Stochastic Methods in Optimization

Comparative analysis of Stochastic Optimization methods with binary classification using a CNN

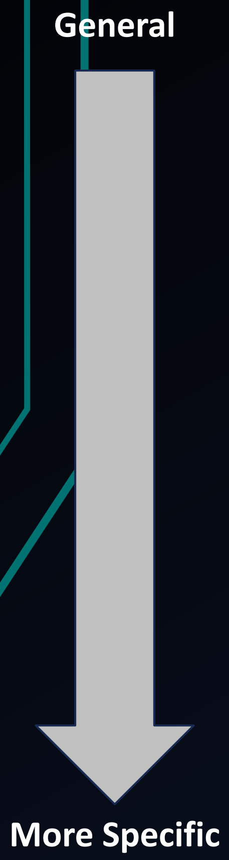
New York Institute of Technology

DTSC 615/ Fall 21 - Optimization Method for Data Science

Instructor: Dr. Jerry Cheng

Team Members: Michael Trzaskoma, Hui (Henry) Chen, Bofan He, Ephraim Hallford

Stochastic Optimization Methods



- Gradient Descent
- Stochastic Gradient Descent (SGD)
- Adaptive Gradient Algorithm (Adagrad)
- Adaptive Learning Rate Method (Adadelta)
- Root Mean Squared Propagation (RMSProp)
- Adaptive Moment Estimation (ADAM)
- Infinite norm for ADAM (Adamax)

Adaptive Gradient Algorithm (Adagrad)

- Recap:
  GD, SGD, and mini-batch GD try to minimize the loss function with respect to the weights. And all of their learning rate η are constant for all neurons and hidden layers.
- Adagrad Optimizer Propose:
  Use different η for each and every neurons, along with the hidden layers, based on different iterations.
- Why?
  Dataset features:
  - Dense
  - Sparse

cont.

Weights Updating Formula

$$\omega_t = \omega_{t-1} - \eta \frac{dL}{d\omega_{t-1}}$$

- w - weight
- t - current iteration
- η - a constant learning rate
- L - loss

Adagrad

$$\omega_t = \omega_{t-1} - \eta_t^i \frac{dL}{d\omega_{t-1}}$$

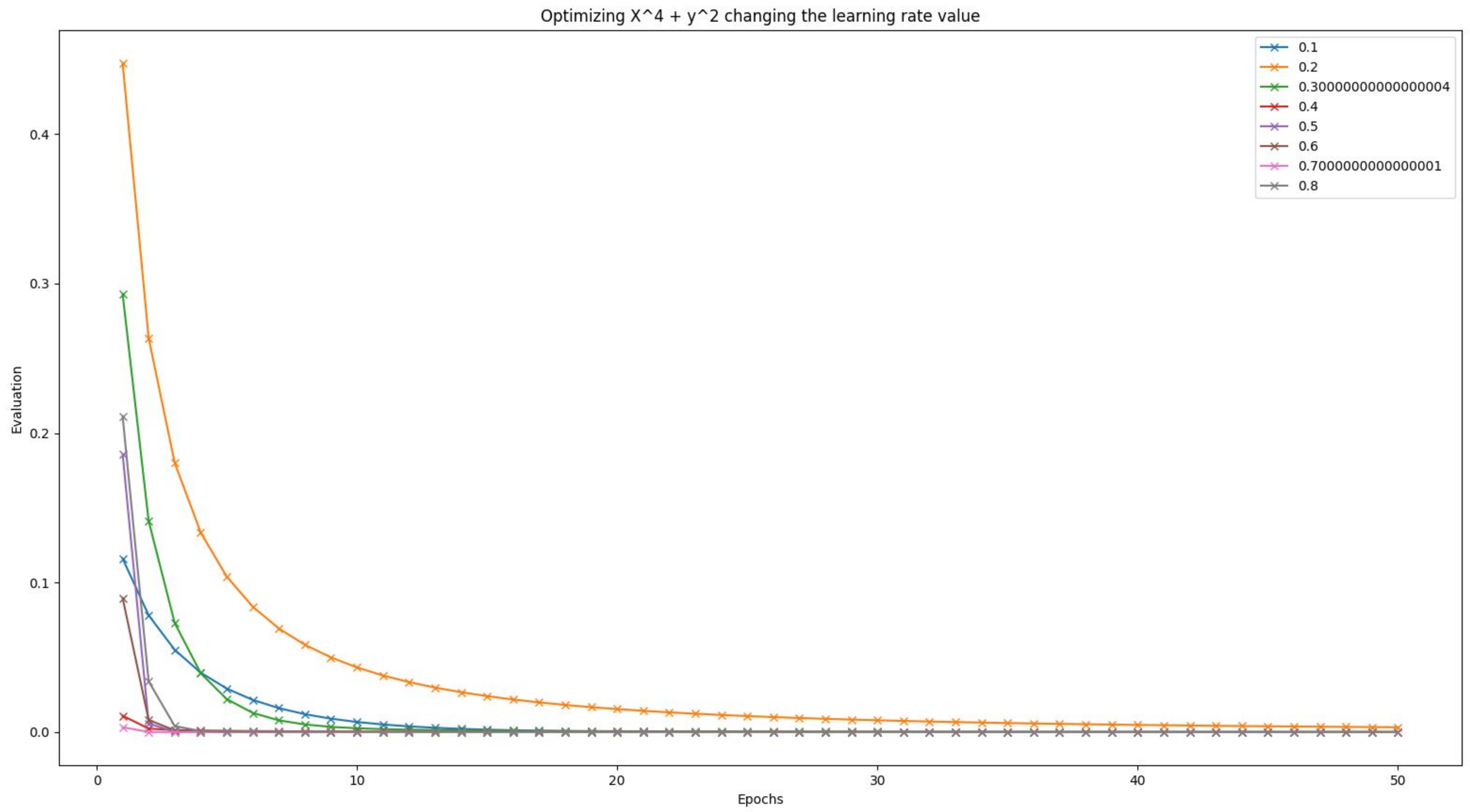
- w - weight
- t - current iteration
- η_t^i - learning rate
- L - loss

cont.

$$\eta_t^i = \frac{\eta}{\sqrt{\alpha_t + \epsilon}}$$

- η - a constant learning rate (default 0.001)
- ϵ - a small constant (positive integer), which (if α_t = 0) prevents η_t^i= ∞ and the problem of GD.
- t - iteration
- $\alpha_t = \sum_{i=1}^{t} \left(\frac{dL}{d\omega_i}\right)^2$

  Since $\left(\frac{dL}{d\omega_i}\right)$ is squared, α_t turns out to be a large value then η_t^i decreases. As a result, w_t gradually decreases (on updation) and converge.


Optimizing X^4 + y^2 changing the learning rate value
Evaluation
Epochs
0.0
0.1
0.2
0.3
0.4
0
10
20
30
40
50
0.1
0.2
0.30000000000000004
0.4
0.5
0.6
0.7000000000000001
0.8

Adaptive Delta: Adadelta

- Recap:

  Adagrad utilizes:
  - Sum of squared gradients and Adaptive learning rate

- Adadelta Optimizer Propose:
  - Updating based on:
    - Decaying squared gradient averages
    - Momentum
    - Previous change in parameters

- Why?
  - Allow recent gradients to still be impactful
  - Remove need of initialization highly impactful variables (learning rate)

Adadelta Equations

Accumulate Gradient

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

Initial: $E[g^2]_0 = 0$

Accumulate Updates

$$E[\Delta x^2]_t = \rho E[\Delta x^2]_{t-1} + (1-\rho)\Delta x_t^2$$

Initial: $E[\Delta x^2]_0 = 0$

- ρ - momentum constant = (0.9 ~ 0.99)
- t - iteration
- g - gradient
- Δx - change in parameter

Adadelta Equations Cont.

