Bachelor thesis

Artificial Intelligence

Radboud University

Semantic role labeling with spiking neural networks using surrogate gradient learning

Author:
Vinzenz Richard Ulrich
s1051253

Supervisor:
Dr. H. Fitz
`Donders Centre for Cognition`
`h.fitz@donders.ru.nl`

Second reader:
Dr R.C. Farinha Duarte
`Donders Centre for Cognition`
`renato.duarte@donders.ru.nl`

July 7th, 2023

Contents

Introduction

Humans are unique in their ability to understand and process language and come up with it to begin with. Human language abilities have enabled sophisticated communication, without which the advent of human civilization may have been impossible. We use language every day, be it while reading or writing text, while speaking with others, or while thinking by ourselves. The way humans, and thus human brains, are able to understand and process language so deeply is not fully understood yet. This makes language processing in the brain an interesting area of inquiry.

Having initially started out with a different topic, namely stock price prediction, this thesis is about the human brain system for language. I determined that it is not possible to build models that perform well for the prediction of stock prices with the resources at my disposal.

Improving the processing of natural language in computers can help with understanding the processing of natural language in the most advanced neural network known to us, the brain. It may also unlock new ways to interact with devices, making new devices possible. Conventional artificial neural networks (ANNs) are inspired by how we think the brain works, as they consist of units (neurons) and connections between units (synapses) that have varying strength (weights). However, this is merely an idealized model of the brain, and neurons are much more complicated than the simple units of an ANN. There are likely memory mechanisms in neurons themselves, which work on varying time scales, to adapt to input and optimize 'performance' (this does not refer to the explicit memory that humans have, but neuronal mechanisms).

We think that the brain works using a spiking behavior between neurons, where once a certain voltage threshold of the input to the neuron is surpassed, the neuron fires a signal through the synapse to the 'following' neurons (Hodgkin & Huxley, 1952). Based on the image of firing neurons in the brain, so-called third-generation neurons, spiking neurons, were adapted for artificial neural networks (Ghosh-Dastidar & Adeli, 2009). They are also called leaky-integrate and fire neurons. Using these spiking neurons in artificial neural networks yields spiking neural networks, which are networks of these neurons. Given the nature of spiking neurons, spiking neural networks simulate real-time neuronal dynamics. They integrate the input over time (new input, analogous to stimuli in the brain, is presented over time), and their output changes over time.

Spiking neural networks can be purely feedforward or have additional recurrent connections which connect neurons with themselves, to integrate output of previous time steps as input, or to other neurons that are in the same or a previous layer in the network structure. Research has suggested that recurrence in spiking neural networks has a significant impact on the spiking behavior of neurons (Pernice et al., 2012). However, it is not clear whether recurrent connections between neurons are what is needed to be able to process high-level temporal dependencies, as memory within the neuron has also shown promising results (Ponghiran & Roy, 2022). Research suggests that delays in the synapses of spiking neurons are sufficient to emulate the additional temporal component introduced by recurrent connectivity, which is inherently temporal (Diehl et al., 2016). This suggests that recurrence may not be needed in spiking neural networks to add temporal dynamics to better be able to model sequential data.

In the past, it was not feasible to train an artificial neural network employing spiking behavior, a spiking neural network, due to the fact that the derivative of the Heaviside step activation function used in spiking neural networks is zero everywhere except in the place of the spike (input equals zero), where it is infinity. This means that backpropagation, the algorithm commonly used to train artificial neural networks in a supervised manner, could not be employed to train the model effectively. In backpropagation, the error between the predicted output and the actual, intended, output is calculated. The predicted output depends on how the model parameters interact with the input. Using the error value, we determine how each parameter in the network should be changed to make the network output more similar to the actual output. Replacing the Heaviside step activation function with a surrogate gradient function enables effective backpropagation (see Methods).

I have applied spiking neural networks using surrogate gradient learning on semantic role labeling. The words that a sentence consists of have semantic roles, which relate to the meaning of the words in the sentence. For example, in the sentence "the dog chases the cat", "the dog" is the agent, "chases" is the action, and "the cat" is the goal. Semantic roles are not a fancy way of expressing parts of speech (like verb, noun, or adjective), but they are a high-level abstraction, helping to comprehend language on the deep level that humans do. Without comprehending semantic roles, it is not possible to understand natural language. For example, in the English sentence "The door was closed by the cook", "the door" is acted on, while "the cook" is the actor. It is only possible to understand that the sentence expresses that the cook did something, and not the door, by determining the semantic roles. In the sentence "The cook closed the door", the actor (the cook) is mentioned first, but the meaning of the sentence is the same. Semantic roles depend on the interaction of words with each other in a sentence. To be able to determine semantic roles, it is necessary to have a mechanism to remember earlier words (stimuli) (Kittilä & Zúñiga, 2014).

Semantic role labeling is the labeling of each word in a sentence with its correct semantic role. Essentially, it consists of mapping strings to meaning. You expose the network to a sentence sequence, and the result is a sequence of predicted labels. Words have varying lengths in the input, as is the case in spoken language. It is not possible to simply assign an absolute semantic role to each word in a language, since the semantic role of a word in a sentence depends on the context and sentence structure, not the individual word that is to be classified. This makes semantic role labeling nontrivial to achieve using computer programs. The most likely semantic role of a word at the beginning of a sentence might also change as new words are presented (Fitz et al., 2020).

Understanding semantic roles is crucial in language acquisition and comprehension of the meaning of language. Semantic roles are central in using language (Alishahi & Stevenson, 2010). Research suggests that children learn semantic roles throughout their development, suggesting that learning semantic roles is needed for language acquisition (Tomasello, 2000). The process through which humans are able to learn to determine semantic roles and how the brain processes language to map language stimuli to meaning is not yet fully understood.

Semantic role labeling can be performed by a human expert, which means that it is possible to do. Since spiking neural networks more closely resemble how we think the brain works (Tavanaei et al., 2019) and we know that humans can perform semantic role labeling, I applied spiking neural networks to semantic role labeling. To understand how humans are able to map words to semantic roles, sentences to meaning, and understand language, it is expedient to investigate how networks of spiking neurons are able to determine semantic roles, taking into account the context of the sentence to do so. This involves investigating how networks of spiking neurons use memory to take into account the context of a sentence when determining the semantic role label for a word (Fitz et al., 2020).

Performing semantic role labeling using spiking neural networks can yield information to understand language processing in the brain, as semantic role labeling is central to it. It is not fully understood what dynamics in the brain occur to enable semantic understanding of language, and how neurons take into account (memorize) the context of the sentence to determine semantic roles. There is recurrent connectivity in the brain, and recurrence is necessary to accurately model the visual pathway of the brain in neural networks (Kietzmann et al., 2019). It is unknown if recurrence is necessary to accurately model language processing in the brain.

Due to the nature of spiking neural networks, working in time, online machine learning is utilized for training. Words are labeled as they are presented to the network, not when all words in a sentence have been presented. Being able to apply spiking neural networks to this task may also help improve the performance of neural networks in processing language, as semantic role labeling enables a deeper understanding of language, which may help in applications like chatbots, virtual assistants, language transcription, and others. It may also help increase neural network performance in other domains if the given task has similarities to semantic role labeling (i.e. e.g. online task, inherent temporal dependencies, etc.). To that end, not only is the dichotomy between feedforward and recurrent spiking neural networks interesting, there also exist different ways of encoding the input. Research suggests that a temporal spike train encoding scheme is well suited for time series (sequential) data (A Spiking Neural Network Based on Temporal Encoding for Electricity Price Time Series Forecasting in Deregulated Markets, 2010), but a temporal encoding incorporating a spatial component may yield better results (see Methods). A temporal input encoding with a spatial component refers to injecting spike trains into a subset of the input neurons, with the neurons from the input layer all projecting their signals to the hidden layer. This is in contrast to a 'spatial encoding', where the temporal data is projected into a subset of the input neurons, and those input neurons project into a subset of hidden neurons (Uhlmann, n.d.).

In order to better understand how the brain processes natural language, and to make neural networks perform better in natural language processing and other machine learning domains, I have investigated whether spiking neural networks using a surrogate gradient function can be used to perform semantic role labeling of English sentences. My research questions were "Performs a spiking neural network trained with a surrogate gradient function with recurrent connectivity better than one without recurrent connectivity on semantic role labeling of English sentences?" and "Performs a spiking neural network trained using a surrogate gradient function better when using a temporal input encoding or a temporal input encoding with a spatial component on semantic role labeling of English sentences?". My hypotheses were "A spiking neural network with recurrent connectivity performs better than a feedforward spiking neural network," and "A spiking neural network using a temporal input encoding with a spatial component performs better than a spiking neural network using a temporal input encoding without a spatial component". A concern that arises with a temporal encoding with a spatial component is that the input that drives network activity is undersized if the frequency of spikes is not adjusted. I hypothesize that during training, the models learn to adapt to the lesser input intensity that projecting into only a subset of the input neurons entails.

Methods

2.1 Spiking neurons

The spiking behavior of neurons in the brain is in contrast to how ANNs work, where the output of a unit (neuron) is a real number that may be, depending on the choice of activation function, between zero and one (sigmoid function), between one and minus one (TanH, or hyperbolic tangent function), greater than or equal to zero (ReLu or rectified linear unit function), or other. Spiking neurons are units in an artificial neural network that calculate activation based on integrating the input and, once the integral of the input reaches a threshold, firing. The input is accumulated over time, decays over time, and if a threshold is surpassed, a signal is sent to the neurons that are connected to the given neuron.

The threshold in artificial spiking neurons is modelled using the Heaviside step function as the activation function (Deng, 2021). This step function takes real-valued input, which is analogous to the voltage in a neuron in the brain. If the input is less than zero, the function returns zero; if the input is greater than zero, it returns one. This is equivalent to a neuron firing since only when the threshold (zero) is surpassed, the neuron sends a signal. The input to the neuron is integrated over time and multiplied with a decay (the 'leaky' part in 'leaky-integrate and fire'). If, at any point in time, it surpasses the threshold, the neuron fires and the voltage in the neuron is reset (Abbott, 1999). Because the decay is applied at every time step, if no new input arrives at the given neuron, the internal activation (voltage) of the neuron will decrease over time. The internal activation of a neuron is the membrane potential (measured in voltage), and spikes are also called action potentials.

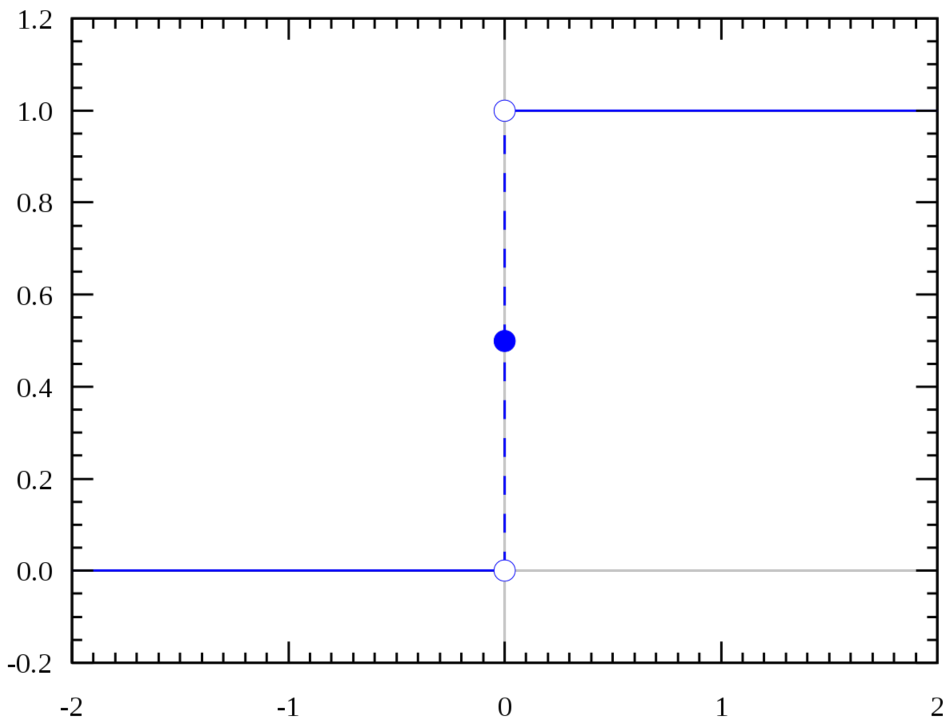


Figure 2.1: The Heaviside step activation function

These are the formulas for the leaky-integrate and fire neuron model used in the spiking neural networks I will be working with (Fzenke, n.d.):

The input to a neuron i in layer l at time step t is calculated as follows:
$I_i^{(l)}(t) = \alpha I_i^{(l)}(t-1) + \sum_j W_{ij} S_j^{(l-1)}(t-1) + \sum_j V_{ij} S_j^{(l)}(t-1)$
Where:

- $I_i^{(l)}(t)$ is the input to neuron i in layer l in time step t,
- α is the decay factor for the input at the previous time step,
- $I_i^{(l)}(t-1)$ is the input at the previous time step,
- $\sum_j W_{ij} S_j^{(l-1)}(t-1)$ is the sum of the weights times the spikes from the previous layer over all the weight-spike pairs for neuron i for the previous time step, and
- $\sum_j V_{ij} S_j^{(l)}(t-1)$ is the sum of the weights of the recurrent connections times the spikes from the same layer over all the weight-spike pairs for neuron i for the previous time step.

