المحور الثالث : خوارزميات التحسين (Optimization Algorithms)

I. مقدمة:

تُعد خوارزميات التحسين من أهم المكونات في مجال **الذكاء الاصطناعي وتعلم الآلة**، حيث تلعب دورًا جوهريًا في **تحسين أداء النماذج** من خلال تقليل الخطأ بين التوقعات والنتائج الحقيقية .الهدف الأساسي من عملية التحسين هو **إيجاد القيم المثلى للمعاملات (Parameters)** التي تجعل النموذج يؤدي بشكل أفضل.

في مجالات مثل **المحاسبة والجباية**، تساعد هذه الخوارزميات على **تحسين تقديرات الإيرادات، كشف الانحر افات المالية، وتحليل البيانات الضريبية** بشكل أكثر دقة وفعالية، مما يرفع من مستوى اتخاذ القرارات المبنية على البيانات.

II. مفهوم وظيفة التكلفة(Cost Function)

1. تعريف

وظيفة التكلفة هي مقياس يحدد مدى دقة النموذج في التنبؤ. يتم تمثيلها عادةً بالرمز J(W)، حيث W تمثل مجموعة المعاملات (Parameters) الخاصة بالنموذج.

الهدف من عملية التدريب هو **تقليل قيمة وظيفة التكلفة** قدر الإمكان. كلما كانت قيمة الدالة أصغر، كلما كان أداء النموذج أفضل.

2. أمثلة على وظائف التكلفة :

- **الانحدار الخطي (Linear Regression)**

$$J(w) = \frac{1}{2m}\sum(\hat{y}_i - y_i)^2$$

حيث:

- m: عدد العينات
- $\hat{y}_i$:القيمة المتوقعة من النموذج
- y_i : القيمة الحقيقية

مثال بسيط:

نفترض أن لدينا نموذجًا يتنبأ بسعر سلعة بناءً على عمرها بالأشهر:

العمر (x)	السعر الحقيقي (y)	السعر المتوقع من النموذج (ŷ)
1	3	2.5
2	4	4.5
3	7	6.0

نريد معرفة مدى "سوء" توقعات النموذج.

نحسب الخطأ (الفارق) لكل نقطة:

x	y	$\hat{y}$	$(\hat{y} - y)^2$
1	3	2.5	$(2.5-3)^2 = 0.25$
2	4	4.5	$(4.5-4)^2 = 0.25$
3	7	6.0	$(6-7)^2 = 1$

مجموع الأخطاء $0.25 + 0.25 + 1 = \mathbf{1.5}$

الخطأ المتوسط (وظيفة التكلفة):

$$J = \frac{1}{2m}\sum(\hat{y} - y)^2 = \frac{1}{2 \times 3} \times 1.5 = 0.25$$

هذا الرقم (0.25) يخبرنا أن النموذج جيد نوعًا ما، لكن يمكن تحسينه أكثر.

- **التصنيف (Classification)**

$$J(w) = -\frac{1}{m}\sum[y_i \times log(\hat{y}_i) + (1 - y_i) \times log(1 - \hat{y}_i)]$$

حيث:

- m: عدد العينات
- $\hat{y}_i$: التوقع الناتج من النموذج

- y_i : الفئة الحقيقية (0 أو 1)

مثال بسيط:

العينة	y(حقيقي)	ŷ(توقع النموذج)	المساهمة في التكلفة
1	1	0.9	$-[1\times\log(0.9)] = 0.105$
2	0	0.1	$-[(1-0)\times\log(1-0.1)] = 0.105$

