Arquitectura de red neuronal elegida:

Red neuronal multicapa (multilayer perceptron - MLP) para clasificación de dígitos MNIST:

Entrada: 784 neuronas (28x28) píxeles aplanados

↓

Capa oculta 1: 128 neuronas (activación ReLU)

↓

Capa oculta 2: 64 neuronas (activación ReLU)

↓

Capa de salida: 10 neuronas (activación Softmax)

Notación matemática:

Símbolo	Descripción	Valores en la red
L	número total de capas	$L = 3$
$n^{[l]}$	número de neuronas en capa l	$n^{[0]}=784, n^{[1]}=128, n^{[2]}=64, n^{[3]}=10$
$W^{[l]}$	matriz de pesos de capa l	Dimensiones $(n^{[l]}, n^{[l-1]})$

Símbolo	Descripción	Valores en la red
$b^{[l]}$	Vector de sesgos de capa l	Dimensiones $(n^{[l]}, 1)$
$a^{[l]}$	Activaciones de capa l	Salida de la capa
$z^{[l]}$	Pre-activaciones de capa l	Antes de función de activación
m	Tamaño del batch	Número de ejemplos
α	Tasa de aprendizaje	Hiperparámetro

1 Forward pass (propagación hacia adelante)

Capa de entrada

$a^{[0]} = x$ * asigna los datos de entrada, como las activaciones de la capa 0

Donde $x \in \mathbb{R}^{784}$ es el vector de entrada (píxeles normalizados entre 0 y 1)

Capas ocultas ($L = 1, 2$)

Transformación lineal:

$$z^{[L]} = W^{[L]} \cdot a^{[L-1]} + b^{[L]}$$

* Calcula una combinación lineal pesada de la activación de la capa anterior.
Cada neurona suma las entradas multiplicadas por sus pesos, más sesgos.

Activación ReLU:

$$a^{[L]} = g^{[L]}(z^{[L]}) = ReLU(z^{[L]}) = \max(0, z^{[L]})$$

$$ReLU(z_i) = \begin{cases} z_i & \text{si } z_i > 0 \\ 0 & \text{si } z_i \leq 0 \end{cases}$$

* Introduce no-linealidad en la red, mantiene valores positivos y convierte negativos a cero
Esto permite a la red aprender patrones complejos

3

Específicamente por capa:

Capa 1: $z^{[1]} = W^{[1]} \cdot a^{[0]} + b^{[1]} \in \mathbb{R}^{128 \times 1}$ * Transforma los 784 pixeles de entrada en 128 valores usando 100,352 pesos (128x784) más 128 sesgos

$a^{[1]} = ReLU(z^{[1]}) \in \mathbb{R}^{128 \times 1}$ * aplica Relu elemento por elemento creando 128 características de nivel medio que detectan patrones básicos (bordes, curvas)

Capa 2: $z^{[2]} = W^{[2]} \cdot a^{[1]} + b^{[2]} \in \mathbb{R}^{64 \times 1}$ * activa las 64 neuronas que representan patrones de alto nivel (formas complejas de dígitos)

4

Capa de salida ($L=3$)

Transformación lineal:

$$z^{[3]} = w^{[3]} \cdot a^{[2]} + b^{[3]} \in \mathbb{R}^{10\times 1}$$

* Genera 10 valores sin procesar (logits) uno por cada dígito (0-9), usando 640 pesos (10x64) más 10 sesgos.

Activación Softmax:

$$a^{[3]} = \text{Softmax}(z^{[3]})$$

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{10} e^{z_j}}$$

* Convierte los 10 dígitos en probabilidades que suman 1, valores altos se simplifican, bajos se reducen. Cada salida representa la confianza de que la entrada sea ese dígito

Propiedades

- $a_i^{[3]} = P(y = \text{clase}_i \mid x)$ – probabilidad de que la entrada sea clase i
- $\sum_{i=1}^{10} a_i^{[3]} = 1$ – Las probabilidades suman exactamente 1
- $0 \leq a_i^{[3]} \leq 1$ – Cada probabilidad está entre 0 y 1

Predicción

$\hat{y} = a^{[3]}$, Clase predicha = $\arg\max(\hat{y})$ * la predicción final es el índice con mayor probabilidad

ejemplo: si $a^{[3]} = [0.01, 0.02, 0.90, 0.03, \ldots]$ (dígito "2").

