**به نام خدا**

**بهینه‌سازی شبکه‌های عصبی و مقدمه‌ای بر یادگیری عمیق (قدم‌به‌قدم)**

**گام ۱: مرور و بهبود مدل NumPy از جلسه قبل**

**۱.۱ مشکلات مدل قبلی:**

* استفاده از **Sigmoid** در لایه پنهان → مشکل **Vanishing Gradients**.
* نرمال‌سازی نشدن داده‌ها → همگرایی کند.
* نرخ یادگیری ثابت → ممکن است مدل بهینه نشود.

**۱.۲ بهبودها:**

* جایگزینی **ReLU** به جای Sigmoid در لایه پنهان.
* نرمال‌سازی داده‌ها با StandardScaler.
* افزودن **لایه Dropout** برای جلوگیری از Overfitting اختیاری

**کد بهبودیافته:** (Week7/Neural\_Network\_Scratch.py)

from sklearn.preprocessing import StandardScaler

*# نرمال‌سازی داده‌ها*

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

class ImprovedNeuralNetwork:

def \_\_init\_\_(self):

self.W1 = np.random.randn(2, 4) \* 0.01 *# مقداردهی اولیه کوچک‌تر*

self.b1 = np.zeros(4)

self.W2 = np.random.randn(4, 1) \* 0.01

self.b2 = np.zeros(1)

def relu(self, x):

return np.maximum(0, x)

def forward(self, X):

self.z1 = np.dot(X, self.W1) + self.b1

self.a1 = self.relu(self.z1) *# تغییر به ReLU*

self.z2 = np.dot(self.a1, self.W2) + self.b2

self.a2 = self.sigmoid(self.z2)

return self.a2

def train(self, X, y, epochs=1000, lr=0.01):

for epoch in range(epochs):

*# ... (مانند قبل، با محاسبات گرادیان برای ReLU)*

**✅ خروجی:**

* مدل باید سریع‌تر همگرا شود و دقت بهتری داشته باشد ~۹۰-۹۵٪.

### ****📌 گام ۲: استفاده از کتابخانه‌های مدرن (Keras) + تنظیم هیپرپارامترها****

#### ****۲.۱ معماری پیشرفته‌تر با Keras:****

from tensorflow.keras.layers import Dropout

model = Sequential([

Dense(16, activation='relu', input\_shape=(2,)), *# لایه پنهان با ۱۶ نورون*

Dropout(0.2), *# 20% از نورون‌ها در آموزش خاموش می‌شوند پایین توضیح داده شده*

Dense(1, activation='sigmoid')

])

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

history = model.fit(X\_train\_scaled, y\_train, epochs=100, validation\_split=0.2)

#### ****۲.۲ تحلیل یادگیری:****

* **رسم نمودار خطا (Loss) و دقت (Accuracy):**

python

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.legend()

plt.show()

**نتیجه:**

* اگر **دقت آموزش** خیلی بیشتر از **دقت اعتبارسنجی** باشد → **Overfitting**.
* راه‌حل: افزایش Dropout، استفاده از Regularization، یا افزودن داده‌های بیشتر.

### ****📌 گام ۳: معرفی یادگیری عمیق (Deep Learning) و CNN****

#### ****۳.۱ چرا شبکه‌های عمیق؟****

* برای داده‌های پیچیده مثل **تصاویر** (پیکسل‌های مجاور همبستگی دارند).
* **معماری CNN:** ترکیب لایه‌های **کانولوشن** (برای تشخیص ویژگی‌های محلی) و **لایه‌های کاملاً متصل**.

#### ****۳.۲ پیاده‌سازی ساده CNN برای MNIST:****

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten

model = Sequential([

Conv2D(32, (3,3), activation='relu', input\_shape=(28,28,1)), *# لایه کانولوشن*

MaxPooling2D((2,2)), *# لایه Pooling*

Flatten(), *# تبدیل به بردار یک‌بعدی*

Dense(64, activation='relu'),

Dense(10, activation='softmax') *# خروجی 10 کلاس (ارقام 0-9)*

])

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

model.fit(X\_train, y\_train, epochs=5)

**✅ خروجی:**

* دقت معمولاً > ۹۸٪ روی داده‌های MNIST.