In essence, the function combines for a neuron the input of the previous time step (decayed by factor α), the weighted sum of spikes incoming from the preceding layer, and the weighted sum of the spikes incoming from the same layer, *if* there is recurrent connectivity in the network.

The internal activation, the voltage, of a neuron i in layer l in time step t is calculated as follows:
$U_i^{(l)}(t) = \beta U_i^{(l)}(t-1) + I_i^{(l)}(t-1) - S_i^{(l)}(t-1)$
Where:

- $U_i^{(l)}(t)$ is the voltage of neuron i in layer l at time step t,
- β is the decay factor for the voltage at the previous time step,
- $U_i^{(l)}(t-1)$ is the voltage at the previous time step,
- $I_i^{(l)}(t-1)$ is the input from the previous time step, and
- $S_i^{(l)}(t-1)$ is the reset which resets the voltage of the neuron if it has spiked.

Essentially, the function combines for a neuron the voltage of the previous time step (decayed by factor β), the input calculated for the previous time step, and resets the voltage if it spiked the previous time step.

The neuron voltage is run through the Heaviside activation function at each time step to check if a spike was produced. The Heaviside step function is mathematically defined as

$$\Theta(x) := \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$$

Where:

- $\Theta(x)$ is the result of the Heaviside step function for input x, and
- x is the input to the function.

The function takes an input x and if it is less than zero, the function returns zero, if the input is more than zero, the function returns one.

2.2 Backpropagation

Using auto-differentiation (automatic differentiation), the derivatives of the error function with respect to each parameter in the model are derived. The goal is to make the predicted output closer to the desired result by using the derivatives to determine how a parameter should be changed to take a step in that direction. The error value is *propagated back* through the network, giving changes to each parameter that, when applied to the respective parameters, yields a network that produces a better prediction. These changes for each weight in the network are a small step opposite to a high-dimensional gradient. For the *idealized* 3D case, imagine there being a minimum, the global minimum of the error function, where you want to get. At each point, you can calculate a gradient, which can be imagined as an arrow pointing upwards towards the steepest slope at that point. In backpropagation, you make small steps in the opposite direction of the gradient (which points upwards), and that way continually move closer to a minimum of the error function (Linnainmaa, 1976). However, gradient descent only finds *a* minimum, not necessarily *the* global (overall) minimum, if it exists. If gradient descent leads towards a local minimum that is sup-optimal, it is not trivial to exit a local minimum towards a better local minimum or the global minimum (simulated annealing can for example be used to attempt this). Backpropagation through time is the algorithm used to train (optimize) networks that incorporate temporal dynamics, such as spiking neural networks or recurrent neural networks (or recurrent spiking neural networks).

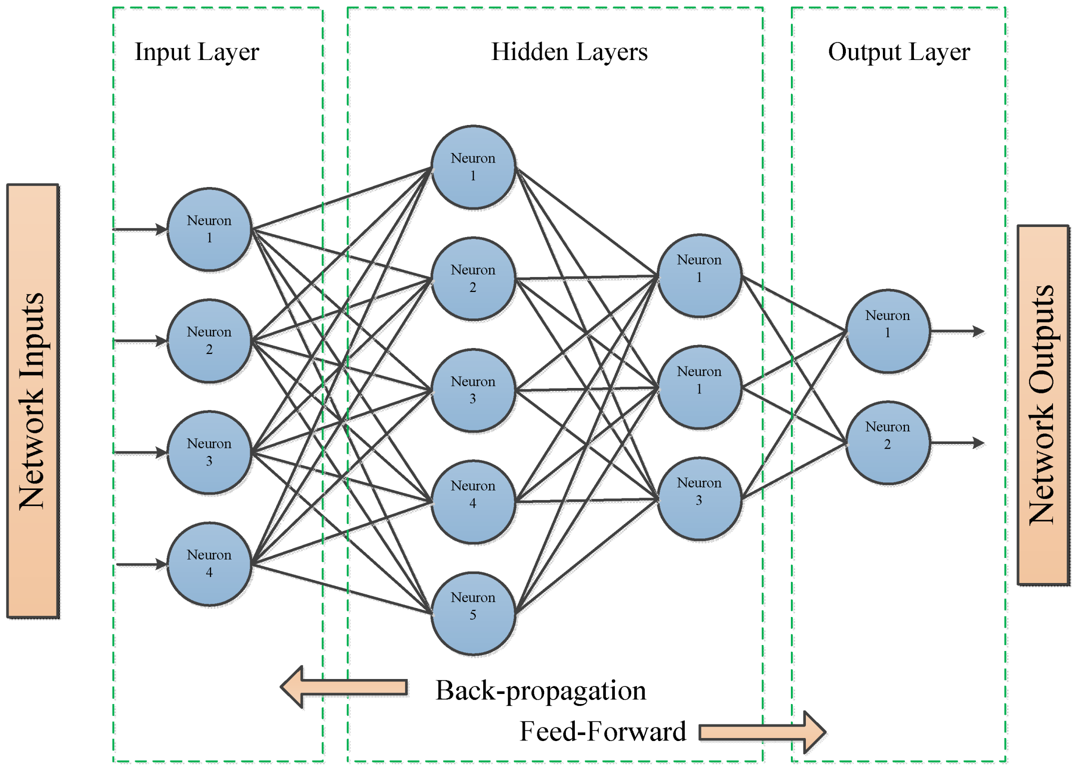


Figure 2.2: Backpropagation propagates changes back through the network. The forward pass is for computing model responses to input.

2.3 Surrogate gradient learning

With the Heaviside step function, the gradient used for calculating the weight changes in each update in backpropagation is almost always zero due to the derivative of the activation function almost always being zero, making propagating the error back into the network to change the weights ineffective. This is because the gradient vanishes if the derivative is zero (see the vanishing gradient problem). Recently, however, there have been advances in training spiking neural networks that have enabled the backpropagation algorithm using gradient descent and auto-differentiation to work on spiking neural networks. By replacing the spiking activation function (Heaviside step function) with a surrogate gradient function, spiking neural networks can be trained (Neftci, 2019). In order to obtain a derivative that is similar to the derivative of the Heaviside step function, but has a non-zero value everywhere, a function that resembles a smoothed-out step function is chosen, namely the Sigmoid function. The derivative of the activation function is then replaced by the derivative of this function. This surrogate gradient function is non-zero everywhere and highest in the place of the spike and quickly approaches zero as one goes to either side of where the spike is. Adjusting factors in the Sigmoid function can be used to change the steepness of its derivative, which affects backpropagation. The surrogate gradient function enables gradient descent. This is called surrogate gradient learning (Neftci, 2019).

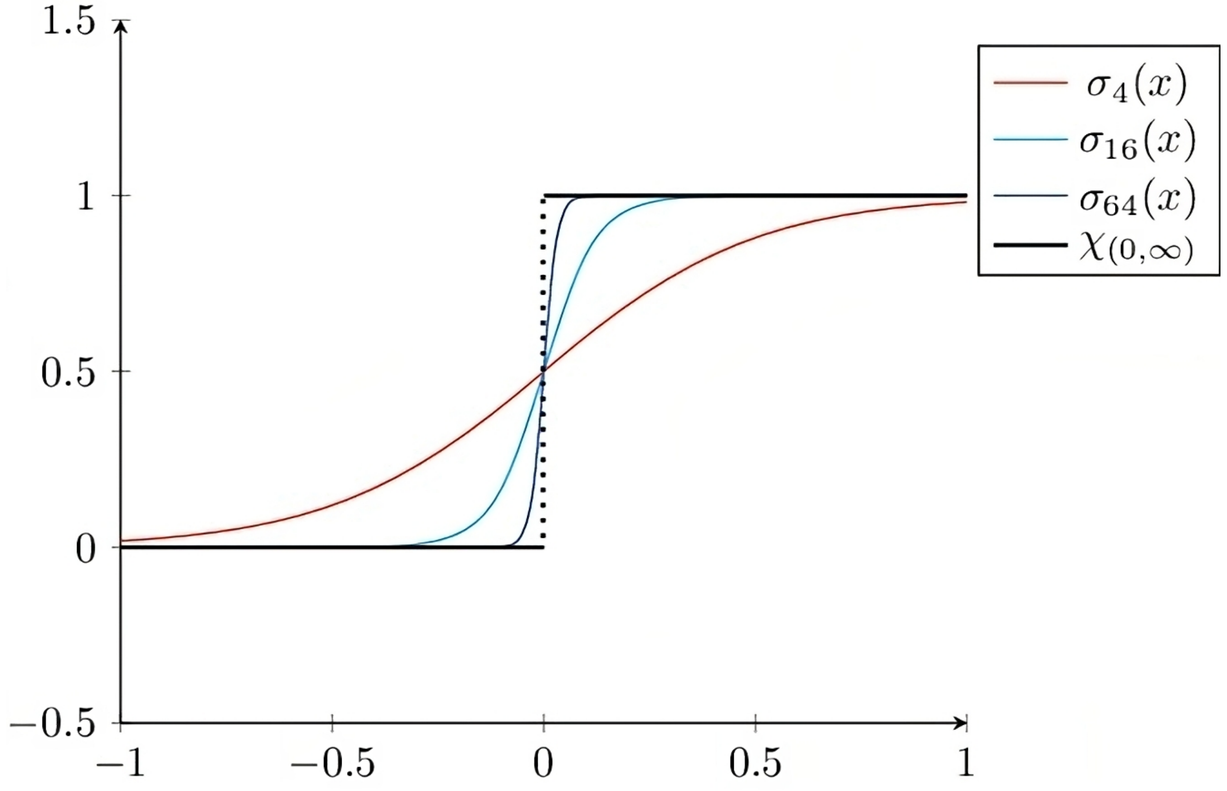


Figure 2.3: The Heaviside step activation function (X) plotted with different sigmoid functions