Root Mean Square of Parameter Change

$$RMS[\Delta x]_t = \sqrt{E[\Delta x^2]_t + \varepsilon}$$

Update Computation

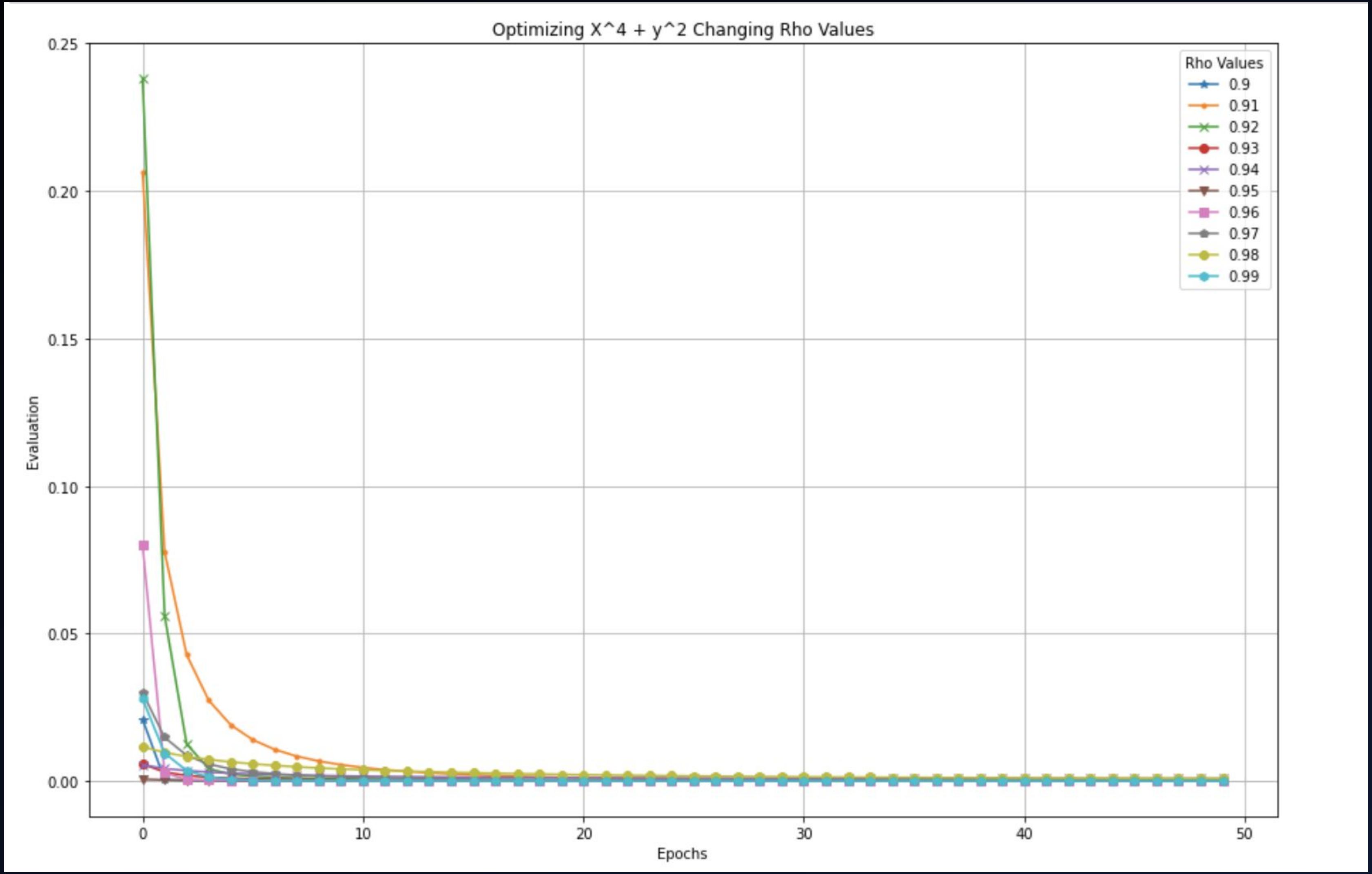
$$\Delta x_t = -\frac{RMS[\Delta x]_{t-1}}{RMS[g]_t} g_t$$

Root Mean Square of Gradients

$$RMS[g]_t = \sqrt{E[g^2]_t + \varepsilon}$$

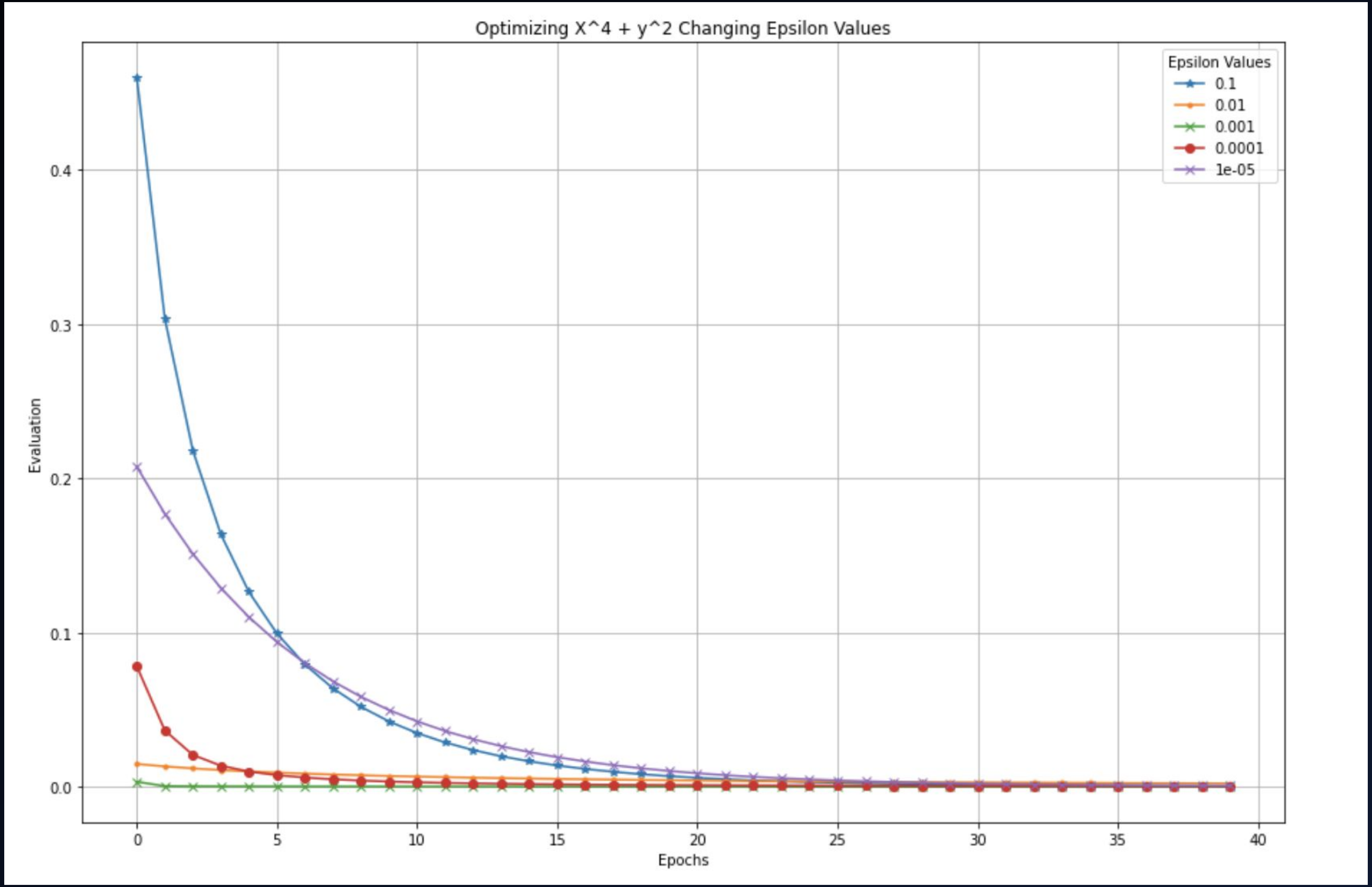
- g - gradient
- Δx - change in parameter
- ε - Very small constant
  = (1^n where n ≤ 0)
- t - iteration

