Day 01. Introduction to Deep Learning and Neural Networks

Lesson of Content

1. Basic Concepts of ML/DL

- Neural Networks
- Activation Functions
- Fully Connected Layer

2. Loss Functions

- MSE (Mean Squared Error)
- Cross-Entropy Loss

3. Optimization Methods

- Backward and Gradient Descent
- SGD (Stochastic Gradient)
- Momentum
- Adam

4. Learning Rate Schedulers

- Step Decay
- Exponential Decay
- Cosine Annealing

Lesson of Content

5. Evaluation Metrics

- Confusion Matrix
- Precision/Recall/Accuracy
- F1-Score

6. Common Problems

- Imbalanced Data
- Underfitting and Overfitting
- Vanishing and Exploding Gradients

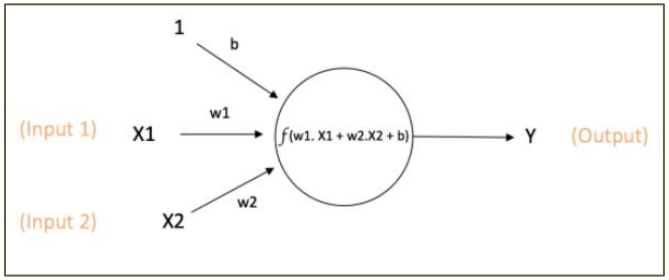
7. Regularization

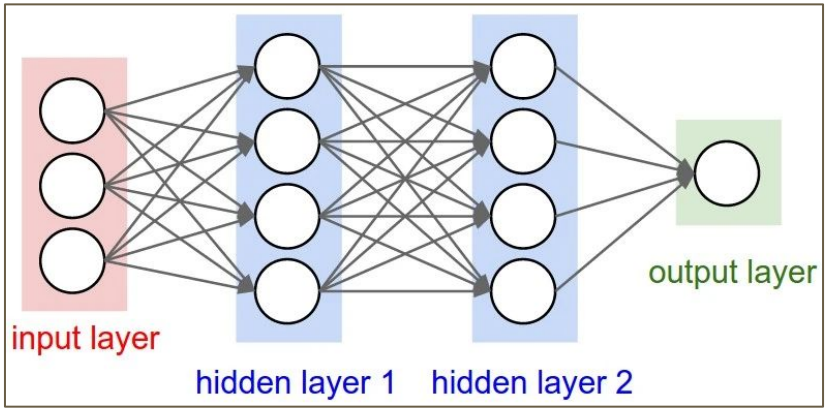
- L1, L2 and Elastic Net
- Dropout
- Early Stopping
- Data Augment

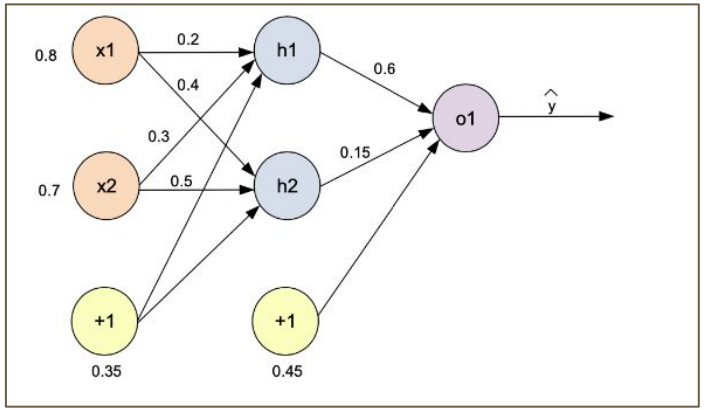
8. Computer Vision and Application

Neural Networks

Mạng Neural Network có ba layers chính: Input Layer, Hidden Layer and Output Layer.



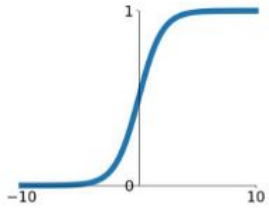




Activate Functions

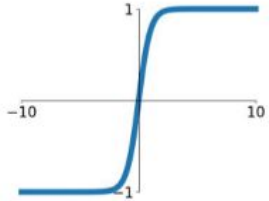
Sigmoid

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



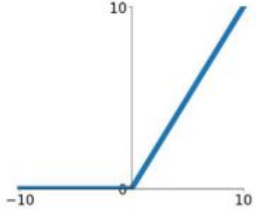
tanh

$$\tanh(x)$$



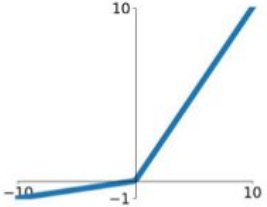
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

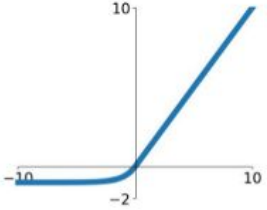


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Why using Activate Functions ?

Nonlinearity: Nếu các hàm Activate là hàm tuyến tính thì kết hợp các hàm tuyến tính với nhau thì cũng sẽ ra một hàm tuyến tính, dẫn đến là mạng có sâu hơn đi nữa thì nó cũng vô nghĩa.
Dữ liệu thực tế thường phức tạp và không tuyến tính, việc dùng các hàm phi tuyến giúp biểu diễn được các mô hình phức tạp hơn.

GateKeeping: Giúp kiểm soát thông tin chảy qua mạng. Chúng quyết định thông tin nào sẽ được truyền qua lớp kế tiếp, giúp mô phỏng quá trình kiểm soát thông tin. Ví dụ ReLU đặt tất các các giá trị đầu vào bằng 0 bằng loại bỏ chúng.

Hàm số tuyến tính:

Một hàm số $f(x)$ được gọi là tuyến tính nếu nó thoả mãn hai tính chất sau:

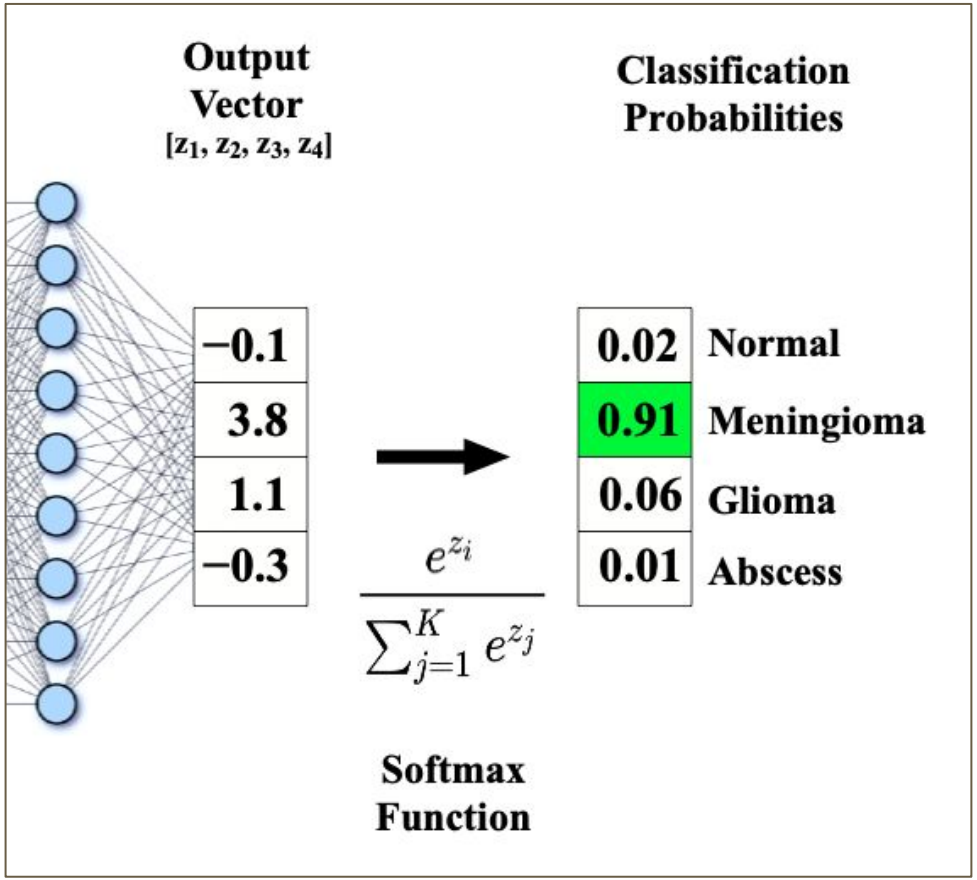
a) Tính chất cộng: $f(x+y) = f(x) + f(y)$ cho mọi x, y.

b) Tính chất nhân với hằng số: $f(\alpha x) = \alpha f(x)$ cho mọi x và mọi số thực α.

Hàm số tuyến tính đơn giản nhất có dạng $f(x) = ax$, với a là một hằng số.