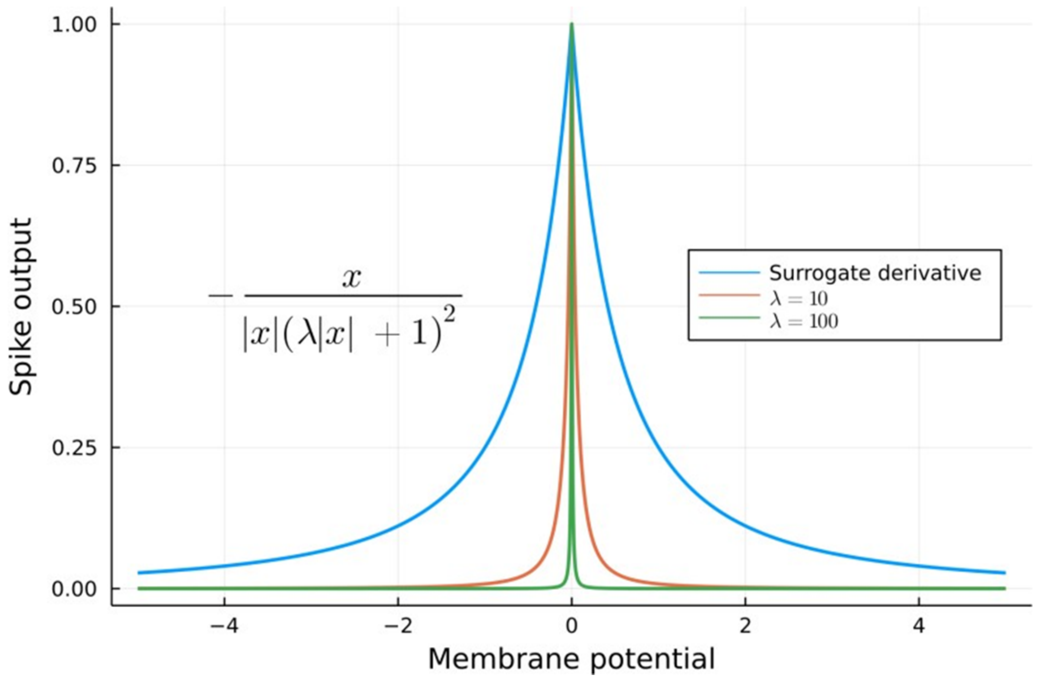


Figure 2.4: The surrogate gradient function

2.4 Data

The dataset was kindly provided to me by Hartmut Fitz, and consists of triplets of an identifier (indicating to which linguistic construction a sentence belongs), an input sentence sequence, and an output label sequence. I have used the input sentence sequence and output label sequences for this thesis. Input sequences are grammatically correct English sentences, split into individual words. Many sentences are unlikely to be expressed by a human being, but the grammar they are generated from is correct English grammar. Plural words and verbs not in infinitive form are split up into separate words. For example, the plural word "bananas" is split into "banana" and "-s", with "-s" indicating that the preceding word is in plural form. The word "walked" is split into "walk" and "-ed". Other past-tense words are also split up into infinitive form and "-ed" ending, such as "went", which is split up into "go" and "-ed". Generally, verbs that are in participle form are represented in this way. For example, a participle form of "run" is represented by the words "run" and "-par". Forms of verbs ending in "ing" or "s" are also split up into two words: "jumping" is split up into "jump" and "-ing", "throws" is split up into "throw" and "-ss", and "touches" is split up into "touch" and "-ss".
These simplifications help to limit the number of input words that have to be processed.

In total, there are 76 input words, which are: "the", "a", "man", "woman", "cat", "dog", "boy", "girl", "father", "mother", "ball", "stick", "apple", "cake", "orange", "banana", "phone", "cup", "sleep", "jump", "dance", "sit", "go", "walk", "drive", "run", "open", "close", "break", "smash", "kick", "chase", "touch", "eat", "scare", "surprise", "bother", "hurt", "put", "prn", "hit", "carry", "push", "give", "throw", "show", "present", "red", "beautiful", "nice", "big", "small", "great", "old", "smelly", "is", "was", "are", "were", "he", "she", "it", "him", "her", "they", "them", "to", "on", ".", "-ing", "-ss", "-ed", "-s", "by", "-par", and "being".

Output sequences consist of labels for each word in the sentence to which they belong. Labels are semantic roles (see the Introduction), conveying meaning of words. There are eight semantic roles in the dataset, namely ”EXPERIENCER”, ”GOAL”, ”THEME”, ”RECIPIENT”, ”AGENT”, ”ACTION”, ”EOS”, and ”PATIENT”. I added the label ’PADDING’, which I assigned for the time period in the spike train after the actual sentence had already stopped, since all sequences had to be padded to the maximum sequence length.

This is an example of an input sentence sequence-output label sequence pair:
[”a”, ”cake”, ”is”, ”being”, ”present”, ”par”, ”to”, ”them”, ”by”, ”the”, ”cat”, ”s”, ”.”]
[”THEME”, ”THEME”, ”ACTION”, ”ACTION”, ”ACTION”, ”ACTION”, ”RECIPIENT”, ”RECIPIENT”, ”AGENT”, ”AGENT”, ”AGENT”, ”AGENT”, ”EOS”]
The English sentence is ”A cake is bring presented to them by the cats.”

In order to be able to train the models in reasonable time, I limited the length of all sentences to ten words. To limit the size of the networks, I reduced the output space to the five most common labels, plus the label ’PADDING’. The five most common labels were ”ACTION”, ”AGENT”, ”GOAL”, ”EOS”, and ”THEME”. All input sentence-output label sequence pairs where the sentence was longer than 10 words long and/or where the output label sequence contained a label other than the five most common ones listed above (plus ’PADDING’) were filtered out. Due to time constraints, I trained the models on 497 pairs. They were tested on 203 pairs. 700 sequences were generated and randomly split into training and test set, using the pseudorandom number generator random.sample(), from the Random Python module.

2.5 Network

For the temporal input encoding, I used 100 input neurons, with each spike pattern being inputted across all neurons. With the temporal input encoding with a spatial component, I used (reserved) five neurons per word. Given that there were 76 unique words present in the dataset, for the temporal input encoding with a spatial component, I arrived at 380 input neurons. I used 25 hidden neurons. Due to limiting the output space to the five most common labels, plus ’PADDING, I had six output neurons.

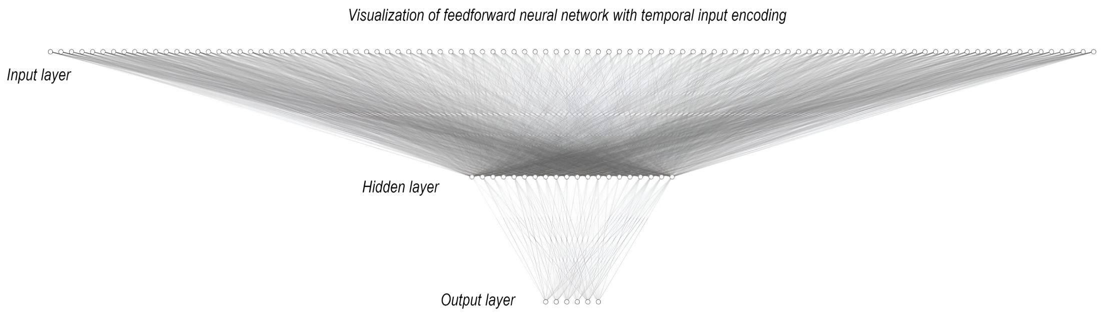

Visualization of feedforward neural network with temporal input encoding
Input layer
Hidden layer
Output layer

Figure 2.5: A feedforward neural network with 100 input neurons is visualized. This matches the number of input neurons in the model without a spatial component

2.6 Input encoding

To perform semantic role labeling with a spiking neural network using surrogate gradient learning, the input sentence sequences have to be encoded in a way that the network can understand. For the purposes of this thesis, I use temporal encoding of words, concatenated for each word in a sentence, to represent an input sequence. This means that for each word, a temporal pattern of spikes is generated that represents that unique word, and for each sentence, the different spike patterns (spike trains) are concatenated. For purely temporal encoding, the patterns have input for each input neuron, whereas for temporal encoding with a spatial component, each word maps to a certain number of input neurons (five in my case).

The specific spike pattern for each word is generated with the Poisson distribution using the Python package Numpy, and the length of the sequence of spikes is varied according to the length of the word, with 50 milliseconds for each character in the word. One time step in the network consists of one millisecond, meaning that there are 50 time steps per character.

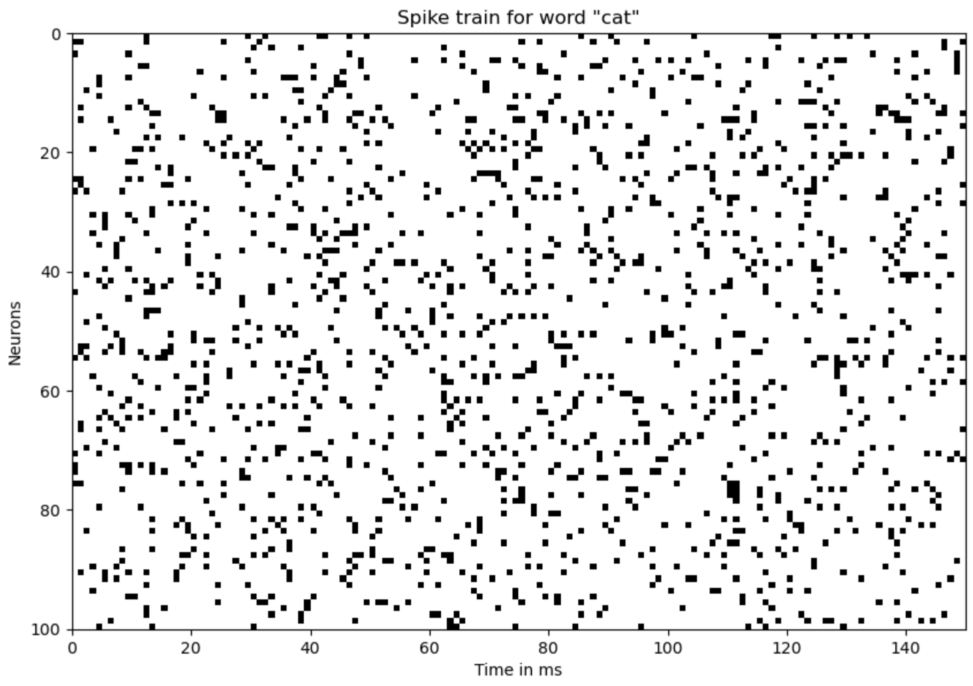


Figure 2.6: Spike train for "cat" without spatial component

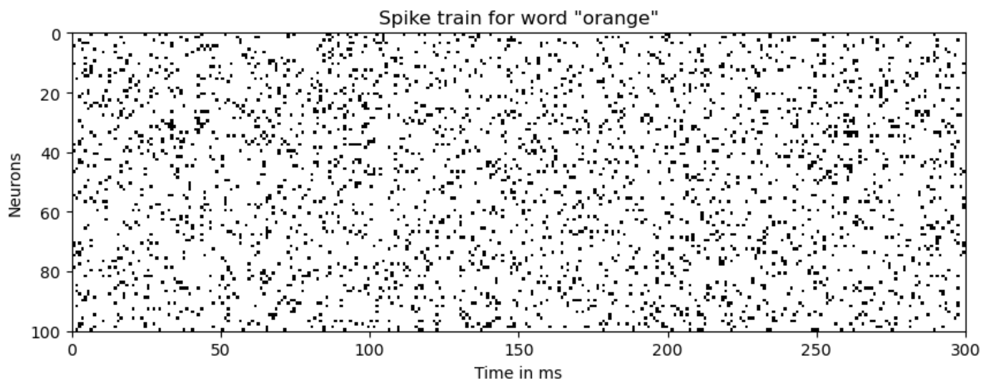


Figure 2.7: Spike train for "orange" without spatial component

The maximum length of all spike trains was that of the sentence that contained the most characters. All other sentences were padded to that maximum length to make processing them in PyTorch possible (all elements in a tensor, the data structure I used, have to have the same length). PyTorch is a Python machine learning library that I used to store input and output sequences and weights, and to optimize the network.

