Optimizers :

Convergence

- **Gradient Descent**: Optimizers, such as gradient descent and its variants, help in adjusting the weights of the neural network to minimize the loss function. They guide the model towards convergence to an optimal or near-optimal solution.
- **Learning Rate Adjustment**: Optimizers manage the learning rate to ensure that the model converges efficiently without overshooting the minimum loss.

2. Speed of Training

- **Efficiency**: Good optimizers can significantly speed up the training process by finding the optimal path to the minimum loss.
- **Stochastic Approaches**: Techniques like Stochastic Gradient Descent (SGD) and Mini-Batch Gradient Descent break down the training process into manageable parts, improving efficiency and speed.

3. Handling Complex Loss Landscapes

- **Local Minima and Saddle Points**: Advanced optimizers can help the model navigate complex loss landscapes with multiple local minima and saddle points, avoiding poor convergence.
- **Momentum and Adaptive Learning**: Optimizers such as Adam, RMSprop, and Adagrad incorporate momentum and adaptive learning rates to effectively traverse the loss surface.

4. Generalization

- **Regularization Techniques**: Some optimizers include mechanisms to improve generalization, reducing the risk of overfitting. They help the model perform well on unseen data.
- **Noise Management**: Stochastic optimizers introduce noise into the training process, which can help the model generalize better by avoiding overfitting

to the training data.

5. Scalability

- **Large Datasets**: Optimizers enable the training of models on large datasets by efficiently managing computations and memory usage.
- **Distributed Training**: They support distributed training across multiple GPUs or machines, ensuring scalability.

6. Adaptive Mechanisms

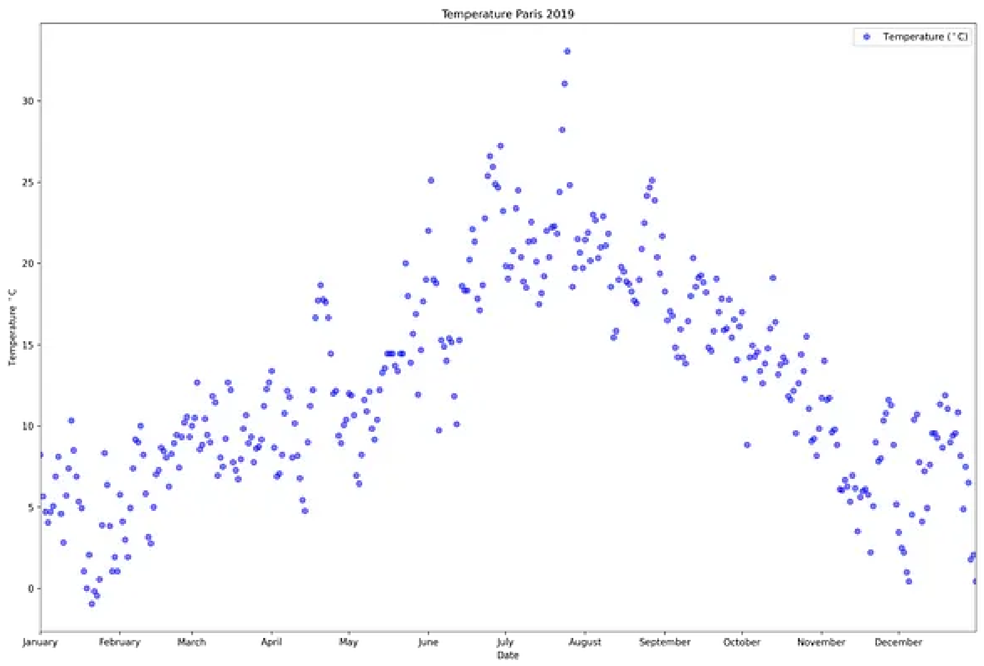
- **Dynamic Adjustments**: Optimizers like Adam, AdaGrad, and RMSprop dynamically adjust learning rates for individual parameters b

Exponentially Weighted Average

The Exponentially Weighted Moving Average (EWMA) is commonly used as a smoothing technique in time series. However, due to several computational advantages (fast, low-memory cost), the EWMA is behind the scenes of many optimization algorithms in deep learning, including Gradient Descent with Momentum, RMSprop, Adam, etc.

