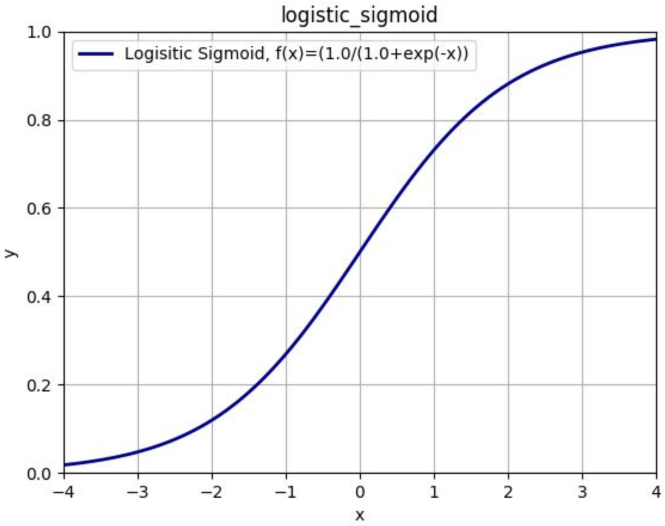
ACTIVATION FUNCTIONS

Logistic Sigmoid

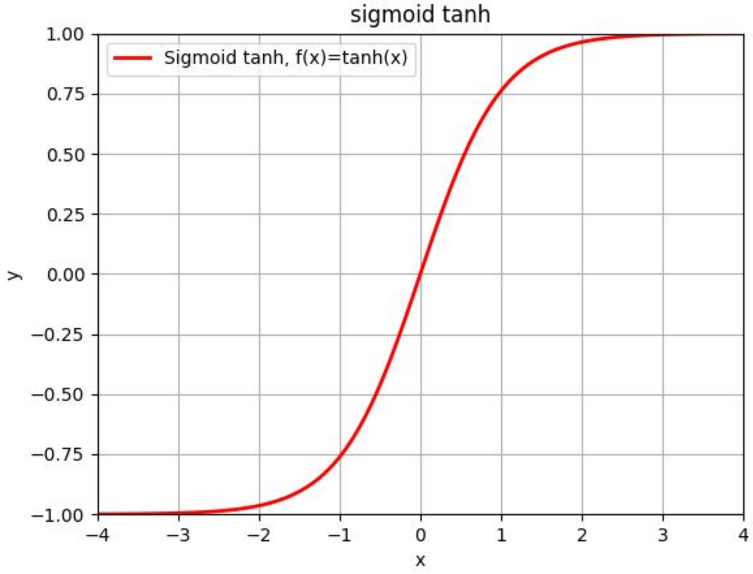
$$f(x) = \frac{1}{1+e^{-x}}$$



- Takes input in the range (-∞,∞) and maps them to the range (0,1)
- Usually used as output activation for binary classification problems: as its output is in the range (0,1) the values can interpreted as probabilities!
- Calculating its gradient is quite easy.
- However, the gradients become very small near either extremities of the function; so the learning becomes very slow.

Hyperbolic Tangent (tanh)

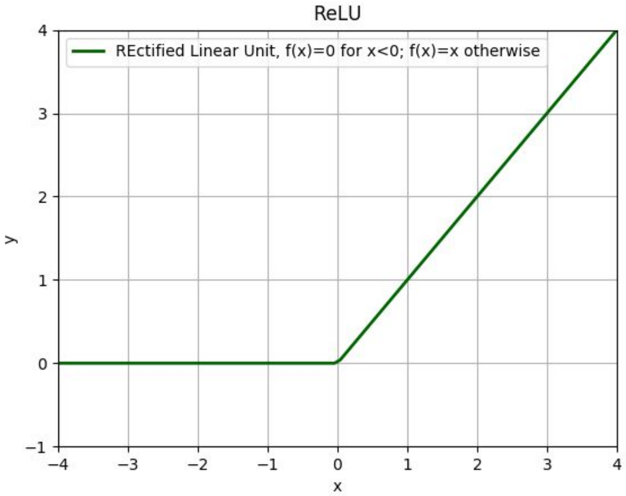
$$f(x) = tanh(x)$$



- Also a sigmoid.
- Takes input in the range (-∞,∞) and maps them to the range (-1,1)
- A strong point about tanh is that strongly negative inputs are mapped towards -1 and strongly positive ones towards +1.
- It is a slightly better choice from the class of sigmoids than the logistic sigmoid; but still suffers from the same problems: saturation near the extremities.

