**Recurrent Neural Networks Tutorial – Introduction to RNNs**

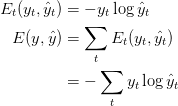
<http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/>

* The idea behind RNNs is to make use of sequential information
* You can think of the hidden state s\_t as the memory of the network. s\_t captures information about what happened in all the previous time steps. The output at step o\_t is calculated solely based on the memory at time t. As briefly mentioned above, it’s a bit more complicated in practice because s\_t typically can’t capture information from too many time steps ago.
* Unlike a traditional deep neural network, which uses different parameters at each layer, a RNN shares the same parameters (U, V, W above) across all steps. This reflects the fact that we are performing the same task at each step, just with different inputs. This greatly reduces the total number of parameters we need to learn.
* The above diagram has outputs at each time step, but depending on the task this may not be necessary. For example, when predicting the sentiment of a sentence we may only care about the final output, not the sentiment after each word. Similarly, we may not need inputs at each time step. The main feature of an RNN is its hidden state, which captures some information about a sequence.

Backpropagation Through Time (BPTT)

\begin{aligned}  s_t &= \tanh(Ux_t + Ws_{t-1}) \\  \hat{y}_t &= \mathrm{softmax}(Vs_t)  \end{aligned}  

Loss:



Chain rule application (e.g. at t=3):

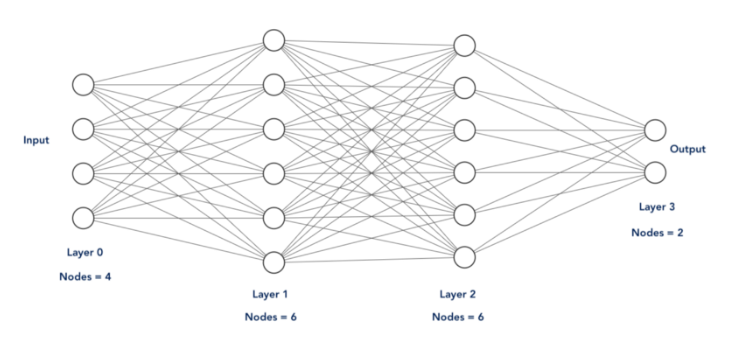


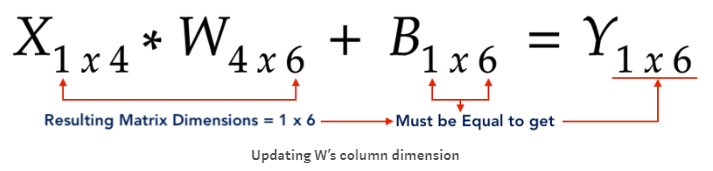
Now, note that s_3 = \tanh(Ux_t + Ws_2) depends on s_2, which depends on W and s_1, and so on. So if we take the derivative with respect to W we can’t simply treat s_2 as a constant! We need to apply the chain rule again and what we really have is this:

“Normale partielle Ableitung an dieser stelle, alle Einflüse auf ds3/dW müssen aufsummiert werden, also alle vorherigen Zustände sk an dieser Stelle.

“The key difference is that we sum up the gradients for W at each time step. In a traditional NN we don’t share parameters across layers, so we don’t need to sum anything. But in my opinion BPTT is just a fancy name for standard backpropagation on an unrolled RNN”

**Representing neural network with vectors and matrices**





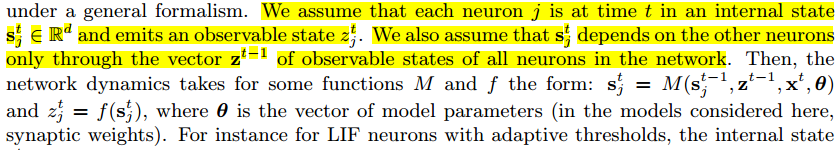
Note: the above illustration is for layer 1 and the matrix representation is always done for the output of one layer. This means Y is in this case representing the output vector of layer 1.

Difference between broadcast alignment and feedback alignment ?