متوسط الخطأ:

$$J=(0.105+0.1052)/2=0.105$$

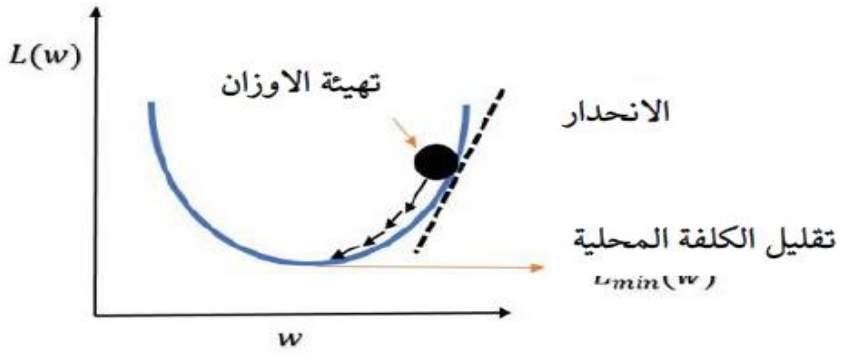
III. خوارزميات التحسين (Optimization Algorithms)

التحسين هو عبارة عن خوارزميات تحاول تقليل دالة الخسارة عن طريق تحسين الأوزان في الشبكة ، أي أن هدفها الرئيسي هو تدريب الشبكات على المشكلة التي نحاول حلها, اي ضبط أوزان الشبكة بحيث تكون قادرة على التعلم . يلعب اختيار خوارزمية التحسين الصحيحة دورًا مهمًا في سرعة تقارب الشبكة و هناك عدة طرق للتحسين.

1. خوارزمية النزول التدرجي(Gradient Descent)

- **الفكرة العامة**

تخيل أنك تقف على جبل وتريد النزول إلى **أدنى نقطة في الوادي** .كل خطوة تأخذها يجب أن تكون **في اتجاه الانحدار** —أي الاتجاه الذي يقل فيه الارتفاع . في الرياضيات، هذا الانحدار يسمى **الميل (Gradient)** كلما نزلت أكثر، تقترب من **القيمة الدنيا لوظيفة التكلفة**.



1 خوارزمية الانحدار

تعتمد خوارزمية **النزول التدرجي** على فكرة بسيطة :نبدأ من نقطة عشوائية، ثم نتحرك **في اتجاه الانحدار (الميل) السالب** لوظيفة التكلفة حتى نصل إلى النقطة التي تكون فيها الدالة **في أدنى قيمة ممكنة**.

- **الصيغة الرياضية**

$$w_j = w_j - \alpha \frac{\partial J(w)}{\partial J(w_j)}$$

حيث:

- α : معدل التعلم(Learning Rate)
- $\frac{\partial J(w)}{\partial J(w_j)}$:المشتقة الجزئية للدالة بالنسبة للمعامل w_j

- **أنواع النزول التدرجي**

1. **النزول التدرجي الكلي:(Batch Gradient Descent)**
   يستخدم جميع بيانات التدريب في كل خطوة تحديث.
2. **النزول التدرجي العشوائي:(Stochastic Gradient Descent)**
   يحدث التحديث بعد كل عينة تدريب واحدة.
3. **النزول التدرجي المصغر:(Mini-Batch Gradient Descent)**
   يستخدم مجموعة صغيرة من العينات في كل تحديث (الأكثر شيوعاً في الممارسة)

- **مثال عملي مبسط (خطوة بخطوة)**

لنفترض أن لدينا نموذجًا بسيطًا جدًا:

$$h(x)=w_0+w_1x$$

ولدينا فقط نقطتان من البيانات:

x	Y
1	2
2	4

نريد أن نعرف القيم المناسبة لـ W_0و W_1بحيث تكون توقعاتنا قريبة من القيم الحقيقية.

