CONVOLUTIONAL NEURAL NETWORK WITH AN OPTIMIZED BACKPROPAGATION TECHNIQUE

RMSProp and Adadelta
Course Project for Neural Networks and Fuzzy Logic, II Semester 2019-20

Paper ID: 52
Vishnu Venkatesh, 2017A3PS1154P
R. Vijay Krishna, 2017A7PS0183P
Reventh Sharma, 2017A1PS0832P

About the Paper

- Problem statement: comparing different optimisation techniques to train a CNN on COIL-100, an image dataset.
- The idea is to modify backpropagation, a standard algorithm used to train neural network, and compare performance of different optimisers.
- During backpropagation, there is a step during which weights are updated – different optimisation methods lead to different ways of updating these parameters.
- The dataset is divided in the ratio of 70:30, training to validation.
- The paper concludes that AdaDelta is the most accurate for the dataset.

AdaDelta Optimiser

- Based on AdaGrad, which is a gradient-based optimisation.
- Deals with the problem of AdaGrad’s learning rate decaying without limit, since it chooses a limited number of previous gradients to compute the expected value of squared gradients.

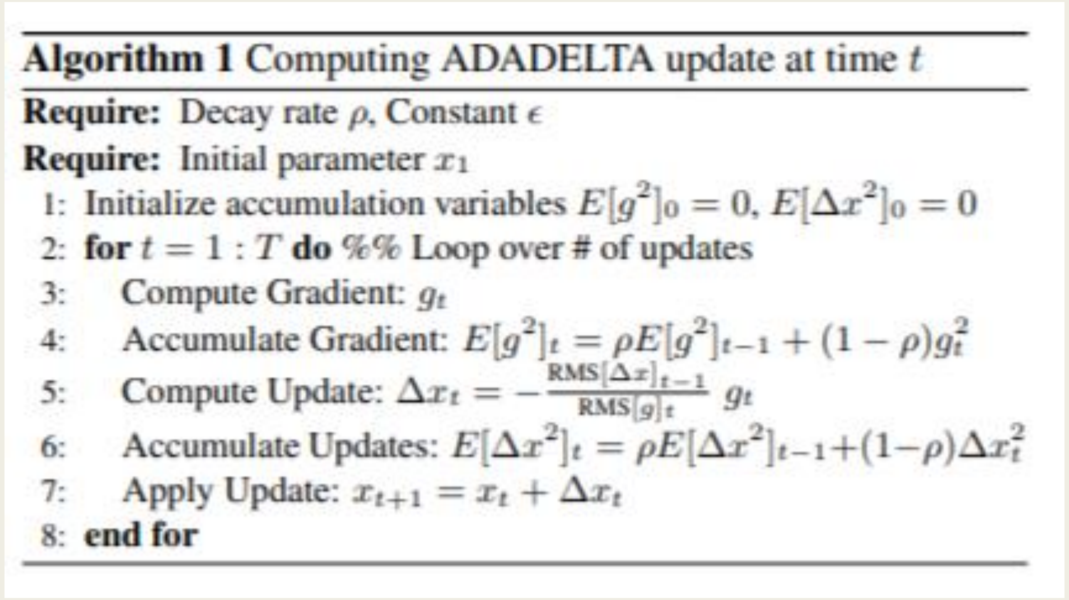
AdaDelta Optimiser (contd.)

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

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

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

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}.$$



Algorithm 1 Computing ADADELTA update at time t

```
Require: Decay rate $\rho$, Constant $\epsilon$
Require: Initial parameter $x_1$
1: Initialize accumulation variables $E[g^2]_0 = 0$, $E[\Delta x^2]_0 = 0$
2: for $t = 1 : T$ do %% Loop over # of updates
3:     Compute Gradient: $g_t$
4:     Accumulate Gradient: $E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho) g_t^2$
5:     Compute Update: $\Delta x_t = -\frac{\text{RMS}[\Delta x]_{t-1}}{\text{RMS}[g]_t} g_t$
6:     Accumulate Updates: $E[\Delta x^2]_t = \rho E[\Delta x^2]_{t-1} + (1-\rho)\Delta x_t^2$
7:     Apply Update: $x_{t+1} = x_t + \Delta x_t$
8: end for
```

RMSProp Optimiser

- Unpublished algorithm, first proposed by Geoff Hinton in lecture 6 of the online course “Neural Networks for Machine Learning”
- Similar to AdaDelta - except the learning rate (η) is fixed, not the average of previous weights like in AdaDelta.
- This constant η value addresses the issue in AdaDelta where (since AdaDelta uses a finite window to find moving RMS average) as gradient updates get smaller parameter updates also get smaller.