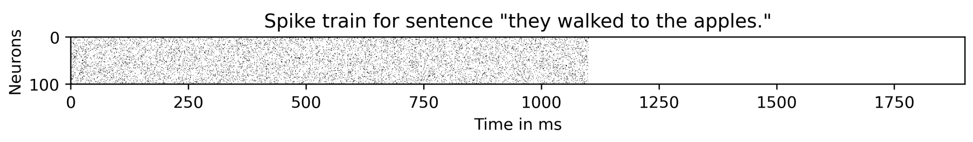
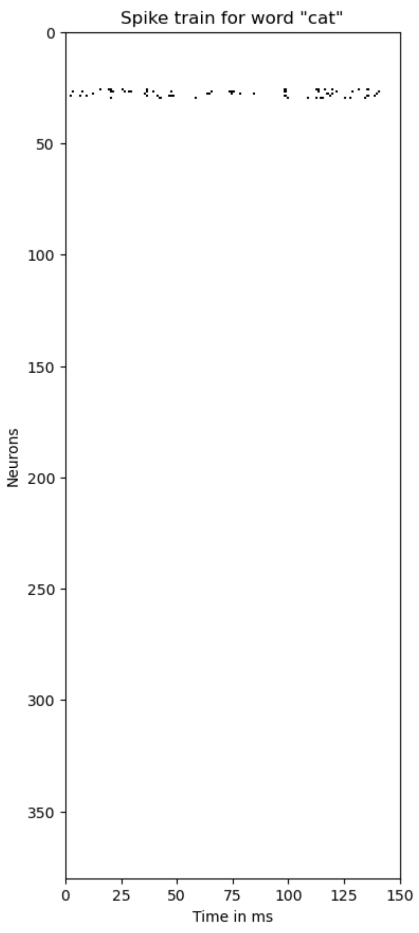


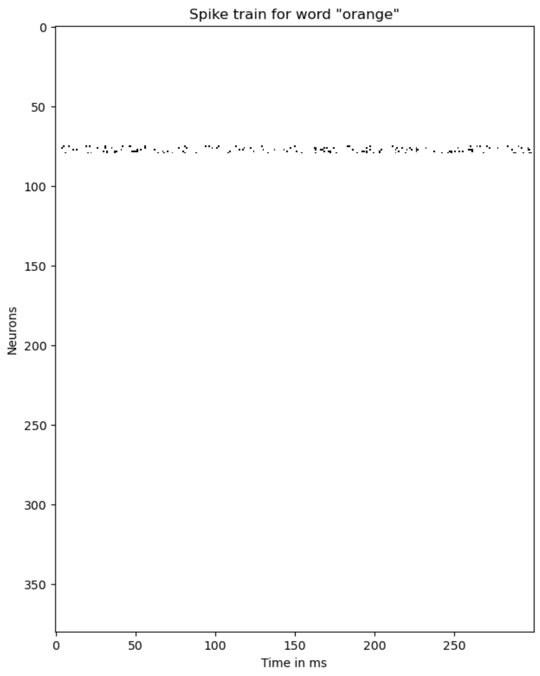
Figure 2.8: Spike train for "they walked to the apples" without spatial component

A rate-based encoding of the input, where a certain input is represented by the number of spikes that are fed into the network within a certain period of time, would make little sense as an input encoding due to the task being a classification task, not a regression task. The 'frozen noise' spike patterns for each word are better suited than a rate-based coding since they are statistically likely to be unique for each word, and coincidental similarities between words are thus less likely than with a rate-based encoding, which avoids unwanted confounding of information (Chase & Young, 2006) (although in the present study, this caveat about rate-based coding was regarding stimuli presented simultaneously). One argument against this is that 'frozen noise' spike patterns do not capture syntactic similarities between words that are similar semantically. However, it is unclear how to represent these similarities in rate-based coding. Spike trains give a more detailed temporal representation than rate-based coding, including spike timings. Furthermore, research suggests that the brain can integrate noisy language stimuli into rebust neural representations (Ding & Simon, 2013). Noisy language stimuli are not equivalent to frozen noise, but spiking neural networks have been shown to be able to use temporally encoded input to accurately classify information (Ghosh-Dastidar & Adeli, 2009). Research suggests that humans are highly competent at learning and remembering random auditory 'frozen noise' (Agus et al., 2010). Other research found that spiking neural networks trained with different learning rates using random temporal data generated with the Ornstein-Uhlenbeck process adapt their connection weights to be able to represent the input (Lehr et al., 2022).

I tested a purely temporal encoding, where each spike pattern is presented to every input neuron, and a temporal encoding with a spatial component ("spatio-temporal" encoding), where each spike pattern is presented to only a specific subset of the input neurons that are reserved for that spike pattern corresponding to a word. However, the input neurons project into the whole set of hidden neurons, which differs from a "spatial encoding"", where the subset of input neurons also only projects into a subset of hidden neurons (Uhlmann, n.d.). The spatial component in the input encoding might add another aspect to how the network can separate input to labels. The idea is that a temporal encoding with a spatial component in the input layer teaches the neural network over time to associate certain output with certain input and certain output with a subset of the input neurons. The separation in the hidden layer that a spatial encoding inherently incorporates is to be learned by the model in training. The separation present in the input layer might help make this separation in the hidden layer, helping in classification (Tavanaei et al., 2019).



(a) Spike train for "cat"



(b) Spike train for "orange"

Figure 2.9: Two spike trains for words with a temporal code with a spatial component

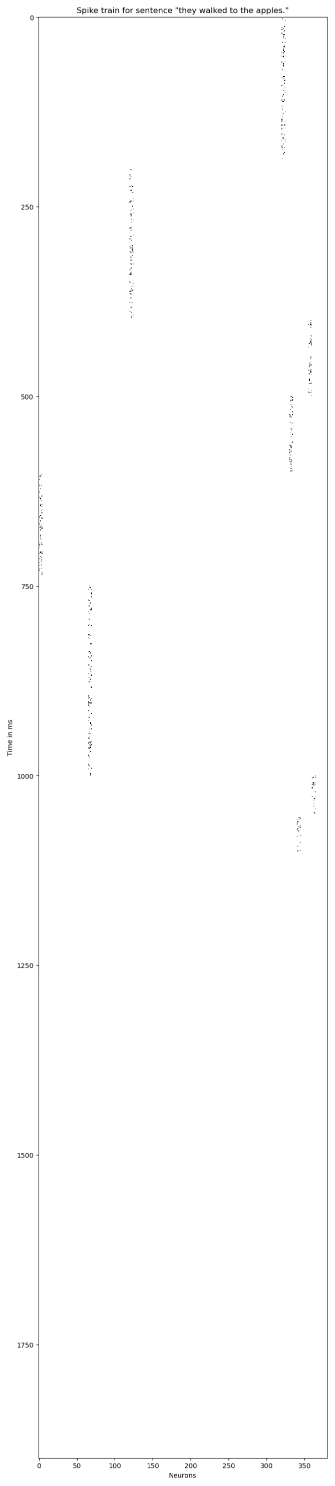


Figure 2.10: Spike train for ”they walked to the apples” with spatial component

The input data on which the model is trained are a three-dimensional PyTorch tensor with dimensions (number of sequences x (50 time steps per character x maximum number of characters in a sentence) x number of input neurons). For the training input set for the temporal models, for example, the dimensions are (497 x 1900 x 100). To test whether the model was able to adapt to the input intensity not being increased for the temporal encoding with a spatial component, I will compare the spike rates in the hidden layer before and after training and visually inspecting weights before and after training.

2.7 Output decoding

The spiking neural network is presented with the spike train, processes the input, and, over time, has a changing output. There is one output neuron for every output label. In the output layer, the voltage of the output neurons is recorded. In the input and hidden layer, the Heaviside step function is applied to the voltages of the neurons in those layers to produce spikes whenever the voltage surpasses the threshold. This is not done in the output layer. The voltages of the neurons in the output layer are used directly to compute the output labels.

The predictions made by the network can be calculated in different ways. Neurons in the output layer of the network produce voltage that changes as new input is presented to the network. It is possible to take the voltages at specific time points and base the prediction of the network for the preceding input on the label corresponding to the output neuron with the highest voltage. One can also take the average of the voltage over a time interval and take the label corresponding to the output neuron with the highest average voltage in the interval as the prediction for the input of the time interval.

Figure 2.11: Output voltage readout for example sentence

To decode the output, the recorded voltage readout (internal activation) of the output neurons over time shows which output neuron had the highest internal activation. To obtain the predicted labels for each word, the voltage readout recording is checked at the time step at the end of each word. The label corresponding to the output neuron with the highest voltage at that time step is assigned to the corresponding word; this is the predicted label.

The actual, intended, output is encoded using one-hot encoding, where the correct label is assigned a one, and the incorrect labels are assigned zeros. This enables comparison of the actual output with the predicted output, since the voltage readout at any given time point can also be interpreted as a probability distribution, indicating which label the network thinks is most likely. For comparing actual output and predicted output, log-softmax is applied to the voltage readouts corresponding to each word in the sentence and compared to the one-hot encoded actual output.

The word mappings are "THEME" $\rightarrow$ 0, "GOAL" $\rightarrow$ 1, "ACTION" $\rightarrow$ 2, "EOS" $\rightarrow$ 3, "AGENT" $\rightarrow$ 4, "PADDING" $\rightarrow$ 5. This means that for label "Agent", the one-hot encoding is [0, 0, 0, 0, 1, 0]. The output data that is trained on are a three-dimensional PyTorch tensor with dimensions (number of sequences x maximum sequence length x one-hot encoding size). For the training output set, for example, the dimensions are (497 x 10 x 6).

2.8 Error/Loss

The error between predicted and actual output can be calculated by taking the negative sum over all classes of the actual output times the logarithm of the predicted output, which is cross-entropy loss, a common function for calculating how different two probability distributions are. The comparison to calculate the error is done in this way. The more similar the probability distributions in cross entropy loss are, the smaller the value that the function returns. Thus, the goal is to minimize the result of the cross-entropy loss comparison.

For calculating the error using cross-entropy loss, this function is used:
$L = -\sum_{i=1}^{n} t_i log(p_i)$ Where:

- L is the error value (or loss),
- n is the number of classes,
- i gives the index of each class,
- t_i is the actual output vector for class i, and
- p_i is the predicted output vector for class i.

For each class, the element-wise product of the vector of the actual labels and the vector of the predictions is calculated and the negative sum of the products is taken. The outcome is a single real number that measures the discrepancy between the actual labels and the expected probability for all classes.

Making changes to the weights of the network, the parameter optimization, is done using the Adam optimizer, a widely used gradient descent optimizer.

2.9 Process

Due to time constraints (since I changed the thesis topic a short while ago), I was not able to train each network for as many epochs as I would have liked. Quickly realizing that training takes too long, I had to use Microsoft Azure for training, as my own computer was not powerful enough. Unfortunately, I was not granted access to the Snellius computation cluster at Radboud University. After the thesis presentation, I rented a server from Hetzner to run a more comprehensive analysis, but the program I used, The Littlest JupyterHub, proved unreliable, and I frequently lost training progress. Shortly after, I was luckily granted access to the Ponyland computation cluster of the Faculty of Science at Radboud University. To limit the time it takes to train the models, I trained them on 500 sentence sequences for as many epochs as time permitted, 50 epochs. Having previously trained the models and inspecting plots of the loss over epochs, I settled on a learning rate of 0.0005. To run my code on the cluster, I split it into four Python files, one for each individual model. I then ran each file individually (in parallel) on a separate server, the models with a "spatio-temporal" input encoding on servers with 64 threads each, and the models with a temporal input encoding on servers with 40 threads each. All models took approximately 2 days (48 hours) to run for the 50 epochs that I ran training.

As alluded to before, I compared feedforward spiking neural networks with recurrent spiking neural networks on the task. Research suggests that frozen recurrent neural networks are not well-suited to implement memory (Ganguli et al., 2008), making it interesting to see whether recurrence will be beneficial when training the network, as I do.