In order to compute the EWMA, you must define one parameter β. This parameter decides how important the current observation is in the calculation of the EWMA.

$$\beta \searrow \quad \Longrightarrow \quad \text{EWMA tracks more closely the original trend}$$



$$\begin{aligned}\beta &= \text{Weight parameter}\\ \theta_t &= \text{Temperature day } t\\ W_t &= \text{EWA for day } t \text{ (set } W_0 = 0)\end{aligned}$$

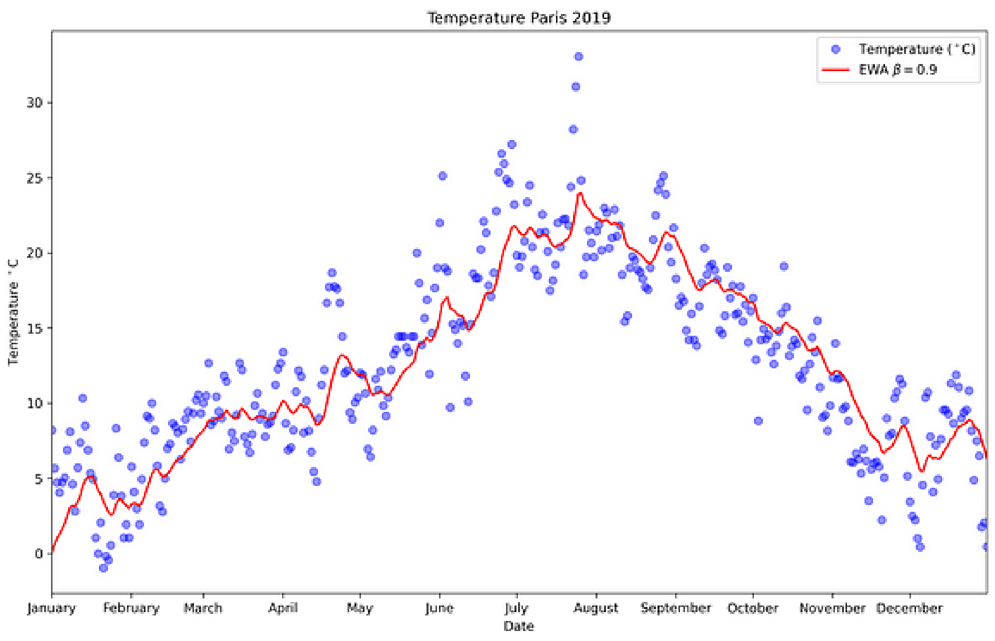
For this example, suppose that β = 0.9, so the EWA aims to combine the temperature of the current day with the previous temperatures.

$$\begin{gathered}W_1 = 0.9 \cdot W_0 + 0.1 \cdot \theta_1\\ W_2 = 0.9 \cdot W_1 + 0.1 \cdot \theta_2\\ \vdots\\ W_t = 0.9 \cdot W_{t-1} + 0.1 \cdot \theta_t\end{gathered}$$

In general to compute the EWA for a given weight parameter β we use

$$W_t = \begin{cases} 0 & t = 0 \\ \beta \cdot W_{t-1} + (1 - \beta) \cdot \theta_t & t > 0 \end{cases}$$

