Theo Gruidroz

CS 577 AS3

1) Loss

L1: $L_i(\theta) = \sum_{j=1}^{K} \left| \hat{y}_j^{(i)} - y_j^{(i)} \right|$

Used in regression problem. Finds the absolute difference between the predicted value and the average value

L2: $L_i(\theta) = \sum_{j=1}^{K} \left(\hat{y}_j^{(i)} - y_j^{(i)} \right)^2$

Used in regression problem. Finds the squared difference between y and $\hat{y}$. Trains faster than L1 but more sensitive to outliers.

Huber: $S_\sigma(d) = \begin{cases} \frac{1}{2} d^2 & \text{if } |d| \leq \sigma \\ \sigma(d - \frac{1}{2}\sigma) & \text{o.w} \end{cases}$

$L_i(\theta) = \sum_{j=1}^{K} S_\sigma(\hat{y}^{(i)} - y^{(i)})$

Used in regression problem. Less sensitive to outliers than L2 but trains slower than L2.

Log-cosh: $L_i(\theta) = \sum_{j=1}^{K} \left(\cosh(\hat{y}^{(i)} - y^{(i)}) \right)$

Reduce sensitivity to outliers

2) First convert to probabilities: $\hat{y}_j^{(i)} = P(y=j \mid x^{(i)}) \quad j \in [1,K]$

Likelihood: $L(\theta) = \prod_{i=1}^{m} \prod_{j=1}^{K} \left(P(y=j \mid x^{(i)}) \right)^{y_j^{(i)}}$

We want to maximize likelihood therefore minimize log likelihood.

$$l(\theta) = -\log L(\theta) = -\sum_{i=1}^{m} \sum_{j=1}^{K} y_j^{(i)} \log\left(P(y=j \mid x^{(i)})\right)$$
$$= -\sum_{i=1}^{m} \sum_{j=1}^{K} y_j^{(i)} \log\left(\hat{y}_j^{(i)}\right)$$

$\therefore$ sample loss: $L_i(\theta) = -\sum_{j=1}^{K} y_j^{(i)} \log\left(\hat{y}_j^{(i)}\right)$

Probability with K classes is $1/K$

$\therefore$ worst lost for random assignment: $\log(K)$

3) For softmax loss, use softmax activation in the output layer and use cross entropy for loss.

$\text{Softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}$ → outputs vector of probabilities for each class

then, cross-entropy loss $L_i(\theta) = -\sum_{j=1}^{K} y_j^{(i)} \log(\hat{y}_j^{(i)})$

Used for multi-class classification

4) Kullback-Leibler loss divergence measure similarity between distributions:

$$L(\theta) = -\sum_{i=1}^{m} \sum_{j=1}^{k} y_j^{(i)} \log\left(\frac{y_j^{(i)}}{\hat{y}_j^{(i)}}\right)$$

Sample loss: $l_i(\theta) = \sum_{j=1}^{k} y_j^{(i)} \log\left(\frac{y_j^{(i)}}{\hat{y}_j^{(i)}}\right)$

$$KL(P||q) = \sum_{i=1}^{m} P(x_i) \log\left(\frac{P(x_i)}{q(x_i)}\right)$$

$$= \underbrace{\sum_{i=1}^{m} P(x_i) \log P(x_i)}_{-\text{entropy}} - \underbrace{\sum_{i=1}^{m} P(x_i) \log q(x_i)}_{\text{cross entropy}}$$

when $P(x_i) = y^{(i)}$ and $q(x_i) = \hat{y}^{(i)}$, there is no difference between cross-entropy or Kullback-Leibler.

5) Hinge loss gives a higher penalty as the distance from the decision boundary increases.

$$L_i(\theta) = \max(0, 1 - y^{(i)} \hat{y}^{(i)})$$

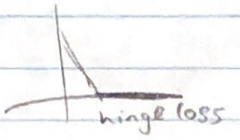
*

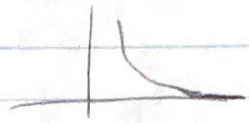
5) For categorical hinge loss, $L_i(\theta) = \sum_{j \neq t} \max(0, \hat{y}_j^{(i)} - \hat{y}_t^{(i)} + 1)$

Sum over incorrect class labels

The worst value before learning is k-1 since we sum k-1 numbers together.

square hinge loss: $L_i(\theta) = \frac{1}{2} \sum_{j=1}^{k} \max(0, \hat{y}_j^{(i)} - \hat{y}_t^{(i)} + 1)^2$





6)

	x^1	x^2	x^3	using 1 as margin
$\hat{y}_1$	0.5	0.4	0.3	$L_1 = \max(0, 1.3-0.5+1) + \max(0, 1.4-0.5+1) = 3.7$
$\hat{y}_2$	1.3	0.8	-0.6	$L_2 = \max(0, 0.4-0.8+1) + \max(0, -0.4-0.8+1) = 0.6$
$\hat{y}_3$	1.4	-0.4	2.7	$L_3 = \max(0, 0.3-2.7+1) + \max(0, -0.6-2.7+1) = 0$
y	1	2	3	

7) Regularization is used to lower the coefficients of the weights to make a more stable solution that will generalize better.