Adadelta Plots



Adjusting ρ while is ε = 0.001

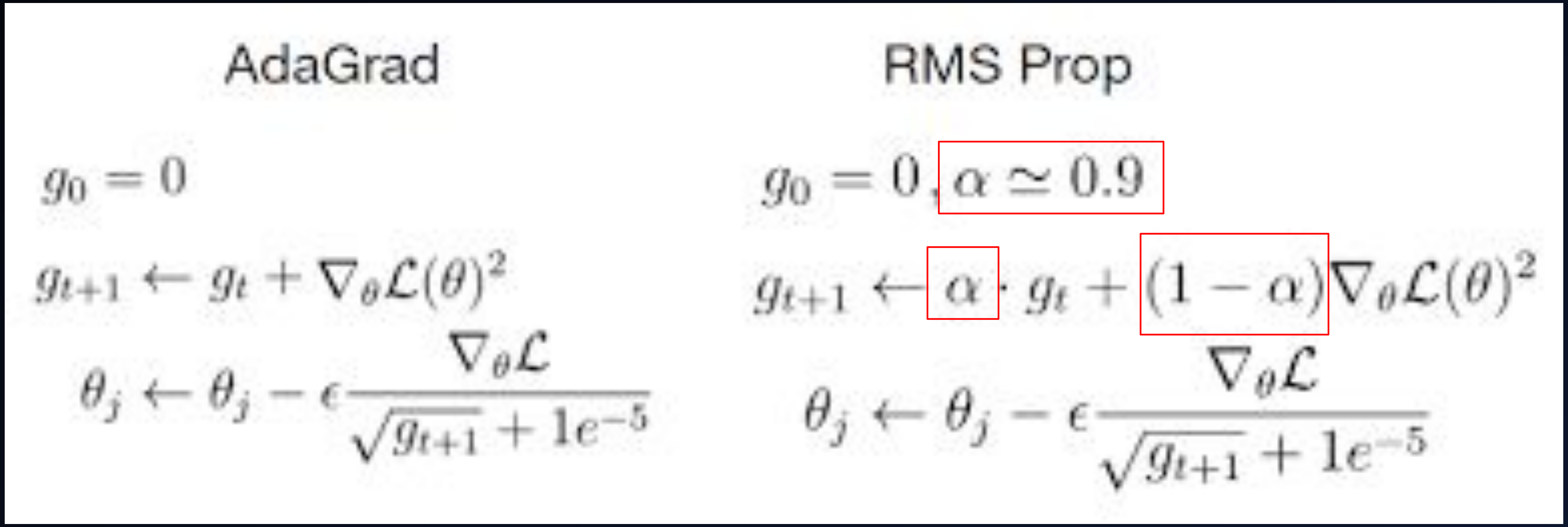
Adadelta Plots Cont.



Adjusting ε while ρ = 0.99

Root Mean Squared Propagation (RMSProp)

- Recap:
  RMSProp, is an extension of gradient descent and the AdaGrad version of gradient descent that uses a decaying average of partial gradients in the adaptation of the step size for each parameter. The use of a decaying moving average allows the algorithm to forget early gradients and focus on the most recently observed partial gradients seen during the progress of the search, overcoming the limitation of AdaGrad.
- RMSProp Propose:
  Gradient descent can be updated to use an automatically adaptive step size for each input variable using a decaying moving average of partial derivatives.
- Why?(Features)
  - allows the step size in each dimension used by the optimization algorithm to be automatically adapted based on the gradients seen for the variable (partial derivatives) over the search.
  - Fast the progress of the search, for big amount of Gradient Descent search to locating the optima faster and accurately



AdaGrad

$$g_0 = 0$$

$$g_{t+1} \leftarrow g_t + \nabla_\theta \mathcal{L}(\theta)^2$$

$$\theta_j \leftarrow \theta_j - \epsilon \frac{\nabla_\theta \mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}}$$

RMS Prop

$$g_0 = 0, \boxed{\alpha \simeq 0.9}$$

$$g_{t+1} \leftarrow \boxed{\alpha} \cdot g_t + \boxed{(1 - \alpha)} \nabla_\theta \mathcal{L}(\theta)^2$$

$$\theta_j \leftarrow \theta_j - \epsilon \frac{\nabla_\theta \mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}}$$

$$E[g^2](t) = \beta E[g^2](t-1) + (1-\beta)\left(\frac{\partial c}{\partial w}\right)^2$$

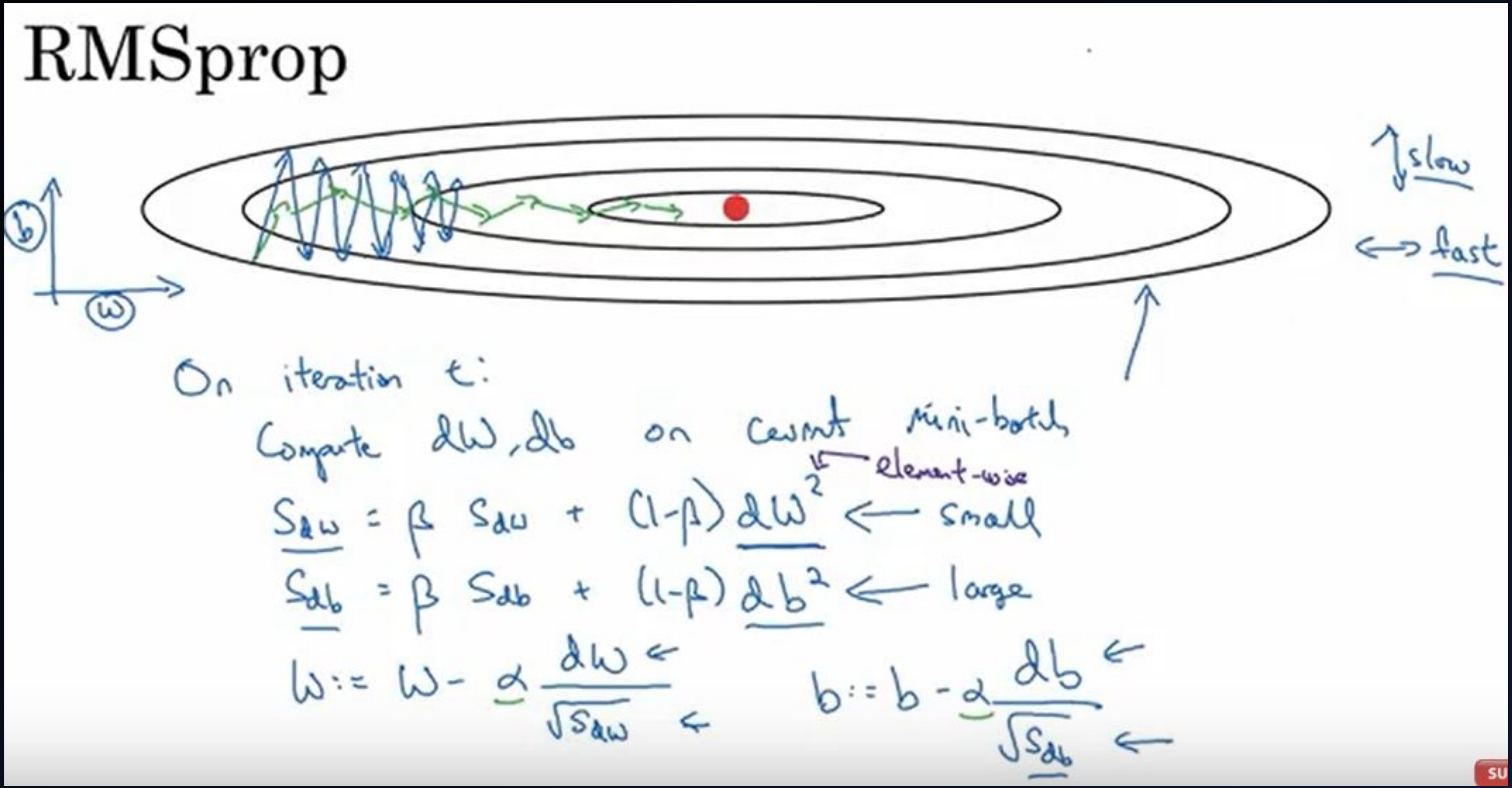
$$w_{ij}(t) = w_{ij}(t-1) - \frac{\eta}{\sqrt{E[g^2]}}\frac{\partial c}{\partial w_{ij}}$$

E[g] : is the moving average of squared gradients

is gradient of the cost function with respect to the weight

eta: is the learning rate and

beta: is moving average parameter as a decaying factor is has nothing to do with the algorithm but help to reduce the learning rate eta for the next epoch(i: iteration).



