Apuntes Semana 8 - 10/04/2025

Rayforth Brenes Rodriguez
Carné: 2018225352

20 de abril de 2025

1. Actualización del Proyecto

Aspectos importantes a considerar:

- La entrega del primer proyecto se ha extendido al martes 29 de abril
- La clase del 22 de abril será virtual
- En el proyecto, se puede omitir el análisis de overfitting y underfitting ya que scikit-learn no proporciona estos datos directamente
- No es necesario implementar el validation set para este proyecto

2. Funciones de Activación

Las funciones de activación son un componente crítico en las redes neuronales que introducen no linealidades necesarias para modelar relaciones complejas en los datos. A continuación se analizan las funciones más relevantes con sus características matemáticas.

2.1. Función Sigmoide

La función sigmoide es una de las funciones de activación más antiguas y ampliamente conocidas:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Características principales:

- Comprime la entrada a un rango entre 0 y 1
- Su derivada es: $\sigma'(x) = \sigma(x)(1 - \sigma(x))$
- Útil para outputs que representan probabilidades

Limitaciones:

- Sufre del problema de *vanishing gradient* en valores extremos
- Su salida no está centrada en cero, lo que puede afectar la convergencia
- Computacionalmente costosa (implica cálculo de exponenciales)

2.2. Función Tangente Hiperbólica (tanh)

La función tanh es similar a la sigmoide pero con un rango diferente:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2}$$

Características:

- Rango entre -1 y 1, centrado en el origen
- Su derivada es: $\tanh'(x) = 1 - \tanh^2(x)$
- Genera salidas con media cercana a cero, lo que ayuda en el proceso de aprendizaje

Limitaciones:

- También sufre del problema de vanishing gradient
- Mayor costo computacional que funciones modernas como ReLU

2.3. ReLU (Rectified Linear Unit)

La función ReLU ha revolucionado el entrenamiento de redes profundas por su simplicidad y eficacia:

$$\text{ReLU}(x) = \text{máx}(0, x) \tag{3}$$

Ventajas:

- Computacionalmente eficiente (solo requiere una comparación)
- No sufre de saturación para valores positivos
- Convergencia más rápida durante el entrenamiento

- Induce representaciones dispersas (muchas activaciones en cero)

Desventajas:

- Problema de "neuronas muertas": cuando las entradas son negativas, la derivada es cero y las neuronas dejan de aprender
- No acotada para valores positivos, lo que puede llevar a explosión de gradientes

2.4. Variantes de ReLU

Para abordar las limitaciones de ReLU, se han desarrollado diversas variantes:

2.4.1. Leaky ReLU

$$\text{LeakyReLU}(x) = \begin{cases} x, & \text{si } x > 0 \\ \alpha x, & \text{si } x \leq 0 \end{cases} \tag{4}$$

Donde α es un valor pequeño (típicamente 0.01) que permite un gradiente no nulo para entradas negativas.

2.4.2. Parametric ReLU (PReLU)

Similar a Leaky ReLU, pero el parámetro α es aprendido durante el entrenamiento para cada neurona.

2.4.3. ELU (Exponential Linear Unit)

$$\text{ELU}(x) = \begin{cases} x, & \text{si } x > 0 \\ \alpha(e^x - 1), & \text{si } x \leq 0 \end{cases} \tag{5}$$

Aporta gradientes suaves para valores negativos y reduce el problema de bias shift.

2.5. Función Softmax

Utilizada principalmente en la capa de salida para problemas de clasificación multiclase:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{6}$$

Características:

- Transforma un vector de números reales en una distribución de probabilidad
- La suma de todas las salidas es siempre 1
- Se usa típicamente con la función de pérdida cross-entropy

3. Backpropagation: Fundamentos Matemáticos

El algoritmo de backpropagation es esencial para el entrenamiento de redes neuronales, permitiendo calcular eficientemente los gradientes necesarios para actualizar los pesos mediante descenso por gradiente.

3.1. El Proceso General

El algoritmo consta de dos fases principales:

1. **Forward Pass**: Se calculan las activaciones de cada capa desde la entrada hasta la salida.
2. **Backward Pass**: Se propaga el error desde la salida hacia atrás, calculando gradientes para actualizar los pesos.

