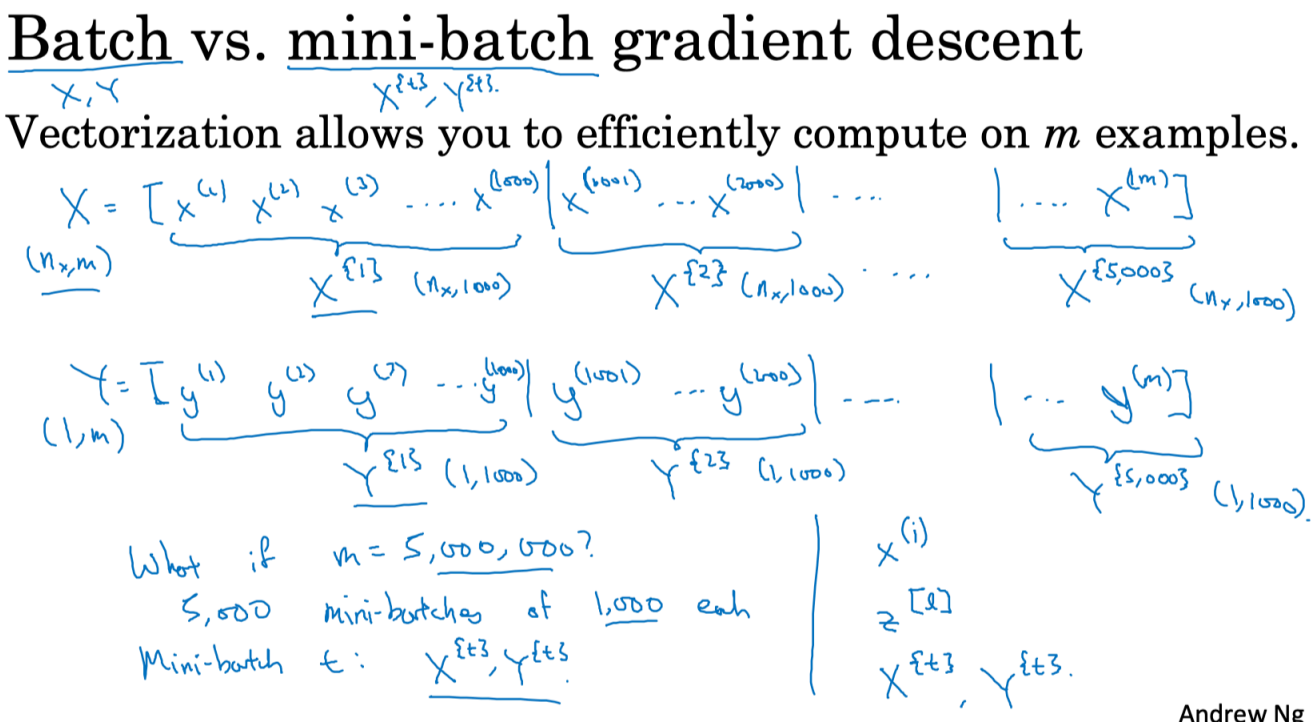
如何在大数据集下加速神经网络训练的算法：

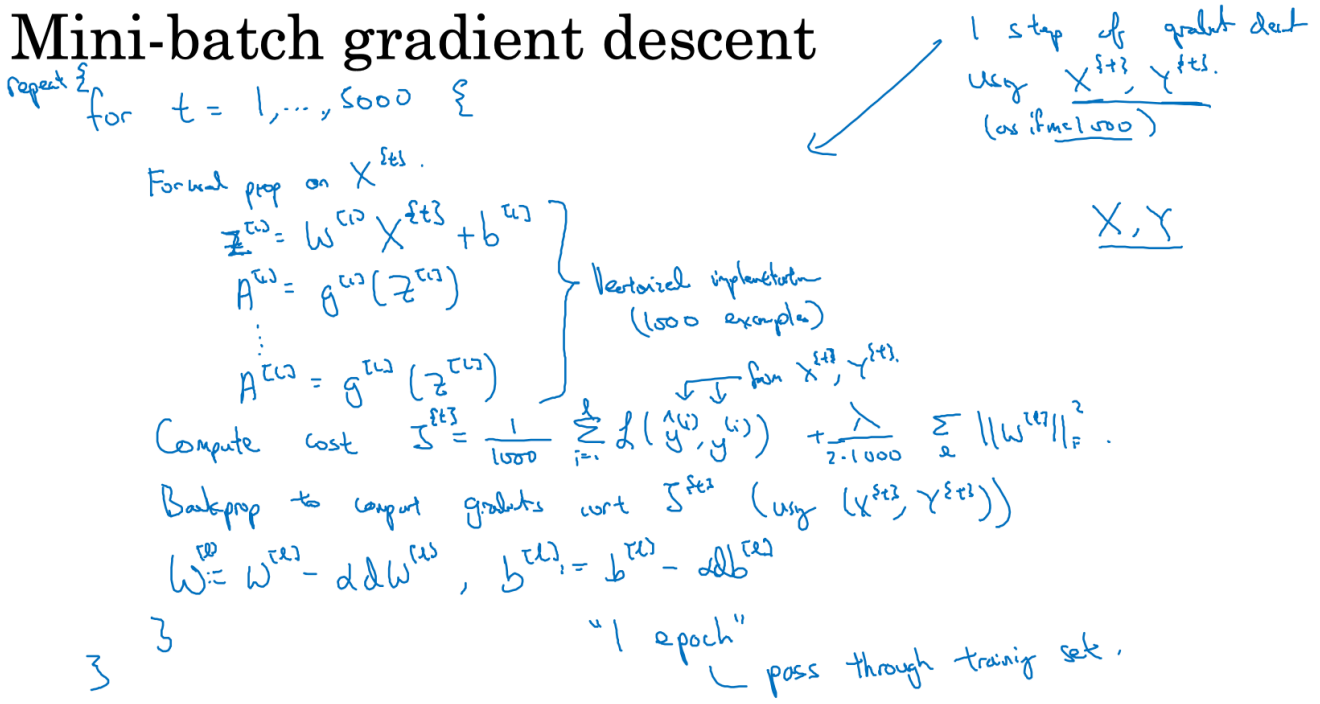
1. Speed up1: Minibatch gradient descent:

   Vectorization 可以加快训练速度但是如果当instance number很大的时候训练速度仍然会很慢，所以minibatch 来分割



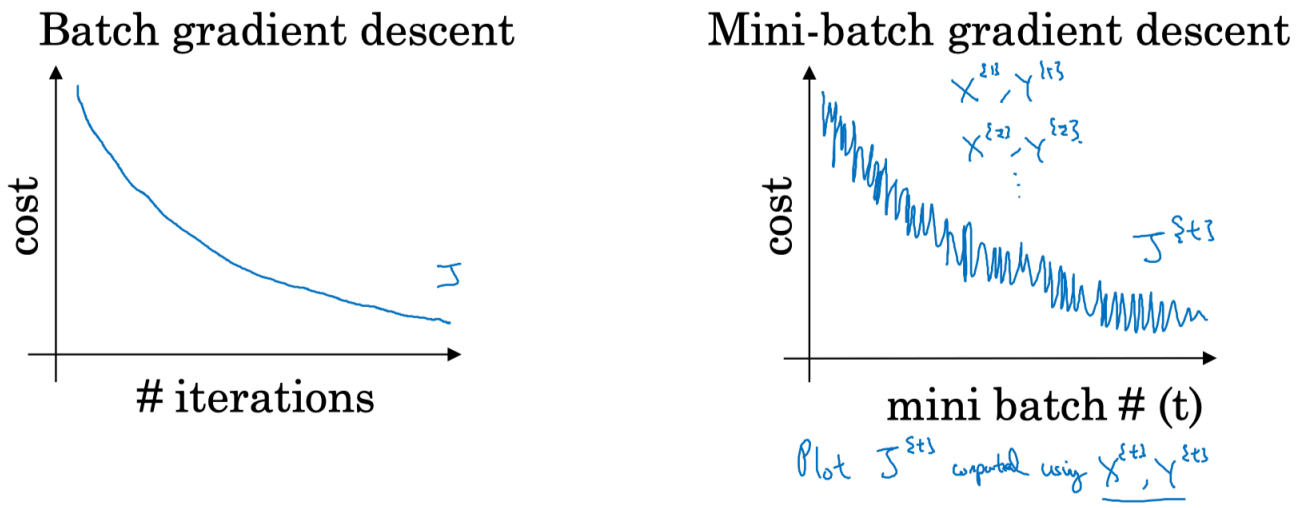
Andrew Ng

正常的batch 1 epoch（1 pass through training set）allow you to take 1 gradient descent step, 如果number of instance m=5000000,每个mini batch是1000，那么如下的mini batch gradient descent每个epoch takes 5000 gradient descent step



因为cost J的计算是针对每个mini batch的，每个mini batch训练集都不一样所以每次计算的J也有可能不同。

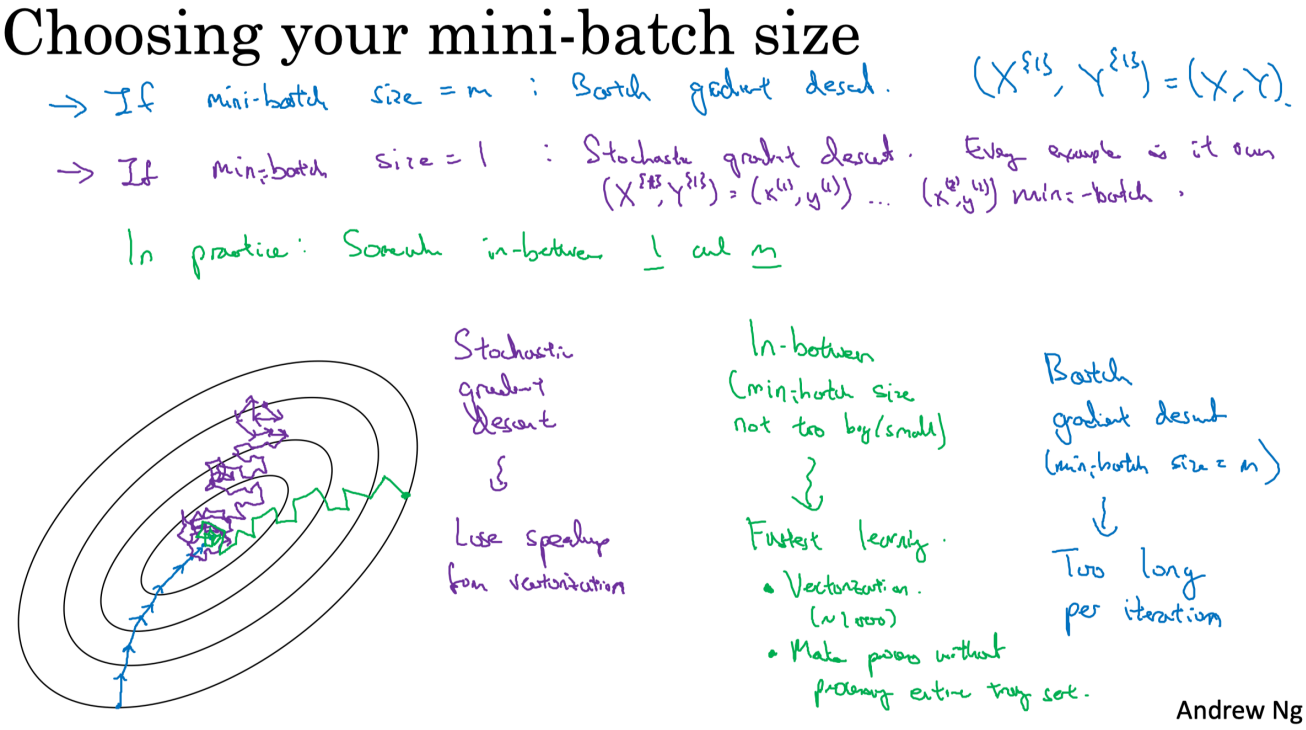
Training with mini batch gradient descent