2.10 Performance metrics

The two measures of performance that I have used are the overall accuracy and the F1 score. The overall accuracy is computed by taking the predicted labels for every word and comparing those labels with the correct labels for every word. The accuracy is then the percentage of correct predictions out of all predictions. The F1 score is a performance metric that gives a more balanced view of the performance of the model. It is the harmonic mean of the precision and recall of the model. Precision is the percentage of correct predictions from the predictions made by the model, and recall is the percentage of data points that belong to a class that were correctly classified as such. They are mathematically defined thusly:
$Precision = \frac{TP}{TP+FP}$; $Recall = \frac{TP}{TP+FN}$. TP refers to true positive predictions (correct predictions), FP refers to false positive predictions (incorrect predictions), and FN refers to false negative predictions (correct predictions missed out on). The harmonic mean is a way of calculating the average that is less prone to outliers and is often used for rates. It is calculated as the reciprocal of the average of the reciprocals: $\overline{x} = \frac{n}{\frac{1}{x_1}+\frac{1}{x_2}+...+\frac{1}{x_n}}$, where $\overline{x}$ is the harmonic mean, n is the number of data points to average, x_1 is the first data point, and x_n is the last data point. Since the F1 score is the harmonic mean of precision and recall, it encourages having a balanced model that has as many correct predictions as possible (precision), but is also as good as possible at classifying data points belonging to a class as such (recall). The F1 score is normally defined for a one-class classification problem ('true' or 'false'), but by summing the instances of true positive, false positive, and false negative, it can be applied to multiclass classification. That is what I did, using the *f1_score* function from the Python module scikit-learn. I set the 'average' parameter of the function to 'micro' to sum the instances for each case (true or false positive, false negative), as described. The F1 score is then calculated as the harmonic mean of the precision and recall of the summations of true positive, false positive, and false negative for all classes.

Generally, a F1 score above 0.5 is considered good, but this is assuming binary classification on one class. Due to the fact that in this case, there are five labels, plus the added label of 'PADDING', there are six classes that are summed over. That means that a lower F1 score already indicates good performance.

Results

These are the training accuracies and F1 scores after training each of the models for 50 epochs (rounded to four decimal places).

	1)	2)	3)	4)
Accuracy	0.8149	**0.8793**	0.8773	0.6660
F1 Score	0.7501	0.7963	0.8172	**0.8202**

1) Feedforward SNN with temporal encoding trained for 50 epochs
2) Recurrent SNN with temporal encoding trained for 50 epochs
3) Feedforward SNN with spatio-temporal encoding trained for 50 epochs
4) Recurrent SNN with spatio-temporal encoding trained for 50 epochs

These are the test accuracies and F1 scores after training each of the models for 50 epochs (rounded to four decimal places).

	1)	2)	3)	4)
Accuracy	0.7734	0.8571	**0.8916**	0.6256
F1 Score	0.7437	0.7962	**0.8215**	0.8176

1) Feedforward SNN with temporal encoding trained for 50 epochs
2) Recurrent SNN with temporal encoding trained for 50 epochs
3) Feedforward SNN with spatio-temporal encoding trained for 50 epochs
4) Recurrent SNN with spatio-temporal encoding trained for 50 epochs

The accuracies and F1 scores for both the training and test sets are decent or good. The feedforward spiking neural network using a temporal input encoding with a spatial component resulted in the best test accuracy and F1 score, indicating that it is best able to generalize to unseen data. Both the feedforward and recurrent spiking neural network using a temporal encoding with a spatial component outperformed both models with a temporal encoding on the F1 score. The training accuracy of the recurrent spiking neural network using a temporal encoding was the highest, while the training F1 score of the recurrent spiking neural network using a temporal code with a spatial component was the highest. For both training and test sets, this model had the lowest accuracy. For training and test set, the feedforward spiking neural network using a temporal code had the lowest F1 score. This model performed best in no condition. The feedforward spiking neural network using a temporal code with a spatial component was the only model where the accuracy and F1 score is higher on the test set than the training set.

I have produced confusion matrices for the models for the test set. In the graphs below, the numbers for each row do not sum up to 203 (the number of data points in the test set) because they are predictions for each word. Each word is looked at as a separate prediction.

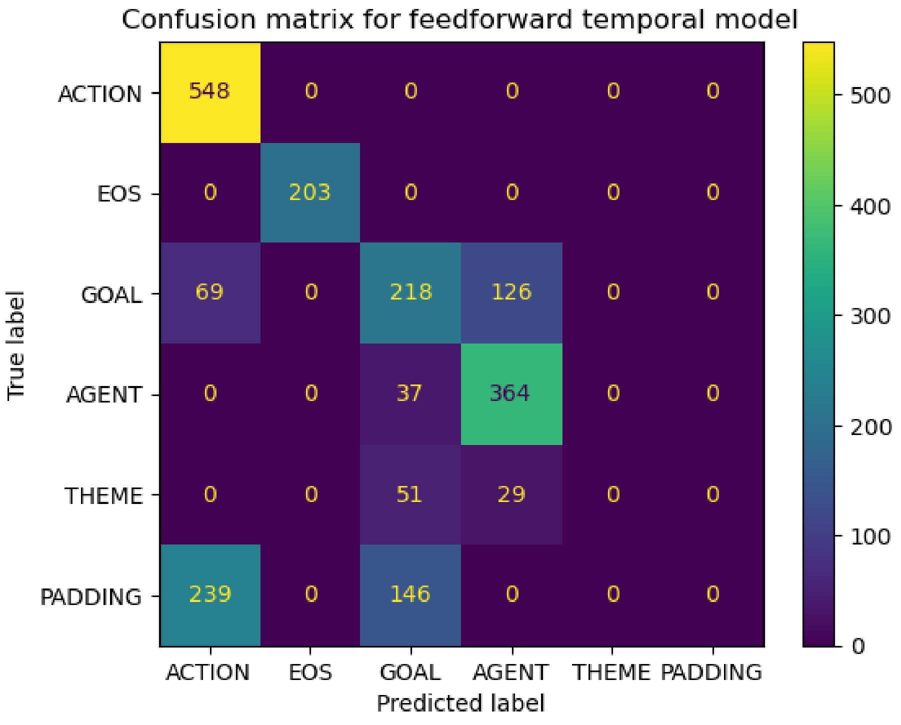


Figure 3.1: Confusion matrix feedforward temporal model

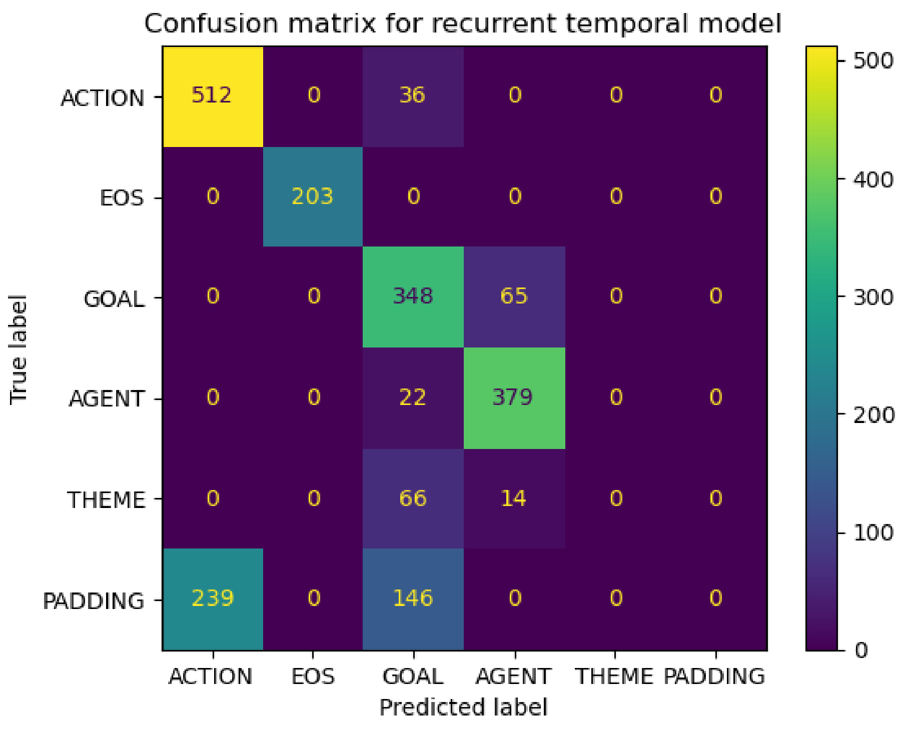


Figure 3.2: Confusion matrix feedforward recurrent model

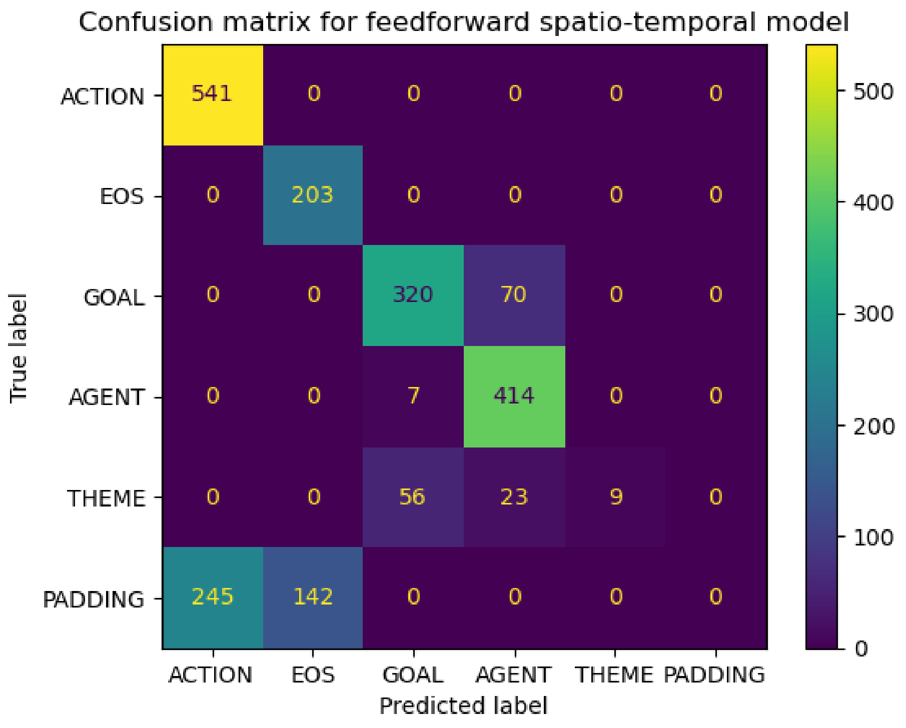


Figure 3.3: Confusion matrix feedforward spatio-temporal model

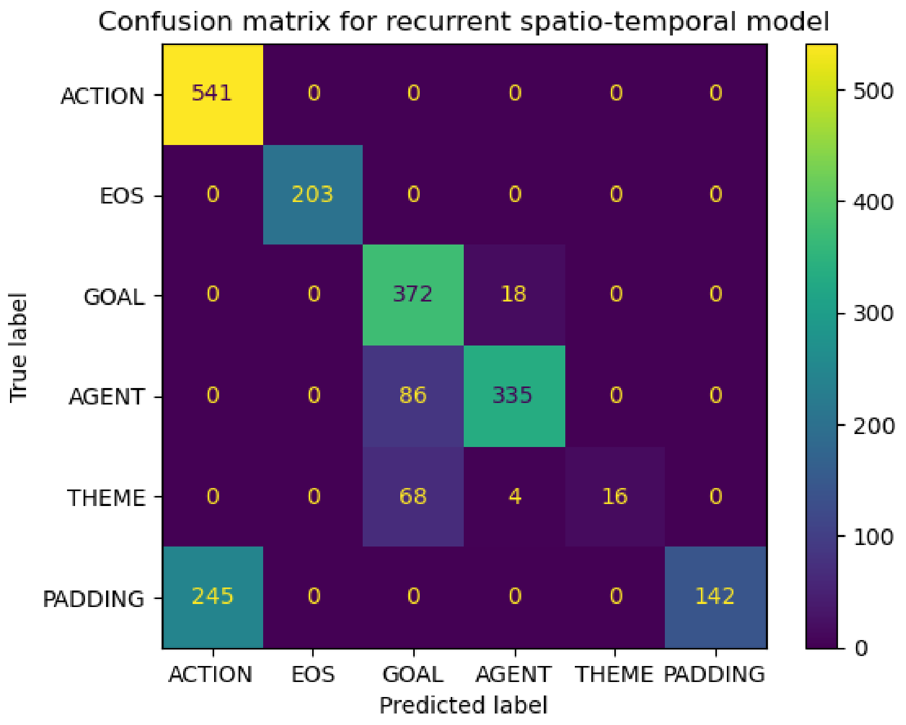


Figure 3.4: Confusion matrix recurrent spatio-temporal model

As can be seen, the recurrent spiking neural network using a temporal code with a spatial component was the only model capable of accurately predicting at least some instances for each class. The other three models are not able to predict the label ”PADDING” at all. The feedforward model using the temporal code with a spatial component is the only other model that in a few cases (nine) accurately predicted the label ”THEME”. Neither model using a temporal code ever predicted the label ”THEME” or ”PADDING”. Despite the feedforward model using a temporal code with a spatial component having a better test accuracy and F1 score, the confusion matrices suggest that its recurrent counterpart generalizes better.

