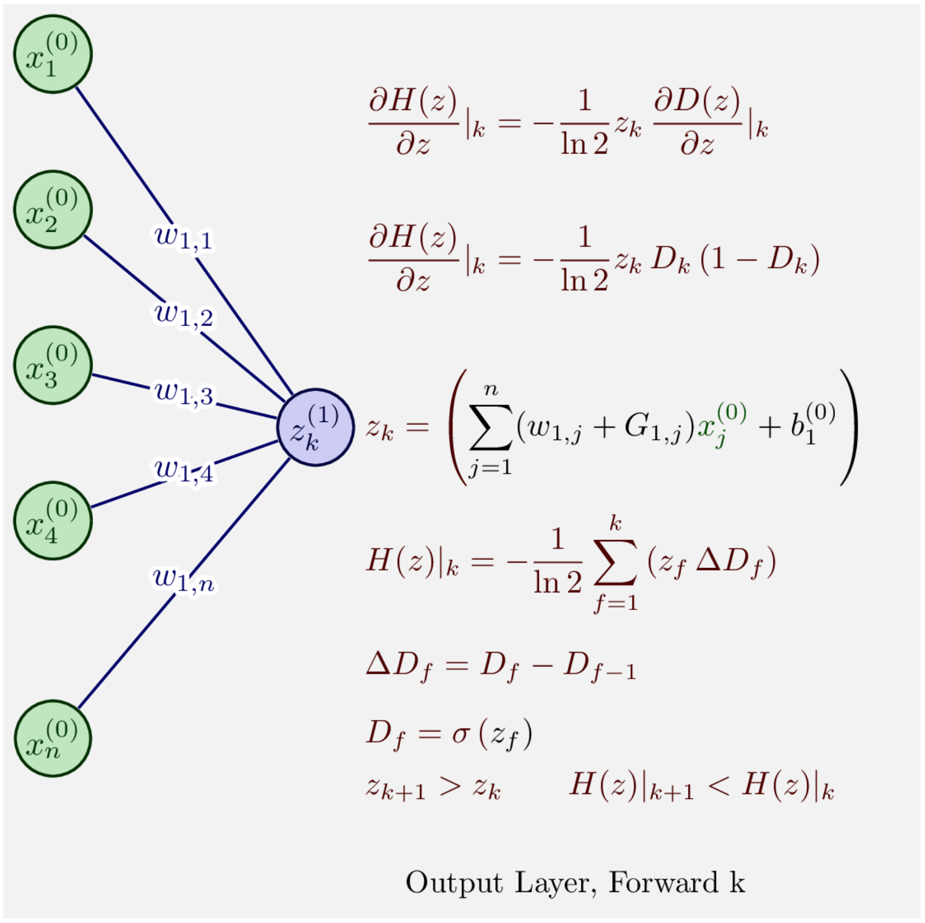

$x_1^{(0)}$
$x_2^{(0)}$
$x_3^{(0)}$
$x_4^{(0)}$
$x_n^{(0)}$
$w_{1,1}$
$w_{1,2}$
$w_{1,3}$
$w_{1,4}$
$w_{1,n}$
$z_k^{(1)}$
$\frac{\partial H(z)}{\partial z}|_k = -\frac{1}{\ln 2} z_k \frac{\partial D(z)}{\partial z}|_k$
$\frac{\partial H(z)}{\partial z}|_k = -\frac{1}{\ln 2} z_k D_k (1 - D_k)$
$z_k = \left(\sum_{j=1}^{n} (w_{1,j} + G_{1,j}) x_j^{(0)} + b_1^{(0)} \right)$
$H(z)|_k = -\frac{1}{\ln 2} \sum_{f=1}^{k} (z_f \, \Delta D_f)$
$\Delta D_f = D_f - D_{f-1}$
$D_f = \sigma(z_f)$
$z_{k+1} > z_k \qquad H(z)|_{k+1} < H(z)|_k$
Output Layer, Forward k

Figure 1: image

Internship Test Assignment: Implementing Entropy-Gradient Parameter Updates with z Mapping Constraints for an Even/Odd Classifier

Objective

Your task is to extend our classical single-layer perceptron model—which classifies MNIST digits as even (0) or odd (1)—by integrating an entropy-gradient framework into the model's parameter update rules. In this approach, you will replace the standard loss-based optimization with updates driven by the gradient of an entropy functional. Additionally, you must enforce a constraint on the knowledge z :

$$z_{i+1} - z_i < \delta,$$

where δ is a fixed parameter that limits how much the knowledge z may change between successive training iterations (not between neurons).

