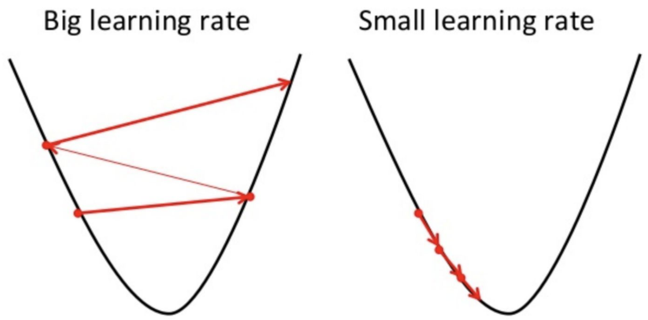
Neural Network Optimizers on Keras

By:
Vignesh Murali
Ziqing Lu
Ritvik Reddy

Optimizer determines the step we take when searching for weights

$$w_{new} = w_{old} - \text{Learning Rate} * \nabla C(w_{old})$$

Experiment Settings



- **Base Model:**

  Cifar 10 CNN Classifier

- **Optimizers:**

  SGD, SGD with Momentum, Adagrad, Adadelta, RMSprop, Adam, Adamax

- **Learning Rate:**

  0.01, 0.001, 0.0001

SGD (Stochastic Gradient Descent)

Image 2: SGD without momentum

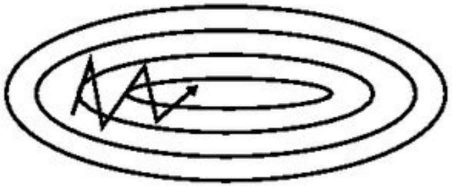


Image 3: SGD with momentum

Best Model In Experiment:

- Learning Rate: 0.01
- Accuracy: 0.85

AdaGrad (Adaptive Gradient)

Best Model In Experiment:

- Learning Rate: 0.01
- Accuracy: 0.803

Uses an adaptive learning rate for each node instead of a single learning rate for all nodes.

RMSprop

Best Model In Experiment:

- Learning Rate: 0.0001
- Accuracy: 0.738

Adaptive Learning Rate

- **Accelerate moving along more gently sloped direction**

- **Different from adagrad:**

Weigh the more recent gradient updates more than the less recent ones

Adam

Best Model In Experiment:

- Learning Rate: 0.0001
- Accuracy: 0.836

MOMENTUM + RMSprop

- **The comprehensive one:**

Applies adaptive learning rate and momentum method

Adadelta

Lets have a brief about ADAGRAD

It's a gradient based algorithm

Low Learning rates for frequently occurring weights and vice versa

For this reason, it is well-suited for dealing with sparse data. Dean et al. have found that Adagrad greatly improved the robustness of SGD and used it for training large-scale neural nets at Google versa

As Adagrad uses a different learning rate for every parameter θi, at every time step t

We are taking partial derivative => $g_{t,i} = \nabla \theta J(\theta t,i).$

Generally $\theta_{t+1,i} = \theta t,i - \eta \cdot gt,i.$

We use an extra term Gt in adgrad $\theta_{t+1,i} = \theta_{t,i} - \eta/(\sqrt{G_{t,ii}+\epsilon}) \cdot g_{t,i}$

Gt is sum of squares of all gradients w.rt to θ until time period t

Adadelta

Adadelta is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate.

The running average E[g2]t at any time only depends upon previous average and current gradients

$E[g_2]_t=\gamma E[g2]t-1+(1-\gamma)gt2.$

In SGD it is $\Delta\theta_t=-\eta\cdot g_{t,i}$

$\theta_{t+1}=\theta_t+\Delta\theta_t$

$\Delta\theta_t=-\eta/(\sqrt{E[g_2]_t+\epsilon}).g_t$

$E[\Delta\theta_2]_t=\gamma E[\Delta\theta_2]_{t-1}+(1-\gamma)\Delta\theta_{2t}$. Since $RMS[\Delta\theta]_t$ is unknown, we approximate it with the RMS of parameter updates until the previous time step

$RMS[\Delta\theta]_t=\sqrt{E[\Delta\theta_2]_t+\epsilon}$ => $\Delta\theta_t=-(RMS[\Delta\theta]_{t-1}/RMS[g_{t}])*g_t$

Adamax

The vt term in the Adam update rule scales the gradient inversely proportionally to the ℓ2 norm of the past gradients (via the vt−1 term) and current gradient |gt|2

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

We can generalize this update to the ℓ_p norm. Note that Kingma and Ba also parameterize β_2 as β_2^p:

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

Norms for large p values generally become numerically unstable, which is why ℓ_1 and ℓ_2 norms are most common in practice. However, ℓ_∞ also generally exhibits stable behavior. For this reason, the authors propose AdaMax (Kingma and Ba, 2015) and show that v_t with ℓ_∞ converges to the following more stable value. To avoid confusion with Adam, we use u_t to denote the infinity norm-constrained v_t:

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

Adamax

$\theta_{t+1} = \theta_t - (\eta / u_t) * m_t$

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

m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively

WHY Adamax Performs Better?

gt is completely ignored when it's close to zero. This means that u1, u2, ..., un are influenced by fewer gradients and this makes the algorithm less susceptible to noise in the gradients making the algorithm more robust .

Adamax

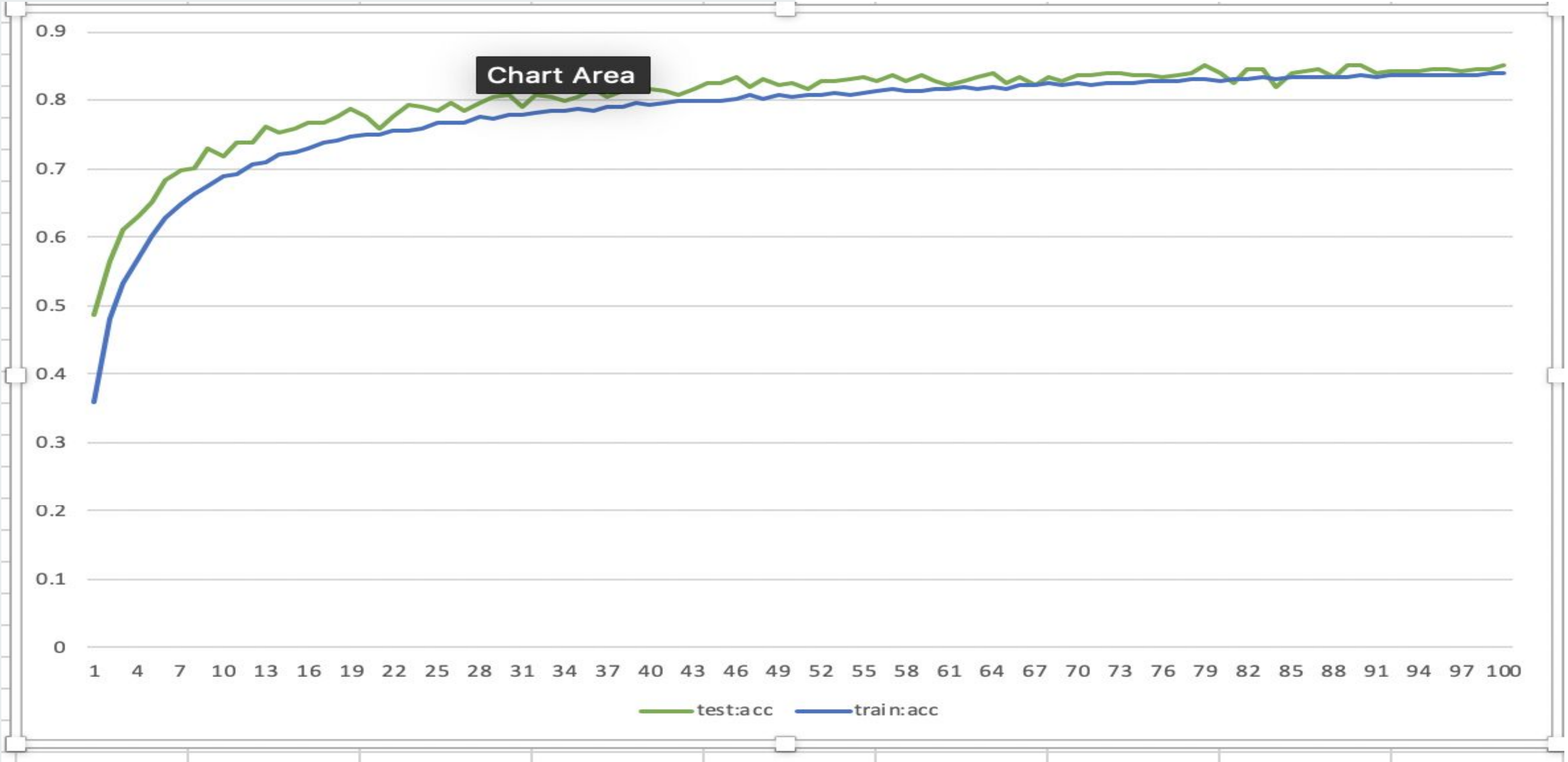
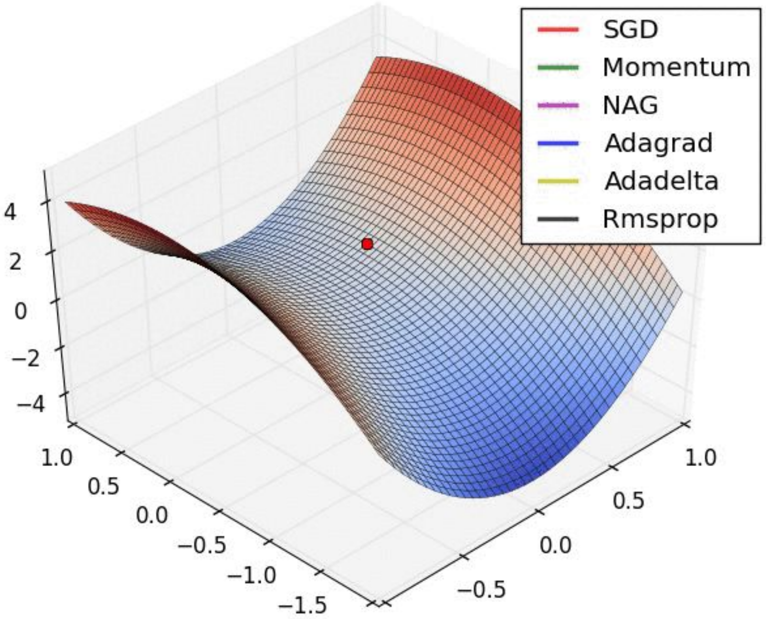


