Neuromorphic Computing

3. Learning in Neuromorphic Systems (Part I & II)

Ben Abdallah Abderazek, Khanh N. Dang
E-mail: {benab, khanh}@u-aizu.ac.jp

This lecture is based on the book ''Neuromorphic Computing Principles and Organization,'' Publisher: Springer; 2022 edition, ISBN-10 : 3030925242, ISBN-13 : 978-3030925246, by Abderazek Ben Abdallah, Khanh N. Dang

Lecture Contents – Part I

1. Learning Methods
2. Conversion from ANN to SNN
   - Converted SNNs
   - Challenges of ANN Conversion
3. Supervised Learning
   - Tempotron
   - ReSuMe.
   - SpikeProp Algorithm
   - Approximate Derivative Method (ADM)
4. Unsupervised Learning
   - Pair-Based STDP Learning Rule
   - Triplet STDP Learning Rule
   - Reward-Modulated STDP Learning
   - Other Variants of STDP Learning Rule
5. Summary

1. Learning Methods

- The learning process aims to **minimize** a defined **objective function** (loss function) to find the combination of the **parameters** that outputs the correct inference results

- The most common way to optimize the loss function is by using **gradient-descent**-based optimization techniques, although other classical optimization methods can be used such as genetic algorithms

- In gradient-descent-based optimization the goal is to find out the **gradients** of the cost/loss function for each learning parameter

- For SNN, there are various training/learning algorithms, such as **supervised backpropagation through time**, **unsupervised STDP learning**, and **ANN to SNN conversion.**

2. Conversion from ANN to SNN:

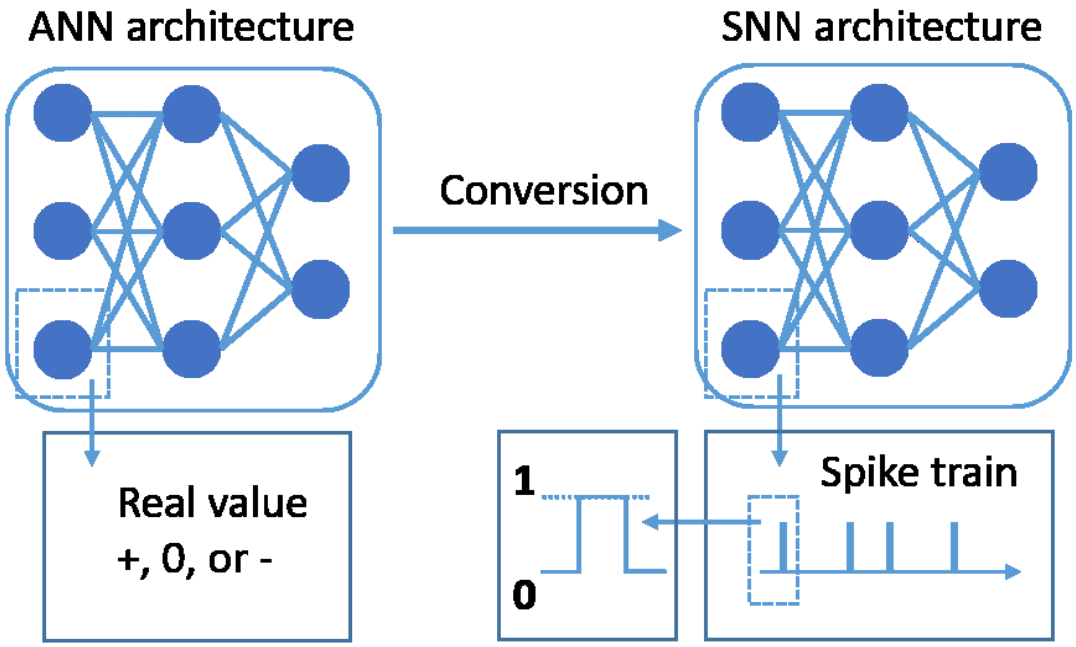


Fig. 3.1: Conversion from ANNs to SNNs.

- The goal is to leverage the state-of-the-art ANN training algorithms in order to reach a competitive performance using the SNN
- There are many challenges in converting ANN to SNN concerning different aspects of the neural network such as the parameters, activation functions, and layers.

2. Conversion from ANN to SNN

- To cope with these challenges, different changes and ideas have been made to meet the requirements of SNN:

  - Use of the abs() function and rectified linear unit (**ReLU**) to avoid **negative** output when for **CNN conversion**

  - The Biases in each layer are set to zero

  - **Max-pooling** is substituted with **linear sub-sampling** which is easily converted to a spike domain

  - **weight normalization** method d to achieve a **near-lossless accuracy** caused by the replacement of ReLUs by IF neurons

2. Conversion from ANN to SNN:

Converted SNNs

- Spiking Deep Belief Networks (DBNs) in which the **frame-driven** system neurons are mapped into an **event-driven** representation

- Another approach to convert CNN to SNN architecture that is suitable for mapping to spike-based neuromorphic hardware is by tackling **negative activations** and **biases** and **non-linearity due to max pooling**

- Training with **noise on output neurons** to have a **robust** model against **spiking variability**