RMSprop

On iteration t:

Compute dW, db on current mini-batch

element-wise

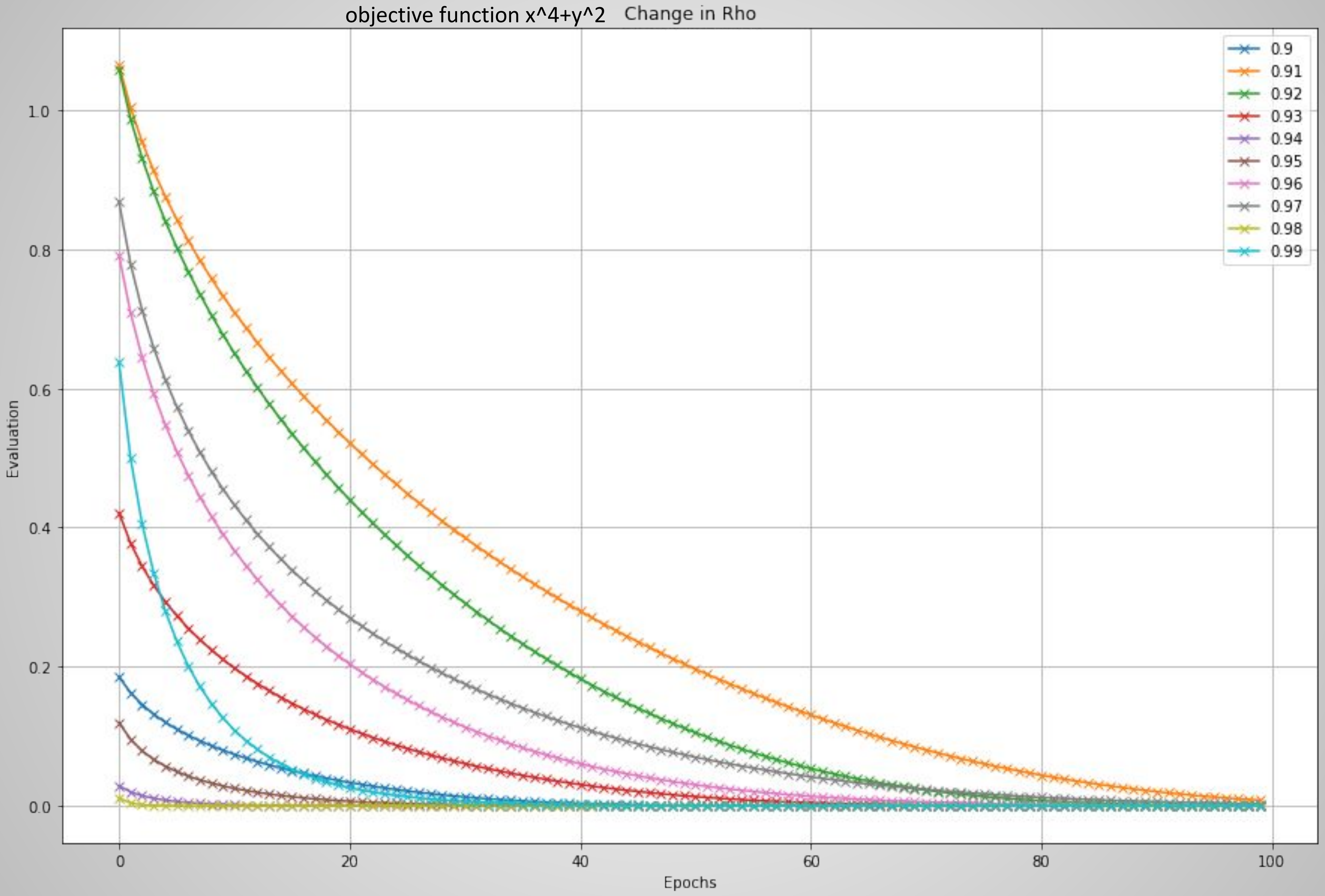
$$\underline{S_{dW}} = \beta S_{dW} + (1-\beta)\underline{dW^2} \leftarrow \text{small}$$

$$\underline{S_{db}} = \beta S_{db} + (1-\beta)\underline{db^2} \leftarrow \text{large}$$

$$W := W - \alpha \frac{dW}{\sqrt{S_{dW}}} \qquad b := b - \alpha \frac{db}{\sqrt{S_{db}}}$$

Source: DeeplearningAI. Andrew Yg. 2017.

objective function x^4+y^2



Adamax - Modifying Adam

Ephraim Hallford

Adam - Adaptive Moment

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

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

- Combines RMSprop and SGD with momentum up to this point.
- We take the running average of the mean, and take the second moment, the squares
- The Second Moment is the (squares of derivative)of the gradients to estimate the gradients as we converge on a minima. This is variance adaptive moment
- b1 is weighted hyperparameter average of momentum
- b2 is weighted for RMSprop
- purpose to dampen oscillation

Adam - Adaptive Moment

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

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

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

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

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

- theta(t+1) future value of parameterΘ_{t+1}
- theta initialized value
- v-hat - bias corrected running mean (used to avoid bias towards zero)
- m-hat - bias corrected running mean (used to avoid bias towards zero)
- decaying factor - constants
- η - a constant learning rate
- Momentum - use exponentially decaying weighted average

Adam - Adaptive Moment

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

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

- mt = running mean
- mt-1 = previous running mean
- gt - vector of partial derivatives of parameters
- gt^2 - second moment of mean is the variance

AdaMax

$$\theta_{t+1} = \theta_t - \frac{\eta}{u_t}\hat{m}_t$$

$$\begin{aligned} u_t &= \beta_2^\infty v_{t-1} + (1 - \beta_2^\infty)|g_t|^\infty \\ &= \max(\beta_2 \cdot v_{t-1}, |g_t|) \end{aligned}$$

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

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

- extension of Adam
- take the max of l norm (magnitude of the vectors)
- Adamax takes infinity norm

AdaMax

$$\theta_{t+1} = \theta_t - \frac{\eta}{u_t}\hat{m}_t$$

$$\begin{aligned} u_t &= \beta_2^\infty v_{t-1} + (1-\beta_2^\infty)|g_t|^\infty \\ &= \max(\beta_2 \cdot v_{t-1}, |g_t|) \end{aligned}$$

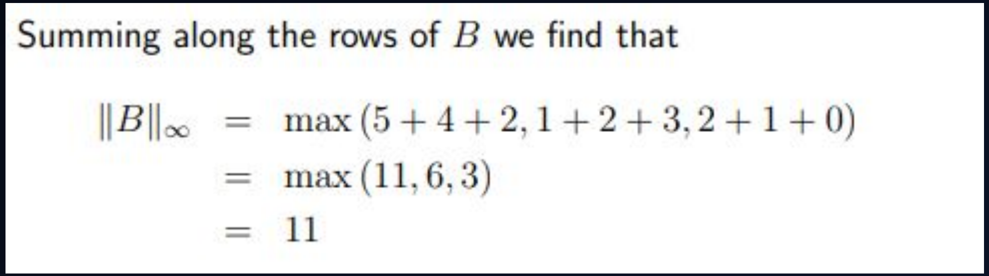
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

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

- We take the Infinity Norm of this matrix to weigh the most significant gradients converge on minima

L2 Lorm and Infinity Norm

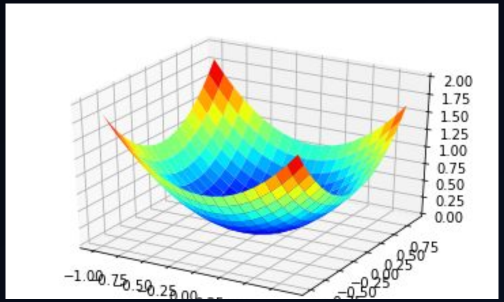
- Norm is the magnitude of the matrix. In this case the vector of the partial derivatives of the gradient at step t, iteratively over t+1
- The main difference between Adam and Adamax we take the Infinity Norm instead of L2 Norm
- Infinity norm is the maximum row sum in the Matrix(vector)