**Biologically inspired alternatives to backpropagation through**

**time for learning in recurrent neural nets - Guillaume Bellec, Franz Scherr**

* BPTT is unrealistic from a biological perspective, since it requires a transmission of error signals backwards in time and in space, i.e., from post- to presynaptic neurons.



Conclusion:

* ztj is the output-vector of neurons j at time t, or in other words a (possible) spike of neuron j
* stj is the weighted sum of inputs (s is a vector) of neurons j at time t (state)
* xt is the input vector at time t
* The general goal is to approximate the gradients of the network error function E

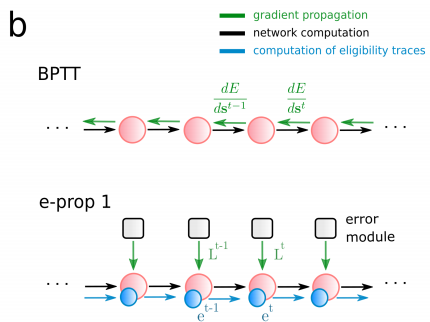
with respect to the model parameters θji

* If the error function E depends exclusively on

the network spikes E(z1, . . . , zT), the fundamental observation for e-prop is that the gradient

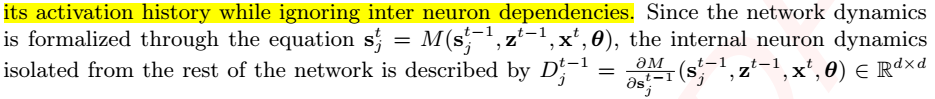
with respect to the weights can be factorized as follows (see Methods for a proof):





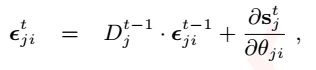
The propagation of error signals backwards in time of BPTT is replaced in e-prop algorithms by an additional computation that runs forward in time: the computation of eligibility traces.

**Eligibility traces**



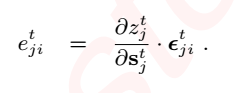
so Dt-jj is basically dstj/dst-1j meaning that it describes only the transition of s through time, ignoring the dependencies from the other neurons states.

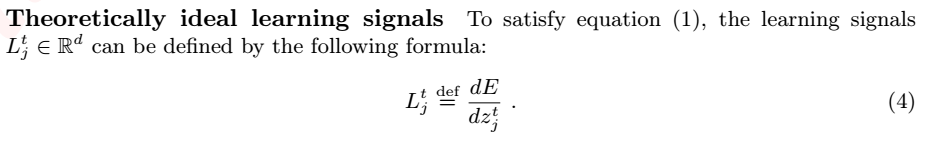
*we formalize the mechanism that retains information about the previous activity at the synapse i → j by the eligibility vector*



This is the eligibility vector and denotes previous activity of the synapse i->j:

* The temporal derivative Dt-1j multiplied by the previous eligibility vector of synapse ji plus the derivative of the state sjt to the weights ji ?
  + Why is the last part necessary if this is supposed to retain the previous activities ?
  + D is basically @leak@
  + Ds/dtheta ist he activitz on the synapse in the current time step
* So basically a sum of the influences of time (D) and the influences of weights (ds/dtheta)?
* And what is the eligibility vector epstj at time t=0?



* The left derivative here (dz/ds) denotes how the existence of a spike z depends on the neurons state s. this derivative multiplied by the eligibility vector

dE/dztj is a total derivative and quantifies how much a current change of spiking

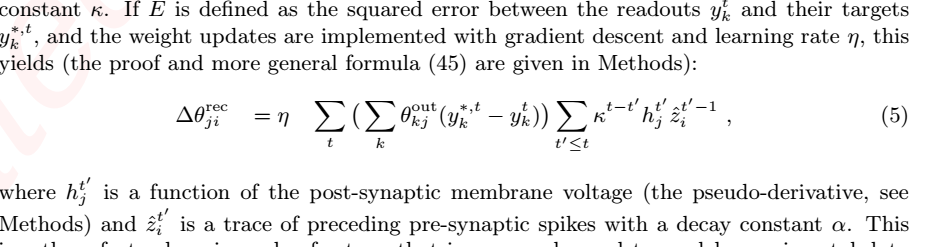
activity might influence future errors. Therefore, a direct computation of the term Ltj needs to back-propagate gradients from the future as in BPTT. However we show that

e-prop tends to work well if the ideal term Ltj is replaced by an online approximation ^Ltj