- A modified **soft LIF** function allowing more neurons types to be utilized

2. Conversion from ANN to SNN:

Converted SNNs

- The **adaptive** spiking neural network has a dynamically adjusted threshold and needs fewer spikes to encode information

- The proposed **spike-based learning** rule for **rate-coded** deep SNNs is hardware-friendly due to the requirement of **less computation** and **memory**

- The application on **temporal coding** schemes to converted SNNs reduce significantly the **spike redundancy** and **memory cost**

2. Conversion from ANN to SNN:

Converted SNNs

- A successful implementation of the conversion on **Inception-V3** with 42 layers (7 convolution layers) demonstrates a 74.60% accuracy on the ImageNet dataset

- A Spiking CNN with four convolution levels, achieved a 90.85% accuracy on the CIFAR-10 dataset

- Two deep spiking neuron networks based on **VGG-16** and **Residual network** architecture

- The Residual Membrane Potential (RMP) spiking neuron, which targets the **spike rate vanishing** issue in SNNs caused by the **hard reset** spiking neuron model

2. Conversion from ANN to SNN:

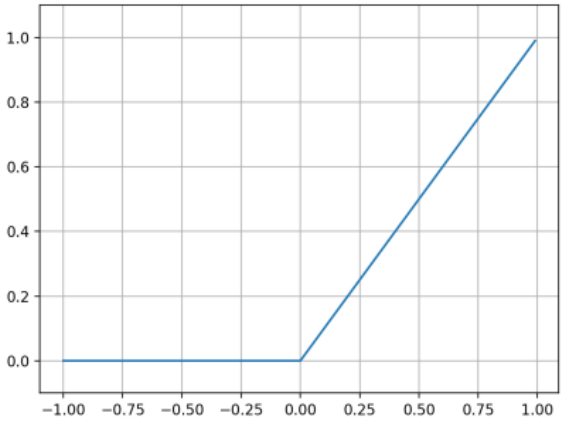
Challenges of ANN Conversion

- The weights and biases and some activation functions' output can have positive or negative values
- However the firing rate in SNN should be positive, therefor the designer needs to tackle negative values during conversion
- A possible solution is to treat positive values as excitatory synaptic input while producing inhibitory synaptic inputs for negative signals
- For this two spiking neurons are needed to represent each input value which leads to a much more complicated architecture

2. Conversion from ANN to SNN:

Challenges of ANN Conversion

- The Rectified linear-unit (**ReLU**) activation function is an efficient way to tackle this problem because it always **maps negative activation** to **zero.**



$$ReLU(z) = max(0, z)$$

ReLU: Rectified Linear Unit

- **Avoiding negative** inputs help also in **faster convergence** compared with equivalent networks with tanh units
- **All biases** are set to **zero** to avoid negatives and reduce the inconvenience.
- The max-pooling operation **reduces** the number of parameters in network, however it comes with **information loss** and **extra complexity**

2. Conversion from ANN to SNN:

Challenges of ANN Conversion

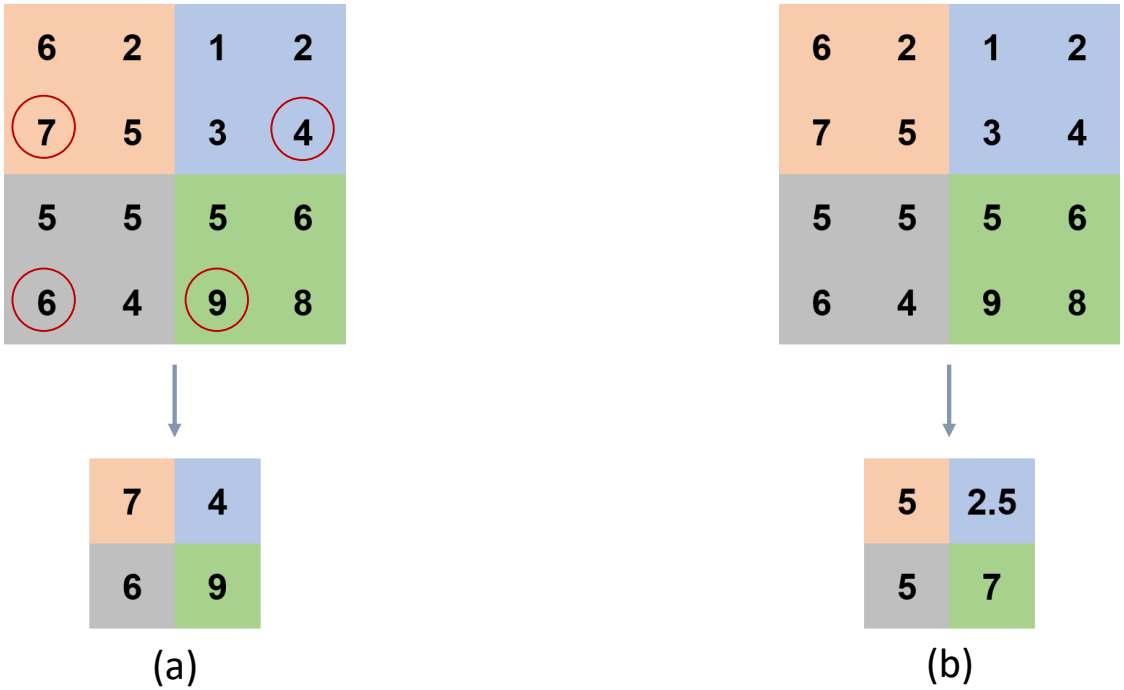


Fig. 3.2: Max-pooling and average-pooling operation (a) Max-pooling. (b) Average-pooling

