Embodiment of Learning in Electro-Optical Signal Processors. Supplementary Material.

Michiel Hermans, Piotr Antonik, Marc Haelterman, Serge Massar

August 22, 2016

1 Experimental setup

The experimental setup depicted in Fig. 1 of the main text uses the following components:

- Superluminescent diode (SLED): Thorlabs, model SLD1550P-A40), center wavelength 1550 nm, FWHM 33 nm.
- Dual input/dual output Mach Zehnder modulators (MZM): EOspace, model number AX-2x2-0MSS-12-PFA-PFA.
- Programmable optical attenuator: Agilent, model 81571A.
- Photodiodes: Terahertz Technologies, model TIA-525.
- Pulse amplifiers: Mini-Circuits, model ZHL-32A+.
- FPGA, ADC, and DAC: 4DSP FMC151 daughter card containing a two-channel DAC and ADC, controlled by a Xilinx Virtex 6 FPGA chip.

Note that the FPGA simultaneously generates the voltage signal that represents $z(t)$ and records the voltage signal representing $a(t)$. The FPGA also performs a minimal signal processing step by selecting and averaging over the middle samples of each masking step (see Section 4.1 for more details). The remaining processing steps are carried out on a PC.

Sending and receiving data to and from the FPGA is currently the main speed bottleneck of the experiment. Even though a single training iteration lasts only about 0.6 seconds for the NARMA10 and VARDEL5 task, most of this time is spent on the communication overhead with the PC (buffering). If the entire experiment were to be performed on the FPGA (which is feasible), a single training iteration would take of the order of milliseconds.

2 Online multiplication using cascaded MZMs

As mentionned in the main text, we use two back to back dual input/dual output Mach-Zehnder modulators for implementation of both Eqs. (2) and (10) from the main text using the same setup. The main fact we use is that the spectrum of the SLD is narrow enough to allow for a large extinction ratio by the MZMs, but is broad enough that the light in the two branches entering MZM2 from MZM1 can be considered incoherent. In the present experiment, the coherence length of the light from the SLD is of the order of a few hundreds of micrometers, which means that a very small difference in path length for the connections in between MZM1 and MZM2 is sufficient to make the two signals incoherent.

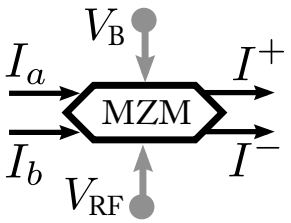


Figure 1: Schematic representation of a dual input / dual output Mach Zehnder modulator. The MZM is driven by the sum of two input voltages: one constant bias voltage V_{B} and a fast signal V_{RF}.

Consider the operation of a single MZM, schematised in Fig. 1. The intensities of the incoming light sources are denoted by I_a and I_b, and the MZM is driven by a voltage V, which is the sum of a constant bias voltage V_{B} and a fast voltage signal V_{RF}. The bias voltage was omitted in the main text to avoid confusion. Taking into account the incoherence between the two input signals, the intensities of the output branches (I^+ and I^-) are given by:

$$\begin{aligned} I^+ &= I_a \frac{1+\sin(V/V_0)}{2} + I_b \frac{1-\sin(V/V_0)}{2}, \\ I^- &= I_a \frac{1-\sin(V/V_0)}{2} + I_b \frac{1+\sin(V/V_0)}{2}, \end{aligned} \qquad (1)$$

with V_0 a constant depending on the MZM.

It is now easy to model the output of the two cascaded MZMs. Suppose the source has an intensity I_0, and no light enters the second input of MZM1. And suppose MZM1 and MZM2 receive voltages V_1 and V_2, respectively. The output intensities I_1^+ and I_1^- of MZM1 are given by

$$\begin{aligned} I_1^+ &= I_0 \frac{1+\sin(V_1/V_0)}{2}, \\ I_1^- &= I_0 \frac{1-\sin(V_1/V_0)}{2}. \end{aligned} \qquad (2)$$

