Name: **Iman Mousaei** Student Number: **98222099**
Course: **Data Science** Date: January 25, 2022
Assignment **5**

1 Theory

1.1

1.1.1 a

working with log function is easier. probability is in range [0,1]. with log, understanding them is easier.

1.1.2 b

The main intuitive difference between the L1 and L2 regularization is that L1 regularization tries to estimate the median of the data while the L2 regularization tries to estimate the mean of the data to avoid overfitting.

1.1.3 c

Momentum can accelerate training speed.
It also can jump us away from the local minima and avoid getting stuck at them, therefore resulting in higher probability of finding global minima.

1.2

1.2.1 neuron value

The notations used here are same as here.
The general formulas:
$neuron - raw - value : z_j^{(L)} = \Sigma w_{ji}^{(L)} a_i^{(L-1)} + b_j^{(L)}$
$neuron - value : a_j^{(L)} = \sigma(z_j^{(L)})$

$sigmoid : \sigma(x) = \dfrac{1}{1+e^{-x}}$

$z_1^{(1)} = 0.5 \times 0.15 + 0.1 \times 0.25 + 0.35 = 0.3825$
$a_1^{(1)} = h_1 = \sigma(z_1^{(1)}) = \dfrac{1}{1+e^{-z_1^{(1)}}} = 0.594$

We can calculate other values the same way:

$z_2^{(1)} = 0.050.20 + 0.100.30 + 0.35 = 0.39$
$a_2^{(1)} = h_2 = 0.596$

$z_1^{(2)} = 0.5940.40 + 0.5960.50 + 0.60 = 1.1356$
$a_1^{(2)} = o_1 = 0.757$

$z_1^{(2)} = 0.5940.45 + 0.5960.55 + 0.60 = 1.1951$
$a_2^{(2)} = o_2 = 0.768$

$C_0 = 0.5[(0.757 - 0.01)^2 + (0.768 - 0.99)^2) = 0.304$

1.2.2 back-propagation

for w8:

$$\frac{\partial C_0}{\partial w_8} = \frac{\partial C_0}{\partial o_2} \cdot \frac{\partial o_2}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial w_8} = (o_2 - y_1) \cdot o_2(1 - o_2) \cdot h_2 = -0.222 \cdot 0.178 \cdot 0.596 = -0.023$$

$w_8' = w_8 - \eta \dfrac{\partial C_0}{\partial w_8} = 0.55 - 0.3 \cdot (-0.023) = 0.467$

and so on for all weights...

1.3

- Try better random initialization for the weights: yes
different weights can cause in different updates for new weights. Resulting in different weights, that may not get stuck in that bad local minima.
- Try mini-batch gradient descent: yes
in mini-batch GD updates more, therefore can result in finding a good minima.
- Try using Adam: yes
Because it uses momentum, which here, can get us out of a bad local minima,
- Try initializing all the weights to zero: no
All 0 weights prevents NN from learning.
- Try tuning the learning rate α: yes
With big learning rates, we can jump out of a bad local minima.

2 Implementation

2.1 GD

Pros:
As whole data is taken at on single time it reaches global minima without any noise(for convex loss functions), but is suitable for small data sets only.
Cons:
Because this method calculates the gradient for the entire data set in one go, the calculation is very slow.
Also it needs high configuration of system to go for the whole dataset at a single time.
Batch gradient descent can converge to a local minimum for non-convex functions and it doesn't guarantee to find global minima.

2.1.1 Learning Rate(η)

Intuitively shows step size at each iteration for update towards local minimum. It means that it determines how big our jump is, for every iteration, to the direction of the minimum of loss function.

2.2 SGD

Pros:
BGD can converge to a local minimum because whole data is taken into consideration at one time, while there may be oscillation for SGD as data is updated frequently and it may jump to a better local minima(hopefully global minima).
Cons:
As SGD is updated more frequently, the cost function will have severe oscillations.

$$W_1 := W_1 - \eta * \frac{\partial Error}{\partial W_1}$$

2.2.1 Learning Rate(η)

Also here, η shows step size at each iteration for update towards local minimum.

2.3 SGD with momentum

Pros:
Faster convergence than traditional SGD.
Cons:
If the momentum is too much, we will most likely miss the local minima, rolling past it, but then rolling backwards, missing it again. If the momentum is too much, we could just swing back and forward between the local minima.
If the momentum is too low, convergence could take a while.

$$\Delta W_1{}^N = -\eta * \frac{\partial Error}{\partial W_1} + \alpha * \Delta W_1{}^{N-1}$$

$$W_1 += \Delta W_1{}^N$$

2.3.1 Learning Rate(η)

Here, η shows step size at each iteration towards local minimum considering only gradients(it doesn't make a difference in the effect of previous histories)

2.3.2 Momentum(α)

α is the momentum coefficient which is in range [0,1]. Alpha (α) determines how much we care about previous gradients for updating the new one. e.g.
In the case of $\alpha = 0$, the formula is just pure SGD.
In the case of $\alpha = 1$, the optimization process takes into account the full history of the previous update.

