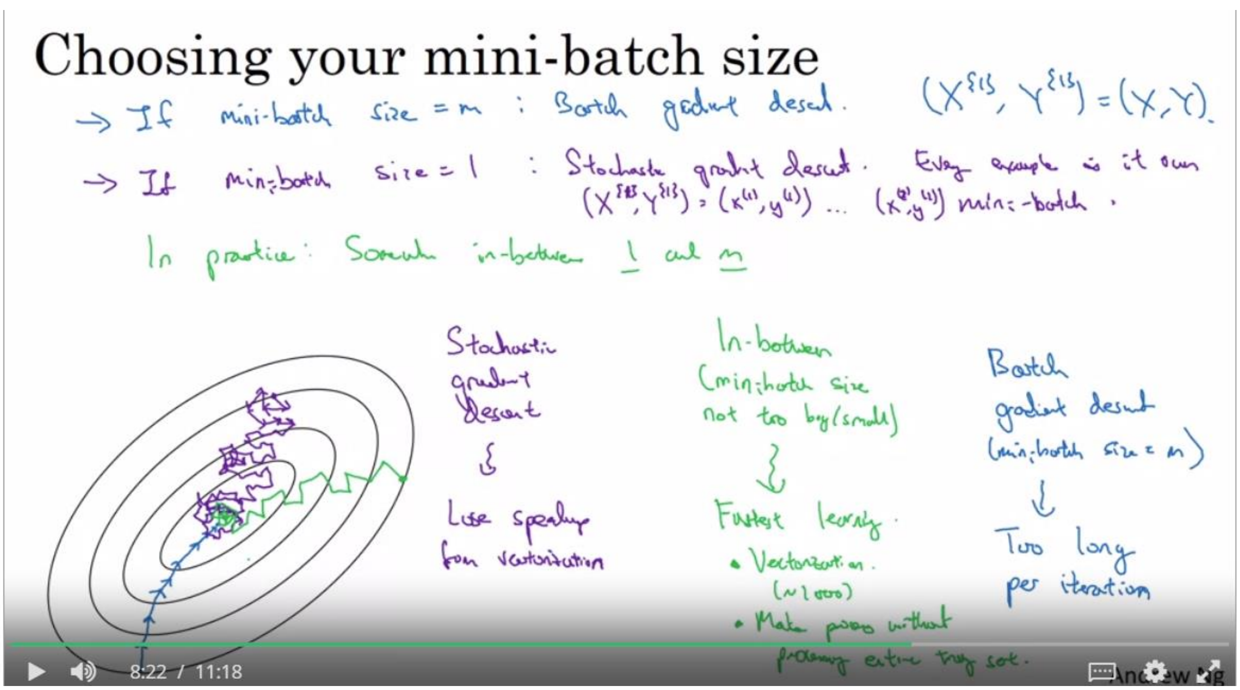
Hyperparameters Tuning

Week 2






Training with mini batch gradient descent
Batch gradient descent
cost
iterations
J
Mini-batch gradient descent
cost
mini batch # (t)
$X^{\{1\}}, Y^{\{1\}}$
$X^{\{2\}}, Y^{\{2\}}$
$J^{\{t\}}$
Plot $J^{\{t\}}$ computed using $X^{\{t\}}, Y^{\{t\}}$


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 $(X^{\{1\}}, Y^{\{1\}}) = (x^{(1)}, y^{(1)})$ ... $(x^{(2)}, y^{(2)})$ mini-batch.
In practice: Somewhere in-between 1 and m
Stochastic gradient descent
Lose speedup 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
8:22 / 11:18
Andrew Ng

Choosing your mini-batch size

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

Typical mini-batch sizes:

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

Make sure mini-batch fit in CPU/GPU memory.
$X^{\{t\}}, Y^{\{t\}}$.

Usually, we choose as size of mini-batch gradient a power of 2; it is more computationally efficient.

Exponentially weighted averages

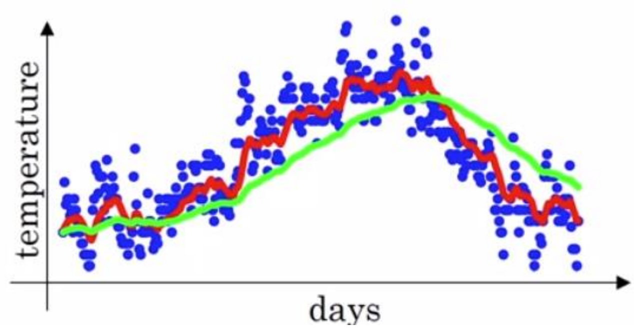
$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$

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

$\beta = 0.98$: $\approx$ 50 days

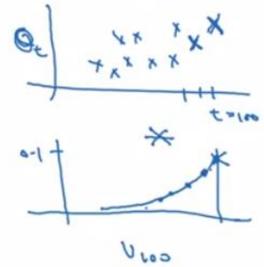
V_t as approximately average over $\approx \frac{1}{1-\beta}$ days' temperature.