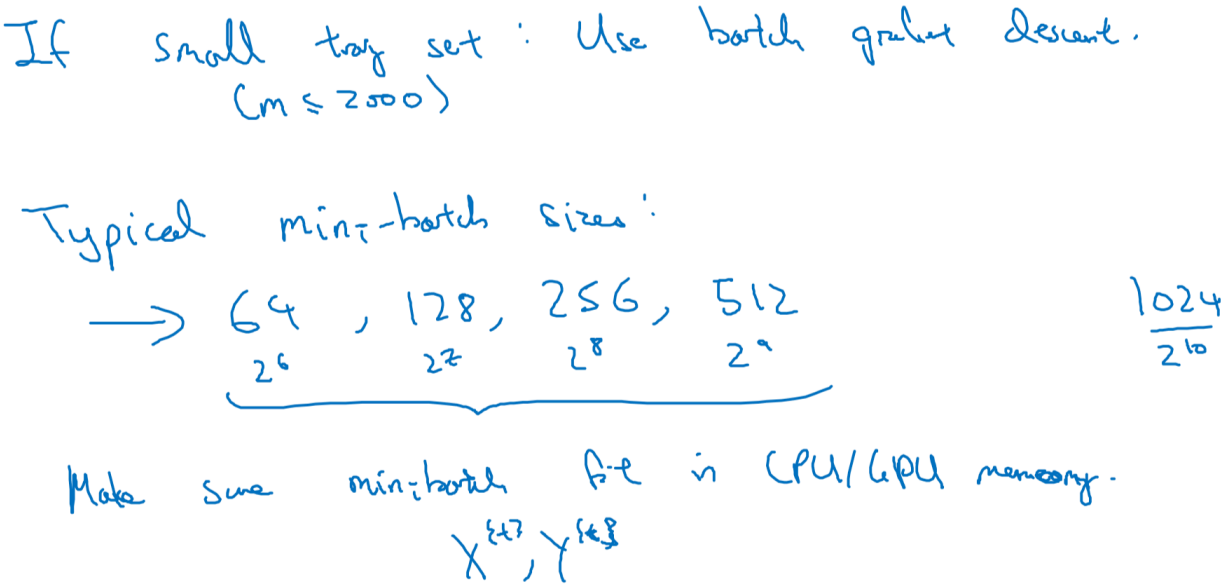
choose your mini-batch size:

蓝色和紫色是极端情况，绿色是最好的情况，紫色这种如果mini batch size太小就不能很好利用vecotrization加速，蓝色的话时间太长了

Choosing your mini-batch size



一般选择2的指数次



2. Exponentially Weighted Averages(EWA):

Exponentially Weighted Averages (EWA) or Exponentially Moving Averages is a statistical measure used to calculate a series of averages of different subsets of the full data set. It's particularly useful in data analysis and signal processing for smoothing out short-term fluctuations and highlighting longer-term trends or cycles.

The main idea behind EWA is to give more weight to recent observations while not completely discarding the older ones. This is achieved by applying a weighting factor that decreases exponentially for older observations. The formula for EWA can be expressed as:

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

where:

- V_t is the value of the EWA at time t,
- θ_t is the actual observation at time t,
- β is the weighting factor (between 0 and 1), and
- V_{t-1} is the value of the EWA at time $t - 1$.

In the context of deep learning, EWA is particularly useful for various reasons:

1. **Smoothing**: EWA helps in smoothing out the updates of parameters or gradients, which can be very noisy due to the stochastic nature of the training data. This smoothing effect can lead to more stable and reliable convergence during the training process.
2. **Momentum**: In optimization algorithms like Gradient Descent, EWA is used to implement momentum, which accelerates the convergence towards the minimum of the loss function. By accumulating a fraction (`\( 1-\beta \)`) of the current and past gradients, the updates become more aligned in the direction of the steepest descent, which can help in faster convergence.
3. **Adaptive Learning Rates**: In advanced optimization algorithms like Adam (Adaptive Moment Estimation), EWA is used to calculate adaptive learning rates for different parameters. This is done by maintaining EWA of both the gradients and the squared gradients, which help in adjusting the learning rate based on the history of gradients. This leads to more efficient training, especially in complex models and high-dimensional spaces.
4. **Robustness**: EWA provides robustness against outliers and noise in the data. By weighting the observations, the impact of any single observation is reduced, making the learning process less sensitive to noise.

Using simple averages in place of EWA in deep learning could lead to significant issues, such as overreacting to noise in the data, slower convergence, or even getting stuck in suboptimal points of the loss landscape. EWA, with its ability to smooth out fluctuations and adapt to the data's underlying trends, provides a more efficient and stable way to track and update important quantities like gradients and parameter updates in deep learning models.