Background

In conventional neural network training, parameters are updated by minimizing a loss function (e.g., crossentropy). Here, we adopt an alternative strategy where entropy is used as the guiding principle. The entropy functional, its gradients, and the parameter update rules are derived from the following equations and constraints.

Mapping from Input to Knowledge z (z mapping)

1. Linear Combination:

$$z_k = \sum_{j=1}^{n} \Big(w_{1,j} + G_{1,j} \Big) x_j^{(0)} + b_1^{(0)}$$

Here:

- $x_j^{(0)}$ are the input features,
- $w_{1,j}$ and $G_{1,j}$ are the two sets of weights,
- $b_1^{(0)}$ is the bias,
- k is the forward step,
- n is the number of input features.

2. Activation Function:

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

The output D_k is then used in subsequent entropy computations.

3. Constraint on z Updates:

To ensure stable learning, **the change in z between consecutive iterations must be limited:**

$$z_{i+1} - z_i < \delta,$$

where:

- δ is a predetermined small constant (suggested range: 0.01 to 0.1),
- i refers to the iteration number (not neuron index),
- This constraint only limits increases in z, not decreases.

Entropy Computation and Gradient Updates

The entropy functional $H(z)|_k$ after the forward step k is given by:

$$H(z) = -\frac{1}{\ln 2} \sum_{f=1}^{k} (z_f \, \Delta D_f)$$

where ΔD_f represents the change in activation from the previous iteration for example f.

After computing z_k and D_k, the gradient of the entropy $H(z)$ with respect to z_k is given by:

$$\left.\frac{\partial H(z)}{\partial z}\right|_k = -\frac{1}{\ln 2} \, z_k \, D_k \, (1 - D_k)$$

Using this gradient, the parameters are updated as follows:

- For $w_{1,j}$:

$$w_{1,j} \leftarrow w_{1,j} - \eta \left(\left.\frac{\partial H(z)}{\partial z}\right|_k \cdot x_j^{(0)} \right)$$

- For $G_{1,j}$:

$$G_{1,j} \leftarrow G_{1,j} - \eta \left(\left.\frac{\partial H(z)}{\partial z}\right|_k \cdot x_j^{(0)} \right)$$

- For $b_1^{(0)}$:

$$b_1^{(0)} \leftarrow b_1^{(0)} - \eta \left(\left.\frac{\partial H(z)}{\partial z}\right|_k \right)$$

Note: In addition to applying these updates, you must ensure that after updating the parameters, the resulting new knowledge z_{i+1} does not violate the constraint:

$$z_{i+1} - z_i < \delta.$$

If an update would violate this constraint, you should scale down the parameter changes proportionally to ensure the constraint is satisfied.

Assignment Requirements

1. **Extend the Classical Model Architecture:**

   - **Dual-Weight Structure:**
     Modify the provided code so that each connection from input $x_j^{(0)}$ to the output node uses the combined parameter $w_{1,j} + G_{1,j}$.
   - **Knowledge Computation:**
     Compute the knowledge as:

$$z_k = \sum_{j=1}^{n} \Big(w_{1,j} + G_{1,j} \Big) x_j^{(0)} + b_1^{(0)}$$

and then obtain:

$$D_k = \sigma(z_k).$$

   - **Enforce the z Mapping Constraint:**

     Ensure that for each training step the change in z adheres to:

$$z_{i+1} - z_i < \delta,$$

where δ is a fixed parameter that you can set as a hyperparameter (recommended range: 0.01 to 0.1).

2. **Implement Entropy-Gradient Based Parameter Updates:**

   Custom Training Loop:

   Bypass TensorFlow's built-in optimizer and implement a custom training loop that:

- Computes the knowledge z_k and output D_k,
- Calculates the entropy gradient:

$$\left. \frac{\partial H(z)}{\partial z} \right|_k = -\frac{1}{\ln 2} \, z_k \, D_k \left(1 - D_k\right),$$

- Updates the parameters $w_{1,j}$, $G_{1,j}$, and $b_1^{(0)}$ accordingly,
- Enforces the constraint $z_{i+1} - z_i < \delta$ by scaling down updates if necessary.
- Adjust dynamically thelearning rate η

1. **Maintain the Even/Odd Classification Task:**

   - Use the MNIST dataset, converting digit labels into even (0) and odd (1) as in the classical example.

- Implement the training loop over the training samples (or mini-batches) using your custom update rule and z mapping constraint.
- For mini-batch training, apply the constraint on a per-example basis.

2. **Testing and Visualization:**
   - After training, evaluate the classifier on the test set and print the overall classification accuracy.
   - Display 10 random test images along with their true even/odd labels and predictions, similar to the provided base code.
   - Track and plot the accuracy throughout training.

