A Proofs

A.1 Kernels & their linearizations for temporal encoders in Mnemosyne

We tested different transformations ϕ and discovered that those leading to most accurate approximation of the softmax-kernel lead to most effective memory mechanisms for Mnemosyne's temporal encoders (see: Sec. 3.3). Our starting variant is the so-called *FAVOR+* mechansim from [16], given as follows for $\Gamma(\mathbf{z}, r) \overset{\mathrm{def}}{=} \frac{1}{\sqrt{r}} \exp(-\frac{\|\mathbf{z}\|^2}{2})$ and $\omega_1, ..., \omega_r \sim \mathcal{N}(0, \mathbf{I}_N)$:

$$\phi_{F+}(\mathbf{z}) = \Gamma(\mathbf{z}, r) \left(\exp(\omega_1^\top \mathbf{z}), ..., \exp(\omega_r^\top \mathbf{z})\right)^\top \tag{9}$$

Random vectors $\omega_1, ..., \omega_r$ form a block-orthogonal ensemble (see: [16]). We applied also its improvement relying on the so-called *hyperbolic cosine random features*, where $\prod$ is the concatenation operator:

$$\phi_{HF+}(\mathbf{z}) = \Gamma(\mathbf{z}, r) \prod_{i=1}^{\frac{r}{2}} (\exp(\omega_i^\top \mathbf{z}), \exp(-\omega_i^\top \mathbf{z}))^\top \tag{10}$$

Both randomized transformations provide **unbiased** estimation of the softmax-kernel, yet the latter one (that can be cast as modified ϕ_{F+} via the antithetic Monte Carlo trick) has provably lower approximation variance.

A.1.1 The curious case of linearization with bounded features

The last variant for the efficient estimation of the softmax-kernel we applied, is a very recent mechanism *FAVOR++* from [39], given as:

$$\phi_{F++}(\mathbf{z}) = \frac{D}{\sqrt{r}} \prod_{i=1}^{r} \exp(-\widehat{A}\|\omega_i\|_2^2 + B\omega_i^\top \mathbf{z} + C\|\mathbf{z}\|^2)^\top,$$

where we have: $\widehat{A} = -A$, $B = \sqrt{1 + 4\widehat{A}}$, $C = -\frac{1}{2}$, $D = (1 + 4\widehat{A})^{\frac{N}{4}}$, $A = 1 - \frac{1}{\rho}$ and $\rho \in (0, 1)$ is a free parameter. As opposed to the previous variants, mechanism $\phi_{F++}(\mathbf{z})$ provides an estimation via **bounded** random variables (since $\widehat{A} > 0$), leading to stronger concentration results (beyond second moment) and still unbiased approximation.

The optimal choice of ρ depends on the kernel inputs. The formula for ρ optimizing the variance of the kernel matrix estimation $\mathcal{K} = [\mathrm{K}(\mathbf{q}^i, \mathbf{k}^j)]_{i,j=1,...,M}$ induced by the softmax-kernel K (in the bi-directional case) is not tractable. However choosing ρ by optimizing certain derivative of the variance-objective was showed to work well in several applications [39]:

$$\rho^* = \frac{\sqrt{(2\gamma + N)^2 + 8N\gamma} - 2\gamma - N}{4\gamma} \tag{11}$$

for $\gamma = \frac{1}{M^2} \sum_{i=1}^{M} \sum_{j=1}^{M} \|\mathbf{q}^i + \mathbf{k}^j\|^2$. Since γ can be rewritten as: $\gamma = \frac{1}{M^2}(\sum_{i=1}^{M} \|\mathbf{q}^i\|_2^2 + \sum_{j=1}^{M} \|\mathbf{k}^j\|_2^2 + 2\mathbf{q}^\top \mathbf{k})$ for $\mathbf{q} = \sum_{i=1}^{M} \mathbf{q}^i$ and $\mathbf{k} = \sum_{j=1}^{M} \mathbf{k}^j$, computing ρ^* in the bi-directional setting can be clearly done in time linear in M as a **one-time** procedure. Then the computation of $\mathbf{h}_{\mathrm{Mne}}(M)$ follows. Compute-time per memory-vector remains $O_M(1)$.

However Mnemosyne's temporal encoder applied uni-directional attention. In the uni-directional case, we have: $\gamma_t = \frac{1}{t} \sum_{j=1}^{t} \|\mathbf{q}^t + \mathbf{k}^j\|^2 = \frac{1}{t}(\|\mathbf{q}^t\|_2^2 + \sum_{j=1}^{t} \|\mathbf{k}^j\|_2^2 + 2(\mathbf{q}^t)^\top \mathbf{k}(t))$, where $\mathbf{k}(t) = \sum_{j=1}^{t} \mathbf{k}^j$ for $t = 1, ..., M$. Instead of one γ, we now have M values γ_t since not all memories are known at once. We can still achieve $O_1(M)$ compute-time per γ_t, repeating the trick from the bi-directional case, but that would need to be followed by the re-computation of $\phi(\mathbf{k}^\mu)$ (with new ρ-parameter) for $\mu = 1, ..., t$ which of course is not possible since vectors $\{\mathbf{k}\}_{\mu=1}^{t}$ are not explicitly stored (and for a good reason - computational benefits), see: Eq. 3.