Rectified Linear Unit (ReLU)

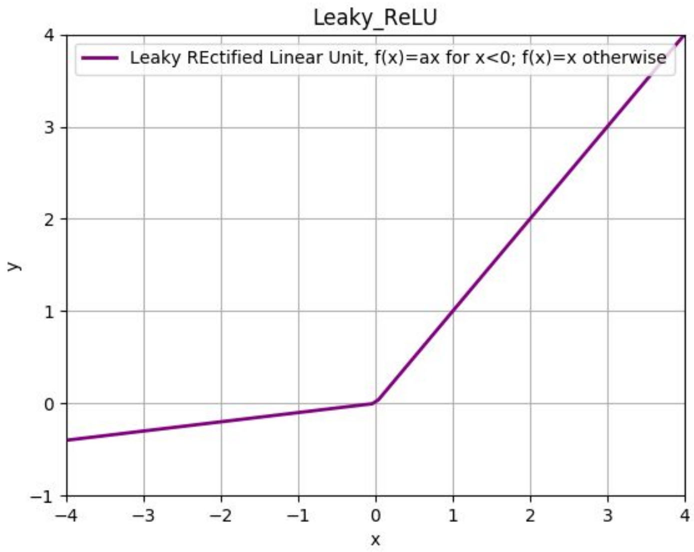
$$f(x) = \begin{cases} \mathrm{x} & \mathrm{x} \geq 0 \\ 0 & \mathrm{x} < 0 \end{cases}$$



- Takes its name from the half-wave rectifier from electronics
- A unit employing the rectifier function as the activation is called the Rectified Linear Unit (ReLU).
- Most popular activation for deep neural networks.
- One problem is that it is unbounded for positive x.
- Also, some neurons in the network become stagnant after a while and remain in the “dead” state. This typically arises when the learning rate is set too high.

Leaky ReLU

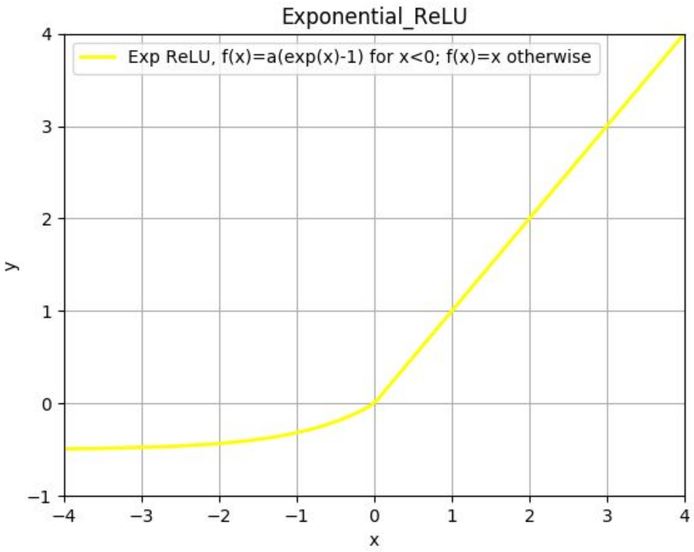
$$f(x) = \begin{cases} \mathrm{x} & \mathrm{x} \geq 0 \\ \alpha\, \mathrm{x} & \mathrm{x} < 0 \end{cases}$$



- To mitigate the dead neurons problem, a sort of "leak" is added to the ReLU function.
- The leak is also linear, but with a small slope (0.01 or so).
- Some researchers have reported success with this function; but still, the ReLU remains the most popular activation for deep learning, with the learning rate not too high.