- Max-pooling is widely used in ANN and especially CNN. Does a max operation on the inputs. With that, it extracts only the significant features from the provided data
- Average pooling, on the other hand, presents a solution to represent arriving spikes in a better way for SNN by taking the average of the incoming input and eliminating winner-takes-all

2. Conversion from ANN to SNN:

Challenges of ANN Conversion

- Max-pooling does not reflect the **actual maximum firing rate**, therefore the average-pooling operation is a better option that enables a **linear function** to be implemented in SNNs

- Also the temporal "time-to-first-spike" encoding is used to select the first neuron that **responds** as this neuron is considered to be the **most robust** response to the stimulus

- However the **temporal coding** scheme is **not ideal** for the ANN-SNN **conversion** process.

- The **gating function** for spiking max-pooling allows only the spikes from the neuron with the **highest firing rate** by **estimating** the presynaptic firing rates

2. Conversion from ANN to SNN:

Challenges of ANN Conversion

- **Approximation errors** might occur, in **time-stepped** simulations of SNNs, due to the constraints that the firing rate is mapped to the range of $[0, r_{max}]$

- This implies that receiving a **perfect representation** of activation from ANNs to SNNs is **non-trivial**

- A relevant **high threshold** can hardly be exceeded, leading to an **underestimation** of the actual firing rate. On the contrary, **over-activation** spike trains or high input weights would give rise to **high** firing rates

- Rescaling the weights using a **model-based** or **data-based** normalization approach can deal with this issue.

2. Conversion from ANN to SNN:

Challenges of ANN Conversion

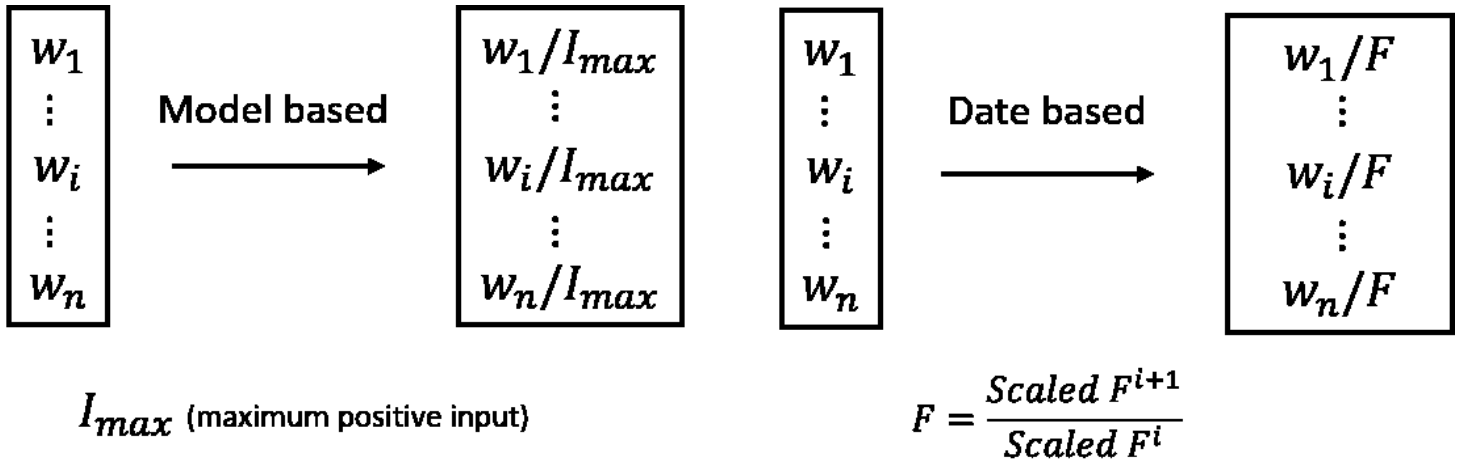


Fig. 3.3: Weight normalization technique.

- The use of the weight normalization method to achieve high accuracy and reduced latency.
- Model-based normalization, requires only the information of the network weights
- Data-based normalization approach scales the weights according to the actual activation of the network in response to data.

2. Conversion from ANN to SNN:

Challenges of ANN Conversion

- Taking into account the actual operation of the SNN during the conversion process **decreases** the **temporal delay** of the neuron and ensures an appropriate firing **threshold**

- In order to detect and discard **outlier activations**, preserving the encoded information of biases jointly scaled with input weights and a **max-norm mechanism** can be used

- Also with a **strong normalization** further combined with **batch-normalization** the whole process achieves a more significant **speedup**

2. Conversion from ANN to SNN:

Challenges of ANN Conversion

- The previously seen solutions focused on the balance of firing rates, spiking threshold, and input weights
- Efficient algorithms that improve computation efficiency should also be adopted
- Lower-compute spiking neurons with fewer spikes can be realized using sparse coding and L2-norm as a cost function. Therefore the overall firing rates are reduced.
- Using **dropout** or trained Stacked Auto-Encoder with a **zero-masking** filter will accelerate the classification task because fewer input spikes will be needed for quick output