Thickening Mnemosyne's memory: To obtain efficient uni-directional Mnemosyne's memory cell also for the ϕ_{F++}-mechanism, we propose to "thicken" in that setting the hidden state from Eq. 3, replacing $\mathbf{h}_{\mathrm{Mne}}(t) = (\mathbf{N}_t, \Psi_t)$ with $\mathbf{H}_{\mathrm{Mne}}(t) = (\{\mathbf{N}_t^\rho\}_{\rho \in \Omega}, \{\Psi_t^\rho\}_{\rho \in \Omega}, \Sigma_t, \Lambda_t)$, where we have: $\Sigma_t = \sum_{j=1}^{t} \mathbf{k}^j$, $\Lambda_t = \sum_{j=1}^{t} \|\mathbf{k}^j\|_2^2$ and furthermore: $\mathbf{N}_t^\rho$, Ψ_t^ρ correspond to versions of $\mathbf{N}_t$ and Ψ_t

respectively, using parameter ρ to define mapping ϕ. The set Ω is obtained by discretizing interval $(0, 1)$ into a fixed number of chunks c (and effectively quantizes $\rho \in (0, 1)$). The strategy is now clear: when the new pattern comes, we first update the entire thickened state, and then compute ρ^*. We finalize by finding $\rho \in \Omega$ closest to ρ^* to transform an input and using for that the "slice" of the hidden state corresponding to ρ. We see that all these operations can be made efficiently with only c-multiplicative term (**independent** from the number of patterns M) in space and time complexity.

FAVOR++ mechanism, as FAVOR+, can also be adapted to its hyperbolic cosine variant. In practice FAVOR+ mechanism worked similarly to FAVOR++, yet the proper adaptation of the latter one was important, since (see: Sec. 4), this variant provides strongest theoretical guarantees for the capacity of the entire compact associative memory model.

A.2 The proof of the extended version of Theorem 4.3

We start by providing an extended version of Theorem 4.3, enriched with the exact formula of the variance of $\Delta(E_{\mathrm{rand}})$. We prove it below. We borrow the notation from Sec. A.1.

Theorem A.1 (storage of compact associative memories)**.** *Denote by $\xi^1, ..., \xi^M \in \{-1, +1\}^N$ the memory-vectors. Assume that the Hamming distance between any two memory-vectors is at least τN for some $\tau > 0$. Take some $0 < \rho < \frac{\tau}{2}$. Then the following is true for any memory-vector ξ^l for $l = 1, ..., \mu$ and any input $\widehat{\xi^l} \in \mathcal{B}(\xi^l, \rho N)$ as long as $M \leq \exp(2N(\tau - 2\rho))\frac{1-e^{-2}}{2e^2}$: the expected change of the energy of the compact associative memory system $\Delta(E_{\mathrm{rand}})$ associated with flipping the value of the dimension of $\widehat{\xi^l}$ is positive if that operation increases the distance from its close neighbor ξ^l and is negative otherwise. Furthermore, the variance of $\Delta(E_{\mathrm{rand}})$ is of the form:*

$$\mathrm{Var}(\Delta(E_{\mathrm{rand}})) = \frac{1}{r}(V_1 + V_2 - 2V_3 - V_4 - V_5 + 2V_6) \tag{12}$$

where:

$$V_1 = \sum_{\mu_1, \mu_2 \in \{1,...,M\}} \Psi(\xi^{\mu_1} + \xi^{\mu_2} + 2\widehat{\xi^l}), \quad V_2 = \sum_{\mu_1, \mu_2 \in \{1,...,M\}} \Psi(\xi^{\mu_1} + \xi^{\mu_2} + 2\tilde{\xi}^l)$$
$$V_3 = \sum_{\mu_1, \mu_2 \in \{1,...,M\}} \Psi(\xi^{\mu_1} + \xi^{\mu_2} + \widehat{\xi^l} + \tilde{\xi}^l) \quad V_4 = \sum_{\mu_1, \mu_2 \in \{1,...,M\}} \exp((\xi^{\mu_1})^\top \widehat{\xi^l}) \exp((\xi^{\mu_2})^\top \widehat{\xi^l})$$
$$V_5 = \sum_{\mu_1, \mu_2 \in \{1,...,M\}} \exp((\xi^{\mu_1})^\top \tilde{\xi}^l) \exp((\xi^{\mu_2})^\top \tilde{\xi}^l) \quad V_6 = \sum_{\mu_1, \mu_2 \in \{1,...,M\}} \exp((\xi^{\mu_1})^\top \widehat{\xi^l}) \exp((\xi^{\mu_2})^\top \tilde{\xi}^l) \tag{13}$$

for $\tilde{\xi}^l$ denoting $\widehat{\xi^l}$ with one of its dimensions flipped and:

$$\Psi(\mathbf{x}) \stackrel{\mathrm{def}}{=} D^4 \exp(-2N)(1 + 8\widehat{A})^{-\frac{N}{2}} \exp\left(\frac{B^2}{2(1 - 8\widehat{A})}\|\mathbf{x}\|^2\right) \tag{14}$$

