Redes Neuronales

Minería de Datos (CC3074) - 2026

Redes Neuronales

Semestre 01, 2026

Importancia

Los modelos anteriores tienen una limitación en común.

Cada uno asume una forma específica de relación entre las variables.

Recap

| Modelo | Supuesto implícito | | --- | --- | | Regresión lineal | Relación lineal entre variables | | DT | Fronteras paralelas a los ejes | | SVM | Separación local basada en proximidad | | KNN | La distancia captura la similitud relevante |

Problemática

Los datos del mundo real no siempre encajan en esas formas.

Imágenes, audio, texto, series de tiempo complejas:

Ningún modelo clásico captura su estructura de forma natural.

Solución

Aprender la función directamente desde los datos.

Sin asumir una forma previa.

Sin límites en la complejidad de la relación que se puede modelar.

Eso es lo que hacen las redes neuronales.

La neurona artificial

La unidad básica de una red neuronal se llama neurona o nodo.

Está inspirada vagamente en la neurona biológica.

Qué hace una neurona

Recibe entradas, las pondera, las suma y aplica una transformación.

`$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$`

`$salida = f(z)$`

Donde:

- `$x_i$` → valores de entrada
- `$w_i$` → pesos (lo que el modelo aprende)
- `$b$` → sesgo (bias)
- `$f$` → función de activación

Los pesos son el modelo

Los pesos `$w_i$` determinan qué tan importante es cada entrada.

Al inicio se inicializan aleatoriamente.

El entrenamiento los ajusta hasta que la red produce buenas predicciones.

Arquitectura

Una red neuronal organiza las neuronas en capas.

Tipos de capas

| Capa | Descripción | | --- | --- | | Entrada | Recibe los datos crudos. Una neurona por variable. | | Ocultas | Procesan la información. Pueden haber varias. | | Salida | Produce la predicción final. |

Cómo fluye la información

Los datos entran por la capa de entrada.

Pasan por cada capa oculta de izquierda a derecha.

La capa de salida produce la predicción.

Este proceso se llama propagación hacia adelante (forward pass).

Red profunda

Una red con dos o más capas ocultas se llama red profunda.

De ahí el término aprendizaje profundo (deep learning).

Más capas → mayor capacidad de aprender patrones complejos.

Funciones de activación

Sin una función de activación, una red neuronal es solo una regresión lineal.

La función de activación introduce no-linealidad.

Eso es lo que permite aprender relaciones complejas.

ReLU — Rectified Linear Unit

`$f(z) = max(0, z)$`

- Si la entrada es positiva, la deja pasar.
- Si es negativa, la convierte en cero.

Es la más usada en capas ocultas.

Simple, computacionalmente eficiente y efectiva en la práctica.

Sigmoid

`$f(z) = 1 / (1 + e^{-z})$`

Comprime cualquier valor al rango (0, 1).

Útil en la capa de salida para clasificación binaria.

En valores extremos, el gradiente se vuelve casi cero (problema de gradiente).

Softmax

Convierte un vector de valores en una distribución de probabilidades.

La suma de todas las salidas es 1.

Se usa en la capa de salida para clasificación multiclase.

Cada salida representa la probabilidad de pertenecer a esa clase.

Tanh

`$f(z) = (e^z - e^{-z}) / (e^z + e^{-z})$`

Comprime los valores al rango (-1, 1).

Centrada en cero, a diferencia de sigmoid.

Usada ocasionalmente en capas ocultas.

Usos

| Capa | Función recomendada | | --- | --- | | Capas ocultas | ReLU (por defecto) | | Salida — clasificación binaria | Sigmoid | | Salida — clasificación multiclase | Softmax | | Salida — regresión | Ninguna (lineal) |

Aprendizaje

El aprendizaje consiste en ajustar los pesos para minimizar el error.

La función de pérdida

La red hace predicciones. Las predicciones tienen errores.