The intensity I_2^+ at the first output branch of MZM2 is then:

$$I_2^+ = I_0 \frac{(1+\sin(V_1/V_0))(1+\sin(V_2/V_0))}{4} \quad (3)$$

$$+I_0 \frac{(1-\sin(V_1/V_0))(1-\sin(V_2/V_0))}{4} \quad (4)$$

$$= \frac{I_0}{2}\left[1+\sin(V_1/V_0)\sin(V_2/V_0)\right]. \quad (5)$$

In the experiment MZM1 receives a constant bias signal on top of an RF driving signal, such that $V_1/V_0 = \pi/2 + V_1'/V_0$, with V_1' the RF signal. We can thus write:

$$I_2^+ = \frac{I_0}{2}\left[1+\cos(V_1'/V_0)\sin(V_2/V_0)\right].$$

We use the setup in two modes. In the forward mode, $V_1' = 0$, so that the cascaded MZMs behave as:

$$I_2^+ = \frac{I_0}{2}\left[1+\sin(V_2/V_0)\right],$$

i.e., the transfer function acts as a sinusoidal function for the input argument V_2/V_0, which is equal to the sum of the input signal $z(t)$ and the system state $a(t)$. Note that a constant offset $I_0/2$ is added to the output. We use, however, amplifiers with a high-pass filter to drive the MZMs, which remove the DC offset. Therefore, once the loop is closed, this constant bias is removed, and we effectively end up with Eq. (2) in the main text.

In the backwards mode, we drive MZM1 with a voltage V_1' proportional to $a(q-D) + z(q-D)$. MZM2 is driven with a signal proportional to $\bar{e}(q) + e(q)$, but scaled down sufficiently such that $\sin(V_2/V_0) \approx V_2/V_0 = \bar{e}(q) + e(q)$. This means that in the backwards mode we can write:

$$I_2^+ = \frac{I_0}{2}\left[1+\cos\left(a(q+D)+z(q+D)\right)\left(\bar{e}(q)+e(q)\right)\right],$$

which is (up to the constant bias, and the factor μ which is imposed later by the optical attenuator) the desired functionality for the adjoint system (see Eq. (10) in the main text).

3 Derivation of gradients and adjoint system

3.1 Setting up the problem

We wish to find the gradient of a cost function C w.r.t. the parameters that can be optimised. In order to achieve this we have to use the chain rule through all the dependencies that describe the system. We will then obtain the backward equations given in the main text. Figure 2 gives a schematic of how the forward and backward equations must be implemented experimentally. Figure 3 depicts the information flow in the forward and backward systems.

We first recall the relevant equations describing the forward system. The input signal $z(t)$, formed by concatenating the input masks weighted with the current input sample s_i can be rewritten as

$$z(t) = s_{\lceil t/T \rceil} m(t \bmod T) + m_b(t \bmod T), \quad (6)$$

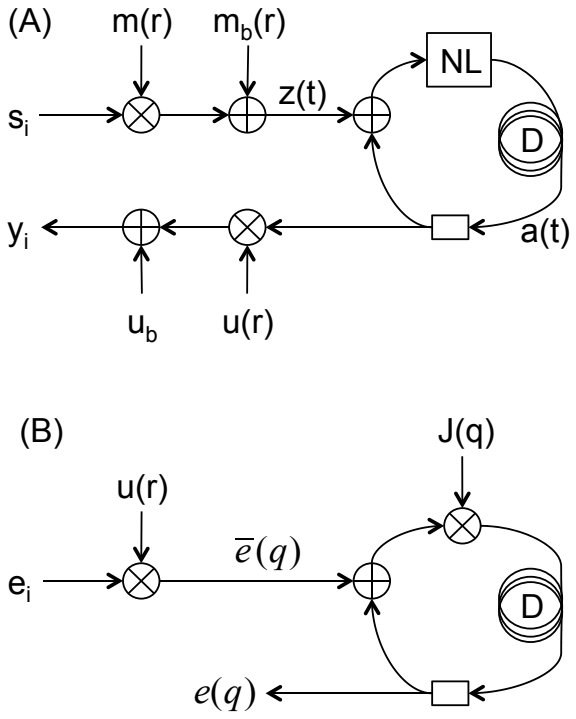


Figure 2: **A**: Schematic depiction of the forward system, as given in the main text and in Supplementary Material Eqs. (6, 7, 8). **B**: Schematic depiction of the backward system, as given in the main text and in Supplementary Material Eqs. (12, 17), where q is the backwards time.

where $\lceil . \rceil$ indicates the ceiling function, so that $\lceil t/T \rceil = i$ gives the index of s_i corresponding to the time t. We use the modulo operation in the argument of the input masks to indicate that the masks are repeated over time. Next we write down the expression for the reservoir state $a(t)$:

$$a(t+D) = \mu \sin\left(a(t) + z(t)\right). \quad (7)$$

Finally, we can write the formula for the output instances y_i as follows:

$$y_i = u_b + \int_0^T dr\; u(r) a_i(r), \quad (8)$$

with $a_i(r) = a(r+(i-1)T)$, the i-th segment of the recording of $a(t)$.

