***Implement a Basic Spiking Neural Network (SNN) for Binary Classification***

**Introduction to Spiking Neural Networks (SNNs)**

Spiking Neural Networks (SNNs) are a class of biologically-inspired neural models that process data using discrete-time events called **spikes**. Unlike traditional neural networks, which rely on continuous activation functions, SNNs model neuron behaviour more closely by transmitting information only when membrane potentials reach a specific threshold. This leads to greater **temporal sparsity**, **energy efficiency**, and potential deployment on neuromorphic hardware.

**Generating Spike Trains with snntorch**

The first step in training an SNN is to convert static input data (like MNIST images) into **spike trains**. This is achieved using encoders such as **rate encoding** and **latency encoding**.

* **Rate Encoding**: The pixel intensity determines the likelihood of spiking at each time step.
* **Poisson Process**: A probabilistic method to generate spikes over time using the input intensity as the firing probability.

**Building and Training an SNN on MNIST**

This tutorial demonstrates how to construct and train a **multi-layer SNN** using surrogate gradients.

* **Leaky Integrate-and-Fire (LIF) neurons** are used for spiking behaviour.
* beta=0.9 is the decay factor for membrane potential.
* Uses **temporal processing** over multiple time steps.

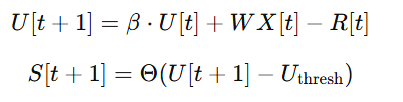
**Surrogate Gradient Training**

Backpropagation is not directly applicable in SNNs due to the non-differentiable spike function. So, **surrogate gradients** are used — approximating the gradient of the spike function for training.

* Loss is computed based on the **number of output spikes** matching the target class.
* Optimizer and training loop use standard PyTorch syntax.

**Mathematical Model of LIF Neuron**

A Leaky Integrate-and-Fire neuron evolves as:



* U: membrane potential
* β: decay factor
* W: input weights
* S[t]: spike output
* Θ: step function

After spiking, a **reset mechanism** ensures the neuron drops back below threshold.

**Outcome**

After training:

* The SNN classifies two digits (e.g., 0 and 1) from MNIST.
* Performance is comparable to traditional models in low-resource setups.
* Real advantage: sparsity, temporal coding, energy efficiency.

***Develop a basic Q-learning agent to understand the fundamentals of reinforcement learning.***

**Introduction to Q-Learning**

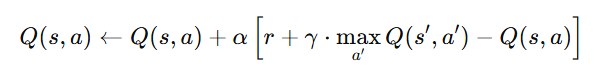
Q-learning is a **model-free reinforcement learning algorithm** that enables an agent to learn the best actions to take in a given environment to maximize long-term rewards. Unlike supervised learning, Q-learning does not need labelled data—it learns by interacting with the environment and observing rewards.

It is **off-policy**, meaning it learns the value of the optimal policy independently of the agent’s actions.

* **Environment**: The world with which the agent interacts.
* **Agent**: Learner/decision maker.
* **State (S)**: Current situation of the agent.
* **Action (A)**: Choices available to the agent.
* **Reward (R)**: Feedback from the environment.
* **Q-value (Q[s][a])**: Expected cumulative reward from state s taking action a.

**Bellman Equation**

The Q-value update rule is:



Where:

* α: Learning rate (how much to update)
* γ: Discount factor (importance of future rewards)
* r: Immediate reward
* s′: Next state
* maxa′Q(s′,a′): Maximum future reward

**Q-Learning Algorithm**

1. Initialize the Q-table with zeros.
2. For each episode:
   * Initialize the state.
   * Repeat for each step:
     + Choose action using an **ε-greedy** policy:
       - With probability ε, select random action (exploration).
       - With probability 1−ε, select action with max Q-value (exploitation).
     + Perform the action; observe reward and next state.
     + Update Q-value using Bellman Equation.
     + Set new state = next state.
   * Repeat until the goal is reached.

***Combine biologically inspired learning rules (STDP) with reinforcement learning techniques (Q-learning)***

**Introduction**

In traditional machine learning, reinforcement learning (RL) and supervised learning often rely on backpropagation. However, **biological neural systems learn without backpropagation**, instead using **local learning rules** like **Spike-Timing Dependent Plasticity (STDP)**.

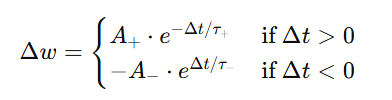
This objective explores how **STDP**, a biologically plausible mechanism of synaptic change, can be integrated with **Q-learning**, to create reinforcement learning agents that align more closely with real neural circuits, particularly in **spiking neural networks (SNNs)**.

**What is STDP?**

**Spike-Timing Dependent Plasticity (STDP)** is a **Hebbian learning rule** that updates synaptic weights based on the precise timing of spikes between presynaptic and postsynaptic neurons.

**STDP Rule:**

* **If pre-synaptic neuron spikes before post-synaptic** → **Weight is increased** (Long-Term Potentiation)
* **If post-synaptic neuron spikes before pre-synaptic** → **Weight is decreased** (Long-Term Depression)



Where:

* Δt=tpost−tpre
* A+, A\_ are learning rates
* τ+, τ-are decay constants

**Combining STDP with Q-Learning**

In RL, learning involves associating **states and actions with rewards**. When combined with STDP:

* **Spiking activity encodes state/action representations**
* **STDP adjusts synaptic weights locally**
* **Global reward signals (like dopamine spikes) modulate the plasticity**

This is often termed **Reward-Modulated STDP (R-STDP)**.

**R-STDP Update Rule**

Δw=R⋅STDP

Where R is the reinforcement (reward or punishment). This integrates local spike-based learning with a global reinforcement signal, approximating **policy gradient updates** in an SNN context.

* SNNs with R-STDP can learn to navigate environments and solve control tasks.
* Modulating STDP with **reward signals** allows **unsupervised learning to become goal-directed**.
* **Eligibility traces** act as short-term memory for credit assignment (which STDP lacks alone).

**Neuromatch**

**Simple spiking model** using the Brian2 simulator:

* Neurons are connected with STDP synapses.
* A **dopamine signal** acts as a reward modulator.
* A spike-triggered eligibility trace stores the temporal correlation between spikes.
* This shows a synapse whose weight changes only if reward is present.
* Modifies weight using Hebbian principle **only when timing AND reward align**.

**Practical Integration with Q-learning**

To apply STDP in Q-learning:

1. **State and action representations** are encoded using spiking neurons.
2. A **global Q-learning reward signal** (based on state transitions) modulates the STDP.
3. An **SNN replaces or augments the Q-table**, learning to represent Q-values via synaptic strengths.

This method is useful for:

* Energy-efficient neuromorphic control
* Learning with **limited or no supervision**
* Applications in **robotics**, **neuromorphic chips**, and **bio-inspired AI**