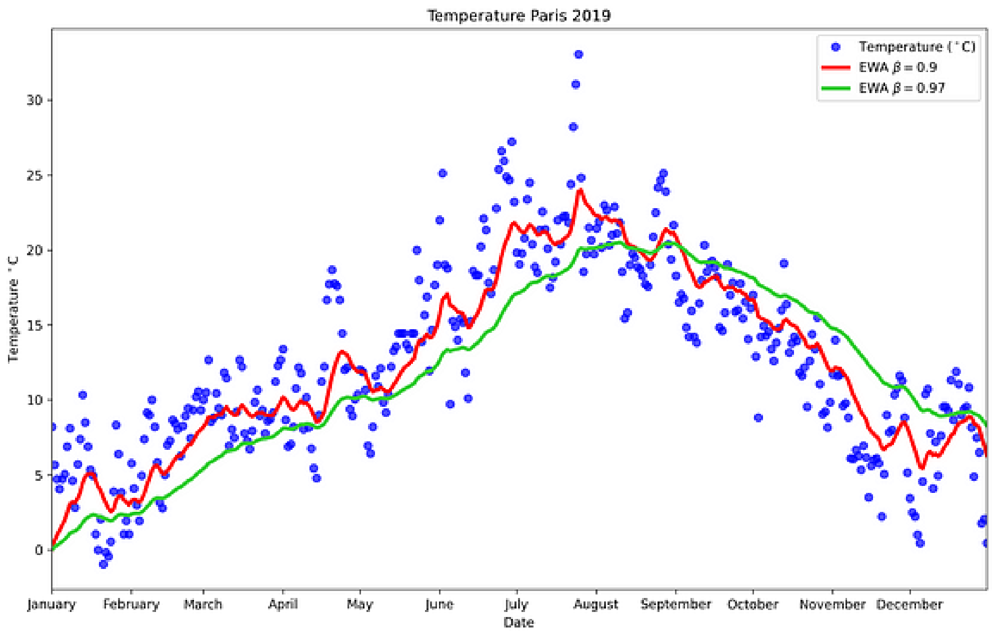
If we plot this in red, we can see that what we get is a moving average of the daily temperature, it's like a smooth, less noisy curve.



$$W_t = \underbrace{\beta \cdot W_{t-1}}_{\text{trend}} + \overbrace{(1 - \beta) \cdot \theta_t}^{\text{current value}}$$

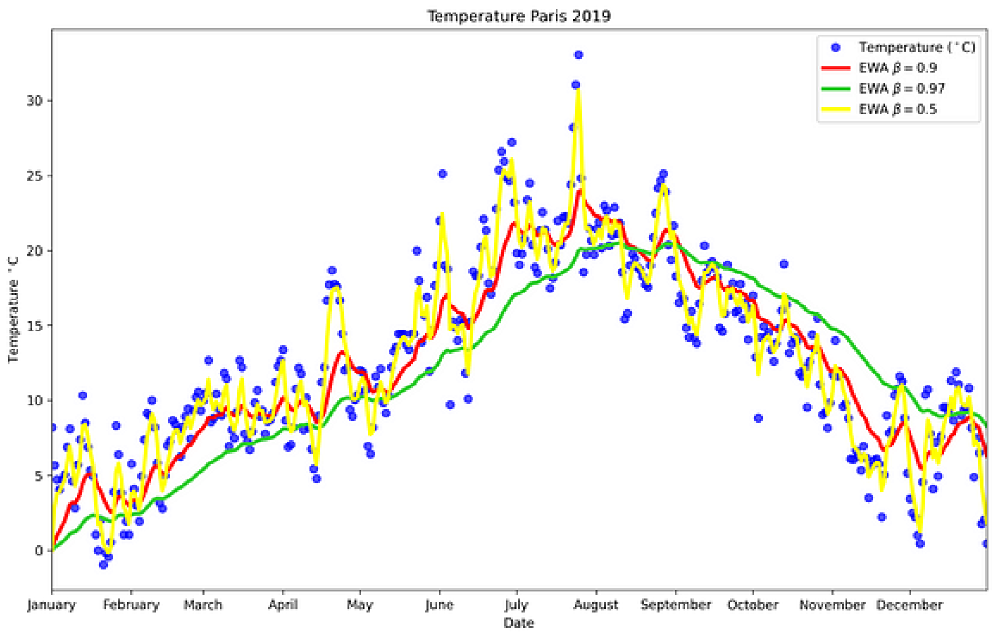
We can see that the value of β determines how important the previous value is (the trend), and (1-β) how important the current value is.