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

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

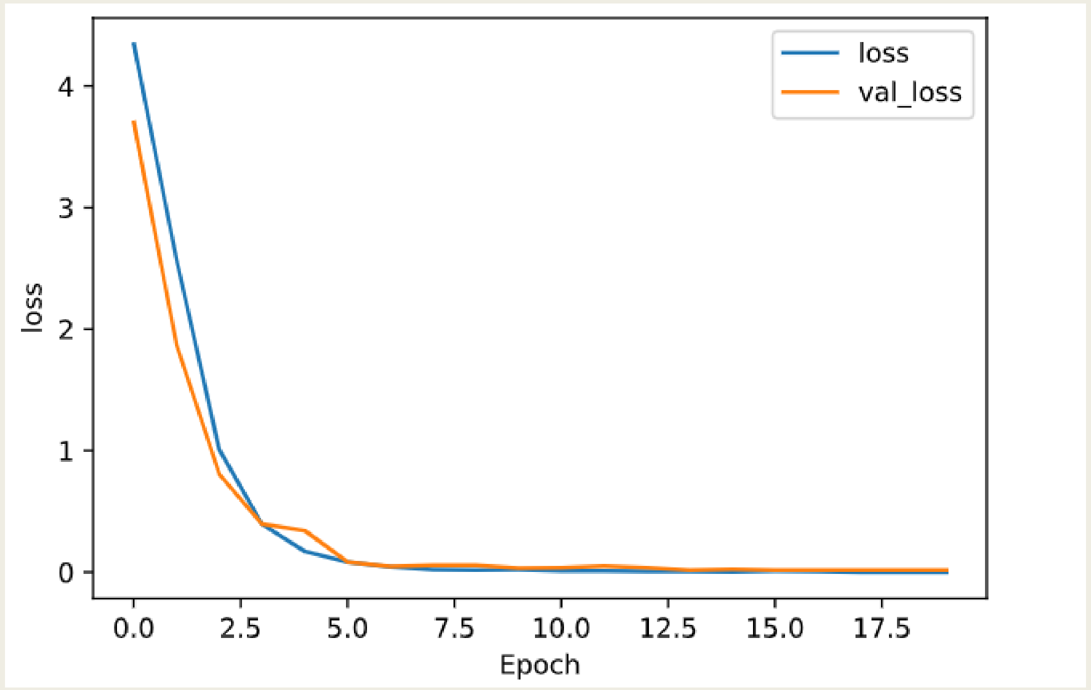
Project

- 3 conv layers, 2 pooling layers, 1 FC layer, 1 softmax layer in the network
- 7200 images in the Coil dataset – 72 color images (in different orientations, objects rotated at 5 degrees) of 100 objects.
- Code: generator, one-hot encoder, main function that initialises Keras network.
- Validation losses very close to the results obtained in paper

Code Structure

- Dataset: COIL-100, 100 images in 72 different orientations each
- Batch generator: Provides images batch-wise, which is a good practice especially if the full dataset cannot fit into memory
  - *Take images as a list*
  - *Rescales each image into required size*
  - *Populates the final image and labels mini-batch vectors in random order*
- One-hot encoder: Associates a vector with each scalar, and each vector is the same size.
- Model generation: Generating two different models with two different optimisers
- Main function: Generates a single batch of training and validation images, and fits the model on the dataset for both models