الخطوة 1: البداية بقيم عشوائية

نفترض:

$W_0=0 , W_1=0$

إذن التوقعات ستكون:

- للنقطة الأولى : $\hat{y}=0+0\times1=0$
- للنقطة الثانية : $\hat{y}=0+0\times2=0$

الخطوة 2: نحسب الأخطاء

x	y	$\hat{y}$	$(\hat{y}-y)$
1	2	0	-2
2	4	0	-4

الخطوة 3: نحسب الميل(Gradient)

الميل يخبرنا إلى أي اتجاه يجب أن نغيّر القيم.

$$\frac{\partial J(w)}{\partial J(w_0)} = \frac{1}{m}\sum(\hat{y} - y)$$

$$\frac{\partial J(w)}{\partial J(w_1)} = \frac{1}{m}\sum(\hat{y} - y) \times x$$

لدينا m=2:

$$\frac{\partial J(w)}{\partial J(w_0)} = \frac{1}{2}((-2) + (-4)) = -3$$

$$\frac{\partial J(w)}{\partial J(w_1)} = \frac{1}{2}((-2) \times 1 + (-4) \times 2) = -5$$

الخطوة 4: تحديث القيم

نختار معدل تعلم صغير α=0.1

$$w_0 = w_0 - \alpha \frac{\partial J(w)}{\partial J(w_0)}$$

$$w_1 = w_1 - \alpha \frac{\partial J(w)}{\partial J(w_1)}$$

نحسب:

$$w_0 = 0 - 0.1 \times (-3) = 0.3$$

$$w_1 = 0 - 0.1 \times (-5) = 0.5$$

الخطوة 5: نحسب التوقعات الجديدة

x	y	التوقع الجديد $\hat{y}$=0.3+0.5x
1	2	0.8

x	y	التوقع الجديد $\hat{y}=0.3+0.5x$
2	4	1.3

الأخطاء أصبحت أصغر من قبل .إذا كررنا العملية (أي نحسب الميل مرة أخرى ونجري تحديثًا جديدًا)، سنقترب أكثر من القيم الصحيحة (2 و 4).
بعد تكرارات كثيرة، سنجد تقريبًا:

$W_0 \approx 0, W_1 \approx 2$

أي أن النموذج المثالي هو: $h(x)=2x$ وهو يطابق العلاقة الحقيقية بين x و y .

2. خوارزمية Adam (Adaptive Moment Estimation)

- **الفكرة الأساسية:**

خوارزمية **Adam** تعتبر تطويرًا لخوارزميات Gradient Descent التقليدية، وتجمع بين ميزتين رئيسيتين هما:

1. **العزم Momentum :**
   - يساعد على **تسريع عملية التعلم وتقليل الاهتزازات** أثناء النزول في وديان دالة التكلفة.
   - الفكرة: لا تعتمد فقط على التدرج الحالي، بل تأخذ في الاعتبار **اتجاه التدرجات السابقة** لتسريع التحرك في الاتجاه الصحيح.
2. **جذر متوسط التربيع للتدرجات RMSProp :**
   - يقوم **بتعديل معدل التعلم لكل معامل** حسب تغيراته السابقة.
   - الفكرة: إذا كانت التدرجات كبيرة، يقلل معدل التعلم لتجنب القفز الكبير؛ وإذا كانت صغيرة، يزيد معدل التعلم لتسريع التعلم.

- **مقارنة Adam و Gradient Descent التقليدي:**

المعيار	Adam	Gradient Descent
معدل التعلم	متكيف لكل معامل حسب التدرجات	ثابت لجميع المعاملات

المعيار	Adam	Gradient Descent
سرعة التعلم	أسرع وأكثر كفاءة	أبطأ
الاستقرار	أكثر استقرارًا، يقلل الاهتزازات	أقل استقرارًا مع الشبكات العميقة
الاستخدام النموذجي	الشبكات العصبية الكبيرة والعميقة	النماذج البسيطة

خلاصة : Adam هو الخيار القياسي لمعظم التطبيقات العملية، بينما Gradient Descent التقليدي مفيد للتجارب التعليمية أو النماذج البسيطة.