La función de pérdida convierte esos errores en un solo número.

Ese número es lo que el entrenamiento intenta reducir.

MSE — Regresión

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

Penaliza errores grandes de forma cuadrática.

Un error de 10 pesa 100 veces más que un error de 1.

Usar cuando la salida es un valor continuo (precio, temperatura, edad).

Entropía cruzada — Clasificación

Mide la diferencia entre la distribución de probabilidades predicha y la real.

Penaliza fuertemente cuando el modelo asigna poca probabilidad a la clase correcta.

`$L = -\sum_{c} y_c \cdot \log(\hat{y}_c)$`

- Clasificación binaria → entropía cruzada binaria (salida: sigmoid)
- Clasificación multiclase → entropía cruzada categórica (salida: softmax)

Formalmente

La función de pérdida define qué significa equivocarse.

Cambiarla cambia el comportamiento del modelo.

Es la definición del problema que se está resolviendo.

Gradiente descendente

Los pesos empiezan aleatoriamente. Las predicciones son malas.

El objetivo es mover los pesos hacia donde la pérdida es menor.

La analogía de la montaña

Imagina que estás en una montaña con los ojos cerrados.

Quieres llegar al punto más bajo del valle.

Cada paso: sentís la inclinación del terreno bajo tus pies.

Te movés en la dirección que baja más.

Eso es el gradiente descendente.

El gradiente

El gradiente indica la dirección en que la pérdida sube más rápido.

Para reducir la pérdida, nos movemos en la dirección opuesta.

`$w \leftarrow w - \eta \cdot \nabla L$`

- `$\nabla L$` → dirección en que sube el error
- `$\eta$` → learning rate: qué tan grande es el paso
- El signo negativo → nos movemos en la dirección contraria

El ciclo de entrenamiento

1. Forward pass: la red hace una predicción con los pesos actuales.
2. Se calcula la pérdida: ¿qué tan mal estuvo?
3. Backpropagation: se calcula el gradiente de cada peso.
4. Se actualizan los pesos un paso en la dirección que reduce la pérdida.
5. Repetir con el siguiente batch.

Después de miles de iteraciones, los pesos convergen a valores que minimizan el error.

Backpropagation

El gradiente descendente necesita saber cuánto culpar a cada peso.

Una red tiene miles o millones de pesos. Calcular eso manualmente es imposible.

Regla de la cadena

El error se conoce en la capa de salida.

Backpropagation lo propaga hacia atrás, capa por capa.

En cada capa calcula: ¿cuánto contribuyó este peso al error final?

Esa contribución es el gradiente del peso.

Lógica

Si un peso amplificó una señal que llevó a una predicción incorrecta → recibe más culpa → se ajusta más.

Si un peso tuvo poco efecto en la salida → recibe poca culpa → se ajusta poco.

Distribuye la responsabilidad del error de forma proporcional.

Hiperparámetros de una red neuronal

Afectan la capacidad, velocidad y estabilidad del entrenamiento.

Arquitectura

| Hiperparámetro | Descripción | | --- | --- | | Número de capas ocultas | Profundidad de la red | | Neuronas por capa | Ancho de cada capa | | Función de activación | Tipo de no-linealidad |

Entrenamiento

| Hiperparámetro | Descripción | | --- | --- | | Learning rate | Tamaño del paso en cada actualización | | Épocas | Cuántas veces el modelo ve el conjunto completo | | Batch size | Cuántas instancias procesa por actualización | | Optimizador | Variante de gradiente descendente |

Regularización

| Hiperparámetro | Descripción | | --- | --- | | Dropout | Fracción de neuronas desactivadas al azar | | L1 / L2 | Penalización sobre el tamaño de los pesos | | Early stopping | Detener cuando el error de validación no mejora |

Learning rate

Muy importante.

Controla qué tan grandes son los pasos del gradiente descendente.