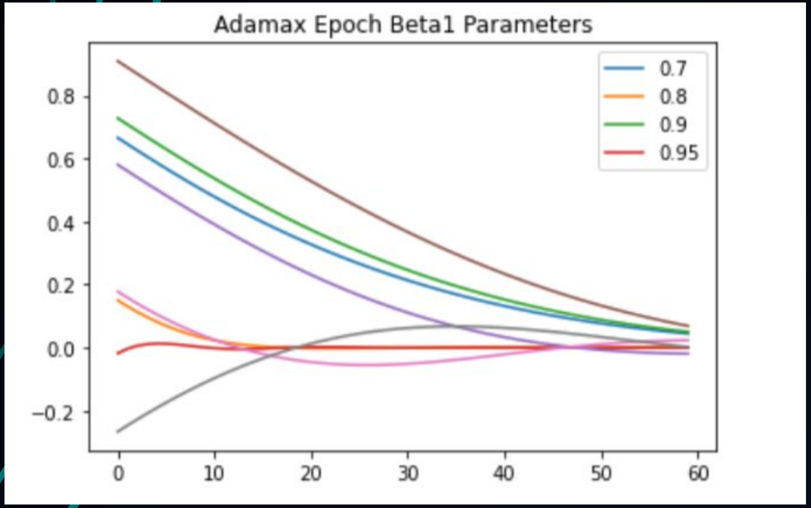


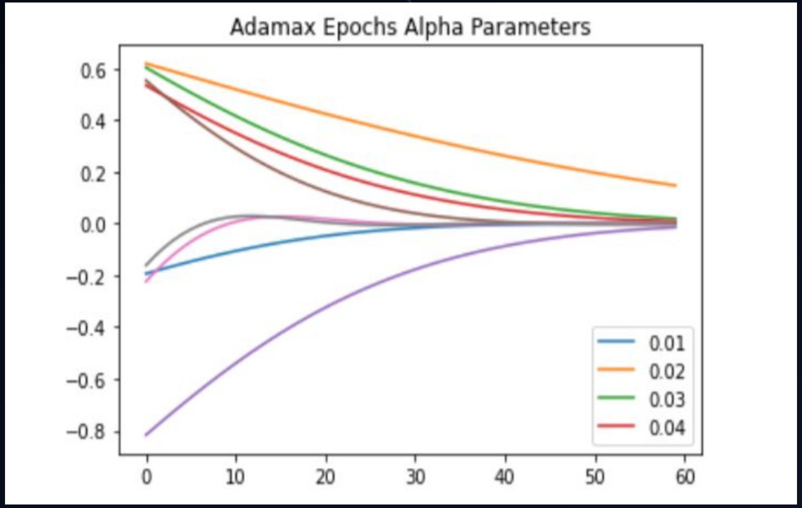
Summing along the rows of B we find that

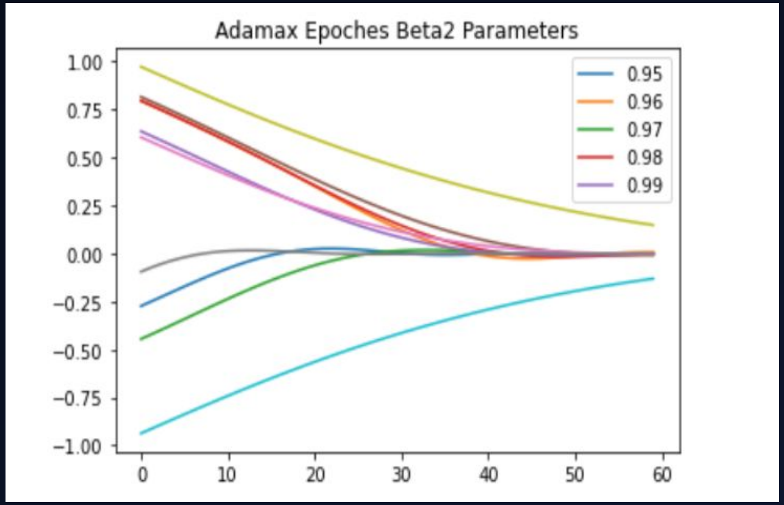
$$\begin{aligned} \|B\|_\infty &= \max(5+4+2, 1+2+3, 2+1+0) \\ &= \max(11, 6, 3) \\ &= 11 \end{aligned}$$

Performance with $x^4 + y^2$



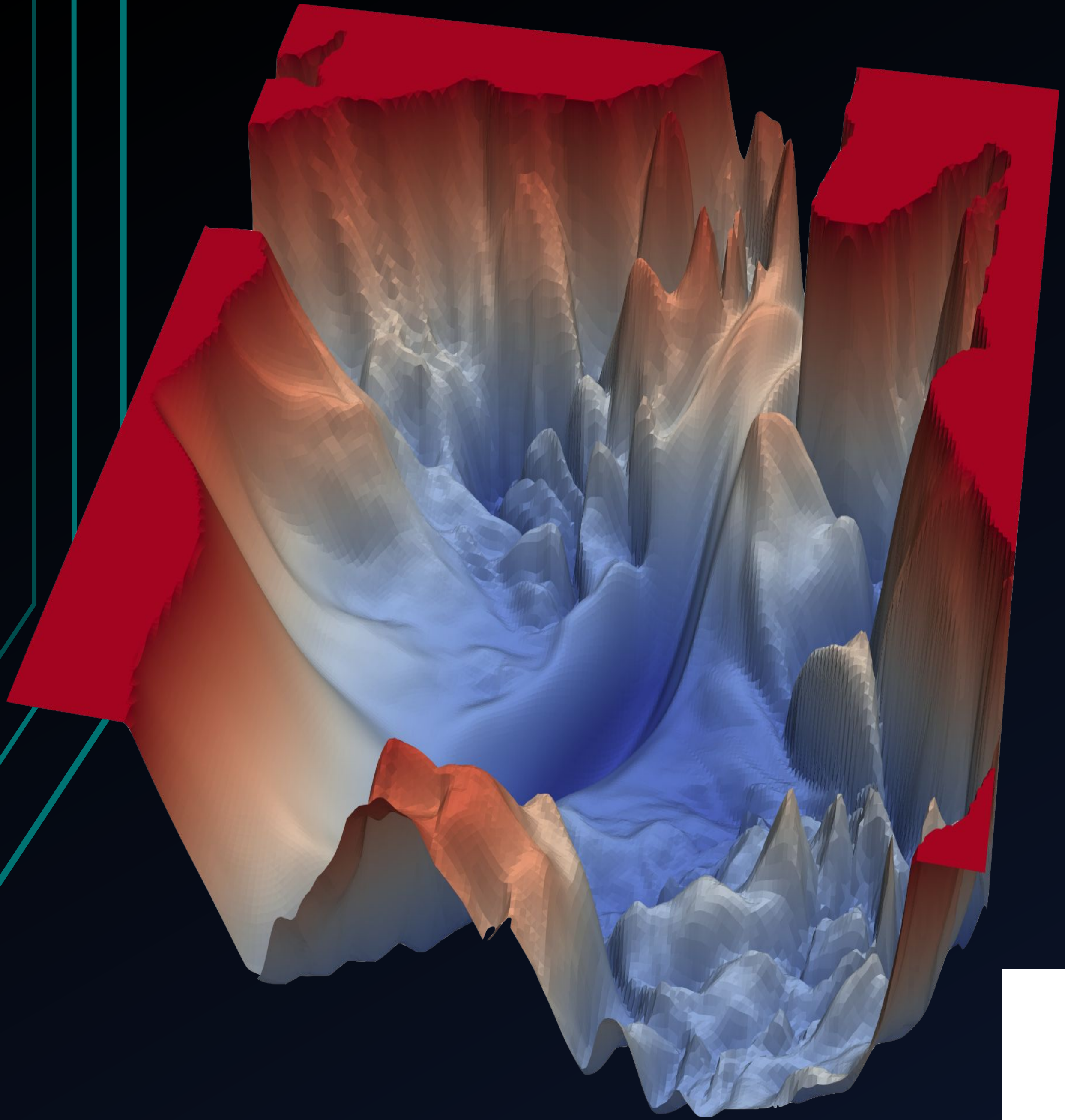






For this function, Adamax performs best when beta1 close to .95-.99, alpha > .02 (this is learning rate), beta2 close to .99

This makes sense since decaying factor must be close to 1. This is the rate at which the momentum increases at k+1 step. Alpha must be greater than .02. We can fine tune these parameters to improve performance


VG56 Neural Net architecture loss function landscape (https://www.cs.umd.edu/~tomg/projects/landscapes/)

Minimizing the loss function in a neural network performs poorly when using traditional gradient descent means due to computational expense and often it being non-Convex