Quantitave Results: Adadelta

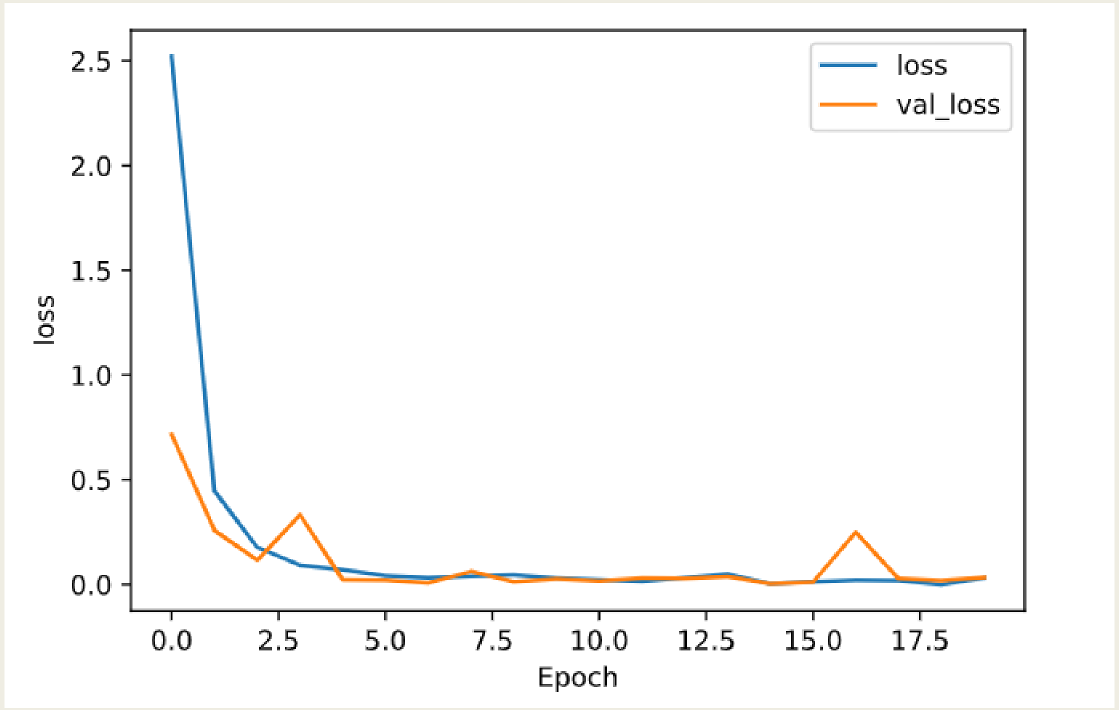


Training accuracy	1.0000
Validation loss	0.0150
Validation accuracy	0.9958

Hyperparameters (AdaDelta)

- Learning rate (η) = 0.02
- Weight for weighted average (ρ) = 0.96
- Smoothing factor (ϵ) = 1e-03

Quantitative Results: RMSProp



Training accuracy	0.9927
Validation loss	0.0335
Validation accuracy	0.9912

Hyperparameters (RMSProp)

- Learning rate (η) = 0.001
- Weight for weighted average (ρ) = 0.9
- Momentum = 0.0
- Smoothing factor (ϵ) = 1e-07

Qualitative Results

- AdaDelta fared slightly better than RMSProp, which is the conclusion reached in the paper.
- This makes sense because AdaDelta’s learning rate is adaptive while RMSProp’s is fixed.
- Accuracies achieved are very close to the ones in the paper. The difference probably comes from the fact that the authors (presumably) used the full batch with five iterations, while we ended up having to cut down on batch size (and increase number of epochs) due to RAM constraints.
- Mini-batch gradient descent is a popular technique, where the average gradient of a subset of images is computed instead of using all the images (batch gradient descent: takes up too much storage) or going image by image (stochastic gradient descent: takes too long to converge)
- Both optimisers had very similar accuracies, which means they reach the same minima.

Learning Outcomes/Takeaways

- Structure of an end-to-end machine learning project, from scratch
- Using Tensorflow to build and train neural network models
- Different optimisers and how they work (AdaDelta and RMSProp in particular, and the techniques they're built on, like AdaGrad).
- Importance of hyperparameters to the overall performance of the model