2.4 AdaGrad

Pros:
It adjusts the learning rate for each parameter during the learning phase based on historical information, to improve the convergence of the algorithm and its prediction accuracy. Therefore eliminating the need to manually tune the learning rate

Cons:
Learning rate diminishes in AdaGrad.
As the sum of squares of the gradients accumulates, the learning rhythm becomes slow.

$$\delta_i = \delta_{i-1} + \nabla^2\theta_i$$

$$\theta_i := \theta_{i-1} - \frac{\eta}{\sqrt{\delta_i}+\varepsilon} * \nabla\theta_i$$

2.4.1 η

It's the initial learning rate, which here, indicates how much to care about history when updating new weight.

2.5 RMSProp

Pros:
RMSProp converge faster than GD or SGD.
Also adaptive learning rate.
Any new input data points do not dramatically change the gradients, and will hence converge to the local minima with a smoother and shorter path.

Cons:
Because of its smoother path that we discussed, it takes a while to find minima.(but still faster than SGD)

$$\delta_i = \alpha * \delta_{i-1} + (1-\alpha) * \nabla^2 \theta_i$$

$$\theta_i := \theta_{i-1} - \frac{\eta}{\sqrt{\delta_i + \varepsilon}} * \nabla \theta_i$$

2.5.1 η

Also here, it's the initial learning rate, which here, indicates how much to care about history when updating new weight.

2.5.2 Decay Rate(α)

this decaying term provides faster convergence and forgets the cumulative sum of gradient history. It means we can determine how much the sum matters and how much the history in updating the histories.

2.6 Adam

Pros:
Adaptive learning rate and momentum for each parameter.
Slightly better than RMSProp(since Adam is practically RMSProp with momentum).
Easy to implement.
Computationally efficient.
Small memory requirements.
Invariant to diagonal scale change of gradients.
Very suitable for problems that are large in terms of data and/or parameters.
Suitable for problems with very noisy or sparse gradients.
Hyperparameters are intuitive to interpret and usually require little adjustment.
Learning rate does not diminish as in AdaGrad
Cons:
Does not "look ahead" like NAG

$$\delta_{M_i} = \beta_1 * \delta_{M_i} + (1 - \beta_1) * \nabla\theta_i$$

$$\delta_{V_i} = \beta_2 * \delta_{V_i} + (1 - \beta_2) * \nabla^2\theta_i$$

$$\widetilde{\delta_{M_i}} = \frac{\delta_{M_i}}{1-\beta_1}$$

$$\widetilde{\delta_{V_i}} = \frac{\delta_{V_i}}{1-\beta_2}$$

$$\theta_i := \theta_{i-1} - \frac{\eta}{\sqrt{\widetilde{\delta_{V_i}}+\varepsilon}} * \widetilde{\delta_{M_i}}$$

2.6.1 η

It's the initial learning rate, which here, indicates how much to care about **first** history when updating new weight.

2.6.2 First Momentum(β_1) and Second Momentum(β_2)

They also determine how much the history is important in updating histories, in first sum and second sum, respectively.

References

[1] https://medium.com/analytics-vidhya/l1-vs-l2-regularization-which-is-better-d01068e6658c

[2] https://www.youtube.com/watch?v=tIeHLnjs5U8

[3] https://towardsdatascience.com/neural-network-optimizers-from-scratch-in-python-af76ee087aab

[4] https://www.youtube.com/watch?v=mdKjMPmcWjY

[5] https://towardsdatascience.com/optimizers-88694509311c

[6] https://medium.com/swlh/strengths-and-weaknesses-of-optimization-algorithms-used-for-machine-learning-58926b1d69dd

[7] https://www.linkedin.com/pulse/pros-cons-some-machine-learning-algorithms-yulieth-zuluaga-g%C3%B3mez/?trk=portfolio_article-card_title

[8] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.57.5612rep=rep1type=pdf

[9] https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

[10] https://www.cs.toronto.edu/ tijmen/csc321/slides/lecture$_s lides_l ec6.pdf$

[11] https://arxiv.org/pdf/1412.6980.pdf

[12] https://www.geeksforgeeks.org/how-to-implement-a-gradient-descent-in-python-to-find-a-local-minimum/

[13] https://iopscience.iop.org/article/10.1088/1742-6596/1743/1/012002/pdf

[14] https://www.kdnuggets.com/2020/12/optimization-algorithms-neural-networks.html

[15] https://towardsdatascience.com/gradient-descent-animation-1-simple-linear-regression-e49315b24672