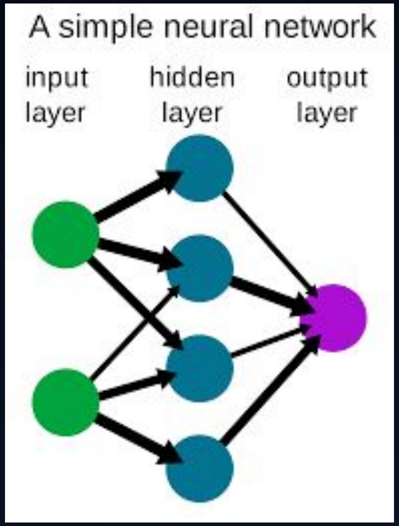
Purp

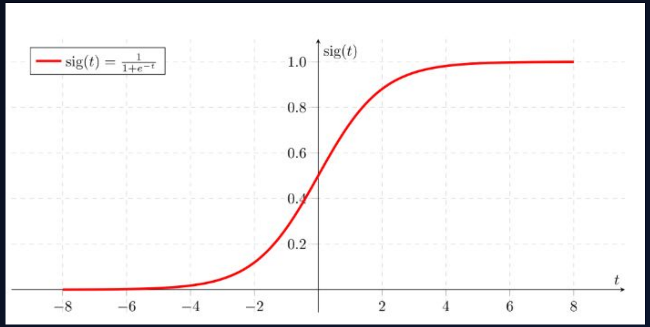
- Stochastic methods can be used to arrive at the local minima by using randomized points instead of calculating true gradient for each step in the gradient descent method
- The stochastic gradient descent for point of random variable tends to have the similar direction, albeit deviating by its variance
- SGD provides a means of optimizing these loss functions
- In the context of binary image classification, we are using binary cross entropy, where we iterate over finite sum to C where p is a pdf of predicted values and (1-p) observed is output to differentiate between binary choice (ie how much error between both)

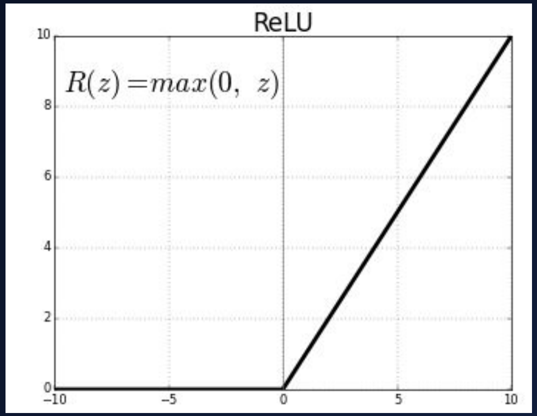
$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Neural Nets

- Inspired by biological neurons
- Employ input, hidden layers and output layer to assign weights and bias to input values to find patterns
- adds non-linearity via an activation function Most commonly used are Sigmoid and Relu (Rectified Linear).
- Minimize error between expected and observed through back propagation





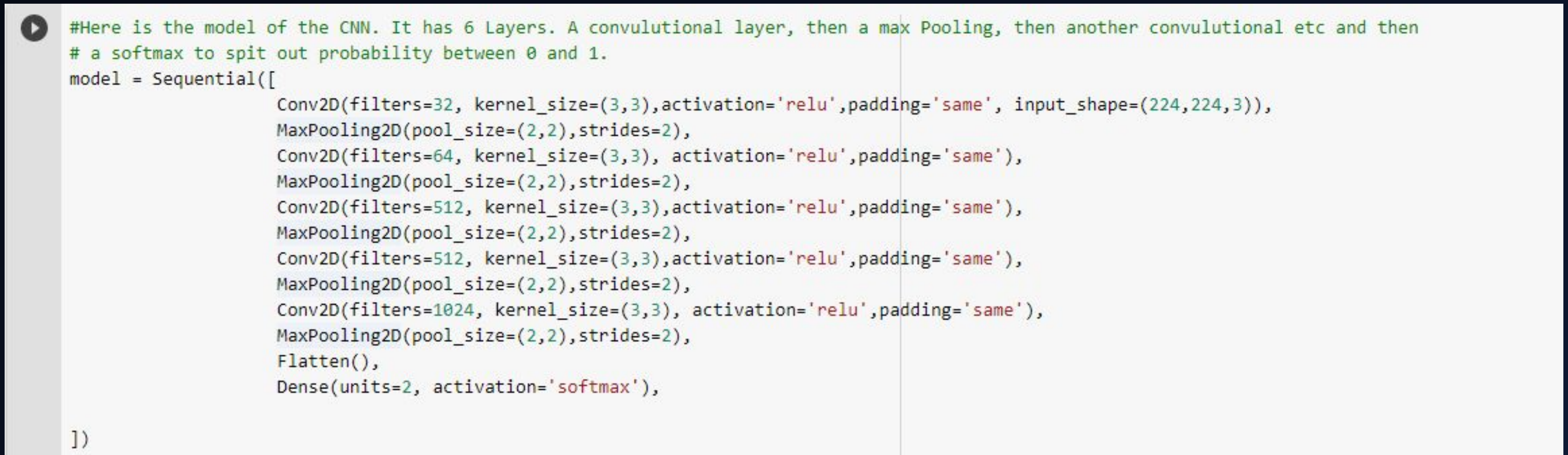


Optimization Objective for CNN

- Explore basics of neural network theory and NN architecture
- Explore Optimization of Loss Function for Finite Sum of NN
- Introduce stochastic methods that use simulation methods to minimize the loss function
- Evaluate these stochastic methods relative performance for model fitting using CNNs as a case study

Binary Classification of Kaggle Dogs and Cat Dataset

- We use CNN to classify if picture of dog is a dog, or cat etc.
- We can build CNN in Keras using Sequential() and Conv2d and either MaxPooling or AvgPooling Layers, flatten and dense layer with softmax function. Softmax provides probability between 0 and 1 the picture is either a dog or a cat as 2d array [1,0] is cat, [0,1] is dog. We use 2500 images for each class (dog or cat) to train model. Drop() can also be used to prevent overfitting.

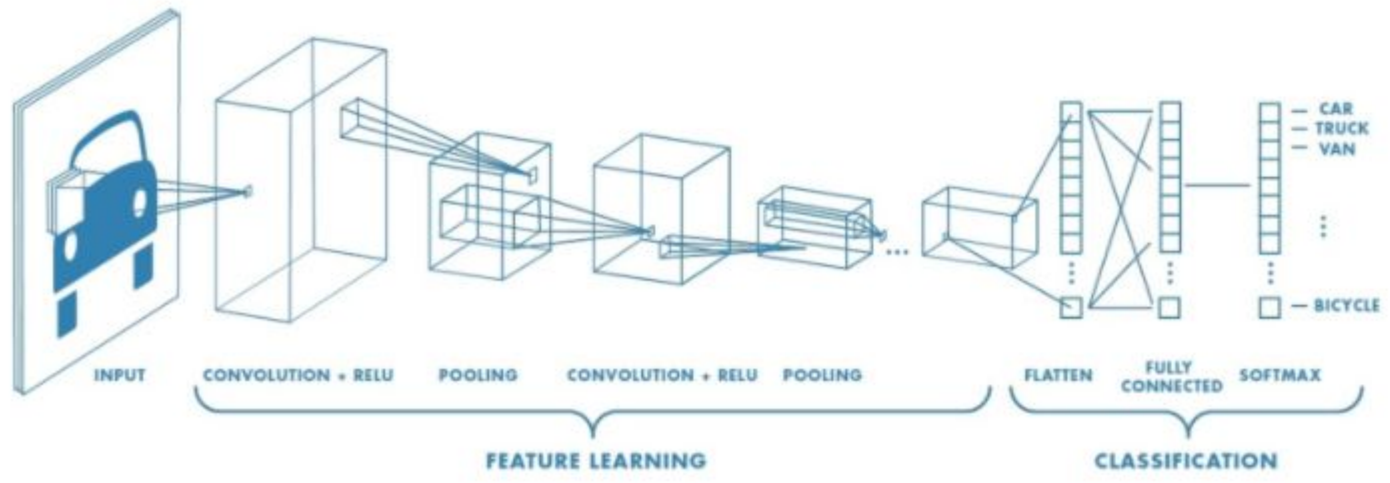


```
#Here is the model of the CNN. It has 6 Layers. A convulutional layer, then a max Pooling, then another convulutional etc and then
# a softmax to spit out probability between 0 and 1.
model = Sequential([
                    Conv2D(filters=32, kernel_size=(3,3),activation='relu',padding='same', input_shape=(224,224,3)),
                    MaxPooling2D(pool_size=(2,2),strides=2),
                    Conv2D(filters=64, kernel_size=(3,3), activation='relu',padding='same'),
                    MaxPooling2D(pool_size=(2,2),strides=2),
                    Conv2D(filters=512, kernel_size=(3,3),activation='relu',padding='same'),
                    MaxPooling2D(pool_size=(2,2),strides=2),
                    Conv2D(filters=512, kernel_size=(3,3),activation='relu',padding='same'),
                    MaxPooling2D(pool_size=(2,2),strides=2),
                    Conv2D(filters=1024, kernel_size=(3,3), activation='relu',padding='same'),
                    MaxPooling2D(pool_size=(2,2),strides=2),
                    Flatten(),
                    Dense(units=2, activation='softmax'),

])
```

CNN Layers

Layout of the Components:



The basic layers of CNN.

