ARTIFICIAL NEURAL NETWORKS - WEEK 3

Optimizing the Loss Function

Dr. Aamir Akbar

Director of both AWKUM AI Lab and AWKUM Robotics, Final Year Projects (FYPs) coordinator, and lecturer at the department of Computer Science
Abdul Wali Khan University, Mardan (AWKUM)

CONTENTS

WHAT IS THE LOSS FUNCTION IN ANN?

The `loss`, also known as the `cost or error`, represents how well the model's predictions match the actual target values. In other words, the difference between the predicted value (or label) $\hat{y}$ and the actual label y.

`Note: Loss is evaluated after a single training step and returns a value to assess the quality of the model or the prediction.`

The choice of loss function depends on the type of machine learning task being performed, such as `classification`, `regression`, or `clustering`.

The choice of loss function affects the training dynamics, convergence behavior, and generalization performance of the model.

HOW TO DEFINE A LOSS FUNCTION FOR AN ANN?

A loss function $\mathcal{L}$ can be defined as:

$$\mathcal{L} : Y \times Y \to \mathbb{R}^{+} \tag{1}$$

Equation 1 describes a loss function that operates on pairs of target outputs and returns a non-negative real number, reflecting the discrepancy between predicted and true outputs.

- Y represents the set of possible target outputs or labels. In the context of supervised learning, it typically refers to the set of all possible classes or regression targets.
- $Y \times Y$ represents all possible pairs of target outputs.
- $\mathbb{R}^{+}$ indicates that the output of the loss function is constrained to be non-negative, meaning that the loss value cannot be negative.

In other words, $\mathcal{L}(y, \hat{y})$ is a measure of how bad it is to predict $\hat{y}$ given that the true label is y.

SOME COMMON EXAMPLES OF LOSS FUNCTIONS

1. **Absolute value loss or L1 loss** is defined as:

$$\mathcal{L}_1(y, \hat{y}) = |y - \hat{y}| \tag{2}$$

2. **Quadratic or L2 loss** is defined as:

$$\mathcal{L}_2(y, \hat{y}) = (y - \hat{y})^2 \tag{3}$$

3. **0-1 loss** is defined as:

$$\mathcal{L}_{01}(y, \hat{y}) = \mathbb{I}(y \neq \hat{y}) = \begin{cases} 0 & \text{if } y = \hat{y} \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

4. **Cross-entropy loss** is defined as:

$$\mathcal{L}_{CE}(y, \hat{y}) = ylog(\hat{y}) + (1 - y)log(1 - \hat{y}) = \begin{cases} log(\hat{y}) & when\ y = 1 \\ log(1 - \hat{y}) & when\ y = 0 \end{cases} \tag{5}$$

SOME COMMON EXAMPLES OF LOSS FUNCTIONS

Note: *$\mathcal{L}_{CE}$ is only defined for binary outcomes y, while the predicted label $\hat{y}$ can be any value and is usually assumed to be a class probability.*

Element wise: Suppose, a learning model $\mathcal{A}$, for the i-th observation, x_i is the i-th data point, and y_i is the actually observed value, the predicted value of the model is:

$$\hat{y}_i = f(x_i) \tag{6}$$

Depending on the ML task, e.g., classification or regression, different Loss functions are useful.

LOSS FUNCTIONS FOR CLASSIFICATION

1. **Mean Mis-Classification Error** is defined as:

$$MMCE = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_{01}(y_i, \hat{y}_i) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{I}(y_i \neq \hat{y}_i) \tag{7}$$

2. **Binary cross-entropy loss** is defined as:

$$BCE = -\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_{CE}(y_i, \hat{y}_i) \tag{8}$$

3. For the final modal performance, **Accuracy** is defined as:

$$ACC = \frac{1}{n}\sum_{i=1}^{n}1 - \mathcal{L}_{01}(y_i, \hat{y}_i) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{I}(y_i = \hat{y}_i) \tag{9}$$

Note: *Accuracy is a metric used to evaluate the performance of a classification model. It is often defined as* $\mathcal{L}_{01}$*, However, this doesn't mean that accuracy itself is a loss function.*

LOSS FUNCTIONS FOR REGRESSION

1. **Mean Absolute Error** is defined as:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_1(y_i, \hat{y}_i) \tag{10}$$

2. **Mean Squared Error** is defined as:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_2(y_i, \hat{y}_i) \tag{11}$$

3. **Root Mean Squared Error** is defined as:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_2(y_i, \hat{y}_i)} \tag{12}$$

WHAT IS OPTIMIZATION?