Vectorización para batch de m ejemplos:

$$Z^{[L]} = W^{[L]} \cdot A^{[L-1]} + b^{[L]} \quad A^{[L]} = g^{[L]}(Z^{[L]})$$

* procesar múltiples ejemplos simultáneamente para eficiencia computacional. Cada columna de $A^{[L]}$ es un ejemplo diferente

Donde $A^{[L]}, Z^{[L]} \in \mathbb{R}^{n^{[L]} \times m}$

(matriz donde cada columna es un ejemplo)

2. función de error (loss function)

Cross-entropy loss para un ejemplo:

$$L(\hat{y}, y) = -\sum_{i=1}^{10} y_i \cdot \log(\hat{y}_i)$$

* Mide que tan incorrecta es la predicción. Penaliza fuertemente predicciones erróneas con alta confianza

Como y es one-hot encoding (ej: $[0,0,1,0,0,0,0,0,0,0]$ para el dígito 2) se simplifica a:

$$L(\hat{y}, y) = -\log(\hat{y}_c)$$

donde c es la clase verdadera

Interpretación:

- Si $\hat{y}_c = 0.9$ (predicción correcta): $L = -\log(0.9) = 0.105$ (error bajo)
- Si $\hat{y}_c = 0.1$ (predicción incorrecta): $L = -\log(0.1) = 2.303$ (error alto)
- Si $\hat{y}_c \to 1$: $L \to 0$. (perfecto)
- Si $\hat{y}_c \to 0$: $L \to \infty$ (pésimo)

7

Cost función para batch de m ejemplos:

$$J(w,b) = \frac{1}{m} \sum_{j=1}^{m} L(\hat{y}^{(j)}, y^{(j)})$$

$$J(w,b) = -\frac{1}{2m} \sum_{j=1}^{m} \sum_{i=1}^{10} y_i^{(j)} \cdot \log(\hat{y}_i^{(j)})$$

* promedia el error sobre todos los ejemplos del batch. es la métrica que queremos minimizar durante el entrenamiento

<u>Con regularización L_2:</u>

$$J_{reg}(w,b) = J(w,b) + \frac{\lambda}{2m} \sum_{l=1}^{L} \|w^{[l]}\|_F^2$$

$$\|w^{[l]}\|_F^2 = \sum_i \sum_j (w_{ij}^{[l]})^2$$

* añade una penalización por pesos grandes para prevenir overfitting. Fuerza a la red a usar pesos mas pequeños y distribuidos; mejorando la generalización

donde λ es el parámetro de regularización (típicamente 0.001 a 0.1)

8

3. back propagation (propagación hacia atras)

Notación de gradientes:

Símbolo	significado
$dz^{[l]}$	$\frac{\partial \mathcal{L}}{\partial z^{[l]}}$
$dw^{[l]}$	$\frac{\partial \mathcal{L}}{\partial w^{[l]}}$
$db^{[l]}$	$\frac{\partial \mathcal{L}}{\partial b^{[l]}}$
$dA^{[l]}$	$\frac{\partial \mathcal{L}}{\partial A^{[l]}}$

capa de salida ($l=3$):

para Softmax + Cross-entropy

$$dz^{[3]} = a^{[3]} - y = \hat{y} - y$$
$$dw^{[3]} = dz^{[3]} \cdot (a^{[2]})^T$$
$$db^{[3]} = dz^{[3]}$$

9

capas ocultas ($l=2,1$):

gradiente respecto a activaciones previas

$$dA^{[l]} = (w^{[l+1]})^T \cdot dz^{[l+1]}$$

gradiente respecto a pre-activaciones (con derivada de ReLU)

$$dz^{[l]} = dA^{[l]} \odot g'^{[l]}(z^{[l]})$$

$$g'^{[l]}(z^{[l]}) = ReLU'(z^{[l]}) = \begin{cases} 1 & \text{si } z^{[l]} > 0 \\ 0 & \text{si } z^{[l]} \leq 0 \end{cases}$$

gradientes de parámetros

$$dw^{[l]} = dz^{[l]} \cdot (a^{[l-1]})^T$$
$$db^{[l]} = dz^{[l]}$$

10

especificación por capa:

Capa 3:

$$dz^{[3]} = a^{[3]} - y \in \mathbb{R}^{10\times 1}$$
$$dw^{[3]} = dz^{[3]} \cdot (a^{[2]})^T \in \mathbb{R}^{10\times 64}$$
$$db^{[3]} = dz^{[3]} \in \mathbb{R}^{10\times 1}$$