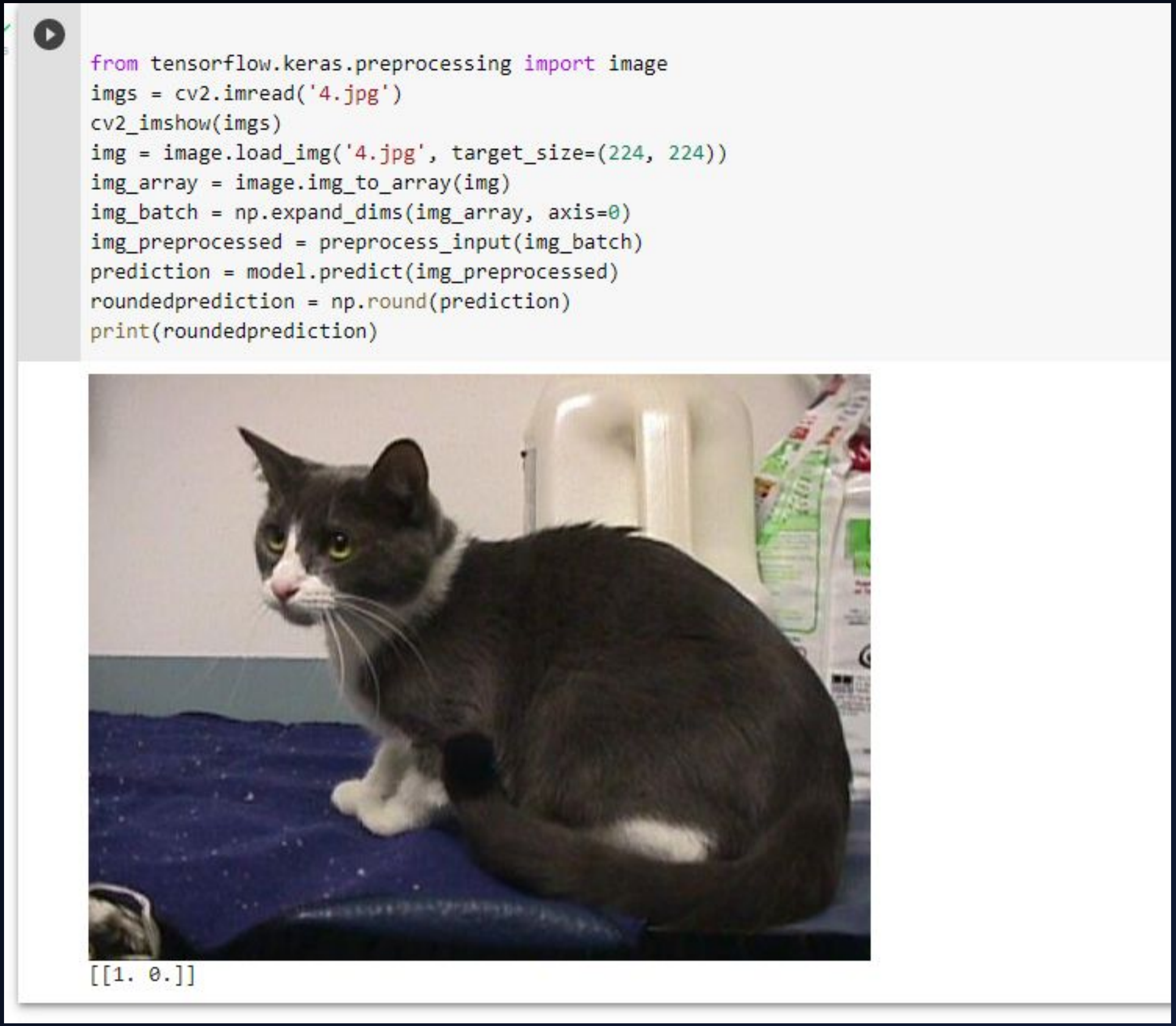
Chatterjee, H. S. (2021, August 3). *A Basic Introduction to Convolutional Neural Network*. Medium. https://medium.com/@himadrisankarchatterjee/a-basic-introduction-to-convolutional-neural-network-8e39019b27c4

Comparative Analysis of Optimizers over CNN

We use model.predict() to determine predictions in our cnn with keras

We must scale an image with OpenCV and keras.processing so it is same size to feed into CNN. We can also use a video with cv2.videocapture() or with live camera feed such as cell phone.

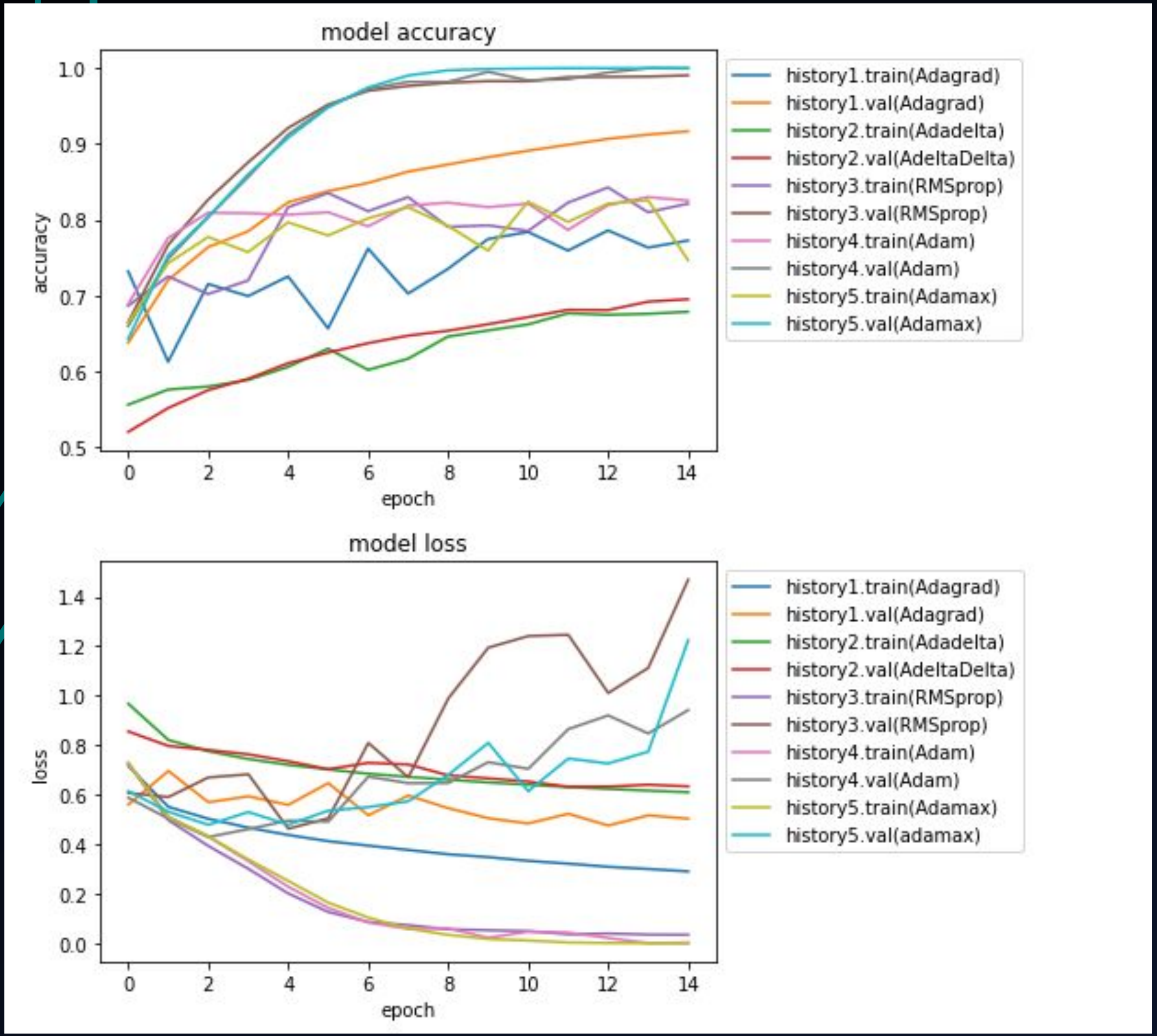
Model accurately predicts that this is a cat as 2d array [1,0]



```
from tensorflow.keras.preprocessing import image
imgs = cv2.imread('4.jpg')
cv2_imshow(imgs)
img = image.load_img('4.jpg', target_size=(224, 224))
img_array = image.img_to_array(img)
img_batch = np.expand_dims(img_array, axis=0)
img_preprocessed = preprocess_input(img_batch)
prediction = model.predict(img_preprocessed)
roundedprediction = np.round(prediction)
print(roundedprediction)
```

```
[[1. 0.]]
```

