Funciones de costo, Optimizadores, inicialización y regularización

Física Computacional 2
Ph.D. Santiago Echeverri Arteaga

Funciones de pérdida clasificación

* **Binary Crossentropy:** $-\sum_{i=1}^{N} y_i \log_b(p_i)$
* **BinaryFocalCrossentropy:** $-\sum_{i=1}^{N} (i - p_i)^{\gamma} \log_b(p_i)$
* **Hinge Loss:** $\max(1 - y_i\hat{y}); y_i, \hat{y} \in \{-1,1\}$
* **Cosine Similarity:** $-\frac{1}{N}\sum_{i=1}^{N} \frac{y_i \cdot \hat{y}}{|y_i||\hat{y}|}$
* **Huber Loss:** $\begin{cases} \frac{(y_i - \hat{y})^2}{2} & \text{if } |y_i - \hat{y}| < \delta < 0 \\ \delta(|y_i - \hat{y}| - \frac{\delta}{2}) & \text{if } |y_i - \hat{y}| < \delta \end{cases}$
* **KLDivergence (Kullback-Leibler divergence):** $\sum_{i=1}^{N} P(i)\frac{P(i)}{Q(i)}$
* **LogCosh:** $\sum_{i=1}^{N} \log(\cosh(\hat{y} - y_i))$

Funciones de pérdida regresión

* MeanAbsoluteError: $\frac{1}{n}\sum_{i=1}^{N}|\hat{y}-y_i|$
* MeanAbsolutePercentageError: $\frac{1}{n}\frac{\sum_{i=1}^{N}|\hat{y}-y_i|}{\sum_{i=1}^{N}\hat{y}}$
* MeanSquaredError: $\frac{1}{n}\sum_{i=1}^{N}(\hat{y}-y_i)^2$
* MeanSquaredLogarithmicError: $\sqrt{\left(\log(\hat{y}+1)-\log(y_i+1)\right)^2}$
* Poisson Loss: $\left\langle \hat{y}-y_i\log(\hat{y})\right\rangle$

Regularización

* Penalizar añadiendo una función de peso en la función de pérdida (L1 y L2 las más usadas)
* Dropout: Perder aleatoriamente neuronas durante el entrenamiento
* Terminación rápida: Termina cuando alguna métrica deje de mejorar

Optimizadores: Momentum

* Cinemática tenemos que $x^{k+1} \approx x^k + v^k t$ cuando $v^{k+1} = v^k + a^k t$ para dos k y $k+1$ suficientemente cercanos.
* En gradient descent podemos tomar un "momentum" intrínseco a los pesos para que continue su estado de "movimiento" hacia otros valores hasta que se le ejerza una "fuerza contraria" (gradiente)
* Los pesos se actualizan como la posición $w^{k+1} = w^k - \eta v^{k+1}$ y las velocidades de actualización como la velocidad de una partícula donde ρ es el momentum $v^{k+1} = \rho v^k + \frac{\partial J}{\partial w}$
* Momentum ayuda a superar mínimos locales y puntos de inflexión
* Nesterov Momentum: $v^{k+1} = \rho v^k + \alpha \frac{\partial (J - \eta v^k)}{\partial w}$

Optimizadores: AdaGrad

* Escala la actualización para cada peso por separado de forma que la activación de los que más varían sea menor **(Learning rate cada vez más bajo)**
* Mientras actualiza lleva la suma de las actualizaciones previas
* $w^{k+1} = w^k - \frac{\eta}{\sqrt{G^k + \eta}} \frac{\partial J}{\partial w}$ **con** $G^k = G^{k-1} + \left(\frac{\partial J}{\partial w}\right)^2$

Optimizadores: Root Mean Squared Propagation

* Es como AdaGrad pero en lugar de usar la suma de los anteriores gradientes, se le asigna menos peso a los valores anteriores y más peso a los actuales
* Se adapta mejor a las actualizaciones nuevas (Más eficiente que AdaGrad)
* $w^{k+1} = w^k - \frac{\eta}{\sqrt{G^k + \eta}} \frac{\partial J}{\partial w}$ con $G^k = \beta G^{k-1} + (1-\beta)\left(\frac{\partial J}{\partial w}\right)^2$

Optimizadores: ADAM (Adaptative Moment Estimator)

* **AdaGrad:** Mantiene el learning rate anterior. Bueno para problemas con gradientes dispersos (e.g. Procesamiento de lenguaje natural y visión computacional).

* **Root Mean Square Propagation (RMSProp):** Mantiene el learning rate anterior pero lo adapta más fácilmente al promedio de las magnitudes de los gradientes actuales. Es bueno en problemas no estacionarios y en el manejo del ruido.

* En lugar de adaptar el learning rate basado solo en el promedio como en RMSProp, ADAM hace uso del segundo momento de los gradientes (varianza no centrada).

* Donde se definen $m^k = \beta_1 m^{k-1} + (1-\beta_1)\frac{\partial J}{\partial w}$, $v^k = \beta_2 v^{k-1} + (1-\beta_2)\frac{\partial J}{\partial w}$, $\hat{v}^k = \frac{v^k}{1-\beta_2^k}$, $\hat{m}^k = \frac{m^k}{1-\beta_1^k}$

* Los pesos se actualizan: $w^{k+1} = w^k - \frac{\eta}{\sqrt{\hat{v}^k} + \epsilon}\hat{m}^k$

Normalización por batch (por cada mini-batch)

* Hacer escalado estándar antes de las funciones de activación
* Luego hacer un mapeo afín (más parámetros de ajuste)
* Luego se pasa a la función de activación
* Para predicción se hace el escalado respecto al promedio de la población y la desviación estándar
* "Reduce el corrimiento interno de la covarianza, no necesita dropout, no necesita bias, incrementa el learning rate"

Inicializar los pesos

* Para no tener los mismos gradientes, los pesos deben ser inicializados a partir de una función de probabilidad plana, sin embargo se deben ajustar cuidadosamente los rangos.

* Default: $\left[-\frac{1}{\sqrt{L_{in}}}, \frac{1}{\sqrt{L_{in}}}\right]$ Montavon, G., Orr, G., & Müller, K. R. (Eds.). (2012). *Neural networks: tricks of the trade* (Vol. 7700). springer.

* Xavier method: $\left[-\frac{\sqrt{G}}{\sqrt{L_{in}+L_{out}}}, \frac{\sqrt{G}}{\sqrt{L_{in}+L_{out}}}\right]$ Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249-256). JMLR Workshop and Conference Proceedings.

* He Method - ReLU Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249-256). JMLR Workshop and Conference Proceedings.