Exponential Linear Unit (ELU)

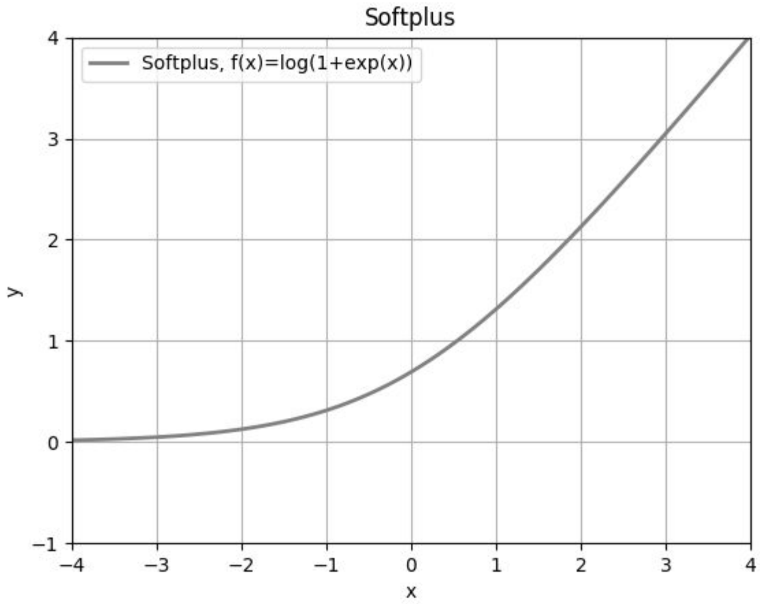
$$f(x) = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



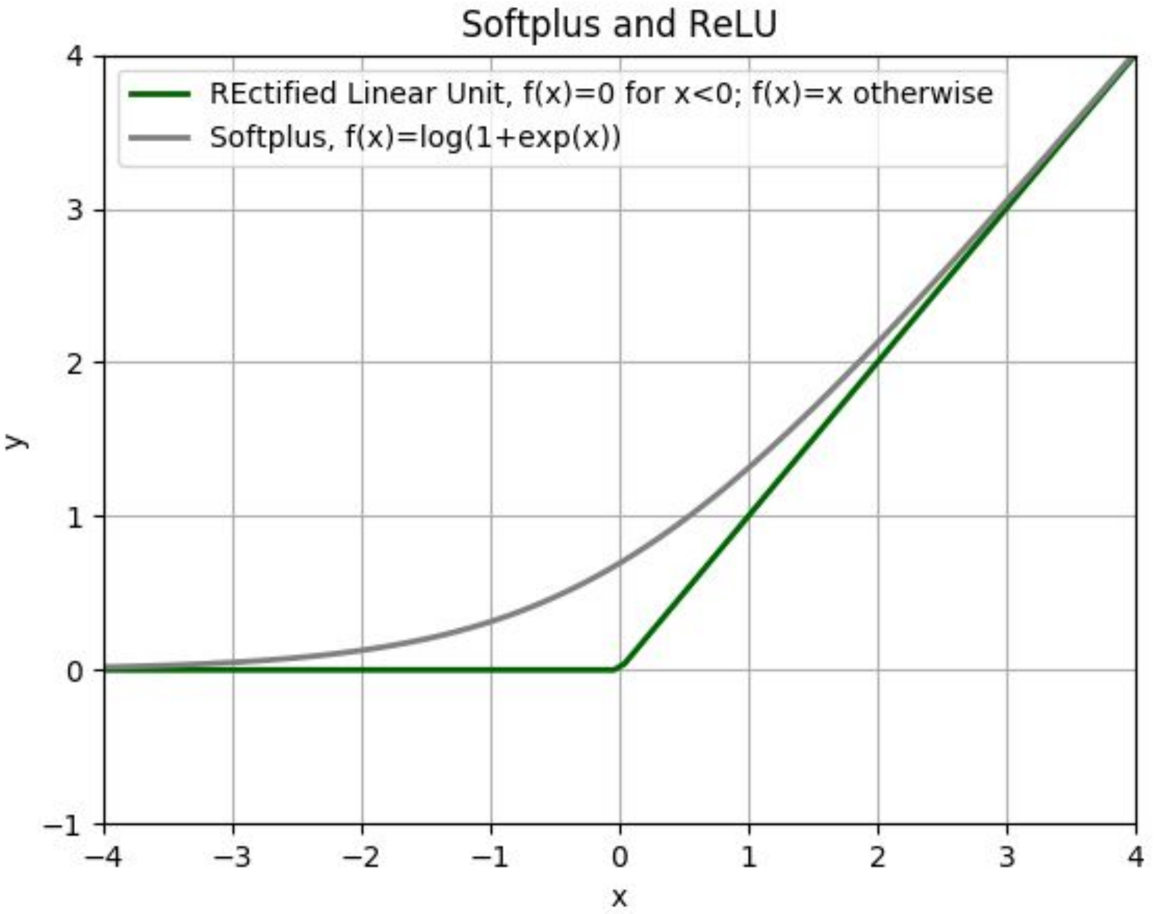
- Another variation of the leaky ReLU is the Exponential Linear Unit (ELU) where the leak is in the form of the exponential function.