Capa 2:

$$dA^{[2]} = (w^{[3]})^T \cdot dz^{[3]} \in \mathbb{R}^{64\times 1}$$
$$dz^{[2]} = dA^{[2]} \odot \text{ReLU}'(z^{[2]}) \in \mathbb{R}^{64\times 1}$$
$$dw^{[2]} = dz^{[2]} \cdot (a^{[1]})^T \in \mathbb{R}^{64\times 128}$$
$$db^{[2]} = dz^{[2]} \in \mathbb{R}^{64\times 1}$$

Capa 1:

$$dA^{[1]} = (w^{[2]})^T \cdot dz^{[2]} \in \mathbb{R}^{128\times 1}$$
$$dz^{[1]} = dA^{[1]} \odot \text{ReLU}'(z^{[1]}) \in \mathbb{R}^{128\times 1}$$
$$dw^{[1]} = dz^{[1]} \cdot (a^{[0]})^T \in \mathbb{R}^{128\times 784}$$
$$db^{[1]} = dz^{[1]} \in \mathbb{R}^{128\times 1}$$

para batch de m ejemplos

$$dw^{[L]} = \frac{1}{m} \cdot dz^{[L]} \cdot \left(A^{[L-1]}\right)^T$$
$$db^{[L]} = \frac{1}{m} \sum (\text{suma por filas de } dz^{[L]})$$

Con regularización L2:

$$dw^{[L]} = \frac{1}{m} dz^{[L]} \cdot (A^{[L-1]})^T + \frac{\lambda}{m} \cdot w^{[L]}$$

4. descenso por gradiente (gradient descent)

Ecuaciones de actualización:

Para cada capa $l = 1, 2, 3$:

$$w^{[l]} := w^{[l]} - \alpha \cdot dw^{[l]}$$
$$b^{[l]} := b^{[l]} - \alpha \cdot db^{[l]}$$

donde α es la tasa de aprendizaje (típicamente entre 0.001 y 0.1)

Variantes:

1. Batch gradient descent
usa todo el dataset en cada iteración
2. Stochastic gradient descent (SGD)
Actualiza con un ejemplo a la vez
3. Mini-batch gradien descent
usa batches pequeños (32, 64, 128 ejemplos) más comun

Algoritmo completo:

Inicialización:

$$w^{[l]} \sim N(0, \sigma^2),\ b^{[l]} = 0$$

Para cada época:

Para cada mini-batch (X_{batch}, Y_{batch}):

1. forward pass:

$A^{[0]} = X_{batch}$

Para $l = 1$ hasta L:

$Z^{[l]} = w^{[l]} \cdot A^{[l-1]} + b^{[l]}$

$A^{[l]} = g^{[l]}(Z^{[l]})$

2. Calcular loss:

$$J = -\frac{1}{m} \sum_i \sum_j Y_{ij} \cdot \log(A_{ij}^{[L]})$$

3. Backward pass:

$dz^{[L]} = A^{[L]} - Y$

para $L = L$ hasta 1 (reversa):

$dw^{[L]} = \frac{1}{m} \cdot dz^{[L]} \cdot (A^{[L-1]})^T$

$db^{[L]} = \frac{1}{m} \sum dz^{[L]}$

si $L > 1$:

$dA^{[L-1]} = (w^{[L]})^T \cdot dz^{[L]}$

$dz^{[L-1]} = dA^{[L-1]} \odot g'^{[L-1]}(z^{[L-1]})$

4. Actualización

Para $L = 1$ hasta L:

$w^{[L]} := w^{[L]} - \alpha \cdot dw^{[L]}$

$b^{[L]} := b^{[L]} - \alpha \cdot db^{[L]}$

Optimizadores Avanzados:

momentum:

$$V_{dw}^{[L]} := \beta \cdot V_{dw}^{[L]} + (1-\beta) \cdot dw^{[L]}$$

$$V_{db}^{[L]} := \beta \cdot V_{db}^{[L]} + (1-\beta) \cdot db^{[L]}$$

$$W^{[L]} := w^{[L]} - \alpha \cdot V_{dw}^{[L]}$$

$$b^{[L]} := b^{[L]} - \alpha \cdot V_{db}^{[L]}$$

donde $\beta \approx 0.9$ Controla cuánto "momentum" retener.

