Gated Linear Networks

Joel Veness [*1] **Tor Lattimore** [*1] **David Budden** [*1] **Avishkar Bhoopchand** [*1] **Christopher Mattern** [1]
Agnieszka Grabska-Barwinska [1] **Eren Sezener** [1] **Jianan Wang** [1] **Peter Toth** [1] **Simon Schmitt** [1] **Marcus Hutter** [1]

Abstract

This paper presents a new family of backpropagation-free neural architectures, Gated Linear Networks (GLNs). What distinguishes GLNs from contemporary neural networks is the distributed and local nature of their credit assignment mechanism; each neuron directly predicts the target, forgoing the ability to learn feature representations in favor of rapid online learning. Individual neurons can model nonlinear functions via the use of data-dependent gating in conjunction with online convex optimization. We show that this architecture gives rise to universal learning capabilities in the limit, with effective model capacity increasing as a function of network size in a manner comparable with deep ReLU networks. Furthermore, we demonstrate that the GLN learning mechanism possesses extraordinary resilience to catastrophic forgetting, performing comparably to a MLP with dropout and Elastic Weight Consolidation on standard benchmarks. These desirable theoretical and empirical properties position GLNs as a complementary technique to contemporary offline deep learning methods.

1. Introduction

Backpropagation has long been the de-facto credit assignment technique underlying the successful training of popular neural network architectures such as convolutional neural networks and multilayer perceptions (MLPs). It is well known that backpropagation enables these networks to learn highly-relevant task-specific features. However, this method is not without its limitations. Contemporary neural networks trained via backpropagation require many epochs of training over massive datasets, limiting their effectiveness for data-efficient online learning. Interpretibility limitations can also prevent their application in domains where a human understandable solution is a mandatory requirement. Their effectiveness is further limited in the continual learning setting by their tendency to catastrophically forget previously learnt tasks. Although various meta-learning [OWR+19] algorithms such as Elastic Weight Consolidation [KPR+17, EWC] have been effective in compensating for these limitations, it is interesting to explore whether alternative methods of credit assignment can give rise to complementary neural models with different strengths and weaknesses.

This paper introduces one alternative model family, Gated Linear Networks (GLNs), and studies their contrasting properties. The distinguishing feature of a GLN is its distributed and local credit assignment mechanism. This technique is a generalization of the PAQ family [Mah00, Mah05, Mah13] of online neural network models, which are well-known in the data compression community for their excellent sample efficiency [Mah13, Kno17]. By interpreting these systems within an online convex programming [Zin03] framework as a sequence of data dependent linear networks coupled with a choice of gating function, we are able to provide a new algorithm and gating mechanism that opens up their usage to the wider machine learning community.

GLNs have a number of desirable properties. Their local credit assignment mechanism is derived by associating a separate convex loss function to each neuron, which greatly simplifies parameter initialization and optimization, and provides significant sample efficiency benefits when learning online. Importantly, we show that these benefits do not come at the expense of capacity in practice, which adds further weight to previously obtained asymptotic universality results [VLB+17].

GLNs possess excellent online learning capabilities, which we demonstrate by showing performance competitive with batch-trained MLPs on a variety of standard classification, regression and density modeling tasks, using only a single online pass through the data. In terms of interpretibility, we show how the data-dependent linearity of the predictions can be exploited to trivialise the process of constructing meaningful saliency maps, which can be of great reassurance to practitioners that the model is predicting well for the right reasons. Perhaps most interestingly, we demonstrate that our credit assignment mechanism is extraordinarily resilient to catastrophic forgetting, achieving performance competitive with EWC on a standard continual learning benchmark with no knowledge of the task boundaries.

[*]Equal contribution [1]DeepMind.

2. Background

In this section we review some necessary background on geometric mixing, a parametrised way of combining probabilitstic forecasts, and show how to adapt its parameters using online convex programming. Later we will combine this method with a gating mechanism to define a single neuron within a GLN.

Geometric Mixing. Geometric Mixing is a simple and well studied ensemble technique for combining probabilistic forecasts. It has seen extensive application in statistical data compression [Mat12, Mat13]. Given $p_1, p_2, \ldots, p_d$ input probabilities predicting the occurrence of a single binary event, geometric mixing predicts $\sigma(w^\top \sigma^{-1}(p))$, where $\sigma(x) := 1/(1+\mathrm{e}^{-x})$ denotes the sigmoid function, σ^{-1} defines the logit function, $p := (p_1, \ldots, p_d)$ and $w \in \mathbb{R}^d$ is the weight vector which controls the relative importance of the input forecasts. One can easily show the following identity:

$$\sigma\left(w^\top \sigma^{-1}(p)\right) = \frac{\prod_{i=1}^d p_i^{w_i}}{\prod_{i=1}^d p_i^{w_i} + \prod_{i=1}^d (1-p_i)^{w_i}},$$

which makes it clear that that geometric mixing implements a type of product of experts [Hin02] operation. This leads to the following interesting properties: setting $w_i = 1/d$ is equivalent to taking the geometric mean of the d input probabilities; if the jth component of w_j is 0 then the contribution of p_j is ignored, and if $w = 0$ then the geometric mixture predicts $1/2$; and finally, due to the product formulation, every forecaster has "the right of veto", in the sense that a single p_i close to 0 coupled with a $w_i > 0$ drives the geometric mixture prediction close to zero.

Online Convex Programming Formulation. We now describe how to adapt the geometric mixing parameters using online convex programming [Zin03, Haz16]. Let $\mathcal{B} := \{0, 1\}$. As we are interested in probabilistic prediction, we assume a standard online learning framework for the logarithmic loss, where at each round $t \in \mathbb{N}$ a predictor outputs a binary distribution $q_t : \mathcal{B} \to [0, 1]$, with the environment responding with an observation $x_t \in \mathcal{B}$, causing the predictor to suffer a loss $\ell_t(q_t, x_t) = -\log q_t(x_t)$ before moving onto round $t + 1$.

