3.0 OVERVIEW PyTorch `autograd`

24 August 2025 02:47 AM Avinash Yadav

(*) 'autograd' → PyTorch's automatic differentiation tool.

The Why?

Let's take a mathematical relation as:

$$y = x^2$$

Now let's say we have to write a python script for a given 'x' we need to find the derivative of 'y' with respect to 'x'

i.e. for given x we want $\frac{dy}{dx}$

it is simple. i.e. we have to find

$$y = x^2$$

$$\boxed{\frac{dy}{dx} = 2x}$$

→ Now very easily we can code this expression

and whenever we keep any new value of 'x' we can easily get the the value of $\frac{dy}{dx}$

X	$\frac{dY}{dX}$
2	4

3	6
4	8
5	10

```python
def dy_dx(x):
    return 2*x

dy_dx(3)
```

since this was easier to do, we solve it manually and later coded it as well.

But what if, we have a complex relation/equation which we have to differentiate, then what we'll do in that case?

i.e. for example:

$$Y = x^2$$

$$Z = \sin(Y)$$

we want $\Rightarrow$ $\frac{dz}{dx}$

Now again we have to do the same task, but this time it is little difficult, Now we have to Chain Rule of differentiation in order to solve this.

i.e. for $\frac{dz}{dx} = \frac{dz}{dY} \times \frac{dY}{dx}$

$$\downarrow \qquad \downarrow$$

$$\frac{d(\sin Y)}{dY} \times \frac{d(x^2)}{dx}$$

$$\downarrow \qquad \downarrow$$

$$\cos(y) \quad \times \quad (2x)$$

$$2x \cdot \cos(y)$$

x^2

$$\frac{dz}{dx} = 2x \cdot \cos(x^2)$$

Now again we got the formula and for this we can again write the code such that for a given 'x' we will get the derivative of 'z' wrt 'x' i.e. $\frac{dz}{dx}$

```python
import math

def dz_dx(x):
    return 2 * x * math.cos(x**2)

dz_dx(4)
```

Now what if we got one more level of difficulty. let's say we have:

$$y = x^2$$
$$z = \sin(y)$$
$$u = e^z$$

& we want $\frac{dy}{dx}$

Therefore now for $\frac{dy}{dx}$ we need: $\frac{dy}{dx} = \frac{dy}{dz} \times \frac{dz}{dy} \times \frac{dz}{dx}$

whatever the final expression will get, then we have to code that.

So what we saw above that:
As the complexity of nested function increases, the difficulty of finding their derivatives and coding them also increases.

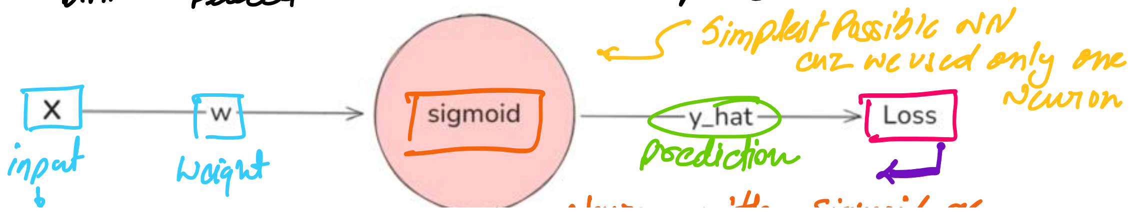
{ Nested fxn → Complex → derivative → difficult
└→ code → difficult }

So we are doing this because 'nested function' and finding their derivatives are very much closely related to deep learning.

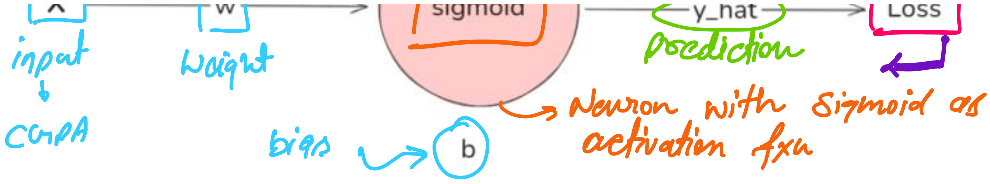
dataset with 1 feature & label
↓ cGPA ↓ placed

{ we have to train a NN on this data which will take CGPA & predict whether student will be placed or not

cgpa	placed
9.11	1
8.9	1
7	0
6.56	1
4.56	0



7	0
6.56	1
4.56	0



Training process

1. ✓ Forward pass - Compute the output of the network given an input.
2. ✓ Calculate loss - Calculate the loss function to quantify the error.
3. ✓ Backward pass - Compute gradients of the loss with respect to the parameters.
4. Update gradients - Adjust the parameters using an optimization algorithm (e.g., gradient descent).

