Scaling Up Deep Learning for fMRI Brain Models

Sana Ahmadi

Concordia University

Supervisors:

Prof. Tristan Glatard

Prof. Pierre Bellec

A report for comprehensive exam
Requirements for the PhD in Computer Science

December 2021

Acknowledgment

This work is a part of the Courtois NeuroMod project. The goal of Courtois NeuroMod project is to train artificial neural networks using intensive experimental data on individual human brain activity and behaviour. This work is also funded by the Canada Research Chairs Program.

Abstract

Brain encoding and decoding using functional magnetic resonance imaging (fMRI) are active research areas in neuroscience. In the encoding tasks the brain responses are computed using stimuli features while in the decoding tasks the stimuli features are computed based on brain responses. The main goal of brain encoding and decoding models is to understand the fundamental principles of brain organization. This aim is achievable by leveraging two fundamental factors: large scale dataset and efficient model. To achieve high performance, dataset should be extended in terms of stimuli diversity and the number of subjects. By increasing the number of stimuli categories, brain encoding and decoding models are able to learn the huge feature space of stimuli. Furthermore, increasing the number of subjects improve the model generalisability. The second factor to improve the understanding of the complex computations in the human brain is leveraging of efficient models. Deep Learning (DL) techniques are gaining much popularity due to promising results for brain encoding and decoding.

Even though the large-scale datasets and efficient models improve the performance of brain encoding and decoding noticeably, they increase the computational and memory requirements. Consequently, the scalability of DL systems is essential for adaptable and efficient training. This means that big data techniques are necessary to manage the training process in terms of memory, time, and accuracy.

In this report, we first review the state-of-the-art brain encoding and decoding approaches. These approaches are based on Convolution Neural Networks(CNNs), Generative Adversarial Networks (GANs), and Recurrent neural networks (RNNs). In the second part, we review memory management and Parallelization-based approaches for efficient training process in the large scale deep learning systems. Finally, the state of the art bench marking results of different hardware platforms (GPU, CPU and TPU) will be reviewed. In this report, we investigate the strengths and limitations of different big data approaches with respect to brain encoding and decoding models.

List of Contents

List of Figures

List of Tables

1 Introduction

The human brain is a hierarchical computing system with billions of neurons as the computing units. Cognitive neuroscience aims to discover the principles of functional brain organization by leveraging brain imaging data. There are several methods for brain imaging including functional Magnetic Resonance Imaging (fMRI), electroencephalography (EEG), Magnetoencephalography (MEG), and functional Near Infrared Spectroscopy (fNIRS).

Particularly, fMRI is an efficient method to measuring time varying changes of the blood-oxygen-level-dependent (BOLD) signal at the different parts of the brain. When a neuronal region is activated, the BOLD signal level starts to increase since the oxygenated blood flows to this tissue. After 5 seconds the BOLD signal level is at its maximum value. When there is no more activation in the tissue, the BOLD signal level decreases to its basic state after 15 seconds [18, 45].

The brain activity data is measured in the 3D voxels, each voxel has a specific dimension (a cube in space) and value. The spatial resolution of the fMRI image depends on the voxel size. The standard resolution is 1mm and the whole brain is considered a 3D matrix with size of $256 \times 256 \times 256$ voxels. With this dimensions, the whole brain data is presented with 17 million values for each time point (volume). As fMRI imaging is a time series process then for one scanning run a series of 3D images are recorded [58, 19].

The fMRI data with high resolution helps cognitive neuroscience researchers to understand brain functions with more details. There are three challenge in cognitive neuroscience research area: 1) determine the content of representations in the specific region of brain, 2) describe how the stimuli feature space is encoded in the specific region of brain, and 3) bridge these two levels of information. To address these challenges, brain encoding and decoding approaches analyze fMRI data to understand the complex computations in the brain system. In the encoding tasks the brain responses are computed using stimuli features. In the decoding tasks the stimuli features are computed based on brain responses [12]. Figure 1 illustrates the brain encoding and decoding tasks which are inverse of each other. An example of a brain decoding model is a classifier where the voxel values are as input of classifier and the class of stimuli images (inanimate object, animate object, food, outdoor scene,...) is predicted. The input of the encoder model is features of an image and the brain response is predicted. For instance, the features of images are extracted using pre-trained networks such as VGG26, Alexnet. In the next step, the brain encoding can be trained using the output of different layers of CNN and brain responses.

In the literature, enormous computational methods have been proposed for brain encoding and decoding with respect to different type of stimuli such as auditory [24], language [40], emotion [53], and visual stimuli [28]. Among different approaches, deep learning has shown significant performance. Deep learning approaches extract hierarchical features from the training samples which is essential to understand brain hierarchical mechanism.