In the case of geometric mixing, we first define our parameter space to be a non-empty convex set $\mathcal{W} \subset \mathbb{R}^d$. As the prediction depends on both the d dimensional input predictions p_t and the parameter vector $w \in \mathcal{W}$, we abbreviate the loss at time t, given target x_t, using parameters w by

$$\ell_t^{\text{GEO}}(w) := -\log\left(\text{GEO}_w(x_t\,;\,p_t)\right), \tag{1}$$

with $\text{GEO}_w(1; p_t) := \sigma(w^\top \sigma^{-1}(p_t))$ and $\text{GEO}_w(0\,;\,p_t) := 1 - \text{GEO}_w(1\,;\,p_t)$. One can show that $\ell_t^{\text{GEO}}(w)$ is a convex function of w [Mat13] and that the gradient of the loss with respect to w is given by

$$\nabla \ell_t^{\text{GEO}}(w) = \left(\text{GEO}_w(1; p_t) - x_t\right) \text{logit}(p_t). \tag{2}$$

Furthermore we can bound the 2-norm of the gradient of the loss with $\|\nabla \ell_t^{\text{GEO}}(w)\|_2 \leq \sqrt{d} \log\left(\frac{1}{\varepsilon}\right)$ provided that $p_t \in [\varepsilon, 1-\varepsilon]^d$ for some $\varepsilon \in (0, 1/2)$ for every time t. These properties of the sequence of loss functions make it possible to apply one of the many different online convex programming techniques to adapt w at the end of each round. In this paper we restrict our attention to Online Gradient Descent [Zin03], with $\mathcal{W}$ equal to some choice of hypercube, for reasons of computational efficiency. This gives a $O(\sqrt{T})$ regret bound with respect to the best $w^* \in \mathcal{W}$ chosen in hindsight provided an appropriate schedule of decaying learning rates is used.

3. Gated Geometric Mixing

We define the GLN neuron as a *gated geometric mixer*, which we obtain by adding a contextual gating procedure to geometric mixing. Here, contextual gating has the intuitive meaning of mapping particular input examples to particular sets of weights. The key change compared with normal geometric mixing is that now our neuron will also take in an additional type of input, *side information*, which will be used by the contextual gating procedure to determine an active subset of the neurons weights to use for a given example. In typical applications the side information will simply be the input features associated with a given example.

More formally, associated with each neuron is a context function $c : \mathcal{Z} \to \mathcal{C}$, where $\mathcal{Z}$ is the set of possible *side information* and $\mathcal{C} = \{0, \ldots, k-1\}$ for some $k \in \mathbb{N}$ is the *context space*. Given a convex set $\mathcal{W} \subset \mathbb{R}^d$, each neuron is parametrized by a matrix $W = \left[w_0 \ldots w_{k-1}\right]^\top$ with each row vector $w_i \in \mathcal{W}$ for $0 \leq i < k$. The context function c is responsible for mapping a given piece of side information $z_t \in \mathcal{Z}$ to a particular row $w_{c(z_t)}$ of W, which we then use with standard geometric mixing.

In other words, a Gated Geometric Mixer can be defined in terms of geometric mixing as

$$\text{GEO}_W^c(x_t\,;\,p_t, z_t) := \text{GEO}_{w_{c(z_t)}}(x_t\,;\,p_t), \tag{3}$$

with the associated loss function $-\log\left(\text{GEO}_W^c(x_t\,;\,p_t, z_t)\right)$ inheriting all the properties needed to apply Online Convex Programming directly from Equation 1. The key intuition behind gating is that it allows each neuron to be able to specialize its weighting of input predictions based on some particular property of the side information.

Universal context functions. We now introduce a *half-space gating* mechanism that is tailored towards machine

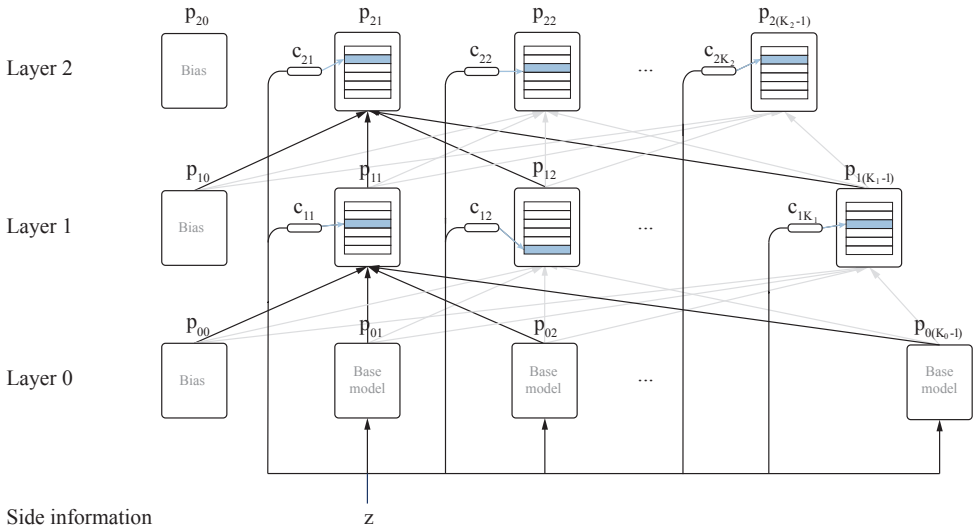

Figure 1. A graphical depiction of a Gated Linear Network. Each neuron receives inputs from the previous layer as well as the broadcasted side information z. The side information is passed through all the gating functions, whose outputs $s_ij = c_{ij}(z)$ determine the active weight vectors (shown in blue).

learning applications whose input features lie in $\mathbb{R}^d$. Although not the focus of this work, its worth noting that this choice gives rise to universal approximation capabilities for sufficiently large GLNs [VLB+17]. Once we are in a position to describe the learning dynamics of multiple interacting neurons, the rationale for this class of context functions will become more clear. Exploring alternative gating mechanisms is an exciting area for future work.