在指数加权平均（Exponentially Weighted Averages）中，参数 beta 控制了加权平均中考虑历史信息的程度。具体地，加权平均可以被看作是过去数值的加权和，其中权重随时间指数级减小。权重的减少速度由 beta 值决定。

当 beta 接近 1 时，权重减少得非常慢，这意味着更早的值在平均中仍然有相对较大的权重。因此，平均值涵盖（或“记住”）的历史信息范围会很广。这个范围通常可以用 $\frac{1}{1-\beta}$ 这个公式来近似估计，它给出了加权平均大约覆盖过去多少个值。

- 当 beta = 0.9 时，$\frac{1}{1-0.9} = 10$，这表明加权平均大约覆盖过去 10 个值。
- 当 beta 增加到 0.999 时，$\frac{1}{1-0.999} = 1000$，加权平均覆盖的历史范围扩大到大约 1000 个值。
- 当 beta 进一步增加到 0.9995 时，$\frac{1}{1-0.9995} = 2000$，加权平均覆盖的历史范围进一步扩大到大约 2000 个值。

因此，即使是很小的 beta 值的变化（例如从 0.999 到 0.9995），当 beta 已经非常接近 1 时，也会导致加权平均覆盖的历史范围显著增加，这就是为什么在这种情况下，加权平均会从覆盖大约最后 1000 个样本变化到覆盖大约最后 2000 个样本。这种效应说明了在参数接近 1 时，对其进行微小调整可能会对算法的行为产生重大影响。

ChatGPT

Exponentially Weighted Averages (EWA) 的几何意义和本质可以从以下几个方面来理解：

几何意义

1. **衰减系数**: EWA 通过衰减系数 β 来调节历史数据的影响力。当 β 接近 1 时，历史数据的影响力更大，曲线更平滑；当 β 接近 0 时，近期数据的影响力更大，曲线对近期变化更敏感。这种权重分配在几何上表现为对数据点的平滑处理，可以看作是一种 "滑动" 或 "滤波" 效果，它平衡了曲线的波动和平滑度。
2. **无限级数**: 从几何角度看，EWA 实际上可以表示为当前点与所有之前点的加权平均，其中每个点的权重随着与当前点的距离增加而指数级减少。这意味着，尽管理论上所有过去的观测值都对当前的平均值有所贡献，但实际上距离当前点越远的观测值其贡献度越小。

本质

EWA 的本质是通过指数级衰减的权重来平衡最近的观测值和过去的历史信息。它的核心在于，通过适当选择 β，可以灵活调控历史数据对当前估计值的影响程度，从而使其既能快速响应近期的变化，又能保持对历史趋势的一定记忆。

为什么要提出它

EWA 被提出的原因主要在于其在数据分析、信号处理和时间序列预测等领域中的广泛应用需求，尤其是在处理有噪声的数据时，需要一种能够平滑短期波动、突出长期趋势的方法。EWA 提供了一种有效的技术手段来滤除随机波动，使得数据的主要趋势更加明显，从而更好地进行分析和预测。

起源

EWA 的起源可以追溯到统计学和信号处理领域中的移动平均技术。最初，简单移动平均（SMA）被用来平滑时间序列数据，但其对所有历史观测值给予了相同的权重，忽略了最近观测值的重要性。随后，为了解决这一问题，并提高平滑技术的灵活性和响应速度，人们提出了 EWA 方法，它通过指数级的权重分配来强调最近的数据点，同时还保留了对过去数据的一定记忆能力。这种方法很快就因其有效性和灵活性而在多个领域得到广泛应用。

3. Bias correction:用来correct初始阶段的Exponentially Weighted Averages(可有可无)

Bias correction is a technique used to adjust exponentially weighted moving averages (EWMA) for the initial bias introduced when these averages are initialized. This bias is particularly significant at the beginning of the data series, where the lack of prior data points results in a skewed average. Let's break down the explanation provided into simpler terms and illustrate how bias correction works.

The Problem with Initialization

When you start calculating an exponentially weighted moving average, you typically initialize the first value of the moving average (V_0) to 0. This is a practical choice but introduces a bias in the initial calculations because the true average is not 0. For example, if the first temperature reading (Θ_1) is 40 degrees Fahrenheit, and you use a weighting factor (β) of 0.98, the first calculated average (V_1) would be $0.02 \times 40 = 0.8$. This value is much lower than the actual temperature, indicating a bias.

How Bias Correction Works

To correct this bias, especially at the start, you can adjust the calculated moving average by dividing it by a factor that accounts for the initial lack of data. This factor is $1 - \beta^t$, where t is the current time step (or the number of data points you have processed).

For instance, on day 2 ($t = 2$), the bias correction factor would be $1 - 0.98^2$. This corrects the moving average by scaling it up, making it closer to the actual temperatures. The corrected average becomes:

$$\text{Corrected } V_2 = \frac{V_2}{1-0.98^2}$$

This makes V_2 a better estimate of the actual average temperature for the first two days, as it compensates for the initial underestimation caused by starting with $V_0 = 0$.

Impact Over Time

As t increases (i.e., as more data points are included), β^t approaches 0, making the bias correction factor ($1 - \beta^t$) approach 1. This means that the bias correction becomes less significant over time, and eventually, the corrected and uncorrected values converge. This is why, in the long run, the bias correction doesn't alter the moving average significantly, and the two curves (with and without bias correction) overlap.