Optimization is the study of how to do things in the `best` possible way. Depending on context, best might mean `fastest, cheapest, biggest, most profitable, most efficient`, or some other notion of optimality.

WHAT IS OPTIMIZATION?

Optimization is the study of how to do things in the `best` possible way. Depending on context, best might mean `fastest`, `cheapest`, `biggest`, `most profitable`, `most efficient`, or some other notion of optimality.

Activity: An Optimization Problem

Imagine designing a rectangular box optimized to accommodate as many items as feasible, under two particular considerations. 1) The box must possess a square cross-sectional area, measuring x inches in width and x inches in depth. 2) It must adhere to the size restrictions for carry-on luggage imposed by a specific airline's overhead bins. As per their regulations, the sum of the box's width, depth, and height must not surpass 45 inches. What value of x produces a box with the greatest achievable volume?

OPTIMIZATION - DESIGNING BOX

Solving with common sense

```
width = 10in, depth = 10in, height = 25in
so, 10 + 10 + 25 = 45
and, volume = 10 × 10 × 25 = 2,500 cubic inches
```

Would a cube-shaped box be better?

```
15 + 15 + 15 = 45
volume = 15 × 15 × 15 = 3,375 cubic inches
```

OPTIMIZATION - DESIGNING BOX

Problem Formulation Using Algebra

Since x = width = depth,
so, we have width plus depth = $2x$.
Since width + depth cannot exceed 45 inches,
so, height = $45 - 2x$.
Volume = $x \cdot x \cdot (45 - 2x) = (45x^2 - 2x^3)$
Therefore, $V(x) = (45x^2 - 2x^3)$.

OPTIMIZATION - DESIGNING BOX

Plotting x horizontally and $V(x)$ vertically

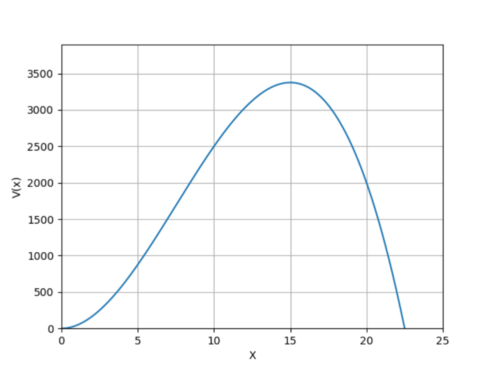


Figure: we see that the curve rises up and reaches its maximum when $x = 15$ inches, as expected, and then descends back to zero.

OPTIMIZATION - DESIGNING BOX

How to find that maximum with differential calculus?

Since the slope is measured by the derivative, `the derivative must be zero at the maximum`. Therefore, to find that maximum with differential calculus, take the derivative of V and set it equal to 0.

1. $\frac{dV}{dx} = \frac{d}{dx}(45x^2) - \frac{d}{dx}(2x^3)$
2. $\frac{dV}{dx} = 45 \cdot 2x^{2-1} - 2 \cdot 3x^{3-1}$
3. $\frac{dV}{dx} = 90x - 6x^2$

Now, we set the derivative equal to zero and solve for x:

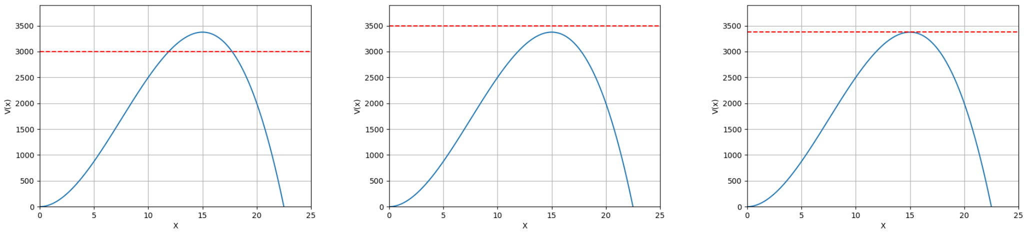
4. $90x - 6x^2 = 0$
5. $x(90 - 6x) = 0$

Now, either $x = 0$ or $90 - 6x = 0$. If $x = 0$, then $V(x) = 0$. If $90 - 6x = 0$, then $x = \frac{90}{6} = 15$.

OPTIMIZATION - DESIGNING BOX

How did the derivative solve the problem?

Horizontal lines `below the maximum` intersect the curve at two points (i.e., a and b), whereas horizontal lines `above the maximum` don't intersect the curve at all. `At the maximum`, the two points collide and become `sort equal`.



OPTIMIZATION - DESIGNING BOX

At the maximum, the two points a and b become `sort of equal`: $a \approx b$. This leads to $V(a) = V(b)$ such that:

1. $45a^2 - 2a^3 = 45b^2 - 2b^3$
2. $45a^2 - 45b^2 = 2a^3 - 2b^3$
3. $45(a - b)(a + b) = 2(a - b)(a^2 + ab + b^2)$
4. $45(a + b) = 2(a^2 + ab + b^2)$

when a and b do become equal as they merge at the maximum, the equation becomes:

5. $45(2a) = 2(a^2 + a^2 + a^2)$
6. $90a = 6a^2$

Solutions are $a = 0$ and $a = 15$. 1) $a = 0$ gives a box of minimum volume; it has zero width and depth and hence has zero volume. That's of no interest. 2) $a = 15$ gives the box of maximum volume, the answer we've been expecting: 15 inches is the optimal width and depth.

LOSS OPTIMIZATION

A loss function measures how well the model's predictions match the actual target values. The goal of optimization is to `find the set of model parameters that minimize this loss function`, therefore, improving the model's performance on the task at hand.

LOSS OPTIMIZATION

A loss function measures how well the model's predictions match the actual target values. The goal of optimization is to `find the set of model parameters that minimize this loss function`, therefore, improving the model's performance on the task at hand.

The objective is to find the network weights that `achieve the lowest loss`.

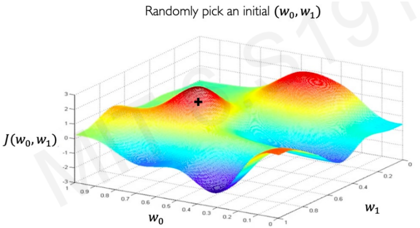
$$W^* = \underset{W}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, f(x_i; W))$$

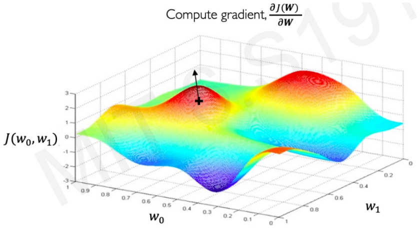
The value of the loss function, which measures how well the model is performing on a given task, depends on the weights of the neural network.