Halfspace gating. Given a normal $v \in \mathbb{R}^d$ and offset $b \in \mathbb{R}$, consider the associated affine hyperplane $\{x \in \mathbb{R}^d : x \cdot v = b\}$. This divides $\mathbb{R}^d$ in two, giving rise to two half-spaces, one of which we denote

$$H_{v,b} = \{x \in \mathbb{R}^d : x \cdot v \geq b\}.$$

The associated half-space context function is then given by $\mathbb{1}_{H_{v,b}}(z)$, where $\mathbb{1}_{\mathcal{S}}(s) := 1$ if $s \in \mathcal{S}$ and 0 otherwise.

Context composition. Richer notions of context can be created by composition. In particular, any finite set of m context functions $\{c_i : \mathcal{Z} \to \mathcal{C}_i\}_{i=1}^m$ with associated context spaces $\mathcal{C}_1, \ldots, \mathcal{C}_m$ can be composed into a single higher order context function $c : \mathcal{Z} \to \mathcal{C}$, where $\mathcal{C} = \mathcal{C}_1 \times \ldots \times \mathcal{C}_d \cong \{0, ..., |\mathcal{C}| - 1\}$ by defining $c(z) = (c_1(z), ..., c_d(z))$.

For example, we could combine $m = 4$ different halfspace context functions into a single context function with a context space containing $|\mathcal{C}| = 16$ elements. From here onwards, whenever this technique is used, we will refer to the choice of m as the *context dimension*.

4. Gated Linear Networks

We now introduce *Gated Linear Networks*, which are feedforward networks composed of many layers of gated geometric mixing neurons as shown in Figure 1. Each neuron in a given layer outputs a gated geometric mixture of the predictions from the previous layer, with the final layer consisting of just a single neuron. In a supervised learning setting, a GLN is trained on (side information, base predictions, label) triplets $(z_t, p_t, x_t)_{t=1,2,3,...}$ derived from input-label pairs (z_t, x_t). There are two types of input to neurons in the network: the first is the side information z_t, which can be thought of as the input features; the second is the input to the neuron, which will be the predictions output by the previous layer, or in the case of layer 0, some (optionally) provided base predictions p_t that typically will be a function of z_t. Each neuron will also take in a constant bias prediction, which helps empirically and is essential for universality guarantees [VLB+17].

GLN architecture. A GLN is a network of gated geometric mixers organized in $L + 1$ layers indexed by $i \in \{0, \ldots, L\}$, with K_i models in each layer. Neurons are indexed by their position in the network when laid out on a grid; for example, neuron (i, k) will refer to the kth neuron of the i layer and p_{ik} will refer to the output of neuron (i, k). The output of layer i will be denoted by p_i. The zeroth layer of the network is called the *base layer*, whose output p_0 will typically be instantiated via scaling or squashing each component of the current side information z to lie within $[\varepsilon, 1 - \varepsilon]$. The nonzero layers are composed of gated geometric mixing neurons. Associated to each of these will

Was confused by this. But OK, combining 4 half-space context functions gives a 4-digit bi

be a fixed context function $c_{ik} : \mathcal{Z} \to \mathcal{C}$ that determines the behavior of the gating at neuron (i, k). In addition to the context function, for each context $c \in \mathcal{C}$ and each neuron (i, k) there is an associated weight vector $w_{ikc} \in \mathbb{R}^{K_{i-1}}$ which is used to geometrically mix the inputs whenever active. The bias outputs p_{i0} for $0 \le i \le L$ can be set to be any constant $\beta \in [\varepsilon, 1-\varepsilon] \setminus \{0.5\}$. Given a $z \in \mathcal{Z}$, a weight vector for each neuron is determined by evaluating its associated context function. For layers $i \ge 1$, the kth node in the ith layer receives as input the vector p_{i-1} of dimension K_{i-1} of predictions of the preceding layer.

GLNs are data dependent linear networks. Without loss of generality, here we assume that the network is estimating the probability of the target being positive. The output of a single neuron is the geometric mixture of the inputs with respect to a set of weights that depend on its context, namely

$$p_{ik}(z) = \sigma\left(w_{ikc_{ik}(z)} \cdot \sigma^{-1}\left(p_{i-1}\left(z\right)\right)\right).$$

The output of layer i can be written in matrix form as

$$p_i(z) = \sigma(W_i(z)\, \sigma^{-1}(p_{i-1}(z)))\,, \tag{4}$$

where $W_i(z) \in \mathbb{R}^{K_i \times K_{i-1}}$ is the matrix with kth row equal to $w_{ik}(z) = w_{ikc_{ik}(z)}$. Iterating Equation 4 once gives

$$p_i(z) = \sigma\left(W_i\left(z\right)\sigma^{-1}\left(\sigma\left(W_{i-1}\sigma^{-1}\left(p_{i-2}\left(z\right)\right)\right)\right)\right).$$

Observing that the logit and sigmoid functions cancel, simplifying the ith iteration of Equation 4 gives

$$p_i(z) = \sigma\Big(W_i(z)W_{i-1}(z)\ldots W_1(z)\,\sigma^{-1}(p_0(z))\Big), \tag{5}$$

which shows the network behaves like a linear network [BH89, SMG13], but with weight matrices that are data-dependent. Without the data dependent gating, the product of matrices would collapse to a single linear mapping and provide no additional modeling power over a single neuron [MP69].

Local learning in GLNs. We now describe how the weights are learnt in a Gated Linear Network using Online Gradient Descent (OGD) [Zin03] locally at each neuron. They key observation is that as each neuron (i, k) in layers $i > 0$ is itself a gated geometric mixture, all of these neurons can be thought of as individually predicting the target. Thus given side information z and from Equations 1 and 3, each neuron (i, k) suffers a loss convex in its active weights $u := w_{ikc_{ik}(z)}$ of

$$\ell_t(u) := \quad -\log\left(\text{GEO}_u\left(x_t\,;\, p_{i-1}\right)\right).$$

Algorithmically, a single step of OGD consists of two parts: a gradient step, and then a projection back into some convex weight space $\mathcal{W}$. The gradient step can be trivially obtained from Equation 2. It is well known that the projection step can be implemented via clipping if the convex set $\mathcal{W}$ is a scaled hypercube. In our case this can be achieved if we force every component of each weight vector, for each neuron, to lie within $[-b, b]$ for some constant $b > 1$.

