**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.

**توضیحات کد :**

model = tf.keras.models.Sequential([

    tf.keras.layers.Dense(4, activation='relu', input\_shape=(2,)),  # لایه پنهان با 4 نورون

    tf.keras.layers.Dense(1, activation='sigmoid')                 # لایه خروجی

]

### ****۱. انتخاب****1****برای لایه خروجی با****sigmoid

* **دلیل عدد ۱**:
  + این مدل برای **طبقه‌بندی باینری** (دوکلاسه) طراحی شده است (مثلاً تشخیص خریدار/غیرخریدار).
  + در چنین مسائلی، لایه خروجی فقط به **یک نورون** نیاز دارد که احتمال تعلق به کلاس 1  را پیش‌بینی کند.
  + اگر مسئله **چندکلاسه** بود (مثلاً ۳ کلاس)، تعداد نورون‌های خروجی برابر با تعداد کلاس‌ها می‌شد (مثلاً ۳) و از  softmax  به جای sigmoid استفاده می‌شد.
* **دلیل**sigmoid:
  + تابع sigmoid خروجی را بین 0 و 1 نگه می‌دارد که برای تفسیر به عنوان **احتمال** مناسب است.
  + مثال: خروجی 0.8 به معنای ۸۰٪ احتمال تعلق به کلاس  1 است.

### ****۲. انتخاب****4****برای لایه پنهان با****relu

* **دلیل عدد ۴**:
  + این یک **انتخاب تجربی** بر اساس پیچیدگی مسئله است. از آنجا که داده‌های مثال ساده بودند (مثلاً سن و درآمد)، **۴ نورون** کافی است.
  + اگر داده‌ها پیچیده‌تر بودند (مثلاً تصاویر با ابعاد بالا)، تعداد نورون‌ها افزایش می‌یافت (مثلاً ۱۲۸ یا ۲۵۶).
  + قانون کلی: **هرچه داده پیچیده‌تر، نورون‌های بیشتر**. اما باید از **بیش‌برازش (Overfitting)** جلوگیری کرد.
* **دلیل**relu:
  + relu (Rectified Linear Unit) یک تابع فعال‌سازی **غیرخطی** است که به مدل کمک می‌کند روابط پیچیده را یاد بگیرد.
  + مزایا:
    - محاسبات سریع.
    - جلوگیری از مشکل **محو گرادیان** (Vanishing Gradient) در مقایسه با توابعی مثل tanh.

### ****۳. ملاحظات کلی در طراحی معماری مدل****

| **پارامتر** | **توضیح** | **مثال برای داده‌های پیچیده‌تر** |
| --- | --- | --- |
| **تعداد لایه‌های پنهان** | برای داده‌های ساده، یک لایه کافی است. | ۲-۳ لایه برای داده‌های پیچیده. |
| **تعداد نورون‌ها** | معمولاً بین ۵ تا ۱۰۰ نورون در هر لایه (بسته به پیچیدگی داده). | ۱۲۸ نورون برای پردازش تصویر. |
| **تابع فعال‌سازی** | relu برای لایه‌های پنهان، sigmoid/softmax برای لایه خروجی. | tanh برای خروجی‌های بین -۱ و ۱. |

### ****۴. مثال با داده‌های پیچیده‌تر****

اگر داده‌های شما **۱۰ ویژگی** داشته باشد و یک مسئله **سه‌کلاسه** باشد، معماری مدل ممکن است به این شکل باشد:

model = tf.keras.Sequential([

tf.keras.layers.Dense(64, activation='relu', input\_shape=(10,)), *# لایه پنهان با 64 نورون*

tf.keras.layers.Dense(32, activation='relu'), *# لایه پنهان دوم*

tf.keras.layers.Dense(3, activation='softmax') *# لایه خروجی برای 3 کلاس*

])

### ****۵. چگونه تعداد بهینه نورون‌ها را پیدا کنیم؟****

* **روش آزمون و خطا**: با مقادیر مختلف آزمایش کنید (مثلاً ۴، ۸، ۱۶) و دقت مدل را مقایسه کنید.
* **استفاده از روش‌هایسیستماتیک** مانند **HyperparameterTuning** مثلاً GridSearchCV در scikit-learn.

