The magic of Backpropagation and Gradient descent

By Sharon George

Git hub : https://github.com/sharonsibi/Backpropagation-ML-assignment.git

https://github.com/sharonsibi/Backpropagation-ML-assignment.git

What is in here!

1. Basics of Backpropagation
2. How Backpropagation works
3. Understanding Backpropagation with Analogies
4. Gradient Descent
5. Types of Gradient Descent
6. Variants of Gradient Descent
7. Where is backpropagation used

Backpropagation Basics

- The optimal neural network model has **minimized error function** and **maximized accuracy**. But how do we achieve this?
- Here comes **Backpropagation**—a feedback mechanism that helps refine the model to get optimized results.
- Backpropagation short for **Backward Propagation of Error** is a technique to calculate how changes to any of the weights of a neural network will affect the accuracy of model predictions.

2. How Backpropagation works?

Forward Pass –Predictions are made

Error Calculation – Compare prediction vs. actual output.

Backward Pass – Error propagates backward using **chain rule**.

Weight Update – Weights adjust using **Gradient Descent.**

3. Backpropagation through Analogies

Let's assume Machine Learning models as a bowl of pasta :

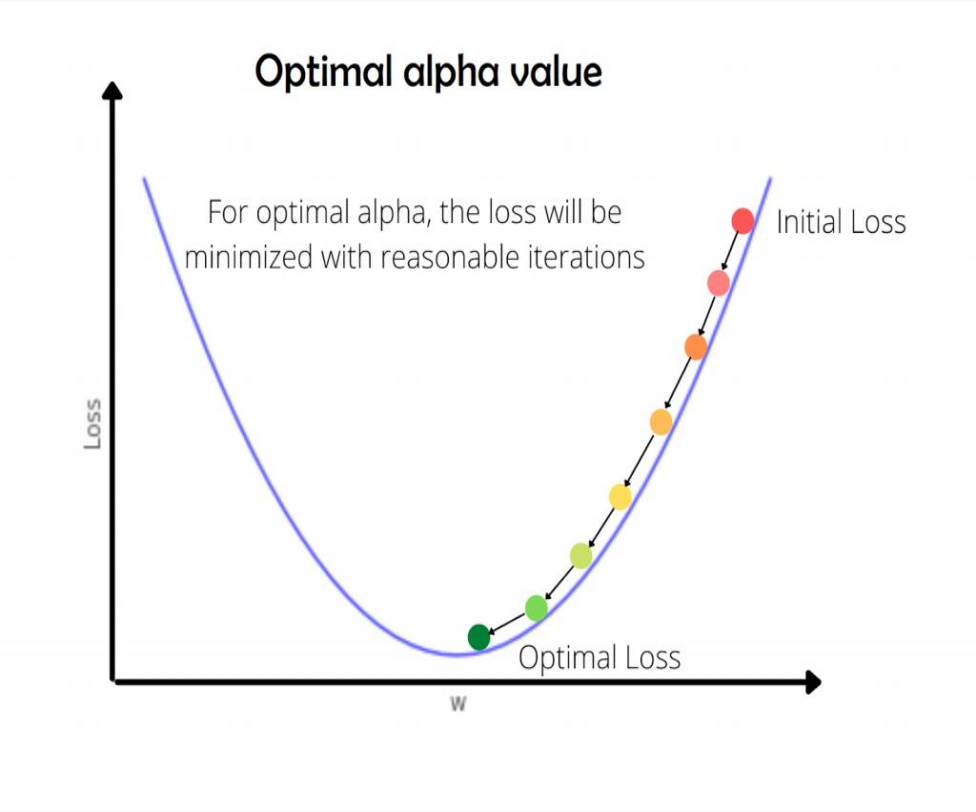
•**Gradient Descent** is the **chef** – It keeps tasting and adjusting the recipe (weights) to make the dish perfect.

•**Backpropagation** is the **recipe book** – It tells the chef what went wrong with the last dish and how to improve the next one.

•**Neural Networks** are the **pasta** – They become more flavorful (accurate) as the chef (Gradient Descent) follows the right recipe (Backpropagation) to adjust the ingredients (weights).

4. Gradient Descent

- Gradient Descent's **goal** is to minimize the error function through **iterative weight adjustments**, and repeating the process until the optimal point is reached. Essentially, it is a key optimization method used in backpropagation to achieve better results.
- **W(new) = W(old) - step size * gradient**
- If the step size is too large, it may overshoot the optimal point, while a **smaller** step size increases the number of **iterations.**



5. Types of Gradient Descent:

- **Batch Gradient Descent** – Uses the **entire dataset** for updates these are slow but very stable.
- **Stochastic Gradient Descent (SGD)** – we can sample a small fraction and compute a gradient step. This process is known as stochastic gradient
- **Mini-Batch Gradient Descent** – Uses **small batches**

Analogy : Imagine you're training for a weightlifting competition. Instead of adjusting your technique after lifting just **one weight(SGD)** or waiting until you've completed an entire month of training (Batch GD), you evaluate your progress **every few days(mini batch).** This balances learning speed and stability .Hence Mini batch is **faster than Batch GD, more stable than SGD**

6. Variants of GD :

RMSprop (Root Mean Square Propagation) –Uses a **moving average** of past gradients .Reduces large swings in weight updates, preventing divergence. Its like walking by adjusting speed **based on past mistakes.**

This works well for **deep networks**

Adam (Adaptive Moment Estimation) – Combines **SGD** and **RMS**. Uses momentum to smooth weight updates and accelerate convergence. Faster convergence, works well for **most ML problems.**

The major differences between them can be seen through **graph in further tutorials**

Where are these techniques used ?

Image Recognition Used in **CNNs** for object detection and facial recognition.

Natural Language Processing (NLP) -Helps train **chatbots** like ChatGPT! and **LSTMs** for language translation, and text summarization.

Autonomous Vehicles -Enables **self-driving cars** to recognize roads, obstacles, and pedestrians by learning from millions of images.

Medical Diagnosis - Used in **deep learning models** to detect diseases from X-rays and MRIs with high accuracy.

```
s | + Code + Text
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Normalize images to [0,1] range
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Flattening images
x_train = x_train.reshape(x_train.shape[0], 28 * 28)
x_test = x_test.reshape(x_test.shape[0], 28 * 28)

# Lets bring a neural network
model = keras.Sequential([
    keras.layers.Dense(128, activation='relu', input_shape=(784,)),
    keras.layers.Dense(10, activation='softmax')
])

# Compile model with backpropagation (SGD optimizer)
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Training the model
history = model.fit(x_train, y_train, epochs=10, batch_size=20, validation_data=(x_tes

# Ploting loss curve
plt.figure(figsize=(8,6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid()
plt.show()
```

This code is used for backpropagation.

We experimented with several optimizers, including Adam, RMSprop, and SGD, and found that the **Adam** optimizer **performed best** for smaller datasets, offering best speed and stability.

The batch size can vary, but an optimal range for smaller datasets is between **10 and 50.**

7. Refrences:

Reference List Citation: James, G., Witten, D., Hastie, T., and Taylor, J. (2023) 'Backpropagation and Gradient Descent in Neural Networks', 0105: *Machine Learning and Neural Networks*. University of Hertfordshire. Unpublished assignment.

In-text Citation: *(James et al., 2023, pp. 428-429)*

Medium Article Citation: Yen, N. (2023) 'A Beginner's Guide to Gradient Descent', *Medium*, 5 May. Available at: https://medium.com/@yennhi95zz/4-a-beginners-guide-to-gradient-descent-in-machine-learning-773ba7cd3dfe [Accessed 24 March 2025].