However, it is unclear whether with more training the generalization performance of the models would not have improved significantly. Training a model for many epochs has been shown to be an effective technique for improving generalization performance, not necessarily leading to overfitting (Power, 2022).

These are the graphs of the loss over the course of training epochs for the models using temporal encoding:

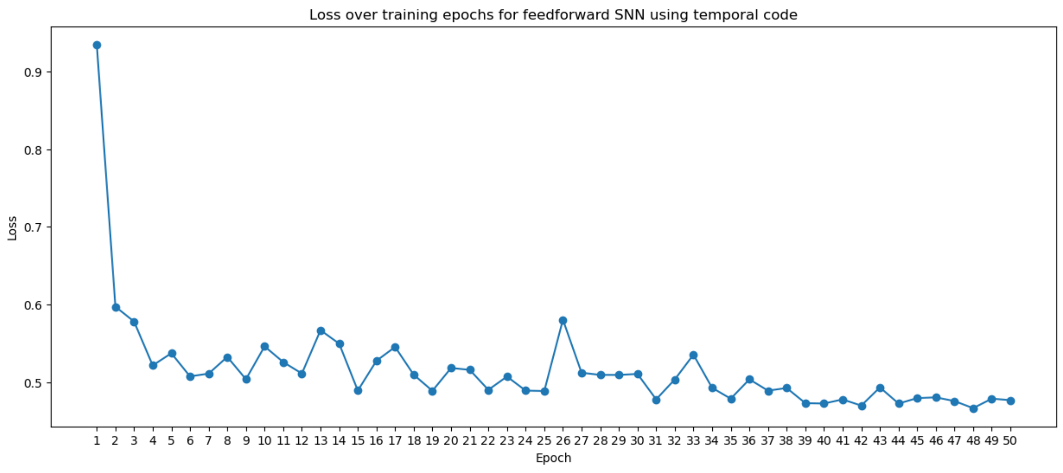


Figure 3.5: Loss plot feedforward temporal model

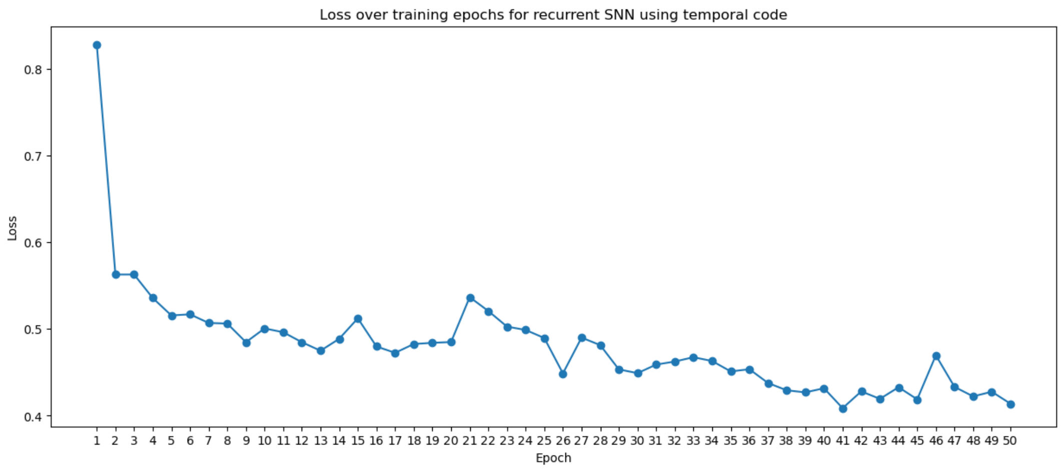


Figure 3.6: Loss plot recurrent temporal model

Fluctuations can be seen in both graphs, indicating that the learning rate might be too high, although the number of epochs for which the models have been trained is probably not enough to make a definitive statement.

These are the graphs of the loss over the course of training epochs for the model using temporal input encoding with a spatial component:

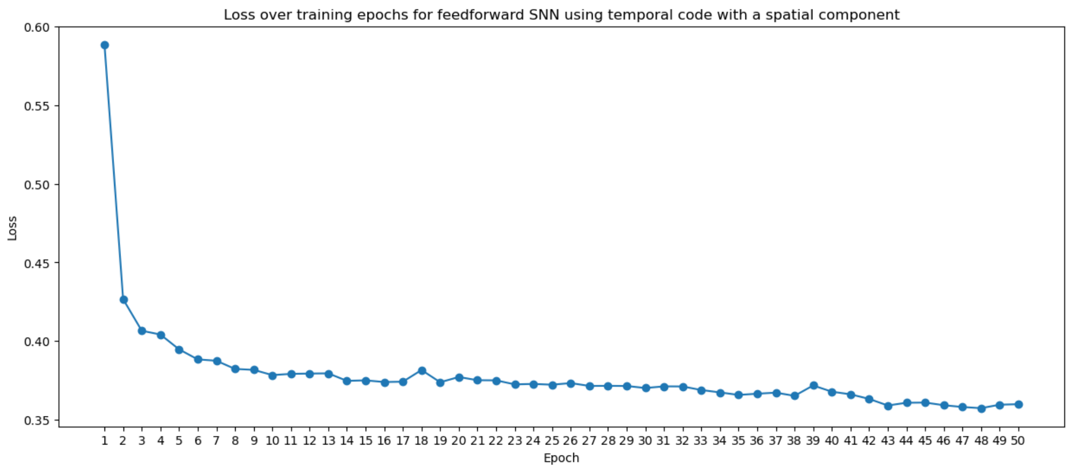


Figure 3.7: Loss plot feedforward spatio-temporal model

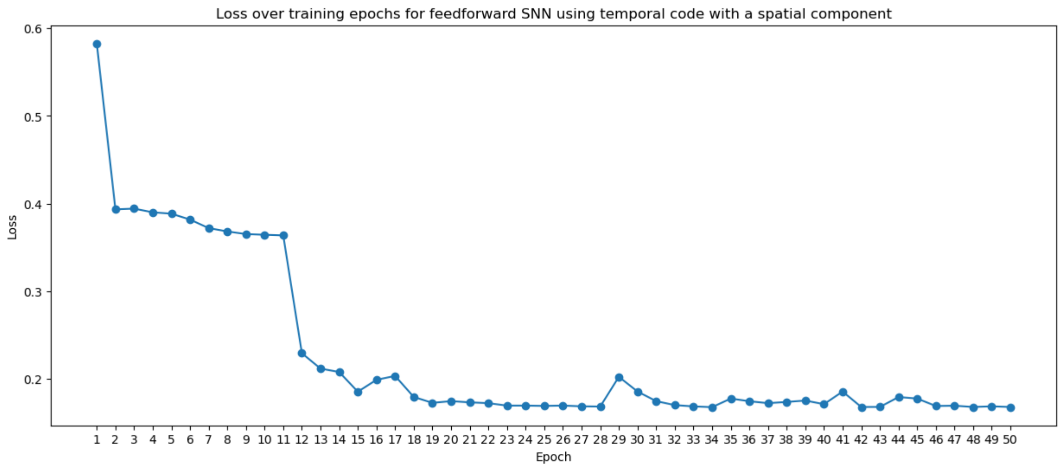


Figure 3.8: Loss plot recurrent spatio-temporal model

There are fewer fluctuations visible in these two graphs, potentially indicating that the two larger models train better with the learning rate used, which may be due to the larger number of parameters that these two models have. In the graph for the recurrent spiking neural network using a temporal code with a spatial component, a sudden drop in the loss value can be seen from epoch eleven to epoch twelve. This is an interesting phenomenon, which could be caused by the optimizer slowly moving through a plateau. This is called the barren plateau problem (Liu, 2022). A local minimum seems to have been found between the beginning of the eleventh and the end of the twelfth epoch, which the optimizer then moves towards.

Taking a look inside a network, we can see what the weight matrices of the networks from the input to the hidden layers look like, represented as heat maps. All graphs that show the weight matrices that follow use the same color mapping. Each small square in the heat maps represents a weight. Darker squares indicate lower weights and brighter squares indicate larger weights. I used the symmetrical log scale to scale from real numbers (the weights) to colors because the weights that are visible at the bottom of the heat maps for the models using a temporal code with a spatial component were drastically different from the rest of the weights, making them look very similar. This means that these heat maps are slightly inaccurate representations for the sake of visibility.

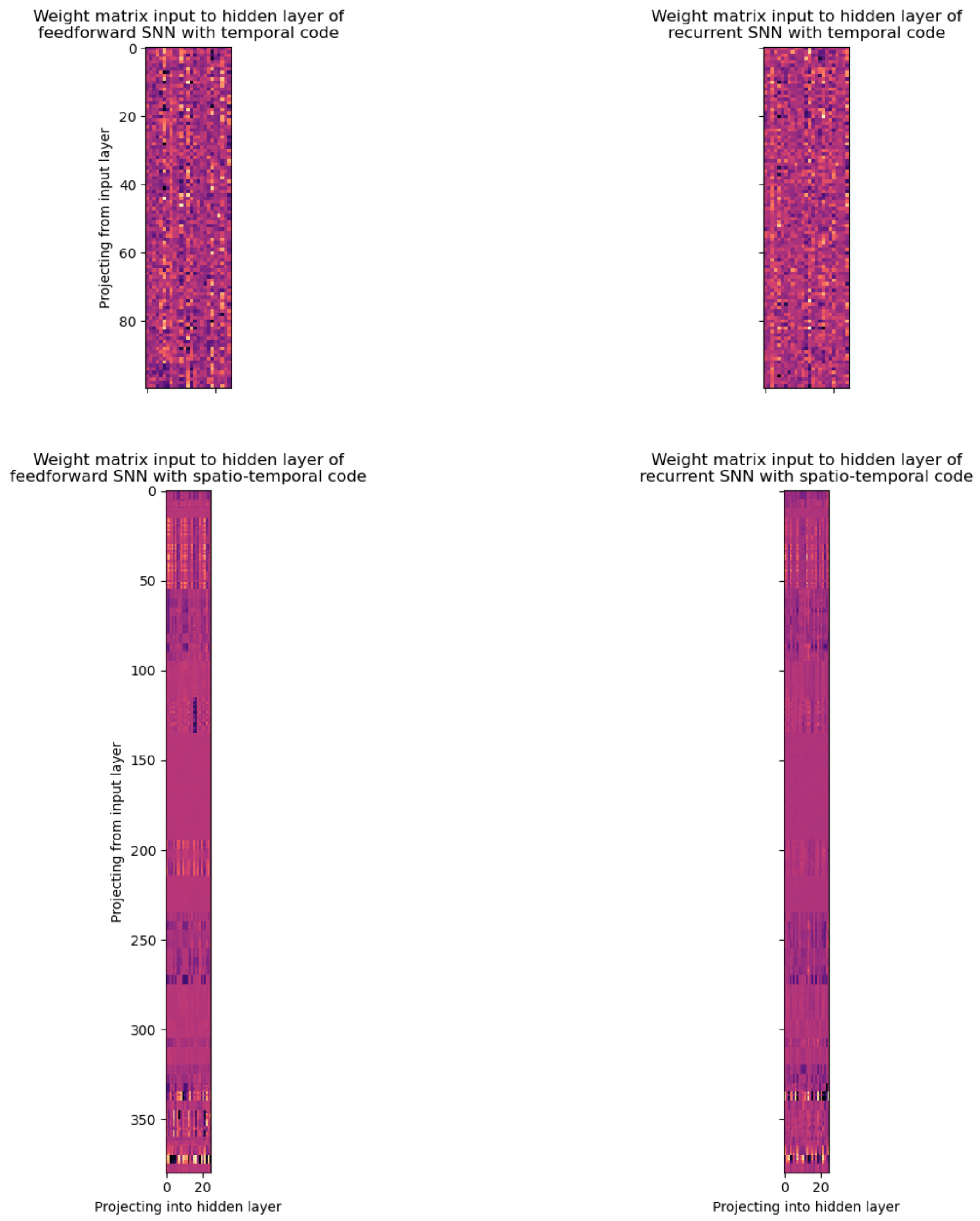


Figure 3.9

These are two matrices of untrained weights, represented with a symmetrical log scale for the color mapping.