Efecto del learning rate

| Valor | Comportamiento | | --- | --- | | Demasiado alto | El modelo no converge; oscila o diverge | | Demasiado bajo | Convergencia extremadamente lenta; puede quedar atrapado | | Adecuado | Convergencia estable y eficiente |

Rango típico a explorar

`[0.1, 0.01, 0.001, 0.0001]`

Siempre en escala logarítmica, no lineal.

El punto de partida más común: `0.001`.

Learning rate schedule

El learning rate no tiene que ser constante.

Se puede reducir progresivamente durante el entrenamiento.

Permite pasos grandes al inicio y fine tunning al final.

Épocas y Batch size

Época (epoch)

Una época es una pasada completa por todos los datos de entrenamiento.

Pocas épocas → subajuste: el modelo no ha aprendido suficiente.

Muchas épocas → sobreajuste: el modelo memoriza los datos de entrenamiento.

Batch size

Cuántas instancias se procesan antes de actualizar los pesos.

Batch size = 1 (SGD puro)

- Actualiza los pesos después de cada instancia.
- Muy ruidoso — el gradiente varía mucho entre pasos.
- Puede escapar de mínimos locales, pero converge lento.

Batch size = 32 – 128 (mini-batch)

- El balance más usado en la práctica.
- Gradiente suficientemente estable, velocidad razonable.
- Valores típicos: `32`, `64`, `128`, `256`.

Batch size = dataset completo (batch GD)

- El gradiente es exacto — sin ruido.
- Muy lento en datasets grandes.
- Puede quedar atrapado en mínimos locales al no tener ruido.

Relación entre épocas y batch size

Batch size pequeño → más actualizaciones por época.

Batch size grande → menos actualizaciones por época, pero más estables.

Valores típicos: `32`, `64`, `128`, `256`.

Arquitectura: capas y neuronas

Cuántas capas ocultas

| Capas ocultas | Cuándo usarlo | | --- | --- | | 1 | Problemas simples, datos tabulares | | 2 – 3 | La mayoría de problemas de minería de datos | | Muchas | Imágenes, texto, audio (deep learning especializado) |

Más capas no siempre es mejor.

Cada capa añade complejidad y riesgo de sobreajuste.

Cuántas neuronas por capa

No hay una regla universal.

Puntos de partida razonables:

- Empezar con un número entre el tamaño de entrada y el de salida.
- Probar potencias de 2: `32, 64, 128, 256`.

- Arquitectura decreciente: más neuronas cerca de la entrada, menos cerca de la salida.

Principio general

Empezar con una red pequeña.

Aumentar la complejidad solo si el modelo tiene alto sesgo (subajuste).

Si hay sobreajuste, primero aplicar regularización antes de reducir la arquitectura.

Optimizadores

El gradiente descendente tiene variantes que mejoran la convergencia.

SGD — Stochastic Gradient Descent

La variante más básica.

Actualiza los pesos con el gradiente de un mini-batch.

Ruidoso pero efectivo con momentum.

Adam — Adaptive Moment Estimation

El optimizador más usado en la práctica.

Combina momentum y tasa de aprendizaje adaptativa por parámetro.

Converge rápido y es robusto a la elección del learning rate.

Punto de partida recomendado para la mayoría de problemas.

RMSprop

Adapta el learning rate basándose en la magnitud reciente de los gradientes.

Funciona bien en problemas con gradientes muy variables.

Común en redes recurrentes.

Comparación

| Optimizador | Velocidad | Estabilidad | Cuándo usarlo | | --- | --- | --- | --- | | SGD + momentum | Media | Alta | Cuando se quiere control total | | Adam | Alta | Media-Alta | Punto de partida por defecto | | RMSprop | Alta | Media | Redes recurrentes, datos ruidosos |

Regularización en redes neuronales

Las redes neuronales son modelos muy expresivos.

Sin regularización, tienden a sobreajustarse, especialmente con pocos datos.

Dropout

Durante el entrenamiento, se desactivan neuronas al azar.