The previous studies have made promising results in brain encoding and decoding models, however these approaches suffer from limited size of fMRI datasets i.e pairs of (fMRI, stimuli). The limited size of dataset leads to over-fitting problem in deep learning based approaches [9, 12]. Over the past few years, the quantity and quality of fMRI datasets have increased rapidly. The size of fMRI datasets increase in different scales such as increasing the number of subjects, scanning hours for each subject, spatio-temporal resolution [6, 42]. Consequently, there is an urgent call to develop scalable brain encoding and decoding models due to computation and memory bottlenecks. In this report, we will see how big data approaches provide efficient and practical computing platforms for large scale brain encoding and decoding models.

In chapter 2, we present the state of the art DL- based brain encoding and decoding approaches

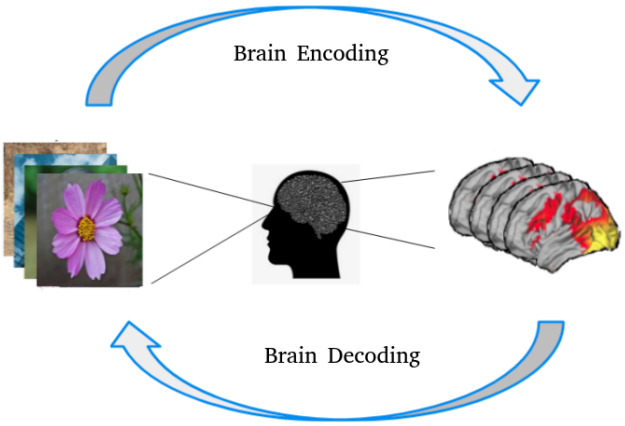


Figure 1: Brain encoding and decoding

with a focus on visual stimuli. In chapter 3, we will answer this question: why is scalability important for brain encoding and decoding? In chapter 4 and 5, we present the Parallelism-based and Memory-based big data techniques respectively. These techniques provide an efficient and adaptable training process for large scale DL systems. Furthermore, the drawbacks and limitations of these approaches will be investigated with respect to the brain encoding and decoding models. In chapter 6, state of the art infrastructures for Deep Learning are introduced. At the end of this report, we are able to recognize which of the big data technique and platform should be selected for a brain encoding and decoding models.

2 Review of brain encoding and decoding models

Brain encoding and decoding are central research areas in the cognitive neuroscience field. In brain encoding and decoding, a DL model is trained using pairs of (fMRI, stimuli) to discover the relationship between brain activity and stimuli. In the encoding, the brain activities are predicted while in the decoding tasks the stimuli features are predicted. In the following, we present the state of the art brain encoding and decoding methods.

2.1 Correspondence between CNNs and the visual cortex

Convolutional neural networks (CNNs) are inspired by the human brain system. They are able to extract information in a hierarchical way using convolution and pooling operations. Thanks to efficient extracted features, CNN- based models have achieved human-level performance in various computer vision tasks such as object detection and image segmentation. The architecture of these models relies on a biological neural network with virtual neurons. Hence Neuroscience discoveries play an important role in developing new models. On the other hand, comparing brain functions and CNNs provides significant insight into the brain system. It is shown that computer vision models has great impact on decoding the process of pattern recognition in the brain. Therefore what we might see in the near future is a real cycle between Neuroscience and developing efficient DL modles.
In paper [54], the authors compared the activation of CNN units with brain response to the dynamic

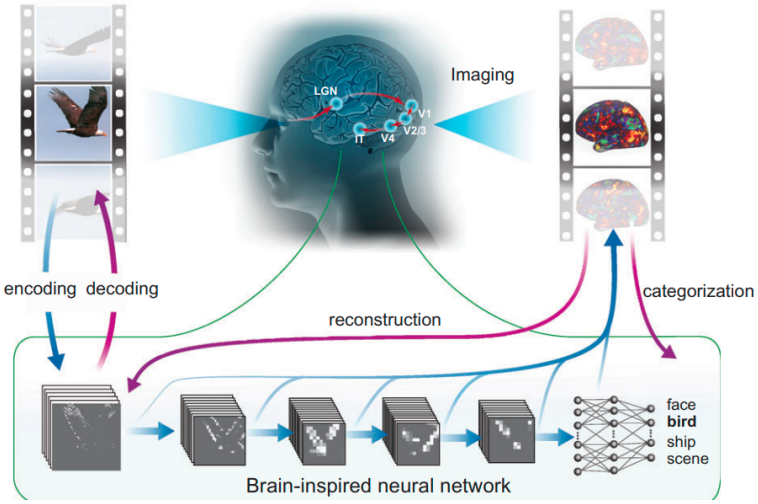


Figure 2: Comparing the activation of CNN units with brain response. This figure was extracted from [54]

