nature machine intelligence

Article

https://doi.org/10.1038/s42256-024-00902-x

# Engineering flexible machine learning systems by traversing functionally invariant paths

Received: 8 October 2022

Accepted: 19 August 2024

Published online: 3 October 2024

Check for updates

Guruprasad Raghavan[1,2] ✉, Bahey Tharwat[3], Surya Narayanan Hari[1], Dhruvil Satani[1], Rex Liu[1] & Matt Thomson[1] ✉

Contemporary machine learning algorithms train artificial neural networks by setting network weights to a single optimized configuration through gradient descent on task-specific training data. The resulting networks can achieve human-level performance on natural language processing, image analysis and agent-based tasks, but lack the flexibility and robustness characteristic of human intelligence. Here we introduce a differential geometry framework—functionally invariant paths—that provides flexible and continuous adaptation of trained neural networks so that secondary tasks can be achieved beyond the main machine learning goal, including increased network sparsification and adversarial robustness. We formulate the weight space of a neural network as a curved Riemannian manifold equipped with a metric tensor whose spectrum defines low-rank subspaces in weight space that accommodate network adaptation without loss of prior knowledge. We formalize adaptation as movement along a geodesic path in weight space while searching for networks that accommodate secondary objectives. With modest computational resources, the functionally invariant path algorithm achieves performance comparable with or exceeding state-of-the-art methods including low-rank adaptation on continual learning, sparsification and adversarial robustness tasks for large language models (bidirectional encoder representations from transformers), vision transformers (ViT and DeIT) and convolutional neural networks.

Artificial neural networks now achieve human-level performance on machine learning tasks ranging from natural language understanding and image recognition to game playing and protein structure prediction. Recently, transformer-based models with self-attention have emerged as the state-of-the-art architecture across data modalities and task paradigms including natural language understanding, computer vision, audio processing, biological sequence analysis and context-sensitive reasoning[1–4]. Although transformer models can exhibit emergent behaviours including zero-shot task performance, models are commonly fine-tuned to increase performance and human accessibility[4–6]. In the 'foundation model' paradigm[3], transformers with $10^8$ to $10^{12}$ parameters are, first, pre-trained over large datasets on self-supervised tasks such as masked language modelling, causal language modelling or image masking[5–8]. Following self-supervised training, models are fine-tuned to increase performance on specific applications including question/answer or instruction following.

[1]Department of Biology and Biological Engineering, California Institute of Technology, Pasadena, CA, USA. [2]Yurts AI, San Francisco, CA, USA. [3]Alexandria University, Alexandria, Egypt. ✉e-mail: guru@yurts.ai; mthomson@caltech.edu

Networks can be further sparsified or quantized to reduce memory and computation requirements in deployment environments.

Due to the central role of model adaptation for foundation model optimization and deployment, many algorithms have emerged for updating model weights to increase performance without experiencing a catastrophic loss of the knowledge gained during self-supervised pre-training. However, a challenge is that in artificial neural networks, network function is encoded in the mathematical weights that determine the strength of connections between neural units (Fig. 1a,b). Gradient descent procedures train networks to solve problems by adjusting the weights of a network based on an objective function that encodes the performance of a network on a specific task. Learning methods, like backpropagation and low-rank adaptation (LoRA)[9] gradient descent, adjust the network weights to define a single, optimal weight configuration to maximize performance on a task-specific objective function using training data. However, for all the current methods, network training alters network weights, inevitably resulting in the loss of information gained from previous training tasks or pre-training.

Although many methods have been developed to achieve network adaptation without information loss, methods remain primarily grounded in empirical results. The machine learning community would benefit from mathematical tools that provide general insights and unification of model adaptation strategies within a common theoretical framework. Methods like LoRA, orthogonal gradient descent (OGD), relevance mapping networks (RMNs) and elastic weight consolidation (EWC) propose different criteria for updating weights in directions that do not impact performance on previously learned tasks. Yet, most current methods are based on local heuristics, for example, selecting gradient steps that are orthogonal to gradient steps taken for previously learned tasks. As in continual learning (CL) paradigms, sparsification frameworks execute heuristic prune/fine-train cycles to discover a compact, core subnetwork capable of executing the desired behaviour with decreased memory, power and computational requirements. Mathematical tools that provide a deeper insight into how the global, geometric structure of weight space enables or complicates adaptation might provide both conceptual principles and new algorithms.

Unlike contemporary artificial neural nets, neural networks in the human brain perform multiple functions and can flexibly switch between different functional configurations based on context, goals or memory[10]. Neural networks in the brain are hypothesized to overcome the limitations of a single, optimal weight configuration and perform flexible tasks by continuously 'drifting' their neural firing states and neural weight configurations, effectively generating large ensembles of degenerate networks[11–15]. Fluctuations might enable flexibility in biological systems by allowing neural networks to explore a series of network configurations and responding to sensory input.

Here we develop a geometric framework and algorithm that mimics aspects of biological neural networks by using differential geometry to construct path-connected sets of neural networks that solve a given machine learning task. Conceptually, we consider path-connected sets of neural networks, rather than single networks (isolated points in weight space) to be the central objects of study and application. By building sets of networks rather than single networks, we search within a submanifold of weight space for networks that solve a given machine learning problem and accommodate a broad range of secondary goals. Historically, results in theoretical machine learning and information geometry have pointed to the geometry of a model's loss landscape as a potential resource for model adaptation. An emergent property of over-parameterized models is the existence of parameter degeneracy where multiple settings of the model parameters can achieve identical performance on a given task. Geometrically, parameter degeneracy leads[16] to 'flat' objective functions[17–19] along which network weights can be modified without loss of performance. To discover invariant subspaces, we introduce a Riemannian metric into weight space—a tensor that measures (at every point in parameter space) the change in the model output given an infinitesimal movement in model parameters. The metric provides a mathematical tool for identifying low-dimensional subspaces in weight space where parameter changes have little impact on how a neural network transforms input data. Riemannian metrics are widely used in physical theories to study the dynamics of particles on curved manifolds and spacetimes. Geometrically, the metric discovers flat directions in weight space along which we can translate a neural network without changing functional performance.

Using the Riemannian weight-space metric, we develop an algorithm that constructs functionally invariant paths (FIPs) in weight space that maintain network performance and 'search out' for other networks that satisfy additional objectives. The algorithm identifies long-range paths in weight space that can integrate new functionality without information loss. We apply the FIP framework to natural language (bidirectional encoder representations from transformers (BERT)), vision transformers (ViT and DeIT) and convolutional neural network (CNN) architectures. Our framework generates results that meet or exceed state-of-the-art performance for adaptation and sparsification tasks on academic-grade hardware. Our approach provides mathematical machinery that yields insights into how low-dimensional geometric structures can be harnessed for model adaptation without information loss. More broadly, we consider language models as objects acted on by transformations in series and we show that the transformations of models are intrinsically non-Abelian. The unified framework provides a general attack on a range of model adaptation problems and reveals connections between the mathematical theory of differential geometries and the emergent properties of large language and vision models.

## Riemannian metric: mathematical framework

We develop a mathematical framework that allows us to define and explore path-connected sets of neural networks that have divergent weight values but similar outputs on training data. We view the weight space of a neural network as a Riemannian manifold equipped with a local distance metric[20,21]. Using differential geometry, we construct paths through weight space that maintain the functional performance of a neural network and adjust the network weights to flow along a secondary goal (Fig. 1a). The secondary goal can be general; therefore, the framework can be applied to train networks on new classification tasks, sparsify networks and mitigate adversarial fragility.

The defining feature of a Riemannian manifold is the existence of a local distance metric. We construct a distance metric in weight space that defines the distance between two nearby networks to be their difference in output. We consider a neural network to be a smooth function $f(\mathbf{x};\mathbf{w})$ that maps an input vector $\mathbf{x} \in \mathbb{R}^k$ to an output vector $f(\mathbf{x};\mathbf{w}) = \mathbf{y} \in \mathbb{R}^m$, where the map is parameterized by a vector of weights $\mathbf{w} \in \mathbb{R}^n$ that are typically set in training to solve a specific task. We refer to $W = \mathbb{R}^n$ as the weight space of the network, and we refer to $\mathcal{Y} = \mathbb{R}^m$ as the output space (Fig. 1b,c)[22]. For pedagogical purposes, we will consider the action of $f$ on a single input $\mathbf{x}$. Supplementary Note 1 shows that our results naturally extend to an arbitrary number of inputs $\mathbf{x_i}$.

We initially ask how the output $f(\mathbf{x};\mathbf{w})$ of a given neural network changes for small changes in network weights. Given a neural network with weights $\mathbf{w_t}$ and fixed input $\mathbf{x}$, we can compute the output of the perturbed network $\mathbf{w_t} + \mathbf{dw}$ for an infinitesimal weight perturbation $\mathbf{dw}$ as follows:

$$f(\mathbf{x}, \mathbf{w_t} + \mathbf{dw}) \approx f(\mathbf{x}, \mathbf{w_t}) + \mathbf{J_{w_t}}\,\mathbf{dw}, \tag{1}$$

where $\mathbf{J_{w_t}}$ is the Jacobian of $f(\mathbf{x}, \mathbf{w_t})$ for a fixed $\mathbf{x}$, $J_{ij} = \frac{\partial f_i}{\partial w_j}$, evaluated at $\mathbf{w_t}$.

Thus, the total change in network output for a given weight perturbation $\mathbf{dw}$ is

https://doi.org/10.1038/s42256-024-00902-x

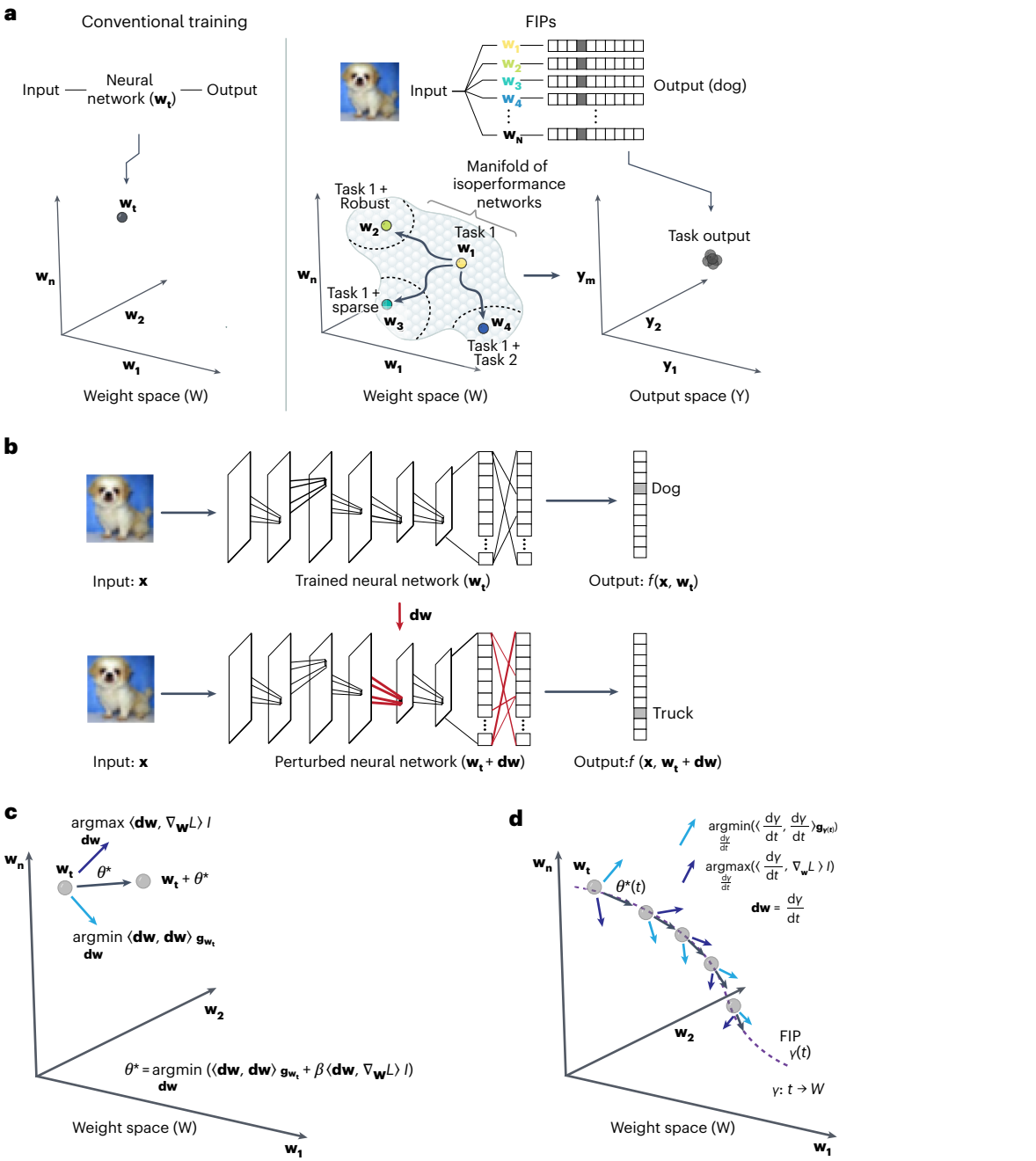



**Fig. 1 | Differential geometric framework for constructing FIPs in weight space. a**, Left: conventional training on a task finds a single trained network ($\mathbf{w_t}$) solution. Right: the FIP strategy discovers a submanifold of isoperformance networks ($\mathbf{w_1}$, $\mathbf{w_2}$...$\mathbf{w_N}$) for a task of interest, enabling the efficient search for networks endowed with adversarial robustness ($\mathbf{w_2}$), sparse networks with high task performance ($\mathbf{w_3}$) and for learning multiple tasks without forgetting ($\mathbf{w_4}$). **b**, Top: a trained CNN with weight configuration ($\mathbf{w_t}$), represented by lines connecting different layers of the network, accepts an input image $\mathbf{x}$ and produces a ten-element output vector, $f(\mathbf{x}, \mathbf{w_t})$. Bottom: perturbation of network weights by $\mathbf{dw}$ results in a new network with weight configuration $\mathbf{w_t} + \mathbf{dw}$ with an altered output vector, $f(\mathbf{x}, \mathbf{w_t} + \mathbf{dw})$, for the same input, $\mathbf{x}$. **c**, The FIP algorithm identifies weight perturbations $\boldsymbol{\theta}^*$ that minimize the distance moved in output space and maximize alignment with the gradient of a secondary objective function ($\nabla_{\mathbf{w}} L$). The light-blue arrow indicates an $\epsilon$-norm weight perturbation that minimizes distance moved in output space and the dark-blue arrow indicates an $\epsilon$-norm weight perturbation that maximizes alignment with the gradient of the objective function, $L(\mathbf{x}, \mathbf{w})$. The secondary objective function $L(\mathbf{x}, \mathbf{w})$ is varied to solve distinct machine learning challenges. **d**, Path sampling algorithm defines FIPs, $\gamma(t)$, through the iterative identification of $\epsilon$-norm perturbations ($\boldsymbol{\theta}^*(t)$) in the weight space.

$$\begin{aligned} |f(\mathbf{x}, \mathbf{w_t} + \mathbf{dw}) - f(\mathbf{x}, \mathbf{w_t})|^2 &= \mathbf{dw}^{\mathrm{T}} \left( \mathbf{J}_{\mathbf{w_t}}(\mathbf{x})^{\mathrm{T}} \mathbf{J}_{\mathbf{w_t}}(\mathbf{x}) \right) \mathbf{dw} \\ |\langle \mathbf{dw}, \mathbf{dw} \rangle_{\mathbf{g_{w_t}}}|^2 &= \mathbf{dw}^{\mathrm{T}}\, \mathbf{g}_{\mathbf{w_t}}(\mathbf{x})\, \mathbf{dw} \end{aligned}, \quad (2)$$

where $\mathbf{g}_{\mathbf{w_t}}(\mathbf{x}) = \mathbf{J}_{\mathbf{w_t}}(\mathbf{x})^{\mathrm{T}} \mathbf{J}_{\mathbf{w_t}}(\mathbf{x})$ is the metric tensor evaluated at the point $\mathbf{w_t} \in W$ for a single data point $\mathbf{x}$. The metric tensor is an $n \times n$ symmetric matrix that allows us to compute the change in network output given the movement of the network along any direction in weight space as $\langle \mathbf{dw}, \mathbf{dw} \rangle_{\mathbf{g_{w_t}}(\mathbf{x})}$. As we move through the weight space, the metric tensor $W$ continuously changes, allowing us to compute the infinitesimal change in network output and move along a path $\boldsymbol{\gamma}(\mathbf{t})$ as the tangent vector $\psi(t) = \frac{\mathrm{d}\gamma(t)}{\mathrm{d}t}$.