Take a value of β = 0.98 and plot it in green, notice that the curve is *smoother* because the trend now is more important (and the current temperature value is less important), so it will adapt more slowly when the temperature changes.



Lets try the other extreme and set β = 0.5, this way the graph you get is *noisier*, because it is more susceptible to the current temperature (and this includes outliers).

But it adapts more quickly to changes in temperature.



If you want to understand the meaning of the parameter β, you can think of the value

$$n_\beta = \frac{1}{1-\beta}$$

as the numbers of observations used to adapt your EWA.

β	n_β	**EWA**
0.9	10	Adapts normal
0.98	50	Adapts slowly
0.5	2	Adapts quickly

In order to go a little bit deeper into the intuitions of what this algorithm actually does.

Lets expand the 3rd term (W_3) using the main equation:

$$W_3 = 0.9 \cdot W_2 + 0.1 \cdot \theta_3$$
$$W_2 = 0.9 \cdot W_1 + 0.1 \cdot \theta_2$$
$$W_1 = 0.9 \cdot \underbrace{W_0}_{0} + 0.1 \cdot \theta_1$$

Plugin W_1 in W_2 and then in W_3:

$$W_3 = 0.9 \cdot \underbrace{(0.9\underbrace{(0.9 \cdot 0 + 0.1 \cdot \theta_1)}_{W_1} + 0.1 \cdot \theta_2)}_{W_2} + 0.1 \cdot \theta_3$$

Simplifying

$$W_3 = 0.1(\theta_3 + 0.9\theta_2 + 0.9^2\theta_1)$$

Here it is quite clear what the roll of β = 0.9 parameter is in EWA, we can see older observations are given lower weights. The weights fall exponentially as the data point gets older hence the name exponentially weighted.

In general, we have:

$$W_t = (1-\beta)(\theta_t + \beta \cdot \theta_{t-1} + \beta^2 \cdot \theta_{t-2} + \cdots + \beta^{t-3} \cdot \theta_3 + +\beta^{t-2} \cdot \theta_2 + +\beta^{t-1} \cdot \theta_1)$$

or the closed formula:

$$W_t = (1-\beta)\cdot\sum_{k=1}^{t}\beta^{t-k}\theta_k$$

If you are a visual learner, this is another approach:

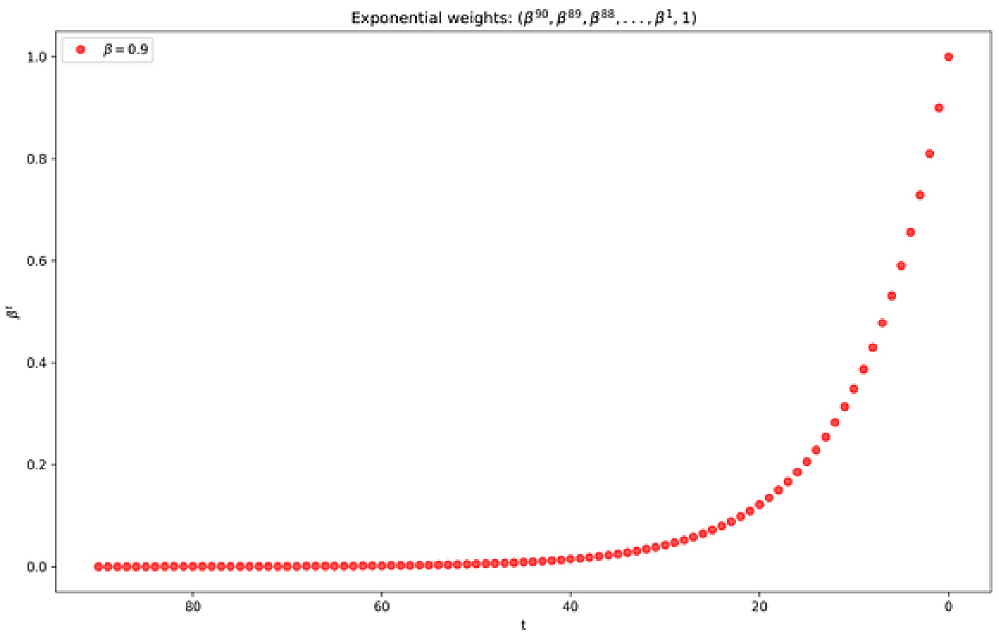
Rewrite W_t using the dot product (*):