Forward Pass Computation

1. Linear Transformation:

W x b

$$\boxed{z = w \cdot x + b}$$

2. Activation (Sigmoid Function):

$$\boxed{y_{\text{pred}} = \sigma(z) = \frac{1}{1+e^{-z}}}$$

3. Loss Function (Binary Cross-Entropy Loss):

$$\boxed{L = -\left[y_{\text{target}} \cdot \ln(y_{\text{pred}}) + (1 - y_{\text{target}}) \cdot \ln(1 - y_{\text{pred}})\right]}$$

After this step we'll come to how much wrong our NN is going in predict the true placed value.

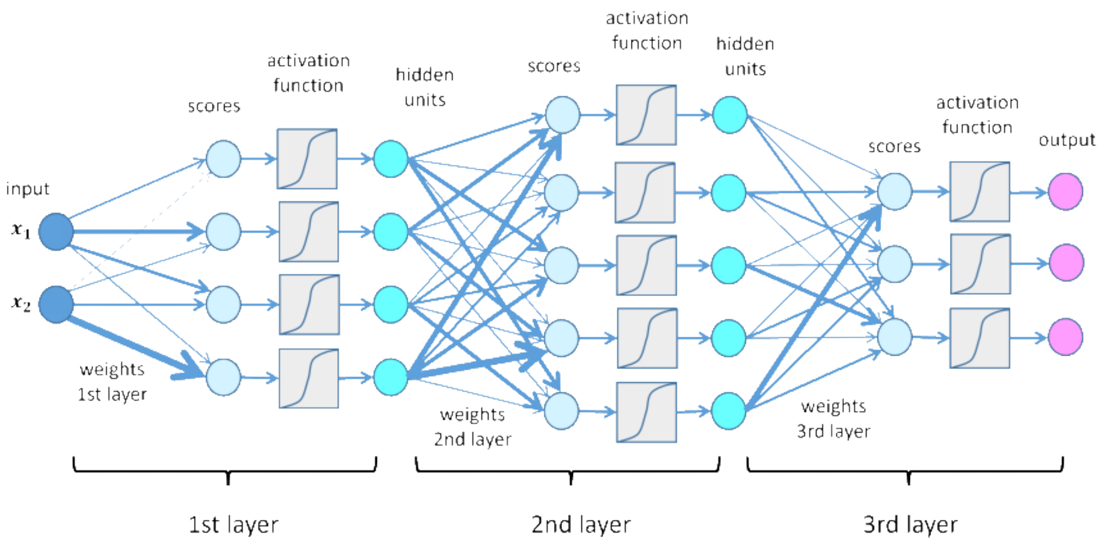
It means Now we have to compute $\frac{\partial L}{\partial w}$ & $\frac{\partial L}{\partial b}$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y_{pred}} \times \frac{\partial y_{pred}}{\partial z} \times \frac{\partial z}{\partial w}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial y_{pred}} \times \frac{\partial y_{pred}}{\partial z} \times \frac{\partial z}{\partial b}$$

Again here we have to apply the chain rule to get the derivatives.

Since we have this NN with only 1 neuron.



Now if this NN is bigger with many

Why if this NN is bigger with many Neurons ?

→ Imagine the kind of complexity

This issue is resolved by 'autograd'

Looking at a top level view of this NN, we see that this is a nested function, where we are first finding 'z' the feeding it to activation function to get out 'ypred' and then computing the loss based on it.

Different Kinds of Loss Functions

	Linear Regression	Mean Squared Error
	Logistic Regression	Cross-Entropy Loss
	Decision Tree Classifier	Information Gain or Gini impurity
	Decision Tree Regressor	Mean Squared Error
	Random Forest Classifier	Information Gain or Gini impurity
	Random Forest Regressor	Mean Squared Error
	Support Vector Machines (SVMs)	Hinge Loss
	k-Nearest Neighbors	No loss function
$P(B\|A) = \frac{P(B \cap A)}{P(A)}$	Naive Bayes	No loss function
	Neural Networks	Regression: Mean Squared Error Classification: Cross-Entropy Loss
	AdaBoost	Exponential loss
	Gradient Boosting | LightGBM | CatBoost | XGBoost	Regression: Mean Squared Error Classification: Cross-Entropy Loss
	KMeans Clustering	Within Class ?? Sum of Squares (WCSS)