We can extend the metric construction to cases in which we consider a set of $N$ training data points $\mathbf{X}$ and view $\mathbf{g}$ as the mean of the metrics derived from individual training examples, such that $\mathbf{g}_{\mathbf{w}(\mathbf{X})} = \Sigma_{i=1}^{N} \mathbf{g}_{\mathbf{w}}(\mathbf{x_i})/N$ for $x_i \in \mathbf{X}$ or in expectation, $\mathbf{g}_{\mathbf{w}} = \mathbb{E}_{x \approx p_{\text{data}}(x)}[\mathbf{g}_{\mathbf{w}(\mathbf{x})}]$, and $\mathbf{g}_{\mathbf{w}(\mathbf{X})}$ remains $n \times n$ (Supplementary Note 1). For a single data point, our construction of $\mathbf{g}_{\mathbf{w}}$ is identical to the neural tangent kernel (NTK), which is constructed as a kernel function of pairs of data points, $\Theta(x_i, x_j) = J(x_i)^{\mathrm{T}} J(x_j)$ (ref. 23), so that $\Theta(x_i, x_i) = g_i(x_i)$ (refs. 23–26). However, we distinguish our interpretation of $\mathbf{g}$ as a distance metric on $W$, the network parameter space from the NTK as a matrix-valued kernel function on data points $x_i$ and $x_j$. The NTK $\Theta(x_i, x_j)$ arises through the analysis of the dynamics of a neural network under a gradient flow of the mean squared error. The NTK is interpreted as a kernel function defined on pairs of data points $x_i$ and $x_j$, providing an analogy between the training dynamics of a neural network and kernel-based learning methods. Alternately, we interpret $\mathbf{g}_{\mathbf{w}}$ as a metric on a Riemannian parameter manifold $(W, \mathbf{g}_{\mathbf{w}})$. At each point in the weight space, the metric defines the length, $\langle \mathbf{dw}, \mathbf{dw} \rangle_{\mathbf{g}_{\mathbf{w}}}$, of a local perturbation to network weights $\mathbf{dw}$ as the perturbation's impact on the functional output of the network (Fig. 1b,c) averaged over all the data points in $\mathbf{X}$. The metric motivates the construction and analysis of network paths in weight space and consideration of properties including velocity, acceleration and notion of a geodesic path in weight space.

At every point in the weight space, the metric allows us to discover directions $\mathbf{dw}$ of movement that have a large or small impact on the output of a network. As we move along a path $\gamma(t) \subset W$ in weight space, we sample a series of neural networks over time $t$. Using the metric, we can define a notion of 'output velocity' as $\mathbf{v} = \frac{\mathrm{d}f(\mathbf{x}, \gamma(t))}{\mathrm{d}t}$, which quantifies the distance a network moves in output space for each local movement along the weight-space path $\gamma(t)$. We seek to identify FIPs in weight space along which the output velocity is minimized for a fixed magnitude change in weight. To do so, we solve the following optimization problem:

$$\psi^*(t) = \underset{\frac{\mathrm{d}\gamma}{\mathrm{d}t}}{\operatorname{argmin}} \left\langle \frac{\mathrm{d}\gamma}{\mathrm{d}t}, \frac{\mathrm{d}\gamma}{\mathrm{d}t} \right\rangle_{\mathbf{g}_{\gamma(\mathbf{t})}}, \quad \text{such that } \left\langle \frac{\mathrm{d}\gamma}{\mathrm{d}t}, \frac{\mathrm{d}\gamma}{\mathrm{d}t} \right\rangle_{I} = \epsilon \tag{3}$$

where we attempt to find a direction $\psi^*(t)$ along which to perturb the network, such that it is $\epsilon$ units away from the base network in the weight space (in the Euclidean sense, $\langle \frac{\mathrm{d}\gamma}{\mathrm{d}t}, \frac{\mathrm{d}\gamma}{\mathrm{d}t} \rangle_{I} = \epsilon$) and minimize the distance moved in the networks' output space, given by $\langle \frac{\mathrm{d}\gamma}{\mathrm{d}t}, \frac{\mathrm{d}\gamma}{\mathrm{d}t} \rangle_{\mathbf{g}_{\gamma}(\mathbf{t})}$. Here $I$ is an identity matrix, with the inner product $\langle \frac{\mathrm{d}\gamma}{\mathrm{d}t}, \frac{\mathrm{d}\gamma}{\mathrm{d}t} \rangle_{I}$ capturing the Euclidean distance in weight space[27]. The optimization problem is a quadratic program at each point in weight space. The metric $\mathbf{g}$ is a matrix that takes on a specific value at each point in weight space, and we aim to identify vectors $\boldsymbol{\psi}^*(t) = \frac{\mathrm{d}\gamma(t)}{\mathrm{d}t}$ that minimize the change in the functional output of the network.

We will often amend the optimization problem with a second objective function $L(\mathbf{x}, \mathbf{w})$. We can enumerate paths that minimize the functional velocity in the output space and move along the gradient of the second objective ($\nabla_{\mathbf{w}} L$). We define a path-finding algorithm that identifies a direction $\psi^*(t)$ in the weight space by minimizing the functional velocity in the output space and moves along the gradient of the second objective ($\nabla_{\mathbf{w}} L$):

$$\psi^*(t) = \underset{\frac{\mathrm{d}\gamma}{\mathrm{d}t}}{\operatorname{argmin}} \left( \left\langle \frac{\mathrm{d}\gamma}{\mathrm{d}t}, \frac{\mathrm{d}\gamma}{\mathrm{d}t} \right\rangle_{\mathbf{g}_{\gamma(t)}} + \beta \left\langle \frac{\mathrm{d}\gamma}{\mathrm{d}t}, \nabla_{\mathbf{w}} L \right\rangle_{I} \right), \quad \text{such that } \left\langle \frac{\mathrm{d}\gamma}{\mathrm{d}t}, \frac{\mathrm{d}\gamma}{\mathrm{d}t} \right\rangle_{I} = \epsilon \tag{4}$$

where the first term $\langle \frac{\mathrm{d}\gamma}{\mathrm{d}t}, \frac{\mathrm{d}\gamma}{\mathrm{d}t} \rangle_{\mathbf{g}_{\gamma(t)}}$ identifies functionally invariant directions, the second term $\langle \frac{\mathrm{d}\gamma}{\mathrm{d}t}, \nabla_{\mathbf{w}} L \rangle_{I}$ biases the direction of motion along the gradient of a second objective and $\beta$ weighs the relative contribution of the two terms. When $L = 0$, the algorithm merely constructs paths in weight space that are approximately isofunctional ($\theta^*(t) = \psi^*(t)$), that is, the path is generated by steps in the weight space comprising networks with different weight configurations and preserving the input–output map. $L(\mathbf{x}, \mathbf{w})$ can also represent the loss function of a second task, for example, a second input classification problem. In this case, we identify vectors that simultaneously maintain performance on an existing task (via term 1) as well as improve performance on a second task by moving along the negative gradient of the second-task loss function $\nabla_{\mathbf{w}} L$. We consider constructing FIPs with different objective functions ($L(\mathbf{x}, \mathbf{w})$) similar to applying different 'operations' to neural networks that identify submanifolds in the weight space of the network accomplishing distinct tasks of interest.

To approximate the solution to equation (4), in large neural networks, we developed a numerical strategy that samples points in an $\epsilon$ ball around a given weight configuration, and then performs gradient descent to identify vectors $\boldsymbol{\theta}^*(t)$. We note that the performance of a neural network on a task is typically evaluated using a loss function $L : \mathbb{R}^m \to \mathbb{R}$, so that $L(f(\mathbf{x}; \mathbf{w})) \in \mathbb{R}$. Networks with a constant functional output $f(x, w)$ along a path $\gamma(t)$ will also have constant loss $L(f(\gamma(t); w))$. As gradient descent training minimizes the loss by finding $\frac{\partial L}{\partial w_i} = 0$, the evaluation of loss curvature requires second-order methods to discover flat or functionally invariant subspaces. We find that working directly with $f(x; w)$ allows us to identify functionally invariant subspaces through first-order quantities $\frac{\partial f_i}{\partial w_j} = J_{ij}$, and thus, we can compute the metric using the output of automatic differentiation procedures commonly used in training. Working with $f(\mathbf{x}, w)$ instead of $L$ also provides additional resolution in finding invariant subspaces since $f(x; w)$ is often a vector-valued versus scalar-valued function.

We note that the mathematical framework provides avenues for immediate generalization to consider paths of constant velocity, paths that induce a constant rate of performance change; such paths are known as geodesics[28]. On any Riemannian manifold, we can define geodesic paths emanating from a point $x_{\mathrm{p}}$ as paths of constant velocity $\langle \frac{\mathrm{d}\gamma}{\mathrm{d}t}, \frac{\mathrm{d}\gamma}{\mathrm{d}t} \rangle_{\mathbf{g}_{\gamma(t)}} = v_0$ (Supplementary Note 4.1) that satisfy the following geodesic equation:

$$\frac{\mathrm{d}^2 w_\eta}{\mathrm{d}t^2} + \Gamma^{\eta}_{\mu\nu} \frac{\mathrm{d}w_\mu}{\mathrm{d}t} \frac{\mathrm{d}w_\nu}{\mathrm{d}t} = 0, \tag{5}$$

where $\Gamma^{\eta}_{\mu\nu}$ specifies the Christoffel symbols ($\Gamma^{\eta}_{\mu\nu} = \sum_r \frac{1}{2} g^{-1}_{\eta r} \left( \frac{\partial g_{r\mu}}{\partial x^\nu} + \frac{\partial g_{r\nu}}{\partial x^\mu} - \frac{\partial g_{\mu\nu}}{\partial x^r} \right)$) on the weight manifold. Such geodesic paths have a constant, potentially non-zero, rate of performance decay on the previous task during adaptation. The Christoffel symbols record infinitesimal changes in the metric tensor ($\mathbf{g}$) along a set of directions on the manifold (Supplementary Information). Since the computation and memory for evaluating Christoffel symbols scales as a third-order polynomial of network parameters ($\mathcal{O}(n^3)$), we propose the optimization equation (4) for evaluating 'approximate' geodesics in the manifold.

## Metric tensor spectrum defines a functionally invariant subspace

Before exploring practical applications of the FIP algorithm for neural network adaptation, we make a series of mathematical observations that connect the dimension of functionally invariant subspaces to the spectral properties of the metric tensor $\mathbf{g}$ at a given position $\mathbf{w}_t$ in the weight space. The metric tensor provides a local measure of functional (input–output) change induced by the perturbation along $\mathbf{dw}$ as $\langle dw, dw \rangle_g$. By construction (as the matrix product $J^{\mathrm{T}}J$), $\mathbf{g}$ is a positive semi-definite, symmetric matrix, so that $\mathbf{g}$ has an orthonormal eigenbasis $\mathbf{v}_i$ with eigenvalues $\{\lambda_i\}$ ($\lambda_i \geq 0$). The eigenvalues $\lambda_i$ locally determine how a perturbation along the eigenvector $\mathbf{v}_i$ will alter the functional

performance as $\langle v_i, v_i \rangle = \lambda_i$ in which movement along an arbitrary vector $w$ induces $\langle w, w \rangle = \sum \lambda_i c_i^2$, where $c_i$ are coefficients of $w$ in the $v_i$ basis. Intuitively, the eigenvalues $\lambda_i$ convert weight changes into a change in functional performance.

Locally, the subspace of $W$ spanned by eigenvectors $v_i$ associated with small eigenvalues (for example, $\lambda_i \ll \epsilon$, where $\epsilon = 10^{-3}$), can be defined as a functionally invariant subspace of weight space as movement within the subspace induces a negligible ($\epsilon$) change in the input–output characteristics of the network. Therefore, the spectrum of $\mathbf{g}$ determines the dimension of the functionally invariant subspace. The Jacobian matrix $J$ for a single data point has dimensions $m \times n$, where $m$ is the networks' output size and $n$ is the number of weights; therefore, $\text{rank}(\mathbf{g}) \leq m$ since $g = J^T J$. When extended to $N$ data points, $\text{rank}(\mathbf{g}(\mathbf{X})) \leq mN$. When $mN < n$, then $\mathbf{g}$ has $n - mN$ eigenvalues with $\lambda_i = 0$. As $n$ can be large (commonly, $n > 10^7$ if not $n > 10^8$ with $mN \approx 10^5$ for many tasks), $\mathbf{g}$ will be rank deficient resulting in a functionally invariant sub-space of non-zero dimension. As $mN \to n$ for increasing training data, random matrix theory has considered the distribution of eigenvalues generated by a random matrix of the form $\mathbf{g} = \frac{1}{n} M^T M$, where $M_{ij}$ are i.i.d. with entries $M_{ij}$ having mean 0 and variance $\sigma^2$; therefore, $\mathbf{g}$ takes the form of random covariance matrices[29]. The Marchenko–Pastur law indicates that such matrices have a substantial density of eigenvalues $\lambda_i \to 0$ as $\frac{mN}{n} \to 1$ for $M$, an $mN \times n$ matrix (Supplementary Note 2). The density of eigenvalues $p(\lambda)$ can be shown to scale like $p(\lambda) \approx \frac{1}{\sqrt{\lambda}}$ as $\lambda \to 0$ and $\frac{mN}{n} \to 1$. Empirically, Supplementary Fig. 1 shows the empirical eigenspectrum, for example, CNN and MLP networks have a substantial fraction of eigenvalues with $\lambda_i < 10^{-3}$.

More globally, as we move along a path $\gamma(t)$ that could point along a $\lambda_i = 0$ eigendirection in weight space, the metric varies along the path. In general, the eigenvectors can rotate and the structure of the eigenspectrum can shift. In this case, the Christoffel symbols ($\Gamma^{\eta}_{\mu\nu}$ in equation (5)) are functions of the derivatives of the metric, and track how the metric $\mathbf{g}$ evolves along a path $\gamma(t)$ in weight space. The geodesic equation, formally, identifies paths in weight space with constant velocity (zero acceleration) despite a path's varying metric tensor and therefore can be applied to discover training paths that exhibit a constant change in network function per unit training time. The evaluation of the Christoffel symbols is computationally expensive for large networks, and the iterative solution of equation (4) provides an interactive solution for numerically computing invariant paths.