3.2. Nomenclatura y Definiciones

Para formalizar el proceso de backpropagation, utilizamos la siguiente notación:

- $a^{[l]}$: Vector de activaciones de la capa l
- $z^{[l]}$: Vector de pre-activaciones de la capa l
- $W^{[l]}$: Matriz de pesos que conecta la capa $l-1$ con la capa l
- $b^{[l]}$: Vector de sesgos para la capa l
- $g^{[l]}$: Función de activación en la capa l
- L: Función de pérdida

3.3. Ecuaciones Fundamentales

3.3.1. Forward Pass

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]} \tag{7}$$

$$a^{[l]} = g^{[l]}(z^{[l]}) \tag{8}$$

3.3.2. Backward Pass

Para la capa de salida L:

$$\frac{\partial L}{\partial z^{[L]}} = \frac{\partial L}{\partial a^{[L]}} \cdot g'^{[L]}(z^{[L]}) \tag{9}$$

Para las capas ocultas l:

$$\frac{\partial L}{\partial z^{[l]}} = (W^{[l+1]})^T \frac{\partial L}{\partial z^{[l+1]}} \cdot g'^{[l]}(z^{[l]}) \tag{10}$$

3.3.3. Actualización de Parámetros

$$\frac{\partial L}{\partial W^{[l]}} = \frac{\partial L}{\partial z^{[l]}}(a^{[l-1]})^T \tag{11}$$

$$\frac{\partial L}{\partial b^{[l]}} = \frac{\partial L}{\partial z^{[l]}} \tag{12}$$

3.4. Análisis de un Caso Simple

Consideremos una red neuronal con una sola neurona en cada capa y función de activación sigmoide. La función de pérdida es el Error Cuadrático Medio (MSE):

$$L = \frac{1}{2}(a^{[L]} - y)^2 \tag{13}$$

El cálculo del gradiente para actualizar los pesos sería:

$$\frac{\partial L}{\partial W^{[l]}} = (a^{[l]} - y) \cdot a^{[l]}(1 - a^{[l]}) \cdot a^{[l-1]} \tag{14}$$

3.5. Redes con Múltiples Neuronas

En redes con múltiples neuronas por capa, la notación se extiende:

- $a_j^{[l]}$: Activación de la neurona j en la capa l
- $W_{j,k}^{[l]}$: Peso que conecta la neurona k de la capa $l-1$ con la neurona j de la capa l

El cálculo de pre-activaciones se convierte en:

$$z_j^{[l]} = \sum_{k=1}^{n_{l-1}} W_{j,k}^{[l]} a_k^{[l-1]} + b_j^{[l]} \tag{15}$$

Donde n_{l-1} es el número de neuronas en la capa $l-1$.

4. Funciones de Pérdida

Las funciones de pérdida cuantifican la diferencia entre las predicciones de la red y los valores objetivo.

4.1. Error Cuadrático Medio (MSE)

Para problemas de regresión:

$$L_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{16}$$

4.2. Cross-Entropy Loss

Para problemas de clasificación binaria:

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)] \tag{17}$$

Para clasificación multiclase con softmax:

$$L_{CE} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c}) \tag{18}$$

Donde C es el número de clases y $y_{i,c}$ es 1 si el ejemplo i pertenece a la clase c y 0 en caso contrario.

5. Optimización del Entrenamiento

5.1. Descenso por Gradiente

El algoritmo básico de actualización de parámetros:

$$\theta = \theta - \alpha\nabla_\theta L \tag{19}$$

Donde α es la tasa de aprendizaje y $\nabla_\theta L$ es el gradiente de la función de pérdida respecto a los parámetros.

5.2. Técnicas de Regularización

Para evitar el sobreajuste:

- **L1 y L2 Regularización**: Añade términos de penalización a la función de pérdida basados en la magnitud de los pesos
- **Dropout**: Desactiva aleatoriamente neuronas durante el entrenamiento para prevenir codependencias
- **Batch Normalization**: Normaliza las activaciones dentro de cada lote, lo que acelera el entrenamiento y actúa como regularizador

6. Consideraciones Prácticas

6.1. Selección de Funciones de Activación

- Para **capas ocultas**: ReLU es generalmente la primera opción; si hay problemas de neuronas muertas, considerar variantes como Leaky ReLU
- Para **capas de salida**:
  - Clasificación binaria: Sigmoide

    - Clasificación multiclase: Softmax
    - Regresión: Lineal (sin activación)
- Para **redes recurrentes**: TanH o GRU/LSTM con sus propias funciones de activación internas

6.2. Inicialización de Pesos

La inicialización adecuada de pesos es crucial para el entrenamiento eficiente:

- **Xavier/Glorot**: Diseñada para funciones de activación con derivadas cercanas a 1 (como sigmoide y tanh)
- **He**: Optimizada para funciones ReLU y sus variantes

7. Caso de Estudio: MNIST

El conjunto de datos MNIST es un ejemplo típico para ilustrar conceptos de redes neuronales:

- Contiene 60,000 imágenes de entrenamiento y 10,000 de prueba de dígitos escritos a mano
- Cada imagen es de 28×28 píxeles, lo que resulta en 784 features
- El objetivo es clasificar correctamente los dígitos del 0 al 9

7.1. Arquitectura Típica

Una arquitectura básica para MNIST podría ser:

- **Capa de entrada**: 784 neuronas (una por píxel)
- **Capas ocultas**: 1-2 capas con 128-512 neuronas cada una, con activación ReLU
- **Capa de salida**: 10 neuronas con activación softmax (una por dígito)