2. Conversion from ANN to SNN:

Challenges of ANN Conversion

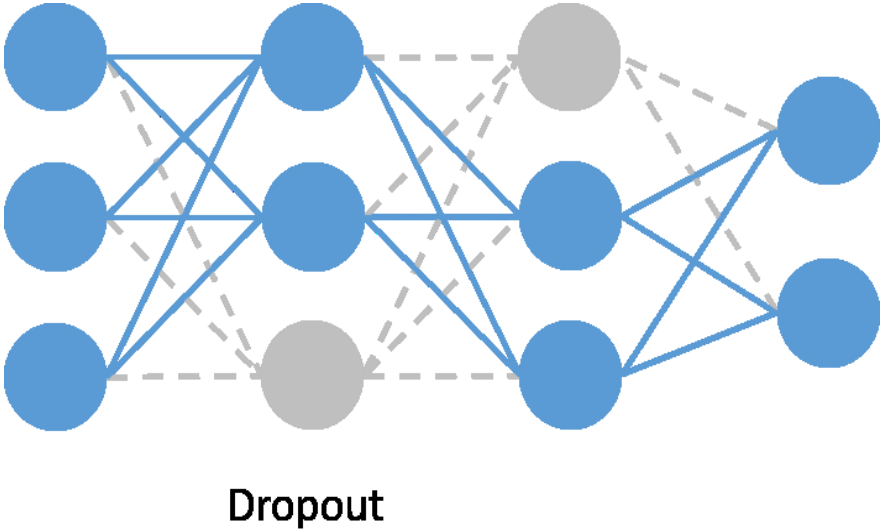


Fig. 3.4: Dropout Method during learning.

- Dropout method is a very known method in training ANN. It consists of dropping random neurons during the training.
- This method helps the model to be more robust and have good accuracy even if some data is missing.

2. Conversion from ANN to SNN:

Challenges of ANN Conversion

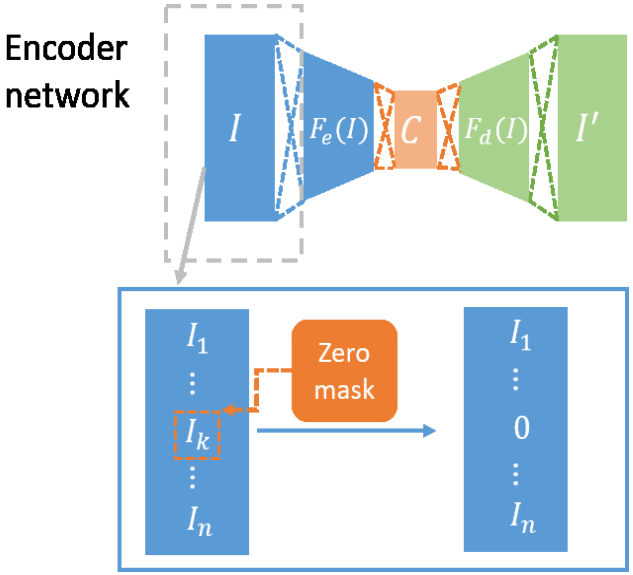


Fig. 3.5: Stacked Auto-Encoder are one of the self-supervised learning methods

- Its purpose is to compress the data by creating an encoder that reduces the dimensionality of the input data and a decoder that tries to reproduce the input data
- A zero mask can be applied to a random patch of the input data while training leading to faster training and a robust model

2. Conversion from ANN to SNN:

Challenges of ANN Conversion

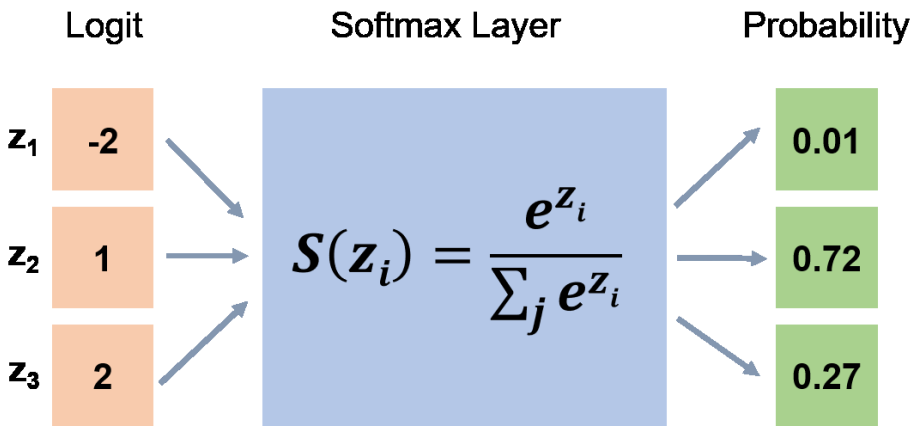


Fig. 3.6: Softmax function

- The softmax function in the output layer is used to normalize the input values into a valid probability distribution that sums to one.