## Applications: FIP framework

### FIP enables CL with ViT vision transformers

The FIP framework allows us to address a series of model adaptation goals within a common geometric framework. To demonstrate CL, we applied the FIP to adapt the ViT vision image transformer and BERT language model in CL without catastrophic forgetting. We train a neural network on a base task and modulate the weights in the network to accommodate additional tasks by solving the optimization problem in equation (4), setting $L(\mathbf{x}, \mathbf{w})$ as the classification loss function specified by the additional task, whereas $\langle \frac{d\gamma}{dt}, \frac{d\gamma}{dt} \rangle_{\mathbf{g}_{\text{Task1}}}$ measures the distance moved in the networks' output using the metric from the initial task (in equation (4)). To accommodate the additional tasks, we append output nodes to the base network and solve the optimization problem for a fixed value of $\beta$ by simultaneously minimizing the distance moved in the networks' output space (Fig. 1c, light-blue arrow) corresponding to the first task and maximizing alignment with the gradient of $L(\mathbf{x}, \mathbf{w})$ encoding the classification loss from the second task. In this manner, we construct an FIP (Fig. 5a, purple dotted line) in weight space generating a network that performs both Task 1 and Task 2.

We used a standard CL task[30] (Fig. 2a). We split the Canadian Institute for Advanced Research (CIFAR)-100 dataset into a series of subtasks in which each subtask requires the network to identify ten object categories (Fig. 2a). Previously, the state-of-the-art performance for this task was achieved by the ResNet CNN[30]. ViT networks can potentially realize vast performance gains over ResNet architectures as the baseline performance for ResNet is ~80% accuracy. We observed that ViT exhibits 94.5% accuracy when fine-tuned on the CIFAR-100 dataset[8,31] for generative replay, achieving a CIFAR-100 CL accuracy of ~80%.

We applied the FIP algorithm to achieve CL on SplitCIFAR using both ViT-B (86M parameters) and ViT-H (632M parameters) architectures. In each case, a single task requires the network to learn to recognize 10 CIFAR classes where we used 5,000 images (total, 6,000 images) per class with a batch size of 512. We used a single NVIDIA RTX2060 6 GB for ViT-B and RTX3090 24 GB for ViT-H (Fig. 2b). After continually training 5 SplitCIFAR subtasks, ViT-B achieved a mean performance of 91.2% and ViT-H achieved 89.3% compared with 94.5% performance for ViT-B that was simultaneously trained on all the 50 CIFAR classes without CL (Fig. 2b,c). Training for ViT-B using an NVIDIA RTX2060 6 GB machine took ~3.5 h for each subtask with the FIP and ~2.5 h with fine tuning. Training for ViT-H using an NVIDIA RTX3090 24 GB machine took ~4.8 h for each subtask with the FIP and ~3.9 h with standard fine tuning.

Thus, the FIP procedure enables ViT-B and ViT-H to learn new tasks without information loss and achieves a higher performance than CL methods that have been conventionally applied to the ResNet network[31]. The strength of our method is that it can be applied to fine-tune any existing transformer or CNN architecture.

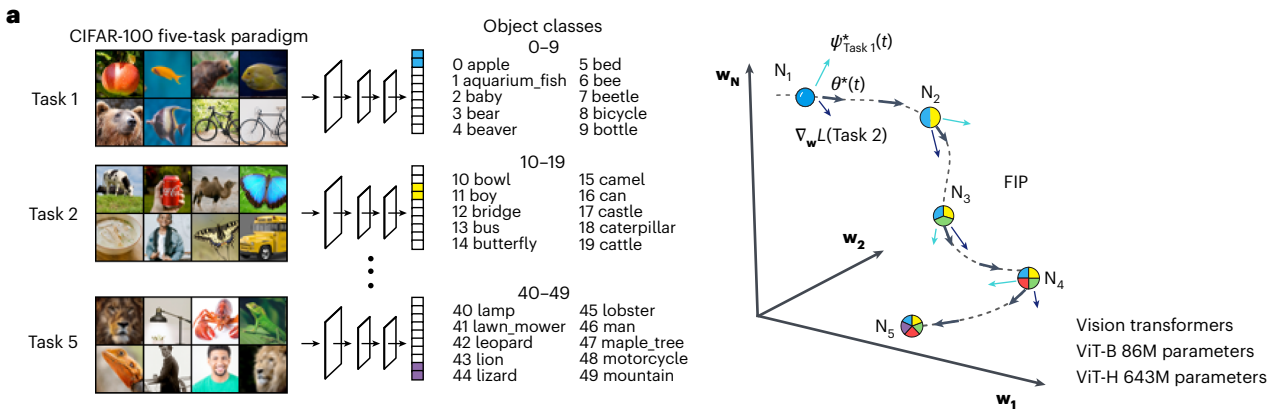
### Comparison of FIP with LoRA on CL with ViT

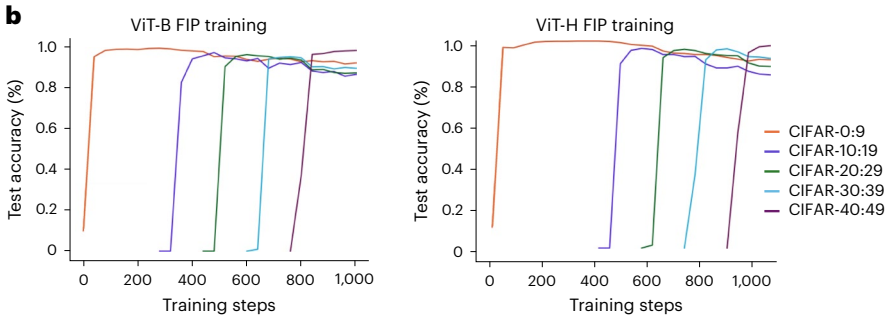
The LoRA approach to network fine tuning was recently introduced to enable the fine tuning of large transformer networks including GPT-3 (ref. 32). To fine-tune pre-trained networks on new tasks, LoRA forces weight updates $W_0$ to a network to be low rank through a matrix factorization strategy, where $W_0$ is generated as the product $W_0 = AB$ of matrices $A$ and $B$ with inner dimension $r$; here $r$ is set to be a number much smaller than network size $k$ ($r \ll k$).

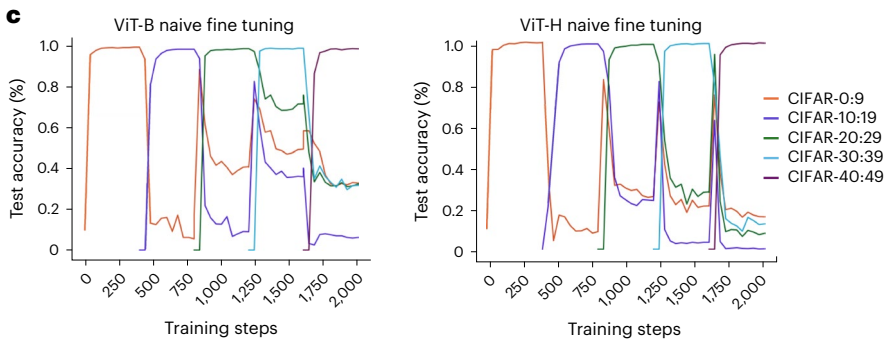
Although not explicitly designed to alleviate catastrophic forgetting, LoRA is discussed in many venues including HuggingFace (HF) as a technique that mitigates catastrophic forgetting. Further, LoRA is widely applied for fine-tuning tasks on large networks in industrial settings and therefore represents an important comparison point for FIP. We applied LoRA to iteratively learn components of the SplitCIFAR task

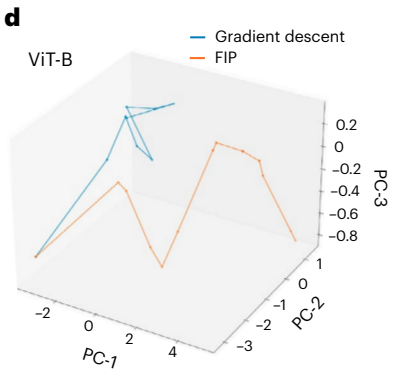
**Fig. 2 | Vision transformers learn sequential tasks without catastrophic forgetting by traversing FIPs. a**, Five-task CL paradigms in which each task is a ten-way object classification, where the classes are taken from CIFAR-100. Right: schematic of FIP construction in weight space to sequentially train networks on five tasks using ViT-Base (ViT-B) and ViT-Huge (ViT-H). **b**,**c**, Test accuracy for ViT-B and ViT-H using FIP (**b**) or naive (**c**) fine tuning. Following continual training, FIP achieves 91.2% and 89.3% test accuracy using ViT-B and ViT-H, respectively, for all five tasks. Baseline performance for ViT-B trained on all five tasks simultaneously is 94.5%. Training for ViT-B using an NVIDIA RTX2060 6 GB machine took ~3.5 h for each subtask with the FIP and ~2.5 h with fine tuning. Training for ViT-H using an NVIDIA RTX3090 24 GB machine took ~4.8 h for each subtask with the FIP and ~3.9 h with fine tuning. **d**, Principal components analysis (PCA) plots of FIP (orange) and gradient descent fine-tuning (blue) weight-space path, showing that the FIP allows long-range exploration of the weight space. **e**, Test accuracy for SplitCIFAR Task 1 (red) and Task 2 (blue) over epochs for ViT-H fine tuning using LoRA with ranks 256, 16 and 1, showing performance decay over time on Task 1 as test accuracy on Task 2 increases. **f**, Scatter plot of LoRA performance for the fine tuning of ViT-H on Task 1 and Task 2 SplitCIFAR tasks compared with the average performance of FIP fine tuning across SplitCIFAR on ViT-B, ViT-H and ResNet18, as well as RMN[30] for ResNet18 and a four-layer CNN. CNNs are indicated in red and transformers, in blue.

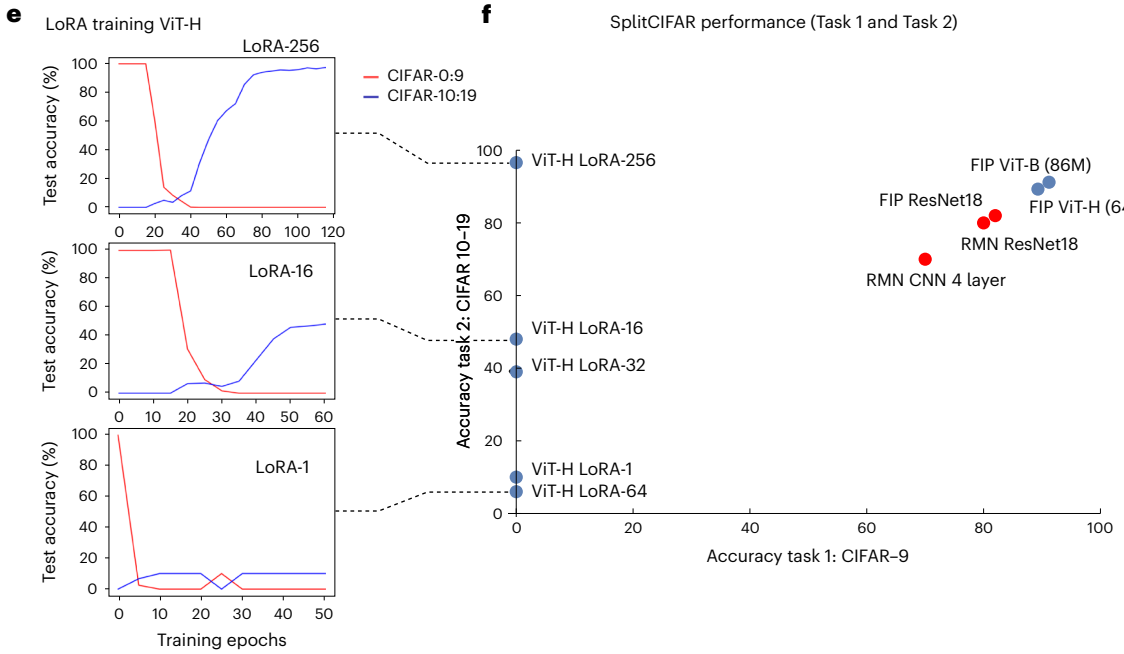
https://doi.org/10.1038/s42256-024-00902-x



(Fig. 2) using ViT-H (640M parameters) spanning a range of $r = \text{rank}(W_0)$ from 1 to 256. We found that LoRA exhibits signatures of catastrophic forgetting independent of rank in these tests (Fig. 2e,f). When ViT-H is trained to achieve 99% accuracy on Task 1 (CIFAR-0:9 task), the network loses accuracy on Task 1 as it is trained via LoRA on Task 2 (CIFAR-10:19 task) (Fig. 2e,f and Extended Data Table 1). Following the application of LoRA, ViT-H achieves 96.6% accuracy on Task 2 for $r = 256$ at the expense of 0% accuracy on Task 1 (Fig. 2e,f and Extended Data Table 1). When rank = 1, the LoRA accuracy was limited to 10% on Task 2 and still lost accuracy on Task 1 (Task 1 accuracy, 0%). Thus, LoRA fine tuning leads to a collapse in accuracy on Task 1 when fine-tuned to perform an additional task.

### Performance of FIP on CL with CNN architectures

In addition to transformers, the FIP framework can also be applied to CNN architectures, which provides a helpful point of comparison with previous CL methods. On image analysis tasks like CIFAR-10, transformers like ViT outperform CNN architectures on classification accuracy metrics. However, CNNs are widely used in computer vision and have been the architecture used for most prior CL work. We applied FIP to ResNet18 to study CL on SplitCIFAR and compared with RES-CIFAR[30], EWC[33], RMN[30], generative replay[31] and gradient episodic memory (GEM)[34]. We found that the FIP could achieve CL with accuracy that meets or exceeds other state-of-the-art approaches (Fig. 2f, Extended Data Table 2 and Supplementary Fig. 2), with RMN achieving 80% accuracy on SplitCIFAR compared with 82% for FIP on ResNet14. In general, the global accuracy of the resulting CNN was lower than the transformer ViT, consistent with the network's relative performance on the baseline CIFAR task. The results demonstrate that FIP performs well on transformer and CNN architectures and on par or above other state-of-the-art approaches. Although the FIP algorithm has conceptual similarities with EWC, the mathematical generality of the FIP allows the approach to scale to perform multiple iterative incremental learning tasks and to explicitly construct FIPs that traverse the weight space (Fig. 2d).

### FIP enables CL with BERT NLP transformer

Next, we demonstrated the flexibility of the FIP approach by using the method to fine-tine the BERT network on the Internet Movie Database (IMDb) sentiment analysis task following initial training on Yelp full-five-star review prediction task (Fig. 3). The BERT network has a total of 12 layers, or transformer blocks, with 12 self-attention heads in each layer and a total of 110M parameters[5]. Training BERT to detect customer opinions of a product based on text reviews left on websites like Yelp or IMDb results in catastrophic forgetting, especially when sequentially training on multiple user-review datasets (say, Yelp reviews followed by IMDb) (Fig. 3a). The FIP maintains BERT performance on Yelp reviews (at 70%; blue) and increasing its accuracy on IMDb review classification (from 0% to 92%; orange) (Fig. 3b). Potentially, BERT has as an initial accuracy of 0% on IMDb due to differences in the outputs for each task, which are binary for IMDb but five-star scoring for Yelp. The FIP in BERT weight space (Fig. 3) is much longer than the route taken by conventional training, enabling the global exploration of the BERT weight landscape to identify networks that simultaneously maintain performance on Yelp reviews and learn the IMDb sentiment classification. Conventional fine tuning of BERT on IMDb reviews increases its performance on sentiment classification on IMDb (from 0% to 92%; orange) and abruptly forgets the sentiment analysis on Yelp reviews (dropping from an accuracy of 69.9% to 17%; blue) within 30 training steps (Fig. 3b). We also compared the performance of FIP with LoRA on the natural language processing (NLP) training task (Fig. 3d–f). We found that LoRA exhibits a considerable performance decay on the Yelp task and learning IMDb across ranks (Fig. 3d,f). LoRA also exhibits anticorrelation between Yelp and IMDb accuracy across training epochs (Fig. 3d). We also found that the FIP algorithm generally induces more extensive weight change than LoRA measured by the Frobenius norm of weight updates for both approaches (Fig. 3e).