Algorithm 1 GLN$(\Theta, z, p, x, \eta, \text{update})$.
Perform a forward pass and optionally update weights.

Input: GLN weights $\Theta \equiv \{w_{ijc}\}$
Input: side info z, base predictions $p \in [\varepsilon; 1-\varepsilon]^{K_0-1}$
Input: binary target x, learning rate $\eta \in (0, 1)$
Input: boolean *update* (controls if we learn or not)
Output: estimate of $\mathbb{P}[x = 1 \mid z, p]$
$p_0 \leftarrow (\beta, p_1, p_2, \ldots, p_{K_0-1})$
for $i \in \{1, \ldots, L\}$ **do** {loops over layers}
  $p_{i0} \leftarrow \beta$
  for $j \in \{1, \ldots, K_i\}$ **do** {loops over neurons}
    $p_{ij} \leftarrow \text{CLIP}_{\varepsilon}^{1-\varepsilon}\left[\sigma\left(w_{ijc_{ij}(z)} \cdot \sigma^{-1}(p_{i-1})\right)\right]$
    if update **then**
      $\Delta_{ij} \leftarrow -\eta\,(p_{ij} - x)\,\sigma^{-1}(p_{i-1})$
      $w_{ijc_{ij}(z)} \leftarrow \text{CLIP}_{-b}^{b}[w_{ijc_{ij}(z)} + \Delta_{ij}]$
    end if
  end for
end for
return p_{L1}

Weight initialization. One benefit of a convex loss is that weight initialization is less important in determining overall model performance, and one can safely recommend deterministic initialization schemes that favor reproducibility of results. While other choices are possible, we found the initialization $w_{ikc} = 1/K_{i-1}$ for all i, k, c to be a good choice empirically, which causes geometric mixing to initially compute a geometric average of its input.

Algorithm. A single prediction step, as well as a single step of learning using Online Gradient Descent, can be implemented via a single forward pass of the network as shown in Algorithm 1. Here we make use of a subroutine $\text{CLIP}_{\varepsilon}^{1-\varepsilon}[x] := \min\{\max(x, \varepsilon), 1-\varepsilon\}$. Generating a prediction requires computing the active contexts from the given side information for each neuron, and then performing L matrix-vector products. Under the assumption that multiplying a $m \times n$ by $n \times 1$ pair of matrices takes $O(mn)$ work, the total time complexity to generate a single prediction is $O(\sum_{i=1}^{L} K_i K_{i-1})$ for the matrix-vector products, which in typical cases will dominate the overall runtime. Note that updating the weights does not affect this complexity.

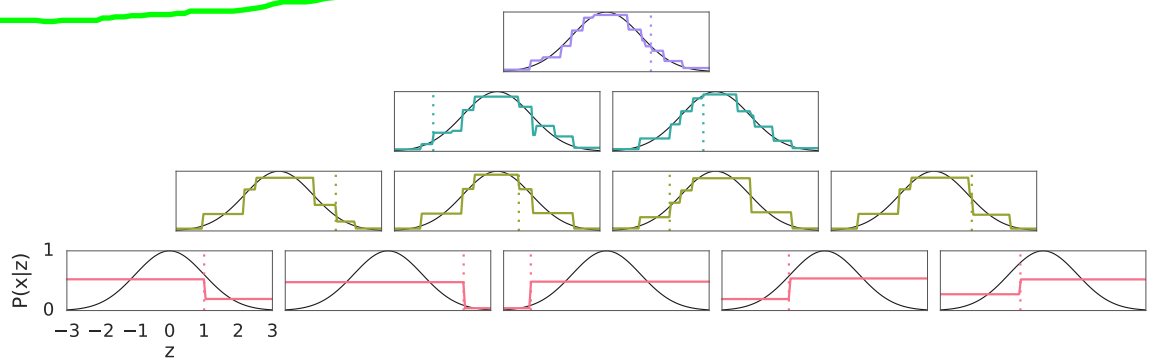

Figure 2. Output of a four layer network with random half-space contexts after training to convergence. Each box represents a non-bias neuron in the network, the function to fit is shown in black, and the output distribution learnt by each neuron is shown in colour (for example, red for the first layer and purple for the top-most neuron). All axes are identical, as labeled in the bottom left neuron. The dashed coloured lines represent the sampled hyperplane for each neuron.

Random halfspace sampling. Here we describe how we generate a diverse set of halfspace context functions in practice. As we are interested in higher dimensional applications, it is necessary to sample hyperplanes in a manner that addresses the curse of dimensionality. Consider a halfspace context function: $c(z; v, b) = 1$ if $z \cdot v \geq b$; or 0 otherwise. To sample v, we first generate an i.i.d. random vector $x = (x_1, ..., x_d)$ of dimension d, with each component of x distributed according to the unit normal $\mathcal{N}(0, 1)$, and then divide by its 2-norm, giving us a vector $v = x/||x||_2$. This scheme uniformly samples points from the surface of a unit sphere. The scalar b is sampled directly from a standard normal distribution.

The motivation for this approach is two-fold. First, With large d, the hyperplanes defining each half-space are orthogonal with high probability; i.e. this choice should help to chop the data up in complementary ways given a limited number of gates. Second, suppose we have a set of m different gating functions $c_i(z; v_i, b_i)$ for 1 to m. Now consider the binary vector: $g = (c_1(z; v_1, b_1), ..., c_m(z; v_m, b_m))$. This *signature* vector g of input z has the property [Cha02] that different z's which are close in terms of cosine similarity will map to similar signatures. For a GLN, this gives rise to the desirable property that inputs close in cosine distance will map to similar products of data dependent matrices, i.e. they will predict similarly.