L_1 makes weights sparse (concentrate weights) while L_2 makes weights smaller while spreading them.

$L_1 \rightarrow R(\theta) = \sum_{i,j} |\theta_{i,j}|$

$L_2 \rightarrow R(\theta) = \sum_{i,j} \theta_{i,j}^2$

The larger lambda (regularization term coefficient), the more important the regularization compared to the loss term. Tune lambda to find the best value for the model. If the model overfits, increase lambda.

8) For L1, $R(\theta) = \sum |\theta_i| \Rightarrow \frac{\partial}{\partial \theta} R(\theta) = \text{sign}(\theta)$
subtract $\lambda \text{sign}(\theta)$ during gradient descent
For L2, $R(\theta) = \sum \theta_i^2 \Rightarrow \frac{\partial}{\partial \theta} R(\theta) = 2\theta$
subtract $\lambda\theta$ during gradient descent
↳ weight decay

9) $\hat{y} = \sigma(W_x + b)$
kernel regulizes W → most comon
bias regulizes b → if function expected to output small values
activity regulizes $\hat{y}$ so W and b.
if function expected to output very close to zero

Optimization

1) If the derrivative of a loss is not linear, it is hard to find an explict solution using direct computation of gradients and therefore, back propagation can be used. Back propagation is easier to compute numerical computation of gradients is used for verification on simplified network.

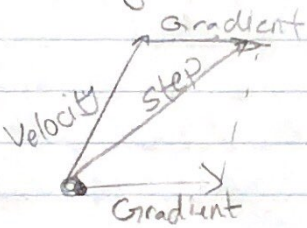
2) In gradient descent, you update the weights after processing all examples whereas in Stochastic Gradient Descent, the weights are updated after every example. SGD is expected to converge faster since the weights are updated more frequently.

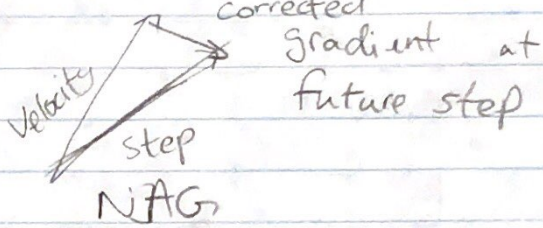
3) The smaller the batch size, the faster the process but the larger the batch size the more accurate the next weights are. ✱

3) - Choosing the learning rate
- The loss may be too sensitive to the parameters
- Could be stuck in a local minima
- Minibatch gradient descent are noisy

4) Poor conditioning: It smooths out by averaging with previous gradients
minimum/saddle point: The velocity vector at these locations helps to get out of these points
Noisy gradient: Smoothed out by moving average

5)





NAG

SGD + Momentum
NAG uses a corrected gradient to determine the next step, simple momentum only uses the actual gradient.
the accelerate term: $\beta(v^{(i+1)} - v^{(i)})$
↳ hyper parameter
Subtracting velocities

6) Step decay: every k iteration $\eta \leftarrow \eta/2$
exponential decay: $\eta = \eta_0 \cdot e^{-kt}$
fractional decay: $\eta = \eta_0 / (1+kt)$

find x such that $f(x)=0$

7) For Newton's method, start with guess x_0, update Δx such that $f(x_0+\Delta x)=0$

$$\Delta x = \frac{f(x_0)}{f'(x_0)}$$

Heissian matrix: $\nabla(\nabla J(\theta_0))$ $H = \begin{bmatrix} \frac{\partial J^2}{\partial \theta_1 \partial \theta_1} & \cdots & \frac{\partial J^2}{\partial \theta_1 \partial \theta_n} \\ \frac{\partial J^2}{\partial \theta_n \partial \theta_1} & \cdots & \frac{\partial J^2}{\partial \theta_n \partial \theta_n} \end{bmatrix}$

8) The condition number: $\frac{SV_1}{SV_m}$ ← largest singular value of Hessian matrix (SV)
← smallest SV.

When there is poor conditionning, the condition numbers are high making the problem more difficult.