Loss Function Name	Description	Function
Regression Losses		
Mean Bias Error	Captures average bias in prediction. But is rarely used for training.	$\mathcal{L}_{MBE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))$
Mean Absolute Error	Measures absolute average bias in prediction. Also called L1 Loss.	$\mathcal{L}_{MAE} = \frac{1}{N}\sum_{i=1}^{N}\|y_i - f(x_i)\|$
Mean Squared Error	Average squared distance between actual and predicted. Also called L2 Loss.	$\mathcal{L}_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2$
Root Mean Squared Error	Square root of MSE. Loss and dependent variable have same units.	$\mathcal{L}_{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2}$
Huber Loss	A combination of MSE and MAE. It is parametric loss function.	$\mathcal{L}_{\text{Huberloss}} = \begin{cases} \frac{1}{2}(y_i - f(x_i))^2 & : \|y_i - f(x_i)\| \le \delta \\ \delta(\|y_i - f(x_i)\| - \frac{1}{2}\delta) & : otherwise \end{cases}$
Log Cosh Loss	Similar to Huber Loss + non-parametric. But computationally expensive.	$\mathcal{L}_{\text{LogCosh}} = \frac{1}{N}\sum_{i=1}^{N} log(cosh(f(x_i) - y_i))$
Classification Losses (Binary + Multi-class)		
Binary Cross Entropy (BCE)	Loss function for binary classification tasks.	$\mathcal{L}_{BCE} = \frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(x_i)) + (1 - y_i) \cdot log(1 - p(x_i))$
Hinge Loss	Penalizes wrong and right (but less confident) predictions. Commonly used in SVMs.	$\mathcal{L}_{\text{Hinge}} = max(0, 1 - (f(x) \cdot y))$
Cross Entropy Loss	Extension of BCE loss to multi-class classification.	$\mathcal{L}_{CE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} . log(f(x_{ij}))$ $N : samples; M : classes$
KL Divergence	Minimizes the divergence between predicted and true probability distribution	$\mathcal{L}_{KL} = \sum_{i=1}^{N} y_i \cdot log(\frac{y_i}{f(x_i)})$

1. **Linear Regression:** *Mean Squared Error (MSE). This can be used with and without regularization, depending on the situation.*

2. **Logistic regression:** *Cross-entropy loss or Log Loss, with and without regularization.*

3. **Decision Tree and Random Forest:**
   1. *Classifier: Gini impurity or information gain.*
   2. *Regressor: Mean Squared Error (MSE)*

4. **Support Vector Machines (SVMs):** *Hinge loss. It penalizes both wrong and right (but less confident) predictions. Best suited for creating max-margin classifiers, like in SVMs.*

5. **k-Nearest Neighbors (kNN):** *No loss function. kNN is a non-parametric lazy learning algorithm. It works by retrieving instances from the training data, and making predictions based on the k nearest neighbors to the test data instance.*

6. **Naive Bayes:** *No loss function. Naive Bayes doesn't have an explicit "loss function" in the same way iterative algorithms do because its parameters are determined directly through frequency counts and conditional probabilities, not through iterative optimization. Instead of a traditional training loop, it uses the maximum likelihood estimation principle, which, under the assumption of conditional independence, yields a simple, direct, closed-form solution for its parameters, effectively skipping the need for an explicit loss function during training*

7. **Neural Networks:** *They can use a variety of loss functions depending on the type of problem. The most common ones are:*
   1. *Regression: Mean Squared Error (MSE).*
   2. *Classification: Cross-Entropy Loss.*

8. **AdaBoost:** *Exponential loss function. AdaBoost is an ensemble learning algorithm. It combines multiple weak classifiers to form a strong classifier. In each iteration of the algorithm, AdaBoost assigns weights to the misclassified instances from the previous iteration. Next, it trains a new weak classifier and minimizes the weighted exponential loss.*

9. **Other Boosting Algorithms:**
   1. *Regression: Mean Squared Error (MSE).*
   2. *Classification: Cross-Entropy Loss.*

10. **Kmeans:** *The K-Means loss function, known as Inertia or within-cluster sum of squares (WCSS), is the sum of the squared Euclidean distances between each data point and its assigned cluster's centroid.*