Proof. Take a memory $\xi^l \in \{-1, +1\}^N$ and an input $\widehat{\xi^l} \in \mathcal{B}(\xi^l, \rho N)$. Denote by $\mathrm{neg}(\widehat{\xi^l}, i)$ a vector obtained from $\widehat{\xi^l}$ by replacing $\widehat{\xi^l}(i)$ with $-\widehat{\xi^l}(i)$. Let us study the change of the energy of the system as we flip the value of the ith dimension of the input $\widehat{\xi^l}$ since the sign of this change solely determines the update that will be made. We have the following:

$$\Delta(E_{\mathrm{rand}}) = E(\mathrm{neg}(\widehat{\xi^l}, i); \xi^1, ..., \xi^M) - E(\widehat{\xi^i}; \xi^1, ..., \xi^M) = E_{\mathrm{signal}} + E_{\mathrm{noise}}, \tag{15}$$

where:

$$E_{\mathrm{signal}} = \frac{1}{r}\sum_{k=1}^{r}(W_k^l - Z_k^l), \tag{16}$$

$$E_{\mathrm{noise}} = \frac{1}{r}\sum_{k=1}^{r}\sum_{\mu \in \{1,...,M\} \backslash \{l\}} (W_k^\mu - Z_k^\mu), \tag{17}$$

and furthermore: $W_k^i = a_k^i b_k$, $Z_k^i = a_k^i c_k$ for:

$$\begin{aligned} a_k^i &= D\exp(-\frac{N}{2})\exp(B\omega_k^\top \xi^i - \widehat{A}\|\omega_k\|_2^2), \\ b_k &= D\exp(-\frac{N}{2})\exp(B\omega_k^\top \widehat{\xi}^l - \widehat{A}\|\omega_k\|_2^2), \\ c_k &= D\exp(-\frac{N}{2})\exp(B\omega_k^\top \mathrm{neg}(\widehat{\xi}^l, i) - \widehat{A}\|\omega_k\|_2^2). \end{aligned} \tag{18}$$

If $\omega_1, ..., \omega_r \sim \mathcal{N}(0, \mathbf{I}_N)$ then, from the fact that E_{rand} is the unbiased estimation of E_{reg}, we get:

$$\begin{aligned} \mathbb{E}[X_k] &= \exp((\xi^l)^\top \widehat{\xi}^l), \\ \mathbb{E}[Y_k] &= \exp((\xi^l)^\top \mathrm{neg}(\widehat{\xi}^l, i)), \\ \mathbb{E}[W_k^\mu] &= \exp((\xi^\mu)^\top \widehat{\xi}^l), \\ \mathbb{E}[Z_k^\mu] &= \exp((\xi^\mu)^\top \mathrm{neg}(\widehat{\xi}^l, i)), \end{aligned} \tag{19}$$

This is a direct consequence of the OPRF-mechanism introduced in [39]. Variables: X_k, Y_k, W_k^μ and Z_k^μ for $\mu = 1, ..., M$ are simply unbiased estimators of the softmax-kernel values obtained via applying OPRF-mechanism. Let us now compute the expected change of the energy of the system:

$$\mathbb{E}[\Delta(E_{\mathrm{rand}})] = \mathbb{E}[E_{\mathrm{signal}}] + \mathbb{E}[E_{\mathrm{noise}}], \tag{20}$$

where:

$$\mathbb{E}[E_{\mathrm{signal}}] = \frac{1}{r}\sum_{k=1}^{r}(\mathbb{E}[X_k] - \mathbb{E}[Y_k]) = \frac{1}{r}\sum_{k=1}^{r}\Big(\exp((\xi^l)^\top \widehat{\xi}^l) - \exp((\xi^l)^\top \mathrm{neg}(\widehat{\xi}^l, i))\Big) \tag{21}$$

and

$$\begin{aligned} \mathbb{E}[E_{\mathrm{noise}}] = \frac{1}{r}\sum_{k=1}^{r}\sum_{\mu\in\{1,...,M\}\setminus\{l\}}(\mathbb{E}[W_k^\mu] - \mathbb{E}[Z_k^\mu]) = \\ \frac{1}{r}\sum_{k=1}^{r}\sum_{\mu\in\{1,...,M\}\setminus\{l\}}\Big(\exp((\xi^\mu)^\top \widehat{\xi}^l) - \exp((\xi^\mu)^\top \mathrm{neg}(\widehat{\xi}^l, i))\Big) \end{aligned} \tag{22}$$

We will first upper bound $|\mathbb{E}[E_{\mathrm{noise}}]|$. We have:

$$\begin{aligned} |\mathbb{E}[E_{\mathrm{noise}}]| &\leq \frac{1}{r}\sum_{k=1}^{r}\sum_{\mu\in\{1,...,M\}\setminus\{l\}}\Big(\exp((\xi^\mu)^\top \widehat{\xi}^l) + \exp((\xi^\mu)^\top \mathrm{neg}(\widehat{\xi}^l, i))\Big) \\ &\leq \sum_{k=1}^{r}\sum_{\mu\in\{1,...,M\}\setminus\{l\}}(\exp(N(1-2(\tau-\rho))) + \exp(N(1-2(\tau-\rho)+\frac{2}{N}))) \\ &\leq 2M\exp(N(1-2(\tau-\rho)+\frac{2}{N})) \end{aligned} \tag{23}$$

We will now consider two cases:

Case 1: $\widehat{\xi}^l(i) = \xi^l(i)$:

In this setting, flipping the value of the ith dimension of the input vector increases its distance from the close neighbor. Therefore in this case we would like the energy change of the system to be positive (so that the flip does not occur). From the Equation 21, we obtain:

$$\begin{aligned} \mathbb{E}[E_{\mathrm{signal}}] \geq \frac{1}{r}\sum_{k=1}^{r}(\exp(N(1-2\rho)) - \exp(N(1-2\rho)-2))) = \\ \exp(N(1-2\rho))(1-e^{-2}) \end{aligned} \tag{24}$$

Thus we obtain:

$$\mathbb{E}[\Delta(E_{\text{rand}})] \geq \exp(N(1-2\rho))(1-e^{-2}) - 2M\exp(N(1-2(\tau-\rho)+\frac{2}{N})) \tag{25}$$

Therefore, if the following holds:

$$M \leq \exp(2N(\tau-2\rho))\frac{1-e^{-2}}{2e^2}, \tag{26}$$

then $\mathbb{E}[\Delta(E_{\text{rand}})] > 0$.