#### استفاده از Dropout(0.2) :

#### استفاده از Dropout(0.2) در شبکه‌های عصبی یک تکنیک **منظم‌سازی (Regularization)** است که به جلوگیری از **بیش‌برازش (Overfitting)** کمک می‌کند. در اینجا به زبان ساده توضیح می‌دهم که چرا و چگونه از آن استفاده می‌کنیم:

### ****دلیل استفاده از Dropout****:

1. **جلوگیری از بیش‌برازش**:
   * وقتی مدل بیش از حد روی داده‌های آموزشی سازگار شود (حتی نویزها را هم یاد بگیرد)، Dropout با "خاموش کردن" تصادفی برخی نورون‌ها، از این اتفاق جلوگیری می‌کند.
2. **اجبار به یادگیری جمعی**:
   * با غیرفعال کردن ۲۰% نورون‌ها در هر بار آموزش، مدل مجبور می‌شود به جای تکیه بر چند نورون خاص، از **تمام شبکه** برای پیش‌بینی استفاده کند.
3. **شبیه‌سازی مدل‌های ترکیبی**:
   * در هر مرحله آموزش، یک معماری متفاوت از شبکه را امتحان می‌کند (مثل آموزش ۱۰۰۰ مدل مختلف!).

### 🛠 ****چگونه کار می‌کند؟****

* **در طول آموزش**:
  + در هر گام به‌صورت تصادفی ۲۰% از نورون‌های لایه مورد نظر **غیرفعال** می‌شوند.
  + فقط ۸۰% نورون‌ها در محاسبات مشارکت می‌کنند.
  + این کار در **هر بار فراخوانی** مدل در حین آموزش تکرار می‌شود.
* **در طول پیش‌بینی (Test Time)**:
  + تمام نورون‌ها فعال می‌مانند.
  + خروجی نورون‌ها ضرب در 0.8 می‌شود (جبران اثر غیرفعال بودن ۲۰% نورون‌ها در آموزش).

### ✅ ****چه زمانی از Dropout استفاده کنیم؟****

1. **شبکه‌های عمیق** (تعداد لایه‌های زیاد)
2. **داده‌های آموزشی کم** (احتمال بیش‌برازش بالا)
3. **وقتی مدل روی داده Train خوب عمل می‌کند ولی روی Test ضعیف است**

### ****تشبیه ساده****:

فرض کنید شما برای امتحان نهایی، به جای تکیه بر یک دوست خاص (ممکن است بعضی سوالات را بلد نباشد!)، از **گروهی از دوستان** سوال می‌پرسید. هر بار تصادفی ۲۰% از آن‌ها غایب هستند، بنابراین مجبور می‌شوید به **دانش جمعی** تکیه کنید!

#### مفهوم CNN یا کانولوشن :

کانولوشن (Convolution) یک عملیات ریاضی و پایه‌ای در پردازش سیگنال و بینایی کامپیوتر است که در شبکه‌های عصبی کانولوشنی (CNN) برای استخراج ویژگی‌ها از داده‌هایی مانند تصاویر استفاده می‌شود. درک آن به زبان ساده:

### 1. ****مفهوم اصلی****:

کانولوشن مانند یک "فیلتر هوشمند" است که روی تصویر حرکت می‌کند و ویژگی‌های خاصی (مثل لبه‌ها، بافت‌ها) را شناسایی می‌کند.

### 2. ****اجزای کلیدی****:

* **کرنل (Kernel)**: یک ماتریس کوچک (مثلاً 3x3) که حاوی مقادیر یادگرفتنی است.
* **استراید (Stride)**: تعداد پیکسل‌هایی که کرنل در هر حرکت جابه‌جا می‌شود.
* **پدینگ (Padding)**: اضافه کردن حاشیه به تصویر برای حفظ ابعاد.

### 3. ****مثال ساده****:

فرض کنید یک تصویر سیاه‌وسفید 5x5 دارید و یک کرنل 3x3:

تصویر: کرنل:

1 0 1 0 1 1 0 1

0 1 0 1 0 0 1 0

1 0 1 0 1 1 0 1

0 1 0 1 0

1 0 1 0 1

کانولوشن با ضرب عناصر همپوشان و جمع نتایج محاسبه می‌شود.

### 4. ****چرا در CNN استفاده می‌شود؟****

* **حفظ رابطه مکانی**: پیکسل‌های مجاور را با هم تحلیل می‌کند.
* **کاهش پارامترها**: نسبت به لایه‌های Fully-Connected پارامترهای کمتری دارد.
* **یادگیری سلسله‌مراتبی**: ابتدا لبه‌ها، سپس اشکال پیچیده‌تر را می‌آموزد.