In what follows, for the sake of generality and of simplicity of notation, we take the input and output masks to be continuous functions of time. We denote functional derivatives with respect to time dependent functions as ordinary derivatives. The case, relevant to practical implementations, in which the masks depend on a finite number of parameters, is discussed in section 4.1. For simplicity in the derivations we will assume, unless indicated otherwise, that all variables, both in continuous time t and discrete time i, are defined for i and t going from $-\infty$ to ∞. If we have a specific finite input sequence s_i with $i \in \{1, \cdots, L\}$, we simply extend this beyond these bounds assuming that all extra s_i are equal to zero. Similarly, we assume that $z(t)$ is zero if

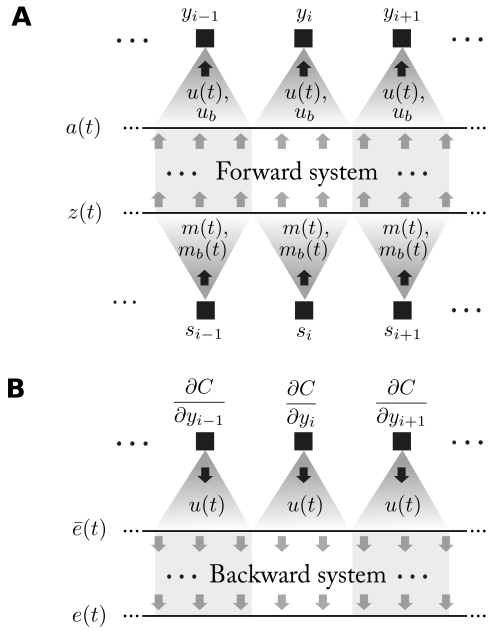


Figure 3: **A**: Schematic depiction of information flow when the system is used in the forward direction. On the bottom, the input sequence s_i is converted to a continuous-time signal $z(t)$ (with time running from left to right). Each instance in the sequence is multiplied with the finite-length masking signal $m(t)$ and added to $m_b(t)$. These sequences are then concatenated in time to form $z(t)$, the input to the forward system. The output $a(t)$ of the forward system is then converted into an output sequence y_i by segmenting $a(t)$ in time, and multiplying the segments with the output masks $u(t)$, and integrating over each of them. **B**: Schematic depiction of the information flow in the "backwards" mode. The derivatives $\partial C/\partial y_i$ are used as an input sequence. They are multiplied by $u(t)$ which now plays the role of input mask. This yields the signal $\bar{e}(t)$ that serves as input for the backward system. The output of the backward system is $e(t)$.

$\lceil t/T \rceil \notin \{1, \cdots, L\}$. Subsequently, if we sum or integrate over i or t without indicating limits, this indicates a summation or integration from $-\infty$ to ∞. In practice it turns out that if we only have a finite sequence, we only need to compute states over its corresponding time span. Similarly, when performing backpropagation, we only need to compute backwards over the same time span. All states outside of this interval do not influence the gradient computation, which means there are no problems in considering only finite intervals. This matters as in realistic training scenarios we typically train on relatively short sequences (in the case of the present paper of length 100).

3.2 Output mask gradient

For the output masks we can write

$$\frac{dC}{du(r)} = \sum_i \frac{\partial C}{\partial y_i} \frac{dy_i}{du(r)}. \tag{9}$$

For example, if the cost function we wish to minimise is the squared error over the interval of the input sequence:

$$C = \sum_{i=1}^{L} (y_i - y_i^*)^2,$$

$$e_i = \frac{\partial C}{\partial y_i} = 2(y_i - y_i^*) \text{ for } i \in \{1, \cdots, L\}.$$

$$\frac{\partial C}{\partial y_i} = 0 \text{ for } i \notin \{1, \cdots, L\}.$$

The second factor in Eq. (9) we can get from Equation 8:

$$\frac{dy_i}{du(r)} = a_i(r),$$

such that the gradient for the output mask $u(t)$ is simply given by

$$\frac{dC}{du(r)} = \sum_i \frac{\partial C}{\partial y_i} a_i(r),$$

or, given the fact that $\partial C/\partial y_i = 0$ outside the interval in which the sequence is defined:

$$\frac{dC}{du(r)} = \sum_{i=1}^{L} \frac{\partial C}{\partial y_i} a_i(r). \tag{10}$$