* **The three variants of e-prop are variants of these approximations. all the resulting gradient estimations described below can be computed online and depend only on quantities accessible within the neuron j or the synapse i → j**

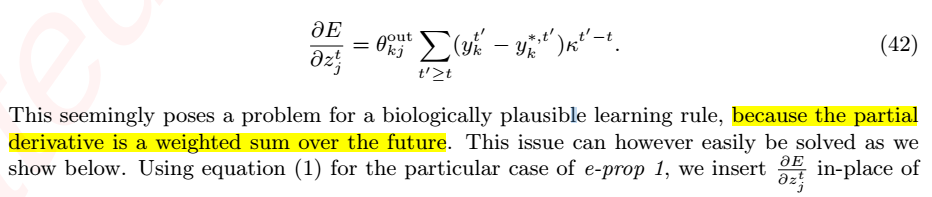
*e-prop1 - Learning signals that arise from broadcast alignment*

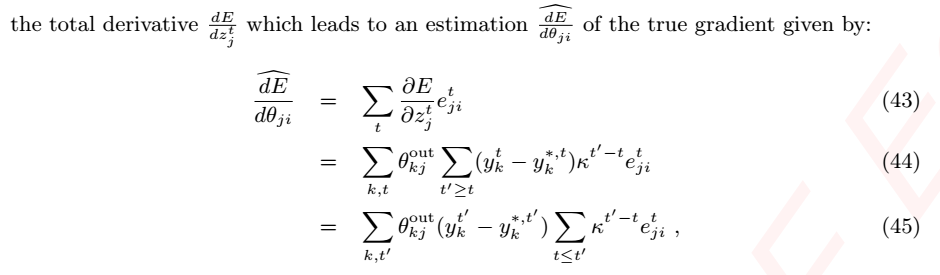
* broadcast alignment to the unrolled feedforward version of a recurrent network, one still runs into the problem that an error broadcast to an earlier time-slice or layer would have to go backwards in time.
* In e-prop1 this can be avoided



* ??

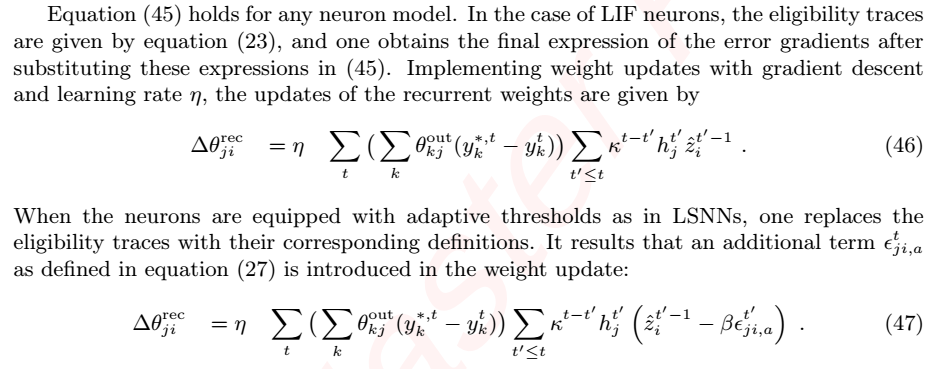
Broadcast alignment (Lillicrap et al., 2016) suggests the replacement of θ outkj by a random feedback matrix. The learning signal is a neuron-specific random projection of signed error signals y∗tk − ytk. Hence we replaced the weights θoutkj with random feedback weights denoted Brandom





where κ ∈ [0, 1] defines the leak and boutk denotes the readout bias. The leak factor κ is given by e−δt/τout , where δt is the discrete time step and τout is the membrane time constant

* Im not understanding the sum indices in this equation, how is the approximated Error derivative any less dependent of future steps ? isn’t that what t’ implies ? is t’ now in the past ?