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



Exponentially weighted averages

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



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

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

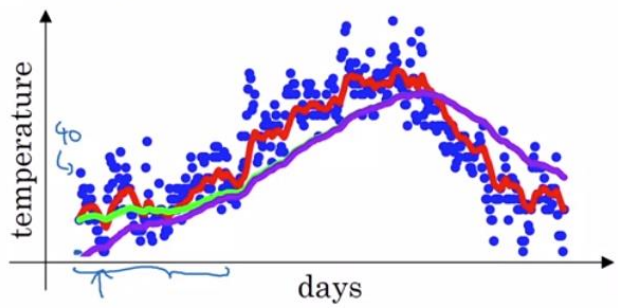
$\boxed{v_{98} = 0.9v_{97} + 0.1\theta_{98}}$

...

$V_{100} = 0.1\,\theta_{100} + 0.9\,\cancel{v_{99}}\,(0.1\,\theta_{99} + 0.9\,\cancel{v_{98}})$ $\quad 0.1\,\theta_{98} + 0.9\,v_{97}$

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

Bias correction



$\beta = 0.98$

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

$V_0 = 0$

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

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

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

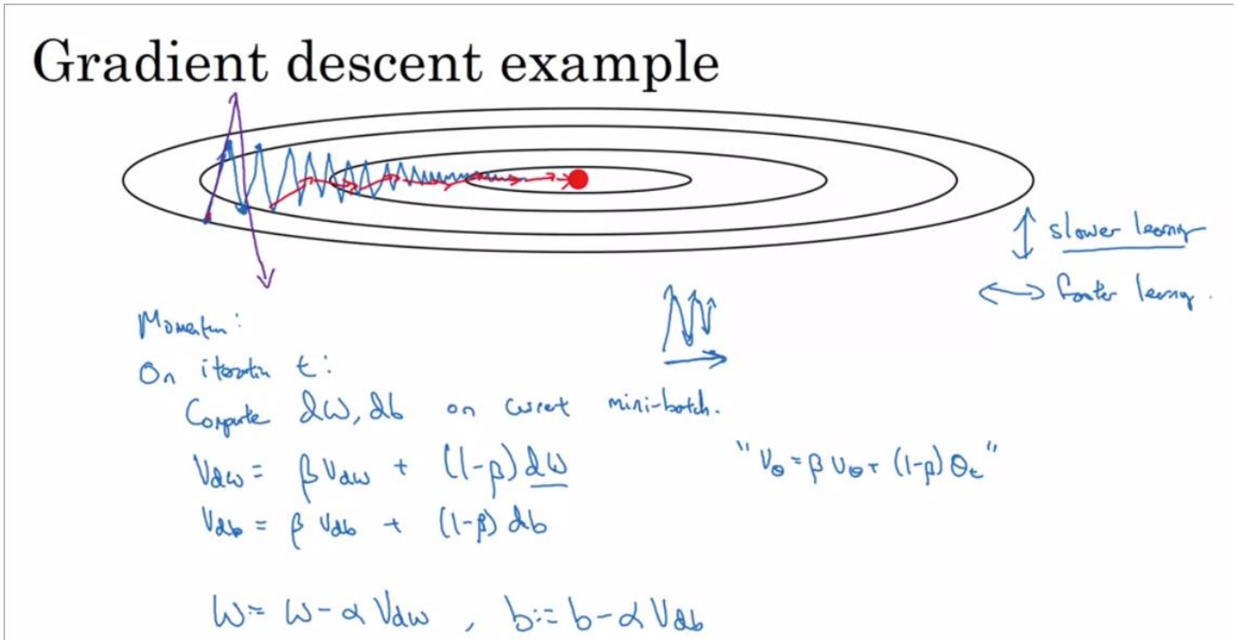
$= 0.0196\,\theta_1 + 0.02\,\theta_2$

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

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

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

Gradient descent with momentum



Implementation details

On iteration t:

Compute dW, db on the current mini-batch

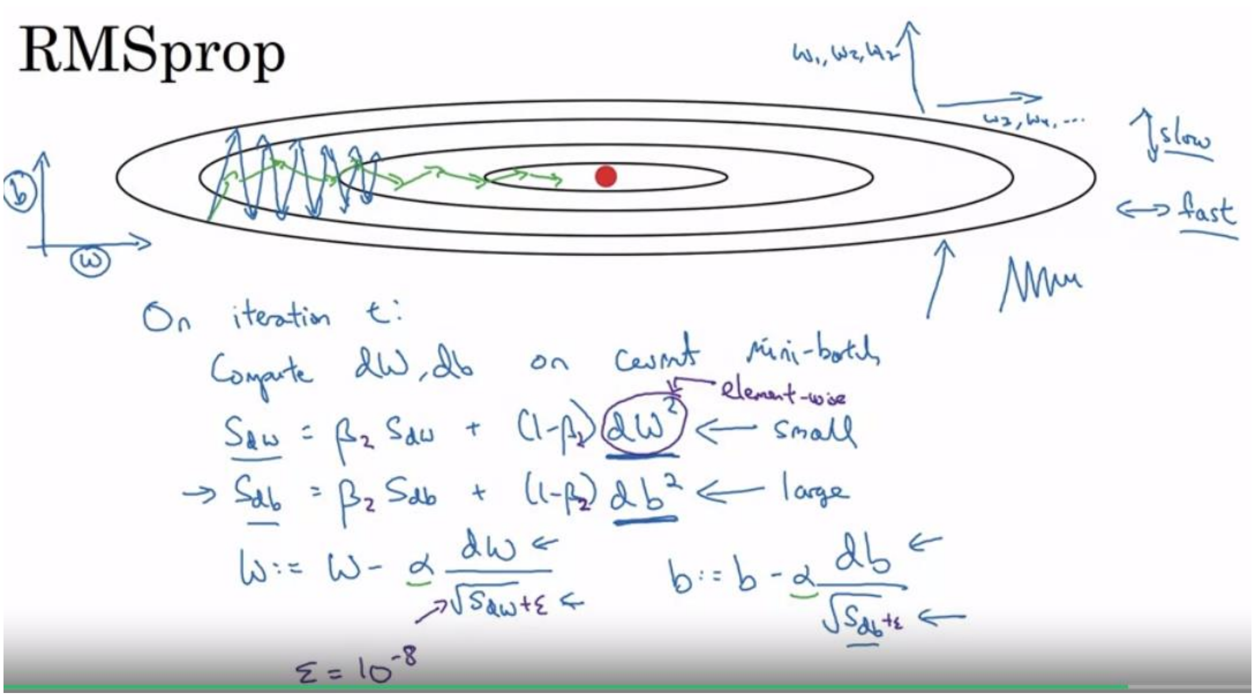
$$v_{dW} = \beta v_{dW} + (1-\beta)dW$$

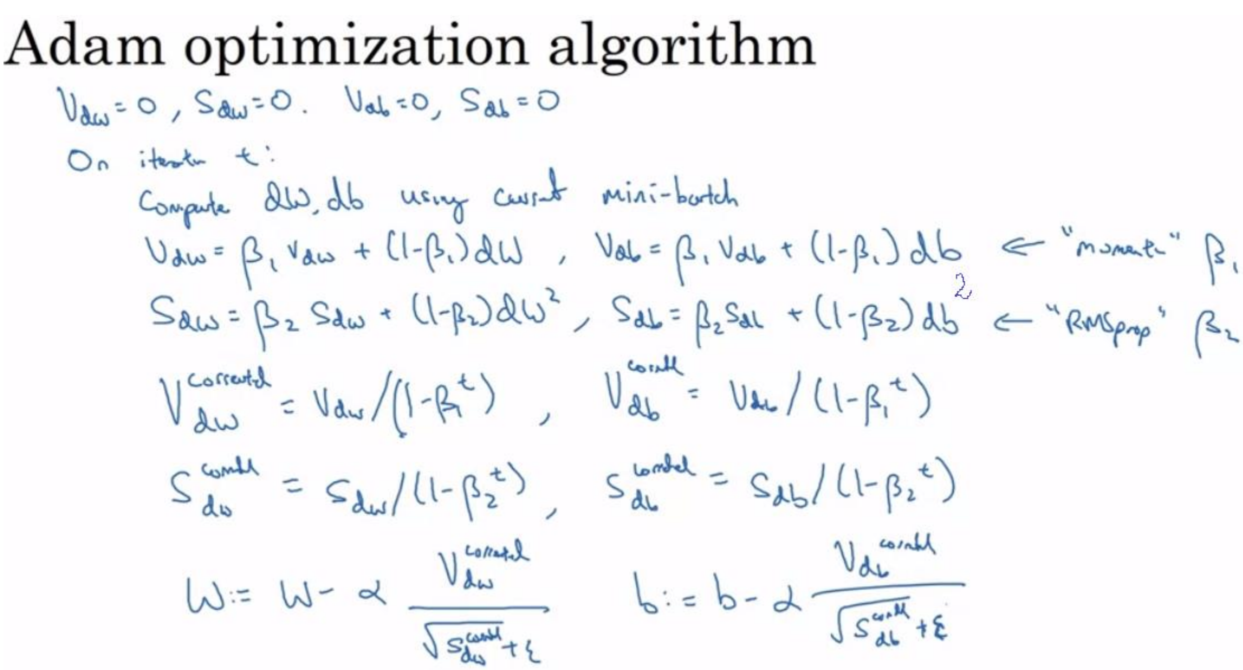