Similarly we find that

$$\frac{dC}{du_b} = \sum_{i=1}^{L} \frac{\partial C}{\partial y_i}.$$

3.3 Input mask gradient

The case of the input masks is more involved. Working out the chain rule we find:

$$\begin{aligned} \frac{dC}{dm(r)} &= \sum_i \frac{\partial C}{\partial y_i} \frac{dy_i}{dm(r)}. \\ &= \sum_i \frac{\partial C}{\partial y_i} \int dt' \, \frac{\partial y_i}{\partial a(t')} \frac{da(t')}{dm(r)}. \\ &= \int dt' \, \bar{e}(t') \frac{da(t')}{dm(r)}, \end{aligned} \tag{11}$$

where we have used

$$\bar{e}(t') = \frac{\partial C}{\partial a(t')} = \sum_i \frac{\partial C}{\partial y_i} \frac{\partial y_i}{\partial a(t')} \ .$$

From Equation 8 we can obtain (using a modulo function in the argument of $u(r)$):

$$\frac{\partial y_i}{\partial a(t')} = \delta_{i,\lceil t'/T \rceil} u(t' \bmod T),$$

i.e., equal to zero when t' did not fall in the segment of time used to produce y_i, and equal to the output mask otherwise. This yields:

$$\begin{aligned} \bar{e}(t') &= u(t' \bmod T) \frac{\partial C}{\partial y_{\lceil t'/T \rceil}} \\ &= u(r) e_i \end{aligned} \tag{12}$$

where $r = t' \bmod T$ and $i = \lceil t'/T \rceil$. In other words, $\bar{e}(t)$ is produced by masking the sequence $\partial C/\partial y_i$ with the output mask $u(r)$.

The second factor in Equation 11 we work out as follows. Using the chain rule we get

$$\frac{da(t')}{dm(t)} = \int dt''\ \frac{da(t')}{dz(t'')}\frac{dz(t'')}{dm(t)}. \tag{13}$$

and

$$\frac{da(t')}{dz(t'')} = \frac{\partial a(t')}{\partial z(t'')} + \int dt'''\ \frac{\partial a(t')}{\partial a(t''')}\frac{da(t''')}{dz(t'')}.$$

From equation 7 we obtain the partial derivatives:

$$\begin{aligned}\frac{\partial a(t')}{\partial z(t'')} &= \frac{\partial a(t')}{\partial a(t'')} \\ &= \mu\delta(t'-t''-D)\cos(a(t'-D)+z(t'-D)),\end{aligned}$$

Or, more compactly:

$$\frac{\partial a(t')}{\partial z(t'')} = \delta(t'-t''-D)J(t'),$$

with

$$J(t') = \mu\cos(a(t'-D)+z(t'-D)).$$

This yields

$$\frac{da(t')}{dz(t'')} = J(t')\left[\delta(t'-t''-D) + \frac{da(t'-D)}{dz(t'')}\right].$$

By filling in the expression for $da(t'-D)/dz(t'')$ recursively we can write this as:

$$\frac{da(t')}{dz(t'')} = \sum_{i=0}^{\infty}\left[\delta(t'-t''-iD)\prod_{j=0}^{i-1}J(t'-jD)\right]. \tag{14}$$

By filling in Equation 14 in Equation 13, and inserting the result in Equation 11 we obtain:

$$\frac{dC}{dm(r)} = \int dt'\ dt''\ \bar{e}(t')\sum_{i=0}^{\infty}\delta(t'-t''-iD)\prod_{j=0}^{i-1}J(t'-jD)\frac{dz(t'')}{dm(r)}. \tag{15}$$

We can solve the integral over t' explicitly. We denote this by $e(t'')$:

$$\begin{aligned}e(t'') &= \int dt'\ \bar{e}(t')\sum_{i=0}^{\infty}\delta(t'-t''-iD)\prod_{j=0}^{i-1}J(t'-jD) \\ &= \sum_{i=0}^{\infty}\bar{e}(t''+iD)\prod_{j=0}^{i-1}J(t''+(i-j)D) \\ &= \sum_{i=0}^{\infty}\bar{e}(t''+iD)\prod_{j=1}^{i}J(t''+jD).\end{aligned} \tag{16}$$

