TP # 6: Differential Perceptron (Sigmoid) Molecular Descriptor Classification

Dr. Samir Kenouche – November 15, 2025

Abstract

This document presents an *advanced matrix formulation* of a differentiable (sigmoid) perceptron applied to molecular descriptor classification. We give full symbolic derivations of forward, backward, and parameter-update formulae, then compute every arithmetic step for the **first four** gradient-descent iterations on a small, concrete molecular dataset. All steps are presented in matrix form and expanded into elementwise arithmetic to ensure reproducibility by the students.

Contents

1 Problem statement and notation

We consider binary classification of molecules using a single logistic neuron (sigmoid perceptron). Each molecule i has a descriptor vector $\mathbf{x}^{(i)} \in \mathbb{R}^d$ and a binary label $y^{(i)} \in \{0,1\}$. The dataset with N samples is written as the matrix

$$\mathbf{X} = \begin{bmatrix} (\mathbf{x}^{(1)})^\top \\ (\mathbf{x}^{(2)})^\top \\ \vdots \\ (\mathbf{x}^{(N)})^\top \end{bmatrix} \in \mathbb{R}^{N\times d}, \qquad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix} \in \{0,1\}^N.$$

Parameters: weight column vector $\mathbf{W} \in \mathbb{R}^{d\times 1}$ and scalar bias $b \in \mathbb{R}$. For compactness we will sometimes write bias as a separate scalar added to each row. Model (vectorised over the batch):

$$\mathbf{z} = \mathbf{X}\,\mathbf{W} + b\,\mathbf{1}_N, \qquad \mathbf{a} = \sigma(\mathbf{z}) = \frac{1}{1+e^{-\mathbf{z}}} \quad \text{(elementwise)}.$$

Loss (average binary cross-entropy):

$$\mathcal{L}(\mathbf{W}, b) = -\frac{1}{N}\left(\mathbf{y}^\top \log \mathbf{a} + (\mathbf{1}-\mathbf{y})^\top \log(\mathbf{1}-\mathbf{a})\right).$$

2 Differential (analytic) gradients — full derivation

We derive gradients symbolically before numerical evaluation.

For a single sample index i,

$$z^{(i)} = \mathbf{x}^{(i)\top}\mathbf{W} + b, \qquad a^{(i)} = \sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}.$$

Single-sample loss:

$$\ell^{(i)} = -\big(y^{(i)} \log a^{(i)} + (1-y^{(i)})\log(1-a^{(i)})\big).$$

Derivative of $\ell^{(i)}$ w.r.t. (with respect to) $z^{(i)}$ (use chain rule and $\frac{d}{dz}\sigma(z) = \sigma(z)(1-\sigma(z))$):

$$\begin{aligned}
\frac{d\ell^{(i)}}{dz^{(i)}} &= -\left(y^{(i)}\frac{1}{a^{(i)}}\frac{da^{(i)}}{dz^{(i)}} - (1-y^{(i)})\frac{1}{1-a^{(i)}}\frac{da^{(i)}}{dz^{(i)}}\right)\\
&= -\frac{da^{(i)}}{dz^{(i)}}\left(\frac{y^{(i)}}{a^{(i)}} - \frac{1-y^{(i)}}{1-a^{(i)}}\right)\\
&= -a^{(i)}(1-a^{(i)})\left(\frac{y^{(i)}(1-a^{(i)}) - (1-y^{(i)})a^{(i)}}{a^{(i)}(1-a^{(i)})}\right)\\
&= -(y^{(i)} - a^{(i)}) = a^{(i)} - y^{(i)}.
\end{aligned}$$