* acumula un promedio móvil de gradientes pasados.
Crea "inercia" que acelera el aprendizaje en direcciones consistentes y amortigua oscilaciones

* actualiza los pesos usando la velocidad acumulada en lugar del gradiente directo, Cruza mesetas más rápido

Primer momento (momentum): $m_{dw} := \beta_1 \cdot m_{dw} + (1-\beta_1)\, dw$

* Promedio móvil exponencial del gradiente

$m_{db} := \beta_1 \cdot m_{db} + (1-\beta_1) \cdot db$

segundo momento (RMS prop): $\beta_2 \cdot V_{dw} + (1-\beta_2) \cdot dw^2$

* promedio móvil exponencial del gradiente al cuadrado (magnitud). adapta la tasa de aprendizaje por parámetro

$V_{db} := \beta_2 \cdot v_{db} + (1-\beta_2) \cdot db^2$

Corrección de bias: $\hat{m}_{dw} := \frac{m_{dw}}{1-\beta_1^t}$, $\hat{m}_{db} := \frac{m_{db}}{1-\beta_1^t}$ * corrige el sesgo inicial hacia cero en las primeras iteraciones

$\hat{V}_{dw} := \frac{v_{dw}}{1-\beta_2^t}$, $\hat{v}_{db} := \frac{V_{db}}{1-\beta_2^t}$

actualización: $w := w - \alpha \cdot \frac{\hat{m}_{dw}}{\sqrt{\hat{v}_{dw}} + \epsilon}$

* Divide el momentum por la raíz de la varianza. normaliza el paso: parámetros con gradientes grandes reciben pasos más pequeños y viceversa. es como tener una tasa de aprendizaje adaptativa por parámetro

$b := b - \alpha \cdot \frac{\hat{m}_{db}}{\sqrt{\hat{v}_{db}} + \epsilon}$

Típicamente: $\beta_1 = 0.9$ (momentum), $\beta_2 = 0.999$ (varianza), $\epsilon = 10^{-8}$ estabilidad numérica

* Combina lo mejor de momentum y RMS prop. es el optimizador más popular por que funciona bien "out of the box" sin mucho tuning

5. Explicación intuitiva:

El proceso de aprendizaje:

analogía: la función de error $J(w,b)$ es como un paisaje montañoso. queremos encontrar el valle más bajo (mínimo

El gradiente: $\nabla J = (dw, db)$ indica:

- Dirección: hacia dónde sube más rápido
- Magnitud: qué tan empinada es la pendiente

gradient descent: Nos movemos cuesta abajo (dirección opuesta al gradiente)

Ciclo de entrenamiento:

1. Predecir: con pesos actuales (Forward pass)
2. Medir: el error (Loss function)
3. Calcular: como cambiar pesos (Backpropagation)
4. Ajustar: pesos en dirección correcta (gradient descent)
5. Repetir: hasta convergencia

Interpretación de gradientes:

- Si $dw^{[L]} > 0$: aumentar w_{ij} incrementa error → disminuir peso
- Si $dw_{ij}^{[L]} < 0$: aumentar w_{ij} disminuye error → aumentar peso
- Si $|dw_{ij}^{[L]}|$ es grande: Ese peso tiene gran impacto
- Si $|dw_{ij}^{[L]}|$ es pequeño: ese peso tiene poco impacto

Rol de tasa de aprendizaje α:

Valor	efecto	Consecuencia
Grande (0.1)	pasos grandes	Rápido pero puede oscilar
Pequeño (0.001)	pasos pequeños	lento pero estable
Óptimo	Balance	Convergencia eficiente

Monitoreo del entrenamiento:

Señales de buen entrenamiento:

- loss disminuye consistente
- accuracy aumenta
- Convergencia suave

Problemas Comunes:

- loss oscila → α muy grande
- loss estancado → α muy pequeña o mínimo local
- training loss ↓, validación loss ↑ → overfitting

Ejemplo numérico simplificado:
para un peso w simple con $\alpha = 0.1$:

Iteración	w	$\hat{y}$	error	gradiente	nuevo w
0	0.50	0.80	0.040	-0.40	0.54
1	0.54	0.85	0.023	-0.30	0.57
2	0.57	0.91	0.008	-0.18	0.59

el error disminuye y w converge gradualmente al valor óptimo

Resumen de dimensiones

Capa	$W^{[l]}$	$b^{[l]}$	$z^{[l]}$	$a^{[l]}$
1	128 x 784	128x1	128x1	128x1
2	64 x 128	64x1	64x1	64x1
3	10 x 64	10x1	10x1	10x1

Total de parámetros:

- Pesos: $(128 \times 784) + (64 \times 128) + (10 \times 64) = 109,376$
- Sesgos: $128 + 64 + 10 = 202$
- total: 109,578 parámetros entrenables