Bias correction可以前几步用后面就可以用标准的EWA了，也可以按如下方式整合

在实际应用中，标准的指数加权平均（EWA）公式和带有偏差修正的EWA公式是根据情境选择使用的。一开始，当数据量较少且你想要减少初始化带来的偏差影响时，可以使用带有偏差修正的公式。随着时间的推移，当你拥有足够多的数据点时，标准EWA公式的结果将趋于稳定，此时偏差修正的影响会变得非常小，几乎可以忽略不计。

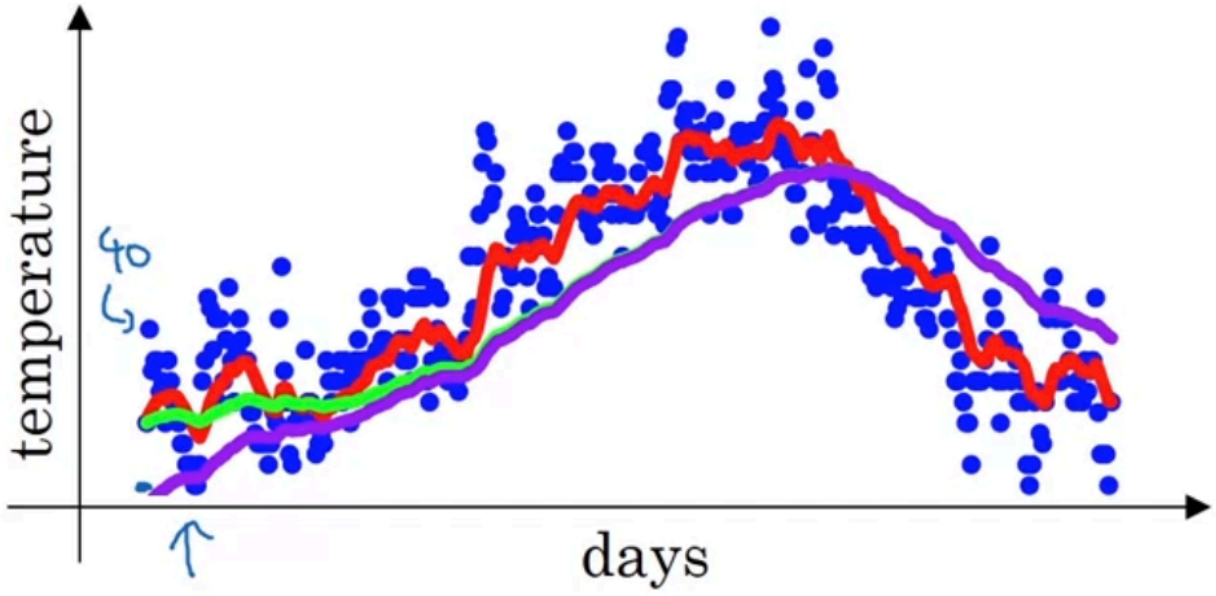
如果你想将这两个概念整合到一个公式中，可以考虑在每个时间步长t都应用偏差修正，这样就不需要选择何时使用标准公式，何时使用偏差修正公式。整合后的公式会在计算每个时间点的加权平均值后立即应用偏差修正：

1. 首先，按照标准的EWA公式计算加权平均值:
   $V_t = \beta V_{t-1} + (1-\beta)\Theta_t$
2. 然后，应用偏差修正:
   $\text{Corrected } V_t = \frac{V_t}{1-\beta^t}$

整合后的公式表示为：
$$\text{Integrated EWA with Bias Correction} = \frac{\beta V_{t-1} + (1-\beta)\Theta_t}{1-\beta^t}$$

这个整合后的公式在每一步都考虑了偏差修正，确保了在序列的早期阶段得到更准确的估计。随着t的增加，β^t趋于0，偏差修正的分母趋于1，整合公式逐渐过渡到标准的EWA公式。这样，你就不需要担心何时转换公式，因为整合公式自动处理了这一过程。

刚开始是紫色的这条线，估值偏小所以要bias correction，然后变成绿色的线：



ChatGPT

偏差修正（bias correction）之所以要除以 $1-\beta^t$ 而不是 β^t，是因为我们需要调整初始时期因 V_0 初始化为 0 而引入的偏差。这种偏差导致初始的平均值被低估了，因为在开始的几个时间步长内，指数加权平均（EWA）的计算基本上只是对最初几个 Θ_t 的简单平均，而没有足够的历史数据来平衡。

解释偏差修正的原理：

- **不使用偏差修正时的问题：**在 EWA 的初始阶段，由于 V_0 通常初始化为 0，所以初始的 V_t 值会被显著低估。例如，如果 β 很高（接近 1），那么 $V_1 = \beta V_0 + (1-\beta)\Theta_1$ 会因为 $V_0 = 0$ 而接近于 $(1-\beta)\Theta_1$，显著低于实际的 Θ_1 值。
- **偏差修正的目的：**偏差修正的目的是通过一个调整因子来提升这些初期的 V_t 值，从而更接近真实的平均值。在初始时期，因为 β^t 接近 1（尤其是当 β 接近 1 且 t 较小时），$1-\beta^t$ 会是一个较小的数。通过将 V_t 除以这个较小的数，我们可以放大 V_t，以补偿因初始化为 0 而引入的低估偏差。
- **为什么是 $1-\beta^t$：**选择 $1-\beta^t$ 作为修正因子，是因为它随着时间的推移而增大，从而在初始阶段提供较大的调整（因为此时 β^t 接近 1，所以 $1-\beta^t$ 接近 0），而随着时间的推移，调整效果逐渐减小（因为 β^t 趋向于 0，$1-\beta^t$ 趋向于 1）。这意味着随着时间的推移，偏差修正的影响会逐渐减少，当我们有足够多的数据点时，修正因子几乎不会对 V_t 产生影响，保留了 EWA 的长期行为。

总之，偏差修正通过除以 $1-\beta^t$ 来增加初期的加权平均值，以补偿初始偏差，并随着时间的推移逐渐减少这种调整，从而在维持长期 EWA 行为的同时改善短期估计的准确性。

4. Speed up2: Gradient Descent with Momentum:

2. Momentum Concept:

The idea of momentum comes from physics, where it refers to the tendency of a moving object to continue moving in the same direction. In the context of gradient descent, momentum helps to accelerate the optimization process in the direction of the steepest descent and dampens the oscillations. It does this by keeping track of the 'velocity' (a weighted sum of previous gradients) and using this velocity to update the parameters, rather than using just the current gradient.