Case 2: $\widehat{\xi^l}(i) = -\xi^l(i)$:

In this setting, flipping the value of the ith dimension of the input vector decreases its distance from the close neighbor. Therefore in this case we would like the energy change of the system to be negative (so that the flip does not occur). From the Equation 21, we obtain:

$$\begin{aligned}\mathbb{E}[E_{\text{signal}}] \leq \frac{1}{r}\sum_{k=1}^{r}(\exp(N(1-2\rho)) - \exp(N(1-2\rho)+2))) = \\ \exp(N(1-2\rho))(1-e^2)\end{aligned} \tag{27}$$

Thus we obtain:

$$\mathbb{E}[\Delta(E_{\text{rand}})] \leq \exp(N(1-2\rho))(1-e^{2}) + 2M\exp(N(1-2(\tau-\rho)+\frac{2}{N})) \tag{28}$$

Therefore, if the following holds:

$$M \leq \exp(2N(\tau-2\rho))\frac{e^{2}-1}{2e^2}, \tag{29}$$

then $\mathbb{E}[\Delta(E_{\text{rand}})] < 0$. Note that the bound from Inequality 26 is stronger than the one from Inequality 29. That completes the proof of the first part of the theorem.

Now we will compute the variance of $\Delta(E_{\text{rand}})$. Denote:

$$Z_k = \sum_{\mu\in\{1,\ldots,M\}}(W_k^{\mu} - Z_k^{\mu}) \tag{30}$$

Note that if $\omega_1, \ldots, \omega_r$ are chosen independently then Z_k for $k = 1, \ldots, r$ are independent. The following is true:

$$\begin{aligned}\mathrm{Var}(\Delta(E_{\text{rand}})) = \mathrm{Var}(E_{\text{signal}} + E_{\text{noise}}) = \mathrm{Var}\left(\frac{1}{r}\sum_{k=1}^{r}\sum_{\mu\in\{1,\ldots,M\}}(W_k^{\mu} - Z_k^{\mu})\right) \\ = \mathrm{Var}(\frac{1}{r}\sum_{k=1}^{r}Z_k) = \frac{1}{r^2}\sum_{k=1}^{r}\mathrm{Var}(Z_k) = \frac{1}{r^2}\sum_{k=1}^{r}\mathrm{Var}\left(\sum_{\mu\in\{1,\ldots,M\}}(W_k^{\mu} - Z_k^{\mu})\right) \\ = \frac{1}{r^2}\sum_{k=1}^{r}\left(\mathbb{E}\left[\left(\sum_{\mu\in\{1,\ldots,M\}}(W_k^{\mu} - Z_k^{\mu})\right)^2\right] - \left(\mathbb{E}\left[\sum_{\mu\in\{1,\ldots,M\}}(W_k^{\mu} - Z_k^{\mu})\right]\right)^2\right)\end{aligned} \tag{31}$$

Therefore we have:

$$\begin{aligned}\mathrm{Var}(\Delta(E_{\mathrm{rand}})) = \frac{1}{r^2}\sum_{k=1}^{r}\left(\sum_{\mu_1,\mu_2\in\{1,...,M\}}\mathbb{E}[W_k^{\mu_1}W_k^{\mu_2}] + \sum_{\mu_1,\mu_2\in\{1,...,M\}}\mathbb{E}[Z_k^{\mu_1}Z_k^{\mu_2}]\right)\\ -\frac{2}{r^2}\sum_{k=1}^{r}\sum_{\mu_1,\mu_2\in\{1,...,M\}}\mathbb{E}[W_k^{\mu_1}Z_k^{\mu_2}]\\ -\frac{1}{r^2}\sum_{k=1}^{r}\left(\sum_{\mu_1,\mu_2\in\{1,...,M\}}\mathbb{E}[W_k^{\mu_1}]\mathbb{E}[W_k^{\mu_2}] + \sum_{\mu_1,\mu_2\in\{1,...,M\}}\mathbb{E}[Z_k^{\mu_1}]\mathbb{E}[Z_k^{\mu_2}]\right)\\ -\frac{2}{r^2}\sum_{k=1}^{r}\sum_{\mu_1,\mu_2\in\{1,...,M\}}\mathbb{E}[W_k^{\mu_1}]\mathbb{E}[Z_k^{\mu_2}]\end{aligned} \tag{32}$$

Note that from the fact that our random feature map based estimators are unbiased, we get (as we already noted before in Equation 19 and put here again for Reader's convenience):

$$\begin{aligned}\mathbb{E}[W_k^{\mu}] = \exp((\xi^{\mu})^{\top}\widehat{\xi}^l),\\ \mathbb{E}[Z_k^{\mu}] = \exp((\xi^{\mu})^{\top}\mathrm{neg}(\widehat{\xi}^l, i)),\end{aligned} \tag{33}$$

Let us now define:

$$\Psi(\mathbf{x}) = D^4\exp(-2N)\exp(B\omega^{\top}\mathbf{x} - 4\widehat{A}\|\omega\|_2^2). \tag{34}$$

Note that the following is true:

$$\begin{aligned}\mathbb{E}[W_k^{\mu_1}W_k^{\mu_2}] = \Psi(\xi^{\mu_1} + \xi^{\mu_2} + 2\widehat{\xi}^l)\\ \mathbb{E}[Z_k^{\mu_1}Z_k^{\mu_2}] = \Psi(\xi^{\mu_1} + \xi^{\mu_2} + 2\mathrm{neg}(\widehat{\xi}^l, i))\\ \mathbb{E}[W_k^{\mu_1}Z_k^{\mu_2}] = \Psi(\xi^{\mu_1} + \xi^{\mu_2} + \widehat{\xi}^l + \mathrm{neg}(\widehat{\xi}^l, i))\end{aligned} \tag{35}$$

Thus it remains to find closed-form formula for $\Psi(\mathbf{x})$ for any given $\mathbf{x} \in \mathbb{R}^N$.

