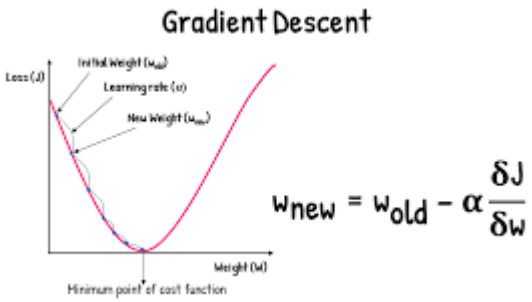
Question 1

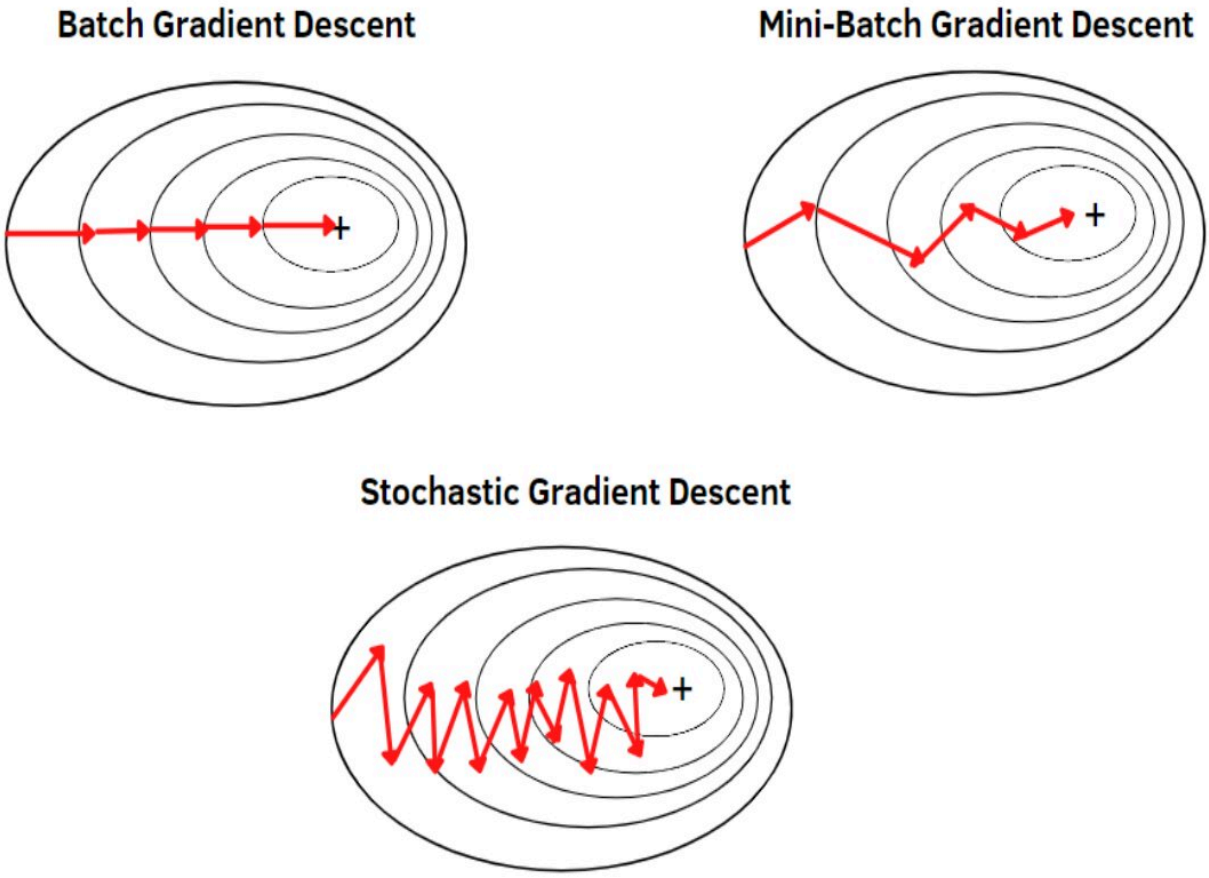
Gradient Descent (GD): Batch Gradient Descent; Stochastic Gradient Descent (SGD); Mini-Batch Gradient Descent