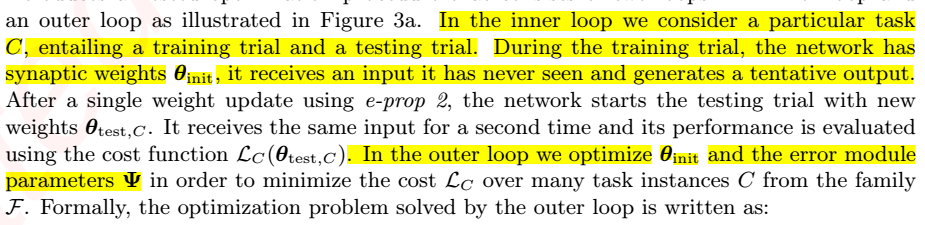


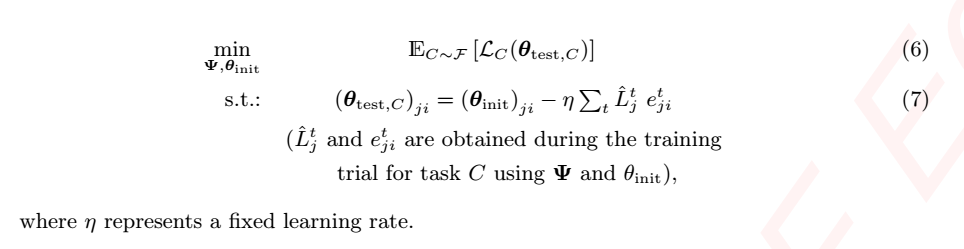
* If this is the final equation, why does the eligibility trace look so different from the above definition ? ht’j seems to be the approximation of the spike derivation dz/ds, but epsilontji looks way different.

**e-prop2**

* In e-prop 2 we apply the Learning-to-Learn (L2L) framework to train separate neural networks – called error modules – to produce suitable learning signals for large families of learning tasks
* Since this outer loop is not meant to model an online learning process, we are not concerned here by the backpropagation through time that is required in the outer loop
* Why ?

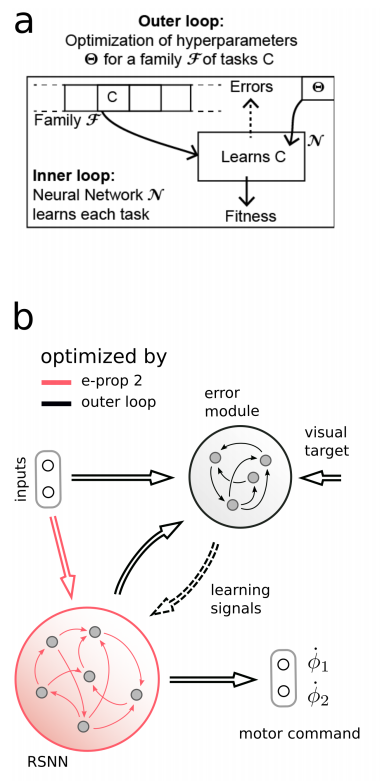
The main characteristic of e-prop 2 is that the learning signals Ltj are produced by a trained error module, which is modeled as a recurrent network of spiking neurons with synaptic weights Ψ. It receives the input xt, the spiking activity in the network zt and target signals y∗,t. Note that the target signal is not necessarily the target output of the network, but can be more generally a target state vector of some controlled system.





is this correct?

* What we do here is to minimize the cost function (loss) over all tests of all tasks. After the first training run, weights θ are updated according to the Learning signal L and the eligibility traces of the synapse i->j. The Learning signal L is determined in the outer L2L loop, which is a error module consisting of an RSNN with weights Ψ and target signals y∗,t (target output or target state), where the outler L2L NN learns with BPTT
* Important: the outer loop L2L RSNN does NOT one-shot learning! The outer NN is trained sufficiently with different tasks
* Isnt one-shot learning in this case extremely dependent of the learning rate ?