### 5. ****کاربردهای عملی****:

* تشخیص لبه‌ها در تصاویر
* شناسایی الگوهای تکرارشونده
* کاهش نویز

### 6. ****مثال کد در Keras****:

from tensorflow.keras.layers import Conv2D

model.add(Conv2D(

filters=32, *# تعداد فیلترها*

kernel\_size=(3, 3), *# ابعاد کرنل*

activation='relu',

input\_shape=(28, 28, 1) *# ابعاد تصویر ورودی*

))

7. **تفاوت با لایه‌های معمولی**:

| **ویژگی** | **لایه کانولوشن** | **لایه Dense معمولی** |
| --- | --- | --- |
| اتصالات | محلی (همسایگی) | تمامی به تمامی |
| پارامترها | کم (اشتراکی) | بسیار زیاد |
| حفظ ساختار | بله | خیر |

### 8. ****تشبیه انسانی****:

مثل وقتی که با یک ذره‌بین بخش‌های کوچک تصویر را بررسی می‌کنید تا جزئیات را ببینید، سپس این اطلاعات جزئی را ترکیب می‌کنید تا کل تصویر را درک کنید.

### کانولوشن یکی از قدرتمندترین ایده‌ها در پردازش تصویر است که به مدل‌های امروزی اجازه می‌دهد اشیا را با دقت بالا تشخیص دهند!

#### توضیح کد

Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1))

Conv2D(

filters=32, *# تعداد فیلترها (تعداد ویژگی‌های استخراج شده)*

kernel\_size=(3, 3), *# ابعاد پنجره کانولوشن (height, width)*

activation='relu', *# تابع فعال‌سازی (غیرخطی‌سازی)*

input\_shape=(28, 28, 1) *# ابعاد ورودی (height, width, channels)*

)

### 🎯 ****توضیح پارامترها****:

1. filters=32:
   * تعداد فیلترهایی که شبکه یاد می‌گیرد.
   * هر فیلتر یک ویژگی خاص (مثل لبه، رنگ، بافت) را تشخیص می‌دهد.
   * مثلاً 32 فیلتر → 32 ویژگی مختلف استخراج می‌شود.
2. kernel\_size=(3, 3):
   * اندازه پنجره‌ای که روی تصویر حرکت می‌کند.
   * رایج‌ترین اندازه‌ها: (3,3) یا (5,5).
   * هرچه کوچکتر باشد، ویژگی‌های موضعی‌تر را می‌گیرد.
3. activation='relu':
   * تابع فعال‌سازی ReLU (max(0, x)).
   * مزایا:
     + محاسبات سریع
     + حل مشکل vanishing gradient
     + غیرخطی‌سازی مدل
4. input\_shape=(28, 28, 1):
   * ابعاد تصویر ورودی:
     + 28: ارتفاع
     + 28: عرض
     + 1: کانال‌های رنگی (1=سیاه‌سفید، 3=رنگی)

### 💡 ****نکات کاربردی****:

* **چرا 32 فیلتر؟**  
  معمولاً با اعداد پایین (32، 64) شروع می‌کنیم و در لایه‌های بعدی افزایش می‌دهیم.
* **چرا kernel\_size فرد است؟**  
  برای داشتن مرکز متقارن (مثلاً 3x3 یک پیکسل مرکزی دارد).
* **تغییرات در لایه‌های بعدی**:  
  در لایه‌های بعدی نیازی به input\_shape نیست (مدل به طور خودکار تشخیص می‌دهد).

#### توضیح کد MaxPooling2D((2,2)),

MaxPooling2D((2,2)) در شبکه‌های عصبی کانولوشنی (CNN) یک لایه **کاهش ابعاد (Downsampling)** است که با حفظ مهم‌ترین ویژگی‌ها، حجم محاسبات را کاهش می‌دهد. در اینجا به زبان ساده توضیح می‌دهم چگونه کار می‌کند و چرا مفید است:

### 🌟 ****کارکرد لایه MaxPooling****:

1. **ورودی**: یک تنسور تصویر (مثلاً ابعاد 24x24 پیکسل).
2. **پنجره Pooling**: یک پنجره 2x2 روی تصویر حرکت می‌کند.
3. **عملیات**: از هر پنجره 2x2، **بیشینه مقدار (Max)** را انتخاب می‌کند.
4. **خروجی**: تنسری با ابعاد نصف شده (مثلاً 12x12).

### 🎯 ****مثال تصویری****:

ورودی (4x4): پنجره 2x2: خروجی (2x2):

[1, 3, 2, 5] [1, 3] [4, 5]

[4, 2, 1, 0] → [4, 2] → [3, 2]

[0, 1, 3, 2] [0, 1]

[2, 3, 1, 2] [2, 3]

* **نتیجه**: [4, 5, 3, 2] (بزرگ‌ترین مقادیر از هر پنجره).