Mục tiêu của Gradient Descent là tối thiểu hóa hàm mất mát (Loss Function) bằng cách điều chỉnh các tham số (trọng số và bias) của mô hình để dự đoán ngày càng chính xác hơn

Gradient Descent giúp ta xác định được hướng và tốc độ thay đổi của hàm mất mát

a này quá lớn thì giúp tốc độ nhanh nhưng sẽ dễ bị lòng vòng quanh điểm cực tiểu, khó chính xác hơn a quá nhỏ thì quá trình chạy rất lâu, nhưng mô hình sẽ chính xác hơn tiến gần hơn đến giá trị cực tiểu



Batch Gradient Descent

Gradient được tính toán dựa trên toàn bộ tập dữ liệu tại mỗi bước cập nhật.

- Công thức cập nhật:

$$w := w - \eta \cdot \nabla J(w)$$

Trong đó:

- $\nabla J(w) = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial J(w)}{\partial w}$: Gradient trung bình trên toàn bộ tập dữ liệu.
- n: Số lượng mẫu trong tập dữ liệu.

Quy trình:

- Duyệt qua toàn bộ dữ liệu.
- Tính toán hàm mất mát và gradient trên toàn bộ tập dữ liệu.
- Cập nhật tham số w.
- Lặp lại quy trình cho đến khi hội tụ.

Ưu điểm:

- Gradient mượt mà và chính xác (do tính toán dựa trên toàn bộ dữ liệu). Thích hợp với dữ liệu nhỏ.

Nhược điểm:

- Rất chậm khi dữ liệu lớn, vì mỗi lần cập nhật cần tính toán trên toàn bộ dữ liệu.

Stochastic Gradient Descent (SGD)

Gradient được tính toán và cập nhật sau mỗi mẫu dữ liệu.

- Công thức cập nhật:

$$w := w - \eta \cdot \nabla J(w; x^{(i)})$$

Trong đó:

- $x^{(i)}$: Một mẫu dữ liệu ngẫu nhiên từ tập dữ liệu.
- $\nabla J(w; x^{(i)})$: Gradient tính dựa trên một mẫu.

Quy trình:

- Chọn ngẫu nhiên một mẫu $x(i)$từ tập dữ liệu.
- Tính toán hàm mất mát và gradient trên mẫu đó.
- Cập nhật tham số w.
- Lặp lại quy trình với một mẫu khác cho đến khi hội tụ.

Ưu điểm:

- Rất nhanh (do chỉ tính gradient trên một mẫu tại mỗi bước).
- Có thể thoát khỏi local minima nhờ tính ngẫu nhiên.

Nhược điểm:

- Gradient dao động mạnh, dẫn đến hội tụ không ổn định.
- Thích hợp hơn với dữ liệu lớn hoặc thời gian thực.

Mini-Batch Gradient Descent

Gradient được tính toán trên một nhóm nhỏ dữ liệu (mini-batch) thay vì toàn bộ dữ liệu hoặc một mẫu duy nhất.

Công thức cập nhật:

$$w := w - \eta \cdot \nabla J(w; X^{(batch)})$$

Trong đó:

- $X^{(batch)}$: Một tập nhỏ gồm m mẫu dữ liệu (mini-batch).
- $\nabla J(w; X^{(batch)}) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial J(w)}{\partial w}$: Gradient trung bình trên mini-batch.

Quy trình:

- Chia dữ liệu thành các nhóm nhỏ (mini-batches). Với mỗi mini-batch:
- Tính toán hàm mất mát và gradient trên mini-batch.

- Cập nhật tham số w.
- Lặp lại quy trình cho đến khi hội tụ.