9) Replace Hessiam matrix with $B^{(1)} = diag\left(\sum_{j=1}^{i} \nabla J(\theta^{(j)}) \nabla J(\theta^{(j)})^T\right)$

$$B^{(1)} = \begin{bmatrix} \sqrt{\sum \left(\frac{\partial J}{\partial \theta_1}\right)^2} & & \\ & \ddots & \\ & & \sqrt{\sum \left(\frac{\partial J}{\partial \theta_n}\right)^2} \end{bmatrix} \quad B^{(i)^{-1}} = \begin{bmatrix} \frac{1}{\sqrt{\sum \left(\frac{\partial J}{\partial \theta_1}\right)^2}} & & \\ & \ddots & \\ & & \frac{1}{\sqrt{\sum \left(\frac{\partial J}{\partial \theta_n}\right)^2}} \end{bmatrix}$$

10) The problem with Adagrad is that the step size becomes smaller as iteratives progress since we normalize by elementwise sum of square gradients. RMSProp uses a decay factor when adding new gradients to the gradient sum.

11) Adam combines RMSprop (scale by sum of gradient elements) with momentum. First moment is the velocity with momentum. Second moment is the element wise step size scale.

12) Line search: Finds the best step size instead of a fixed step size It comes with the cost of another optimization problem at each step.
Best step size: $\eta^* = \underset{\eta}{\operatorname{argmin}} f(x + \eta u)$
To solve for η^*, find the explicit computation or use gradient descent or perform simple line search.
Bracketing: less costly than gradient than gradient descent but does not guarantee optimal solution

Given bracket $[a, b, c]$
$x = \frac{b+c}{2}$
if $f(x) \leq f(b) \Rightarrow [b, x, c]$
if $f(x) > f(b) \Rightarrow [a, b, x]$
} continue with smaller bracket until the bracket is small enough

Alternative: gradient descent which is more expensive but ensures optimal solution.

13) Quasi-Newton approximate the Hessian inverse using gradient evaluation (less expensive). A BFGS algorithm is less expensive than Newton. The disadvantages of BFGS compared to Adam is that it requires a large set of examples and the advantage is that it computes the inverse cost in $O(n^2)$

Regularization

1) Weight decay is equivalent to adding regularization term to loss function. It is done by multiplying each coefficient by $\beta \in [0,1]$ and as iterations progress, weights that are not reinforced decay to 0.

2) Early stopping stops when validation error increases instad of when training error stop decreasing. This helps prevent overfitting. There are 2 strategies to reuse validation data:
   1) Retrain on all data using the number of iterations determined from validation
      step 1: How many steps to train
      step 2: Train for # steps
   2) Continue training from previous weights with entire data while validation loss is bigger than train loss.
      step 1: what is desired loss
      step 2: Train till loss is reached

3) By adding synthetic data to increase variability in training which leads to better generalization. Augment by adding features, data domain, noise, or by transforming data.

4) At each training stage, drop out units in fully connected layers with probability $(1-p)$, where p is a hyperparameter. Removed nodes are reinstated with original weights in the subsequent stage. The advantages is that it reduces co-adaptation, overfitting, dependency on a single node. It increases training speed. The disadvantage is that it causes longer training since not units are available at each st

5) By multiplying the output at each node by P. This is equivalent to computing expected value for 2^n dropped-out networks.

$$\hat{y} = E_D[f(a,D)] = \int p(D) f(x,D)\, dD$$

6) It does the same input normalization inside the network: $\hat{z}_j^{(i)} = \frac{z_j^{(i)} - \mu_j}{\sigma_j}$ (μ_j = mean, σ_j = std)

$\hat{z}_j^{(i)}$: output of j-th unit for i-th batch example.

During training, because batches are random, BN adds randomness into the training and so reduces overfitting.

7) $\{z^{(i)}\}_{i=1}^{n} \rightarrow \{\hat{z}^{(i)}\}_{i=1}^{n} \rightarrow \{\tilde{z}^{(i)}\}_{i=1}^{n}$

$\tilde{z}_j^{(i)} = \underbrace{\gamma_j}_{\text{scale}} \hat{z}_j^{(i)} + \underbrace{\beta_j}_{\text{shift}}$

γ_j and β_j, are learned. The network can learn to cancel BN if there is no need for it if $\gamma_j = \sigma_j$ and $\beta_j = \mu_j$.

They are used to help network converge better.

Good initial values: $\beta = 0$, $\gamma = 1$.

8) Ensemble classifier train multiple independent models and use majority vote or average during testing. This reduces overfitting. Possible strategies:

- Change data
- Change parameters
- Record multiple snapshots of the model during training (varying learning rate).