Softmax Function

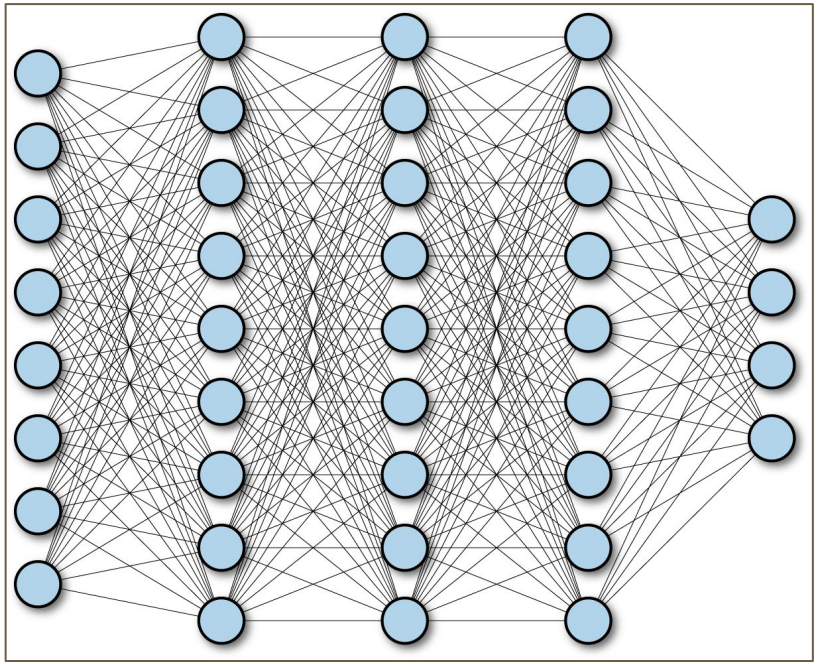
- Phân loại đa lớp



Fully Connected Layer

Là một mạng Neural đầy đủ.
Thường dùng cho các mục đích :

- **Resizing:** Dùng để chuyển đổi kích thước của dữ liệu đầu vào thành kích thước ngõ ra mong muốn.
- **Classification:** Trong một số tác vụ phân loại, lớp FC cuối thường bằng với số lượng class. Và kết hợp với softmax, giúp tính xác suất dự đoán phân loại cho từng class.
- **Feature Combination:** Thường FC đặt sau khi feature extraction, nó giúp các feature được kết hợp với nhau, giúp mô hình hiểu được mối quan hệ giữa các feature.

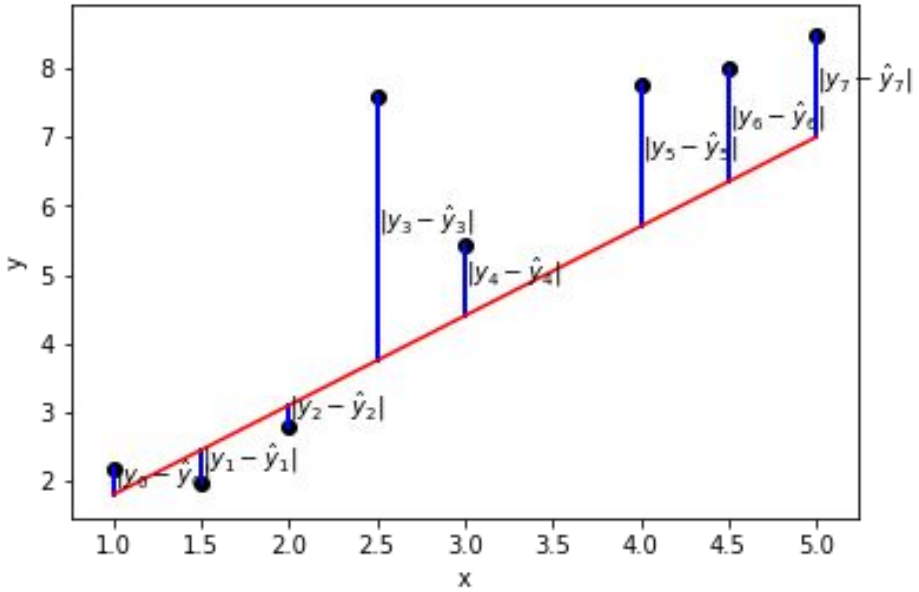


MSE (Mean Squared Error)

Thường dùng cho **Regression Problems**. Mục đích để giảm thiểu bình phương sai số giữa giá trị dự đoán và giá trị thực tế.

Mean Error Squared

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$



Cross-Entropy Loss

Để tính độ sai khác giữa hai phân phối P và Q

$$D_{KL}(P \parallel Q) = \sum_i P(i) \log\left(\frac{P(i)}{Q(i)}\right) = \sum_i P(i).log(P(i)) - P(i).log(Q(i))$$

Áp dụng cho yTrue và yPred

$$D_{KL}(yTrue \parallel yPred) = \sum_i yTrue(i) \log\left(\frac{yTrue(i)}{yPred(i)}\right)$$

$$= \sum_i yTrue(i).log(yTrue(i)) - yTrue(i).log(yPred(i))$$

Mà yTrue thường onehot nên chỉ có 0 hoặc 1:

$yTrue.log(yTrue) = 0$ khi yTrue = 0, 1

Nên công thức gọn lại:

$$D_{KL}(yTrue \parallel yPred) = \sum_i -yTrue(i).log(yPred(i))$$

```python
# Tạo mô hình
model = SimpleClassifier(input_size, num_classes)

# Hàm loss là Cross-Entropy Loss
criterion = nn.CrossEntropyLoss()

# Tạo ví dụ dữ liệu và nhãn
input_data = torch.randn(1, input_size)  # Ví dụ đầu vào (batch size = 1)
target = torch.LongTensor([2])           # Nhãn của ví dụ (lớp 2)

# Tối ưu hóa bằng gradient descent
optimizer = optim.SGD(model.parameters(), lr=0.1)

# Forward pass
outputs = model(input_data)

# Tính loss bằng Cross-Entropy Loss
loss = criterion(outputs, target)
```

Cross-Entropy Loss

Thường dùng cho **Classification Problems**. Nó đo lường sự tương đồng giữa phân bố xác suất dự đoán và phân bố thực tế.

Formula for binary classification:

$$\text{Cross-Entropy} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

Formula for multi-class classification:

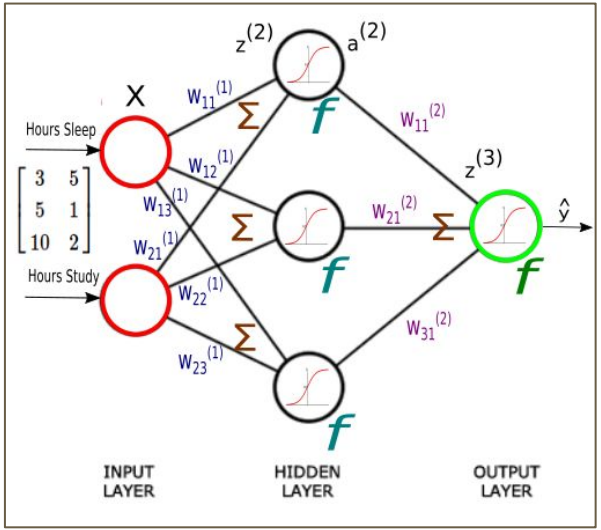
$$\text{Cross-Entropy} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}\log(\hat{y}_{ij})$$

where m is the number of classes.

python Copy code

```python
from sklearn.metrics import log_loss
loss = log_loss(y_true, y_pred)
```

Forward and Backward

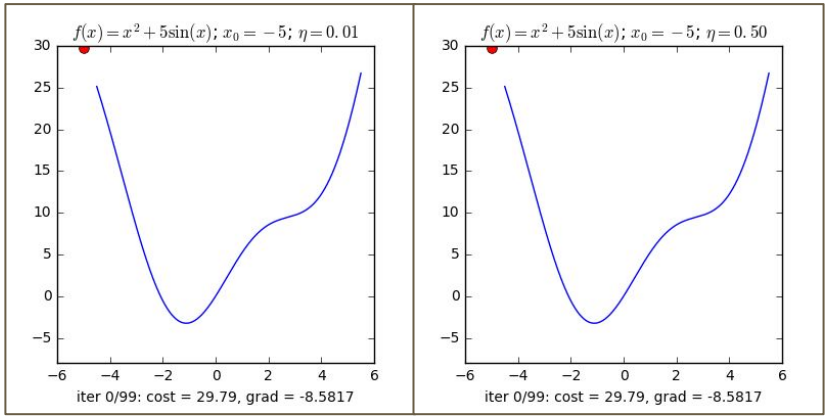


Quá trình tính từ input cho tới khi ra giá trị output được gọi là **Forward propagation.**
=> Để tính được thì các giá trị W_11(1), W_12(1), W_13(2),... được khởi tạo ngẫu nhiên gọi là **Initial Weight**

Để xác định mô hình có tốt hay không thì sẽ có quá trình tính Loss.
=> Mục đích tối ưu Loss càng nhỏ, chứng tỏ mô hình càng tốt.

Bản chất Loss là một hàm của các trọng số. Việc đi tối ưu Loss chính là tìm bộ trọng số **W*** để hàm loss nhỏ nhất.
=> Quá trình đi cập nhật trọng số W* gọi là **Backward Propagation.**

Gradient Descent



Trong thực tế, việc tìm kiếm nghiệm để cho một hàm số đạt cực trị bằng cách giải phương trình là khó và không phải lúc nào cũng tìm được.