From the proof of Theorem 3.1 in [39], we get for $A < 0$:

$$\mathbb{E}[\exp(A\|\omega\|^2 + B\omega^{\top}\mathbf{x})] = (1-2A)^{-\frac{N}{2}}\exp\left(\frac{B^2}{2(1-2A)}\|\mathbf{x}\|^2\right) \tag{36}$$

Thus we obtain:

$$\Psi(\mathbf{x}) = D^4\exp(-2N)(1+8\widehat{A})^{-\frac{N}{2}}\exp\left(\frac{B^2}{2(1-8\widehat{A})}\|\mathbf{x}\|^2\right) \tag{37}$$

Plugging to Equation 32 formulae from Equation 33 and Equation 35 and utilizing Equation 37 for Ψ, we obtain the formula for the variance from the statement of the Theorem. □

B Experiment details

B.1 Warm-up for Mnemosyne and other optimizers: additional results

Preliminaries: At each timestep t, gradient $\nabla f(\mathbf{x}_t)$ is input to the optimizer. The gradient is pre-processed as proposed in [2]. Coordinate-wise Mnemosyne's using two temporal encoders is applied. The Mnemosyne's memory cell interfaces with the rest of the system similarly to any RNN-cell. Each cell uses exponential discount factor $\tau = 0.1$, $r = 16$ random projections, 16 hidden dimensions and 1 attention head. The memory cell output is fed to a fully connected layer, returning the update to be applied to the NN parameters of the optimizee.

Meta-training: We refer to training optimizer's parameters θ as *meta-training* to distinguish from the optimizee NN training. Mnemosyne's optimizer is meta-trained on MNIST classification task with 3

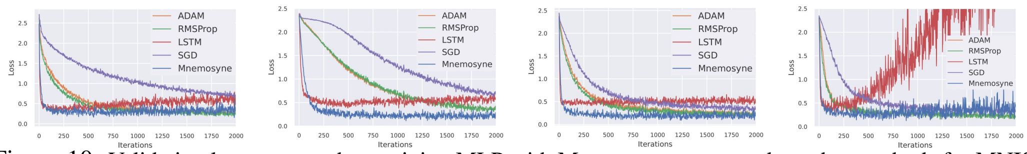

Figure 10: Validation loss curves when training MLP with Mnemosyne compared to other methods for MNIST image classification. Optimization curves for 4 different MLP architectures in this order: (1 layer, 20 hidden dim, $\mathrm{sigmoid}$ activation), (2 layers, 20 hidden dim, $\mathrm{sigmoid}$ activation), (1 layer, 40 hidden dim, $\mathrm{sigmoid}$ activation), (1 layer, 20 hidden dim, relu activation) are shown.

small MLP and 3 small ViT models. The optimizee MLPs are sampled from this hyperparameter distribution: $l \in [1, 2]$ hidden layers of size in range $[20, 40]$ and $\mathrm{sigmoid}$ or relu activation function. The optimizee ViTs have $l \in [1, 3]$ layers, $h \in [1, 3]$ heads, with hidden dimension in range $[16, 64]$, mlp dimension in range $[16, 64]$ and head dimension in range $[8, 16]$. The optimizee task is to train the model for 100 steps on batches of 64 image-class examples.

Hybrid loss function to improve generalization: To promote generalization, we use the random-scaling trick proposed by [42]. Mnemosyne's optimizer is meta-trained by gradient descent using Adam optimizer with learning rate $\eta = 3e^{-4}$ to minimize a combination of two loss functions. The first is the task loss given by the sum of optimizee losses in a truncated roll-out of 5 MNIST training steps. The other one is an imitation loss given by the mean squared error between Mnemosyne's updates and expert-optimizer (Adam) updates for same inputs. Importantly, this imitation loss is different from the one proposed in [12] which uses off-policy expert roll-outs for imitation. In our case, we provide expert supervision for the on-policy updates. This mitigates the problem of divergence from expert's trajectory, often observed in behaviour cloning. Our imitation loss acts as a regularizer which prevents Mnemosyne's optimizer from over-fitting on the optimizee task that it is trained on. We emphasize that expert's learning rate $\eta_{\mathrm{exp}} = 3e^{-2}$ **was not obtained via any tuning process**.

Our optimizer model has minimal input feature engineering and our meta-training setup is significantly simpler than those considered in the literature [44, 12, 42, 65]. Even so, we can successfully apply Mnemosyne's optimizer to a variety of tasks due to its efficient memory mechanism. Furthermore, Mnemosyne's memory cells can be easily combined with any of the existing L2L methods that use LSTMs for memory-encoding.