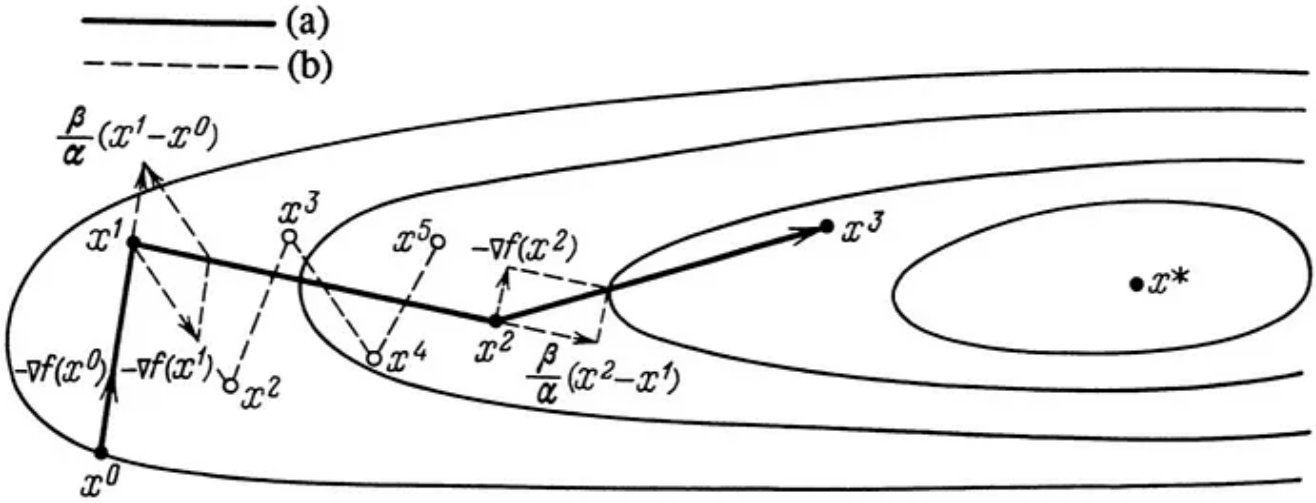
Ưu điểm:

- Cân bằng giữa tốc độ (nhanh hơn Batch GD) và ổn định (ít dao động hơn SGD).
- Phù hợp với hầu hết các bài toán thực tế.
- Tận dụng tốt tài nguyên GPU/TPU nhờ xử lý trên batch.

Nhược điểm:

- Phụ thuộc vào kích thước mini-batch:
- Mini-batch quá nhỏ → Gần giống SGD (dao động lớn).
- Mini-batch quá lớn → Gần giống Batch GD (chậm).

Momentum



Momentum được thêm vào Gradient Descent bằng cách sử dụng một vector vận tốc v để lưu giữ động lượng(một phần thông tin gradient của các bước trước đó để cải thiện việc cập nhật tham số w)

$$v := \beta v - \eta \nabla J(w)$$

- v: Vận tốc (vector lưu động lượng).
- β: Hệ số động lượng (Momentum term), thường nằm trong khoảng 0.9 đến 0.99.
- η: Tốc độ học (Learning rate).
- $\nabla J(w)$: Gradient của hàm mất mát.

Tăng tốc hội tụ:

Trong các rãnh hẹp của hàm mất mát, Gradient Descent thông thường sẽ dao động chậm chạp vì gradient thay đổi nhỏ ở hướng chính (hướng dốc xuống) nhưng lớn ở hướng vuông góc.

Momentum tăng tốc di chuyển theo hướng dốc xuống nhờ tích lũy động lượng.

Giảm dao động:

Ở các vùng mà gradient đổi hướng nhanh (ví dụ: gần điểm tối ưu), Momentum giúp làm giảm dao động nhờ trọng số β áp dụng lực cản ở các bước không cần thiết.

Backpropagation learning rate

Đây là các phương pháp tối ưu hóa trong học máy và học sâu, tự động điều chỉnh tốc độ học tập (α) dựa trên:

- Gradient hiện tại.
- Lịch sử gradient của các tham số.