### Neural network sparsification with FIP algorithm

The critical aspects of the FIP framework are that the framework generalizes and addresses a broad range of machine learning meta-problems by considering a more general set of secondary objective functions. In particular, we next apply the FIP framework to perform sparsification, reducing the number of non-zero weights, which is important for reducing the memory and computational footprint of a network[35]. To sparsify neural networks, we solve equation (4), the core FIP optimization problem, with a secondary loss function $L(w_t, w, p)$ that measures the Euclidean distance between a network and its $p$-sparse projection obtained by setting $p$% of the networks' weights to zero (Fig. 4a).

Using the framework, we sparsified the vision transformer DeIT, which has been used for benchmarking sparsification methods[36] on vision transformers. The paradigm uses the ImageNet 1,000-object image classification task (ImageNet1K dataset), and attempts to sparsify DeIT. DeIT-Base (DeIT-B) is an 86M parameter transformer model that was derived from ViT[37]. We use the FIP algorithm to set the weight parameters to zero without loss of performance on the ImageNet1K classification task. The simplicity of the FIP algorithm allowed us to achieve an entire range of target sparsities ranging from 0% to 80%. We found that FIP had performance very near to that of the SViT network at the benchmark of 40% sparsity, with FIP performing at 80.22% and SViT performing at 81.56% accuracy (Fig. 4b(i)), with compute times given (Fig. 4b(ii)) on an NVIDIA RTX3090 24 GB. Additionally, we applied FIP to perform the sparsificaiton of CNN architectures (Supplementary Fig. 3) for the Modified National Institute of Standards and Technology (MNIST) and CIFAR tasks, showing that the FIP can also achieve high sparsification values and maintain accuracy in the ResNet and LeNet CNN architectures (Supplementary Fig. 3).

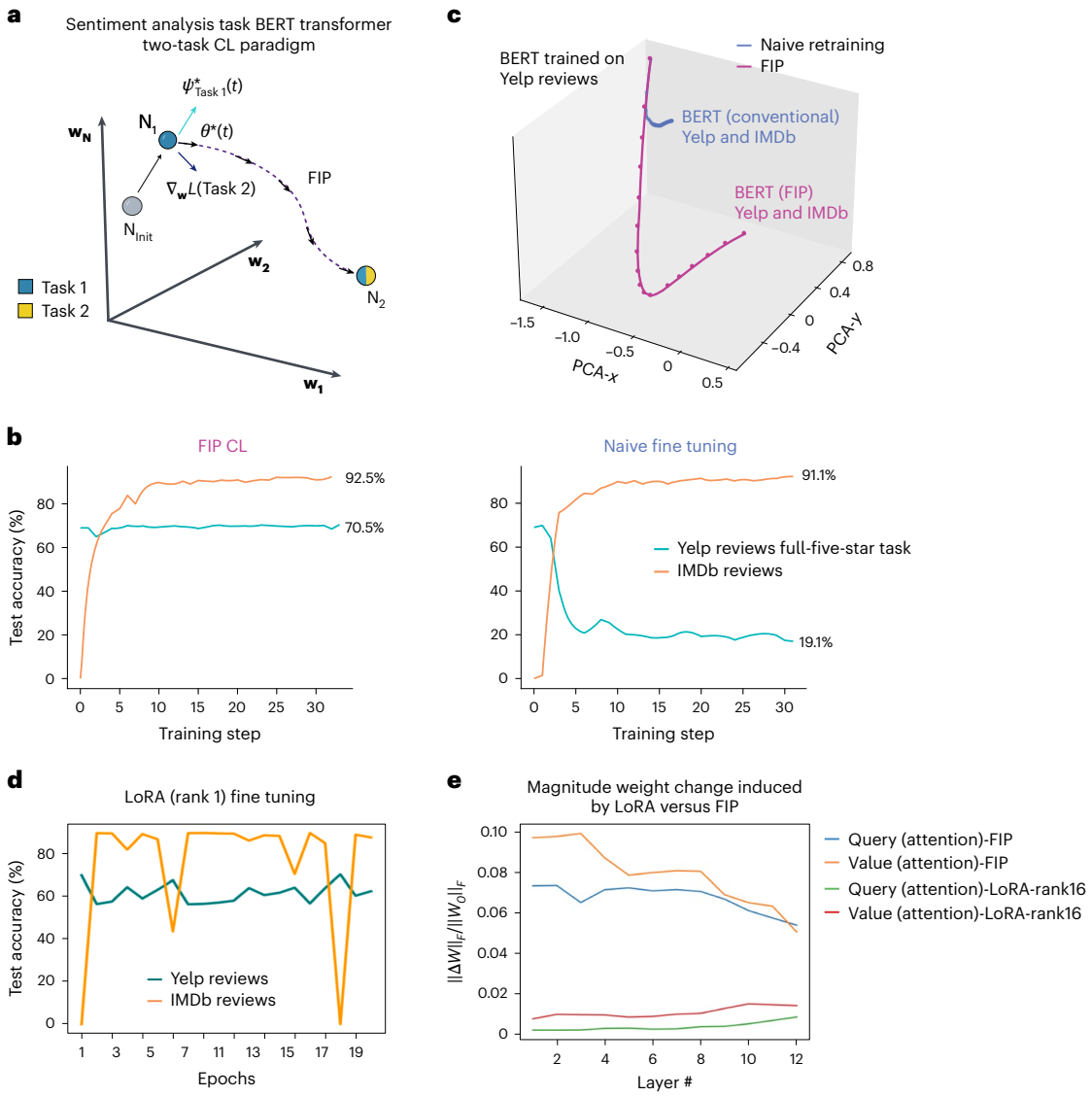
Then, we applied FIP to sparsify the BERT base from 0% to 80% sparsity for all the general language understanding evaluation (GLUE) NLP tasks. For this task, we obtained an NVIDIA A100 machine from PaperSpace. The GLUE benchmark consists of three categories of natural language understanding tasks: (1) single-sentence tasks (corpus of linguistic acceptability (CoLA) and Stanford sentiment (SST-2)), (2) similarity and paraphrase tasks (Microsoft research paraphrase corpus (MRPC), Quora question pairs (QQPs) and Semantic Text Similarity Benchmark (STS-B)) and (3) inference tasks (multi-genre natural language inference (MNLI), question-answering natural language inference (QNLI) and recognizing textual entailment (RTE)). Again, because of its ease of use and efficiency, we were able to span the entire range of sparsities identifying differential performance across GLUE tasks (Fig. 4c and Supplementary Fig. 4). FIP was able to generate sparse versions of BERT that had 81.65% accuracy at 50% sparsity on the SST-2 task. We provide compute times in seconds for the sparsification of BERT on MRPC, which efficiently runs on an NVIDIA A100 machine (Fig. 4d).

### FIP ensemble confers robustness against adversarial attack

The path-connected sets of networks generated by the FIP can also be applied to perform inference and increase the robustness of inference tasks to data perturbation. Although deep CNNs have achieved remarkable performance on image recognition tasks, human-imperceptible additive perturbations, known as adversarial attacks, can be applied to an input image and induce catastrophic errors in deep neural networks (Fig. 5b). The FIP algorithm provides an efficient strategy to increase network robustness and mitigate adversarial failure by generating path-connected sets of networks with diverse weights. We then apply the path-connected network sets to perform robust image classification by averaging their output.

To demonstrate that the FIP algorithm can mitigate adversarial attacks, we trained a 16-layered CNN—VGG16—with 130M parameters

https://doi.org/10.1038/s42256-024-00902-x




**f**

| Algorithm | LoRA rank | $\alpha$ (LoRA) | Yelp accuracy % | IMDb accuracy % |
|---|---|---|---|---|
| LoRA | 1 | 2 | 70.5 → 61.10 | 82.70 |
| LoRA | 4 | 8 | 70.5 → 48.00 | 90.00 |
| LoRA | 8 | 16 | 70.5 → 50.60 | 89.60 |
| LoRA | 16 | 32 | 70.5 → 30.30 | 91.40 |
| LoRA | 32 | 64 | 70.5 → 23.90 | 91.40 |
| FIP | – | – | 70.5 → 70.0 | 92.5 |

**Fig. 3 | BERT transformer learns sequential sentiment analysis tasks without catastrophic forgetting by traversing an FIP. a**, Schematic of FIP construction in weight space to sequentially train networks on two tasks. We initially train the BERT transformer on the Yelp five-star review prediction task and then add the IMDb sentiment analysis task. **b**, Test accuracy for BERT trained through CL via the FIP or with conventional fine tuning. **c**, Weight-space paths for conventional fine tuning and FIP-based training. **d**, Test accuracy for BERT fine tuning on IMDb sentiment analysis following training on the Yelp five-star review task. LoRA exhibits fluctuations in test accuracy on IMDb. **e**, Normalized change in the Frobenius norm of network weights induced by FIP and LoRA, showing that FIP induces a larger total weight change in BERT parameters during fine tuning than LoRA; this points to a longer-range exploration of weight space. **f**, LoRA settings (rank and $\alpha$) and associated accuracy changes on Yelp during fine tuning on IMDb. The network is first trained on the Yelp full-five-star review task to achieve 70.5% accuracy and then adapted using LoRA or FIP for the IMDb movie review task. The table contains LoRA rank and $\alpha$ parameters as well as accuracy on Yelp and IMBD tasks. Networks lose accuracy on Task 1 when fine-tuned with LoRA across rank settings.

to classify CIFAR-10 images with 92% test accuracy. We, then, generated adversarial test images using the projected gradient descent (PGD) attack strategy. On adversarial test images, the performance of VGG16 dropped to 37% (Fig. 5b and Supplementary Fig. 1). To mitigate the adversarial performance loss, we applied the FIP algorithm to generate an ensemble of functionally invariant networks by setting $L = 0$ in the optimization problem in equation (4) and setting $\langle \frac{dy}{dt}, \frac{dy}{dt} \rangle_{\mathbf{g}_{\text{CIFAR10}}}$ to be the distance moved in the networks' output space for CIFAR-10 images.

https://doi.org/10.1038/s42256-024-00902-x

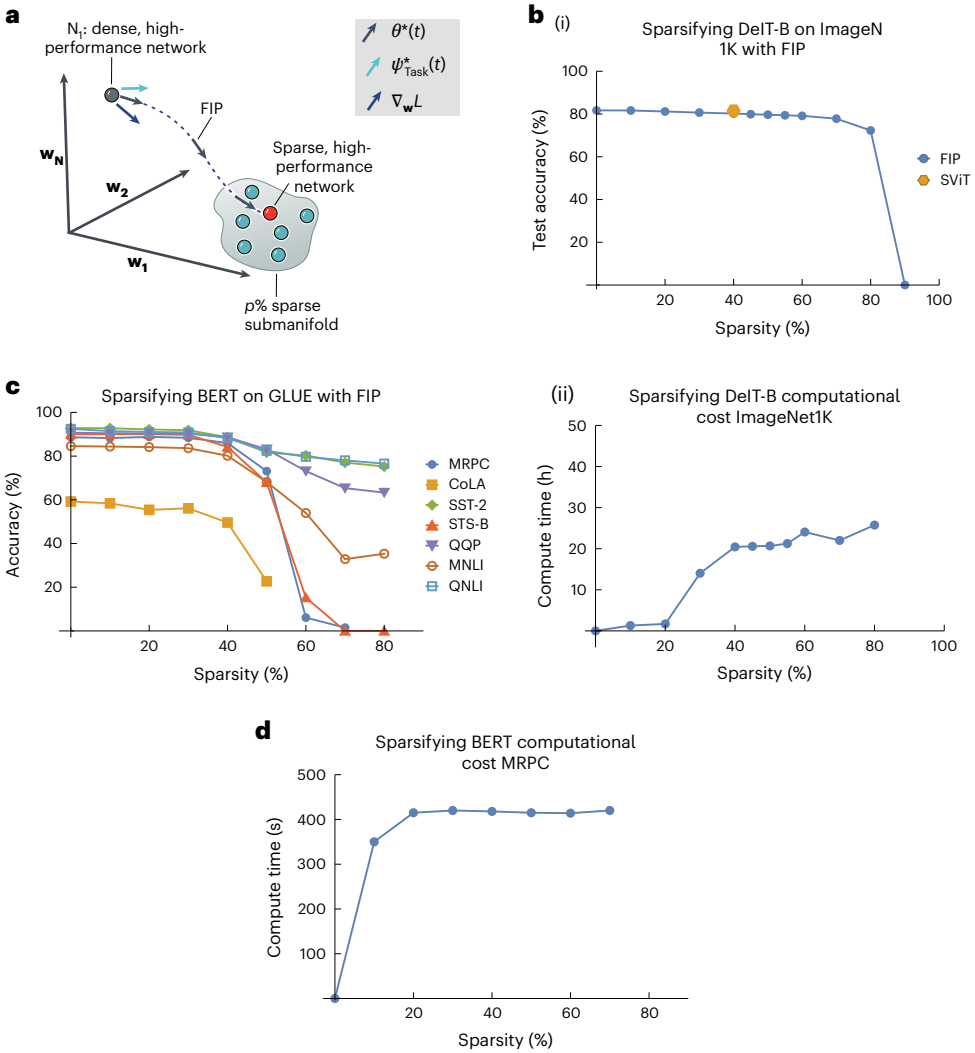



**Fig. 4 | Sparsification of vision and language transformers with FIP. a**, We applied the FIP algorithm to generate sparse versions (network weights set to 0) of the DeIT-B vision transformer performing the ImageNet1K image classification task. Using FIP, we generated networks with sparsity (fraction of weights set to 0) ranging from 0 to 80%. We compared network performance with sparsification reported for DeIT-B using a state-of-the-art method[36]. FIP could achieve very minimal reductions in performance until ~80% sparsity, but further attempts at parameter reduction failed. **b**, Compute time (in hours) for DeIT sparsification. **c**, Sparsification of BERT on the nine GLUE NLP tasks. Sparsification performance is task dependent. **d**, Compute time (in seconds) for the MRPC task on an NVIDIA A100 machine.

We use the FIP ensemble to classify images by summing the 'softmaxed' outputs of the ensemble.

Using an ensemble of ten networks sampled along an FIP, we achieve an accuracy of 55.61 ± 1.1%, surpassing the performance of the DeepNet ensemble (composed of 10 independently trained deep networks) by 20.62% (Fig. 5c). The FIP ensemble's adversarial performance also surpasses other state-of-the-art ensemble approaches including adaptive diversity-promoting (43.84 ± 7.8%) ensemble and the fast geometric ensembling (41.7 ± 0.34) method. The two factors contributing to the FIP ensemble's robustness are (1) high intra-ensemble weight diversity, calculated by the representation diversity score; and (2) low coherence with a trained surrogate network (used to generate adversarial images) (Fig. 5e,f). FIP networks have a higher representation diversity score in their early processing layers, from layer 1 to layer 6, compared with the DeepNet ensemble, indicating that individual networks in the FIP ensemble extract different sets of local features from the adversarial image, preventing networks from relying on similar spurious correlations for image classification. We speculate that weight/parameter diversity in the FIP ensemble leads to differential susceptibility to adversarial examples but consistent performance on training and test examples. The approach has analogies with the model soup approaches explored in ref. 38.