Do đó người ta dùng Gradient Descent để tìm giá trị cực tiểu x* để hàm f(x) đạt giá trị cực tiểu

Vậy giá trị x_0 được khởi tạo theo nhiều cách:

1. **Phân phối Gaussian (Normal distribution):**
   Tên công thức: Gaussian Initialization
   Trọng số $\sim \mathcal{N}(0, 1)$
2. **Phân phối Uniform:**
   Tên công thức: Uniform Initialization
   Trọng số $\sim \text{Uniform}(-a, a)$
3. **Phân phối Xavier/Glorot:**
   Tên công thức: Xavier/Glorot Initialization
   Trọng số $\sim \mathcal{N}\left(0, \sqrt{\frac{2}{\text{số lượng đầu vào} + \text{số lượng đầu ra}}}\right)$
4. **Phân phối He:**
   Tên công thức: He Initialization
   Trọng số $\sim \mathcal{N}\left(0, \sqrt{\frac{2}{\text{số lượng đầu vào}}}\right)$

Tên công thức: Phân phối Gaussian

$$\mathcal{N}(\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$x_{t+1} = x_t - \eta f'(x_t)$$

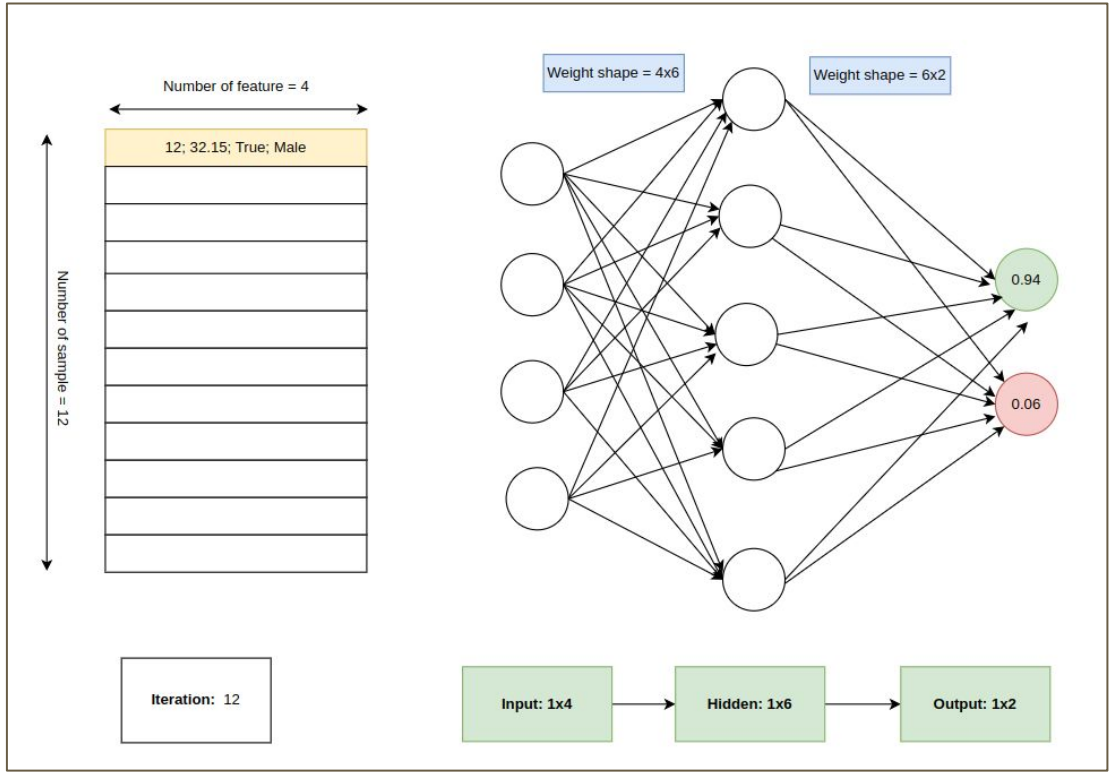
Nghiệm 1 biến

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta f(\theta_t)$$

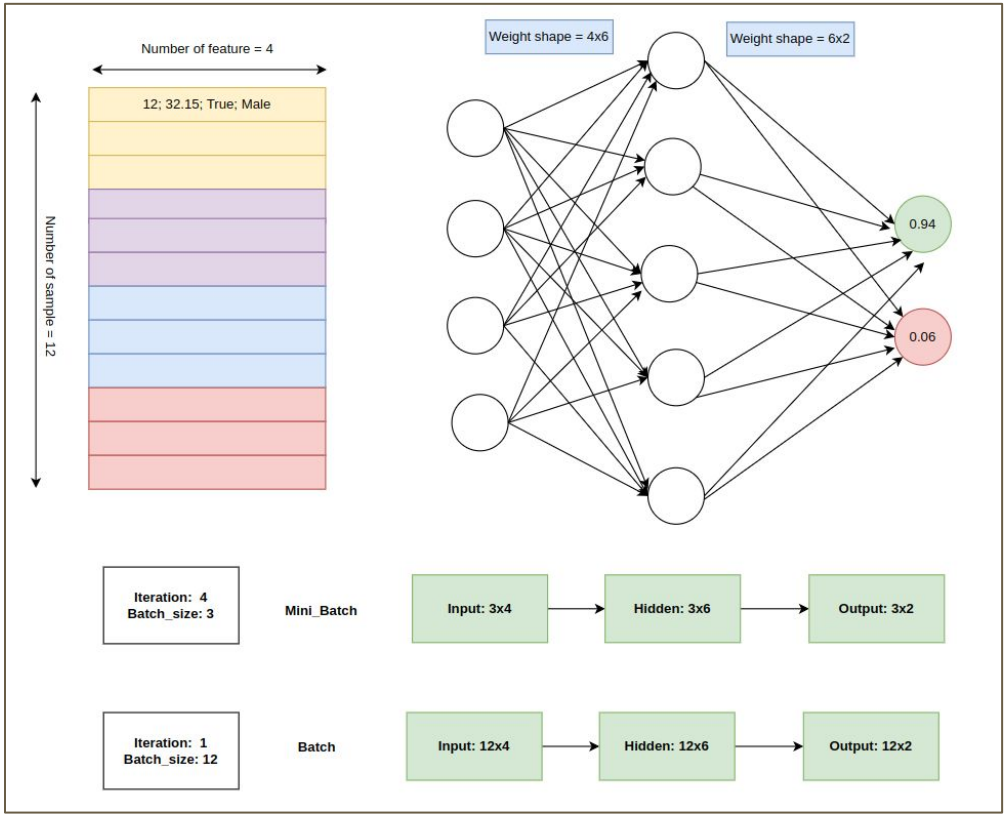
Nghiệm đa biến

Stochastic Gradient (SGD)

epochs: Là số lần duyệt qua toàn bộ dữ liệu.
Iterations: Là số lần cập nhật trọng số.



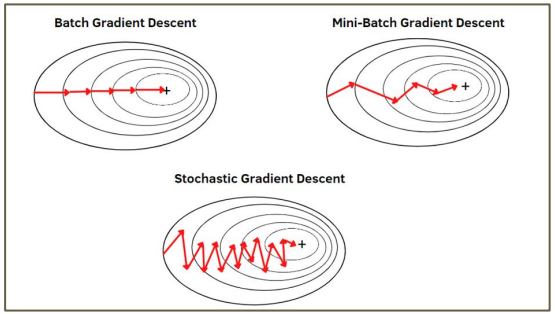
Mini-Batch vs Batch Gradient Descent



Why using Mini-Batch ?

- Nếu dùng SGD thì việc tối ưu trên từng điểm dữ liệu thì việc tối ưu sẽ rất giao động
- Nếu dùng Batch GD thì tối ưu ổn định, nhưng yêu cầu tính toán quá lớn dẫn đến thời gian huấn luyện lâu.