Khác với cách tiếp cận cố định hoặc giảm tốc độ học tập dựa trên lập lịch, các bộ tối ưu này điều chỉnh tốc độ học tập cho từng tham số riêng biệt một cách thích ứng, giúp mô hình hội tụ nhanh hơn và ổn định hơn.

Adaptive Gradient Algorithm:

- Tăng tốc độ học tập cho các tham số ít thay đổi, giảm tốc độ cho các tham số thường xuyên thay đổi.
- Sử dụng tổng bình phương gradient tích lũy để điều chỉnh.

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{G_t + \epsilon}} \cdot \nabla J(\theta_t)$$

- G_t: Tổng bình phương gradient từ bước đầu tiên đến t.
- ϵ: Giá trị nhỏ để tránh chia cho 0.

ưu điểm: Tự động điều chỉnh tốc độ học tập.

nhược điểm: Gradient tích lũy quá lớn có thể làm giảm tốc độ học tập quá mức (vấn đề hội tụ chậm).

Root Mean Square Propagation: Khắc phục nhược điểm của Adagrad bằng cách sử dụng trung bình động theo mũ của gradient.

$$E[g^2]_t = \beta \cdot E[g^2]_{t-1} + (1 - \beta) \cdot g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{E[g^2]_t + \epsilon}} \cdot \nabla J(\theta_t)$$

- $E[g^2]_t$: Trung bình động theo mũ của gradient bình phương.
- β: Hệ số trung bình động (thường chọn $\beta = 0.9$).

Ưu điểm: Giữ tốc độ học tập ổn định hơn Adagrad.

Nhược điểm: Cần tinh chỉnh β phù hợp.

Adaptive Moment Estimation: Kết hợp Momentum (trung bình động của gradient) và RMSprop (trung bình động của gradient bình phương).

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t$$

- m_t, v_t: Trung bình động của gradient và gradient bình phương.
- β_1, β_2: Hệ số trung bình động (thường chọn $\beta_1 = 0.9, \beta_2 = 0.999$).

Ưu điểm: Mạnh mẽ và phổ biến nhất. Thích hợp với dữ liệu lớn và mạng phức tạp.

Nhược điểm: Dễ bị lệ thuộc vào các siêu tham số β1 và β2.

Thì các phương pháp của Adaptive Learning Rate Optimizers thì tuy là có tốc độ học tập a tổng quát nhưng nó lại thay đổi khác nhau dựa trên các điểm neuron khác nhau trong lớp, mỗi điểm sẽ có mỗi a dựa theo gradient và lịch sử gradient

Code

[ ] ↳ *4 ô bị ẩn*

Ques 2

[ ] ↳ *34 ô bị ẩn*

Ques 3

1. Định nghĩa học máy sâu

Học máy sâu (Deep Learning) là một nhánh của học máy (Machine Learning) dựa trên các mạng nơ-ron nhân tạo (Artificial Neural Networks) có nhiều lớp (multi-layer). Các mô hình học sâu được thiết kế để tự động trích xuất và học các đặc trưng từ dữ liệu, thường dùng để giải quyết

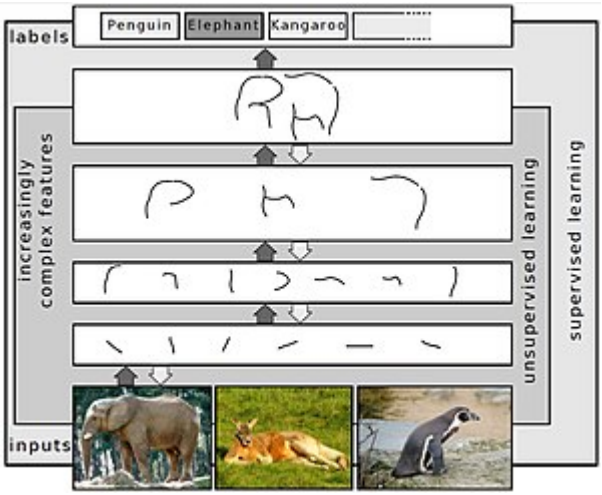
các bài toán phức tạp như nhận diện hình ảnh, xử lý ngôn ngữ tự nhiên (NLP), và dự đoán chuỗi thời gian.

