cs577 Assignment 3

Jorge Gonzalez Lopez
A20474413
Department of Computer Science
Illinois Institute of Technology

March 19, 2021

Part 1 (theoretical questions):

Loss:

1

L1: $L_i(\theta) = \sum_{j=1}^{k} |\hat{y}_j^{(i)} - y_j^{(i)}|$ → Better when there are outliers (lower error)

L2: $L_i(\theta) = \sum_{j=1}^{k} (\hat{y}_j^{(i)} - y_j^{(i)})^2$ → Generally better as it has analytical solution.

– Both minimize distance between known and predicted values in regression problems.

Huber: $L_\delta(d) = \begin{cases} \frac{1}{2} d^2 & \text{if } |d| \leq \delta \\ \delta(d - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$ → $L_i(\theta) = \sum_{j=1}^{k} L_\delta \quad (\hat{y}^{(i)} - y^{(i)})$

– Robust regression loss with outliers. Quadratic for small values and linear for large values.

Log-cosh: $L_i(\theta) = \sum_{j=1}^{k} \log(\cosh(\hat{y}_j^{(i)} - y_j^{(i)}))$ → $\log(\cosh(d)) \approx \begin{cases} d^2/2 & \text{if } d \text{ is small} \\ |d| - \log(2) & \text{otherwise} \end{cases}$

– Same advantages as Huber. Robust against outliers.

2 $L_i(\theta) = -\sum_{j=1}^{k} y_j^{(i)} \cdot \log(\hat{y}_j^{(i)})$

Log-likelihood → $l(\theta) = -\log L(\theta) = -\log\left(\prod_{i=1}^{m}\prod_{j=1}^{k}(p(y=j/x^{(i)}))^{y_j^{(i)}}\right) = \underbrace{-\sum_{i=1}^{m}\sum_{j=1}^{k} y_j^{(i)} \cdot \log(\hat{y}_j^{(i)})}_{\text{cross-entropy loss}}$

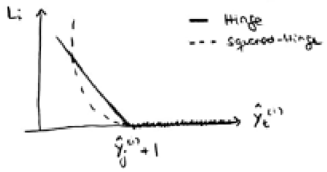
3 $\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$ → it is used as the activation function of the output layer in multi-class classification problems with cross-entropy loss.

4 Kullback-Liebler: $L_i(\theta) = -\sum_{j=1}^{k} y_j^{(i)} \cdot \log\left(\frac{y_j^{(i)}}{\hat{y}_j^{(i)}}\right)$

This function would be a measure of how alike both probability distributions are. This loss is similar to categorical cross-entropy when $y^{(i)}$ does not change.

5 Hinge: $L_i(\theta) = \sum_{j \neq t}^{k} \max(\emptyset, \hat{y}_j^{(i)} - \hat{y}_t^{(i)} + 1)$ → $\begin{cases} \text{Positive}: & \hat{y}_t^{(i)} < \hat{y}_j^{(i)} + 1 \quad (\text{incorrect}) \\ \text{Negative}: & \hat{y}_t^{(i)} > \hat{y}_j^{(i)} + 1 \quad (\text{correct}) \end{cases}$

Squared-Hinge: $L_i(\theta) = \frac{1}{2}\sum_{j \neq t}^{k} \max(\emptyset, \hat{y}_j^{(i)} - \hat{y}_t^{(i)} + 1)^2$



→ The hinge loss increases the bigger the distance between the incorrect and the correct values.

6

	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$
$\hat{y}_1$	0.5	0.4	0.3
$\hat{y}_2$	1.3	0.8	-0.6
$\hat{y}_3$	1.4	-0.4	2.7
y	1	2	3

$L_1 = \max(0, 1.3 - 0.5 + 1) + \max(0, 1.4 - 0.5 + 1) = 3.7$

$L_2 = \max(0, 0.4 - 0.8 + 1) + \max(0, -0.4 - 0.8 + 1) = 0.6$

$L_3 = \max(0, 0.3 - 2.7 + 1) + \max(0, -0.6 - 2.7 + 1) = 0$

(7) The purpose of adding a regularization term to the loss function is to get a simpler and more stable solution that will generalize better.

L1 ⇒ makes weights sparse

L2 ⇒ makes weights smaller while spreading.

To choose λ → hyper-parameter tunning (trying different values)

(8) L1: $R(\theta) = \sum_{i,j} |\theta_{i,j}|$, L2: $R(\theta) = \sum_{i,j} \theta_{i,j}^2$

⊛ $\frac{\partial R(\theta)}{\partial \theta} = \text{sign}(\theta)$ ⊛ $\frac{\partial R(\theta)}{\partial \theta} = 2\theta$

⊛ This is how they affect the gradients

(9) In Keras there are 3 different types of regularizations in a layer:

- kernel regularization: regularize and reduce w
- bias " : " " " b
- activity " : " " " $\hat{y}$ (and so w and b)

$\hat{y} = g(wx + b)$

$g(\cdot)$ ⇒ activation function

Optimization:

① Back-propagation is easy to compute and can use symbolic derivatives in python. Numerical computation is too slow but can be used for results verification.

② SGD performs the backpropagation and parameters update for every sample and GD performs the backpropagation and parameters update taking into consideration all the samples. Hence, SGD will converge faster (less time expensive) and GD will be more accurate.

③ The bigger the batch-size, the slower the learning process but the bigger the accuracy and the smaller the batch-size, the faster the learning but the smaller the accuracy.
Problems: the value of the learning rate, the loss being more sensitive to one parameter, the local minimum or saddle points and noisy gradient estimates.

④ - Local minimum / saddle: there is a velocity vector even at such locations.
- Noisy gradients: smoothed out by moving average.
- Poor conditioning: smoothed out by averaging with previous gradients.