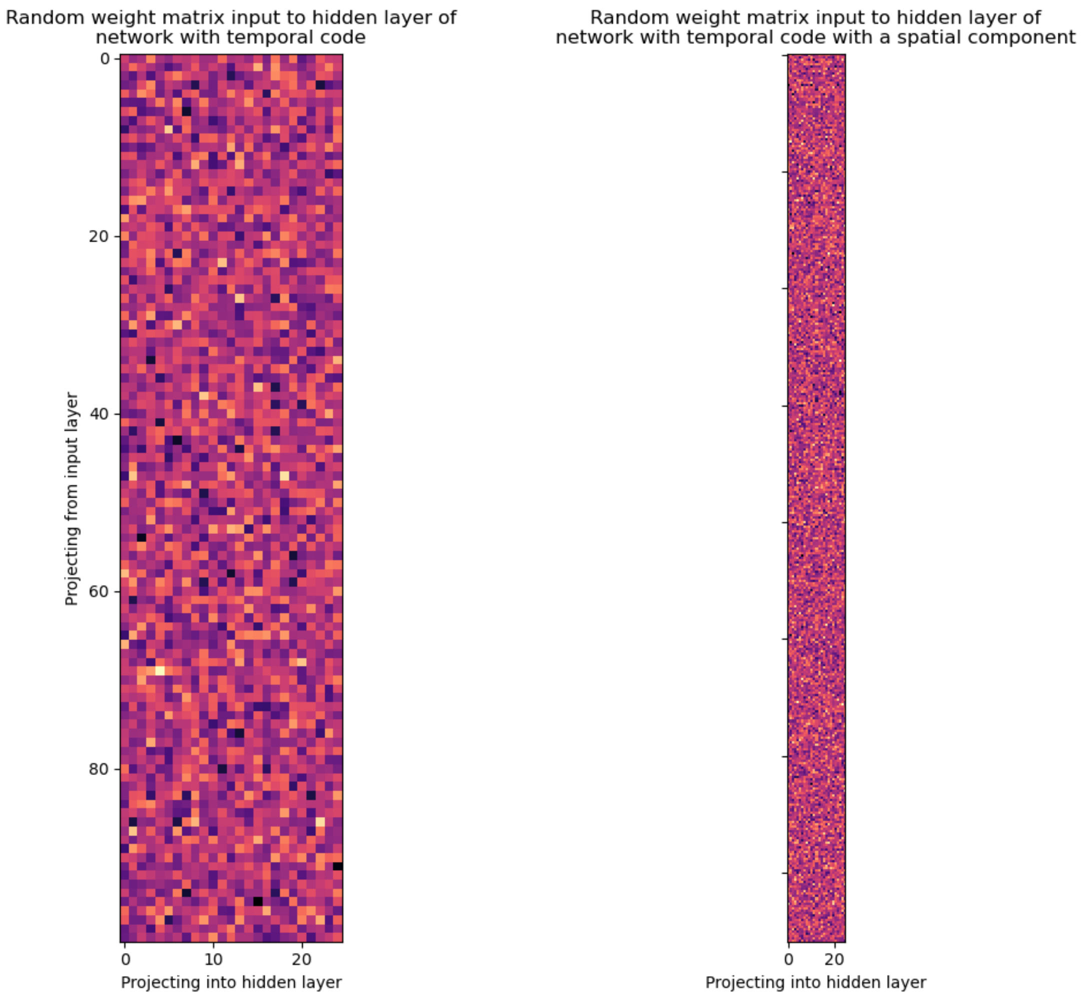


Figure 3.10

Comparing the left-hand heat map to the heat maps of the corresponding models, it is noticeable that in the matrices of trained weights, there are columns of stronger (brighter weights), which indicates that the model adjusted to fit the input. This is even more visible for the left-hand heat map and the corresponding models with a spatial component. The matrices of trained weights clearly show formed patterns.

The following are the heat maps for the weight matrices from the hidden layer to the output layer for all models. For these heat maps, the mapping from real-valued weight to color is linear because there are no extreme values to compensate for.

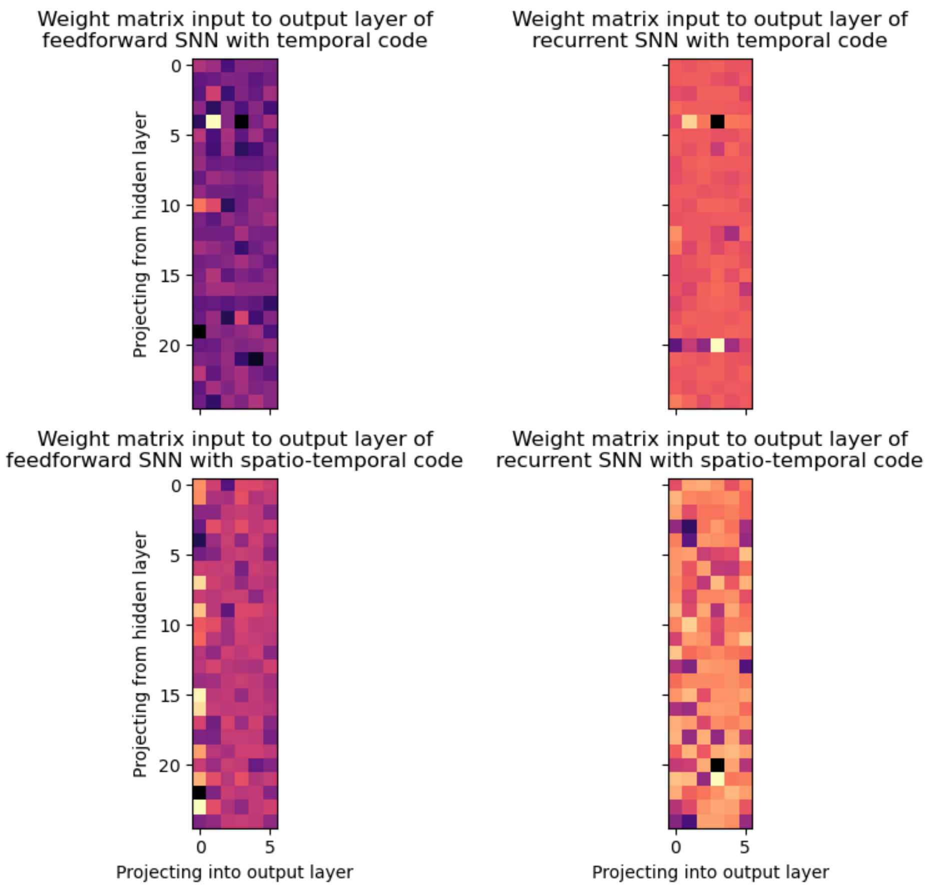


Figure 3.11

It seems that the different patterns visible here could indicate different classification and thus separation strategies that the models learned. What stands out is that the recurrent model using a temporal input encoding has mostly weights that are similar, except for weights projecting from neurons four and twenty in the hidden layer to the output layer. This might indicate that these neurons learned to recognize abstractions or dynamics in the input that are helpful for classification.

There are similarities between the heat maps of the models using a temporal encoding with a spatial component. Neither of the two models has many strong or weak connections projecting into output neurons three to five (two to four in the plot), although there are exceptions to this in both models. This might indicate a pattern in the linguistic dynamics of the input, have something to do with the frequency of the labels belonging to these neurons, or be due to another reason.

The following is again a matrix of the same dimensions of untrained weights, to determine whether the models fitted onto the data.

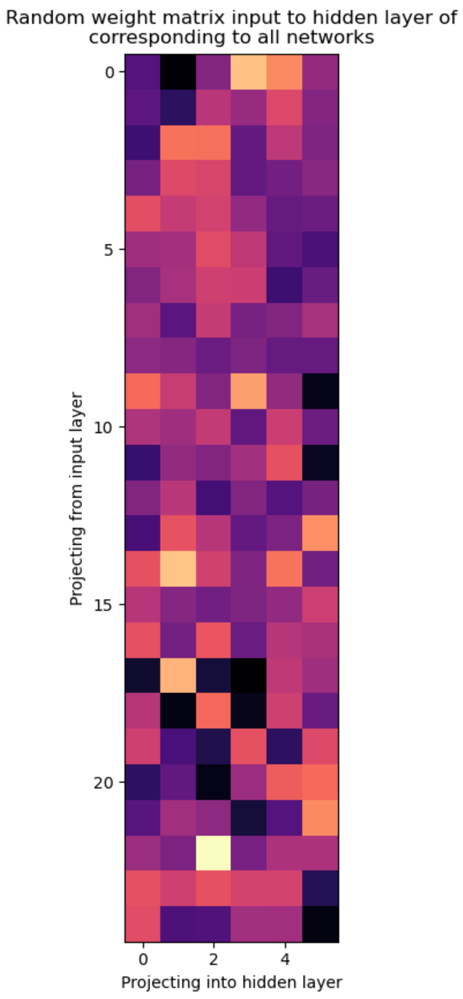


Figure 3.12

This heat map loosely resembles the heat map of the second weight matrix of the feedforward model using a temporal code, which indicates that this model was not fitted to a large degree on the data, as you would otherwise notice bigger differences between the heat maps. This could also indicate that the model needed to be trained for more epochs, or that the dynamics before the output layer were already enough to separate the input to a large degree. The heat maps of the other models look significantly different from the random heat map.

The following are the heat maps for the recurrent weight matrices. Since two of the models are not recurrent, this essentially gives insight into whether the recurrent models actually learned patterns using recurrence. Linear scale mapping from weight values to color is again used.