$$W_t = (1-\beta)\cdot(\beta^0, \beta^1, \ldots, \beta^{t-2}, \beta^{t-1}) * (\theta_t, \theta_{t-1}, \ldots, \theta_2, \theta_1)$$

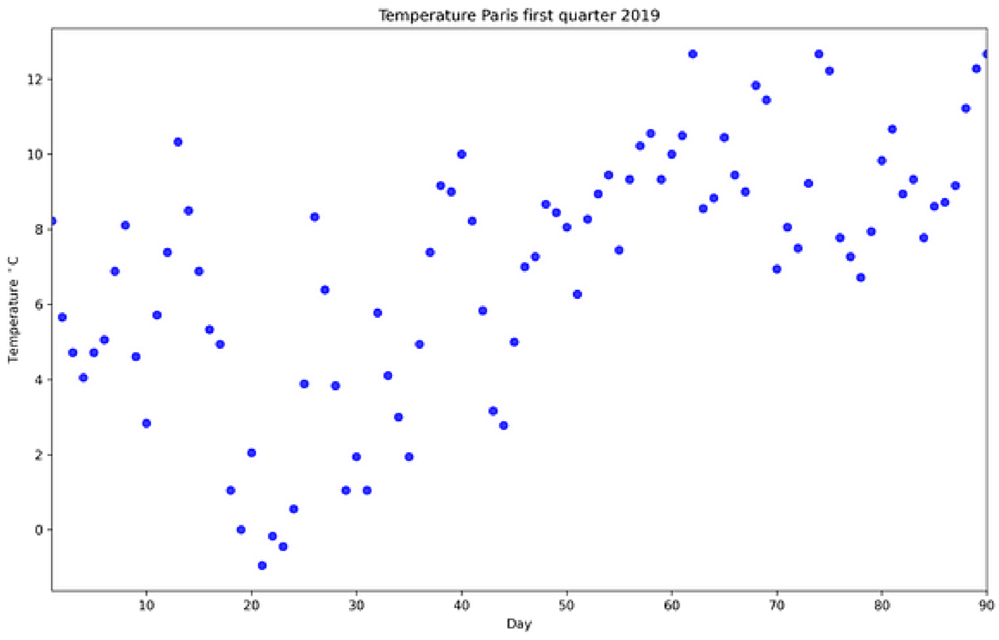
And then think W_t as the product of the following two plots:

- Exponential Weights
- Temperature

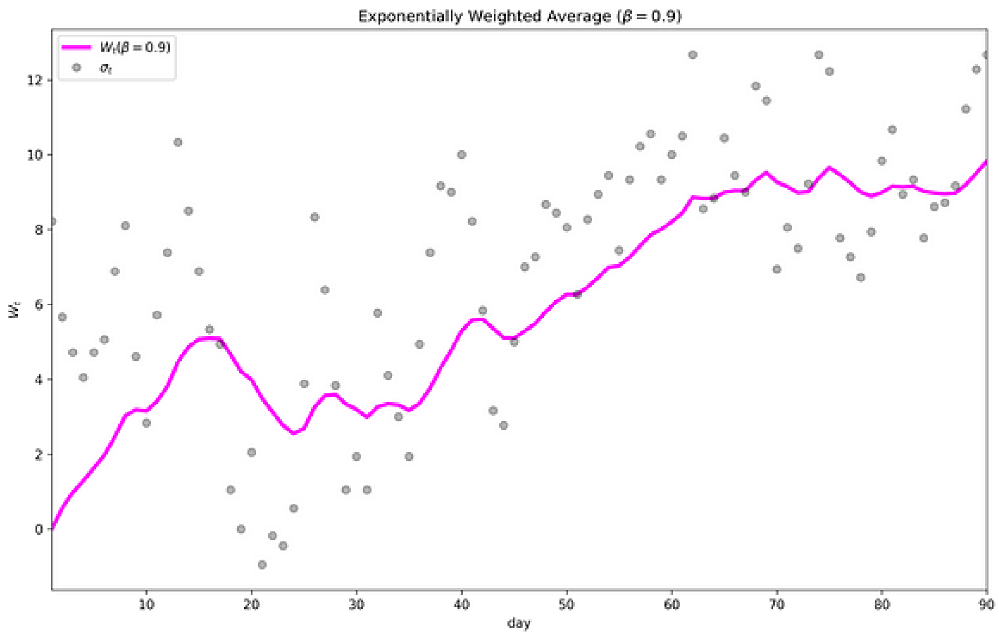
In this plot we see how the weights decay when t increase:



Then plot the temperature



Now in order to get W_t, just multiply each point (weights and temperature) and add them up, doing this for each t we obtain the magenta curve.



Explanation of the Momentum Components

1. **Velocity Term (vtv_tvt)**:
   - The velocity term is an exponentially weighted moving average of the gradients.
   - It incorporates a fraction (β) of the previous velocity and the current gradient.

     β\beta
   - This helps to smooth out the updates, reducing oscillations and leading to faster convergence.
2. **Momentum Coefficient (β\betaβ)**:
   - The momentum coefficient determines the weight given to the previous velocity.
   - A higher β (close to 1) puts more weight on the accumulated gradients, providing more smoothing.

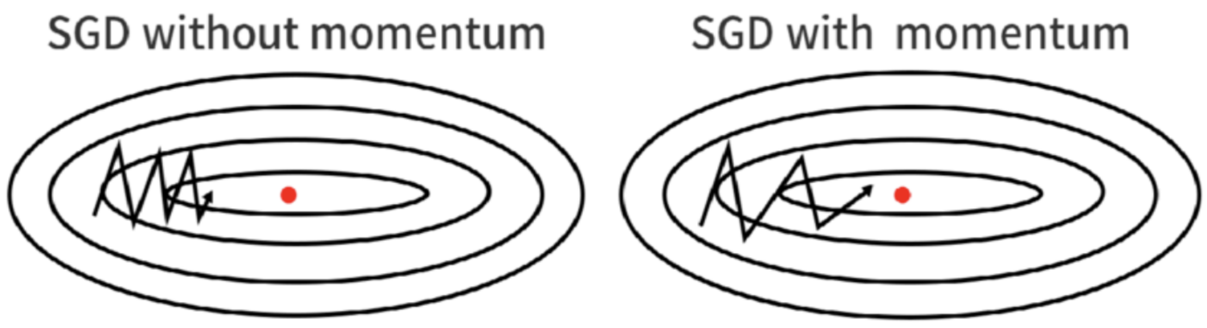
     β\beta
3. **Update Rule**:
   - The weights are updated by subtracting the learning rate times the velocity term.

- This update rule allows the optimizer to accelerate in directions with consistent gradients, leading to faster convergence.

Example of Momentum

Imagine a ball rolling down a hill:

- **Without Momentum**: The ball may oscillate back and forth, making slow progress towards the bottom.
- **With Momentum**: The ball gains speed in the direction of consistent descent and smooths out oscillations, reaching the bottom faster.


SGD without momentum
SGD with momentum