Thus the convenient simplification holds (batch form):

$$\nabla_{\mathbf{z}}\mathcal{L} = \frac{1}{N}(\mathbf{a}-\mathbf{y}).$$

Then by matrix calculus,

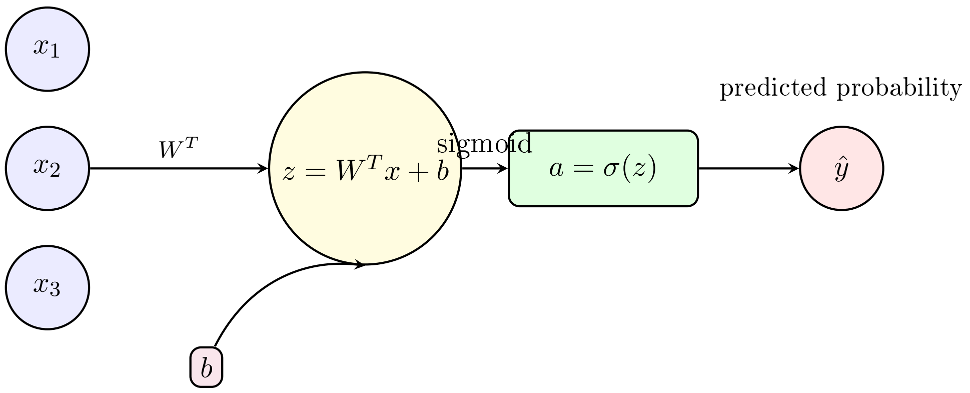
$$\nabla_{\mathbf{W}}\mathcal{L} = \mathbf{X}^\top(\nabla_{\mathbf{z}}\mathcal{L}) = \frac{1}{N}\mathbf{X}^\top(\mathbf{a}-\mathbf{y}), \qquad \nabla_b\mathcal{L} = \mathbf{1}_N^\top(\nabla_{\mathbf{z}}\mathcal{L}) = \frac{1}{N}\mathbf{1}_N^\top(\mathbf{a}-\mathbf{y}).$$

These are exact analytic (differential) gradients used for gradient descent.

3 Numerical worked example: molecular descriptors

We now apply the above to a small molecular dataset and show all computations for the first four iterations.

3.1 Perceptron Scheme



3.2 Dataset and initialisation

We use $N = 3$ molecules and $d = 3$ descriptors per molecule (e.g. hydrophobicity, polar surface, partial charge). The feature matrix, labels, and initial parameters are:

$$\mathbf{X} = \begin{bmatrix} 1.00 & 0.50 & 0.20 \\ 1.50 & -0.30 & 0.80 \\ 0.30 & 0.70 & -0.50 \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}.$$

Initial weights and bias (column vector):

$$\mathbf{W}^{(0)} = \begin{bmatrix} 0.10 \\ -0.20 \\ 0.30 \end{bmatrix}, \qquad b^{(0)} = 0.0, \qquad \eta = 0.1.$$

We will compute: $\mathbf{z}^{(t)}$, $\mathbf{a}^{(t)}$, $\mathcal{L}^{(t)}$, $\nabla_{\mathbf{W}}\mathcal{L}^{(t)}$, $\nabla_b\mathcal{L}^{(t)}$, and updates for $t = 0, 1, 2, 3$ (four iterations index starting at 0).

Iteration 0 (initial parameters)

Forward: pre-activation and activation

Compute pre-activations (matrix multiplication):

$$\mathbf{z}^{(0)} = \mathbf{X}\mathbf{W}^{(0)} + b^{(0)}\mathbf{1}_3 = \begin{bmatrix} 1.0 & 0.5 & 0.2 \\ 1.5 & -0.3 & 0.8 \\ 0.3 & 0.7 & -0.5 \end{bmatrix} \begin{bmatrix} 0.1 \\ -0.2 \\ 0.3 \end{bmatrix} = \begin{bmatrix} 0.06 \\ 0.45 \\ -0.26 \end{bmatrix}.$$

Elementwise sigmoid:

$$\mathbf{a}^{(0)} = \sigma(\mathbf{z}^{(0)}) = \begin{bmatrix} \sigma(0.06) \\ \sigma(0.45) \\ \sigma(-0.26) \end{bmatrix} = \begin{bmatrix} 0.51499550 \\ 0.61063923 \\ 0.43536371 \end{bmatrix}.$$