IV. أشهر دوال التنشيط (Activation Functions)

- **Sigmoid**

$$\sigma(x) = sigmoid(x) = \frac{1}{1 + e^{-x}}$$

- **المميزات** :تحوّل القيم إلى ال المجال [0,1]، مناسبة للنواتج الاحتمالية.
- **العيوب** :مشكلة **Vanishing Gradient** عند القيم الكبيرة جداً أو الصغيرة جداً.

- **Softmax**

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

- **المميزات** :تحويل متجه القيم إلى احتمالات (مجموعها = 1)، تستخدم في تصنيف متعدد الفئات.
- **العيوب** :غير مناسبة للطبقات المخفية.

- **Tanh (Hyperbolic Tangent)**

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- **المميزات** :خرّاج بين [-1,1]، مركزية حول الصفر ⟵ أفضل من sigmoid للطبقات المخفية.
- **العيوب** :ما زالت تعاني من **Vanishing Gradient**

- **ReLU (Rectified Linear Unit)**

$$ReLU(x) = \max(0, x)$$

- **المميزات** :سريعة، تقلل مشكلة Vanishing Gradient.
- **العيوب** :قد تحدث مشكلة (**Dead Neurons** قيمة سالبة دائمًا ⟵ تخرج 0)

- **Leaky ReLU**

$$\text{Leaky ReLU}(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}$$

- **المميزات** :تحل مشكلة **Dead Neurons** بإعطاء قيمة صغيرة للقيم السالبة.
- **العيوب** :يحتاج لاختيار α مناسب.

- **Swish**

$$\text{Swish}(x) = x \cdot \sigma(x) = x \cdot \frac{1}{1 + e^{-x}}$$

- **المميزات** :سلس، يعطي أداء أفضل من ReLU في بعض الشبكات العميقة.
- **العيوب** :أكثر تعقيدًا حسابيًا من ReLU

- **ELU (Exponential Linear Unit)**

$$\text{ELU}(x) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1), & x \leq 0 \end{cases}$$

- **المميزات** :يساعد على تقليل مشكلة Vanishing Gradient ، يعطي خرّاج سالب للقيم السالبة ⟵ أسرع تقارب.
- **العيوب** :يحتاج إلى حساب الأُسّي ⟵ أثقل حسابيًا.

- **SELU (Scaled ELU)**

$$\mathrm{SELU}(x) = \lambda \cdot \mathrm{ELU}(x)$$

- **المميزات** :يسمح بـ **Self-Normalizing Neural Networks** يحافظ على المتوسط والانحراف المعياري عبر الطبقات تلقائيًا.
- **العيوب** :يعمل بشكل أفضل مع تهيئة محددة للوزن وبيانات معينة.

ملاحظة :

عندما يتم تدريب الشبكة العصبية باستخدام خوارزمية **Backpropagation**، يتم حساب **التدرجات (Gradients)**من الطبقة الأخيرة نحو الطبقات الأولى.

لكن في الشبكات العميقة جدًا (عدد كبير من الطبقات)، يحدث التالي:

- التدرجات تصبح **أصغروأصغر** كلما رجعنا إلى الوراء عبر الطبقات.
- في بعض الأحيان تنخفض إلى قيم **قريبة جدًا من الصفر**.
- هذا يجعل الطبقات الأولى **تقريبًا لا تتعلم شيئًا**.

هذه الظاهرة تُسمى:
= Vanishing Gradient تلاشي التدرجات

V. الخاتمة

- **Gradient Descent** هي الطريقة الأساسية لتحديث المعاملات تدريجيًا.
- **Adam**تحسن من Gradient Descent عبر **Momentum**و **RMSProp** لتسريع التعلم وزيادة الاستقرار.
- اختيار **دالة التنشيط** يؤثر على أداء الشبكة، وهناك دوال تناسب الطبقات المخفية وأخرى الطبقة الأخيرة.

- في التطبيقات العملية، غالبًا ما نستخدم **Adamمع ReLU أو ELU للطبقات المخفية و Softmaxللطبقة الأخيرة في التصنيف متعدد الفئات**.