It's straightforward to prove that $e(t)$ is equal to the expression as presented in the main text (with arguments shifted by D):

$$e(t) = J(t+D)(e(t+D)+\bar{e}(t+D)). \tag{17}$$

Indeed, if we recursively fill in the expression for $e(t+D)$ in Eq. 17, we obtain Eq. 16. Using this we can reduce Equation 15 to

$$\frac{dC}{dm(r)} = \int dt''\ e(t'')\frac{dz(t'')}{dm(r)}.$$

From the expression of $z(t)$ we find that

$$\frac{dz(t'')}{dm(r)} = \delta(t'' \bmod T - r)s_{\lceil t/T\rceil}.$$

Inserting this we can find the final expression for the gradient for the input mask:

$$\frac{dC}{dm(r)} = \sum_i s_i e_i(r),$$

or, again using the fact that we defined $s_i = 0$ for $i \notin \{1,\cdots,L\}$:

$$\frac{dC}{dm(r)} = \sum_{i=1}^{L} s_i e_i(r), \tag{18}$$

with $e_i(r) = e(r-(i-1)T)$, the i-th segment of the time trace of $e(t)$. Similarly for $m_0(t)$ we can write:

$$\frac{dC}{dm_b(r)} = \sum_{i=1}^{L} e_i(r). \tag{19}$$

3.4 Multiple inputs/outputs

The above explanation is easily extended to multiple input and output dimensions. Suppose we have a multivariate time series $\mathbf{s}_i$, where the k-th element at time step i is denoted by $\mathbf{s}_i[k]$. We can then easily construct $z(t)$ by defining as many input masks $m_k(t)$ as there are input dimensions and adding them all up:

$$z(t) = \sum_k \mathbf{s}_{\lceil t/T\rceil}[k]m_k(t \bmod T) + m_b(t \bmod T),$$

The desired output can similarly exist of a multivariate time series with elements $\mathbf{y}_i^*[l]$. To produce an output $\mathbf{y}_i[l]$ we simply define an output mask $u_l(t)$ and bias u_l^0 for each output channel:

$$\mathbf{y}_i[l] = u_l^0 + \int_0^T dt u_l(r)a_i(r).$$

The same procedure can now be used to determine the gradients with respect to the multivariate input and output masks. We find:

$$\frac{dC}{du_l(r)} = \sum_{i=1}^{L}\frac{dC}{d\mathbf{y}_i[l]}a_i(r), \tag{20}$$

and

$$\frac{dC}{du_l^0} = \sum_{i=1}^{L}\frac{dC}{d\mathbf{y}_i[l]}. \tag{21}$$

The source of the BP equation is now

$$\bar{e}(t') = \sum_l u_l(t' \bmod T) \frac{dC}{dy_{\lceil t'/T \rceil}[l]} \ , \tag{22}$$

the recurrence for the error $e(t)$, eq. (17), is unchanged, and one has

$$\frac{dC}{dm_k(r)} = \sum_{i=1}^{L} \mathbf{s}_i[k] e_i(r) \ .$$

4 Implementation details

4.1 Mask parametrisation

While the aforementioned theory is generally valid for continuous-time signals, an experimental setup is limited by the finite bandwidth of the DAC/ADC, and the analog electronic parts. To make sure that these effects play a limited role, we parametrise the input and output masks as piecewise constant functions, which has been common practice for reservoirs of this type [3]. To this end we divide the delay D into an integer number N_D of equal time segments, called *masking steps*. Next we ensure that the masking period T has a total duration that is also contains an integer number N_T of masking steps, in our case one less than the delay: $N_T = N_D - 1$. This allows for the mixing of the states over time, as detailed in [3].

The input and output masks are picked to be constant for the duration of each masking step. This implies that $z(t)$ is piecewise constant. The fact that both T and D are an integer number of masking steps makes that changes in $a(t)$ only occur in between the masking steps, i.e., they are synchronised with the masking steps, and this is valid for the backwards pass too. In short, $a(t)$, $\bar{e}(t)$ and $e(t)$ are all piecewise constant signals, with values that remain constant during each masking step.

In practice this allows us to reduce effects of noise by averaging the signals representing $a(t)$ and $e(t)$ over several measuring samples during a single masking step. Typically we pick a set of samples from the middle of each masking step, and discard those at the beginning and the end as they may contain artefacts caused by the limited bandwidth of the ADC. More importantly, it allows us to make a discrete time approximation of the entire system. For example, let's consider equation 8. The mask $u(t)$ is made up of N_T constant segments of equal length, with values during the segments denoted u_k. Similarly, each segment $a_i(r) = a\left(t - (i-1)T\right)$ is piecewise constant, with values we can for example denote with a_k^i. The integral reduces to

$$y_i = u_b + \sum_{k=1}^{N_T} a_k^i u_k,$$

(where we absorbed the factor T that emerges from the integration into the values u_k). Each particular value a_k^i can be interpreted as the state of the k-th 'neuron' or 'node' state during the i-th instance of the input sequence. We can still use the expressions for the gradients in Equations 10, 18 and 19. Indeed, by construction, the gradient for the output mask $u(t)$ for the duration of a single masking step is a constant (as $a(t)$ remains constant over the segment). The same holds for the gradients for the input masks. This implies that $u(t)$ and $m(t)$ remain piecewise constant during training, and we can in practice describe them simply as lists of values instead of a continuous-time function.