La fracción desactivada es el hiperparámetro `dropout rate`.

| Dropout rate | Efecto | | --- | --- | | 0.0 | Sin regularización | | 0.2 – 0.3 | Regularización leve | | 0.5 | Regularización moderada (valor clásico) | | > 0.5 | Regularización agresiva; puede causar subajuste |

Durante la evaluación, todas las neuronas están activas.

L2 — Weight Decay

Añade una penalización proporcional al cuadrado de los pesos.

Pesos grandes son costosos → el modelo los evita.

Equivalente a la regularización Ridge en regresión lineal.

Ayuda cuando hay muchos pesos y poco dato.

Early Stopping

Monitorear el error en el conjunto de validación durante el entrenamiento.

Detener cuando el error de validación deja de mejorar.

Evita memorizar los datos de entrenamiento sin necesidad de ajustar épocas manualmente.

Es la forma de regularización más práctica y más usada.

Batch Normalization

Normaliza las activaciones dentro de cada capa.

Estabiliza el entrenamiento y permite usar learning rates más altos.

Reduce la dependencia de la inicialización de pesos.

Estrategia de tuneo

Hay demasiados hiperparámetros para explorarlos todos a la vez.

Se necesita una estrategia.

Orden recomendado de tuneo

1. Learning rate — el más importante; empieza aquí.
2. Arquitectura — número de capas y neuronas.

3. Batch size — ajustar si el entrenamiento es inestable.
4. Regularización — dropout o L2 si hay sobreajuste.
5. Optimizador — Adam por defecto; cambiar solo si hay razón.

Búsqueda aleatoria vs. en grilla

| Método | Ventaja | Desventaja | | --- | --- | --- | | Grilla (Grid Search) | Exhaustivo dentro del espacio definido | Costoso si hay muchos hiperparámetros | | Aleatoria (Random Search) | Explora mejor el espacio con menos evaluaciones | No garantiza cubrir todas las combinaciones | | Bayesiana | Aprende qué regiones son prometedoras | Más compleja de implementar |

Para redes neuronales, la búsqueda aleatoria supera a la grilla en la mayoría de casos.

Un hiperparámetro puede ser irrelevante; la grilla desperdicia evaluaciones en él.

Diagnóstico por curvas de aprendizaje

Monitorear la pérdida en entrenamiento y validación durante el entrenamiento.

| Patrón | Diagnóstico | Acción | | --- | --- | --- | | Ambas pérdidas altas | Subajuste | Más capas, más neuronas, más épocas | | Pérdida de entrenamiento baja, validación alta | Sobreajuste | Dropout, L2, early stopping, menos capas | | Pérdida de validación oscila mucho | Learning rate alto | Reducir learning rate | | La validación deja de mejorar | Época óptima alcanzada | Aplicar early stopping |

Comparación con otros modelos

| Aspecto | Red Neuronal | SVM | Random Forest | Reg. Logística | | --- | --- | --- | --- | --- | | Complejidad del modelo | Muy alta | Media | Media | Baja | |

Interpretabilidad | Muy baja | Baja | Media | Alta | | Datos necesarios | Muchos | Pocos-Medios | Medios | Pocos | | Entrenamiento | Lento | Lento | Medio | Rápido | | Tuneo requerido | Mucho | Medio | Poco | Poco | | Escala con datos grandes | Excelente | Deficiente | Buena | Buena |

¿Cuándo usar redes neuronales?

- Datasets grandes (decenas de miles de instancias o más)
- Datos no estructurados: imágenes, texto, audio
- Cuando otros modelos han alcanzado su límite de rendimiento
- Cuando se tiene capacidad computacional adecuada

¿Cuándo NO usar redes neuronales?

- Datasets pequeños (menos de ~1,000 instancias)
- Se requiere interpretabilidad del modelo
- Recursos computacionales limitados
- El problema puede resolverse con un modelo más simple