Loss and backward: BCE and gradients

Loss (average BCE):

$$\mathcal{L}^{(0)} = -\frac{1}{3}\sum_{i=1}^{3}\left[y^{(i)}\log a^{(i)} + (1 - y^{(i)})\log(1 - a^{(i)})\right] = 0.81280652.$$

Error term (batch):

$$\nabla_{\mathbf{z}}\mathcal{L}^{(0)} = \frac{1}{3}(\mathbf{a}^{(0)} - \mathbf{y}) = \frac{1}{3}\begin{bmatrix} 0.51499550 - 1 \\ 0.61063923 - 0 \\ 0.43536371 - 1 \end{bmatrix} = \begin{bmatrix} -0.16166817 \\ 0.20354641 \\ -0.18887876 \end{bmatrix}.$$

Gradient wrt weights (matrix form):

$$\nabla_{\mathbf{W}}\mathcal{L}^{(0)} = \mathbf{X}^\top(\nabla_{\mathbf{z}}\mathcal{L}^{(0)}) = \begin{bmatrix} 1.0 & 1.5 & 0.3 \\ 0.5 & -0.3 & 0.7 \\ 0.2 & 0.8 & -0.5 \end{bmatrix} \begin{bmatrix} -0.16166817 \\ 0.20354641 \\ -0.18887876 \end{bmatrix} = \begin{bmatrix} 0.08718782 \\ -0.27364647 \\ 0.22460954 \end{bmatrix}.$$

Gradient wrt bias:

$$\nabla_b\mathcal{L}^{(0)} = \mathbf{1}_3^\top(\nabla_{\mathbf{z}}\mathcal{L}^{(0)}) = -0.14633385.$$

Parameter update

Gradient descent update (learning rate $\eta = 0.1$):

$$\mathbf{W}^{(1)} = \mathbf{W}^{(0)} - 0.1\,\nabla_{\mathbf{W}}\mathcal{L}^{(0)} = \begin{bmatrix} 0.09128122 \\ -0.17263535 \\ 0.27753905 \end{bmatrix},$$

$$b^{(1)} = b^{(0)} - 0.1\,\nabla_b\mathcal{L}^{(0)} = 0.01463339.$$

Iteration 1

Forward

Compute pre-activation with updated parameters:

$$\mathbf{z}^{(1)} = \mathbf{X}\mathbf{W}^{(1)} + b^{(1)}\mathbf{1}_3 = \begin{bmatrix} 0.07510474 \\ 0.42537705 \\ -0.21759652 \end{bmatrix}.$$

Activations:

$$\mathbf{a}^{(1)} = \sigma(\mathbf{z}^{(1)}) = \begin{bmatrix} 0.51876736 \\ 0.60476921 \\ 0.44581450 \end{bmatrix}.$$

Loss and backward

Loss:

$$\mathcal{L}^{(1)} = 0.79747916.$$

Gradients:

$$\nabla_{\mathbf{z}}\mathcal{L}^{(1)} = \frac{1}{3}(\mathbf{a}^{(1)} - \mathbf{y}) = \begin{bmatrix} -0.16041088 \\ 0.20158974 \\ -0.18473850 \end{bmatrix}.$$

$$\nabla_{\mathbf{W}}\mathcal{L}^{(1)} = \mathbf{X}^\top\nabla_{\mathbf{z}}\mathcal{L}^{(1)} = \begin{bmatrix} 0.08655518 \\ -0.26999231 \\ 0.22155386 \end{bmatrix}, \quad \nabla_b\mathcal{L}^{(1)} = -0.14354964.$$