Comparative Analysis of Optimizers over CNN

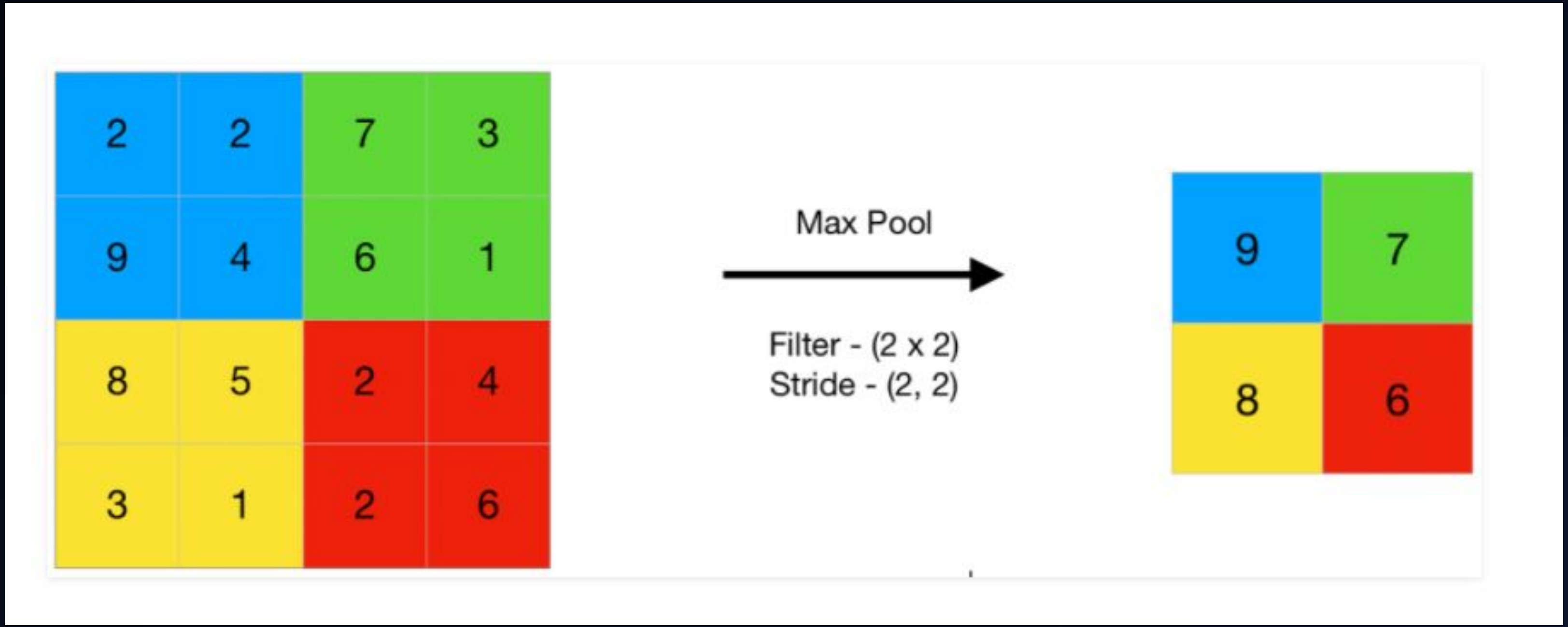


As illustrated by accuracy of Comparative analysis RMSprop, Adam, and Adamax are top performing optimizers learning rate = .0001 optimizers at default parameters

Weak Performer: Adagrad

Weakest Performer: Adadelta - needs more epochs to reach minima

Max Pooling



GeeksforGeeks. (2021, July 29). *CNN | Introduction to Pooling Layer*. https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/

Why Optimize?

Faster optimization leads to quicker results in fewer iterations. Less computational expense leads to accessibility (IoT).

Multiclass classification with n classes increase complexity and thereby increase NN architecture complexity and time to train.

Improved accuracy and reduced overfitting with simpler NN architectures allows for more rapid deployment of NN on mobile devices for computer vision, NLP, and other classification problems.

Combination of Optimization Algorithms may fit specific problems.

References

Voulodimos, A., Doulamis, N., Doulamis, A. and Protopapadakis, E., 2018. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, *2018*.

Lydia, A. and Francis, S., 2019. Adagrad—an optimizer for stochastic gradient descent. *Int. J. Inf. Comput. Sci*, *6*(5).

Zeiler, M.D., 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Mukkamala, M.C. and Hein, M., 2017, July. Variants of rmsprop and adagrad with logarithmic regret bounds. In *International Conference on Machine Learning* (pp. 2545-2553). PMLR.

Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

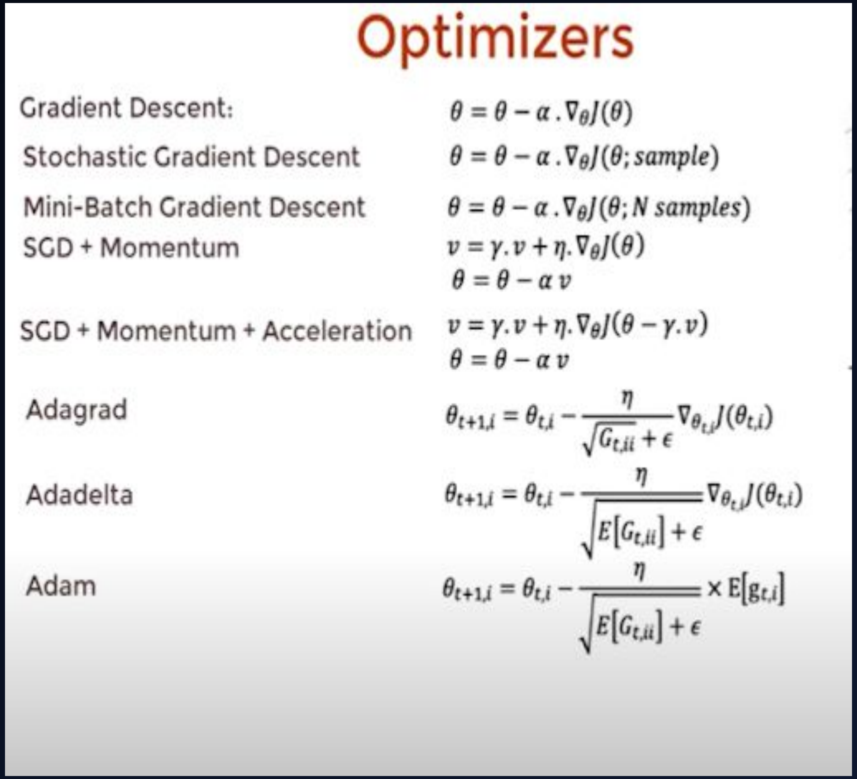
Balles, L. and Hennig, P., 2018, July. Dissecting adam: The sign, magnitude and variance of stochastic gradients. In *International Conference on Machine Learning* (pp. 404-413). PMLR.

Dhillon, A. and Verma, G.K., 2020. Convolutional neural network: a review of models, methodologies and applications to object detection. *Progress in Artificial Intelligence*, *9*(2), pp.85-112.

Thank You

The main Difference is modifying alpha, which controls learning rate.

Q & A



Optimizers

Gradient Descent:	$\theta = \theta - \alpha \,.\, \nabla_\theta J(\theta)$
Stochastic Gradient Descent	$\theta = \theta - \alpha \,.\, \nabla_\theta J(\theta; sample)$
Mini-Batch Gradient Descent	$\theta = \theta - \alpha \,.\, \nabla_\theta J(\theta; N\ samples)$
SGD + Momentum	$v = \gamma . v + \eta . \nabla_\theta J(\theta)$ $\theta = \theta - \alpha\, v$
SGD + Momentum + Acceleration	$v = \gamma . v + \eta . \nabla_\theta J(\theta - \gamma . v)$ $\theta = \theta - \alpha\, v$
Adagrad	$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \nabla_{\theta_{t,i}} J(\theta_{t,i})$
Adadelta	$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[G_{t,ii}] + \epsilon}} \nabla_{\theta_{t,i}} J(\theta_{t,i})$
Adam	$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[G_{t,ii}] + \epsilon}} \times \mathrm{E}[g_{t,i}]$