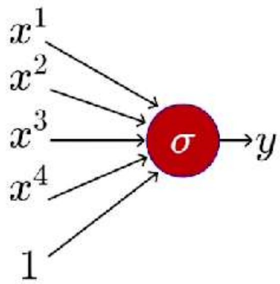
# Gradient Descent w/ Adaptive Learning Rate




$$y = \frac{1}{1+e^{-(w^T x + b)}}, \quad x = \{x^1, x^2, x^3, x^4\}$$

$$w = \{w^1, w^2, w^3, w^4\}$$

For a single given point $(x, y)$:

$$\nabla w^1 = (f(x) - y) * f(x) * (1 - f(x)) * x^1$$

$$\nabla w^2 = (f(x) - y) * f(x) * (1 - f(x)) * x^2 \text{ \& so on.}$$

If there are $n$ points, we can sum the gradients over all $n$ points to get total gradient.

$$\nabla w^2 = \sum_{i=1}^{n} (f(x) - y) * f(x) * (1 - f(x)) x_i^2$$

What happens if the feature $x^2$ is very sparse (if its value is 0 for most inputs)?

$\nabla w^2$ will be 0 for most inputs & won't get enough updates. | as $x^2 = \{0, 0, 0, \dots 1, 0, 0 \dots 0 \dots\}$

However, if the sparse feature $(x^2)$ is important, we would want to take updates to $w^2$ more seriously

# AdaGrad

## Intuition

Decay the learning rate for parameters in proportion to their update history (more updates mean more decay)

## Update Rule

$$\boxed{v_t = v_{t-1} + (\nabla w_t)^2}$$

$$\boxed{w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \epsilon}} * \nabla w_t}, \quad \boxed{b_{t+1} = b_t - \frac{\eta}{\sqrt{v_t + \epsilon}} * \nabla b_t}$$

```python
def do_adagrad(max_epochs):

  #Initialization
  w,b,eta = -2,-2,0.1
  v_w,v_b,eps = 0,0,1e-8
  for i in range(max_epochs):
    # zero grad
    dw,db = 0,0
    for x,y in zip(X,Y):

        #compute the gradients
        dw = grad_w(w,b,x,y)
        db = grad_b(w,b,x,y)

    #compute intermediate values
    v_w = v_w + dw**2
    v_b = v_b + db**2

    #update parameters
    w = w - eta*dw/(np.sqrt(v_w)+eps)
    b =b - eta*db/(np.sqrt(v_b)+eps)
```

$$v_0 = (\nabla w)^2$$

$$v_1 = (\nabla w_0)^2 + (\nabla w_1)^2$$

$$\vdots$$

$$v_n = \sum_{i=0}^{n} (\nabla w_i)^2$$

Recall that

$$\nabla w = (f(x) - y) * f(x) * (1 - f(x)) * x$$

Since $x$ is sparse, the gradient is zero for most of the steps.

Since $x$ is sparse, the gradient is zero for most of the steps.

$\therefore$ $\dfrac{\eta}{\sqrt{v_t + \epsilon}}$ decays slowly

$v_0 = (\nabla b_0)^2$

$v_1 = (\nabla b_0)^2 + (\nabla b_1)^2$

$\vdots$

$v_n = \sum_{i=0}^{n} (\nabla b_i)^2$

Recall that

$\nabla b = (f(x) - y) * f(x) * (1 - f(x))$

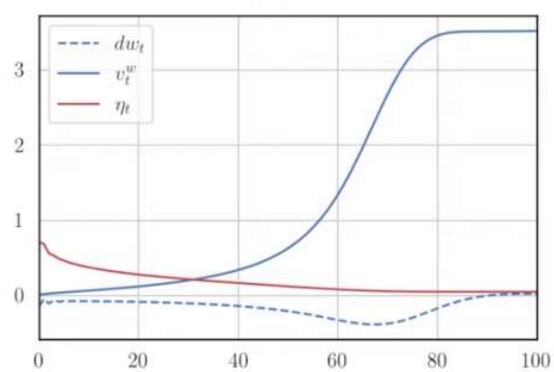
Since $x$ is sparse, the gradient is not zero for most of the steps (unless $x$ takes a large value)