## Generating path-connected sets of language models

Conceptually, the most important aspect of the FIP framework is that it unifies a series of machine learning meta-tasks (CL and sparsification) into a single mathematical framework. Mathematically, when we solve equation (4) with a given secondary loss function, we move an existing network $\mathbf{w}(\mathbf{0})$ along a path in weight space generating a new network $\mathbf{w}(\mathbf{t})$ in which the parameter $t$ increments the length of a path. Each additional loss function generates a new transformation of a base network, for example, generating a network adjusted to accommodate an additional data analysis problem or a secondary objective like sparsification. Network transformation maps can be iterated and applied to generate a family of neural networks optimized for distinct subtasks. The resulting path-connected set of neural networks can then be queried for networks that achieve specific user goals or solve additional machine learning problems.

The problem of customization is particularly important for transformer networks[4,39–41]. Transformers like BERT and Roberta are typically

https://doi.org/10.1038/s42256-024-00902-x

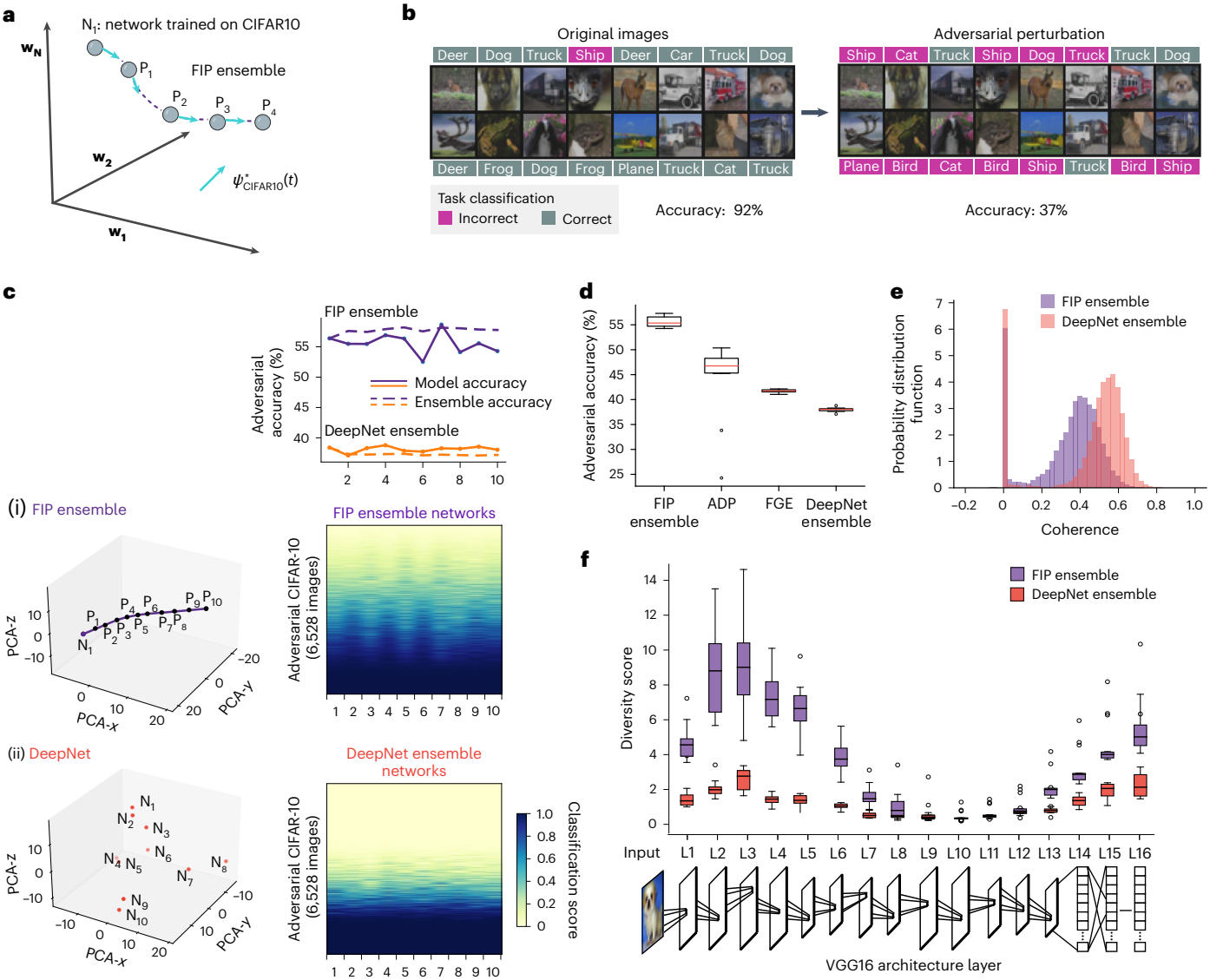



**Fig. 5 | FIPs in weight space generate ensembles of networks that confer adversarial robustness. a**, Schematic to generate FIP ensemble ($P_1$–$P_4$) by sampling networks along the FIP (purple dotted line) beginning at network $N_1$. FIP is constructed by identifying a series of weight perturbations that minimize the distance moved in the networks' output space. **b**, Original CIFAR-10 images (left) and adversarial CIFAR-10 images (right) are shown. The text labels (left and right) above the images are predictions made by a network trained on CIFAR-10. Trained networks' accuracy on the original and adversarial images are also shown (bottom). **c**, Top: the solid line plot shows the individual network performance on adversarial inputs and the dashed line plot shows the joint ensemble accuracy on adversarial inputs for FIP ensemble (purple) and DeepNet ensemble (orange). Left: FIP ensemble in purple ($P_1$–$P_{10}$) (i) and DeepNet ensemble in orange ($N_1$–$N_{10}$) (ii) are visualized on weight-space PCA. Right: heat maps depict the classification score of networks in FIP ensemble (i) and DeepNet ensemble (ii) on 6,528 adversarial CIFAR-10 examples. **d**, Box plot compares the adversarial accuracy (over 10k adversarial examples) across different ensembling techniques ($n = 3$ trials). Box and whiskers represent the interquartile range. ADP, adaptive diversity promoting; FGE, fast geometric ensembling. **e**, Histogram of coherence values for FIP (purple) and DeepNet (orange) ensembles. **f**, Box plot shows the ensemble diversity score across VGG16 layers over $n = 1{,}000$ CIFAR-10 image inputs. The box plot compares adversarial accuracy (over 10k adversarial examples) across different ensembling techniques ($n = 3$ trials). Box and whiskers represent the interquartile range. The cartoon in the bottom depicts the VGG16 network architecture.

trained on generic text corpus like the common crawl, and the specialization of networks on specific domains is a major goal[3,32]. We applied the iterative FIP framework to generate a large number of NLP networks customized for different subtasks and sparsification goals. Transformer networks incorporate layers of attention heads that provide contextual weight for positions in a sentence. Transformer networks are often trained on a generic language processing task like sentence completion in which the network must infer missing or masked words in a sentence. Models are then fine-tuned for specific problems including sentiment analysis, question answering, text generation and general language understanding[39]. Transformer networks are large containing hundreds of millions of weights, and therefore, model customization can be computationally intensive.

We applied the FIP framework to perform a series of model customization tasks through the iterative application of FIP transformations on distinct goals. The BERT network has a total of 12 layers, or transformer blocks, with 12 self-attention heads in each layer and a total of 110M parameters[5]. Using the FIP framework, we sequentially applied two operations (CL and compression (Co)) to BERT models trained on a range of language tasks, by constructing FIPs in the BERT weight space using different objective functions ($L(\mathbf{x}, \mathbf{w})$) and solving the optimization problem in equation (4) (Fig. 6).

https://doi.org/10.1038/s42256-024-00902-x

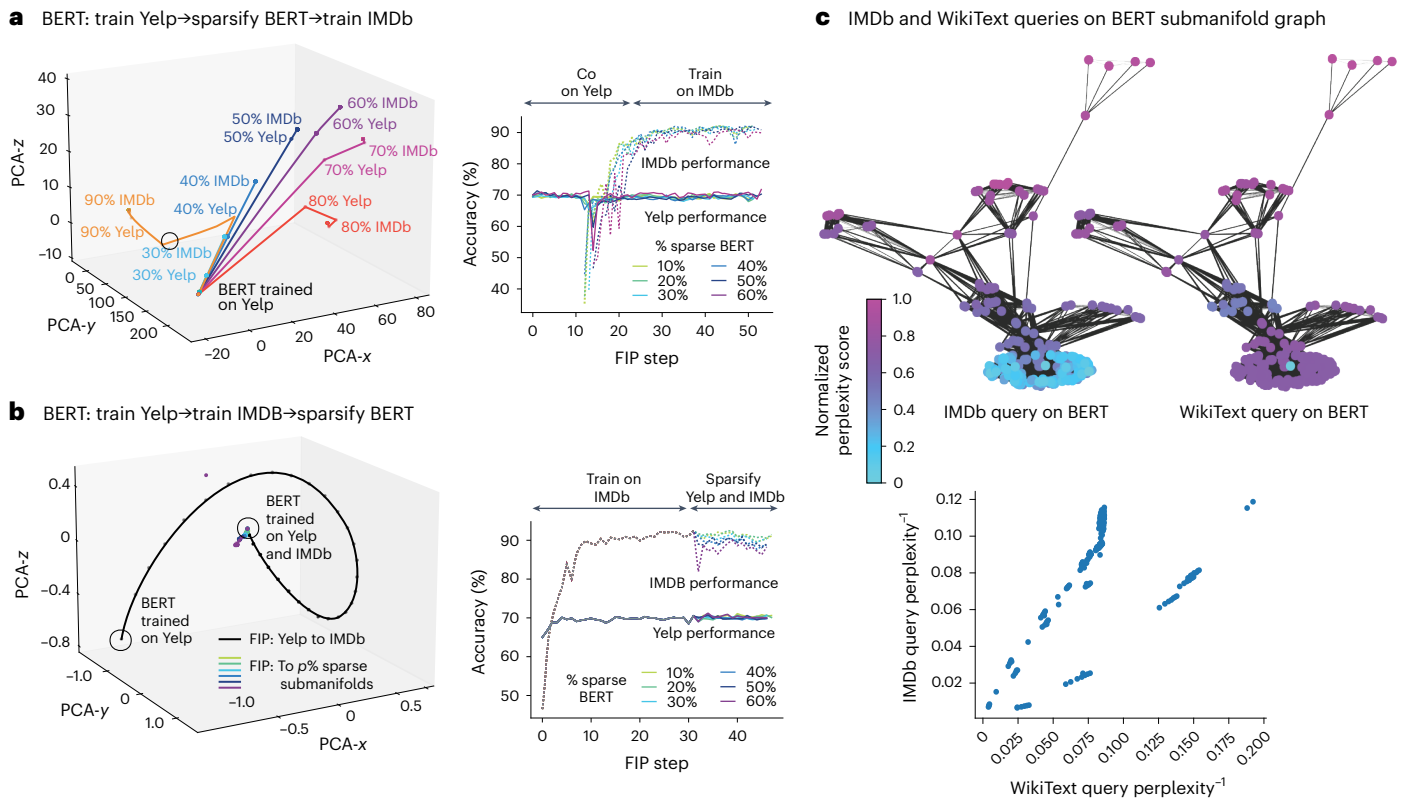



**Fig. 6 | Non-Abelian, iterative transformation of BERT by FIP composition. a**, Left: FIP initially identifies high-performance sparse BERTs (for sparsities ranging from 10% to 80%) followed by re-training on IMDb. Right: BERT accuracy on Yelp (solid) and IMDb (dashed) dataset along the FIP. **b**, Left: FIP initially retrains BERT on new task (IMDb) and then discovers a range of sparse BERTs. Right: BERT accuracy on Yelp (solid) and IMDb (dashed) dataset along the FIP. **c**, Top: graph connected set of 300 BERT models trained on sentence completion on Wikipedia and Yelp datasets, coloured by perplexity scores evaluated using two new query datasets, namely, WikiText and IMDb. Nodes correspond to individual BERT models and edges correspond to the Euclidean distance between BERT models in weight space. Bottom: scatter plot of inverse perplexity scores for two queries—IMDb and WikiText datasets.

We demonstrated that we could apply the CL and Co operations in sequence. Beginning with the BERT base model, we applied the FIP model to generate networks that could perform Yelp and IMDb sentiment analysis, and then compressed the resulting networks to generate six distinct networks with sparsity ranging from 10% to 60%, where all the sparsified networks maintained performance on the sentiment analysis tasks (Fig. 6a). We, then, performed the same operations, but changed the order of operations (Fig. 6b). The models generated through the application of CLCo($w$) and CoCL($w$) achieved similar functional performance in terms of sparsification and task performance but had distinct weight configurations.

In total, we used the FIP framework to generate networks using a set of different natural language tasks and Co tasks, yielding a path-connected set of 300 BERT models (Fig. 6c). The 300 networks define a submanifold of weight space that contains models customized for distinct subtasks. Then, the submanifold provides a computational resource for solving new problems. We can query the resulting FIP submanifold with unseen data by using perplexity as a measure of a networks intrinsic ability to separate an unseen dataset. Using perplexity, we queried the FIP submanifold of BERT networks with IMDb data and WikiText data. We found that the distinct language datasets achieve minimal perplexity on different classes of networks. WikiText obtains optimal performance on CoCL networks and IMDb achieves optimal performance on CLCo networks. These results demonstrate that the FIP framework can generate diverse sets of neural networks by transforming networks using distinct meta-tasks. The submanifolds of weight space can then be inexpensively queried to identify networks pre-optimized for new machine learning problems.

## Related work

The major contribution of our framework is that it provides a unified theoretical and mathematical framework through which a set of problems can be addressed across a variety of neural network architectures. To the best of our knowledge, no single mathematical framework has been applied to address the set of machine learning meta-problems and diversity of neural network architectures that we address with the FIP framework. Further, with the rise of transformer models, we have seen the emergence of architecture-specific strategies for sparsification and CL. We provide a unified framework that can achieve a similar quality of results across different classes of architectures (CNN and transformers) and across different transformer variants. Although representation learning frameworks have been pursued in fields including reinforced learning[42,43], our work develops a geometric framework that identifies invariant subspaces within the parameter space for distinct auxiliary tasks. We demonstrate that the framework scales to transformer models with hundreds of millions of parameters. Therefore, our framework provides a unified theoretical and mathematical model for connecting the geometry of parameter space with different model subtasks as well as practically scaling to provide results in line with the state-of-the-art, more specific approaches. The FIP is also related to Jacobian regularization approaches that have been explored to stabilize model training and prevent over-fitting[44]. Here we review some of the specific approaches that have been studied for CL, sparsification and adversarial robustness.

### CL and catastrophic forgetting

CL has been studied extensively in the machine learning literature[45–47] for classical CNN as well as vision and language transformers[48,49].

A wide variety of approaches have been developed to train neural networks on a sequence of tasks without the loss of prior task performance including regularization-based methods, weight parameter isolation methods and replay-based methods. Many classic CL/catastrophic forgetting (CF) approaches were developed for CNN architectures with <100M parameters. With the emergence of transformer models for NLP and computer vision, approaches have become more model specific with approaches such as regularization-based methods and parameter isolation methods. Regularization-based methods assign constraints to prevent parameter changes to important parameters from prior tasks. The FIP framework can be viewed as a generalized regularization framework that applies at any point in the parameter space. Methods like EWC[33], OGD[50], RMN[30] and synaptic intelligence[51] penalize the movement of parameters that are important for solving previous tasks to mitigate catastrophic forgetting. RMN identifies the relevant units in the pre-trained model that are most important for the new task using a thresholding strategy. OGD constrains movement in the parameter space to move in a subspace that is orthogonal to gradients from prior tasks. OGD was applied to CL tasks on the MNIST data[50] on a small, three-layer multi-layer perceptron. The OGD method requires knowledge of prior task gradients and is a local method, like EWC. Although (locally in the parameter space) it makes sense to constrain changes in the network parameters along the orthogonal subspace, there is no mathematical reason that long-range updates should satisfy this constraint. Past gradient directions become less meaningful as we move away from an initial trained network and begin traversing long-range paths in the parameter space in search of good networks. By using a metric tensor construction that is defined at every point in the parameter space, the FIP framework can traverse long-range paths in the parameter space, sometimes making weight updates that are, in fact, not orthogonal to gradients.