=> Đánh đổi để tối ưu ổn định và thời gian huấn luyện hợp lý



Pseudocode

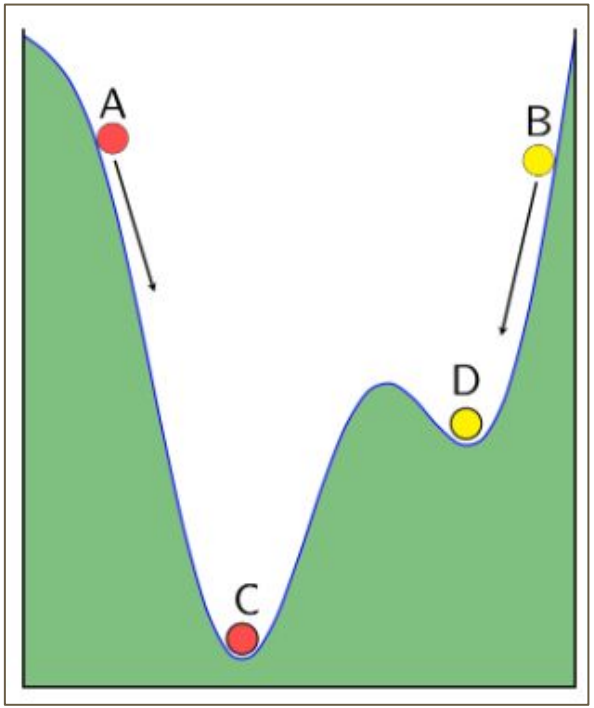
```
Pseudocode: SGD ( Stochastic Gradient Descent)
Initialize weights W randomly

for each epoch do:
  for i = 1 to m do:  # where m is the number of training examples
    compute gradient: grad = compute_gradient(loss_func(W, X[i], y[i]))
    update weights: W = W - learning_rate * grad
  end for
end for


Pseudocode: Mini-Batch Gradient Descent
Initialize weights W randomly
Set batch_sze

for each epoch do:
  for i=1 to m/batch_sze do:
    compute gradient: grad = compute_gradient(loss_func(W, X[i:i+batch_sze], y[i:i+batch_sze]))
    update weights: W = W - learning_rate * grad
  end for
end for

Pseudocode: Batch Gradient Descent
Initialize weights W randomly
for each epoch do:
  compute gradient: grad = compute_gradient(loss_func(W, X, y))
  update weights: W = W - learning_rate * grad
end for
```



Gradient Descent with Momentum

Formula:

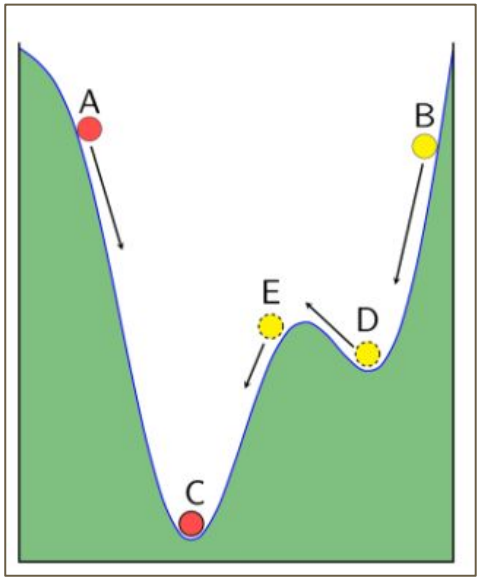
$$v_0 = 0$$

$$v_t = \beta.v_{t-1} + lr.\Delta(w_t)$$

$$w_t = w_{t-1} - v_t$$

Representing v_t by v_0: $v_t = \beta^t.v_0 + \beta^t.lr.\Delta(w_0) + \beta^{t-1}.lr.\Delta(w_1) + ... + \beta.lr.\Delta(w_{t-1}) + lr.\Delta(w_t)$

```
1
2   Pesudocode: Gradient Descent with Momentum
3
4   Initialize weights W randomly
5   Initialize velocity v = 0
6   Set momentum factor gamma
7
8   for each epoch do:
9     compute gradient: grad = compute_gradient(loss_func(W, X, y))
10    update velocity: v = gamma * v + learning_rate * grad
11    update weights: W = W - v
12  end for
```



Adam (Adaptive Moment Estimation)

$$m_0 = 0 \quad (Initialize\ initial\ 1^{st}\ moment\ vector)$$
$$v_0 = 0 \quad (Initialize\ initial\ 2^{nd}\ moment\ vector)$$
$$t = 0 \quad (Initialize\ timestep)$$

$$t = t + 1$$
$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \quad (Update\ biased\ first\ moment\ estimate)$$
$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \quad (Update\ biased\ second\ raw\ moment\ estimate)$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad (Compute\ bias - corrected\ first\ moment\ estimate)$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (Compute\ bias - corrected\ second\ raw\ moment\ estimate)$$
$$\theta_t = \theta_{t-1} - \frac{\mathrm{lr} \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \quad (Update\ parameters)$$

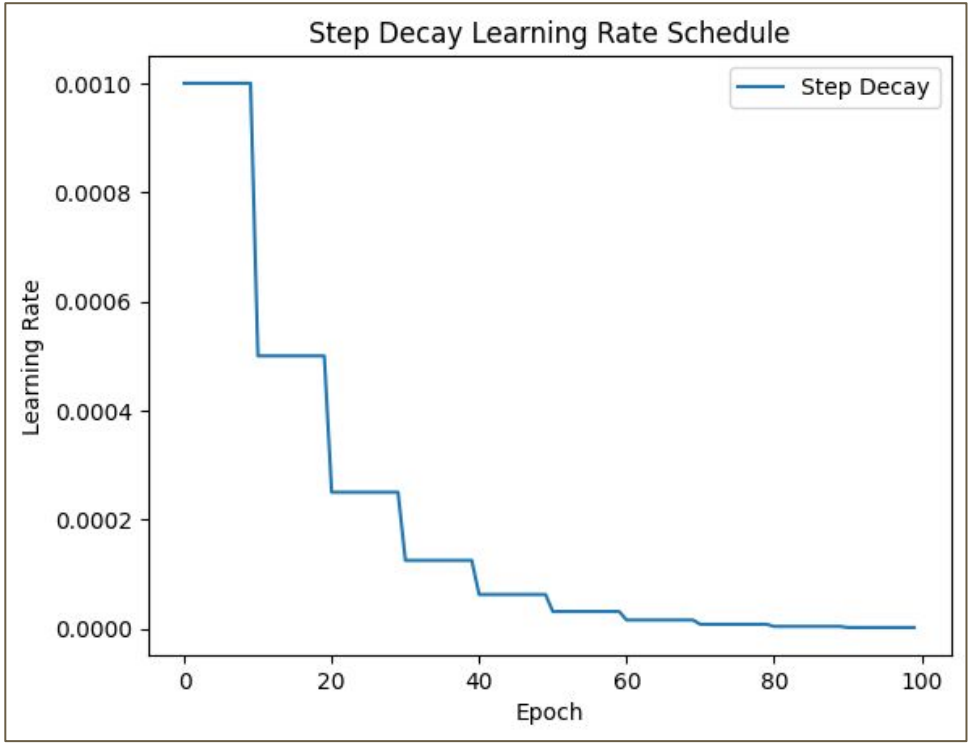
Why Should using Adam ?

- Adaptive learning rates
- More stable
- Less need parameter tuning

Some parameter default:

- lr = 0.001
- beta1 = 0.9
- beta2 = 0.999
- epsilon = 1e-8

Step Decay



Formula: $\mathrm{lr} = \mathrm{lr} \times \mathrm{drop\_rate}$

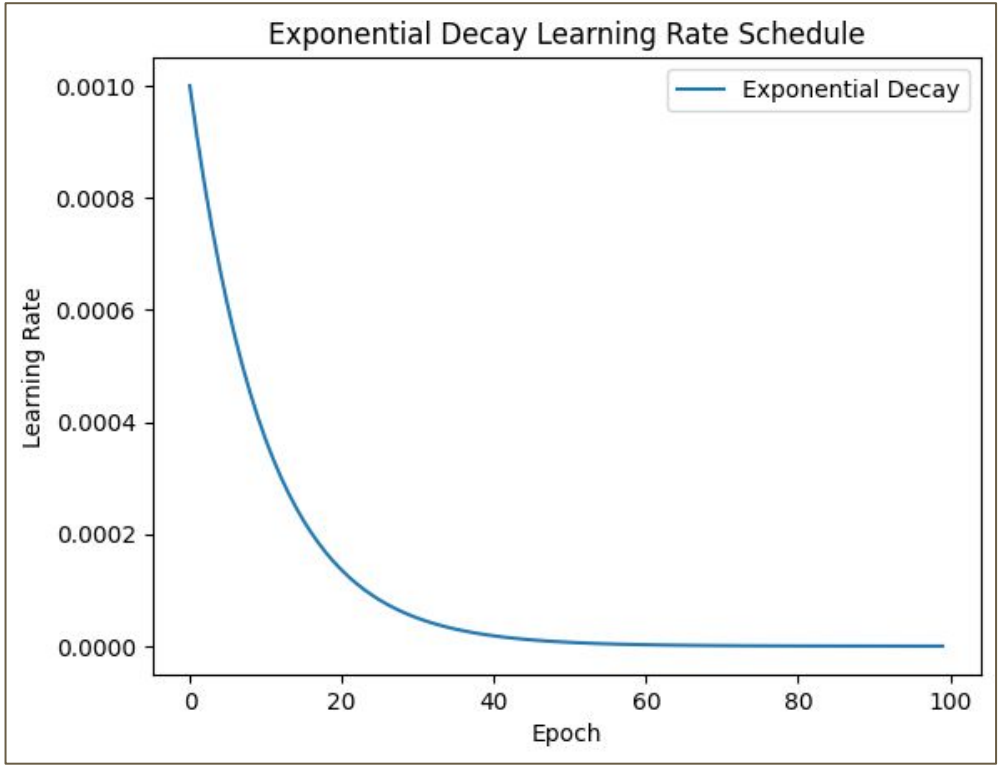
pseudo — Copy code

```
initialize lr_initial, drop_rate, epochs_drop

for epoch in range(epochs):
    if epoch % epochs_drop == 0 and epoch != 0:
        lr = lr * drop_rate

    // update weights with current learning rate
```

Exponential Decay



Formula: $\mathrm{lr} = \mathrm{lr}_{\text{initial}} \times e^{-\text{decay\_rate} \times \text{epoch}}$