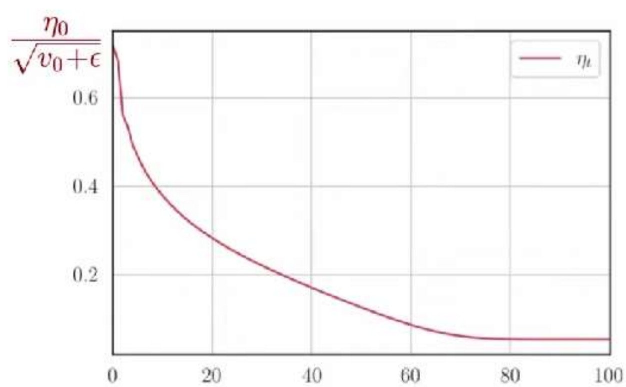
$\therefore$ $v_t$ grows rapidly &

$\dfrac{\eta}{\sqrt{v_t + \epsilon}}$ decays rapidly

$$v_t = v_{t-1} + (\nabla w_t)^2$$

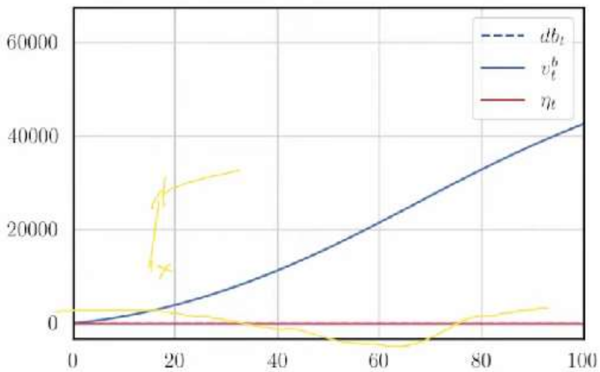
$$\frac{\eta_0}{\sqrt{v_0+\epsilon}} = \frac{0.1}{\sqrt{0.019}} = 0.72$$

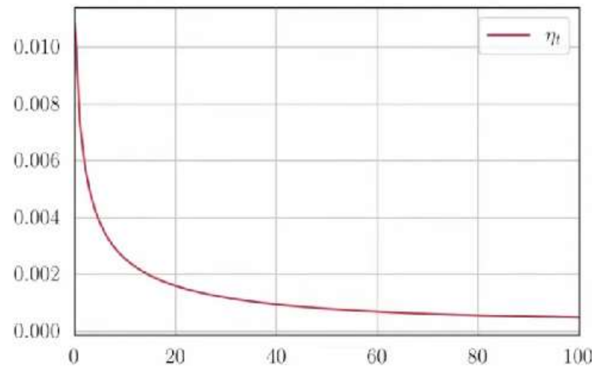


The effective learning rate $\eta_t = \frac{\eta_0}{\sqrt{v_t + \epsilon}}$

$$v_t = v_{t-1} + (\nabla b_t)^2$$




$v_t^b$ grows rapidly because of accumulating gradients.

# RMS Prop

## Intuition

Adagrad decays the learning rate very aggressively (as the denominator grows).
As a result, the frequent parameters will start receiving very small updates because of the decayed learning rate. To avoid this, why not decay the denominator & prevent its rapid growth.

## Update Rule

$$v_t = \beta v_t + (1-\beta)\nabla w_t^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \varepsilon}}\nabla w_t$$

### Adagrad

$$v_t = v_{t-1} + \nabla b_t^2$$

$$v_0 = \nabla b_0^2$$

$$v_1 = \nabla b_0^2 + \nabla b_1^2$$

$$v_2 = \nabla b_0^2 + \nabla b_1^2 + \nabla b_2^2$$

$$v_t = \nabla b_0^2 + \nabla b_1^2 + \dots + \nabla b_t^2$$

Recall that,

$$\nabla b = [(f(x)-y) * f(x) * (1-f(x))]$$

$$\therefore \frac{\eta}{\sqrt{v_t+\varepsilon}} \text{ decays rapidly for } b$$

### RMSProp

$$v_t = \beta v_{t-1} + (1-\beta)\nabla b_t^2$$

$\beta \in [0,1)$ & let $\beta = 0.9$

$$v_0 = 0.1\nabla b_0^2$$

$$v_1 = 0.09\nabla b_0^2 + 0.1\nabla b_1^2$$

$$v_2 = 0.081\nabla b_0^2 + 0.09\nabla b_1^2 + 0.1\nabla b_2^2$$

$$\vdots$$

$$v_t = (1-\beta)\sum_{\tau=0}^{t}\beta^{t-\tau}\nabla b_t^2$$

$$\therefore \frac{\eta}{\sqrt{v_t+\varepsilon}} \text{ decays slowly for } b$$

```python
def do_rmsprop(max_epochs):
  #Initialization
  w,b,eta = -4,4,0.1
  beta = 0.5
  v_w,v_b,eps = 0,0,1e-4

  for i in range(max_epochs):
    # zero grad
    dw,db = 0,0

    for x,y in zip(X,Y):

        #compute the gradients
        dw = grad_w(w,b,x,y)
        db = grad_b(w,b,x,y)

    #compute intermediate values
    v_w = beta*v_w +(1-beta)*dw**2
    v_b = beta*v_b + (1-beta)*db**2

    #update parameters
    w = w - eta*dw/(np.sqrt(v_w)+eps)
    b = b - eta*db/(np.sqrt(v_b)+eps)
```