Update

$$\mathbf{W}^{(2)} = \mathbf{W}^{(1)} - 0.1\,\nabla_{\mathbf{W}}\mathcal{L}^{(1)} = \begin{bmatrix} 0.08262570 \\ -0.14563612 \\ 0.25538366 \end{bmatrix},$$

$$b^{(2)} = b^{(1)} - 0.1\,\nabla_b\mathcal{L}^{(1)} = 0.02898835.$$

Iteration 2

Forward

$$\mathbf{z}^{(2)} = \mathbf{X}\mathbf{W}^{(2)} + b^{(2)}\mathbf{1}_3 = \begin{bmatrix} 0.08786669 \\ 0.38657851 \\ -0.17792091 \end{bmatrix},$$

$$\mathbf{a}^{(2)} = \sigma(\mathbf{z}^{(2)}) = \begin{bmatrix} 0.52245307 \\ 0.59890980 \\ 0.45614770 \end{bmatrix}.$$

Loss and backward

$$\mathcal{L}^{(2)} = 0.78257589,$$

$$\nabla_{\mathbf{z}}\mathcal{L}^{(2)} = \frac{1}{3}(\mathbf{a}^{(2)} - \mathbf{y}) = \begin{bmatrix} -0.15918231 \\ 0.19963660 \\ -0.18128410 \end{bmatrix},$$

$$\nabla_{\mathbf{W}}\mathcal{L}^{(2)} = \mathbf{X}^{\top}\nabla_{\mathbf{z}}\mathcal{L}^{(2)} = \begin{bmatrix} 0.08588736 \\ -0.26638101 \\ 0.21851487 \end{bmatrix}, \quad \nabla_b\mathcal{L}^{(2)} = -0.14082981.$$

Update

$$\mathbf{W}^{(3)} = \mathbf{W}^{(2)} - 0.1\,\nabla_{\mathbf{W}}\mathcal{L}^{(2)} = \begin{bmatrix} 0.07403696 \\ -0.11899802 \\ 0.23353217 \end{bmatrix},$$

$$b^{(3)} = b^{(2)} - 0.1\,\nabla_b\mathcal{L}^{(2)} = 0.04307133.$$

Iteration 3

Forward

$$\mathbf{z}^{(3)} = \mathbf{X}\mathbf{W}^{(3)} + b^{(3)}\mathbf{1}_3 = \begin{bmatrix} 0.10431572 \\ 0.37665192 \\ -0.13478228 \end{bmatrix},$$

$$\mathbf{a}^{(3)} = \sigma(\mathbf{z}^{(3)}) = \begin{bmatrix} 0.52605531 \\ 0.59306533 \\ 0.46635535 \end{bmatrix}.$$

Loss and backward

$$\mathcal{L}^{(3)} = 0.76808632,$$

$$\nabla_{\mathbf{z}}\mathcal{L}^{(3)} = \frac{1}{3}(\mathbf{a}^{(3)} - \mathbf{y}) = \begin{bmatrix} -0.15798156 \\ 0.19768844 \\ -0.17788155 \end{bmatrix},$$

$$\nabla_{\mathbf{W}}\mathcal{L}^{(3)} = \mathbf{X}^{\top}\nabla_{\mathbf{z}}\mathcal{L}^{(3)} = \begin{bmatrix} 0.08518664 \\ -0.26281440 \\ 0.21549522 \end{bmatrix}, \quad \nabla_b\mathcal{L}^{(3)} = -0.13817467.$$

Update

$$\mathbf{W}^{(4)} = \mathbf{W}^{(3)} - 0.1\,\nabla_{\mathbf{W}}\mathcal{L}^{(3)} = \begin{bmatrix} 0.06551830 \\ -0.09271658 \\ 0.21198265 \end{bmatrix},$$

$$b^{(4)} = b^{(3)} - 0.1\,\nabla_b\mathcal{L}^{(3)} = 0.05688880.$$

4 Summary and meaning

- The perceptron computes probabilities via the sigmoid activation; Binary Cross-Entropy matches that probabilistic interpretation and yields the simple error signal $(a - y)$.

