1. **Perceptron:**

- **Khái niệm:**

○ Perceptron là một nút tính toán đơn giản trong mạng nơ-ron, làm nhiệm vụ phân biệt hai lựa chọn như có/không, đúng/sai, bằng cách tổng hợp các tín hiệu đầu vào đã được gán mức quan trọng khác nhau.

○ Sau khi cộng các tín hiệu theo mức quan trọng, perceptron áp dụng một hàm quyết định đơn giản (hàm kích hoạt dạng bước) để đưa ra kết quả 1 hoặc 0, rồi chuyển kết quả đó sang lớp kế tiếp nếu nằm trong một mạng nhiều lớp.

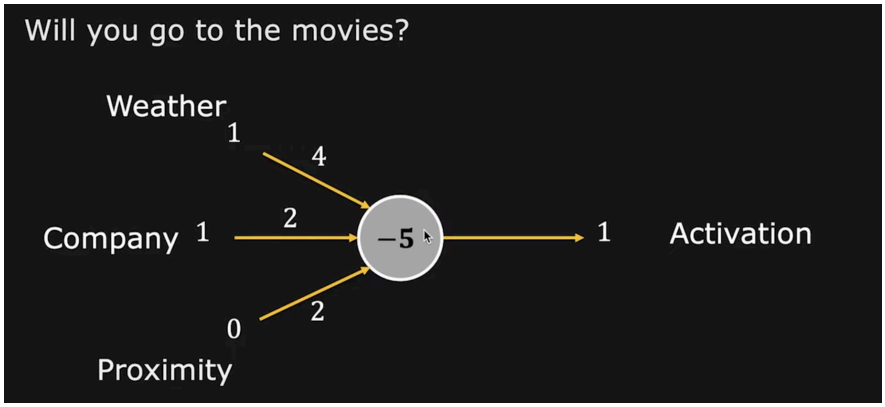
- **Ví dụ:** Bạn cần quyết định là có đi xem phim không.

  ■ Đầu vào: x1 (thời tiết), x2 (có bạn), x3 (rạp gần), mỗi giá trị 0 hoặc 1.

  ■ Trọng số: w1=4, w2=2, w3=2; bias b=−5.

  ■ Z = 4 x1 + 2 x2 + 2 x3 -5

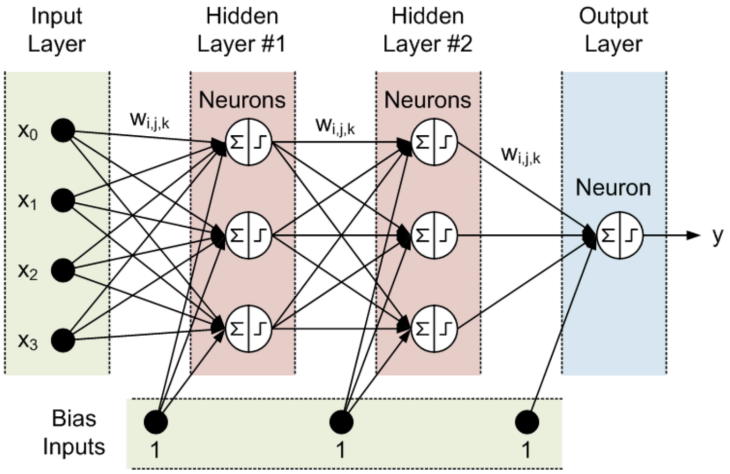
=> z = 4.1 + 2.1 + 2.0 - 5 = 1=> y = 1 (đi)



- **Liên hệ:**

  - Là thành phần cơ bản tạo ra Hidden layer và mạng neutron.
  - Để giúp mô hình học các quan hệ phi tuyến tính thì cần Activation Function (Sigmoid, ReLU, Tanh,...)