Keywords for further reading:

-one-shot learning

-L2L (Lillicrap)

-broadcast alignment

# Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets by Bellec et al.[[1]](#footnote-1)

…(from above)

# L2L Basic Concepts

Learning to Learn (L2L) is a concept that achieves learning a family F of tasks by applying a split architecture of nested Neural Networks

The Neural Network of the outer Loop aims at optimizing a set of hypterparameters 𝚯 which are used by the Neural Network N of the inner loop

[[2]](#footnote-2)

A family

*Sidenote*: This might also represent influences by the evolutionary development of the human brain

Several implementations of the underlying learning algorithms and partitioning between inner and outer loop and the representation of hypterparameters 𝚯 exist :

* BPTT, BP, RL, GA, ES and so on are possible Learning algorithms
* All synaptic weights of N are considered Hypterparameters up to optimization by the outer Neural Network (possibly biologically realistic solution, Hochreiter et al.)
* The Hyperparameters determined by the outer NN are considered priors for the NN N and up to later optimization by the inner loop for a specific task C (Legenstein et al.)
* Assigning parts of the synaptic weights of N to the optimization realm of 𝚯, while others remain up to optimization by N

L2L is a very general biologically plausible approach [[3]](#footnote-3) and therefore a relevant Framework for the field of neuroscience as well.

## LSTM

Long short-term Memory extend classic neural networks by the ability to hold information over time and furthermore protects it from dynamical disturbances as can be observated in RNNs. LSTM cells contain a input-, forget-, and output-gates which are up to L2L optimization in most applications.

Due to the protection the contents of memory cells in LSTM units they pose a solution to vanishing and exploding gradients in BPTT through large number of layers or timesteps (which essentially pose the same workload to BPTT when RNNs are unfolded). (Question: Maas on slide 5 says this is avoided by a recurrent connection with weight 1, but that’s not the same as introducing a LSTM unit is it ?)

* Why does L2L Framework exactly require an LSTM network ? is it because of the above mentioned vanishing / exploding gradient ? I don’t fully understand how this is prevented with LSTM

## Learning to Reinforcement learn by Wang et al.[[4]](#footnote-4)

Application of L2L to reinforcement learning algorithms (meta-RL):

* Inputs of N are s, a, r
* Outputs are π and V(s) (and also backpropagated)
* N is an A2C implementation, learning algorithm is BPTT
* Inner loop LSTM network encodes RL-algorithm and tries to learn an approximation of lower complexity of it

“A key point, which we will emphasize in what follows, is that this learned RL procedure can differ starkly from the algorithm used to train the network’s weights.”[[5]](#footnote-5)

At the episode start, the agents internal state is reset. The agent executes Actions based on its policy, which is at this point based on the distribution D of the prior over the Markov Decision Process. For a number of steps and episodes, the rewards received so far are observed and saved. After execution of a whole episode, the network weights are trained to maximize the overall sum of rewards. [[6]](#footnote-6)

This Meta-RL was applied to two families F of 2-arm bandit tasks with

* Unstructured rewards
* Structured rewards

Where the Meta-RL unit (learned on structured tasks) outperformed the best known algorithms which have been trained on the unstructured rewards[[7]](#footnote-7) (how is this an accomplishment, if the other algorithms learned on the unstructured tasks ?)

## Fast reinforcement learning via slow reinforcement learning by Duan et al.[[8]](#footnote-8)

RL2 aims at bridging the gap between the number of trials needed before learning a task by classic deep RL algorithms and animals or humans.

A slow general purpose RL algorithm learns the RNN weights. These weights encode the algorithm, whereas the activations of the RNN encode the state of the “fast” RL algorithm on the current MDP.[[9]](#footnote-9)

… to be continued

# Applications of Spiking Neural Networks in L2L based architectures [[10]](#footnote-10)

So far L2L-based Frameworks have mostly depended on LSTM-Networks and works considering SNNs in L2L architectures are scarce. However

## Regarding the inner Network: Long short-term memory and learning-to-learn in networks of spiking neurons[[11]](#footnote-11)

* A family F of learning tasks
* An RNN N with hyperparameters ϴ
* Fitness c(C) is determined by test runs of the inner loop (i.e. ϴ is fixed) on random tasks C from the same family F
* ϴ optimization algorithm is BPTT

Family of Tasks F defined by a FFNN, for training the LSNN receives the inputs and the target outputs C’ from the FFNN, thus becoming a supervised learning problem.