- The matrix formulation is efficient: the batch error $(\mathbf{a} - \mathbf{y})$ is multiplied by $\mathbf{X}^{\top}$ to aggregate feature contributions to each weight.

- Each gradient step reduces the loss (observed numerically over the first four iterations). The update moves weights opposite to the gradient direction because we perform gradient *descent*.

Chemical Interpretation

In this molecular descriptor classification problem, each input feature x_j represents a chemical property, such as hydrophobicity, polar surface area, or partial charge of a molecule. The weight w_j quantifies how strongly that descriptor contributes to the predicted probability that a molecule is toxic ($y = 1$). A positive weight means the descriptor increases the likelihood of toxicity, while a negative weight decreases it.

The bias term b acts as a baseline chemical propensity, adjusting the threshold for toxicity independent of specific descriptors. After the linear combination $z = W^T x + b$, the sigmoid activation maps the chemical signal into a probability $a = \sigma(z)$, giving a biologically interpretable likelihood of the molecule being toxic.

The binary cross-entropy loss measures the discrepancy between predicted probabilities and actual observed toxicities, guiding the gradient descent to adjust weights in chemically meaningful directions.

5 Python Code

The following Python code has been reformatted for readability, clarity, and cleaner output, while preserving the same mathematical operations:

Executable Python Code

```python
import numpy as np
# Dr. Samir Kenouche

# Molecular descriptor data
X = np.array([
    [1.0, 0.5, 0.2],  # molecule 1
    [1.5, -0.3, 0.8], # molecule 2
    [0.3, 0.7, -0.5]  # molecule 3
])

y = np.array([1, 0, 1])

# Initialize weights and bias
W = np.array([0.1, -0.2, 0.3])
b = 0.0
eta = 0.1

# Activation function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Training loop for 4 iterations
for t in range(4):
    z = X @ W + b
    a = sigmoid(z)

    # Gradients
    grad = (a - y) / len(y)
    W = W - eta * (X.T @ grad)
    b = b - eta * grad.sum()

    # Loss (binary cross-entropy)
    loss = -np.mean(y * np.log(a + 1e-12) + (1 - y) * np.log(1 - a + 1e-12))

    print(f"Iteration {t}: Loss = {loss:.5f}, W = {W}, b = {b:.5f}")

print("\nFinal weights and bias:")
print(f"W = {W}, b = {b:.5f}")
```

Step-by-Step Calculation of $\mathcal{L}^{(0)}$

This section shows the detailed calculation of the binary cross-entropy loss for the first iteration (iteration 0) of the perceptron training.

Iteration 0 Loss Calculation

Input:

$$X = \begin{bmatrix} 1.0 & 0.5 & 0.2 \\ 1.5 & -0.3 & 0.8 \\ 0.3 & 0.7 & -0.5 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \quad W^{(0)} = \begin{bmatrix} 0.1 \\ -0.2 \\ 0.3 \end{bmatrix}, \quad b^{(0)} = 0$$

Step 1: Linear combination

$$z = XW^{(0)} + b^{(0)} = \begin{bmatrix} 0.06 \\ 0.45 \\ -0.26 \end{bmatrix}$$

Step 2: Sigmoid activation

$$a = \sigma(z) = \frac{1}{1+e^{-z}} \approx \begin{bmatrix} 0.5150 \\ 0.6106 \\ 0.4353 \end{bmatrix}$$

Step 3: Binary cross-entropy loss

$$\mathcal{L}^{(0)} = -\frac{1}{3}\sum_{i=1}^{3}\Big[y_i \log(a_i) + (1-y_i)\log(1-a_i)\Big]$$

Compute each term:

$$-\log(0.5150) \approx 0.663, \quad -\log(1-0.6106) \approx 0.944, \quad -\log(0.4353) \approx 0.832$$