2. Hidden Layer:



- **Khái niệm:** là các tầng nằm giữa đầu vào và đầu ra, không được dán nhãn trực tiếp bởi dữ liệu, nên hành vi của chúng do thuật toán học tự tìm ra để giúp lớp cuối cho kết quả đúng. Chúng tạo ra "đặc trưng" mới từ dữ liệu thô: thay vì dự đoán thẳng từ đầu vào, mô hình học cách biến đổi đầu vào qua một hay nhiều lớp ẩn rồi mới dự đoán ở lớp cuối.
- **Ví dụ: Hãy tưởng tượng nấu ăn**

  Nguyên liệu là đầu vào, món ăn hoàn chỉnh là đầu ra, còn các bước sơ chế-nấu nướng ở giữa chính là lớp ẩn; công thức chỉ ghi món cuối ra sao, chứ không ghi chi tiết từng thao tác nhỏ, nên các bước ở giữa "tự" hình thành sao cho ra món đúng.
- **Liên hệ:**
  - Các trọng số các unit trong hidden layer được cập nhật bằng Gradient Descent.
  - Hidden layer được tạo bởi nhiều Perceptron.

3. Memory based learning

- **Khái niệm:** là cách học dựa trên "ghi nhớ ví dụ": hệ thống giữ lại các ví dụ đã thấy và khi gặp dữ liệu mới thì so sánh xem nó giống ví dụ nào nhất để gán nhãn hoặc dự đoán, thay vì học ra quy tắc tổng quát.
- **Ví dụ:** Dự đoán điểm cho học viên mới: dùng các đặc trưng đơn giản (thời lượng học, số bài luyện) so với học viên cũ; điểm dự đoán là trung bình của nhóm giống nhất.
- **Liên hệ:**
    - Có sự liên hệ với thuật toán KNN vì cũng tương tự cách hoạt động là dự đoán dựa trên so sánh dữ liệu đã quan sát.
    - Có thể kết hợp để so sánh độ giống nhau các samples dựa trên Distance Metrics (Euclidean, Cosine).

4. Gradient descent

- **Khái niệm:** Là cách tối ưu hoá giống như đi xuống dốc để giảm "lỗi": ở mỗi bước, ta đứng tại vị trí hiện tại, xác định hướng dốc nhất khiến lỗi tăng nhanh nhất, rồi bước một bước nhỏ theo hướng ngược lại để lỗi giảm nhanh nhất; lặp lại liên tục cho đến khi đến gần "đáy", nơi lỗi hầu như không giảm thêm.
- **Ví dụ:** Hãy hình dung bạn đang đứng trên sườn đồi và muốn xuống chân đồi nhanh nhất: ở mỗi bước, bạn quan sát hướng nào dốc nhất nếu đi lên sẽ mệt nhất, rồi bước một bước nhỏ theo hướng ngược lại để xuống dốc; cứ lặp lại việc chọn hướng dốc nhất và bước xuống một chút, cho đến khi gần chạm đáy, lúc mà dù có bước tiếp thì cũng không xuống thêm được bao nhiêu.

- **Liên hệ:**
    - Có nhiều biến thể: Stochastic Gradient Descent, Mini-batch GD, Adam nhằm giúp cải thiện tốc độ hội tụ.
    - Gradient Descent được sử dụng để tối ưu Loss Function.

5. Loss Function:

- **Khái niệm:** Là "điểm phạt" cho biết dự đoán sai bao nhiêu so với sự thật (dự đoán càng lệch thì điểm phạt càng lớn, dự đoán khớp hoàn hảo thì điểm phạt gần 0), nhờ vậy mục tiêu "dự đoán tốt" được biến thành một con số cụ thể để so sánh giữa các lần thử hay giữa các mô hình;
- **Ví dụ:** ví dụ nếu giá trị đúng là 10 nhưng dự đoán là 8, ta có thể đặt điểm phạt là 2 (chênh lệch tuyệt đối) hoặc 4 (bình phương chênh lệch), cả hai đều phản ánh mức sai và khuyến khích các dự đoán gần đáp án hơn.
- **Liên hệ:**
    - Các bài toán các đặc điểm khác nhau thì sẽ cần Loss Function khác nhau.