$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$

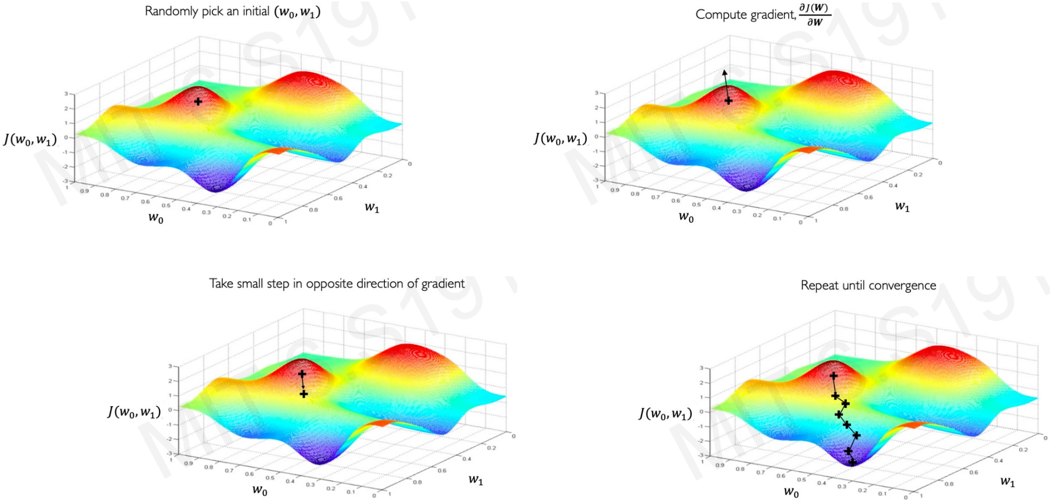
where $J(W)$ represents the overall loss function, which is the average of the individual loss functions $\mathcal{L}_i$ over the entire dataset. W^* represents the optimal value or configuration of the model parameters W that minimizes the overall loss function $J(W)$.

HOW TO ACHIEVE THE LOWEST LOSS?





HOW TO ACHIEVE THE LOWEST LOSS?



ALGORITHM: GRADIENT DESCENT

Input: A dataset $\mathcal{D} = (X, y)$
Initialize model parameters W
Set learning rate η
Set number of iterations E
for $e = 1$ to E **do**
  for $(x_i, y_i) \in \mathcal{D}$ **do**
    Forward pass and compute $\hat{y}$, as in Eq. 6: $\hat{y}_i = f(x_i)$
    Compute the gradient: $\nabla \mathcal{L}_i(y_i, f(x_i; W))$
  end for
  Compute overall gradient: $\nabla \mathcal{L}(W) \leftarrow \frac{1}{n} \sum_{i=1}^{n} \nabla \mathcal{L}_i$
  Update modal parameters: $W \leftarrow W - \eta \nabla \mathcal{L}(W)$
end for

This is `batch gradient descent`, which calculates the gradient using the entire dataset before updating the model parameters. As model parameters are updated once per epoch, it means a full pass over the whole dataset is required before the update can occur. When dealing with large datasets, this is a strong limitation, which motivates the use of stochastic variants.

ALGORITHM: STOCHASTIC GRADIENT DESCENT

Input: A dataset $\mathcal{D} = (X, y)$
Initialize model parameters W
Set learning rate η
Set number of iterations E
for $e = 1$ to E **do**
    for $(x_i, y_i) \in \mathcal{D}$ **do**
        Forward pass and compute $\hat{y}$, as in Eq. 6: $\hat{y}_i = f(x_i)$
        Compute the gradient: $\nabla \mathcal{L}_i(y_i, f(x_i; W))$
        Update model parameters: $W \leftarrow W - \eta \nabla \mathcal{L}_i(W)$
    end for
end for

Note: $\nabla \mathcal{L}(W)$ *denotes the gradient of the overall loss function* $J(W)$ *with respect to the model parameters* W.

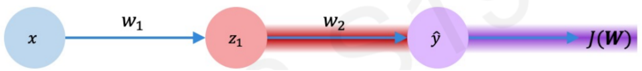
BACKPROPAGATION VS. GRADIENT DESCENT

- **Gradient Descent (GD)**:
  - Optimization algorithm used to minimize the loss function in machine learning models.
  - Iteratively adjusts the parameters of the model in the direction of the steepest descent of the loss function.
  - Updates parameters by subtracting the gradient of the loss function multiplied by a small learning rate.
- **Backpropagation**:
  - Specific algorithm for efficiently computing gradients in a neural network.
  - Utilizes the chain rule of calculus to compute the gradient of the loss function with respect to each parameter in the network.
  - Involves two phases: forward pass (computing predictions) and backward pass (computing gradients).