Results: After meta-training, Mnemosyne's optimizer was tested on NN training tasks with different NN architectures and datasets. Recall that Mnemosyne only saw one ML task of MNIST classifier training for 100 steps during meta-training. Fig. 10 shows that Mnemosyne can optimize MLPs with different NN archtitectures and activation functions on MNIST image classifier training. Note that, Mnemosyne converges significantly faster than popular analytical optimizers, RMSprop and Adam while retaining similar asymptotic performance. Mnemosyne can train NNs for long horizons of thousands of steps while baseline LSTM optimizer [2] struggles to minimize classification loss beyond a few hundred steps.

Transformers: The results were already presented in the main body of the paper (see: Sec. 5.1). We want to add that, as for experiments from Fig. 10, here Mnemosyne's optimizer is faster than standard analytical optimizers and much more stable than LSTM optimizer. Fig. 11 shows the benefit of using expert imitation-loss for long-horizon stability of the Mnemosyne's optimizer.

Our results on training Transformers with Mnemosyne naturally lead to the question of the role that Transformer-based optimizers can play in training Transformers architectures. It is well known that Transformer training requires nontrivial optimization techniques [40], e.g. learning rate schedulers (for that reason SGD was replaced with Adam in Transformer-training). Furthermore, for larger architectures training is slow, often prohibitively (unless the model is trimmed down, for instance by replacing long-range attention modeling with local attention of the controllable attention radius). Attention-based optimizers can potentially address this problem, since they improve convergence (and thus effectively reduce training time) even if meta-trained on much simpler tasks as we show in Fig. 3.

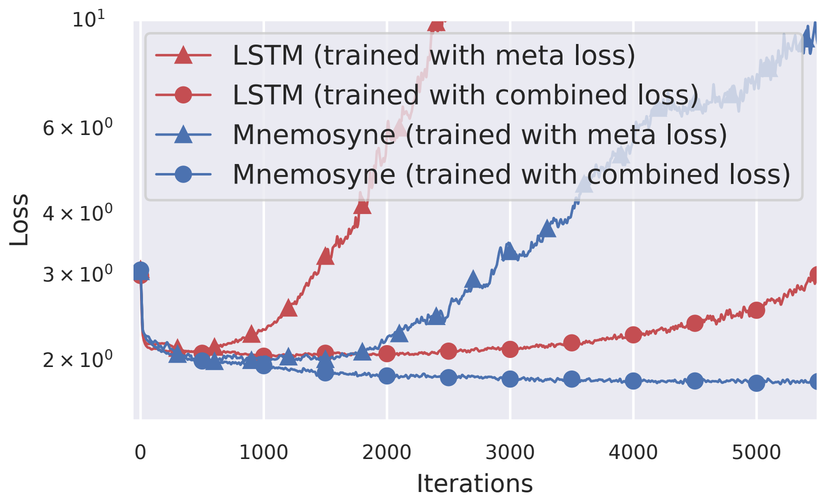

Figure 11: Impact of training the optimizer with combined meta loss and imitation loss can be seen in generalization to a long horizon rollout. All variants were trained only on length 100 rollouts.

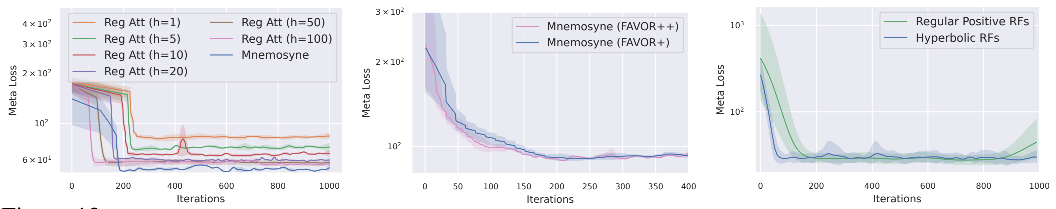

Figure 12: Ablation Studies. **Left:** Comparison of the Mnemosyne's linear CAM with regular attention memory blocks with different history cache lengths (h). **Middle:** Meta-training curves of Mnemosyne optimizer with FAVOR+ and FAVOR++ mechanism for CAM. **Right:** Meta-training curves of Mnemosyne optimizer with different kernel transformation functions for CAM.

B.2 Mnemosyne's CAM mechanism vs regular attention

We have tried to use regular Transformer blocks to encode associative memory for Mnemosyne's temporal module. For applying regular attention to online optimizer autoregressively, a limited-length cache of historical gradients has to be maintained. A self-attention map over the history sequence is generated and used to encode memory. Fig. 12 (left) shows the meta-training curves for regular attention optimizers with different history cache lengths. As we increase the cache length, the performance improves and the memory requirement scales quadratically. Due to this limitation, we could not implement a regular attention based optimizer with cache length more than 100. On the other hand, Performer's memory cell defining CAM can attend to theoretically unbounded history and out-performs regular attention variants with fixed memory requirement.

B.3 Different RF-mechanisms: detailed look

Fig. 12 (middle) compares the performance of Mnemosyne's optimizer applying FAVOR+ and FAVOR++ mechanisms in CAM. FAVOR++ mechanism provides strongest theoretical guarantees for the capacity of the associative memory model. It also leads initially to faster convergence, but asymptotically performs similarly as the FAVOR+ variant. Due to the simpler implementation of FAVOR+, we use it for all experiments with Mnemosyne's optimizer.

Optimizers with both regular positive and hyperbolic random features kernel learn similarly, but the latter has much lower variance (see: Fig. 12 (right)) and thus it became our default choice.