On convergence properties and rates for GLNs. Asymptotic convergence results for GLNs on i.i.d. data can be proven. On-average within each context cell, the prediction converges to the true expected output/probability, and a sufficiently large GLN can represent the true target probabilities arbitrarily well [VLB+17]. For example, Figure 2 shows the converged predictions when using a small GLN to fit a simple parametrized density function. Of course while asymptotic convergence is a useful sanity check for any model, it tells us little about practical finite-time performance. Below we will outline how (good) convergence rates may be obtained for fixed finite sized GLNs to the best locally learnable approximation. Precisely obtaining such bounds is outside the scope of this paper.

For a single gated neuron, one can show [VLB+17] that Online Gradient Descent (OGD) [Zin03] with a learning rate proportional to $1/\sqrt{t}$ has total regret of $O(\sqrt{T})$ with respect to the best $w^* \in \mathcal{W}$ chosen in hindsight. The loss function ℓ_t is exp-concave, so Online Newton Step [HAK07] can improve the regret to $O(\log T)$, but is computationally more expensive. If the expected loss $\ell(w) := \mathbb{E}[\ell_t(w)]$ were strongly convex, then Stochastic Gradient Descent (SGD) with i.i.d. sampling and a learning rate proportional to $1/t$ would also achieve a regret of $O(\log T)$, Unfortunately ℓ_t is flat in all directions orthogonal to logit(p_t), hence not strongly convex. But since ℓ_t is exp-concave (strongly convex in gradient direction), this makes $\ell(w)$ strongly convex in the linear subspace of $\mathcal{W} \subset \mathbb{R}^d$ spanned by $\mathcal{S} :=$ Span(logit(p_1), ..., logit(p_n)). For sample size n larger than d it is plausible that $\mathcal{S} = \mathbb{R}^d$. Even if not, all ℓ_t are exactly constant in directions orthogonal to $\mathcal{S}$, hence the gradient lies in $\mathcal{S}$. Since ℓ is strongly convex in $\mathcal{S}$, SGD will still achieve a regret of $O(\log T)$.

The above implies that the time-averaged weights $\bar{w}_T$ have an instantaneous regret of $O(\log T\ /T)$, and even $O(1/T)$ can be achieved [B+15, Thm.6.2]. In general, OGD algorithms can be converted to achieve these rates even for the current weight w_t [Cut19], and, indeed, SGD achieves the former even unmodified [SZ13]. By strong convexity on $\mathcal{S}$, this implies that the (time-averaged) weights (or at least the outputs) converge with a rate of $\tilde{O}(t^{-1/2})$.

Hence after time $\tilde{O}(1/\varepsilon^2)$ the output of the first layer has converged within $O(\varepsilon)$, after which the input to the next layer becomes approximately i.i.d. A similar analysis should then be possible for the second layer, and so on. With an appropriately delayed learning rate decay, this should

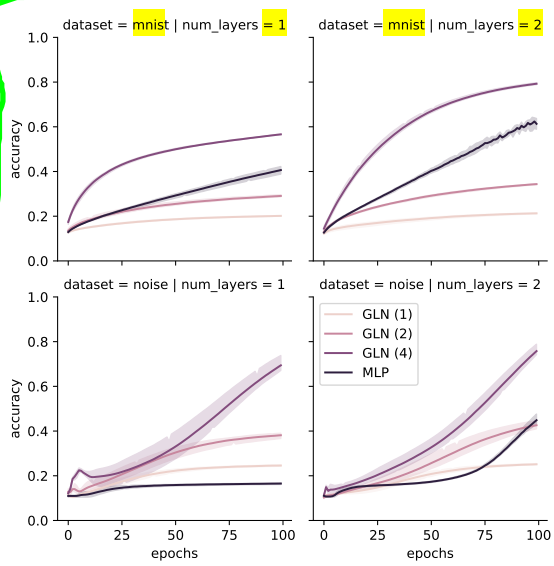

Figure 3. Empirical capacity of GLNs (context dimension showed in parentheses) versus an MLP baseline. Top row represents the MNIST dataset with shuffled labels. Bottom row represents a dataset of uniform noise of the same size and shape.

lead to an overall time bound of $O(L/\varepsilon^2)$ to achieve ε-approximation. For these reasons we use gradient descent with a learning rate proportional to $1/t$ in our experiments described in Section 8.

5. Empirical Capacity of GLNs

Contemporary neural networks have the desirable capability to approximate arbitrary continuous functions given almost any reasonable activation function [Hor91, and others]. GLNs share this property so long as the context capacity is sufficiently expressive. Moreover, [VLB+17] prove that this capacity is *effective* in the sense that gradient descent will eventually find the best feasible approximation. This property is not shared by neural networks trained by backpropagation; it is possible to demonstrate the existence of such weights, but not to guarantee that gradient descent (or any other practical algorithm) will find them. Here we demonstrate the capacity of GLNs in practice by measuring their ability to fit random labelled data.

We ran two sets of experiments: first, using the standard MNIST dataset with shuffled labels; and second, replacing the MNIST images with uniform noise of the same shape and dataset length. These results are presented in Figure 3 compared to an MLP baseline in an equivalent one-vs-all configuration. For GLNs, we select a fixed layer width of 128 and vary both the context dimension and number of layers. For the MLP, we select the number of *neurons* such that the total number of *weights* in the network is equivalent to a GLN with context dimension 4 (the largest considered). The GLN was trained with learning rate 10^{-4} and the MLP using the Adam optimizer [KB14] with learning rate 10^{-5},

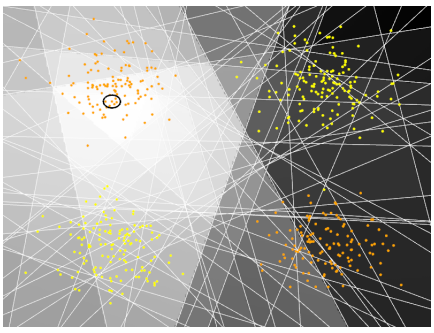

Figure 4. The effect of a single noisy XOR update (circled) on the decision boundaries of a halfspace gated GLN. Sampled hyperplanes for each gate are shown in white.

both selected by conducting a sweep over learning rates from 10^{-1} to 10^{-6}. It is evident from Figure 3 that GLNs have comparable capacity to an equivalently sized MLP in practice, with their ability to memorize training data scaling in both the number of neurons and context dimension.