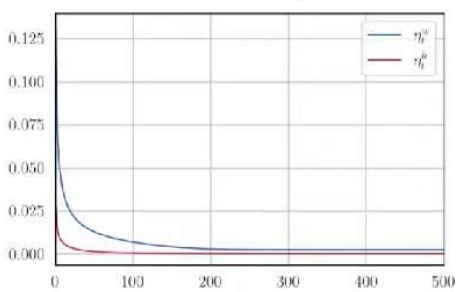
RMSProp will converge faster than AdaGrad by being less aggressive on the decay.

However, there will be many oscillations. Why?

Maybe because after some iterations, the learning rate remains constant, so the algo gets into infinite oscillation around minima.

In AdaGrad, $v_t = v_{t-1} + \nabla w_t$ never decreases despite gradients becoming zero after some iterations. Could this be same for RMSProp?

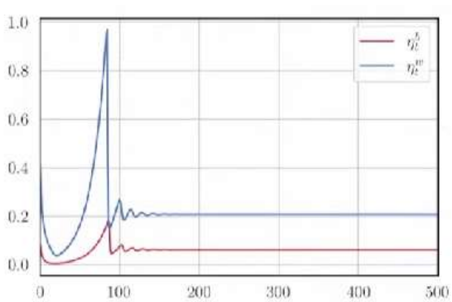
AdaGrad: $v_t = v_{t-1} + (\nabla w_t^2)$




In AdaGrad, the learning rate monotonically decreases, due to ever growing denominator

In RMSProp, the learning rate may increase, decrease or remain constant due to moving average of gradients in the denominator.

RMSProp: $v_t = \beta v_{t-1} + (1-\beta)(\nabla w_t^2)$




Solution ?

Set initial learning rate properly :)

# Ada Delta

Avoids setting initial learning rate $\eta_0$.

for $t$ in range(1, N):

1. $\rightarrow \nabla w_t$
2. $\rightarrow v_t = \beta v_{t-1} + (1-\beta)(\nabla w_t)^2$
3. $\rightarrow \Delta w_t = -\dfrac{\sqrt{u_{t-1}+\varepsilon}}{\sqrt{v_t+\varepsilon}} \nabla w_t$
4. $\rightarrow w_{t+1} = w_t + \Delta w_t$
5. $\rightarrow u_t = \beta u_{t-1} + (1-\beta)(\Delta w_t)^2$

Since $\Delta w_t$ is used to update the weights, it is called Adaptive Delta.

$u_t$ which we compute at $t$, will be used only in the next iteration.

Now the numerator, in the effective learning rate, is a function of past gradients.

Also, we are taking only a small fraction $(1-\beta)$ of $(\Delta w_t)^2$

# Adam (Adaptive Moments)

## Intuition

Do everything RMSProp does to solve the decay problem of AdaGrad. Plus use a cumulative history of the gradients.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla w$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla w_t)^2$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

Bias correction

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$$

Typically,

$\beta_1 = 0.9$

$\beta_2 = 0.999$

```python
def do_adam_sgd(max_epochs):

  #Initialization
  w,b,eta = -4,-4,0.1
  beta1,beta2 = 0.9,0.999
  m_w,m_b,v_w,v_b = 0,0,0,0

  for i in range(max_epochs):
    dw,db = 0,0
    eps = 1e-10
    for x,y in zip(X,Y):

        #compute the gradients
        dw = grad_w_sgd(w,b,x,y)
        db = grad_b_sgd(w,b,x,y)

    #compute intermediate values
    m_w = beta1*m_w+(1-beta1)*dw
    m_b = beta1*m_b+(1-beta1)*db
    v_w = beta2*v_w+(1-beta2)*dw**2
    v_b = beta2*v_b+(1-beta2)*db**2

    m_w_hat = m_w/(1-np.power(beta1,i+1))
    m_b_hat = m_b/(1-np.power(beta1,i+1))
    v_w_hat = v_w/(1-np.power(beta2,i+1))
    v_b_hat = v_b/(1-np.power(beta2,i+1))

    #update parameters
    w = w - eta*m_w_hat/(np.sqrt(v_w_hat)+eps)
    b = b - eta*m_b_hat/(np.sqrt(v_b_hat)+eps)
```

```
w = w - eta*m_w_hat/(np.sqrt(v_w_hat)+eps)
b = b - eta*m_b_hat/(np.sqrt(v_b_hat)+eps)
```

# Why Bias Correction?

We are taking a running average of gradients as $m_t$.
The reason we are doing this is that we don't want to rely too much on the current gradient & instead rely on the overall behaviour of gradients over many timesteps.