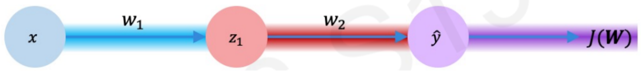
Main Difference:

- Gradient descent is an optimization algorithm used to minimize any differentiable function, while backpropagation is a specific algorithm for efficiently computing gradients in neural networks.

BACKPROPAGATION: CHAIN RULE



$$\frac{\partial J(W)}{\partial w_2} = \frac{\partial J(W)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2} \tag{13}$$



$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1} \tag{14}$$

Equation 14 represents the gradient of the overall loss function $J(W)$ with respect to the weights w_1, calculated using the chain rule and the individual components of the backpropagation process.

PYTHON IMPLEMENTATION USING TENSORFLOW

```python
import tensorflow as tf
from tensorflow.keras.layers import InputLayer
from tensorflow.keras.layers import Dense
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# Generate synthetic data
X, y = make_classification(n_samples=1000,
n_features=5, n_classes=2, random_state=42)

# Split data into train and test sets
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2, random_state=42)


# Define the model
class MLP(tf.keras.Model):
    def __init__(self, input_dim, hidden_dim1, hidden_dim2, output_dim)
        super(MLP, self).__init__()
        self.input_layer = InputLayer(input_shape=(input_dim,))
        self.dense1 = Dense(hidden_dim1, activation="relu")
        self.dense2 = Dense(hidden_dim2, activation="relu")
        self.dense3 = Dense(output_dim,
activation="tanh")

    def call(self, inputs):
        x = self.input_layer(inputs)
        x = self.dense1(x)
        x = self.dense2(x)
        return self.dense3(x)
```

```python
# Create an instance of the model
model = MLP(input_dim=5, hidden_dim1=7, hidden_dim2=7,
output_dim=1)

# Print model summary
model.build((None, 5))
model.summary()

# Compile the model
model.compile(optimizer=tf.keras.optimizers.SGD(0.002),
 loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32,
 validation_data=(X_test, y_test))

# Evaluate the model on test data
loss, accuracy = model.evaluate(X_test, y_test)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

PYTHON IMPLEMENTATION USING PYTORCH

```python
import torch
import torch.nn as nn
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# Generate synthetic data
X, y = make_classification(n_samples=1000, n_features=5, n_classes=2, random_state=42)

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Convert data to PyTorch tensors
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train.reshape(-1, 1), dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test.reshape(-1, 1), dtype=torch.float32)

# Define the model

# Train the model
epochs = 50
batch_size = 32
for epoch in range(epochs):
    for batch_start in range(0, len(X_train_tensor), batch_size):
        batch_end = batch_start + batch_size
        X_batch = X_train_tensor[batch_start:batch_end]
        y_batch = y_train_tensor[batch_start:batch_end]

        optimizer.zero_grad()  # Zero the gradients
        outputs = model(X_batch)  # Forward pass
        loss = criterion(outputs, y_batch)  # Calculate the loss
        loss.backward()  # Backward pass
        optimizer.step()  # Update the parameters

# Evaluate the model on test data
with torch.no_grad():
    test_outputs = model(X_test_tensor)
    test_loss = criterion(test_outputs, y_test_tensor)
    test_accuracy = ((torch.round(torch.sigmoid(test_outputs)) == y_test_tensor).sum().item()) / len(y_test_tensor)
    print("Test Loss:", test_loss.item())
    print("Test Accuracy:", test_accuracy)
```

RESOURCES

To download the source codes used in the previous slides, follow the link:

▸ Download Source Codes

Import the codes into your preferred development environment, such as Visual Studio Code (VS Code), to practice and explore further.

To learn programming in Python, follow my comprehensive 15-week Programming in Python course at:

▸ Programming in Python