Sum: $0.663 + 0.944 + 0.832 = 2.439$
Divide by 3:

$$\mathcal{L}^{(0)} \approx \frac{2.439}{3} \approx 0.813$$

Result:

$$\mathcal{L}^{(0)} \approx 0.813$$

6 Meaning of Binary Cross-Entropy Loss and Comparison with Mean Squared Error

The **binary cross-entropy (BCE) loss** measures how well a model predicts probabilities of binary outcomes (e.g., toxic vs. non-toxic molecules). It compares the predicted probability $a = \sigma(z)$ with the actual label $y \in \{0, 1\}$ using the formula:

$$L_{BCE} = -\frac{1}{N}\sum_{i=1}^{N}\Big[y_i \log(a_i) + (1-y_i)\log(1-a_i)\Big]$$

Interpretation:

- When the prediction a_i is close to the true label y_i, the loss is small.
- When the prediction is far from the true label, the loss increases sharply.
- It is particularly suited for probability outputs (0 to 1) and penalizes confident but wrong predictions heavily.

6.1 Comparison with Mean Squared Error (MSE / RMSE)

- MSE computes the squared difference between the predicted value and the actual label:

$$L_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(a_i - y_i)^2$$

- MSE treats the problem as a regression, while BCE treats it as a probabilistic classification.
- BCE is more sensitive to predictions near 0 or 1, which is important in classification tasks.
- RMSE (square root of MSE) gives the error in the same scale as the predicted probability, but does not penalize confident wrong predictions as strongly as BCE.
- In chemical classification (toxic vs non-toxic), BCE is preferred since it gives a clear probabilistic interpretation of molecule toxicity.

Appendix: Stepwise Summary and Chemical Interpretations

Appendix Overview

This appendix provides a comprehensive, pedagogically-oriented summary of the perceptron calculations, meaning of each function, and strategies to improve chemical classification.

1. Stepwise Calculation Workflow

Step 1: Linear Combination

Formula: $z = W^T X + b$
Meaning: Weighted sum of molecular descriptors plus bias. Each w_j quantifies the contribution of descriptor x_j.

Step 2: Sigmoid Activation

Formula: $a = \sigma(z) = \frac{1}{1+e^{-z}}$
Meaning: Maps linear combination to probability [0,1]; interpretable as the likelihood of a molecule being toxic.

Step 3: Binary Cross-Entropy Loss

Formula: $L = -\frac{1}{N}\sum[y_i \log(a_i) + (1 - y_i)\log(1 - a_i)]$
Meaning: Measures discrepancy between predicted probability and true label; penalizes confident wrong predictions.

Step 4: Gradient Computation

Formula: $\nabla_W L = X^T(a - y)/N$, $\nabla_b L = \sum(a - y)/N$
Meaning: Indicates direction to update weights/bias to minimize loss; derived from chain rule.

Step 5: Weight Update

Formula: $W \leftarrow W - \eta \nabla_W L$, $b \leftarrow b - \eta \nabla_b L$
Meaning: Implements gradient descent; learning rate η controls step size.

2. Function Interpretations and Derivations

Function Roles

- $W^T X + b$: Linear mapping of molecular descriptors.
- $\sigma(z)$: Non-linear transformation ensuring outputs are probabilities.
- L_{BCE}: Loss function guiding parameter optimization.
- $\nabla_W L$, $\nabla_b L$: Gradients from chain rule for backpropagation.

3. Strategies for Improving Chemical Classification

Chemical Classification Improvements

- **Feature Engineering:** Add relevant descriptors (logP, polar surface area, electronic properties).
- **Regularization:** Apply L_1/L_2 penalties to prevent overfitting.
- **Learning Rate Tuning:** Use adaptive optimizers (Adam, RMSProp) for faster convergence.
- **Data Augmentation:** Expand molecular dataset for robustness.
- **Network Complexity:** Consider multi-layer perceptrons for non-linear interactions.
- **Cross-Validation:** Evaluate performance on multiple splits.
- **Interpretability:** Use SHAP or feature importance to understand descriptor influence.