Note that the choice of dealing with bandwidth limitations by using piecewise constant functions is not the only possible avenue. One alternative would be to impose bandwidth constraints on $m(t)$ and $u(t)$, such that the finite signal generator bandwidth and sampling rates form no obstacle in treating the setup as a continuous-time setup. We chose the piecewise-constant constraint as it is more directly related to existing implementations of delay-coupled electro-optical signal processors, and it allows to identify a specific number of 'virtual nodes' (the number of segments within the masking period T). In other words, the choice of N_T determines the 'complexity', or the number of degrees of freedom of the system.

4.2 Gradient descent

We used stochastic gradient descent to train the masks; each iteration we drew a 100 time step sequence to determine a gradient. This sequence was either generated on the fly (in the case of VARDEL5 and NARMA10), or drawn randomly from a training set (TIMIT). Note that as the BP equation is linear, we are in principle free to rescale $\bar{e}$ as we wish. In practice, in order to keep MZM2 in the linear regime, we scaled the input error signal $\bar{e}(t)$ by dividing it by its standard deviation and multiplying with a factor 0.1. The learning rate η we choose equal to 0.25 at the start of the training process, after which it drops linearly to zero throughout the course of the experiment. On top of that we use Nesterov momentum with a momentum factor 0.9 to speed up convergence [2, 4]. Nesterov momentum is a heuristic method that finds widespread use in speeding up convergence of stochastic gradient descent. The idea of momentum in gradient descent is to give parameter updates a certain inertia, meaning that previous parameter updates still count in the current one, which helps with overcoming local minima and speeds up convergence. Nesterov momentum is a simple variation of this principle, where the algorithm measures the gradient one update step ahead in order to change its momentum "ahead of time".

4.3 Robustness

Our work shows that physical BP is robust against imperfections of the physical setup, as illustrated by the following imperfections we were confronted with.

The first imperfection was the high-pass filtering operation of the amplifiers used to drive the MZMs, with a cut-off frequency of 20 kHz. While the high-pass filter is a desirable property (to get rid of voltage bias), this corresponds to a typical time scale of about 8 µs, which is about the same as the loop delay and therefore not negligible. The current experimental setup does not take this filtering operation into account explicitly.

A second imperfection was an imbalance in losses between the two fibres connecting MZM1 with MZM2. A third imperfection was that the system was not perfectly linear during the backwards pass, since MZM2 is never a perfectly linear system. There's also an important trade-off here. One can reduce the residual nonlinearity by reducing the amplitude of the incoming voltage signal that represents $\bar{e}(t)$. But in turn this also reduces the signal-to-noise ratio of the measurement during the backpropagation phase, such that one needs to find a good balance between these two effects.

All these effects are imperfections inherent to the physically implemented backpropagation phase, but both in simulation and in the actual experiments we found that they only had a very minor impact on the training process and the overall performance.

One parameter that turned out to be crucial was the bias voltage of MZM2. The reason is that even a small offset from an effectively zero level introduces a systematic error in the backpropagation process, such that the measured signal (denoted as $e_c(t)$ to indicate that it is corrupted) becomes :

$$e_c(t-D) = J(t)(e_c(t) + \bar{e}(t) + \widetilde{e}),$$

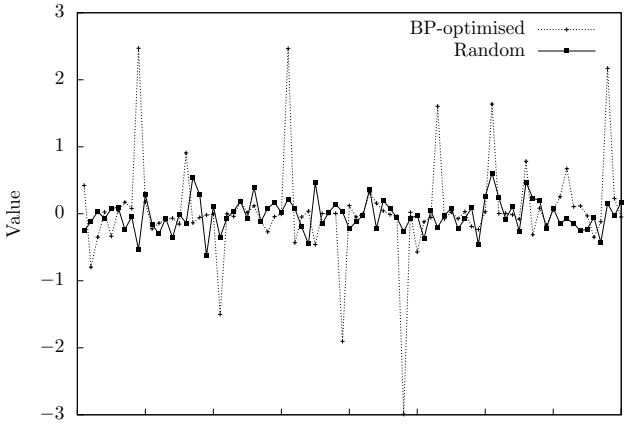
with $\widetilde{e}$ a constant offset caused by an incorrectly set voltage bias of MZM2. It turned out that, in the experiments, keeping this bias level effectively equal to zero was difficult; very slight drifts on the effective working point of the MZM occurred over the course of minutes/hours. Luckily, the backpropagation is a linear process. This means that we can recover $e(t)$ by performing a second measurement right after measuring $e_c(t)$:

$$e_r(t-D) = J(t)(e_r(t) + \widetilde{e}),$$

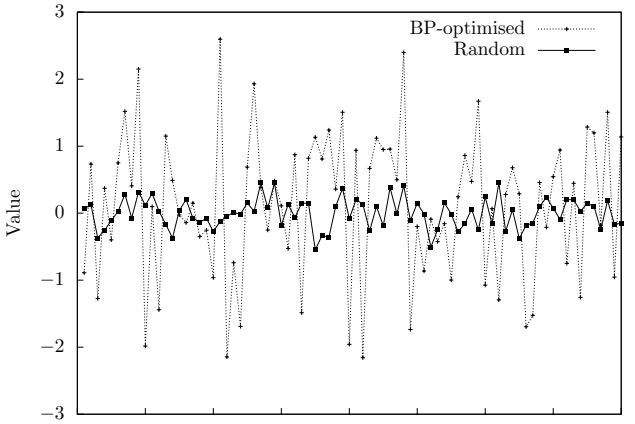
and

$$e(t) = e_c(t) - e_r(t).$$

In other words we simply need to perform two measurements after each other, where in the second one we send a 'zero' input error, and subtract this from the first measurement in order to remove the influence of the offset of MZM2. This turned out to solve the problem.



(a) Input mask m



(b) Bias input mask m_b

Figure 4: Comparison of BP-optimised input masks (dotted curves) and random RC masks (solid curves) for the VARDEL5 task, for $m(r)$ (top panel) and $m_b(r)$ (bottom panel).

5 Tasks

5.1 NARMA10 and VARDEL5

In the case of NARMA10 and VARDEL5 we divided T into 80 equal time intervals ($N_T = 80$), which allowed us to take 16 samples during each masking step, where we averaged over the middle 8 in order to get piecewise-constant values for $a(t)$ and $e(t)$. We chose the number of training iterations at 10,000, 20,000 for VARDEL5 and NARMA10, respectively, chosen heuristically as a trade-off between the time required for an experiment and the final performance. (A single iteration lasted approximately 0,6 s).

The cost functions used for NARMA10 and VARDEL5 are the aforementioned sums of squared errors. We repeated the training cycles 10 times, each time with different random input mask initialisations. Output masks were always

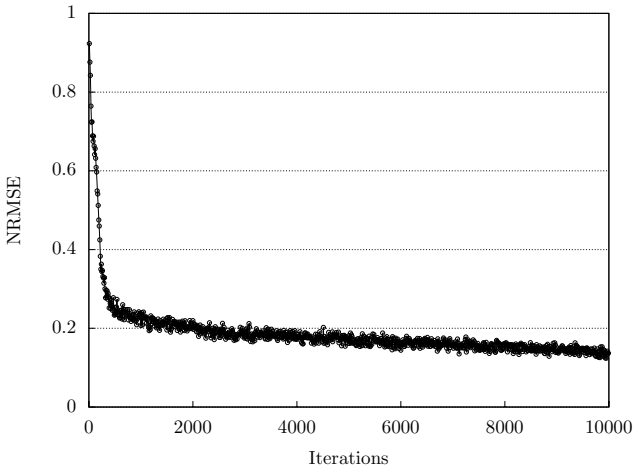


Figure 5: Evolution of the NRMSE during the training process on the VARDEL task. The error falls sharply during the first 300 iterations, and then converges slowly towards 0.15.

initialised at zero. For all backpropagation experiments we set the feedback parameter strength parameter μ effectively equal to one (such that the system is at the 'edge of stability'), which we found to give the best performance.

In Figure 4 we depict the masks $m(r)$ and $m_b(r)$ for the RC implementation (when they are chosen at random), and after optimisation using the BP algorithm, for the VARDEL5 task. One sees that the BP algorithm dramatically changes the input masks. In particular the mask m is very large at some specific values of r, and almost zero for other values. This suggests that in some sense what the optimised reservoir is doing is storing the value of the input on specific neurons, and then keeping it in memory for some time, before mixing it nonlinearly with the input several time steps in the future. In Figure 5 we depict how the NRMSE converges over time for the VARDEL5 task, as the BP algorithm slowly improves the input and output masks.