6. Linear Interpretability of GLNs

In the case where we have an L layer GLN, with K_i neurons on layer i, and with input p_0 of dimension K_0 and side information z, the RHS of Equation 5 can be written as

$$\sigma\Big(\underbrace{W_L(z)\,W_{L-1}(z)\,\ldots\,W_1(z)}_{\text{multilinear polynomial of degree L}}\,\text{logit}(p_0)\Big),$$

where each matrix $W_i(z)$ is of dimension $K_i \times K_{i-1}$, with the jth row constituting the active weights (as determined by the gating) for the jth neuron in layer i. This formulation is convenient for many reasons. First, it allows us to reason about the inductive bias of GLNs by observing that the product of matrices collapses to a multilinear polynomial in the learnt weights, i.e. the depth and shape of the network directly influences how a GLN will generalize. A visual example of the change in decision boundaries resulting from a single halfspace gated GLN update is shown in Figure 4 for the noisy XOR problem. The magnitude of the change is largest within the convex polytope containing the training point, and decays with respect to the remaining convex polytopes according to how many halfspaces they share with the containing convex polytope. This makes intuitive sense, as since the weight update is local, each row of $W_i(z)$ is pushed in the direction to better explain the data independently of each other. Therefore one should think of a halfspace gated GLN as a smoothing technique – input points which cause similar gating activation patterns must have similar outputs.

Aside from reasoning about the inductive bias, the above formulation provides a convenient mechanism for interpreting the learnt weights of a trained GLN. Contemporary neural networks have been criticized by some as "black boxes" that are notoriously difficult to interpret [YCN+15, ZZ18]. Despite their high discrimination power, this can prove prob-

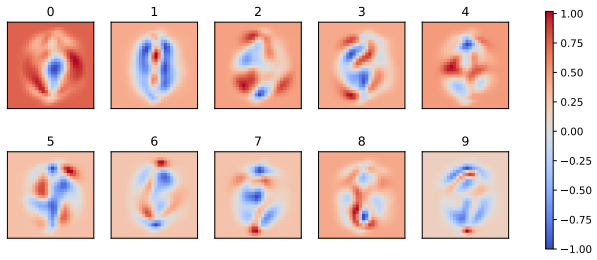

Figure 5. Saliency maps for constituent GLN binary classifiers of one-vs-all MNIST classifier after a single training epoch.

lematic for learning and debugging efficiently at the semantic level as well as for deployment in safety-critical real-world applications. This has led to the development of gradient-based methods for post-hoc network analysis [SVZ13]. Such methods are not necessary for GLNs; for a given input, the collapsed multilinear polynomial of degree L is a weight vector of the same dimension (since $W_L(z)$ has 1 row and $W_1(z)$ has K_0 columns) as the inputs and provides a natural formulation for intuitive saliency maps without any further computational expense. An example of the obtained saliency maps are provided in Figure 5 for a one-versus-all GLN trained as an MNIST classifier. One can clearly see that the characteristic shape of each hand-written character is preserved.

7. Resilience to Catastrophic Forgetting

Humans are able to acquire new skills throughout life seemingly without compromising their ability to solve previously learnt tasks. Contemporary neural networks do not share this ability; if a network is trained on a task A and these weights are used to initialize training for a new task B, the ability to solve A rapidly degrades as training progresses on B. This phenomenon of "catastrophic forgetting" has been well studied for decades [CG88, MC89, Rob95] but continues to limit the applicability of neural networks in continual or lifelong learning scenarios.

Similar to the problem of model interpretability, many algorithms have been developed that augment standard training by backpropagation to address catastrophic forgetting. These methods typically fall into two main categories. The first approach involves replaying previously seen tasks during training using one of many heuristics [Rob95, Car97, RKSL17]. The other common category involves explicitly maintaining additional sets of model parameters related to previously learnt tasks. Examples include freezing a subset of weights [DJV+13, RASC14], dynamically adjusting learning rates [GDDM14] or augmenting the loss with regularization terms with respect to past parameters [KPR+17, ZPG17, SLC+18]. A limitation of these approaches (aside from additional algorithmic and computational complexity) is that they require task boundaries to be provided or accurately inferred.

Unlike contemporary neural networks, we demonstrate that the halfspace-gated GLN architecture and learning rule is naturally robust to catastrophic forgetting without any modifications or knowledge of task boundaries. We focus on the pixel-permuted MNIST continual learning benchmark of [GMX+13, KPR+17], which involves training on a sequence of different tasks where each task is obtained from a different random permutation of the input pixels. We compare the learning and retention characteristics of a GLN against an MLP baseline (of equal number of neurons, using dropout as per the original paper) with and without elastic weight consolidation (EWC) [KPR+17], which is a highly-effective method explicitly designed to prevent catastrophic forgetting by storing parameters of previously seen tasks.

Our results are presented in Figure 6. As we train our models on a growing number of sequential tasks (rows), the performance on all previously learnt tasks (columns) is evaluated. Note that the plotted task indices are not contiguous. It is evident that the GLN outperforms EWC in terms of both initial single-task learning (diagonal) and retention when both are trained for a single pass. Only when EWC is trained for multiple (ten) passes over the data does it exhibit superior performance to a vanilla GLN. In all tests, the GLN substantially outperforms the standard MLP without EWC.

To gain some intuition as to why GLNs are resilient to catastrophic interference, recall from Section 4 that inputs close in terms of cosine similarity will give rise to similar data dependent weight matrices. Since each task-specific cluster of examples is far from each other in signature space, the amount of interference between tasks is significantly reduced, with the gating essentially acting as an implicit weight hashing mechanism.

Quite clever, but this is not alway

8. Online Benchmarking

MNIST Classification. First we explore the use of GLNs for online (single-pass) classification of the deskewed MNIST dataset [LBBH98, GW17]. We use 10 GLNs to construct a one-vs-all classifier, each consisting of 128 neurons per layer with context dimension 4. The learning rate at each step t was set to $\min\{100/t, 0.01\}$. We find that the GLN is capable of impressive online performance, achieving 98% accuracy in a single pass of the training data.