### 💡 ****چرا از MaxPooling استفاده می‌کنیم؟****

1. **کاهش ابعاد داده**:
   * حجم محاسبات را کم می‌کند (سرعت آموزش افزایش می‌یابد).
   * مثال: تصویر 100x100 پس از MaxPooling(2,2) → 50x50.
2. **حفظ ویژگی‌های مهم**:
   * فقط مقادیر بارز (مثل لبه‌های قوی) را نگه می‌دارد.
   * به نویز حساس نیست (برخلاف میانگین‌گیری).
3. **افزایش عدم وابستگی به موقعیت**:
   * حتی اگر ویژگی جابه‌جا شود، همچنان تشخیص داده می‌شود.
4. 🛠 **مقایسه با سایر انواع Pooling**:

| **نوع Pooling** | **عملکرد** | **کاربرد** |
| --- | --- | --- |
| **Max** | انتخاب بیشترین مقدار | رایج‌ترین نوع |
| **Average** | محاسبه میانگین | کاهش نویز |
| **Min** | انتخاب کمترین مقدار | موارد خاص (مثل تشخیص نقاط تاریک) |

### ****نکات کاربردی****:

1. **پارامترهای رایج**:
   * pool\_size=(2,2): متداول‌ترین اندازه.
   * strides=2: گام حرکت (معمولاً برابر با ابعاد پنجره).

**جایگذاری در معماری CNN**:

model = Sequential([

Conv2D(32, (3,3), activation='relu', input\_shape=(28,28,1)),

MaxPooling2D((2,2)), *# معمولاً بعد از لایه کانولوشن*

Conv2D(64, (3,3), activation='relu'),

MaxPooling2D((2,2)),

Flatten(),

Dense(10, activation='softmax')

])

1. **تأثیر روی ابعاد**:
   * اگر ورودی 28x28 باشد، پس از MaxPooling(2,2) → 14x14.

### ****تشبیه انسانی****:

فرض کنید می‌خواهید یک کتاب قطور را خلاصه کنید:

* **کانولوشن**: مثل **هایلایت کردن جمله‌های** مهم.
* **ماکس پولینگ**: مثل انتخاب فقط **یک کلمه کلیدی** از هر **پاراگراف هایلایت‌شده**.

**توضیح کد np.argmax(predictions, axis=1)**

np.argmax یک تابع NumPy است که **اندیس (موقعیت) بزرگ‌ترین مقدار** را در یک آرایه پیدا می‌کند.

یک عمل ساده اما حیاتی در یادگیری ماشین انجام می‌دهد.

### ****کارکرد اصلی****:

* **ورودی (**predictions**)**: یک آرایه از احتمالات (مثلاً خروجی مدل برای ۵ نمونه):

[[0.01, 0.02, 0.90, 0.05, ...], *# احتمال کلاس ۲ برای نمونه اول*

[0.80, 0.10, 0.05, 0.03, ...], *# احتمال کلاس ۰ برای نمونه دوم*

...]

* **عملکرد**:  
  **شماره کلاس با بیشترین احتمال** را برای هر نمونه پیدا می‌کند (پیش‌بینی نهایی).

**خروجی**:

[2, 0, ...] *# برچسب‌های پیش‌بینی شده*

### ****مثال ساده****:

فرض کنید مدل شما برای یک تصویر (عدد "۳") این احتمالات را داده:

[0.01, 0.01, 0.01, **0.92**, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01]

* np.argmax عدد **۳** را برمی‌گرداند (چون چهارمین عنصر با اندیس ۳ بیشترین مقدار را دارد).

### ****چرا این خط مهم است؟****

* مدل‌های طبقه‌بندی معمولاً **احتمالات هر کلاس** را خروجی می‌دهند.
* برای تبدیل این احتمالات به **برچسب قطعی** (مثلاً تشخیص عدد "۳") از argmax استفاده می‌کنیم.

### ****پارامتر****axis=1****چه می‌کند؟****

* در آرایه‌های دوبعدی، axis=1 یعنی **عملیات در هر سطر به صورت مستقل** انجام شود:
  + axis=0: مقایسه بین سطرها
  + axis=1: مقایسه بین ستون‌های هر سطر

*# خروجی مدل برای ۳ تصویر (احتمالات ۱۰ کلاس):*

predictions = np.array([

[0.1, 0.0, 0.0, 0.7, 0.0, 0.0, 0.2, 0.0, 0.0, 0.0], *# کلاس ۳*

[0.0, 0.0, 0.9, 0.0, 0.0, 0.0, 0.1, 0.0, 0.0, 0.0], *# کلاس ۲*

[0.0, 0.8, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2, 0.0, 0.0] *# کلاس ۱*

])

predicted\_labels = np.argmax(predictions, axis=1)

*# نتیجه: [3, 2, 1]*