• The LSNN’s weights W are fixed while learning a function C. W is derived from ϴ , which are the hyper-parameters, and determined by the outer loop. Thus the weights W of the LSNN also encode its learning algorithm (not quite sure if I get this sentence, they encode what the inner loop NN starts with as a prior, but why do they represent the learning algorithm i.e. BPTT and so on ?).

This means, during training of the outer loop NN N, it tries to create a set of weights W (i.e. hyperparameters ϴ) with which the inner loop NN performs well for the given random Tasks C of family F. These weights W are priors for the inner NN for any of the Tasks C from family F and will be refined during the specialized training for the inner NN on specific tasks.

* Did I understand this correctly ?
* “After training the LSNN in the outer loop, the weights ! of the LSNN and the weights of the linear readout are frozen.” (Fuerberg, Anand in L2L pdf): What is even left to adjust for the inner loop neural network, if the weights W are frozen ? Is it only the inner memory of the LSTM units ?

The LSNN weights W are trained with BPTT.

….

Will do more on the “Long short-term memory and learning-to-learn in networks of spiking neurons” by bellec …

## Regarding the outer Network and Learning Signals: Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets [[12]](#footnote-12)

… partly done above, but to be continued…

# Literaturverzeichnis

Anand, Subramoney. 2018. Network of spiking neurons - learn to learn.

Bellec, Guillaume, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. 2018. Long short-term memory and Learning-to-learn in networks of spiking neurons. In *Advances in Neural Information Processing Systems 31*, edited by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, 787–797. Curran Associates, Inc. Available at <http://papers.nips.cc/paper/7359-long-short-term-memory-and-learning-to-learn-in-networks-of-spiking-neurons.pdf>. Accessed 4 June 2019.

Bellec, Guillaume, Franz Scherr, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. n.d. Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets: 37.

Duan, Yan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. 2016. RL$^2$: Fast Reinforcement Learning via Slow Reinforcement Learning. *arXiv:1611.02779 [cs, stat]*. Available at <http://arxiv.org/abs/1611.02779>. Accessed 27 May 2019.

Maass, Wolfgang. n.d. Introduction to L2L, LSTM networks, and key results from Wang et al., 2016: 12.

Wang, Jane X., Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z. Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. 2016. Learning to reinforcement learn. *arXiv:1611.05763 [cs, stat]*. Available at <http://arxiv.org/abs/1611.05763>. Accessed 26 May 2019.

1. **Introduction**
2. **Recurrent Neural Networks and Long short-term memory**
3. **Spiking Neural Networks**
   1. **Neurons, Activation and Synaptic Plasticity**
   2. **Spike-based Neural Codes**
   3. **Learning in Spiking Neural Networks**
      1. **Backpropagation and Feedback-alignment/Braodcast-alignment**
      2. **Error Feedback**
   4. **Performance of SNNs**
4. **Learning to Learn (L2L)**
   1. **Learning to reinforcement learn (meta-RL)**
   2. **L2L in the context of Spiking Neural Networks**
   3. **Implications for Neuroscience and Psychology**
5. **Applications of L2L and Spiking Neural Networks in Robotics**
   1. **(E.g. Navigational tasks with Meta-RL and SNN)**
   2. **(E.g. Speed improvements and Learning New Motor skills via Few-Shot-Learning)**
6. **Challenges**

1. Bellec et al. n.d. [↑](#footnote-ref-1)
2. Maass n.d. [↑](#footnote-ref-2)
3. Anand 2018. [↑](#footnote-ref-3)
4. Wang et al. 2016. [↑](#footnote-ref-4)
5. Ibid. [↑](#footnote-ref-5)
6. Ibid. [↑](#footnote-ref-6)
7. Maass n.d. [↑](#footnote-ref-7)
8. Duan et al. 2016. [↑](#footnote-ref-8)
9. Ibid. [↑](#footnote-ref-9)
10. Maass n.d. [↑](#footnote-ref-10)
11. Bellec et al. 2018. [↑](#footnote-ref-11)
12. Bellec et al. n.d. [↑](#footnote-ref-12)