3. Gradient Descent with Momentum:

In this variant, you introduce a velocity term v and a momentum coefficient β, typically set between 0.9 and 0.99. The update rules become:

$$v_{t+1} = \beta v_t + (1 - \beta)\nabla_\theta J(\theta)$$
$$\theta_{t+1} = \theta_t - \alpha v_{t+1}$$

Here, v_{t+1} is the new velocity, a combination of the fraction β of the old velocity and the fraction $1 - \beta$ of the current gradient. This way, the updates not only consider the current gradient but also the direction and magnitude of past gradients, making the updates smoother and more stable.

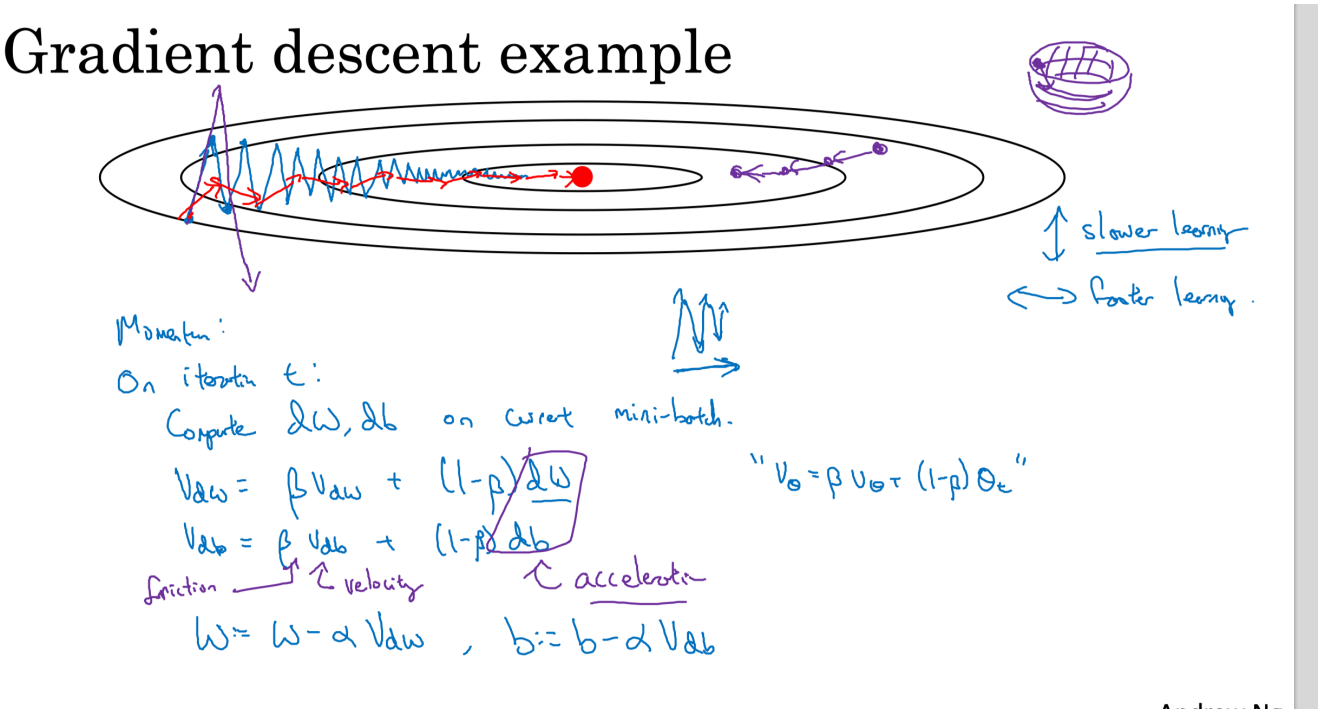
4. Benefits of Momentum:

- **Speeds up convergence:** By accumulating a velocity in directions of persistent reduction in the loss function, it can lead to faster convergence.
- **Reduces oscillation:** By averaging out the updates, it dampens the oscillations in directions that don't consistently contribute to reducing the loss.
- **Helps escape local minima:** The momentum term can help the optimization to overcome small local minima and continue towards finding a better (global) minimum.

5. Parameter Tuning:

Choosing the right momentum coefficient (β) and learning rate (α) is crucial. A higher β means giving more weight to past gradients, which can make the algorithm more resilient to noise but also potentially miss narrow, sharp minima.

In summary, Gradient Descent with Momentum combines the current gradient and past gradients to make more informed updates to the parameters, leading to faster and more stable convergence in optimization problems.