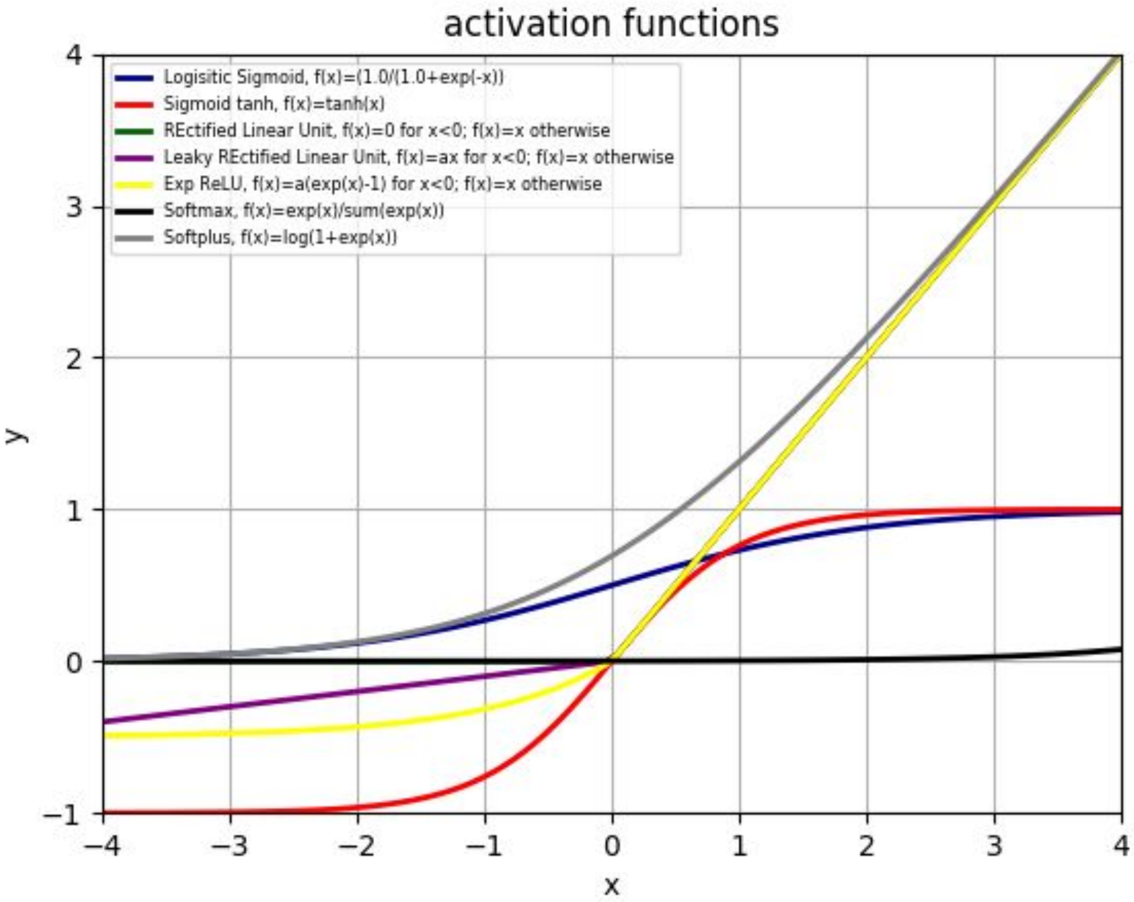
SoftPlus

$$f(x) = log(1 + e^x)$$



- The softplus activation function can be viewed as a continuous and differentiable alternative to the ReLU.
- Instead of having a sharp point at 0 like the ReLU, it is smooth and thus, differentiable.
- Its derivative is easy to calculate and is surprisingly reminiscent of another activation function.