$$v_{db} = \beta v_{db} + (1-\beta)db$$

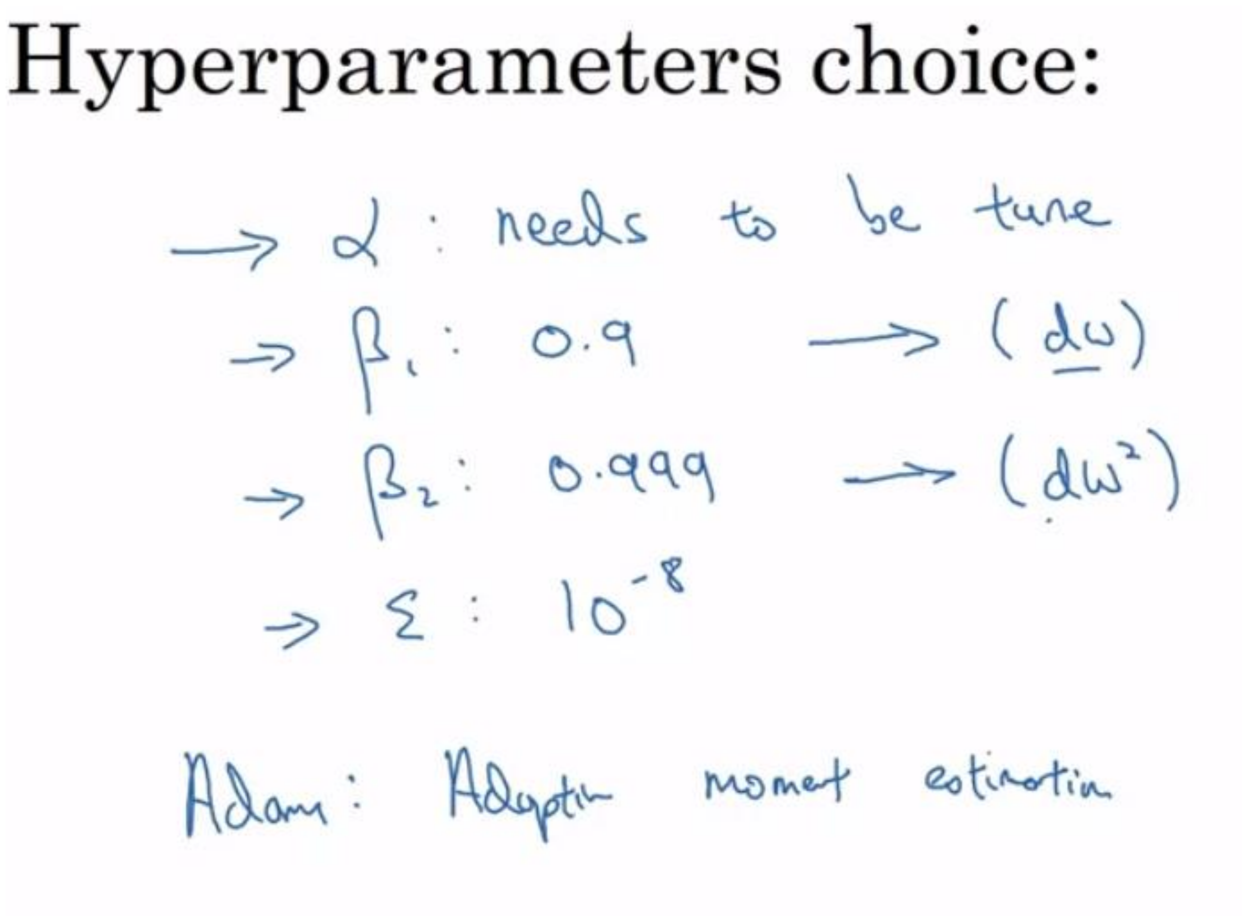
$$W = W - \alpha v_{dW}, \; b = b - \alpha v_{db}$$