Gradient Descent with Momentum 相较于传统的 Gradient Descent 能够更好地避免震荡，并且通常能够更快地收敛，这主要得益于它借鉴了物理中的“动量”（momentum）概念。下面分两部分来解释这一点：

1. 避免震荡的原因

在传统的 Gradient Descent 中，每一步的参数更新都是基于当前的梯度。如果梯度在不同方向上的变化非常不均匀，比如在一个方向上很大而在另一个方向上很小，那么就会导致参数更新时在某些方向上发生较大的摆动，即出现震荡现象。这通常发生在损失函数的形状是延伸的碗形，其中一个维度的斜率比其他维度要陡得多。

当使用 Momentum 方法时，参数更新不仅取决于当前的梯度，还取决于之前梯度的累积。通过结合之前所有步骤的梯度信息（按照其重要性递减地加权），它生成了一个更加平滑的梯度方向。这种平滑效果减少了在任何特定方向上的极端变动，因此在高曲率方向上的摆动会被减弱，使得参数更新路径更加平滑。

2. 为什么叫做“Momentum”

这个术语来自于物理学中的动量概念。在物理学中，动量是质量和速度的乘积，代表了物体运动的“惯性”。一个拥有高动量的物体在停止之前会保持移动状态，即使没有新的力作用在物体上。在优化问题中，"momentum" 帮助算法保持在某一方向上的移动，即使在这个方向上的梯度（力）已经减弱。

当使用 Momentum 方法时，算法保持了对过去梯度方向的“记忆”，这就像物体的惯性那样，使得算法不会在每次梯度变化时都完全改变方向，从而避免在梯度不稳定的情况下频繁变动方向造成震荡。

因此，"Momentum" 这个名字来源于其在参数更新中加入了过去梯度的影响，类似于物理中动量的概念，使得算法在遇到小的或者不一致的梯度时，能够维持在一个稳定的方向上移动，从而更平滑地接近最小值。

5. Speed up3: RMSprop(root mean square prop): it adjusts the learning rate for each parameter, making it smaller for parameters with large gradients and larger for parameters with small gradients.

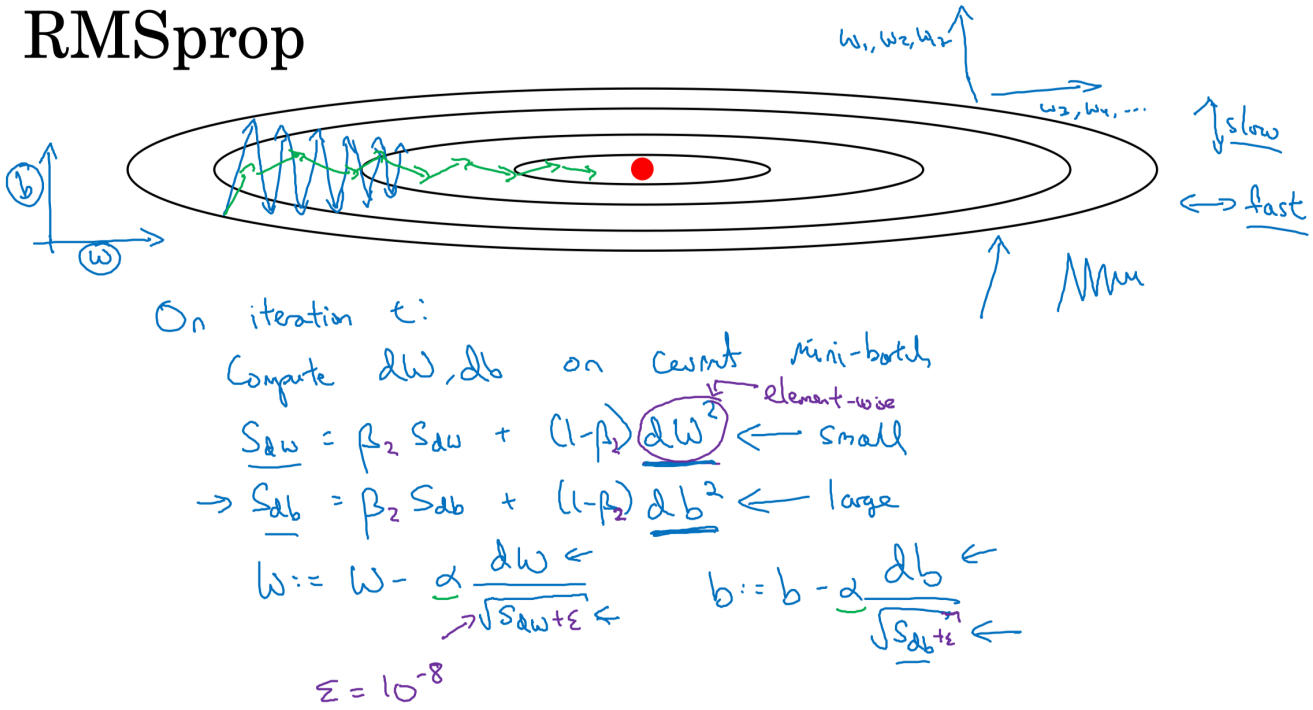
   RMSprop, which stands for Root Mean Square Propagation, is an optimization algorithm designed for training neural networks. It was introduced by Geoffrey Hinton in his Coursera class on Neural Networks. RMSprop is an adaptive learning rate method, which means it adjusts the learning rate for each parameter, making it smaller for parameters with large gradients and larger for parameters with small gradients. This helps in addressing the vanishing or exploding gradients problem, making it especially useful for training deep neural networks.

   The key idea behind RMSprop is to maintain a moving average of the square of gradients and then divide the gradient by the square root of this average to normalize the gradient steps. This normalization balances the step size, reducing the oscillations in vertical directions and aiding smoother convergence in horizontal directions, which is particularly beneficial in scenarios where the surface curves much more steeply in one dimension than in another.

   Here's how RMSprop updates the weights:

   1. Calculate the gradient of the loss function with respect to the parameters, denoted as g_t, at each time step t.
   2. Update the squared gradient moving average $E[g^2]_t$ with the equation $E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)g_t^2$, where β is a decay factor close to 1 (e.g., 0.9) that controls the moving average's sensitivity to recent gradients versus old gradients.
   3. The parameter update $\Delta\theta_t$ at time t is then computed as $\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$, where η is the learning rate, and ϵ is a small constant (e.g., $1e-8$) added to improve numerical stability.
   4. Update the parameters with $\theta_{t+1} = \theta_t + \Delta\theta_t$.