3. **Documentation and Reporting:**
   - **Inline Comments:**
     Comment your code clearly to explain the computation of z, the entropy gradient, the parameter updates, and the enforcement of the z mapping constraint.
   - **README/Report:**

     Provide a brief document (1–2 pages) that:

     – Describes your approach to integrating the entropy-based gradient framework and enforcing the z constraint,
     – Explains how you implemented the dual-weight structure and the custom update rules,
     – Compares the observed training dynamics and final accuracy with those obtained using classical loss-based updates,
     – Discusses any challenges or insights encountered during implementation.

Provided Classical Model Code

Below is the base code for the classical even/odd classifier. Your solution should extend this code to implement the new entropy-gradient updates with explicit z mapping and the constraint $z_{i+1} - z_i < \delta$.

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Normalize images to the range [0, 1]
x_train = x_train / 255.0
x_test = x_test / 255.0

# Flatten the images to be vectors
```

```
x_train = x_train.reshape(-1, 28 * 28)
x_test = x_test.reshape(-1, 28 * 28)

# Convert the digit labels to even/odd labels:
# Even -> 0, Odd -> 1
y_train_even_odd = np.array([label % 2 for label in y_train])
y_test_even_odd = np.array([label % 2 for label in y_test])

# Build a single-layer perceptron model for binary classification
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1, activation='sigmoid', input_shape=(28 * 28,))  # Single output
])

# Compile the model with a binary crossentropy loss function
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train_even_odd, epochs=5, batch_size=32)

# Evaluate the model to check accuracy on even/odd classification
test_loss, test_acc = model.evaluate(x_test, y_test_even_odd)
print(f'Test accuracy: {test_acc:.4f}')

# Generate random indices to select 10 test images for demonstration
random_indices = np.random.randint(0, x_test.shape[0], size=10)

# Plot the 10 random test images along with true and predicted labels
plt.figure(figsize=(15, 6))
for idx, i in enumerate(random_indices):
    image = x_test[i].reshape(28, 28)  # Reshape the flat image back to 28x28 format
    true_label = y_test_even_odd[i]     # True label: 0 (even) or 1 (odd)

    # Predict the label (returns a probability; threshold at 0.5)
    prediction_prob = model.predict(x_test[i:i+1])
    prediction = 1 if prediction_prob[0][0] >= 0.5 else 0

    # Convert numerical labels to text
    true_text = "Odd" if true_label == 1 else "Even"
    pred_text = "Odd" if prediction == 1 else "Even"

    # Plot image in a subplot
    plt.subplot(2, 5, idx+1)
    plt.imshow(image, cmap='gray')
    plt.title(f"True: {true_text}\nPred: {pred_text}")
    plt.axis('off')

plt.tight_layout()

plt.savefig('test-results')
plt.show()
```

Your Task

- **Develop a Custom Training Loop:**
  - Replace the built-in optimizer with your own loop that:
    - Computes z_k using the modified knowledge mapping,
    - Applies the sigmoid to obtain D_k,
    - Computes the entropy gradient $\frac{\partial H(z)}{\partial z}\Big|_k$,
    - Updates the parameters $w_{1,j}$, $G_{1,j}$, and $b_1^{(0)}$ using the custom update rules,
    - Enforces the constraint $z_{i+1} - z_i < \delta$ at each update step by scaling down updates if necessary.
    - Computes the Entropy $H(z)|_k$ using the provided formula.
- **Monitor Performance:**
  - At the end of each epoch (or after a fixed number of iterations), compute and print the test accuracy.
  - Track the evolution of accuracy across epochs.
- **Visualize Results:**
  - After training, display 10 randomly selected test images with their true and predicted labels.
  - Plot the accuracy curve over training iterations or epochs.

Deliverables

- **Source Code:**
  A complete Python script (or Jupyter Notebook) that implements the new entropy-gradient updates with explicit z mapping and the z constraint.

- **Documentation:**
  A README or short report (1–2 pages) explaining:

  - Your implementation strategy,
  - How you incorporated the z mapping, the dual-weight structure, and the constraint $z_{i+1} - z_i < \delta$,
  - The key differences between classical loss-based updates and the entropy-gradient updates,
  - Observations regarding training dynamics and classification accuracy.

- **Execution Instructions:**
  Provide any necessary instructions for installing dependencies and running your code.

Initialization Guidelines

- Initialize $w_{1,j}$ and $G_{1,j}$ with small random values (e.g., scaled by 0.01).
- Start with a small learning rate η (e.g., 0.001-0.01) and adjust dynamically.
- Set δ within the range of 0.01-0.1 and observe its effect on training stability.