We are looking at EXPECTED VALUE of the gradient.

However, we are calculating $E[m_t]$ instead of $E[\nabla w_t]$.
Ideally, we would want $E[m_t]$ to be equal to $E[\nabla w_t]$

$$m_0 = 0$$

$$m_1 = \beta m_0 + (1-\beta)\nabla w_1 = (1-\beta)\nabla w_1$$

$$m_2 = \beta m_1 + (1-\beta)\nabla w_2 = \beta(1-\beta)\nabla w_1 + (1-\beta)\nabla w_2$$

$$= (1-\beta)(\beta\nabla w_1 + \nabla w_2)$$

$$m_3 = (1-\beta)\sum_{z=1}^{3}\beta^{3-z}\nabla w_z$$

$$\boxed{\therefore\ m_t = (1-\beta)\sum_{z=1}^{t}\beta^{t-z}\nabla w_z}$$

Taking expectation on both side,

$$E[m_t] = E\left[(1-\beta)\sum_{z=1}^{t}\beta^{t-z}\nabla w_z\right]$$

$$E[m_t] = (1-\beta)E\left[\sum_{z=1}^{t}\beta^{t-z}\nabla w_z\right]$$

$$E[m_t] = (1-\beta)\, E\left[\sum_{z=1} \beta \quad \nabla w_z\right]$$

$$E[m_t] = (1-\beta) \sum_{z=1}^{t} \beta^{t-z} E[\nabla w_z]$$

Assumption: All $\nabla w_z$ comes from the same distribution

$$E[\nabla w_z] = E[\nabla w] \quad \forall z$$

$$\therefore\ E[m_t] = E[\nabla w](1-\beta) \sum_{z=1}^{t} \beta^{t-z}$$

$$E[m_t] = E[\nabla w](1-\beta)(\beta^{t-1} + \beta^{t-2} + \cdots + \beta^0)$$

$$E[m_t] = E[\nabla w](1-\beta) \frac{1-\beta^t}{1-\beta}$$

The last ratio is the sum of GP w/ common ratio $\beta$

$$E[m_t] = E[\nabla w](1-\beta^t)$$

$$E\left[\frac{m_t}{1-\beta^t}\right] = E[\nabla w]$$

Hence we apply bias correction because the expected value of $\hat{m}_t$ is the same as expected value of $E[\nabla w_t]$

What if we don't do bias correction?

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla w_t)^2, \quad \beta_2 = 0.999$$

Let $\nabla w_0 = 0.1$

$$v_0 = 0.999 * 0 + 0.001(0.1)^2 = 0.00001$$

$$\eta_t = \frac{1}{\sqrt{0.00001}} = 316.22$$

$\sqrt{0.00001}$

$$v_2 = 0.999 * v_0 + 0.001(0)^2 = 0.0000099$$

$$\eta_t = \frac{1}{\sqrt{0.0000099}} = 316.38$$

-> initial steps are very large

# AdaMax & MaxProp

Let's revist $L^p$ norm.

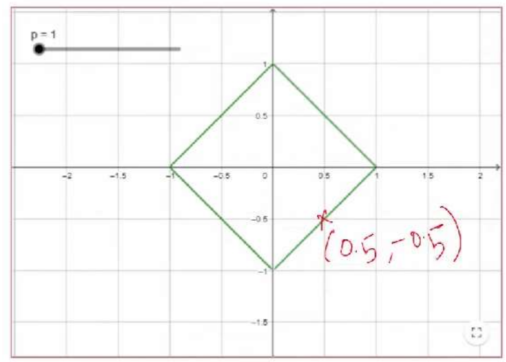
$$L^p = \left(|x_1|^p + |x_2|^p + |x_3|^p + \cdots + |x_n|^p\right)^{\frac{1}{p}}$$

Let's fix $L^p = 1$ & vary $p$ to visualize it.

$$1 = \left(|x_1|^p + |x_2|^p\right)^{\frac{1}{p}}$$

$$1^p = |x_1|^p + |x_2|^p$$

$$1 = |x_1|^p + |x_2|^p$$




We can choose any value for $p \geq 1$

$|x|$ raised to a high value of $p$, becomes too small to represent. This leads to numerical instability.