   RMSprop effectively reduces the learning rate for parameters associated with frequently occurring features, making it adaptive and robust for various problems. It's particularly useful in online and non-stationary settings. Despite its effectiveness, RMSprop has been largely superseded by Adam (Adaptive Moment Estimation), which can be seen as a combination of RMSprop and momentum, adding bias-correction and momentum to the adaptive learning rate for even better performance in practice.



6. Speed Up4: Adam: Adaptive momentum estimation（结合了M omentum和RMSprop）

Adam optimization algorithm

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

On iteration t:

Compute dw, db using current mini-batch

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

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

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

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

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

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

Hyperparameters choice:

→ α : needs to be tune

→ β_1 : 0.9 → (dw)

→ β_2 : 0.999 → (dw^2)

→ ε : 10^{-8}

Adam: Adaptive moment estimation

7. Learning Rate decay:

1. **Motivation for Learning Rate Decay**: The video begins by explaining the need for learning rate decay, particularly in mini-batch gradient descent where steps can be noisy due to the variance in mini-batches. A fixed learning rate (Alpha) might lead to the algorithm wandering around the minimum without converging. Learning rate decay aims to start with a larger Alpha for faster initial learning and then reduce it over time to take smaller steps, allowing the algorithm to oscillate in a tighter region around the minimum and improve convergence.
2. **Implementing Learning Rate Decay**: The concept of an epoch, which is one pass through the entire training set, is introduced as a basis for implementing learning rate decay. The decay is implemented such that the learning rate Alpha is adjusted according to the number of epochs completed. The suggested formula is `Alpha = 1 / (1 + decay rate * epoch num) * Alpha_0`, where `Alpha_0` is the initial learning rate, and the decay rate is a new hyperparameter that needs tuning.
3. **Example of Learning Rate Decay**: An example is provided with an initial learning rate `Alpha_0` of 0.2 and a decay rate of 1. The learning rate is shown to decrease from 0.1 in the first epoch to 0.067 in the second, demonstrating how the learning rate decreases as a function of the epoch number.
4. **Other Decay Techniques**: Beyond the basic decay formula, the video mentions other techniques like exponential decay (`Alpha = decay factor^epoch num * Alpha_0`), which decays the learning rate more rapidly, and decay based on the square root of the epoch number or mini-batch number. Another approach is discrete decay, where the learning rate is halved at specified intervals.
5. **Manual Decay**: The video also touches on manual decay, where the practitioner manually adjusts the learning rate based on the training progress. This approach is noted to be less systematic and more suited to scenarios where only a few models are being trained over a long period.

example:

- Epoch 1: `Alpha = 1 / (1 + 0.05 * 1) * 0.1 = 0.1 / 1.05 ≈ 0.0952`
- Epoch 2: `Alpha = 1 / (1 + 0.05 * 2) * 0.1 = 0.1 / 1.1 ≈ 0.0909`
- Epoch 3: `Alpha = 1 / (1 + 0.05 * 3) * 0.1 = 0.1 / 1.15 ≈ 0.0870`

8. The challenge for optimization:

Saddle Points vs. Local Optima: In high-dimensional spaces, true local optima are exceedingly rare. Instead, saddle points, where the gradient is zero but the function's curvature varies in different dimensions, are much more common.
Plateaus as a Major Challenge: Plateaus, or extensive flat areas in the optimization landscape where the gradient is close to zero, represent a significant challenge by slowing down the learning process.
Optimization Techniques: Techniques like momentum, RMSProp, and Adam are highlighted as useful for accelerating learning, especially in navigating plateaus more efficiently.

鞍点（Saddle Point）在数学和机器学习领域中是一个重要的概念，特别是在优化问题和梯度下降算法的背景下。在多维空间中，鞍点是函数的一个特殊点，在这个点上，函数在某些方向上是局部最小值，而在其他方向上是局部最大值。这意味着，从鞍点出发，函数值沿着某些方向增加，而沿着其他方向减少。

鞍点的特性

- **局部性质**：在鞍点，函数在某些维度上的导数（或梯度的某些分量）为零，这与局部最大值和最小值的性质相似。但不同的是，鞍点既不是局部最大值也不是局部最小值。
- **方向性**：在鞍点处，函数沿某些方向上升，沿其他方向下降。这意味着如果你在鞍点处开始梯度下降，算法的行为将取决于初始的梯度方向，这可能导致优化过程陷入停滞。

在神经网络中的意义

在深度学习和神经网络的训练过程中，鞍点比局部最小值更为常见。神经网络的损失函数通常是高维空间中的复杂非凸函数，这样的空间中鞍点的存在可能会使得基于梯度的优化算法（如梯度下降或其变体）难以找到全局最小值。

处理鞍点的策略

- **使用动量（Momentum）**：动量方法有助于梯度下降算法"冲出"鞍点，因为它在更新参数时考虑了之前梯度的历史，从而增加了穿过鞍点的惯性。
- **自适应学习率算法**：如Adam、RMSprop等，可以在不同的参数维度上自适应调整学习率，有助于更有效地越过鞍点。

鞍点是优化理论中的一个核心概念，对于理解和改进深度学习模型的训练过程至关重要。

因为如果有20000local optima的话，那要么是local min或者max，概率为1/2，如果都是min的话概率如下基本不可能所以就会导致不少max，这样就会形成鞍点：


ω_2
20000
2^{-20000}