Replay-based methods store and replay past data or apply generative models to replay data during training to mitigate catastrophic forgetting[52,53]. Regularization-based methods including EWC constrain the learning of important parameters from previous tasks by assigning constraints to prevent parameter changes. Architecture expansion or dynamic architecture methods like progressive neural networks[54], dynamically expandable networks[55] and super-masks in position[56] adapt, grow or mask the model's architecture, respectively, allowing specialization without interfering with previous knowledge. Parameter isolation methods: these methods isolate specific parameters or subsets of parameters to be task specific, preventing interference with previous knowledge. Examples include the context-dependent gating approach or the learning-without-forgetting method. Instead of modifying the learning objective or replaying data, various methods use different model components for different tasks. In progressive neural networks, dynamically expandable networks and reinforced CL (RCL) the model is expanded for each new task.

### Low-rank fine tuning with LoRA

LoRA[32] was designed to enable the computationally efficient adaptation of large transformer models by forcing weight updates to be low ranked. LoRA achieves impressive performance on the fine tuning of very large models including GPT-3 (175B). LoRA achieves its performance by introducing a low-rank structure heuristic into the weight update through matrix factorization, forcing $\Delta W = AB$, where $A$ and $B$ have an inner dimension $r$, and thus controlling the rank of $W$ to be $r$. We view FIP and LoRA as complementary methods and seek to incorporate low-rank constraints into new versions of the FIP algorithm.

### Sparsification methods

Sparsity methods for neural networks aim to reduce the number of parameters or activations in a network, thereby improving efficiency and reducing memory requirements. Unstructured pruning techniques apply strategies based on network weights or weight gradients to remove weights that do not contribute to network performance. Unstructured sparsity methods seek to remove weights at any position in the network. The lottery ticket hypothesis (LTH) demonstrated that dense networks often contain sparse subnetworks, named winning tickets, that can achieve high accuracy when isolated from the dense network[57]. LTH methods discover these sparse networks through an empirical pruning procedure. Unstructured pruning methods remove inconsequential weight elements using criteria including weight magnitude, gradient and Hessian[47,58,59]. Recent strategies[60,61] dynamically extract and train sparse subnetworks instead of training the full models. Evolutionary strategies including the sparse evolutionary training procedure[62,63] begin with sparse topologies (for example, Erdős–Rényi generated graphs) in training and optimize topology and network weights during training. Historically, sparsity methods have been applied to convolutional and multi-layer perceptron architectures. Recent frameworks have been introduced for the sparsification of transformer architectures. For example, a vision transformer sparsification strategy SViTE[36] was developed that uses model training integrated with prune/grow strategies, achieving the sparsification of the ViT family of transformers. The sparsification of BERT[64] was explored by identifying specific internal network topology that can achieve high performance on NLP tasks with sparse weight distribution through empirical investigation and ablation experiments.

### Robustness to adversarial attack

Finally, we demonstrate that the FIP can generate ensembles of networks with good performance in adversarial attack paradigms. Like CL and sparsificaiton, adversarial attack mitigation has been addressed by a wide range of methods including augmented training with adversarial examples[65], through the use of diversified network ensembles[66], as well as through re-coding of the input[67]. We demonstrate that the FIP framework can generate network ensembles that have intrinsic resistance to adversarial attack compared with base networks.

## Discussion

We have introduced a mathematical theory and algorithm for training path-connected sets of neural networks to solve machine learning problems. We demonstrate that path-connected sets of networks can be applied to diversify the functional behaviour of a network, enabling the network to accommodate additional tasks, to prune weights or generate diverse ensembles of networks for preventing failure to an adversarial attack. More broadly, our mathematical framework provides a useful conceptual view of neural network training. We view a network as a mathematical object that can be transformed through the iterative application of distinct meta-operations. Meta-operations move the network along paths within the weight space of a neural network. Thus, we identify path-connected submanifolds of weight space that are specialized for different goals. These submanifolds can be enumerated using the FIP algorithm and then queried as a computational resource and applied to solve new problems (Fig. 6e).

Fundamentally, our work exploits a parameter degeneracy that is intrinsic to large mathematical models. Previous work has demonstrated that large physical or statistical models often contain substantial parameter degeneracy[16], leading to 'flat' objective functions[17–19] such that model parameters can be set to any position within a submanifold of parameter space without the loss of accuracy in predicting experimental data[16]. In the Supplementary Information, we mathematically show that the neural networks that we analyse have mathematical signatures of parameter degeneracy after training, through the spectral analysis of the metric tensor. Modern deep neural networks contain large numbers of parameters that are fit based on the training data and therefore relate mathematical objects to physical models with large numbers of parameters set by the experimental data. In fact, exact

mappings between statistical mechanics models and neural networks exist[68]. Spectral analysis demonstrates that the weight space contains subspaces where the movement of parameters causes an insignificant change in the network behaviour. Our FIP algorithm explores these degenerate subspaces or submanifolds of parameter space. Implicitly, we show that the exploration of the degenerate subspace can find regions of flexibility in which the parameters can accommodate a second task (a second image classification task) or goal-like sparsification. We apply basic methods from differential geometry to identify and traverse these degenerate subspaces. In the Supplementary Information, we show that additional concepts from differential geometry including the covariant derivative along a weight-space path can be applied to refine paths by minimizing not only the velocity along a weight-space path but also acceleration.

Broadly, our results shift attention from the study of single networks to the path-connected sets of neural networks. Biological systems have been hypothesized to explore a range of effective network configurations due to both fluctuation-induced drift and modulation of a given network by other subsystems within the brain[11–15]. Networks could explore paths through fluctuations or also through the influence of top-down activity. By traversing FIPs, networks could find efficient routes to learn new tasks or secondary goals. By shifting attention from networks as single configurations or points in weight space to exploring submanifolds of the weight space, the FIP framework may help illuminate a potential principle of flexible intelligence and motivate the development of mathematical methods for studying the local and global geometry of functionally invariant solution sets to machine learning problems[69,70].

## Methods

In the main text, we have introduced a geometric framework to solve three core challenges in modern machine learning, namely, (1) alleviating catastrophic forgetting, (2) network sparsification and (3) increasing robustness against adversarial attacks. We will describe the datasets, parameters/hyperparameters used for the algorithms and the pseudo-code for each of the core challenges addressed in the main text.

### Catastrophic forgetting

**Datasets and preprocessing.** The models were tested on two paradigms:

- SplitCIFAR100: ten sequential-task paradigm, where the model is exposed to ten tasks, sampled from the CIFAR-100 dataset. The CIFAR-100 dataset contains 50,000 RGB images for 100 classes of real-life objects in the training set, and 10,000 images in the testing set. Each task requires the network to identify images from ten non-overlapping CIFAR-100 classes.

We performed two operations, namely, (1) network Co and (2) CL on BERT, fine-tuned on different language datasets, downloaded from the HF website.

- Wikipedia: the Wikipedia English datasets are downloaded from https://huggingface.co/datasets/wikipedia. We used the Wikipedia dataset on the masked language model (MLM) task.
- Yelp reviews: the Yelp review dataset is obtained from HF, downloaded from https://huggingface.co/datasets/yelp_review_full.
- IMDb reviews: the IMDb review dataset is obtained from HF, downloaded from https://huggingface.co/datasets/imdb.
- GLUE dataset: the GLUE dataset is obtained from HF, downloaded from https://huggingface.co/datasets/glue. The GLUE tasks performed in this paper are (1) QQPs, (2) SST-2, (3) CoLA, (4) RTE, (5) MNLI, (6) MRPC and (7) QNLI.

**Parameters used.**

- FIP for CL: $\eta = 2 \times 10^{-5}$, $\lambda = 1$, $n$ memories from previous task = 2,000/650,000 = (0.8% previous dataset); optimizer used, AdamW.
- FIP for BERT sparsification: $\eta = 2 \times 10^{-5}$, $\lambda = 1$; optimizer used, AdamW. Final (desired) network sparsities for the GLUE task: task ($p$% sparse): RTE (60% sparse), CoLA (50% sparse), STS-B (50% sparse), QNLI (70% sparse), SST-2 (60% sparse), MNLI (70% sparse), QQP (90% sparse) and MRPC (50% sparse). Final (desired) network sparsities for Wikipedia sentence completion: [10%, 20%...90%].

**Network architectures.** All state-of-art methods for alleviating CF (presented in the main text) in the two-task and five-task paradigm used the same network architectures: the ViT-Base and ViT-Huge transformer variants described at https://huggingface.co/docs/transformers/model_doc/vit. BERT is a popular transformer model with 12 layers (transformer blocks), each with a hidden size of 768, 12 self-attention heads in each layer with a total of 110M parameters (https://huggingface.co/docs/transformers/model_doc/bert). BERT has been pre-trained on 45 GB of Wikipedia data, using the MLM task and next-sentence prediction.

*Sentence completion (masking) tasks*. For the masking tasks (where 15% of the words in the input sentence are masked (or blanked)), the BERT network has an MLM head appended to the network. The MLM head produces a three-dimensional tensor as the output, where the dimensions correspond to (1) the number of sentences in a single batch (batch-size), (2) number of blanked-out words in a sentence and (3) number of tokens in the BERT vocabulary (30,000 tokens).

*Sentence classification tasks*. For the sentence classification task, a sentence classifier head was appended to the BERT architecture. Here the classifier head produces a two-dimensional output tensor, where the dimensions correspond to (1) batch-size and (2) number of unique classes in the classification problem.

### Pseudo-code: FIP construction for CF problems

**Algorithm 1.** FIP construction for CF problems:

**Require** $\lambda$; $\eta$, step-size hyperparameters; $N_T$, number of sequential tasks

**procedure** FIP-CF($\lambda, \eta, N_T$)
  random initialize $\mathbf{w_0}$
  $B_i \leftarrow \{\} \forall i = 1, 2 \ldots N_T$ ▷ buffer with $n_{mem}$ memories from previous tasks
  **for** $i \leftarrow 1$ to $N_T$ **do**
    $\mathbf{w_i} \leftarrow \mathbf{w_{i-1}}$
    $(\mathbf{x}, \mathbf{t}) \leftarrow$ Task $i$ ▷ minibatch of images ($x$) and target labels ($t$) from task $i$
    $B_i \leftarrow B_i \cup \mathbf{x}$ ▷ update buffer
    CEloss $\leftarrow$ Cross-Entropy($f(\mathbf{x}, \mathbf{w_i}), \mathbf{t}$) ▷ classification loss for new task
    Yloss $\leftarrow 0$
    **for** $j \leftarrow 1$ to $i - 1$ **do**
      Yloss += Ydist($f(\mathbf{x}, \mathbf{w_i}), f(B_j, \mathbf{w_{i-1}})$) ▷ distance moved in output space ($Y$)
    **end for**
    $S \leftarrow$ CEloss + $\lambda \times$ Yloss ▷ construct FIP with direction from loss gradient
    $\mathbf{w_i} \leftarrow \mathbf{w_i} - \eta \nabla_{\mathbf{w_i}} S$
  **end for**
  **return** $\mathbf{w_i}$
**end procedure**

**Code specifications.** All the code was written in the PyTorch framework, and the automatic differentiation package was extensively used for constructing the computational graphs and computing gradients

for updating the network parameters. The code for constructing the FIPs for the 2-task and 20-task paradigm was run on Caltech's high-performance computing cluster using a single GPU for a total time of 1 h and 10 h, respectively.

**Parameters used.** The parameters used for the current state-of-art methods across different models and datasets have been selected after grid search to maximize accuracy.

- FIP for 2-task paradigm: $\eta = 0.01$; optimizer used, Adam; weight-decay = $2 \times 10^{-4}$, $\lambda = 1$, $n$ memories from previous task = 500/60,000 (0.8% of the previous dataset).
- EWC for 2-task paradigm: optimizer used, Adam; EWC regularization coefficient ($\lambda$) = 5,000, learning rate = 0.001, batch-size = 128, number of data samples from previous task to construct the Fisher metric = 500.
- FIP for 20-task paradigm: $\eta = 0.01$; optimizer used, Adam; weight-decay = $2 \times 10^{-4}$, $\lambda = 1$, $n$ memories from previous task = 250/2,500 (10% of the previous tasks).
- GEM for 20-task paradigm: $n$ memories from previous task = 250, learning rate = 0.01, number of epochs (per task) = 20, memory strength = 0.5, batch-size = 128.
- EWC for 20-task paradigm: optimizer used, Adam; EWC++, $\alpha = 0.9$, $\lambda = 5{,}000$, learning rate = 0.001, Fisher metric update after 50 training iterations and batch-size = 128.

**Implementation of other CF methods.** We implemented the EWC method by adapting the code available at https://github.com/moskomule/ewc.pytorch. The GEM method was applied by adapting the code available at https://github.com/facebookresearch/GradientEpisodicMemory.

LoRA: LoRA training for the model was done using the PEFT library[71]. We instantiate the base language model (for example, model trained on the Yelp dataset). We create a LoRAConfig object, where we can specify the following parameters: LoRA rank (ranks of 1, 4, 8, 16, 32), LoRA $\alpha$ (scaling factor): (usually we use $\alpha$ = rank × 2). Target modules: 'query' and 'value' in the attention layer (across all the layers of the network) and LoRA dropout = 0.1. In the LoRA experiments, we explored a variety of different learning rates to stabilize training including $10^{-6} \rightarrow 10^{-5}$. For ViT experiments, we use a ramping procedure with a learning rate that starts at $1 \times 10^{-3}$ until it reaches the maximum peak at 1 and then decreases.

Example parameter settings for LoRA on NLP for BERT are as follows.

- LoRA training with rank = 16, $\alpha = 32$; learning rate = $3 \times 10^{-5}$
- LoRA training with rank = 16, $\alpha = 32$; learning rate = $1 \times 10^{-5}$
- LoRA training with rank = 16, $\alpha = 32$; each step corresponds to 400 samples at learning rate = $1 \times 10^{-5}$
- LoRA training with rank = 16, $\alpha = 32$; each step corresponds to 1,600 samples at learning rate = $1 \times 10^{-5}$
- LoRA training with rank = 16, $\alpha = 32$; each step corresponds to 1,600 samples at learning rate = $2 \times 10^{-6}$ (initial training steps)
- Post-convergence: LoRA training with rank = 4, $\alpha = 8$; each step corresponds to 1,600 samples at learning rate = $2 \times 10^{-6}$

## Network sparsification

**Datasets and preprocessing.** The models were sparsified using a well-known image dataset, ImageNet1K (https://huggingface.co/datasets/imagenet-1k), which contains 1,000 object classes and contains 1,281,167 training images, 50,000 validation images and 100,000 test images.

**Network architectures.** We used the DeIT vision transformer from Meta for demonstrating the strategy for constructing FIP in weight space. The network and variants are described at https://huggingface.co/docs/transformers/model_doc/deit. As described, the base DeIT (86M parameters) achieves top-1 accuracy of 83.1% (single-crop evaluation) on ImageNet with no external data.

We also used the BERT dataset available at https://huggingface.co/docs/transformers/model_doc/bert.