- Without the Softmax layer, pure negative inputs arriving at the final layers will not produce any spike

2. Conversion from ANN to SNN:

Challenges of ANN Conversion

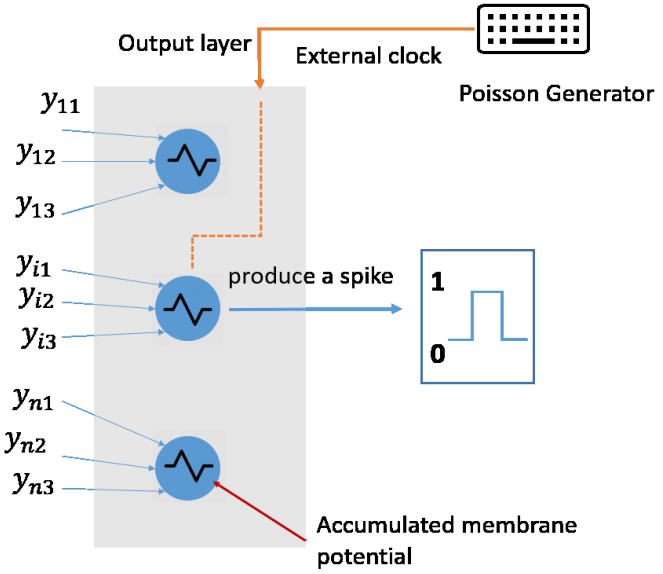
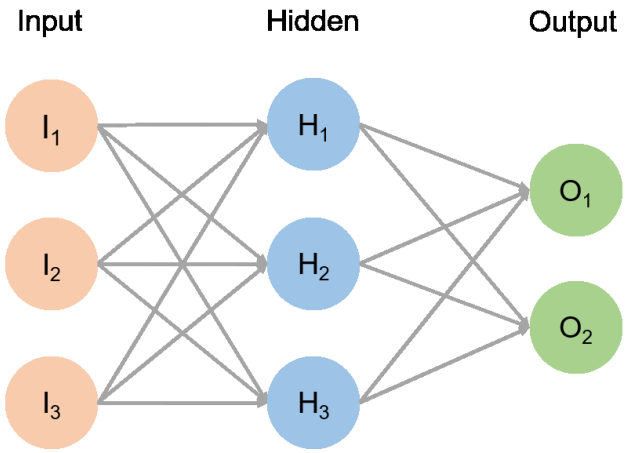


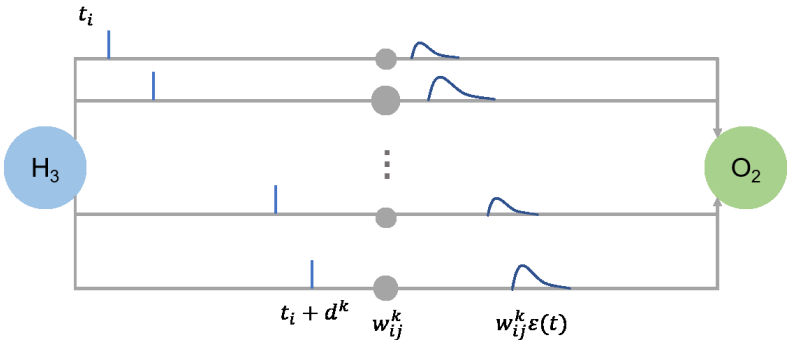
Fig. 3.7: stochastic winner-take-all

- One version of a spiking softmax layer is a stochastic winner-take-all (WTA) mechanism with an external Poisson generator
- the winning neuron is selected according to its membrane potential, and the WTA-circuit allows it to fire at that time step
- classification can directly be inferred based on the computed rate parameters at the softmax layer given the membrane potentials

3. Supervised Learning



(a) Feedforward spiking neural network



(b) Connection consisting of multiple delayed synaptic terminals

Fig. 3.8: Network architecture and connectivity of a spiking neural network.

- The feed-forward spiking neural network contains connections of spiking neurons between layers with multiple delayed synaptic terminals
- Each pre-synaptic terminal corresponds to a sub-connection associated with different decay and synaptic efficacy
- When the sum combination of the internal state variable crosses the threshold θ, a spike is produced by the output neuron

3. Supervised Learning

- The spike-response function ε(t), is used to describe a standard post-synaptic potential (PSP)

$$\varepsilon(t) = \frac{t}{\tau} e^{1 - \frac{t}{\tau}}$$

- Where τ is the membrane decay time constant of a neuron
- the post-synaptic input x_j of neuron j receiving input from neuron i can then be described as the weighted sum of all the pre-synaptic input:

$$x_j(t) = \sum_i \sum_k w_{ij}^k \varepsilon_{ij}^k (t - ti - d_{ij}^k)$$

- Where i belongs to all the presynaptic neurons of neuron j, and t_i is the arrival time of the spike from i

3. Supervised Learning:

Tempotron

- A model of supervised learning for classification tasks, which uses a LIF neuron driven by synaptic afferents

  - Where w_i is the weight of neuron i and one of the postsynaptic neurons

- A postsynaptic potential is induced by an input spike at time t_i

- τ and τ_s denote the decay time constants of membrane integration and synaptic currents and are used to describe the form of the postsynaptic potentials

- The maximum value of PSP is normalized to 1 with a factor V0

3. Supervised Learning:

Tempotron

- The Tempotron learning rule, each synaptic efficacy w_i follows the gradient descent updating mechanism

$$\Delta w_i = \lambda \sum_{t_i < t_{max}} (e^{-\frac{t_{max} - t_i}{\tau}} - e^{-\frac{t_{max} - t_i}{\tau}})$$