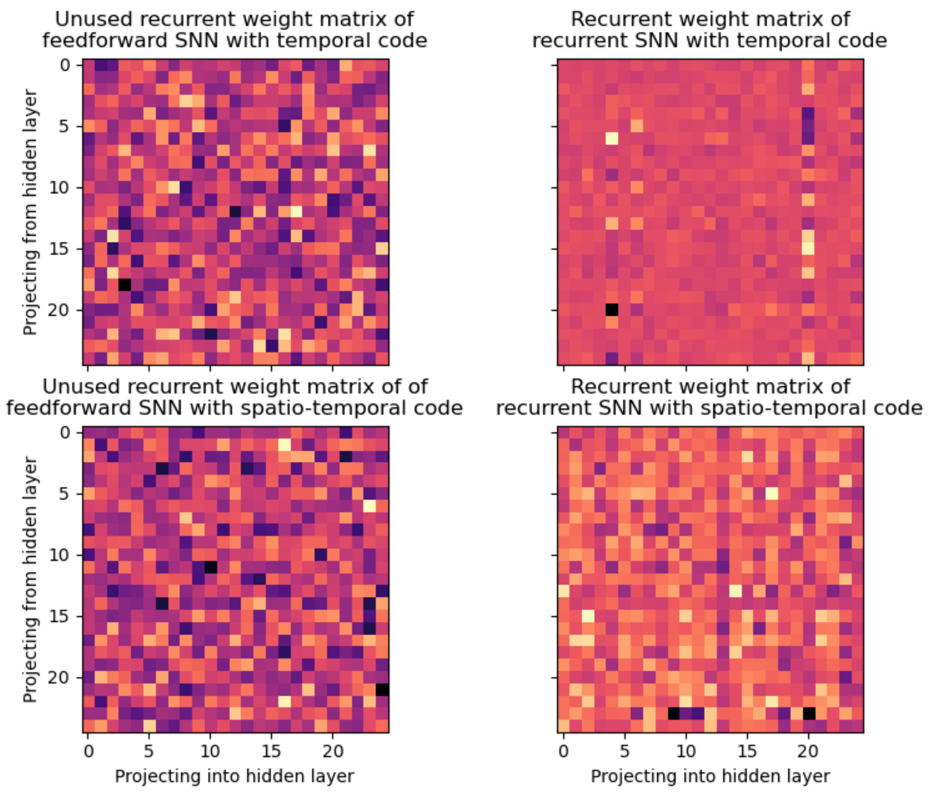


Figure 3.13

As we can see, the heat maps representing the weight matrices on the left-hand side look like random noise, whereas the heat maps on the right-hand side show that the weights were trained and changed to recognize patterns. The recurrent weight matrix belonging to the recurrent spiking neural network using a temporal code has weights that are mostly similar, with notable outliers in the column of neurons projecting back into the fourth and twentieth neurons. Since we saw in the heat map for this model of the weight matrix for the connections from the hidden to the output layer that the weights that stand out project from the fourth and twentieth neurons, this could indicate that these neurons did indeed learn an underlying pattern in the input especially well, compared to other neurons in the hidden layer.

The heat map for the recurrent model using a temporal code with a spatial component does not show such a clear preference for certain neurons. There is a spark difference to the weight matrices that were not used, suggesting that this model learned to utilize recurrent connectivity. This heat map also generally shows brighter colors, suggesting overall larger weights, which could be a way for the network to make up for the lower input intensity resulting from the input spikes being projected into fewer input neurons for the temporal code with a spatial component.

Discussion

These results indicate that spiking neural networks using surrogate gradient learning can be used to perform semantic role labeling on English sentences. The results suggest that recurrence adds to the performance of the neural network model, which contradicts other previous research, although Ganguli et al., (2008) did not train their recurrent network.

I found that models with a temporal input encoding with a spatial component perform better than models with a temporal input encoding without a spatial component. The models with a spatial component generalized better, although the recurrent network without a spatial component had a higher training accuracy.
The models adjusted their weights to fit the input with training, and the loss decreased over epochs.

4.1 Changes I could have made

I would have liked to train the models on more sequences for more epochs, using more neurons and potentially more hidden layers. However, with the training of the models already being very slow, and the time available to me being constrained, I was not able to perform the training on more data with larger models. Nonetheless, the results are interesting in their own right and certainly give a strong indication of the performances of feedforward and recurrent neural networks, and how different input encodings fare.

Another comparison that would have been interesting to perform is comparing the temporal input encodings with and without a spatial component with a spatial input encoding, where certain input neurons are reserved for certain words' spike patterns, but these input neurons also only project their output into a subset of the hidden layer neurons. Since the temporal input encoding did not fare as well as the temporal input encoding with a spatial component in my tests, it would be interesting to see how the two 'extremes' on the spectrum from spatial to temporal input encoding fare when directly compared to each other. In the brain, certain regions are only responsible for certain tasks, suggesting that spatially separated activity might help in improving spiking neural network performance.

Decoding the output in a different way would also be something interesting to investigate. For this thesis, I take the voltage readout over time of the output layer and at the end of each word, check the voltage of which neuron is the highest, the corresponding label to which will be assigned to that word. The error is also calculated using the voltage readout at the time step at the end of a word. One could decode the output by taking the averages of the voltages for the duration of each word, and assigning the label according to which average voltage is the highest. This might yield higher performance.

The time constants that determine the neuronal dynamics are currently not adjusted. Learning these time constants, treating them as model parameters essentially, might be another way to increase performance. This could also give hints as to whether the neuronal decay in the human brain tends to be smaller or larger, although measuring this directly is, of course, a far better approach for that purpose.

4.2 Future work

In its current form, the spiking neural networks of this thesis do not employ working memory in the neurons themselves. In the brain, neurons have internal memory that stores some information about input from the past and modulates the voltage in the present (Linden, 2007). This adds another way of implementing memory in addition to recurrent connections. Since it is not known whether working memory in neurons or recurrence in the brain is responsible for memorizing temporal linguistic dynamics, such a comparison might be able to indicate which approach has more of an impact, or if they work best in unison.

Performing the task of semantic role labeling on other languages could also be a fruitful research endeavor. This could be non-trivial due to different dependencies within a sentence being at play, or the symbols of a language encompassing different meaning than in English.

4.3 Acknowledgements

I would like to thank Hartmut Fitz for supervising this Bachelor thesis project and helping me along the way.
I would like to thank Alex Wegh and Levi Peters, the class mates in the same thesis group, for their help in getting my code to run.

I have adapted my code from Friedemann Zenke and from Gabriel Loye.
I would like to thank my father for a small "research grant" for Microsoft Azure, in order to be able to run the models in time.

References

The Python code accompanying this thesis can be found on GitHub.

The sources for the images can be found here.

1. Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. The Journal of Physiology, 117(4), 500–544.
https://physoc.onlinelibrary.wiley.com/doi/10.1113/jphysiol.1952.sp004764

2. Ghosh-Dastidar, S., & Adeli, H. (2009). SPIKING NEURAL NETWORKS. International Journal of Neural Systems, 19(04), 295–308.
https://www.worldscientific.com/doi/abs/10.1142/S0129065709002002

3. Pernice, V., Staude, B., Cardanobile, S., & Rotter, S. (2012). Recurrent interactions in spiking networks with arbitrary topology. Physical Review E, 85(3).
https://journals.aps.org/pre/abstract/10.1103/PhysRevE.85.031916

4. Ponghiran, W., & Roy, K. (2022). Spiking Neural Networks with Improved Inherent Recurrence Dynamics for Sequential Learning. Proceedings of the . . . AAAI Conference on Artificial Intelligence, 36(7), 8001–8008.
https://ojs.aaai.org/index.php/AAAI/article/view/20771

5. Diehl, P. U., Zarrella, G., Cassidy, A., Pedroni, B. U., & Neftci, E. (2016). Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware.
https://ieeexplore.ieee.org/document/7738691

6. Deng, S. (2021, February 28). Optimal Conversion of Conventional Artificial Neural Networks to Spiking Neural Networks. arXiv.org.
https://arxiv.org/abs/2103.00476

7. Abbott, L. F. (1999). Lapicque's introduction of the integrate-and-fire model neuron (1907). Brain Research Bulletin, 50(5–6), 303–304.
https://linkinghub.elsevier.com/retrieve/pii/S0361923099001616

8. Zenke, F. (n.d.). spytorch/notebooks/SpyTorchTutorial1.ipynb at main · fzenke/spytorch. GitHub. Retrieved June 26, 2023, from
https://github.com/fzenke/spytorch/blob/main/notebooks/SpyTorchTutorial1.ipynb

9. Linnainmaa, S. (1976). Taylor expansion of the accumulated rounding error. BIT Numerical Mathematics, 16(2), 146–160.
https://link.springer.com/article/10.1007/BF01931367

10. Neftci, E. O. (2019, January 28). Surrogate Gradient Learning in Spiking Neural Networks. arXiv.org.
https://arxiv.org/abs/1901.09948

11. Fitz, H., Uhlmann, M., Van Den Broek, D., Duarte, R., Hagoort, P., & Petersson, K. M. (2020). Neuronal spike-rate adaptation supports working memory in language processing. Proceedings of the National Academy of Sciences of the United States of America, 117(34), 20881–20889.
https://www.pnas.org/doi/full/10.1073/pnas.2000222117

12. Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., & Maida, A. S. (2019). Deep learning in spiking neural networks. Neural Networks, 111, 47–63.
https://www.sciencedirect.com/science/article/abs/pii/S0893608018303332?via%3Dihub

13. A spiking neural network based on temporal encoding for electricity price time series forecasting in deregulated markets. (2010, July 1). IEEE Conference Publication — IEEE Xplore.
https://ieeexplore.ieee.org/document/5596676

14. Ganguli, S., Huh, D., & Sompolinsky, H. (2008). Memory traces in dynamical systems. Proceedings of the National Academy of Sciences of the United States of America, 105(48), 18970–18975.
https://www.pnas.org/doi/full/10.1073/pnas.0804451105

15. Linden, D. E. J. (2007). The Working Memory Networks of the Human Brain. The Neuroscientist, 13(3), 257–267.
https://journals.sagepub.com/doi/10.1177/1073858406298480

16. Kittilä, S., & Zúñiga, F. (2014). Recent developments and open questions in the field of semantic roles. Studies in Language, 38(3), 437–462.
https://www.jbe-platform.com/content/journals/10.1075/sl.38.3.01kit

17. Alishahi, A., & Stevenson, S. (2010). A computational model of learning semantic roles from child-directed language. Language and Cognitive Processes, 25(1), 50–93.
https://www.tandfonline.com/doi/abs/10.1080/01690960902840279

18. Tomasello, M. (2000). Do young children have adult syntactic competence? Cognition, 74(3), 209–253.
https://www.sciencedirect.com/science/article/abs/pii/S0010027799000694?via

19. Kietzmann, T. C., Spoerer, C. J., Sörensen, L. K. A., Cichy, R. M., Hauk, O., & Kriegeskorte, N. (2019). Recurrence is required to capture the representational dynamics of the human visual system. Proceedings of the National Academy of Sciences of the United States of America, 116(43), 21854–21863.
https://www.pnas.org/doi/full/10.1073/pnas.1905544116

20. Uhlmann, M. (n.d.). Neurobiological Models of Sentence Processing. Max Planck Institute for Psycholinguistics.
https://repository.ubn.ru.nl/bitstream/handle/2066/226625/226625.pdf

21. Chase, S. M., & Young, E. R. (2006). Spike-Timing Codes Enhance the Representation of Multiple Simultaneous Sound-Localization Cues in the Inferior Colliculus. The Journal of Neuroscience, 26(15), 3889–3898.
https://www.jneurosci.org/content/26/15/3889.full

22. Ding, N., & Simon, J. Z. (2013). Adaptive Temporal Encoding Leads to a Background-Insensitive Cortical Representation of Speech. The Journal of Neuroscience, 33(13), 5728–5735.
https://www.jneurosci.org/content/33/13/5728

23. Agus, T. R., Thorpe, S. J., & Pressnitzer, D. (2010). Rapid Formation of Robust Auditory Memories: Insights from Noise. Neuron, 66(4), 610–618.
https://www.cell.com/neuron/fulltext/S0896-6273(10)00285-0

24. Lehr, A. B., Luboeinski, J., & Tetzlaff, C. (2022). Neuromodulator-dependent synaptic tagging and capture retroactively controls neural coding in spiking neural networks. Scientific Reports, 12(1).
https://www.nature.com/articles/s41598-022-22430-7

25. Power, A. (2022, January 6). Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets. arXiv.org.
https://arxiv.org/abs/2201.02177

26. Liu, J. (2022, June 19). Laziness, Barren Plateau, and Noise in Machine Learning. arXiv.org.
https://arxiv.org/abs/2206.09313