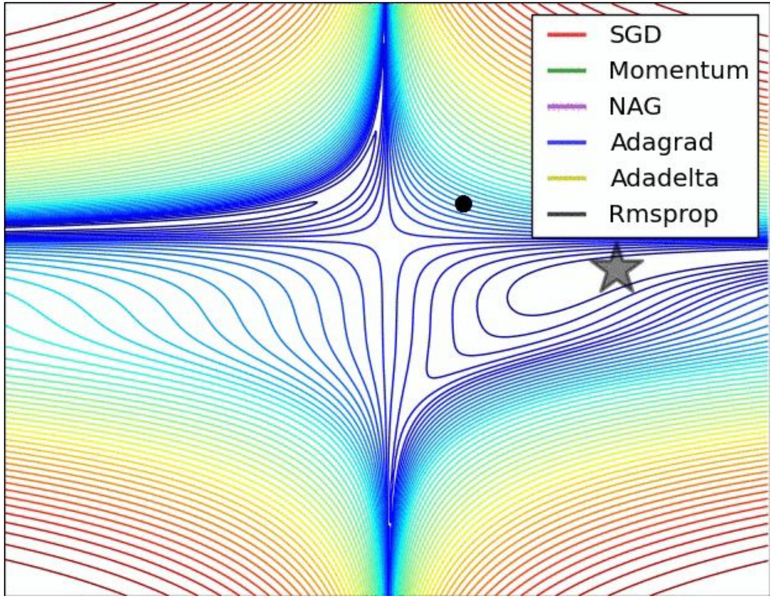
Chart Area
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91 94 97 100
test:acc
train:acc

Conclusion 1 : optimizer with adaptive learning rate and momentum outperformed others

	Optimizer	Best Model Accuracy	Learning Rate	Strategy
★	SGD	0.829	0.01	-
	SGD w/ Nesterov Momentum	0.808	0.01	Momentum
	Adagrad	0.803	0.01	Adaptive Learning Rate
	Adadelta	0.626	0.01	Adaptive Learning Rate
	RMSprop	0.738	0.0001	Adaptive Learning Rate
★	Adam	0.836	0.0001	Adaptive Learning Rate + Momentum
★	Adamax	0.851	0.001	Adaptive Learning Rate + Momentum

Optimizer performance on different surfaces





Conclusion 2: testing with different learning rate gives us direction for learning rate tuning

Optimizer	Trend Of Accuracy	Trend Of Learning Rate (0.01, 0.001, 0.0001)
SGD	↑	↑
Adagrad	↑	↑
Adadelta	↑	↑
RMSprop	↑	↓
Adam	↑	↓
Adamax	No consistent trend observed with tested learning rates	

THANK YOU!