  - t_{max} is the time when the maximal value of the postsynaptic potential is reached without the neuron firing

- The synaptic weight of should decrease if an erroneous output spike occurs in order to decrease it's contribution

- On the other hand, the synaptic weight should increase if the neuron doesn't fire when it should have produced a spike

3. Supervised Learning:

ReSuMe

- A learning rule that is Tempotron-like

- The synaptic efficacy between two pre- and postsynaptic neurons does not only depend on the correlated pair

3. Supervised Learning:

SpikeProp Algorithm

- SpikeProp is an error-back propagating learning algorithm.
- It aims to minimize is the mean squared error defined on the spike times of the output neurons and the desired spike times
- In ANN we have continuous values therefore back prop computes gradients by propagating continuous error signals backward.
- Meanwhile, SNN operates on spike events, therefore the backpropagation accounts for the spike timing of the neurons
- Since the spike timing carries information, the SpikeProp algorithm aims to adjust the weights and the spike timing to get the desired output

3. Supervised Learning:

SpikeProp Algorithm

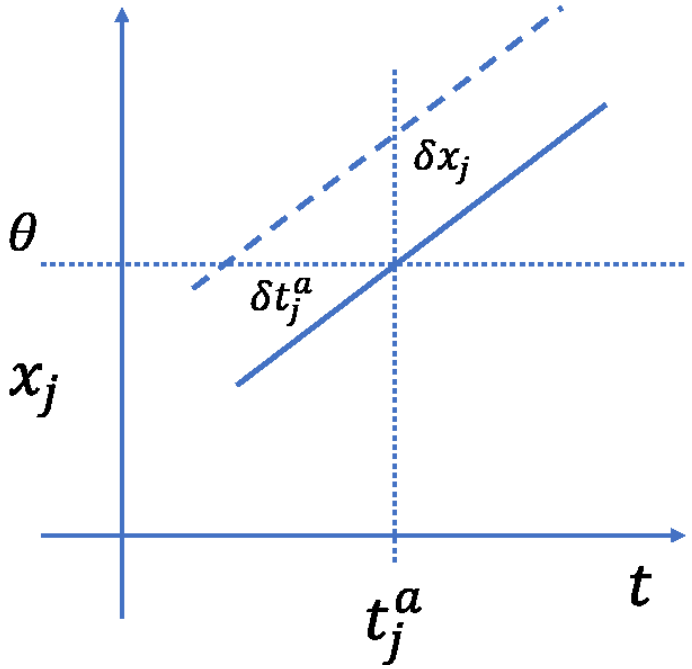


Fig. 3.9: Relationship between δx_j and δt_j for a small region around $t = ta$

- t_j^a is the actual spike timing of neuron *j*, x_j is the threshold post-synaptic input.

- For a small enough region around $t = t_j^a$, the function x_j is approximated by a linear function of *t*

3. Supervised Learning:

SpikeProp Algorithm

- Other methods were proposed to improve the SpikeProp algorithm.
- Adding a momentum term to improve convergence and tackle possible occurrence of local minimum
- More generic architecture, which contains recurrent connections which allow to handle of multiple spikes per neuron at a time
- Learning-rate adjustment algorithm, called resilient propagation (RProp) accelerates the training process. it performs the weight update based on the sign of the gradient
- SpikePropAD and SpikePropR are learning rate adaptation methods and robust versions to tackle the issue of weight convergence

3. Supervised Learning:

SpikeProp Algorithm

- Other methods were proposed to improve the SpikeProp algorithm.
- QuickProp makes use of the second derivative of the error with respect to one weight, assuming it is independent of the others. allowing for more rapid convergence during training.
- In QuickProp, the current weight change depends on the previous weight change, and the error minimum can be slowly reached except for the large step size during the training
- In the original SpikeProp, each connection contains a fixed number of delayed synaptic terminals, and only the weights are trained.
- Allowing delays to be trained during learning, thus reducing the number of synaptic terminals and weights

3. Supervised Learning:

Approximate Derivative Method (ADM)

Input in Hidden layer: $net_k^H(t) = \sum_{j=1}^{J}(w_{jk}^I x_j^I(t))$

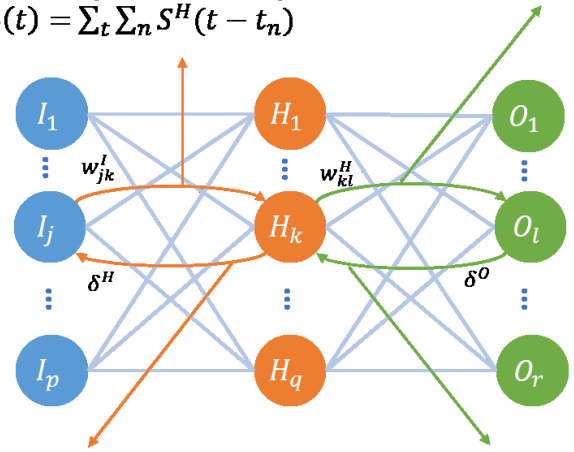
$x_j^I(t) = \sum_t \sum_n S_j^I(t - t_n)$