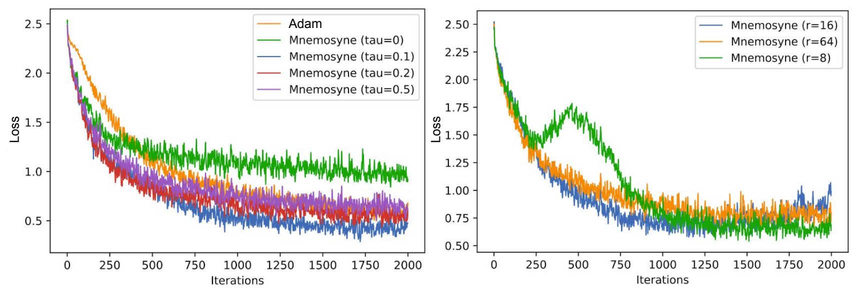

Figure 13: **Left:** The comparison of Mnemosyne applying different discount factors with Adam optimizer in meta-training (MLP optimization). **Right**: The comparison of Mnemosyne applying different number of random features in the hyperbolic cosine random feature mechanism used in CAM.

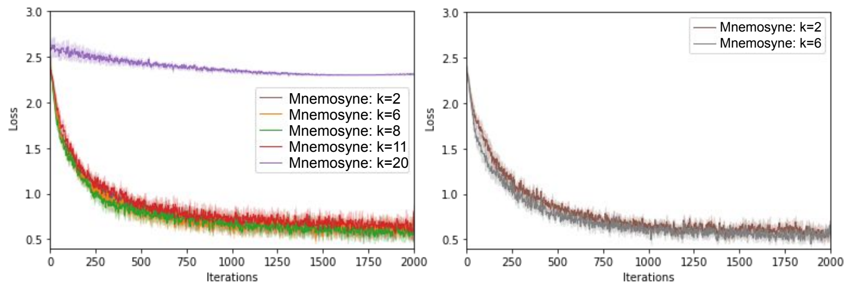

Figure 14: Comparison of the meta-training loss for Mnemosyne variants applying different number of temporal encoders k. Since several variants on the left figure performs similarly, on the right figure we highlight top two. The meta-training is conducted on the MLP optimization tasks and MNIST data.

B.4 Ablations over different discount factors and number of RFs in CAM mechanism

In Fig. 13, we present detailed ablation studies over discount factors τ as well as the number of random features applied by our default CAM mechanism leveraging hyperbolic cosine random features.

B.5 Benchmarking different depths of the temporal module

Finally, we run ablations over different number of temporal encoders in Mnemosyne's temporal block. We noticed that modest increase of the number of encoders improves loss in meta-training and meta-training very deep variants is particularly challenging (as requiring much more data). Since in this paper we decided to use simple meta-training strategies and furthermore increasing the number of temporal encoders did not lead to substantial gains, we decided to choose shallow temporal encoders' architectures. The results are presented in Fig. 14.

B.6 Compute Resources Used

All Mnemosyne optimizer variants were trained and tested on a TPU pod containing 4 TPU v3 chips with JAX. Hundreds of rounds of training and inference were needed to compare different variations, tasks and meta-losses.

B.7 Coordinate-wise Mnemosyne versus hard-coded optimizers for larger ViTs

B.7.1 ViT last-layer fine-tuning

In this study, we benchmarked Mnemosyne on different sizes of ViT architectures: ViT-Base, ViT-Large and ViT-Huge (ViT-B(x), ViT-L(x) and ViT-H(x) respectively, where x defines the patch size), see: Tab: 1. We used the coordinate-wise variant of the Mnemosyne. We run tests on the following datasets: $\mathrm{imagenet2012}$, $\mathrm{places365}$ and $\mathrm{caltech\text{-}birds\text{-}2011}$. We were optimizing the last layer of the ViT-architecture and used Adam expert with learning rate $\eta = 3e^{-2}$ as a regularizer (see: our discussion above on meta-training). The learning rate was not tuned in any way. In fact (as we show below) Adam optimizer applying this learning rate is characterized by the sub-optimal performance.

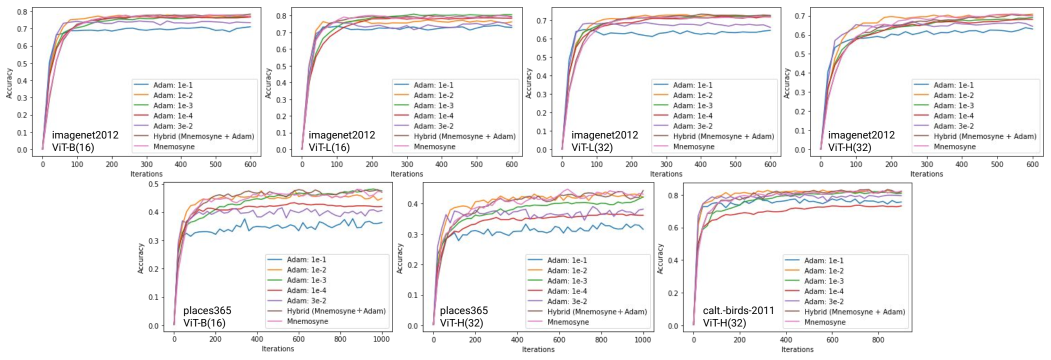

Figure 15: Coordinate-wise Mnemosyne across different ViT architectures and datasets, as described in Sec. B.7.1.