Hyperparameters: α, β $\qquad$ $\beta = 0.9$

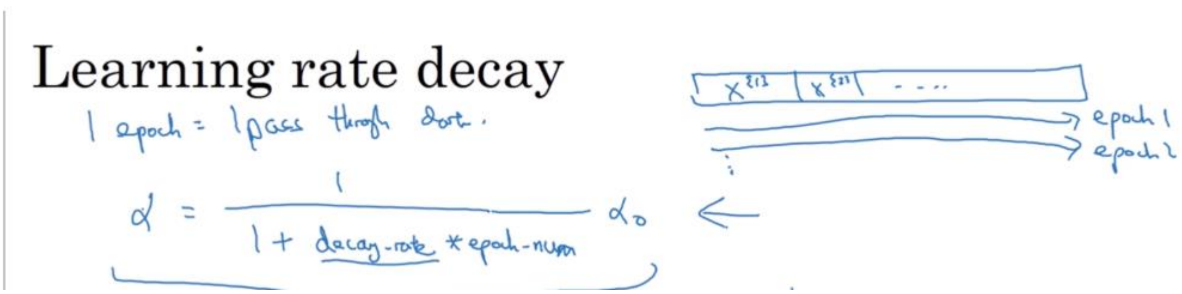
Usually, there is no need to apply bias correction when implementing momentum on gradient descent.







Usually, people use the default values for epsilon and betas, without trying to tune them.

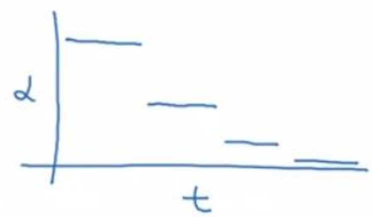


Other learning rate decay methods

formula

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

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

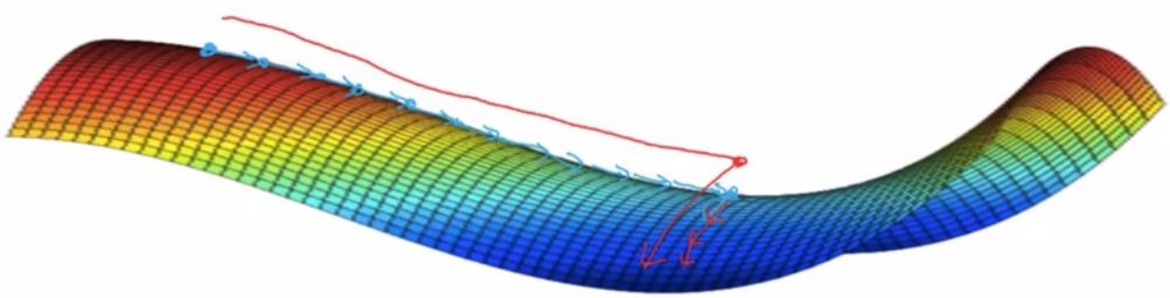


discrete staircase

Manual decay.

In mathematics, a **saddle point** or **minimax point**[1] is a point on the surface of the graph of a function where the slopes (derivatives) in orthogonal directions are all zero (a critical point), but which is not a local extremum of the function.[2] An example of a saddle point is when there is a critical point with a relative minimum along one axial direction (between peaks) and at a relative maximum along the crossing axis. However, a saddle point need not be in this form. For example, the function $f(x,y) = x^2 + y^3$ has a critical point at $(0,0)$ that is a saddle point since it is neither a relative maximum nor relative minimum, but it does not have a relative maximum or relative minimum in the y-direction.

Problem of plateaus



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

Adam on the other hand, clearly outperforms mini-batch gradient descent and Momentum. If you run the model for more epochs on this simple dataset, all three methods will lead to very good results. However, you've seen that Adam converges a lot faster.

Some advantages of Adam include:

- Relatively low memory requirements (though higher than gradient descent and gradient descent with momentum)
- Usually works well even with little tuning of hyperparameters (except α)