Mạng nơ-ron nhân tạo (Artificial Neural Network): Là mô phỏng hoạt động của các tế bào thần kinh trong não bộ con người. Một mạng nơ-ron nhân tạo gồm:

Neuron (nút): Đơn vị cơ bản xử lý thông tin. Kết nối (Connection): Các trọng số (weights) liên kết các neuron, giúp điều chỉnh tín hiệu. Lớp (Layer): Bao gồm nhiều neuron cùng hoạt động. Học sâu khác biệt với học máy truyền thống bởi có nhiều lớp xử lý thông tin hơn, tạo thành kiến trúc "sâu".

Tại sao "deep" (sâu)?

Thay vì chỉ có một hoặc hai lớp, mạng học sâu có thể có hàng chục hoặc hàng trăm lớp, cho phép học và biểu diễn các mối quan hệ phức tạp hơn trong dữ liệu.



2. Cấu trúc cơ bản của học sâu

Học sâu sử dụng mạng nơ-ron nhiều lớp. Mỗi mạng nơ-ron bao gồm:

- Lớp đầu vào (Input Layer): Tiếp nhận dữ liệu thô (ví dụ: hình ảnh, văn bản, âm thanh).
- Lớp ẩn (Hidden Layers): Các lớp trung gian thực hiện tính toán phức tạp. Đây là nơi các đặc trưng được trích xuất tự động.
- Lớp đầu ra (Output Layer): Cung cấp kết quả dự đoán (ví dụ: phân loại hình ảnh thành mèo/chó).

Mỗi "nơ-ron" trong mạng thực hiện phép tính:

$$z = w \cdot x + b$$

Sau đó áp dụng hàm kích hoạt (activation function) để quyết định đầu ra của nơ-ron, như:

ReLU (Rectified Linear Unit):

$f(x)=max(0,x)$

Sigmoid:

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

1.Mạng nơ-ron tích chập (Convolutional Neural Network - CNN)

CNN được thiết kế đặc biệt để xử lý dữ liệu có cấu trúc dạng lưới, như hình ảnh. Điểm mạnh của CNN là khả năng trích xuất đặc trưng không gian (spatial features) bằng các phép tích chập (convolution)

Thành phần chính của CNN:

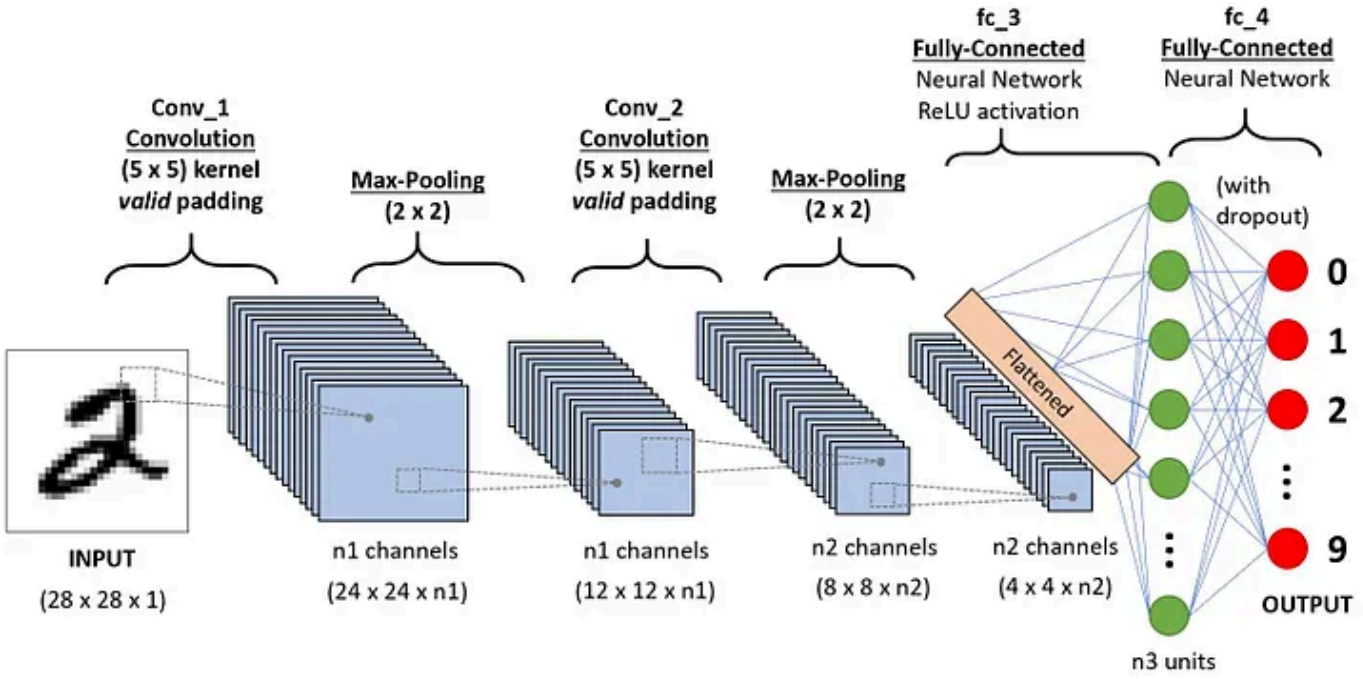
- Lớp tích chập (Convolutional Layer): Áp dụng các bộ lọc (filters) để trích xuất đặc trưng như cạnh, góc, và cấu trúc hình ảnh.
- Lớp gộp (Pooling Layer): Giảm kích thước dữ liệu, giữ lại đặc trưng quan trọng (ví dụ: MaxPooling).
- Lớp kết nối đầy đủ (Fully Connected Layer): Liên kết các đặc trưng đã trích xuất để thực hiện dự đoán.

Ví dụ ứng dụng:

- Nhận diện khuôn mặt: Phát hiện khuôn mặt trong ảnh hoặc video.
- Phân loại hình ảnh: Dự đoán một bức ảnh thuộc loại nào (mèo/chó).
- Phát hiện vật thể: Xác định vị trí và loại vật thể trong ảnh (dùng kiến trúc YOLO, Faster-RCNN).

Kiến trúc nổi bật:

- LeNet (1998): Một trong những CNN đầu tiên, dùng để nhận diện chữ số viết tay.
- ResNet (2015): Mạng CNN sâu với cơ chế kết nối dư (Residual Connections) để tránh mất mát thông tin.



2. Bộ nhớ dài hạn-ngắn hạn (Long Short-Term Memory - LSTM)

LSTM là một dạng của mạng nơ-ron hồi tiếp (Recurrent Neural Network - RNN), được thiết kế để xử lý dữ liệu tuần tự như chuỗi thời gian, văn bản, hoặc âm thanh. LSTM giải quyết vấn đề "vanishing gradient", một hạn chế lớn của RNN truyền thống.

Cấu trúc chính:

LSTM sử dụng các cổng (gates) để kiểm soát luồng thông tin:

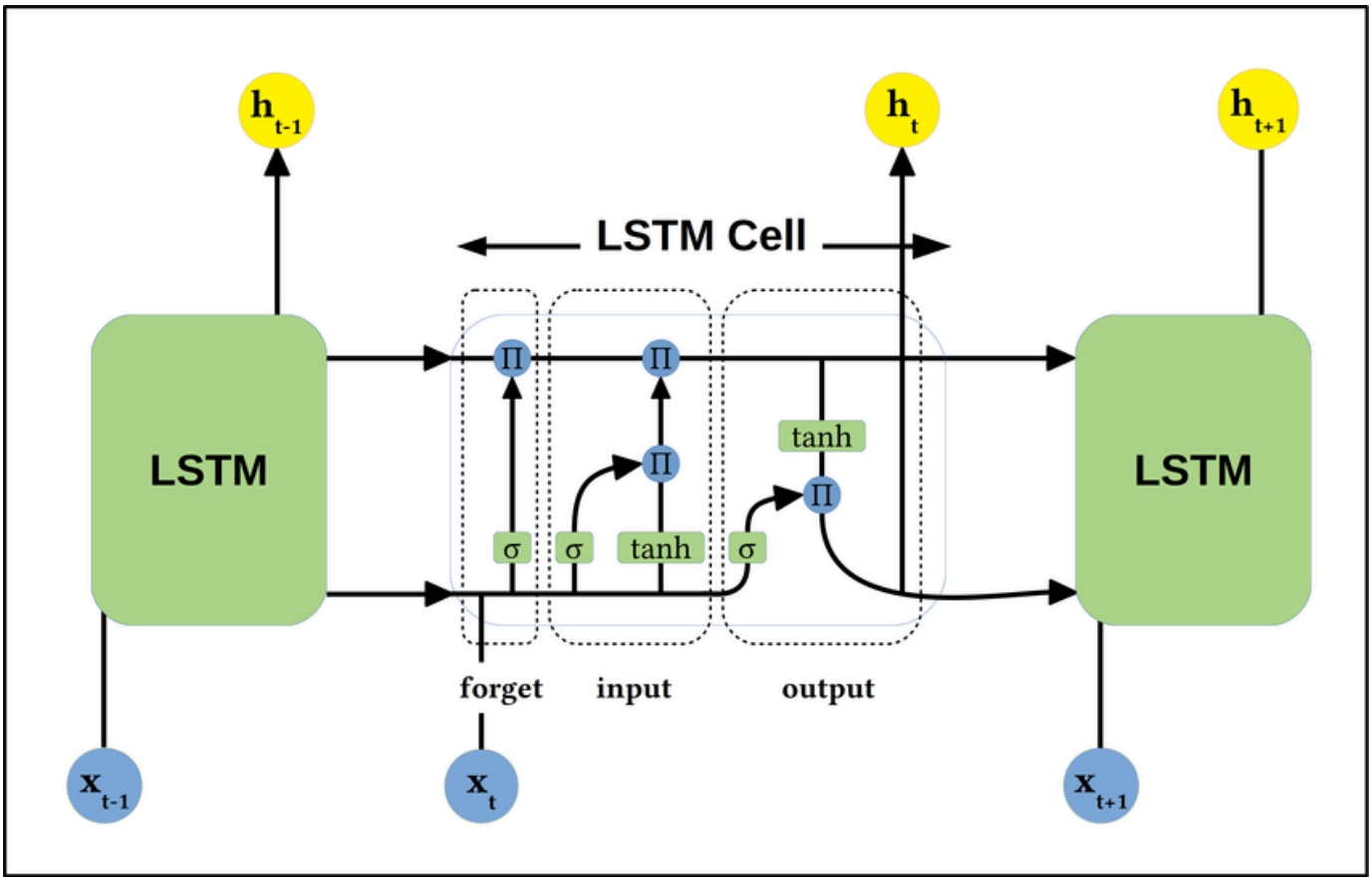
- Cổng quên (Forget Gate): Quyết định thông tin nào sẽ bị loại bỏ.
- Cổng đầu vào (Input Gate): Quyết định thông tin nào được thêm vào trạng thái.
- Cổng đầu ra (Output Gate): Xác định phần nào của trạng thái được xuất ra.