### ****۶. نکته پایانی****

* **برای شروع**، از اعداد کوچک (مثلاً ۴-۸ نورون) استفاده کنید و در صورت نیاز افزایش دهید.
* **اگر مدل Underfit کند** (دقت پایین)، تعداد نورون‌ها یا لایه‌ها را افزایش دهید.
* **اگر مدل Overfit کند** (دقت آموزش بالا ولی تست پایین)، از تکنیک‌هایی مثل **Dropout** یا **Regularization** استفاده کنید.

**7.** **انتخاب**input\_shape=(2,)

پارامتر input\_shape=(2,) در لایه اول مدل Keras/TensorFlow مشخص می‌کند که **هر نمونه از داده‌های ورودی باید چه ساختاری داشته باشد**. در این مورد خاص:

### 🔍 ****معنی****input\_shape=(2,):

* 2: نشان می‌دهد **هر نمونه ورودی ۲ عدد (۲ ویژگی)** دارد.
  + مثال: اگر داده‌های شما شامل **[سن، درآمد]** باشد، input\_shape=(2,) مناسب است.
* ,**در پرانتز**: نشان می‌دهد که این یک **تاپل (tuple)** است (حتی اگر فقط یک عدد داشته باشد).

### 💡 ****چرا این پارامتر مهم است؟****

* مدل باید بداند **ابعاد داده ورودی** چگونه است تا وزن‌ها را به درستی مقداردهی کند.
* فقط در **لایه اول** مدل نیاز به تعریف input\_shape است (لایه‌های بعدی به طور خودکار ابعاد را محاسبه می‌کنند).

### ****مثال‌های کاربردی****:

#### ۱. ****داده با ۲ ویژگی (مثلاً سن و درآمد)****:

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

model = Sequential([

Dense(4, activation='relu', input\_shape=(2,)), *# ورودی: ۲ عدد (مثلاً [25, 30])*

Dense(1, activation='sigmoid')

])

#### ۲. ****داده با ۵ ویژگی (مثلاً قد، وزن، سن، فشار خون، کلسترول)****:

model = Sequential([

Dense(16, activation='relu', input\_shape=(5,)), *# ورودی: ۵ عدد*

Dense(1, activation='sigmoid')

])

#### ۳. ****تصاویر سیاه‌وسفید 28\*28 پیکسل مثل MNIST****

28\*28=784

model = Sequential([

Dense(128, activation='relu', input\_shape=(784,)), *# 28x28 = 784 پیکسل*

Dense(10, activation='softmax')

])

**توضیحات کد :**

#### wrong\_predictions = np.where(y\_pred\_classes != y\_test)[0]

در خط کد wrong\_predictions = np.where(y\_pred\_classes != y\_test)[0]، عدد 0 در انتهای عبارت ([0]) به این معنی است:

### 🔍 ****تجزیه و تحلیل عبارت****:

1. y\_pred\_classes != y\_test:
   * یک آرایه بولین (True/False) ایجاد می‌کند که نشان می‌دهد کدام پیش‌بینی‌ها اشتباه هستند.
   * مثال:

y\_test = [0, 1, 0, 1]

y\_pred\_classes = [0, 0, 0, 1]

y\_pred\_classes != y\_test *# نتیجه: [False, True, False, False]*

np.where(...):

* ایندکس‌های (شماره موقعیت) مقادیر True را در آرایه بولین برمی‌گرداند.
* خروجی این تابع **یک تاپل (tuple)** است که شامل آرایه‌ای از ایندکس‌ها می‌شود.
* مثال:
* np.where([False, True, False, False]) *# نتیجه: (array([1], dtype=int64),)*

[0]:

* چون np.where همیشه یک تاپل برمی‌گرداند (حتی اگر یک آرایه داشته باشد)، با [0] به **اولین عنصر تاپل** (همان آرایه ایندکس‌ها) دسترسی می‌یابیم.
* در مثال بالا:
* wrong\_predictions = np.where([False, True, False, False])[0] *# نتیجه: array([1])*

### ****چرا این کار را می‌کنیم؟****

* هدف: **فهرست کردن ایندکس‌های نمونه‌های اشتباه** در داده‌های تست.
* با این ایندکس‌ها می‌توانیم:
  + نمونه‌های اشتباه را بررسی کنیم.
  + خطاها را تحلیل کنیم.
  + مدل را بهبود دهیم.