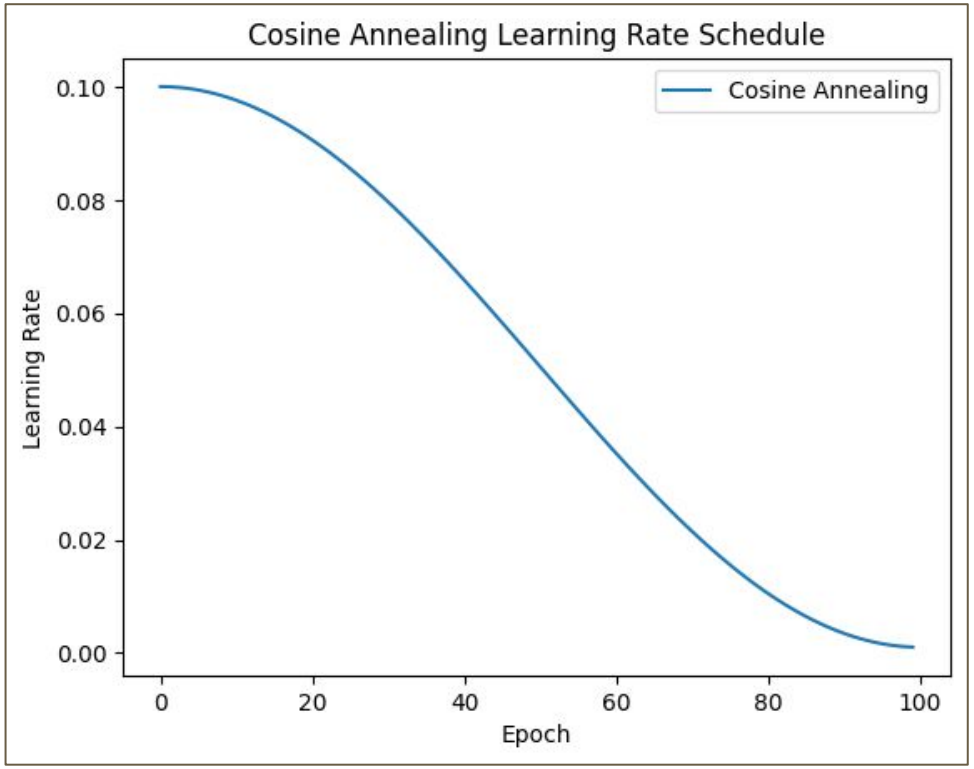
pseudo — Copy code

```
initialize lr_initial, decay_rate

for epoch in range(epochs):
    lr = lr_initial * exp(-decay_rate * epoch)

    // update weights with current learning rate
```

Exponential Decay



Formula: $\mathrm{lr} = \frac{\mathrm{lr}_{\min} + (\mathrm{lr}_{\max} - \mathrm{lr}_{\min})}{2} \times \left(1 + \cos\left(\frac{\mathrm{epoch}}{\mathrm{epochs}} \times \pi\right)\right)$

pseudo

Copy code

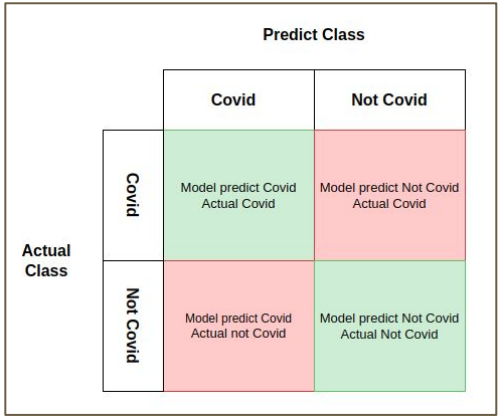
```
initialize lr_min, lr_max

for epoch in range(epochs):
    lr = lr_min + 0.5 * (lr_max - lr_min) * (1 + cos(epoch / epochs * pi))

    // update weights with current learning rate
```

Confusion Matrix

Actual Class	Positive	Negative	
Positive	True Positive (TP)	False Negative (FN) **Type II Error**	**Sensitivity** $\frac{TP}{(TP+FN)}$
Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN+FP)}$
	Precision $\frac{TP}{(TP+FP)}$	**Negative Predictive Value** $\frac{TN}{(TN+FN)}$	**Accuracy** $\frac{TP+TN}{(TP+TN+FP+FN)}$

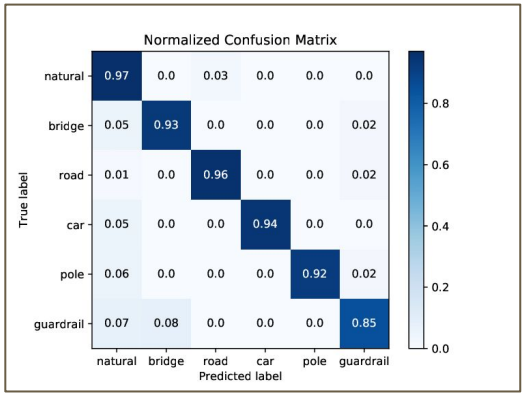


Là một bảng đánh giá performance của classification model. Gồm :

- **TP (True Positive):** Số lượng các mẫu được dự đoán là positive và thực sự là positive.
- **TN (True Negative):** Số lượng các mẫu được dự đoán là negative và thực sự là negative.
- **FP (False Positive):** Số lượng các mẫu được dự đoán là positive nhưng thực sự là negative. Đôi khi còn được gọi là "Type I error".
- **FN (False Negative):** Số lượng các mẫu được dự đoán là negative nhưng thực sự là positive. Đôi khi còn được gọi là "Type II error".

Precision, Recall, Accuracy

		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) Type II Error	Sensitivity $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) Type I Error	True Negative (TN)	Specificity $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	Negative Predictive Value $\frac{TN}{(TN+FN)}$	Accuracy $\frac{TP+TN}{(TP+TN+FP+FN)}$



Recall (hoặc Sensitivity hoặc True Positive Rate): Tỉ lệ của các mẫu positive mà mô hình dự đoán chính xác.

Precision: Tỉ lệ của các mẫu positive mà mô hình dự đoán chính xác so với tổng số mẫu được dự đoán là positive.

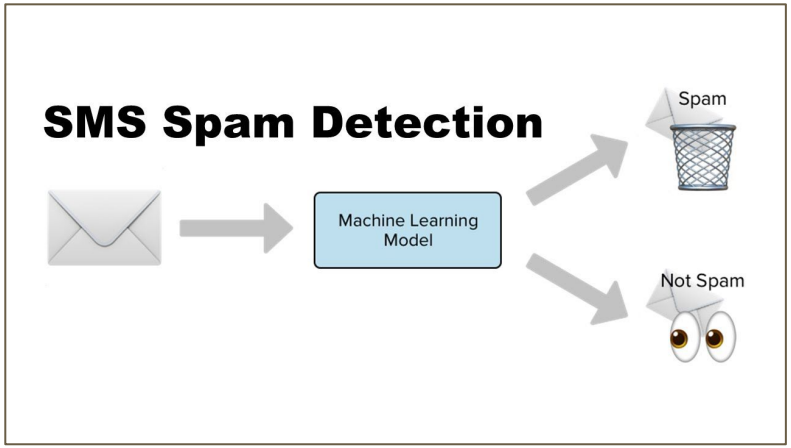
Accuracy: Tỉ lệ của số mẫu dự đoán chính xác so với tổng số mẫu.

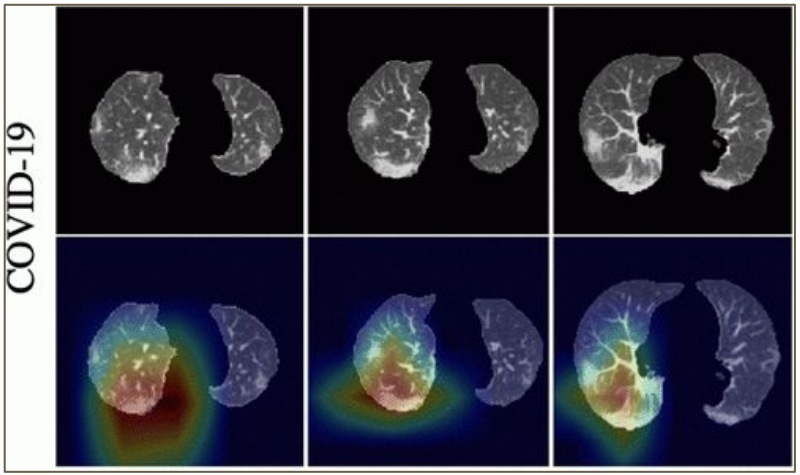
When need ?

- **Dùng Accuracy** khi bạn muốn có một đánh giá tổng quan về độ hiệu quả của mô hình

When using Precision or Recall ??

When using Precision or Recall ??