BÀI TẬP NEUTRAL NETWORK

Quá trình huấn luyện và kết quả:

Mô hình là mạng neural với tổng số 1.313.539 tham số.

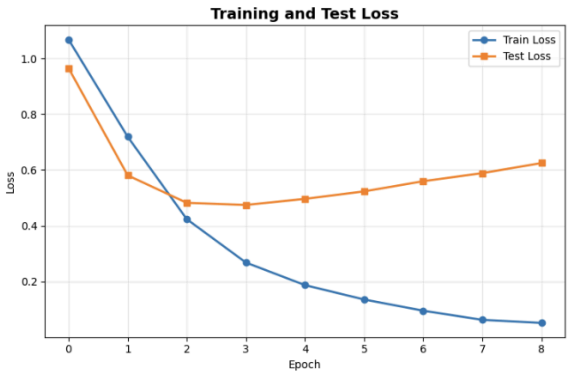
Dữ liệu được tiền xử lý bằng phương pháp TF-IDF với tokenizer tiếng Việt, đặc trưng tối đa 5000.

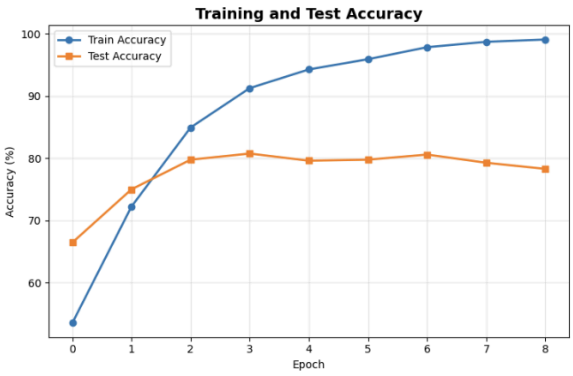
Huấn luyện trên thiết bị GPU với batch size 32 và sử dụng mạng gồm 3 lớp fully connected, có dropout giữa các lớp.

Sử dụng hàm mất mát CrossEntropyLoss và optimizer Adam với learning rate 0.0005 kèm weight decay 1e-4.

Áp dụng cơ chế giảm learning rate khi độ chính xác trên tập kiểm tra không cải thiện và kỹ thuật early stopping với patience 5 epochs.

Kết quả huấn luyện:






```
================================================
TRAINING SUMMARY
================================================
Best test accuracy: 80.76%
Final train accuracy: 99.10%
Final test accuracy: 78.29%
Best epoch: 4
Total parameters: 1,313,539
================================================
```

Huấn luyện trong tối đa 20 epochs, nhưng điểm dừng sớm tại epoch 9 do không cải thiện độ chính xác trên tập kiểm tra.

Độ chính xác tốt nhất trên tập kiểm tra đạt khoảng 80.76% tại epoch 4.

Độ chính xác cuối cùng trên tập kiểm tra là 78.29%.

Độ chính xác cuối cùng trên tập huấn luyện rất cao, đạt 99.10%, cho thấy hiện tượng overfitting.

Loss trên tập huấn luyện giảm mạnh trong quá trình huấn luyện, trong khi loss trên tập kiểm tra giảm ban đầu nhưng sau đó tăng nhẹ, cũng biểu thị mô hình bắt đầu quá khớp sau một số epoch.

Tóm lại, mô hình đã học tốt dữ liệu huấn luyện nhưng hiệu quả tổng quát còn hạn chế do overfitting, việc áp dụng early stopping giúp đạt được hiệu suất tốt nhất trên tập kiểm tra. Để cải thiện, có thể xem xét điều chỉnh thêm kỹ thuật regularization hoặc tăng cường dữ liệu.