Softplus and ReLU
REctified Linear Unit, f(x)=0 for x<0; f(x)=x otherwise
Softplus, f(x)=log(1+exp(x))
4
3
2
1
0
−1
y
−4
−3
−2
−1
0
1
2
3
4
x

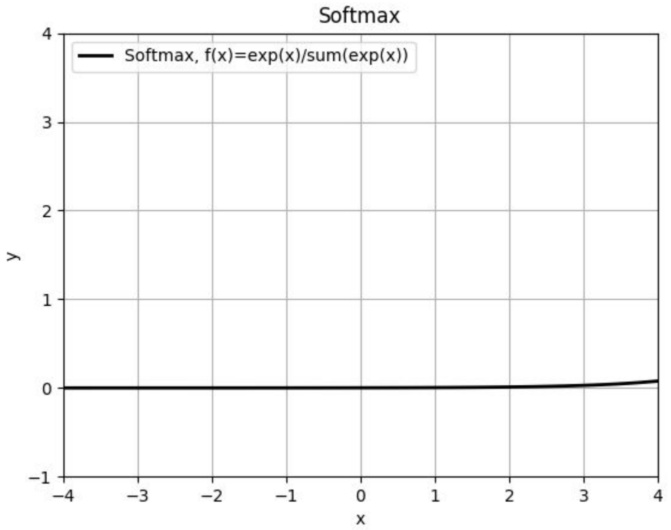

activation functions
Logisitic Sigmoid, f(x)=(1.0/(1.0+exp(-x))
Sigmoid tanh, f(x)=tanh(x)
REctified Linear Unit, f(x)=0 for x<0; f(x)=x otherwise
Leaky REctified Linear Unit, f(x)=ax for x<0; f(x)=x otherwise
Exp ReLU, f(x)=a(exp(x)-1) for x<0; f(x)=x otherwise
Softmax, f(x)=exp(x)/sum(exp(x))
Softplus, f(x)=log(1+exp(x))
y
x
4
3
2
1
0
−1
−4
−3
−2
−1
0
1
2
3
4

Output activation functions

1. For regression: since the target is a continuous variable, the activation function is linear and mostly is $f(x)=x$
2. For binary classification: the target is either 0 or 1; so the logistic sigmoid is used as the activation function as it throws out values between 0 and 1; which can be classified as 0 and 1 based on a threshold (usually 0.5)
3. For multiclass classification: the target is in the form of more than two classes (eg: [good, better, best, bad, worse, worst], or [outstanding, exceeds expectations, acceptable, poor, dreadful, troll] as in the Harry Potter series O.W.L.s!, or image classification like [cat, dog, horse, car, person, aeroplane] etc as in the CIFAR-10 dataset; so the softmax function is used as the output activation.

Softmax

$$f(x) = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}; \text{ for j=1 to K}$$



- The softmax function is usually used as the output activation for multiclass classification problems.
- Not only does it output in the range (0,1) but also the sum of its outputs is 1!
- Thus, it is interpreted as giving the probability that a given output belongs to a particular class.

LOSS FUNCTIONS

Loss Functions for Regression

Mean Absolute Error (MAE, L1 loss)

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - y_i^p|$$

Mean Square Error (MSE, Quadratic loss, L2 loss)

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - y_i^p)^2$$

Smooth Absolute Error (Huber Loss)

$$L_\delta(y, y^p) = \begin{cases} \frac{1}{2}(y - y^p)^2 & |y - y^p| \leq \delta \\ \delta|y - y^p| - \frac{1}{2}(\delta)^2 & \text{otherwise} \end{cases}$$

Log-Cosh Loss (log(cosh))

$$L(y, y^p) = \sum_{i=1}^{N} log(cosh(y - y^p))$$

Loss Functions for Classification

Log Loss (Cross-Entropy)

Binary Classification: (Binary Cross Entropy):

$$L(y, y^p) = -\frac{1}{N}\sum_{i=1}^{N}(y_i log(y_i^p) + (1 - y_i)log(1 - y_i^p))$$