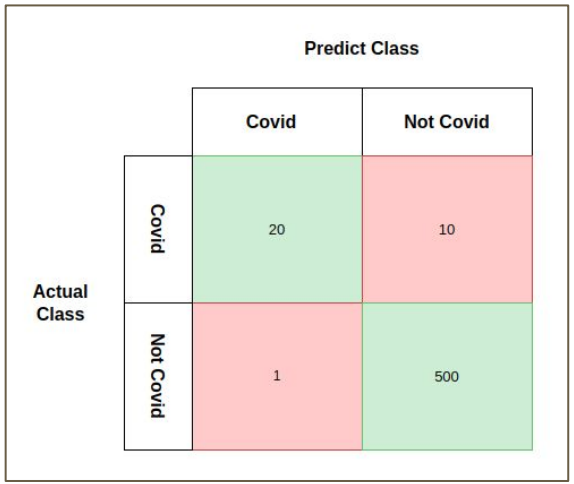


- Spam Detection: Ưu tiên không bỏ lỡ các email quan trọng, có thể bắt thiếu Spam mail.
=> Ưu tiên việc giảm trường hợp FP (Mô hình dự đoán Spam, nhưng thực tế No Spam)
-> Chọn **Precision.**
- Covid Detection: Ưu tiên dự đoán được hết tất cả các bệnh nhân bị covid.
=> Ưu tiên việc giảm trường hợp FN (Mô hình dự đoán Không Covid, nhưng thực tế bị Covid)
-> Chọn **Recall.**

F1-Score and Variant

$$\text{F1 Score} = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$F_\beta = (1 + \beta^2) \times \frac{\text{Precision} \times \text{Recall}}{(\beta^2 \times \text{Precision}) + \text{Recall}}$$



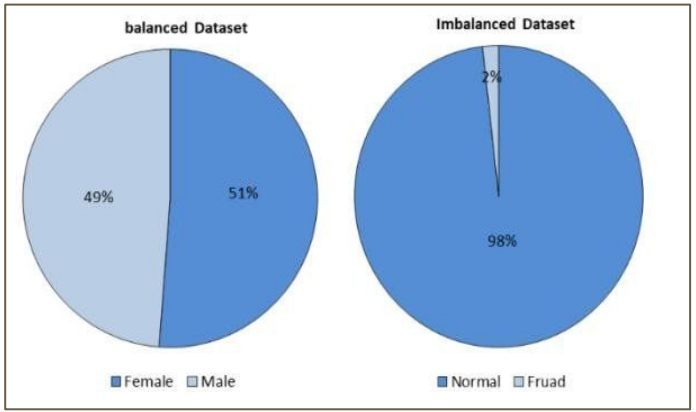
Ở đây, β (beta) là một tham số điều chỉnh.

- Khi β = 1, F-beta trở thành F1-Score, tạo ra sự cân nhắc giữa Precision và Recall.
- Khi β > 1, F-beta đánh trọng số nhiều hơn cho Recall so với Precision.
- Khi β < 1, F-beta đánh trọng số nhiều hơn cho Precision so với Recall.

Ví dụ: F2-Score (β = 2) sẽ đánh trọng số gấp đôi cho Recall so với Precision, làm cho nó trở nên hữu ích trong những tình huống mà việc bỏ sót các trường hợp dương tính là không mong muốn (ví dụ, phát hiện bệnh).

When using F1-score ?

Imbalanced Data



What ?

Xảy ra khi tập dữ liệu có tần suất xuất hiện rất thấp so với các lớp khác.

How to solve this ?

- **Using Resampling (Oversampling and Undersampling):** Cố gắng tạo lại bộ dữ liệu để các lớp có tần suất cân bằng hơn.
- **Weight Loss Function:** Điều chỉnh hàm mất mát để đặt trọng số cao hơn cho lớp thiểu số, giúp mô hình tập trung hơn vào việc phân loại đúng lớp này.

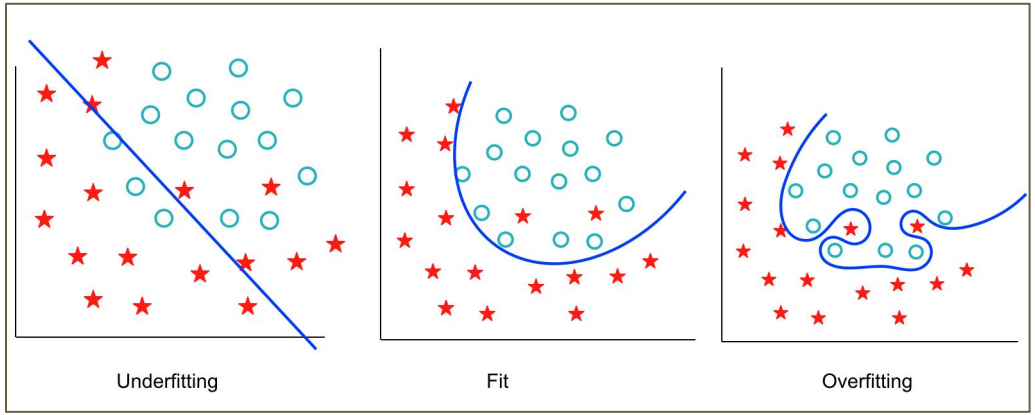
```python
import torch.nn as nn
import torch
import numpy as np

samples_per_classes = np.array([100, 200, 300, 400])
#Chuẩn hoán vector samples_per_classes
class_number = samples_per_classes/sum(samples_per_classes)
#Apply công thức tính weight
class_weights = [1-i for i in class_number]
# Convert numpy to tensor
class_weights_tensor = torch.tensor(class_weights, dtype=torch.float32)
if torch.cuda.is_available():
    class_weights = class_weights.cuda()

criterion = nn.CrossEntropyLoss(weight=class_weights)
```

$$class\ weight = 1 - \left(\frac{number\ of\ samples\ of\ the\ class}{total\ number\ of\ samples} \right)$$

Underfit and Overfit



Underfitting: Xảy ra khi mô hình chưa học được đủ thông tin từ dữ liệu huấn luyện và không thể phù hợp với dữ liệu kiểm tra hoặc dữ liệu thực tế.

Overfitting: Xảy ra khi mô hình quá phức tạp và đã học quá nhiều từ dữ liệu huấn luyện, điều này làm cho nó cực kỳ chính xác trên dữ liệu huấn luyện nhưng không thể tổng quát hóa tốt với dữ liệu kiểm tra hoặc dữ liệu mới.

Solve Underfit and Overfit

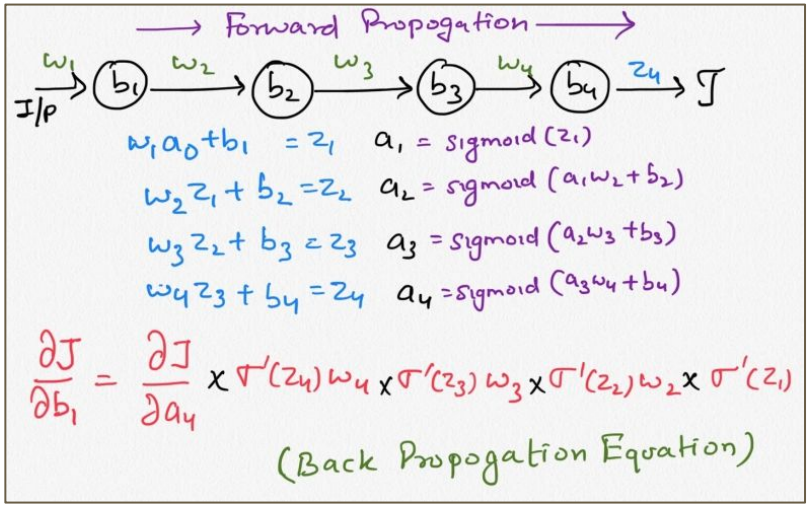
Underfit:

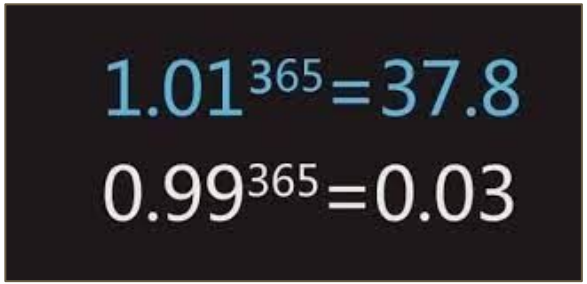
- Tăng độ phức tạp của mô hình: Tăng số lớp ẩn của mạng, ...
- Tăng thời gian huấn luyện

Overfit:

- L1, L2
- Dropout
- Early Stopping
- Data Augmentation

Vanishing and Exploding Gradients





Vanishing:

Khi lan truyền ngược, một số đạo hàm giảm đáng kể, dẫn đến đạo hàm ở các lớp đầu tiến về không. Dẫn đến mô hình không cập nhật được trọng số

=> Không học được

Giải pháp:

- Dùng Activate Function như ReLU
- Skip Connection

Exploding:

Tương tự như trên, khi một số đạo hàm tăng đáng kể, dẫn đến trọng số cũng tăng cao => Mô hình không hội tự được và quá trình học không ổn định

Giải pháp:

- Gradient Clipping
- Sử dụng Regularization (L2)

L1, L2 and Elastic Net

L1 Regularization (Lasso):

Formula:

$$L_{L1}(\theta) = L(\theta) + \lambda||\theta||_1 = L(\theta) + \lambda\sum_{i=1}^{n}|\theta_i|$$

Derivative:

$$\frac{\partial L_{L1}}{\partial \theta_i} = \frac{\partial L}{\partial \theta_i} + \lambda \cdot sign(\theta_i)$$

L2 Regularization (Ridge):

Formula:

$$L_{L2}(\theta) = L(\theta) + \lambda||\theta||_2^2 = L(\theta) + \lambda\sum_{i=1}^{n}\theta_i^2$$

Derivative:

$$\frac{\partial L_{L2}}{\partial \theta_i} = \frac{\partial L}{\partial \theta_i} + 2\lambda\theta_i$$

L1 Regularization: Dùng để giải thích mô hình Machine Learning.