UCI Dataset Classification. We next compare GLNs to a variety of general purpose batch learning techniques (SVMs, Gradient Boosting for Classification, MLPs) in small data regimes on a selection of standard UCI datasets. A 1000-500 neuron GLN with context-dimension 8 was trained with a *single pass* over 80% of instances and evaluated with frozen weights on the remainder. The comparison MLP used ReLU activations and the same number of weights, and was trained for *100 epochs* using the Adam

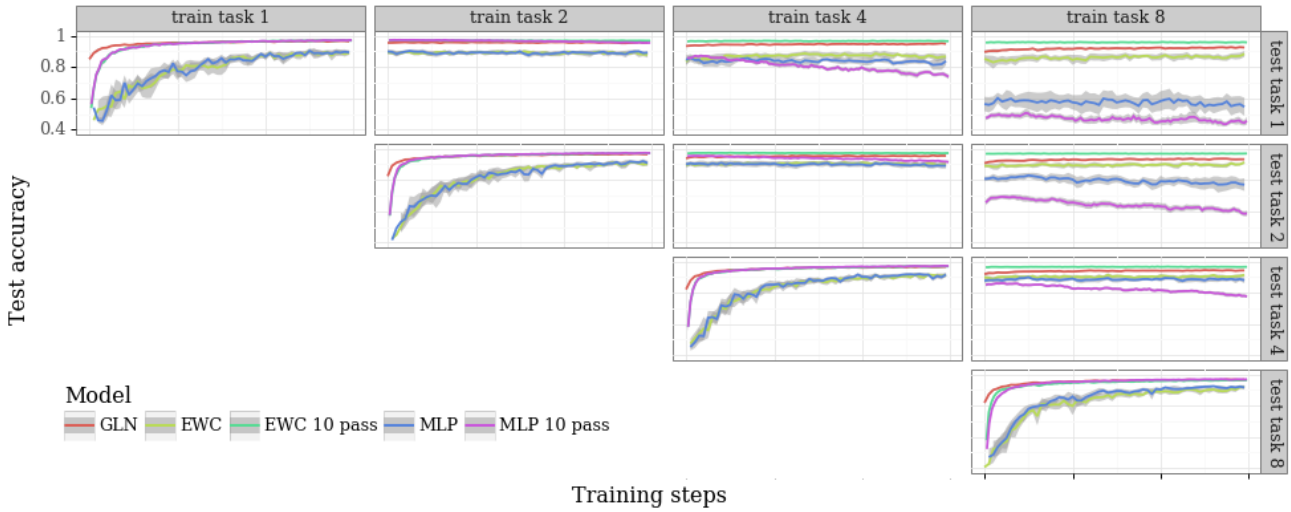

Figure 6. Retention results for permuted MNIST. Models are trained sequentially on 8 tasks (rows) and evaluated on all previously encountered tasks (columns). For example, the top-right plot indicates performance on Task 1 after being trained sequentially on Tasks 1 to 8 inclusive (not all tasks shown). Each model only trains for one epoch per task, with the exception of "EWC 10 pass" and "MLP 10 pass" (shrunken 10-fold on x axis). Error bars denote 95% confidence levels over 10 random seeds.

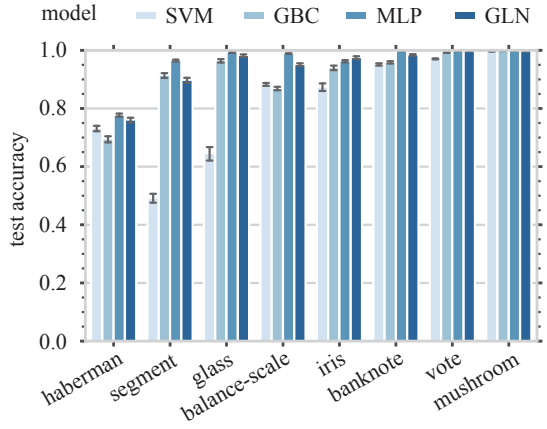

Figure 7. Online (single-pass) GLN classification accuracy on a selection UCI datasets, compared to three contemporary batch methods (Support Vector Machine, Gradient Boosting for Classification, Multi-Layer Perceptron) trained for 100 epochs.

optimizer [KB14] with learning rate 0.001 and batch size 32. The SVM classifier used a radial basis function kernel $K(x, x') = \exp\{-\gamma \|x - x'\|^2\}$ with $\gamma = 1/d$, where d is the input dimension. The GBC classifier was an ensemble of 100 trees of maximum depth 3 with a learning rate of 0.1. The mean and stderr over 100 random train/test splits are shown in the leftmost graph of Figure 7. Here we see that the single-pass GLN is competitive with the best of the batch learning results on each domain.

MNIST Density Modelling. Our final result is to use GLNs and image specific gating to construct an online image density model for the binarized MNIST dataset [LM11], a standard benchmark for image density modeling. By exploiting the chain rule $\mathbb{P}(X_{1:d}) = \prod_{i=1}^{d} \mathbb{P}(X_i \,|\, X_{<i})$ of probability, we constructed an autoregressive density model over the 28×28 dimensional binary space by using 784 GLNs to model the conditional distribution for each pixel; a row-major ordering was used to linearize the two dimensional pixel locations. Running our method online (i.e. a single pass of the concatenated training, validation and test sets) gave an average loss of 79.0 nats per image across the test data, and 80.74 nats per image if we held the parameters fixed upon reaching the test set. These results are close to state of the art [VDOKK16] of any batch trained density model which outputs exact probabilities.

From an MDL or compression perspective, our density modelling results are significantly stronger in the sense that we could couple our model to an adaptive arithmetic decoder and reproduce the original data from a file much smaller than the original input. Contemporary batch trained density models do not have this property; to make a fair comparison, they would need to first encode the parameters of the model before encoding the subsequent compressed data, and state-of-the-art batch train density models typically have compressed size many orders of magnitude larger than the original data.