Multiclass Classification: (Categorical Cross Entropy):

$$L(y, y^p) = -\frac{1}{N}\sum_{i=1}^{N} y_i log(y_i^p)$$

Hinge Loss

$$L(y, y^p) = -\frac{1}{N}\sum_{i=1}^{N} max(1 - y_i y_i^p)$$

OPTIMIZERS

Batch Gradient Descent

Computes the gradient of the entire training set for updation

$$\theta = \theta - \eta \nabla_{\theta} J(\theta)$$

Stochastic Gradient Descent

SGD performs the parameter update for each training example (therefore, its expected to be very slow)

$$\theta = \theta - \eta \nabla_\theta J\left(\theta; x^{(i)}; y^{(i)}\right)$$

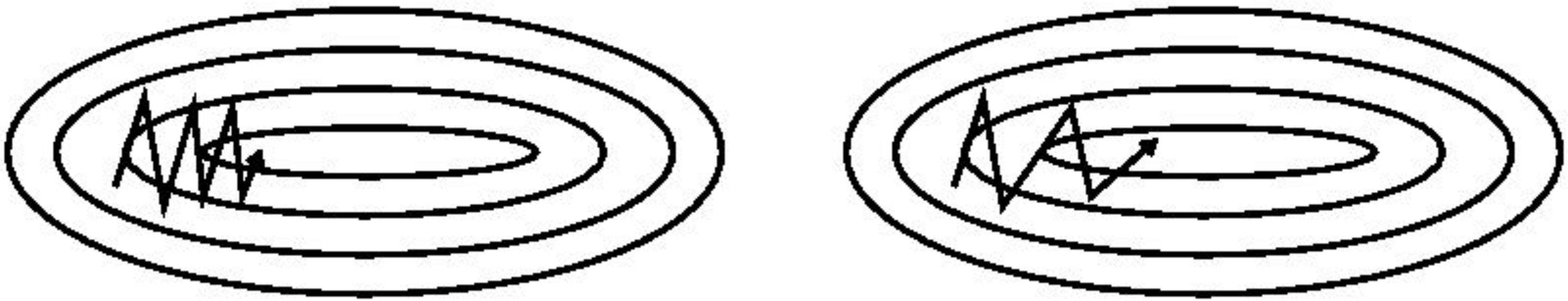
Mini-Batch Gradient Descent

Mini-batch gradient descent combines the previous two approaches and updates the parameters for a small batch (of size n i.e. i to i+n) of training examples.

$$\theta = \theta - \eta \nabla_\theta J\left(\theta; x^{(i:i+n)}; y^{(i:i+n)}\right)$$

Momentum

SGD or even Mini-batch GD may traverse the terrain of the loss function in oscillations. Momentum helps to accelerate the descent in the right direction and minimizes oscillations in the other directions.



$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

Momentum parameter (γ) is usually set to 0.9

AdaGrad

$$g_{t,i} = \nabla_\theta J(\theta_{t,i})$$

Is the gradient at the current time step for the parameter (θ_i)

$$\theta_{t+1,i} = \theta_{t,i} - \eta g_{t,i}$$

Becomes the SGD formula

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii}+\epsilon}} g_{t,i}$$

Is the AdaGrad optimizer where G_t is the diagonal matrix of the squares of g_t's upto the current time step. Hence the name, <u>Ada</u>ptive <u>Grad</u>ient. Usually, η=0.01 and ϵ=10^{-8}

RMSProp

Improvement on AdaGrad. Instead of summing all the previous time step gradients, we take the sum of a batch of the previous time step gradients.
Instead of storing the g_t's, we introduce a running average (moving average) of the sum of the past g_t's

$$g_{t,i} = \nabla_\theta J(\theta_{t,i})$$
$$\theta_{t+1,i} = \theta_{t,i} - \eta g_{t,i}$$

Same as AdaGrad

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

Moving average, with γ=0.9

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_{t,i}$$

The square root is like the RMS velocity formula, hence the name RMSProp.
Usually, γ=0.9, η=0.001 and ϵ=10^{-8}

ADAM

A combination of RMSProp and Momentum.
RMSProp used the decaying average of squares of past gradients.
Momentum used the decaying average of the past gradients.
ADAM (Adaptive Moment estimation) uses both!

$$v_t = \beta_1 v_{t-1} + (1 - \beta_1) g_t$$

Similar to Momentum; the first moment of g_t

$$E_t = \beta_2 E_{t-1} + (1 - \beta_2) g_t^2$$

Similar to RMSProp; second moment of g_t

$$\hat{v}_t = \frac{v_t}{1-\beta_1^t}$$

$$\hat{E}_t = \frac{E_t}{1-\beta_2^t}$$

Updation of estimates of the first and second moments of the gradients, betas are raised to power t.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{E}_t + \epsilon}} \hat{v}_t$$

ADAM updation. Default values are: β_1=0.9, β_2=0.999, η=0.001 and ϵ=10^{-8}