### Pseudo-code: FIP construction for network sparsification

**Algorithm 2.** FIP construction for network sparsification:
**Require**: $\lambda, \eta$
**Require**: $p$, final desired network sparsity (in percentage)
**Require**: $\mathbf{w}_t$, network trained on MNIST or CIFAR-10 dataset

**procedure** FIP-Sparse($\lambda, \eta, p, \mathbf{w}_t$)
  $\mathbf{w} \leftarrow \mathbf{w}_t$
  **while** ($||\mathbf{w}||_0/||\mathbf{w}_t||_0$) NOT $(1 - p/100)$ ▷ until $\mathbf{w}$ not in $p$% sparse submanifold
    $\mathbf{w}_p \leftarrow$ project($\mathbf{w}, p$) ▷ set $p$% of smallest weights to zero
    $L(\mathbf{w}) \leftarrow ||\mathbf{w} - \mathbf{w}_p||_2$
    $\mathbf{x} \leftarrow$ dataset(MNIST or CIFAR) ▷ sample minibatch of images from the dataset
    OPloss $\leftarrow$ odist($f(\mathbf{x}, \mathbf{w}), f(\mathbf{x}, \mathbf{w}_t)$) ▷ distance moved in the output space
    $S \leftarrow$ OPloss $+ \lambda \times L(\mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} S$ ▷ constructing FIP towards sparse submanifold
  **end while**
  **return w**
**end procedure**

**Code specifications.** All the code was written in the PyTorch framework, and the automatic differentiation package was extensively used for constructing the computational graphs and computing gradients for updating the network parameters. The code for constructing the FIPs to the $p$% sparse submanifolds was run on Caltech's high-performance computing cluster using a single GPU for a total time ranging between 2 h and 6 h for final network sparsities below 80%, and between 24 h and 30 h for identifying high-performance networks in submanifolds with larger than 80% sparsity.

**Parameters used.**

- FIP for network sparsification: $\lambda = 1$, $\eta = 0.01$; optimizer used, Adam ($\beta = (0.9, 0.999)$); final (desired) network sparsities for LeNet-300-100 on MNIST: $p$ = [20%, 67%, 89%, 96%, 98.7%, 99%, 99.1%, 99.4%]; final (desired) network sparsities for ResNet20 on CIFAR-10: $p$ = [20%, 36%, 49%, 59%, 67%, 79%, 83%, 89%, 93%, 95%].
- LTH (for LeNet-MNIST): batch-size = 128, model-init = kaiming-normal, batchnorm-init = uniform, pruning-strategy = sparse-global, pruning-fraction = 0.2, pruning-layers-to-ignore = fc.weight, optimizer-name = sgd, learning rate = 0.1, training-steps = 40 epochs.
  LTH (for ResNet20-CIFAR-10): batch-size = 128, model-init = kaiming-normal, batchnorm-init = uniform, pruning-strategy = sparse-global, pruning-fraction = 0.2, optimizer-name = sgd, learning rate = 0.1, training-steps = 160 epochs, momentum = 0.9, gamma = 0.1, weight-decay ≥ 0.0001.

**Implementation of other sparsification methods.** We implemented the LTH for sparsifying both LeNet-300-100 trained on MNIST and ResNet20 trained on CIFAR-10. To do so, we adapted code from https://github.com/facebookresearch/open_lth.

## Adversarial robustness

**Datasets and preprocessing.** The models were trained on the CIFAR-10 dataset and the adversarial examples were generated on the same using the PGD method.

- CIFAR-10: the CIFAR-10 training dataset contains 50,000 RGB images of 10 classes of natural images (like trucks, horses, birds and ships). The test set contains 10,000 additional images from each of the 10 classes.

**Network architecture.** For the adversarial robustness section, we used the VGG16 network[72], which has 16 layers, and a total of 138M trainable parameters.

**Generating an adversarial attack.** We used the PGD method to generate CIFAR-10 data samples that are imperceptibly similar to their original images for humans, but cause performance loss to deep networks. The PGD attack computes the best direction (in image space) to perturb the image such that it maximizes the trained networks' loss on the image and constraining the $L_{inf}$ norm of the perturbation.

The procedure for generating adversarial inputs is detailed below:

- Randomly initialize a VGG16 network and train it on CIFAR-10 (trained network = $\mathbf{w}_t$).
- Take a single image input ($\mathbf{x}$) from the CIFAR-10 dataset and pass it through the trained network, and calculate the gradient of the classification loss (cross-entropy ($C$)) with respect to the input (grad = $\nabla_{\mathbf{x}} C(\mathbf{w}_t, \mathbf{x}, \mathbf{y})$).
- Construct an adversarial input ($\mathbf{x}'$) by taking multiple steps ($S$) in the image input space, where the adversary is within an $\epsilon l_\infty$ bound. $\mathbf{x}^{t+1} = \prod_{x+S}(\mathbf{x}^t + \alpha \mathrm{sgn}(\nabla_{\mathbf{x}} C(\mathbf{w}_t, \mathbf{x}, \mathbf{y})))$. Here we take as many steps ($S$) as required until the adversarial input ($\mathbf{x}^{t+1}$) exits the $\epsilon l_\infty$ bound. We choose $\epsilon = 0.3$ and $\alpha = 2/255$ for generating CIFAR-10 adversarial examples against VGG16 networks.

**Representation diversity score.** We compute the representation diversity score for both ensembles (FIP and DeepNet) by evaluating the standard deviation of the $L_2$ norm of the network's activation across all the networks in the ensemble along each layer for a set of image inputs.

**Coherence between two models.** We compute the overlap of the adversarial subspace between networks in the FIP ensemble and the trained surrogate network by evaluating the cosine distance between the gradients of the loss function of the FIP networks and the trained surrogate network with respect to an input ($x$).

Say, the gradients of the loss function with respect to input ($x$) for the two models are $J_x(\theta_0, x, y)$ and $J_x(\theta_1, x, y)$. The cosine distance between the gradients is evaluated as follows, where CS indicates cosine similarity.

$$\mathrm{CS}(\nabla_x J_0, \nabla_x J_1)) = \frac{< \nabla_x J_0, \nabla_x J_1 >}{|\nabla_x J_0||\nabla_x J_1|}. \quad (6)$$

The cosine distance between the gradients provides a quantitative measure for how likely an adversarial input that affects the surrogate network would attack the model sampled along an FIP.

To evaluate the coherence across all the sampled models in the FIP and a trained surrogate network, we measure the maximum cosine similarity between all pairs of gradient vectors in the set.

$$\max_{a \in 1 \ldots N} \mathrm{CS}(\nabla_x J_a, \nabla_x J_s)). \quad (7)$$

Here $J_a$ refers to the gradient of $N$ networks sampled along the FIP for a single input ($x$), and $J_s$ refers to the gradient of the trained surrogate network for the input ($x$).

### Pseudo-code: FIP for adversarial robust ensembles

**Algorithm 3.** FIP for **adversarially** robust ensembles
**Require**: $\eta$, step size; $\mathbf{w}_t$, network trained on CIFAR-10 dataset; $\boldsymbol{\epsilon} l_\infty$ of adversary perturbation
**Require**: $\delta$, permissible change in output distance; max-iter, number of steps in the FIP

**procedure** FIP-ensemble($\eta$, $\mathbf{w}_t$, $\delta$, $\epsilon$)
  $\mathbf{w} \leftarrow \mathbf{w}_t$
  $ii \leftarrow 0$ ▷ setting counter = 0
  $F \leftarrow \{\}$ ▷ list of networks in the FIP ensemble
  **while** $ii \le$ max-iter
    $(\mathbf{x}, \mathbf{y}) \leftarrow$ dataset(CIFAR-10) ▷ sample minibatch of images from dataset
    $S \leftarrow \mathrm{odist}(f(\mathbf{x}, \mathbf{w}), f(\mathbf{x}, \mathbf{w}_t))$ ▷ output-space distance for varying network weights
    $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} S$ ▷ construct undirected FIP
    $\mathbf{x}' \leftarrow \mathbf{x} + \varepsilon \mathrm{sgn}(\nabla_{\mathbf{x}} C(\mathbf{w}, \mathbf{x}, \mathbf{y}))$
    $H \leftarrow \mathrm{odist}(f(\mathbf{x}, \mathbf{w}), f(\mathbf{x}', \mathbf{w}))$ ▷ output-space distance for perturbed input
    **if** $H \le \delta$ **then**
      $F \leftarrow F \cup \mathbf{w}$
    **end if**
    $ii \leftarrow ii + 1$
  **end while**
  **return** $F$ ▷ returning FIP ensemble with adversarial robustness
**end procedure**

**Code specifications.** All the code was written in the PyTorch framework, and the automatic differentiation package was extensively used for constructing the computational graphs and computing gradients for updating the network parameters. The code for constructing the undirected FIPs in the weight space, followed by sampling a small subset of networks along the FIP, was run on Caltech's high-performance computing cluster using a single GPU for a total time ranging between 2 h and 6 h.

**Parameters used.** To generate ensembles of deep networks, we selected parameters after a grid search to maximize robustness against adversarial failure.

- FIP ensemble: $\eta = 0.01$, $\epsilon = 0.3$, minibatch size = 100, $\delta = 35$ (inputs to the FIP construction/ensemble pseudo-code detailed above).
- Adaptive diversity-promoting ensemble: $\alpha = 2$, $\beta = 0.5$ ($\alpha$ and $\beta$ are parameters maximizing the diversity of ensemble); optimizer used, SGD; momentum = 0.9, learning rate = 0.05, weight-decay = $2 \times 10^{-4}$, batch-size = 128, num-networks-per-ensemble = 3, 5 and 10 (three different ensembles).
- Fast geometric ensembling: model, VGG16; epochs = 40, weight-decay = $3 \times 10^{-4}$, learning-rate-1 = $0.5 \times 10^{-2}$, learning-rate-2 = $1 \times 10^{-2}$, cycles = 2.

**Implementation of other ensemble generation methods for adversarial robustness.** We generated ensembles of deep networks (VGG16) using three state-of-art methods. The first method, 'DeepNet ensemble', was constructed by training multiple independently initialized VGG16 networks. The second method called adaptive diversity promoting is obtained by adapting the code available at https://github.com/P2333/Adaptive-Diversity-Promoting. The third method called fast geometric ensembling is obtained by adapting the code taken from this repository: https://github.com/timgaripov/dnn-mode-connectivity.

**FIP for multiple 'operations' on BERT language model.** We scale our FIP geometric framework to perform multiple operations (like CL and Co) on BERT language models that are very large and are capable of parsing large amounts of text scraped from different internet sources (like, Wikipedia, Yelp, IMDb and so on).

**Datasets and preprocessing.** We performed two operations, namely, (1) network Co and (2) CL on BERT, fine tuned on different language datasets, downloaded from the HF website.

- Wikipedia: the Wikipedia English datasets are downloaded from https://huggingface.co/datasets/wikipedia. We used the Wikipedia dataset on the MLM task.
- Yelp reviews: the Yelp review datasets are obtained from HF, downloaded from https://huggingface.co/datasets/yelp_review_fullhttps://huggingface.co/datasets/yelp_review_full.
- IMDb reviews: the IMDb review datasets are obtained from HF, downloaded from https://huggingface.co/datasets/imdb.
- GLUE dataset: the GLUE dataset is obtained from HF, downloaded from https://huggingface.co/datasets/gluehttps://huggingface.co/datasets/glue. The GLUE tasks performed in this paper are (1) QQPs, (2) SST-2, (3) CoLA, (4) RTE, (5) MNLI, (6) MRPC and (7) QNLI.

**Network architecture.** BERT is a popular transformer model having 12 layers (transformer blocks), each with a hidden size of 768, comprising 12 self-attention heads in each layer having a total of 110M parameters[5]. BERT has been pre-trained on 45 GB of Wikipedia data, using the MLM task, and next-sentence prediction.

*Sentence completion (masking) tasks*. For the masking tasks (where 15% of the words in the input sentence are masked (or blanked)), the BERT network has an MLM head appended to the network. The MLM head produces a three-dimensional tensor as the output, where the dimensions correspond to (1) the number of sentences in a single batch (batch-size), (2) number of blanked-out words in a sentence and (3) number of tokens in the BERT vocabulary (30,000 tokens).

*Sentence classification tasks*. For the sentence classification task, a sentence classifier head was appended to the BERT architecture. Here the classifier head produces a two-dimensional output tensor, where the dimensions correspond to (1) batch-size and (2) number of unique classes in the classification problem.

**Code specifications.** All the code was written in the PyTorch framework, and the automatic differentiation package was extensively used for constructing the computational graphs and computing gradients for updating the network parameters. The code for constructing the FIPs in the BERT weight space for CL on Yelp and IMDb sentiment analysis and for BERT sparsification was run on Caltech's high-performance computing cluster using two GPUs for a total time of 2 h and 6–30 h, respectively.

**Parameters used.**

- FIP for CL: $\eta = 2 \times 10^{-5}$, $\lambda = 1$, $n$ memories from previous task = 2,000/650,000 (0.8% of the previous dataset). Optimizer used, AdamW.
- FIP for BERT sparsification: $\eta = 2 \times 10^{-5}$, $\lambda = 1$. Optimizer used, AdamW. Final (desired) network sparsities for the GLUE task: task ($p$% sparse): RTE (60% sparse), CoLA (50% sparse), STS-B (50% sparse), QNLI (70% sparse), SST-2 (60% sparse), MNLI (70% sparse), QQP (90% sparse), MRPC (50% sparse). Final (desired) network sparsities for Wikipedia sentence completion: [10%, 20%...90%].

**FIP length.** The FIP length is evaluated by sampling a large number of networks along the FIP, and summing the Euclidean distance between all the consecutive pairs of networks. Say, the weights of the networks sampled along the FIP are denoted by $\mathbf{w_1}, \mathbf{w_2}, \mathbf{w_3} \ldots \mathbf{w_n}$.

$$\text{FIP-length} = \sum_{i=2}^{n} ||\mathbf{w_i} - \mathbf{w_{i-1}}||_2$$

**Perplexity score: language models.** Perplexity is an evaluation metric used to measure how 'surprised' a language model is when it encounters a new task. That is, a higher perplexity implies more surprise, suggesting that the language model does not have much insight into how language works. Mathematically, we define perplexity as the exponential of the cross-entropy loss on the evaluation dataset:

$$\text{PPL}(\theta) = \exp\left[\sum_{i=1}^{n_e} l(\theta, x_i)\right],$$

where $\theta$ is the parameter of the BERT model and $x_1, x_2 \ldots x_{n_e}$ are the $n_e$ inputs from the evaluation dataset. $l(\theta, x_i)$ evaluates the cross-entropy loss of a BERT model parameterized by $\theta$ for a single input $x_i$.

### Constructing FIP ensemble for adversarial robustness

**Selection criteria for FIP ensemble.** Having constructed an FIP in the weight space, beginning from a deep network trained on CIFAR-10, we introduce the selection criteria to sample diverse networks along the FIP to construct the FIP ensemble. As we want the FIP ensemble to be robust to adversarial input perturbation, we generate random perturbations in the image space (within an $\epsilon l_\infty$ ball) and compute the distance moved in the networks' output space for a small perturbation in the image space.

We record the distance moved in the networks' output space (across all the networks in the constructed FIP) and plot a distribution of the distance moved in the output space for a small perturbation in the image input space. We find that some networks along the FIP exhibit smaller perturbation in the output space and have a narrower distribution across 10k perturbed training inputs, whereas others exhibit larger perturbation in the output space. We choose networks that exhibit a smaller perturbation in the output space for constructing the FIP ensemble.

## Data availability

All datasets used in the paper were obtained from public data sources and repositories. A complete list of the public data sources with links is available via GitHub at https://github.com/Guruprasad93/FlexibleMachineLearning/blob/main/README.md.

## Code availability

Executable Python code and documentation applying the FIP path-finding algorithms to example problems are available via GitHub at https://github.com/Guruprasad93/FlexibleMachineLearning with DOIs and citations[73]. The code is provided without restriction under the MIT license.