9. Conclusion

We have introduced a new family of general purpose neural architectures, Gated Linear Networks, and studied the desirable characteristics that follow from their use of data-dependent gating and local credit assignment. Their fast online learning properties, easy interpretability, and excellent robustness to catastrophic forgetting in continual learning settings makes them an interesting and complementary alternative to contemporary deep learning approaches.

References

[B+15] Sébastien Bubeck et al. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

[BFH+18] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018.

[BH89] P. Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58, January 1989.

[BHQK20] David Budden, Matteo Hessel, John Quan, and Steven Kapturowski. RLax: Reinforcement Learning in JAX, 2020.

[Car97] Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, Jul 1997.

[CG88] G. A. Carpenter and S. Grossberg. The art of adaptive pattern recognition by a self-organizing neural network. *Computer*, 21(3):77–88, March 1988.

[Cha02] M.S. Charikar. Similarity estimation techniques from rounding algorithms. *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 380–388, 01 2002.

[Cut19] Ashok Cutkosky. Anytime Online-to-Batch, optimism and acceleration. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1446–1454, Long Beach, California, USA, June 2019. PMLR.

[DJV+13] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *CoRR*, abs/1310.1531, 2013.

[GDDM14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun 2014.

[GMX+13] Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks, 2013.

[GW17] Dibya Ghosh and Alvin Wan, 2017. https://fsix.github.io/mnist/

[HAK07] Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69:169–192, 2007.

[Haz16] Elad Hazan. Introduction to online convex optimization. *Foundations and Trends in Optimization*, 2(3-4):157–325, 2016.

[HCNB20] Tom Hennigan, Trevor Cai, Tamara Norman, and Igor Babuschkin. Haiku: Sonnet for JAX, 2020.

[Hin02] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, August 2002.

[Hor91] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Kno17] Byron Knoll, 2017. http://www.byronknoll.com/cmix.html

[KPR+17] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[LBBH98] Yann Lecun, Lon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[LM11] Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In Geoffrey Gordon, David Dunson, and Miroslav Dudk, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 29–37, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[Mah00] Matthew Mahoney. Fast text compression with neural networks. *AAAI*, 2000.

[Mah05] Matthew Mahoney. Adaptive weighing of context models for lossless data compression. *Technical Report, Florida Institute of Technology CS*, 2005.

[Mah13] Matthew Mahoney. *Data Compression Explained*. Dell, Inc, 2013.

[Mat12] Christopher Mattern. Mixing strategies in data compression. In *2012 Data Compression Conference, Snowbird, UT, USA, April 10-12*, pages 337–346, 2012.

[Mat13] Christopher Mattern. Linear and geometric mixtures - analysis. In *2013 Data Compression Conference, DCC 2013, Snowbird, UT, USA, March 20-22, 2013*, pages 301–310, 2013.

[MC89] Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109 – 165. Academic Press, 1989.

[MP69] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[OWR+19] Pedro A. Ortega, Jane X. Wang, Mark Rowland, Tim Genewein, Zeb Kurth-Nelson, Razvan Pascanu, Nicolas Heess, Joel Veness, Alexander Pritzel, Pablo Sprechmann, Siddhant M. Jayakumar, Tom McGrath, Kevin Miller, Mohammad Gheshlaghi Azar, Ian Osband, Neil C. Rabinowitz, András György, Silvia Chiappa, Simon Osindero, Yee Whye Teh, Hado van Hasselt, Nando de Freitas, Matthew Botvinick, and Shane Legg. Meta-learning of sequential strategies. *CoRR*, abs/1905.03030, 2019.

[RASC14] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: An astounding baseline for recognition. *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, Jun 2014.

[RKSL17] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017.

[Rob95] Anthony V. Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connect. Sci.*, 7:123–146, 1995.

[SLC+18] Jonathan Schwarz, Jelena Luketina, Wojciech M. Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning, 2018.

[SMG13] Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *CoRR*, abs/1312.6120, 2013.

[SVZ13] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[SZ13] Ohad Shamir and Tong Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In *International conference on machine learning*, pages 71–79, 2013.

[VDOKK16] Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1747–1756. JMLR.org, 2016.

[VLB+17] Joel Veness, Tor Lattimore, Avishkar Bhoopchand, Agnieszka Grabska-Barwinska, Christopher Mattern, and Peter Toth. Online learning with gated linear networks. *arXiv preprint arXiv:1712.01897*, 2017.

[YCN+15] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[Zin03] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 928–936, 2003.

[ZPG17] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML17, page 39873995. JMLR.org, 2017.

[ZZ18] Quan-shi Zhang and Song-Chun Zhu. Visual interpretability for deep learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, 19(1):27–39, 2018.

Additional Experiment Details

Permuted-MNIST

This section describes the implementation details and hyper-parameters for each experiment.

GLN. We use one-vs-all GLNs composed of $[100, 25, 1]$ neurons per layer and a context dimension of 6. The output of the network is determined by the last neuron.

MLP and EWC. We use a ReLU network with $[1000, 250, 10]$ neurons per layer. We use Adam [KB14] to optimize the cross-entropy loss using mini-batches of 20 data points. For EWC, we draw 100 samples for computing the Fisher matrix diagonals. We also optimize the constant that trades-off remembering and learning via grid-search, and denote it with λ in Table 1.

Model	log-learning rate	dropout	$\log \lambda$
GLN	-4, -3, **-2**, -1	–	–
MLP	-6, -5, **-4**, -3	Yes, **No**	–
EWC	-6, -5, **-4**, -3	Yes, **No**	2, **3**, 4

Table 1. Parameters explored during grid search. The best parameters (shown in bold) are the ones that maximize the average accuracy over 10 random seeds. Logarithms are in base 10.

MNIST saliency map experimental details

We use one-vs-all GLNs composed of $[50, 25, 1]$ neurons per layer and a context dimension of 4. The output of the network is determined by the last neuron. Constant learning rate of 10^{-2} is used to pass though the training data once. Learned weights for bias term are dropped from analysis.

Computing Infrastructure

All experiments were ran on single-GPU desktop PCs. Models are implemented in JAX [BFH+18] making use of Haiku [HCNB20] and RLax [BHQK20].