Activation: $a_{LIF}^H(t) = \sum_t \sum_n S^H(t - t_n)$

Input in Output layer: $net_l^O(t) = \sum_{k=1}^{H}(w_{kl}^H y_k^H(t))$

$y_k^H(t) = \sum_t \sum_n S_k^H(t - t_n)$

Activation: $output = \frac{1}{T} V_{mem}^O(t)$



Error propagation in hidden layer:

$\delta^H = ((w^H)^{Tr} * \delta^O) \cdot a'_{LIF}(net^H)$

Error propagation in output layer:

$\delta^O = (desired\ value\ - actual\ value)/T$

Fig. 3.10: Approximate derivative method

- LIF neurons in hidden layers generate post-spikes if the membrane potential exceeds a threshold and reset the membrane potential
- LIF neurons in the final layer, do not generate any spike, but rather accumulate the weighted sum of pre-spikes till the last time step to quantify the final outputs
- The final errors are propagated backward through the hidden layers and synaptic weights are modified in a direction to reduce the final errors

Ref: Frontiers | Enabling Spike-Based Backpropagation for Training Deep Neural Network Architectures (frontiersin.org)

Lecture Contents – Part II

1. Learning Methods
2. Conversion from ANN to SNN
   - Converted SNNs
   - Challenges of ANN Conversion
3. Supervised Learning
   - Tempotron
   - ReSuMe.
   - SpikeProp Algorithm
   - Approximate Derivative Method (ADM)
4. Unsupervised Learning
   - Pair-Based STDP Learning Rule
   - Triplet STDP Learning Rule
   - Reward-Modulated STDP Learning
   - Other Variants of STDP Learning Rule
5. Summary

4. Unsupervised Learning

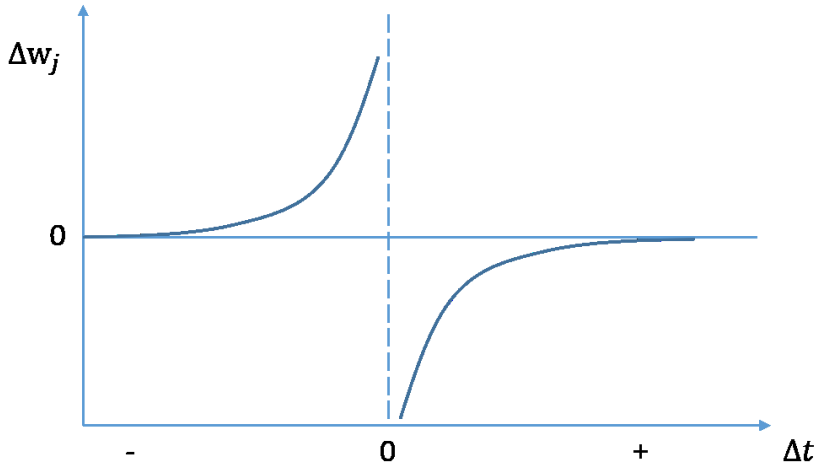


Fig. 3.11: Basic STDP learning rule

- In this figure we can see the change of synaptic weight relating to the temporal difference between a pair of presynaptic and postsynaptic spikes.
- According to the Hebbian rule, synapses increase their efficacy if they persistently participate in the firing of a postsynaptic neuron
- If a pre-synaptic spike arrives before a neuron fires, the weight of that synapse is strengthened, which allows it to contribute more.
- If a pre-synaptic spike arrives after a neuron fires, the weight of that synapse is weakened, reducing its contribution.
- STDP is a Hebbian rule-based for adjusting the strength of connections between neurons.

4. Unsupervised Learning :

Pair-Based STDP Learning Rule

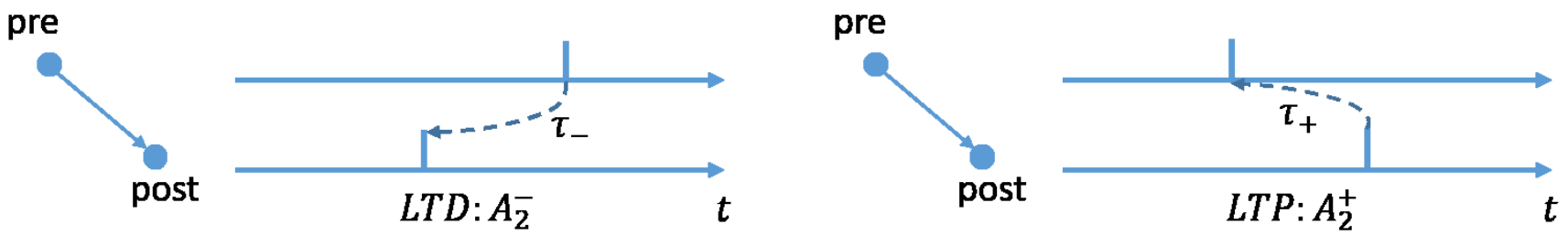


Fig. 3.12: Pair-based STDP learning rule

- The pair-based STDP rule considers the time difference between a pair of pre-synaptic and post-synaptic spikes and updates the potentiation and depression potentials accordingly.
- The weight changes are then based on the values of these potentials at the respective spike times.
- The time constants τ+ and τ- determine the time scales for the decay of the potentiation and depression potentials, respectively.
- A_2^- and A_2^+ represent the amplitudes of weight change controlling long-term depression (LTD) and long-term potentiation (LTP), respectively