## References

1. He, K. et al. Masked autoencoders are scalable vision learners. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* 16000–16009 (IEEE, 2022).
2. Taori, R. et al. Alpaca: a strong, replicable instruction-following model. *Stanf. Center Res. Found. Models* **3**, 7 (2023).
3. Bommasani, R. et al. On the opportunities and risks of foundation models. Preprint at https://arxiv.org/abs/2108.07258 (2021).
4. Brown, T. et al. Language models are few-shot learners. *Adv. Neural Inf. Process. Syst.* **33**, 1877–1901 (2020).
5. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. Preprint at https://arxiv.org/abs/1810.04805 (2018).
6. OpenAI. GPT-4 technical report. Preprint at https://arxiv.org/abs/2303.08774 (2023).
7. Hoffmann, J. et al. An empirical analysis of compute-optimal large language model training. *Adv. Neural Inf. Process. Syst.* **35**, 30016–30030 (2022).
8. Dosovitskiy, A. et al. An image is worth 16x16 words: transformers for image recognition at scale. In *International Conference on Learning Representations* (2020).

9. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning representations by back-propagating errors. *Nature* **323**, 533–536 (1986).
10. Minxha, J., Adolphs, R., Fusi, S., Mamelak, A. N. & Rutishauser, U. Flexible recruitment of memory-based choice representations by the human medial frontal cortex. *Science* **368**, eaba3313 (2020).
11. Mau, W., Hasselmo, M. E. & Cai, D. J. The brain in motion: how ensemble fluidity drives memory-updating and flexibility. *eLife* **9**, e63550 (2020).
12. Stringer, C. et al. Spontaneous behaviors drive multidimensional, brainwide activity. *Science* **364**, eaav7893 (2019).
13. Masset, P., Qin, S. & Zavatone-Veth, J. A. Drifting neuronal representations: bug or feature? *Biol. Cybern.* **116**, 253–266 (2022).
14. Geva, N., Deitch, D., Rubin, A. & Ziv, Y. Time and experience differentially affect distinct aspects of hippocampal representational drift. *Neuron* **111**, p2357–2366.e5 (2023).
15. Driscoll, L. N., Duncker, L. & Harvey, C. D. Representational drift: emerging theories for continual learning and experimental future directions. *Curr. Opin. Neurobiol.* **76**, 102609 (2022).
16. Machta, B. B., Chachra, R., Transtrum, M. K. & Sethna, J. P. Parameter space compression underlies emergent theories and predictive models. *Science* **342**, 604–607 (2013).
17. Hochreiter, S. & Schmidhuber, J. Flat minima. *Neural Comput.* **9**, 1–42 (1997).
18. Hochreiter, S. & Schmidhuber, J. Simplifying neural nets by discovering flat minima. *Adv. Neural Inf. Process. Syst.* **7**, 529–536 (1994).
19. Tsuzuku, Y., Sato, I. & Sugiyama, M. Normalized flat minima: exploring scale invariant definition of flat minima for neural networks using PAC-Bayesian analysis. In *International Conference on Machine Learning* (eds Daumé III, H. & Singh, A.) 9636–9647 (PMLR, 2020).
20. Amari, S. *Information Geometry and its Applications* Vol. 194 (Springer, 2016).
21. Benn, I. & Tucker, R. *An Introduction to Spinors and Geometry with Applications in Physics* (Adam Hilger Ltd, 1987).
22. Mache, D. H., Szabados, J. & de Bruin, M. G. *Trends and Applications in Constructive Approximation* Vol. 151 (Springer Science & Business Media, 2006).
23. Seleznova, M., Weitzner, D., Giryes, R., Kutyniok, G. & Chou, H.-H. Neural (tangent kernel) collapse. *Adv. Neural Inf. Process. Syst.* **36**, 16240–16270 (2024).
24. Jacot, A., Gabriel, F. & Hongler, C. Neural tangent kernel: convergence and generalization in neural networks. *Adv. Neural Inf. Process. Syst.* **31**, 8580–8589 (2018).
25. Golikov, E., Pokonechnyy, E. & Korviakov, V. Neural tangent kernel: a survey. Preprint at https://arxiv.org/abs/2208.13614 (2022).
26. Seleznova, M. & Kutyniok, G. Neural tangent kernel beyond the infinite-width limit: effects of depth and initialization. In *International Conference on Machine Learning* (eds Chaudhuri, K. et al.) 19522–19560 (PMLR, 2022).
27. Weisstein, E. W. Metric tensor. https://mathworld.wolfram.com/ (2014).
28. Tu, L. W. *Differential Geometry: Connections, Curvature, and Characteristic Classes* Vol. 275 (Springer, 2017).
29. Tao, T. & Vu, V. Random covariance matrices: universality of local statistics of eigenvalues. *Ann. Probab.* **40**, 1285–1315 (2012).
30. Kaushik, P., Gain, A., Kortylewski, A. & Yuille, A. Understanding catastrophic forgetting and remembering in continual learning with optimal relevance mapping. In *Fifth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems* (NeurIPS, 2021).
31. van de Ven, G. M., Siegelmann, H. T. & Tolias, A. S. Brain-inspired replay for continual learning with artificial neural networks. *Nat. Commun.* **11**, 4069 (2020).
32. Hu, E. J. et al. LoRA: low-rank adaptation of large language models. In *International Conference on Learning Representations* (2021).
33. Kirkpatrick, J. et al. Overcoming catastrophic forgetting in neural networks. *Proc. Natl Acad. Sci. USA* **114**, 3521–3526 (2017).
34. Lopez-Paz, D. & Ranzato, M. Gradient episodic memory for continual learning. *Adv. Neural Inf. Process. Syst.* **30**, 6470–6479 (2017).
35. Blalock, D., Ortiz, J. J. G., Frankle, J. & Guttag, J. What is the state of neural network pruning? In *Proc. Machine Learning and Systems* (eds. Dhillon, I. et al.) 129–146 (Conference on Machine Learning and Systems, 2020).
36. Chen, T. et al. Chasing sparsity in vision transformers: an end-to-end exploration. *Adv. Neural Inf. Process. Syst.* **34**, 19974–19988 (2021).
37. Touvron, H. et al. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning* (eds Meila, M. & Zhang, T.) 10347–10357 (PMLR, 2021).
38. Wortsman, M. et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning* (eds Chaudhuri, K. et al.) 23965–23998 (PMLR, 2022).
39. Vaswani, A. et al. Attention is all you need. *Adv. Neural Inf. Process. Syst.* **30**, 6000–6010 (2017).
40. Liu, Y. et al. RoBERTa: a robustly optimized BERT pretraining approach. Preprint at https://arxiv.org/abs/1907.11692 (2019).
41. Wolf, T. et al. Transformers: state-of-the-art natural language processing. In *Proc. 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (eds Liu, Q. & Schlangen, D.) 38–45 (Association for Computational Linguistics, 2020).
42. Lyle, C., Rowland, M., Ostrovski, G. & Dabney, W. On the effect of auxiliary tasks on representation dynamics. In *International Conference on Artificial Intelligence and Statistics* (eds Banerjee, A. & Fukumizu, K.) 1–9 (PMLR, 2021).
43. Jaderberg, M. et al. Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations* (2022).
44. Hoffman, J., Roberts, D. A. & Yaida, S. Robust learning with Jacobian regularization. Preprint at https://arxiv.org/abs/1908.02729 (2019).
45. Chen, Z & Liu, B. *Lifelong Machine Learning* Vol. 12 (Springer Nature, 2018).
46. Parisi, G. I., Kemker, R., Part, J. L., Kanan, C. & Wermter, S. Continual lifelong learning with neural networks: a review. *Neural Netw.* **113**, 54–71 (2019).
47. Alzubaidi, L. et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *J. Big Data* **8**, 53 (2021).
48. Ramasesh, V. V., Lewkowycz, A. & Dyer, E. Effect of scale on catastrophic forgetting in neural networks. In *International Conference on Learning Representations* (2022).
49. Mirzadeh, S. I. et al. Architecture matters in continual learning. Preprint at https://arxiv.org/abs/2202.00275 (2022).
50. Farajtabar, M., Azizan, N., Mott, A. & Li, A. Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics* (eds Chiappa, S. & Calandra, R.) 3762–3773 (PMLR, 2020).
51. Zenke, F., Poole, B. & Ganguli, S. Continual learning through synaptic intelligence. In *International Conference on Machine Learning* (eds P recup, D. & Teh, Y.-W.) 3987–3995 (PMLR, 2017).
52. Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T. & Wayne, G. Experience replay for continual learning. *Adv. Neural Inf. Process. Syst.* **32**, 350–360 (2019).

53. Shin, H., Lee, J. K., Kim, J. & Kim, J. Continual learning with deep generative replay. *Adv. Neural Inf. Process. Syst.* **30**, 2994–3003 (2017).
54. Rusu, A. A. et al. Progressive neural networks. Preprint at https://arxiv.org/abs/1606.04671 (2016).
55. Yoon, J., Yang, E., Lee, J. & Hwang, S. J. Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations* (2018).
56. Wortsman, M. et al. Supermasks in superposition. *Adv. Neural Inf. Process. Syst.* **33**, 15173–15184 (2020).
57. Frankle, J. & Carbin, M. The lottery ticket hypothesis: finding sparse, trainable neural networks. In *International Conference on Learning Representations* (2018).
58. Han, S., Pool, J., Tran, J. & Dally, W. Learning both weights and connections for efficient neural network. *Adv. Neural Inf. Process. Syst.* **28**, 1135–1143 (2015).
59. LeCun, Y., Denker, J. & Solla, S. Optimal brain damage. *Adv. Neural Inf. Process. Syst.* **2**, 598–605 (1989).
60. Evci, U., Gale, T., Menick, J., Castro, P. S. & Elsen, E. Rigging the lottery: making all tickets winners. In *International Conference on Machine Learning* (eds Daumé III, H. & Singh, A.) 2943–2952 (PMLR, 2020).
61. Liu, S., Yin, L., Mocanu, D. C. & Pechenizkiy, M. Do we actually need dense over-parameterization? In-time over-parameterization in sparse training. In *International Conference on Machine Learning* (eds Meila, M. & Zhang, T.) 6989–7000 (PMLR, 2021).
62. Mocanu, D. C. et al. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nat. Commun.* **9**, 2383 (2018).
63. Liu, S., Mocanu, D. C., Matavalam, A. R. R., Pei, Y. & Pechenizkiy, M. Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. *Neural Comput. Appl.* **33**, 2589–2604 (2021).
64. Brahma, S., Zablotskaia, P. & Mimno, D. Breaking BERT: evaluating and optimizing sparsified attention. Preprint at https://arxiv.org/abs/2210.03841 (2022).
65. Madry, A., Makelov, A., Schmidt, L., Tsipras, D. & Vladu, A. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations* (2018).
66. Pang, T., Xu, K., Du, C., Chen, N. & Zhu, J. Improving adversarial robustness via promoting ensemble diversity. In *International Conference on Machine Learning* (eds Chaudhuri, K. & Salakhutdinov, R.) 4970–4979 (PMLR, 2019).
67. Buckman, J., Roy, A., Raffel, C. & Goodfellow, I. Thermometer encoding: one hot way to resist adversarial examples. In *International Conference on Learning Representations* (2018).
68. Mehta, P. & Schwab, D. J. An exact mapping between the variational renormalization group and deep learning. Preprint at https://arxiv.org/abs/1410.3831 (2014).
69. Smale, S. Mathematical problems for the next century. *Math. Intell.* **20**, 7–15 (1998).
70. Mumford, D. & Desolneux, A. *Pattern Theory: The Stochastic Analysis of Real-World Signals* (CRC, 2010).
71. Mangrulkar, S. et al. PEFT: state-of-the-art parameter-efficient fine-tuning methods. *GitHub* https://github.com/huggingface/peft (2022).
72. Simonyan, K. & Zisserman, A. Very deep convolutional networks for large-scale image recognition. Preprint at https://arxiv.org/abs/1409.1556 (2014).
73. Raghavan, G. Guruprasad93/FlexibleMachineLearning: FIP for catastrophic forgetting of neural netwrsk. *Zenodo* https://doi.org/10.5281/zenodo.10867285 (2024).


## Acknowledgements

We thank E. Winfree, U. Rutishauser, T. Siapas, V. Chandrasekaran, C. Re, B. V. Roo, J. Feiber, J. Gill and J. Schnitzer for helpful discussions. We thank the Chen Center at Caltech, Heritage Medical Research Institute, the Packard Foundation and the Rothenberg Innovation Initiative for research support.

## Author contributions

G.R. and M.T. developed the differential geometry framework and performed the theoretical and computational analyses of the models. B.T. developed an efficient computational implementation of the FIP and performed the experiments and analysis on vision transformers. D.S., S.N.H. and R.L. developed the code and performed the numerical experiments on NLP tasks and transformers. M.T. and G.R. wrote the paper with contributions from all authors.

## Competing interests

The authors declare no competing interests.

## Additional information

**Extended data** is available for this paper at https://doi.org/10.1038/s42256-024-00902-x.

**Supplementary information** The online version contains supplementary material available at https://doi.org/10.1038/s42256-024-00902-x.

**Correspondence and requests for materials** should be addressed to Guruprasad Raghavan or Matt Thomson.

**Peer review information** *Nature Machine Intelligence* thanks Pratik Chaudhari, Andrey Gromov and the other, anonymous, reviewer(s) for their contribution to the peer review of this work.

https://doi.org/10.1038/s42256-024-00902-x

**Extended Data Table 1 | Hyper-parameter settings for rank, and alpha for LoRA experiments fine-tuning on SplitCIFAR in main text**

| Algorithm | LoRA Rank | alpha (LoRA) | Yelp Accuracy % | IMDB Accuracy % |
|---|---|---|---|---|
| LoRA | 1 | 2 | $70.5 \rightarrow 61.10$ | 82.70 |
| LoRA | 4 | 8 | $70.5 \rightarrow 48.00$ | 90.00 |
| LoRA | 8 | 16 | $70.5 \rightarrow 50.60$ | 89.60 |
| LoRA | 16 | 32 | $70.5 \rightarrow 30.30$ | 91.40 |
| LoRA | 32 | 64 | $70.5 \rightarrow 23.90$ | 91.40 |
| **FIP** | - | - | $70.5 \rightarrow 70.0$ | 92.5 |

Table shows explicit accuracy values for SplitCIFAR fine-tuning using LoRA for different values of LoRA the rank parameter. The final entry (LoRA weights + original weights) shows accuracy for combined application of LoRA and complete network fine-tuning. Bold indicates row showing FIP performance.

https://doi.org/10.1038/s42256-024-00902-x

**Extended Data Table 2 | Application of FIP to SplitCIFAR continual learning with convolutional neural network architectures**

| Algorithm | Network Architecture | Final Accuracy % (Task paradigm) | Ref. |
|---|---|---|---|
| **FIP** | ResNet18 | **82.54** (20) | ours |
| EWC | ResNet18 | 45.0 (20) | Kirkpatrick [33] |
| EWC | CNN 4 layer | 60.3 (5) | Kirkpatrick [33] |
| RMN | Resnet18 | 80.01 (20) | Kaushik [30] |
| RMN | CNN 4 layer | 70.02 (5) | Kaushik [30] |
| GEM | Resnet18 | 68.0 (20) | Lopez-Paz [34] |
| BGR | CNN 5 layer | 80.0 (10) | Van De Ven [31] |

Accuracy of FIP on 20 task SplitCIFAR for CNN architectures and comparison to Elastic Weight Consolidation (EWC), Relevance Mapping Networks (RMN), Gradient Episodic Memory (GEM), Brain inspired generative replay (BR).Results for RMN, EWC, GEM, and BR are compiled from the literature. Bold font used to highlight that FIP has highest accuracy in continual learning task across methods.