⑤ SGD + momentum: $v^{(t+1)} = \rho v^{(t)} - \eta \nabla L(\theta^{(t)})$

NAG: $v^{(t+1)} = \rho v^{(t)} - \eta \nabla L(\theta^{(t)} + \rho v^{(t)})$

- NAG helps by indicating the momentum the "notion" of where is it going. It computes the gradients from where the last momentum was pointing.

⑥ a) Step decay: every k iterations $\rightarrow$ $\eta \leftarrow \eta/2$

b) Exponential decay: $\eta = \eta_0 \cdot e^{-k/t}$ (k: decay rate, t: iteration index)

c) Fractional decay: $\eta = \eta_0 / (1+kt)$

⑦ 1- Find x such that $f(x) = 0$, 2- start with guess x_0

3- Find update Δx such that $f(x_0 + \Delta x) = 0$

- The learning rate corresponds to the inverse Hessian matrix: $\theta^{(i+1)} = \theta^{(i)} - H^{-1} \nabla J(\theta^{(i)})$

(The Hessian matrix: $H = \nabla(\nabla J(\theta))$ and contains the second partial derivatives)

⑧ The condition number corresponds to the largest singular value divided by the smallest singular value (sv_1 / sv_m). A poor conditioning implies very high condition numbers.

(9) AdaGrad replaces the Hessian with a different preconditioner:

$$B = \operatorname{diag}\left(\sum_{j=1}^{i} \nabla J(\theta^{(j)}) \nabla J(\theta^{(j)})^T\right)^{1/2} \rightarrow \theta^{i+1} = \theta^{i} - \eta B^{(i)^{-1}} \nabla J(\theta^{(i)})$$

* Use elementwise scaling of gradients based on history of gradients

(10) In AdaGrad, because we normalized by elementwise sum of square gradients, the step size will become smaller as iterations progress.

To control this, in RMSProp a decay factor is used when adding more gradients to the gradient sum.

(11)

$$m_1^{(i+1)} = \beta_1 m_1^{(i)} + (1-\beta_1) \nabla L(\theta^{(i)})$$ → first moment: velocity with momentum.

$$m_2^{(i+1)} = \beta_2 m_2^{(i)} + (1-\beta_2) \left[\nabla L(\theta^{(i)}) \odot \nabla L(\theta^{(i)}) \right]$$ → second moment: elementwise step size

$$\theta^{(i+1)} = \theta^{(i)} - \eta\, m_1^{(i+1)} \odot 1 / \left(\sqrt{m_2^{(i+1)}} + \varepsilon\right)$$

(12) Gradient descent with a line search and bracketing:

- Given a bracket $[a, b, c]$ , choose a point $x = \frac{b+c}{2}$ so that:

if $\begin{cases} f(x) \leq f(b) & \Rightarrow [b, x, c] \\ f(x) > f(b) & \Rightarrow [a, b, x] \end{cases}$ → continue until the bracket is small enough.

An alternative would be: successive line search

- start with θ_0 and direction set $\{u^{(i)}\}$ → $\begin{cases} \eta^{(i)} = \arg\min_{\eta} f(\theta^{(i)} + \eta u^{(i)}) \\ \theta^{(i+1)} = \theta^{(i)} + \eta^{(i)} \cdot u^{(i)} \end{cases}$

(13) Quasi-Newton methods approximate the Hessian matrix inverse using gradient evaluations to reduce the complexity. They require a large number of examples.

Advantage of BFGS over Newton: the reduction of complexity due to the approximation of the Hessian inverse.

Advantage of BFGS over Adam: more accurate (approximation of the matrix over an estimation)

Disadvantage of BFGS over Adam: needs plenty of memory to store the matrices and does not work with small mini-batches (needs too many samples).

Regularization:

① With Weight Decay, each coefficient is multiplied by $p \in [0,1]$. Hence, as iterations progress, weights that are not reinforced decay to $\emptyset$. And therefore, equivalent to adding regularization term to loss function.

② Early stopping works by stop the training when validation error increases instead of when training error stop decreasing, to prevent overfitting.

To reuse validation data:

- Retrain on all data using the number of iterations determined from validation.
- Continue training from previous weights with entire data when validation loss is bigger than training loss.

③ Data augmentation consists in adding synthetic data to increase variability in training and improve generalization. To do so: augment in feature / data domain, augment by interpolating between examples / adding noise, augment by transforming data and introduce scale / illumination / rotation invariance.

④ At each training step drop out units in fully connected layers with propability of $(1-p)$, where p is a hyperparameter.

Advantages: reduce neurons interaction, reduces overfitting, increases training speed, reduces dependency of every node, distribute features over multiple nodes.

Disadvantages: longer training time due to dropout.

⑤ During testing there isn't any dropout:

- expect output from all units
- Higher total sum of outputs
- Multiply the output of each node by p (ensembled of models that shared parameters).

⑥ Batch normalization works as: $\hat{z}_j^{(i)} = \dfrac{z_j^{(i)} - \mu_j}{\sigma_j}$ → makes sure activations are not saturated; avoids all gradients having the same sign; gives equal importance to all features.

Training: adds randomness because batches are random

Testing: average normalization values computed during training

⑦ Some saturation is needed to terminate learning:

The network can learn to control BN if there is not need for it:

$$\tilde{z}_j^{(i)} = \gamma_j \hat{z}_j^{(i)} + \beta_j \qquad \left(\begin{array}{l} \gamma_j : \text{scale} \\ \beta_j : \text{shift} \end{array} \right\} \text{are learned} \Big)$$

⑧ Ensemble classifiers work by training multiple independent models and use majority vote or average during testing. (Many independent classifiers introduce randomness)

To do so:

- change data
- change parameters
- Record multiple snapshots of the model during training.