For the reservoir computing results we measured average performance as a function of three scaling parameters: the feedback strength parameter μ and the scaling of the input mask and bias mask. Once optimal parameters were determined we ensured that the output masks were trained on an unlimited amount of input training data (in practice we observed the test error for increasing amounts of training data, and stopped as soon as the performance no longer improved). This was to ensure that we have a fair comparison to the backpropagation setup, where we generate unlimited amounts of data too. Each experiment was is repeated 10 times, giving rise to the error bars in Figure 3A in the main text.

5.2 TIMIT

For the TIMIT task we used 1,000,000 training iterations. Because of this large number of iterations we only performed a single full training cycle.

Measurement noise plays a smaller role in a classification task such as this one, and we divided T into 200 masking steps, taking 8 samples in each and averaging over the middle 4, thereby increasing the number of virtual nodes N_T while taking into account the hardware constraints (sample rates of the DAC and ADC). Most likely this number can be increased further for example using only 4 samples per masking steps and averaging over the middle 2. In practice we are also limited by the relatively slow communication between the PC and the FPGA, and increasing N_T increases the amount of data that needs to be transferred, slowing down the experiment considerably. Currently, for 1,000,000 iterations the training took two weeks to complete.

We picked μ at a value slightly under one, but we found in simulations that performance did not strongly depend on it for a broad range of values.

In the case of TIMIT, the goal is to minimise a classification error rate, which is not directly differentiable. One possible strategy is simply to try and minimise the MSE between the output and the target labels (1 for the correct class, zero for all others). Classification would then be performed by the *winner-take-all* approach, where we simply select the output channel with the highest output as the 'winner'. In practice, using MSE for classification suffers from some drawbacks. Most importantly MSE will put a lot of emphasis on producing the exact target values (close to zero or one), while we are only interested in performance after selecting the highest output. A better approach is to use a softmax function at the output, which converts the output values into a set of probabilities, and minimise the cross-entropy with the target probabilities (again, 1 for the correct class and zero for all others) . Details on this strategy can be found for example in [1]. In practice the conversion of the output $\mathbf{y}_i[k]$ into probabilities is performed using the so-called *softmax* function:

$$\mathbf{p}_i[k] = \frac{\exp(\mathbf{y}_i[k])}{\sum_l \exp(\mathbf{y}_i[l])},$$

Tthe cost function is the cross-entropy:

$$C = -\sum_{i=1}^{L} \sum_k \mathbf{t}_i[k] \ln \mathbf{p}_i[k],$$

where we denote the target outputs as $\mathbf{t}_i[k]$. It can then be shown that

$$\frac{dC}{d\mathbf{y}_i[k]} = \mathbf{p}_i[k] - \mathbf{t}_i[k],$$

(and again zero if $i \notin \{1, \cdots, L\}$) This means that the error we have at the output takes on virtually the same form as before, only this time there is the intermediary step of the softmax function. Gradients for the output masks are almost the same as before, except for equations 20 and 21 where we use $\mathbf{p}_i[k] - \mathbf{t}_i[k]$ instead of $\mathbf{y}_i[k] - \mathbf{y}_i^*[k]$. As far as the rest of the BP algorithm goes, we now simply have to mask these 'output errors' to produce $\bar{e}(q)$, and the rest plays out exactly as before.

For the RC approach, optimising the parameters (input scaling, bias scaling and feedback gain) on the hardware would be too costly in terms of time. Therefore we optimised them on a PC using a simulation of the physical

setup. Once we decided on the parameters, we ran all the TIMIT data through the physical setup and recorded all the responses. Next we trained output weights, again using gradient descent with the above cross-entropy loss.

References

[1] Christopher M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, August 2006.

[2] Yurii Nesterov. *A method of solving a convex programming problem with convergence rate* $o(1/k^2)$. Soviet Mathematics Doklady, 27(2) : 372–376, 1983.

[3] Yvan Paquot, Francois Duport, Antoneo Smerieri, Joni Dambre, Benjamin Schrauwen, Marc Haelterman, and Serge Massar. *Optoelectronic reservoir computing*. Scientific Reports, 2: 1–6, 2012.

[4] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. *On the importance of initialization and momentum in deep learning*. In Proceedings of the 30th in- ternational conference on machine learning (ICML-13), pages 1139–1147, 2013.