L2 Regularization: Giảm overfit cho mô hình

Elastic Net Regularization: Kết hợp hai chức năng trên.

Elastic Net Regularization:

Formula:

$$L_{ElasticNet}(\theta) = L(\theta) + \lambda_1||\theta||_1 + \lambda_2||\theta||_2^2$$

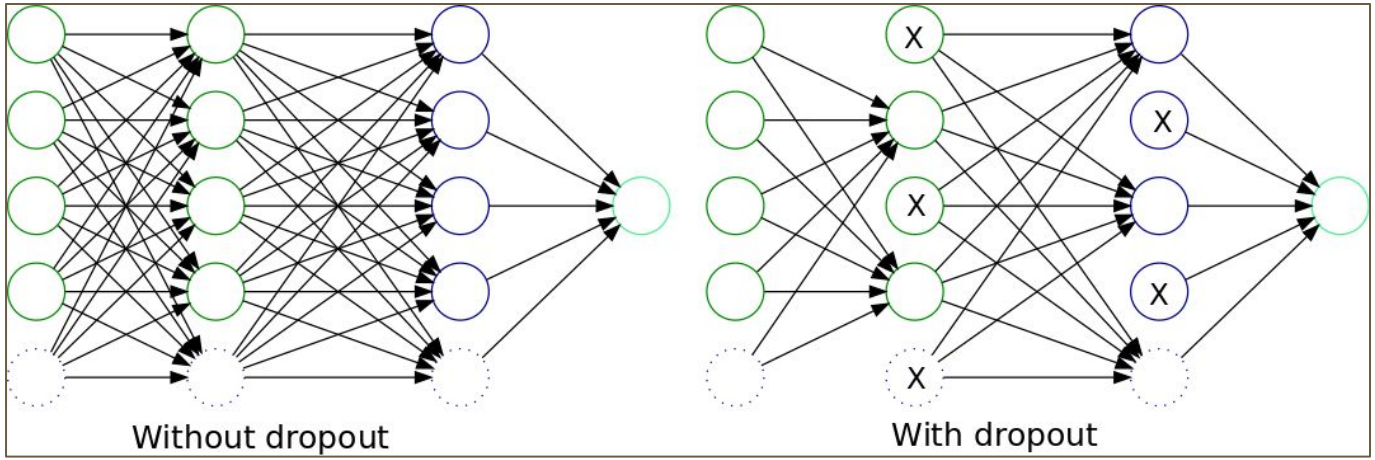
Derivative:

$$\frac{\partial L_{ElasticNet}}{\partial \theta_i} = \frac{\partial L}{\partial \theta_i} + \lambda_1 \cdot sign(\theta_i) + 2\lambda_2\theta_i$$

Dropout

Là một kỹ thuật **regularization** trong học sâu. Nó sẽ loại bỏ ngẫu nhiên các neuron trong mạng, nghĩa là chúng không được cập nhật trong quá trình forward và backward.

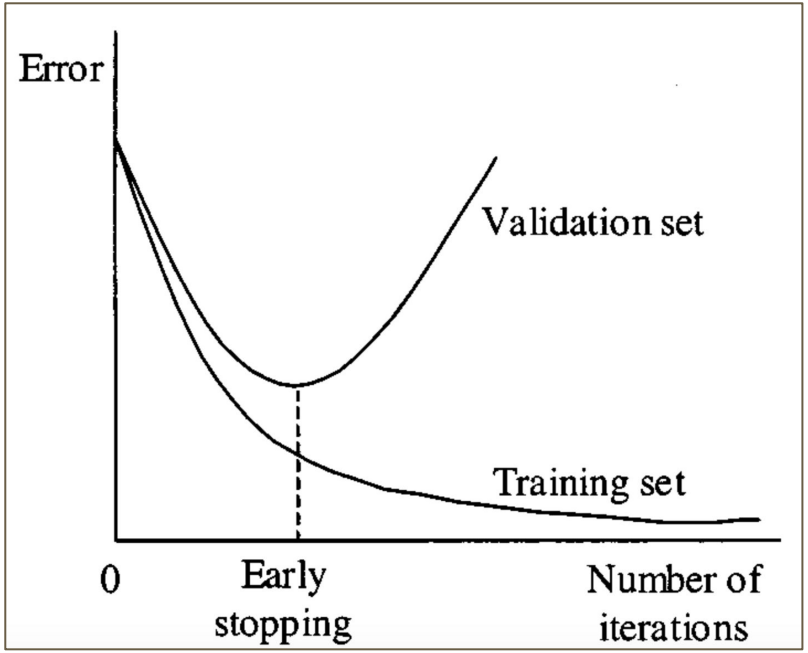
Giúp tránh **overfitting**, tăng cường tính tổng quát và giảm sự phụ thuộc quá mức vào một neuron cụ thể.



Early Stopping

Là kỹ thuật áp dụng khi huấn luyện mô hình với mục đích dừng huấn luyện mô hình khi mô hình không có dấu hiệu tiến bộ. Được kiểm soát bởi thông số patient.

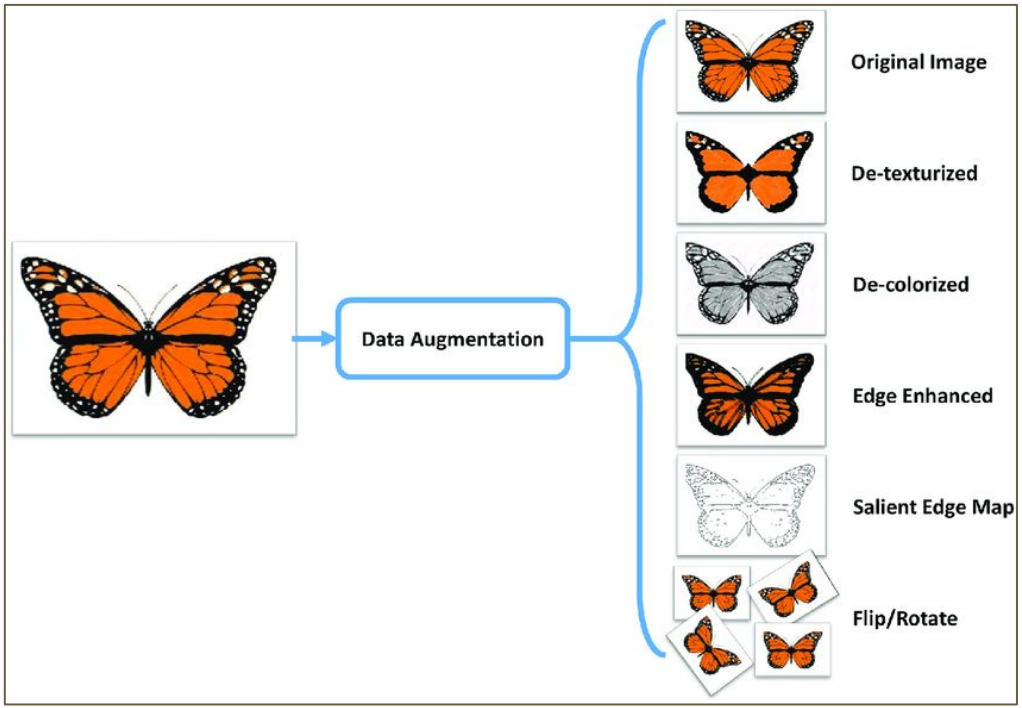
- Patient: Đếm số lần epoch mà không hình không có sự cải thiện.