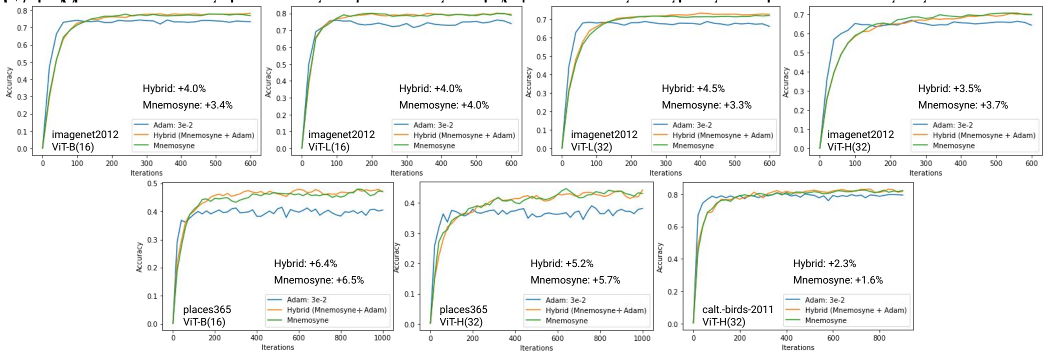

Figure 16: The results from Fig. 15, but narrowed down to the comparison between two Mnemosyne variants and Adam optimizer applying learning used in meta-training of these variants of Mnemosyne. The expert substantially underperforms in all the cases (we explicitly put the gains coming from the two variants of Mnemosyne as compared to the expert variant). This shows that Mnemosyne does not learn to imitate the expert.

We tried two versions of Mnemosyne: (a) a variant that solely optimizes the last layer of ViT (reported in the main body) and (b) the *hybrid* variant, where Mnemosyne is used to optimize the weight-matrix of the last layer and Adam with learning rate $\eta = e^{-3}$, to optimize the bias vector. That learning rate was also not tuned in any particular way and, as before, if applied purely within Adam, produces sub-optimal results. The purpose of that last experiment was to assess how efficient the strategy of optimizing jointly with Mnemosyne and a hand-designed optimizer is. The results are presented in Fig. 15 and Fig. 16. We see that: (a) Mnemosyne without any hyperparameter tuning matches or outperforms optimal Adam variants, (b) it also substantially outperforms Adam variant used as an expert in meta-training. This is valid for both: regular Mnemosyne as well as the hybrid version.

B.7.2 ViT multi-layer fine-tuning

Here, we used a *light* version of coordinate-wise Mnemosyne using a single temporal encoder layer with hidden dimension 8. This reduced the memory requirement of the Mnemosyne optimizer state. We fine-tune ViT-B model on CIFAR-100 dataset with batch size 128. We were able to fine-tune last 2 transformer layers along with the embedding, cls and head layers with Mnemosyne. Rest of the model was fine-tuned with Adam (learning rate = $1e^{-3}$). For comparison, the same baseline Adam variant is to fine-tune the complete model.

Table 1: Hyperparameters for the different ViT models used in this paper

Model	Heads	Layers	Hidden Dim.	MLP Dim.	Params	Patch Size
ViT-Base	12	12	768	3072	86M	16
ViT-Large (16)	24	16	1024	4096	307M	16
ViT-Large (32)	24	16	1024	4096	307M	32
ViT-Huge	32	16	1280	5120	632M	32

B.8 Tensor-wise Mnemosyne versus hard-coded optimizers for ViT-H

We finetuned the embedding and cls layer of ViT-H (see Tab: 1 for hyperparameter) using tensorwise ($\sim 1M$ params), while the head was trained using Adam. The rest of the transformer parameters are fixed to the pre-trained value for all methods. The batch size was set at 128 for all methods.

B.9 Super-Mnemosyne: combining coordinate- and tensor-wise strategies for ViTs

We finetuned the top-8 layers of the ViT-Base model (see Tab: 1) along with the head, cls and embedding layer before we ran out of memory ie $\sim 50M$ parameters with a batch size of 256. Large tensor such as: a) the MLP block withing each layer, b) the head layer was finetuned using lite version of coordinate-wise. Rest of the tensors were finetuned using tensorwise. The bottom 4 layers of the model were kept fixed for Mnemosyne. For Adam baselines we finetuned all layers.

B.10 BERT-pretraining NLP Transformers with Mnemosyne

We trained the Bert base model, whose Hyperparameters are shown in Tab: 2. The details of the training dataset used is shown in Tab: 3. We trained all parameters from scratch for all methods, with a batch size of 512. For the Mnemosyne results shown in Fig: 7, we trained all parameters except the token embedding using Tensorwise Mnemosyne ($\sim 86M$ parameters). The token embedding was trained using Adam with learning rate $1e-4$. For Adam baseline we trained all parameters.

Table 2: Hyperparameters for the Bert base model

Model	Heads	Layers	Hidden Dim.	MLP Dim.	Params	Compute	Loss
Bert-Base	12	12	768	3072	110M	4x2 TPUv3	MLM

Table 3: Dataset used for pre training.

Dataset	# tokens	Avg. doc len.
Books [70]	1.0B	37K
Wikipedia	3.1B	592

B.11 Soft prompt-tuning massive T5XXL Transformers with Mnemosyne

We use coordinate-wise Mnemosyne to prompt-tune [34] T5XXL model [50] (see Table 4 for hyper-parameters) on SuperGLUE benchmark. Batch size 32 was used. The length of the soft-prompt sequence was 30 and each soft-prompt vector was of size 4096, making the total number of trainable parameters 122880.

Table 4: Hyperparameters for the T5XXL model

Model	Encoder Layers	Decoder Layers	Heads	Head Dim.	Embedding Dim.	MLP Dim.	Params	Compute
T5XXL	24	24	64	64	4096	10240	11B	2x2x4 TPUv3