**به نام خدا**

**بخش ۱: مبانی شبکه‌های عصبی (با جزئیات بیشتر)**

**۱.۱ نورون مصنوعی و پرسپترون**

* **فرمول نورون مصنوعی:**

خروجی = تابع\_فعال‌سازی(w1\*x1 + w2\*x2 + ... + wn\*xn + bias)

* **توابع فعال‌سازی پرکاربرد:**
  + **Sigmoid:** برای خروجی احتمالی (بین ۰ و ۱)
  + **ReLU:** برای لایه‌های میانی (محاسبات سریع‌تر)
  + **Softmax:** برای طبقه‌بندی چندکلاسه

#### ****۱.۲ معماری شبکه‌های عصبی****

**.2-1-1لایه‌ها:**

* + **لایه ورودی (Input Layer):** تعداد نورون‌ها = تعداد ویژگی‌ها.
  + **لایه‌های پنهان (Hidden Layers):** هر لایه می‌تواند نورون‌های مختلفی داشته باشد.
  + **لایه خروجی (Output Layer):** بسته به مسئله (۱ نورون برای رگرسیون، چند نورون برای طبقه‌بندی).

**.2-1-2پارامترهای قابل یادگیری:**

* **وزن‌ها (Weights)** و **بایاس‌ها (Biases)**.

### ****بخش ۲: پیاده‌سازی یک شبکه عصبی ساده با NumPy****

#### ****۲.۱ پیاده‌سازی از پایه****

#### در ریپوی خود، فایل Week4/Neural\_Network\_Scratch.py را ایجاد کن

#### ****۲.۲ تفسیر کد:****

* **Forward Pass:** محاسبه خروجی شبکه.
* **Backpropagation:** محاسبه گرادیان‌ها و به‌روزرسانی وزن‌ها.
* **داده‌های ماه (make\_moons):** یک مسئله **غیرخطی** که با مدل‌های خطی قابل حل نیست!

### ****بخش ۳: بهبود مدل با کتابخانه‌های مدرن (TensorFlow/Keras)****

#### ****۳.۱ پیاده‌سازی همان مدل با Keras****

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

model = Sequential([

Dense(4, activation='relu', input\_shape=(2,)), *# لایه پنهان با 4 نورون*

Dense(1, activation='sigmoid') *# لایه خروجی*

])

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

model.fit(X\_train, y\_train, epochs=100, verbose=0)

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f"دقت Keras روی داده تست: {accuracy \* 100:.2f}%")

#### ****۳.۲ مقایسه نتایج:****

* **مدل NumPy:** دقت ~۸۵-۹۰٪
* **مدل Keras:** دقت ~۹۵-۹۸٪ (به دلیل استفاده از **Adam Optimizer** و **ReLU**)

### ****بخش ۴: پروژه عملی (پیشرفته)****

#### ****۴.۱ بهبود مدل NumPy****

* + از **نرمال‌سازی داده‌ها** (StandardScaler) استفاده کن.

#### ****۴.۲ استفاده از دیتاست‌های واقعی****

* **MNISTتشخیص رقم‌های دستنویس:**

from tensorflow.keras.datasets import mnist

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

**توضیحات مربوط به کد بخش Neural\_Network\_Keras**

**توضیحات کد :**

model.fit(X\_train, y\_train, epochs=100) *# مدل 100 بار روی داده‌ها آموزش می‌بیند*

در کد model.fit() در Keras، پارامترهای epochs و verbose نقش مهمی در فرآیند آموزش مدل دارند:

### 1. epochs****دورهای آموزش****

* **تعریف**:  
  تعداد دفعاتی که مدل روی **کل مجموعه داده آموزشی** (از ابتدا تا انتها) آموزش می‌بیند.
  + هر epoch = یک پاس کامل روی تمام داده‌های آموزشی.
* **نکات**:
  + هرچه epochs  بیشتر باشد، مدل فرصت بیشتری برای یادگیری دارد.
  + اگر بیش از حد زیاد باشد، ممکن است مدل **بیش‌برازش (Overfitting)** کند.
  + معمولاً بین ۵۰ تا ۵۰۰ انتخاب می‌شود (بسته به پیچیدگی داده).

### 2. verbose****میزان جزئیات نمایش خروجی****

* **تعریف**:  
  کنترل می‌کند که اطلاعات آموزش چگونه نمایش داده شوند.
* **مقادیر ممکن**:
  + 0: هیچ خروجی نمایش داده نمی‌شود (سکوت).
  + 1: نمایش نوار پیشرفت و اطلاعات هر epoch (پیش‌فرض).
  + 2: نمایش خلاصه‌تر (فقط عدد epoch و خطا).

**نکات**:

* + در محیط‌های تولیدی یا زمانی که نیاز به ذخیره منابع دارید، از verbose=0 استفاده کنید.
  + برای دیباگ مدل، verbose=1 مفید است.

### ****چرا از**** epochs=100****و****verbose=0****استفاده کنیم؟****

* epochs=100:
  + برای داده‌های ساده (مثل make\_moons) معمولاً کافی است.
  + برای داده‌های پیچیده‌تر ممکن است نیاز به افزایش باشد.
* verbose=0:
  + وقتی نیازی به مشاهده روند آموزش نیست (مثلاً در اسکریپت‌های خودکار).
  + برای جلوگیری از شلوغی خروجی.