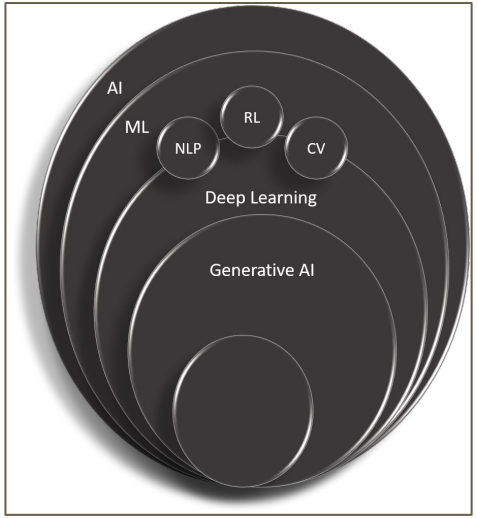

```
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1)
```

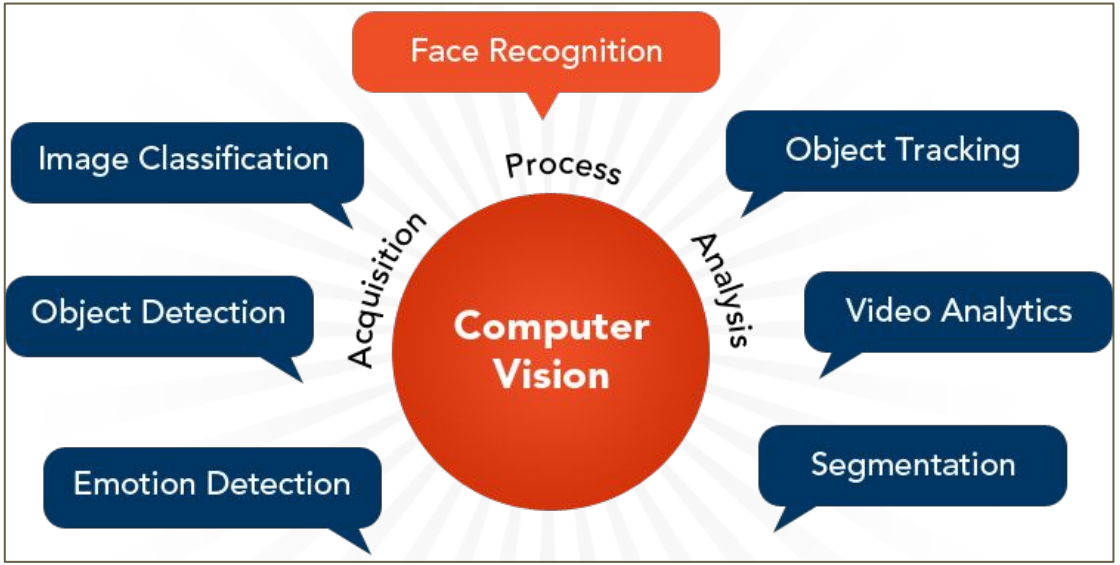
Data Augmentation

- Giúp tăng cường và đa dạng hóa tập dữ liệu.
=> Giúp mô hình tăng khả năng khái quát hóa và học được những đặc trưng chung và giảm Overfitting

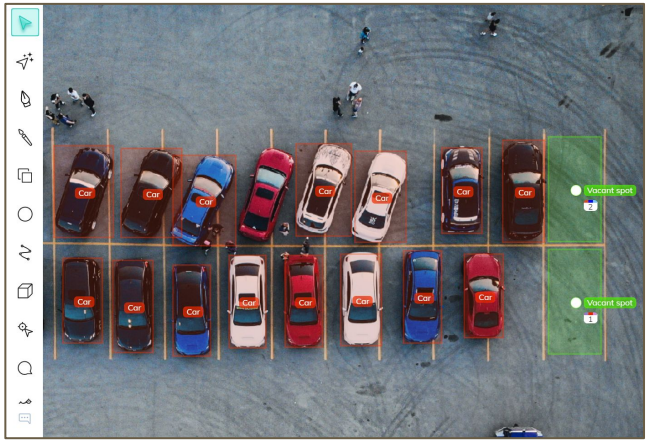


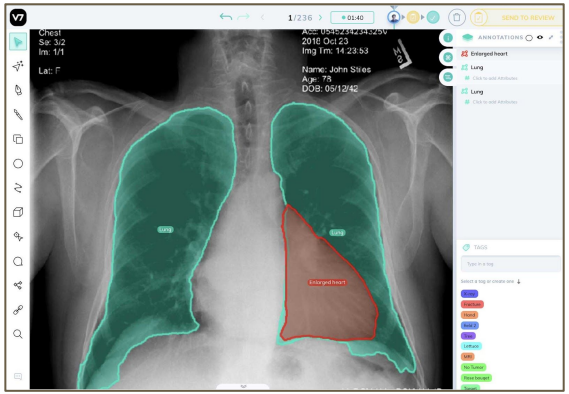
What is Computer Vision ?



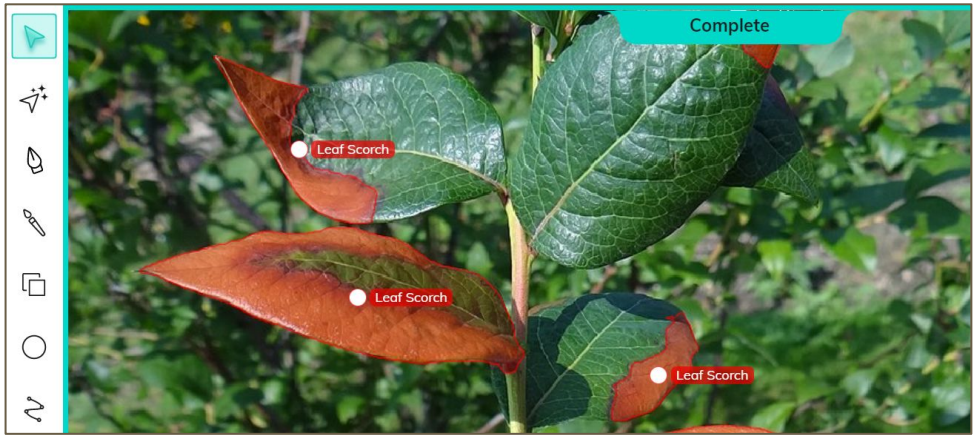


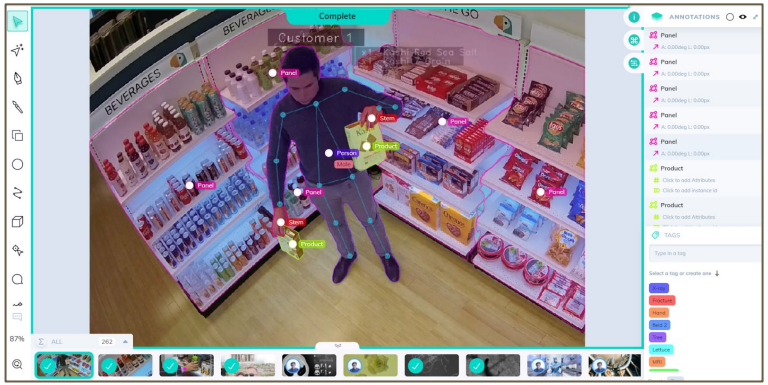
Applications

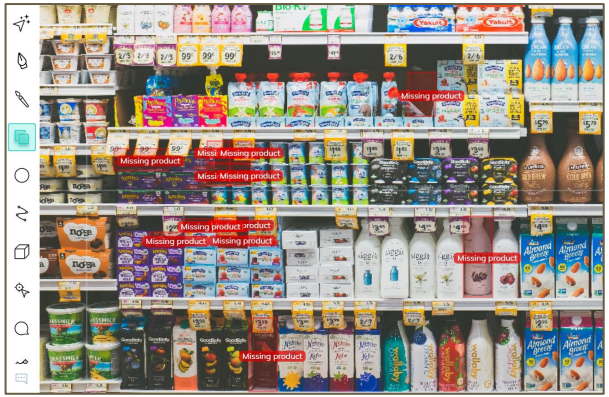


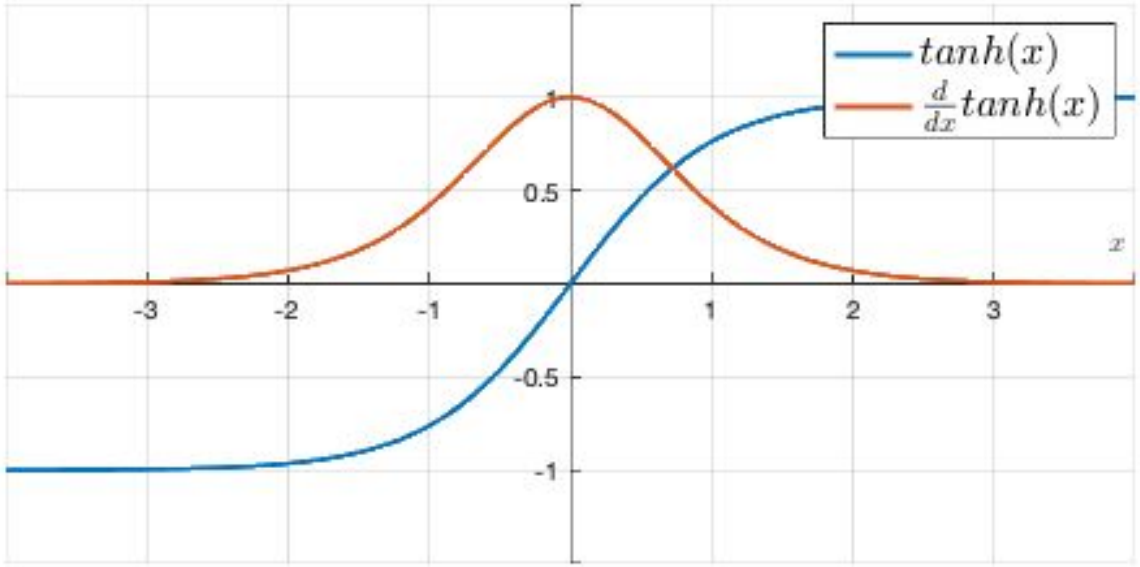


Applications








$tanh(x)$
$\frac{d}{dx}tanh(x)$
1
0.5
-0.5
-1
-3
-2
-1
1
2
3
x