Ví dụ ứng dụng:

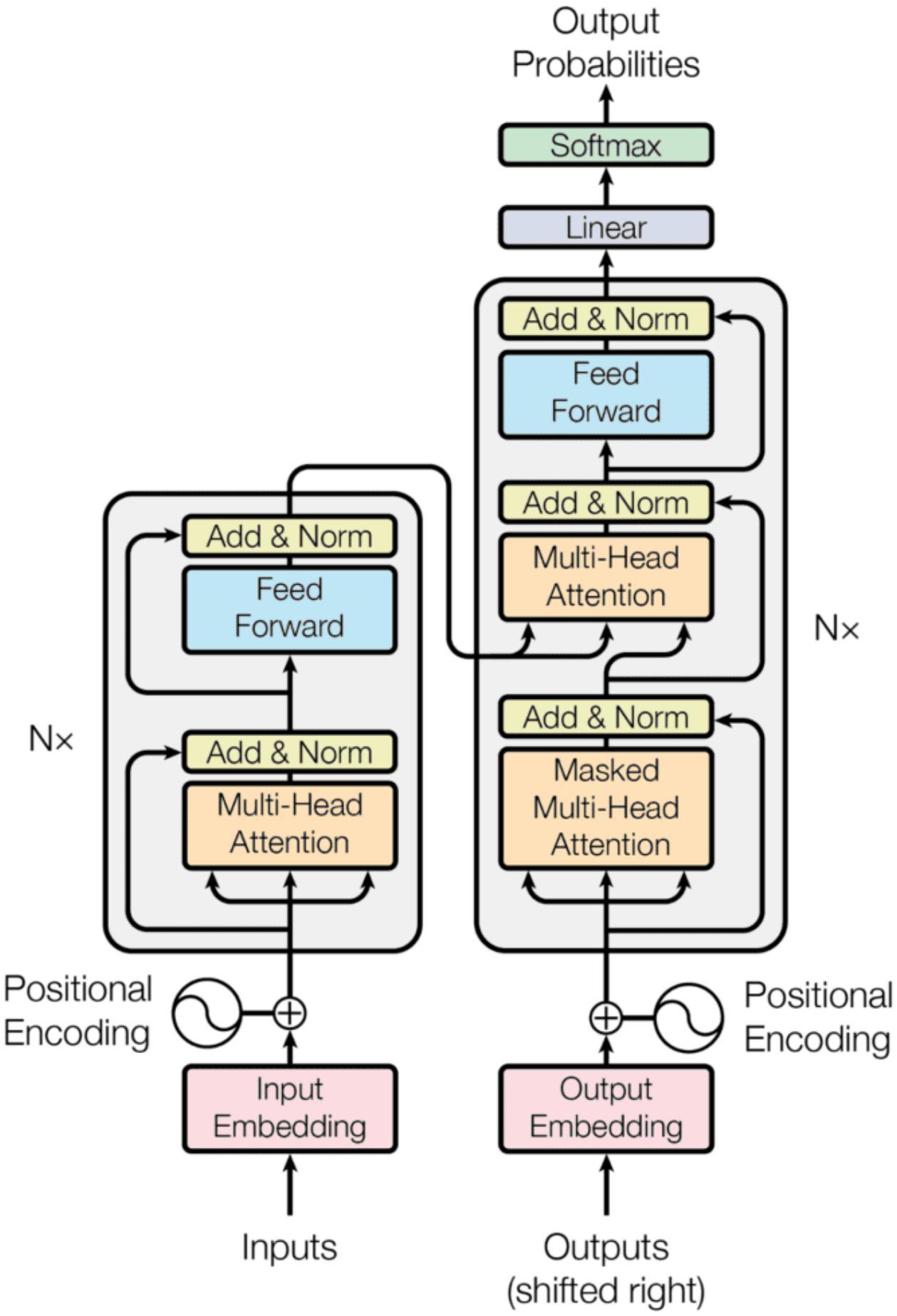
- Dịch máy: Chuyển đổi văn bản từ ngôn ngữ này sang ngôn ngữ khác.
- Dự đoán chuỗi thời gian: Dự báo giá cổ phiếu, lưu lượng giao thông.
- Phân loại cảm xúc: Phân tích cảm xúc trong văn bản (tích cực, tiêu cực).

Kiến trúc cải tiến:

- GRU (Gated Recurrent Unit): Một phiên bản đơn giản hơn của LSTM với hiệu suất tương đương.



3. Bộ biến áp (Transformer)


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)