4. Unsupervised Learning :

triplet-Based STDP Learning Rule

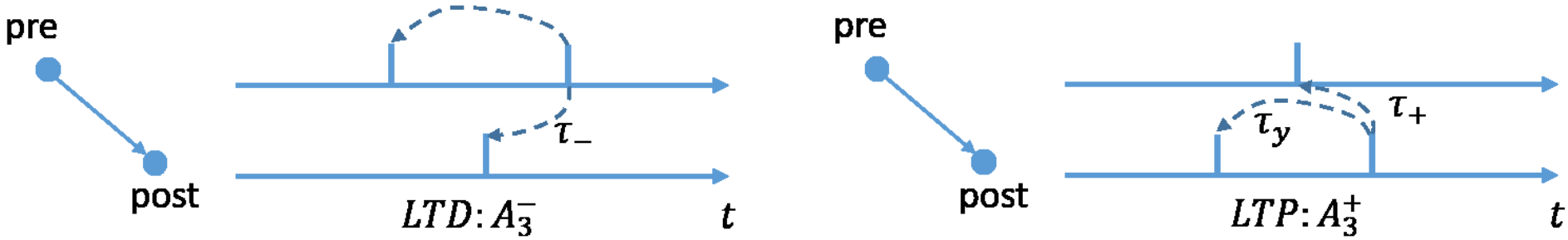


Fig. 3.13: Triplet-based STDP Learning Rule

- In the pair-based STDP it is not possible to capture that with the increase of frequency, there might rise an additional impact from the presynaptic spikes of the following couple on the post-synaptic spike of the previous pair.
- triplet-based STDP can capture if either a pre-post-pre and the post-pre-post scheme is formed

4. Unsupervised Learning :

triplet-Based STDP Learning Rule

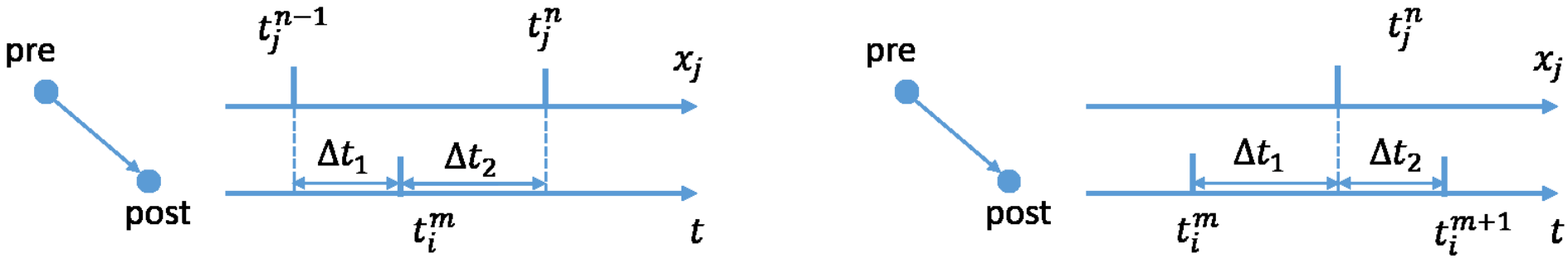


Fig. 3.14: Spike pairing scheme. All-to-All interaction and Nearest-Neighbor interaction scheme

- Triplet-based learning rule is associated with All-to-All interactions
- In Nearest-spike interactions, only the nearest spikes are considered and the weights updates will be performed accordingly
- The Quadruplet protocol by further taking the interaction between a post-pre pair and a pre-post pair into consideration

4. Unsupervised Learning :

Reward-Modulated STDP Learning

- The R-STDP or Reward-Modulated STDP Learning rule is another variant of the STDP learning rule.

- It adds sparse external reinforcement signals that can modulate an SNN in the same way that our brain injects dopamine to Dopaminergic neurons

- This reward mechanism depends on the modulation of dopamine during synaptic adaptation by STDP

- The equation shows how the weight changes are determined for R-STDP: $\dot{w} = e \times (d - b)$ where:
  - $\dot{w}$ is the synaptic weight change
  - e the eligibility trace
  - d the reward function
  - b the baseline

4. Unsupervised Learning :

Other Variants of STDP Learning Rule

- Another variant for STDP is to use one dynamic variable instead of measuring the time difference in the pair-based model

- This variant considers only the voltage dependence of the postsynaptic neuron of membrane with an integrate-and-fire neuron model

- In this model, the updating of the synaptic weight depends on the membrane voltage threshold and a function of postsynaptic spiking activity.

5. Summary:

- The biological brain is a learning machine capable of making complex computations while using small resources and being power efficient.
- The sparse communication among many spiking neurons is the main property that enhances this power efficiency.
- By incorporating learning methods and rules seen in this chapter, SNNs gained popularity and attention in research fields.
- The learning phase, as presented in this chapter, aims to minimize a particular loss function by acquiring the parameters of the network to output the correct results.
- Meanwhile, during inference, the SNN outputs a result based on the input and obtained parameters from the training.