virtual stimuli. They aimed to find a brain hierarchical visual map based on the correlation between the individual brain regions and the extracted features in CNN. In this study 374 video clips in 18 8-min segments and 598 videos in 5 8-min segments were presented to subjects and fMRI data was collected. The content of the frames are labeled in 15 classes. By leveraging a set of voxel-wise linear regressions, the brain responses were predicted. In the training process, 10,214 voxels (approximately 17.2% of the whole brain) were presented to the regression models. The results showed that the activation of CNN layers have high potential to predict the response of different brain regions. In this work, the average prediction correlation (r) across the subjects is 0.8.
Even though the result of this study was promising, we can point to some limitations of this work. The dynamic visual stimuli (movie) provides temporal information. This temporal information can

be helpful to increase the prediction accuracy; however in this study, there is lack of strategy to utilize temporal dynamics information. Applying Long short term memory (LSTM) can be helpful to capture the temporal information. Another limitation of this work is that it ignores the connection between brain regions. As we know, there is an information flow in the forward and backward pass of the brain functions. The Bidirection model (bidirection-LSTM [47]) is a potential approach to solve this drawback to improve the results.

In another work [23], a decoding model was trained to predict the activation of CNN units. The

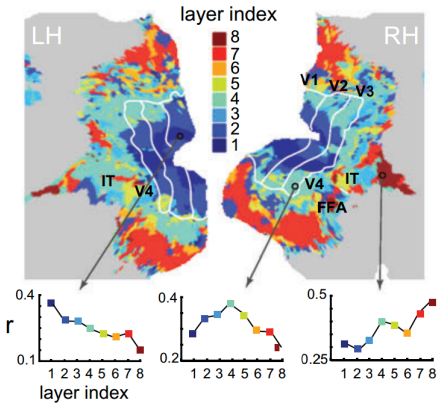


Figure 3: Alignment between the human visual system and the CNN activation. This figure was extracted from [54]

input of this model was brain response and the output was activation of the CNN units. The goal of this study was to measure the similarity between the CNN extracted features and human brain visual system. The similarity was computed by the correlation coefficient. The fMRI signal was recorded when the images from 200 object categories were presented for 5 subjects. Among these categories, 150 categories were randomly selected for the training dataset and 50 categories were considered for the test dataset. The training dataset includes eight images for each category (totally 1200 images). The test dataset contains one image for each category (totally 50 images). The pretrained network VGG19 and Alexnet were used to extract features from the stimuli images. In the brain decoding model, the voxels $V1 - V4$ and HVC regions were considered to present brain activities. A regression model was trained to predict the values of a single CNN unit. Therefore multiple decoder models were trained independently to predict all CNN unit values. Figure 3 shows the correlation coefficient between CNN layers and brain responses.

The main limitation of this study is that the brain model was trained by a small number of image categories. The natural images have expanded dimensional features therefore a limited number of categories leads to a poor decoding model. For instance, it is possible that the voxels which are strongly activated by the low level of the features were mapped with high level features wrongly. Another limitation of this study is that the brain region connections (information flow) were ignored; because the brain activation was trained by a set of independent linear models. The last drawback of this study is that, the authors only considered the $V1 - V4$ and HVC ROIs, while other regions of brain which contain important information were neglected.

2.2 CNN-based brain encoding and decoding models

In [48], the authors proposed a generative model to describe the principles of the computations in neural systems. The goal of this study was to capture the information flow between brain regions.

The proposed model is a CNN based model designed in such a way that the convolution filters represent the stimuli features in a specific brain region. The CNN layers follow the topographical connections between the brain regions to present the information flow in the visual brain system. Fig.4 shows the architecture of the encoding model for the initial visual system regions. The input of the model is 48 frames of dynamic visual stimuli. The size of the kernels in the layers are different. For instance the 3D convolution kernel size in the first layer is $1 \times 1 \times 1$ to account the rentibal computations (linear computing), in the second layer the kernel size is $7 \times 7 \times 7$ to report the V1 region information and in the other layers the kernel size is $3 \times 3 \times 3$. The training parameters are demonstrated with blue rectangles in Fig.4. These parameters are updated in the backpropagation using the difference between real and predicted activation value for each voxel. In this approach, brain was encoded using tensors with four dimensions [C, X, Y, T] where these dimensions indicate channel C, spatial location (x,y), and time step t. A tensor element C [i,:,:,:] is able to present brain activity of a specific brain region. This model was trained using a large scale fMRI dataset. Thirty episodes of seasons 2 to 4 of the Doctor Who TV series were presented to one subject (male, age 27.5).

The main contribution of the NIF model is that instead of modeling brain encoding for each region

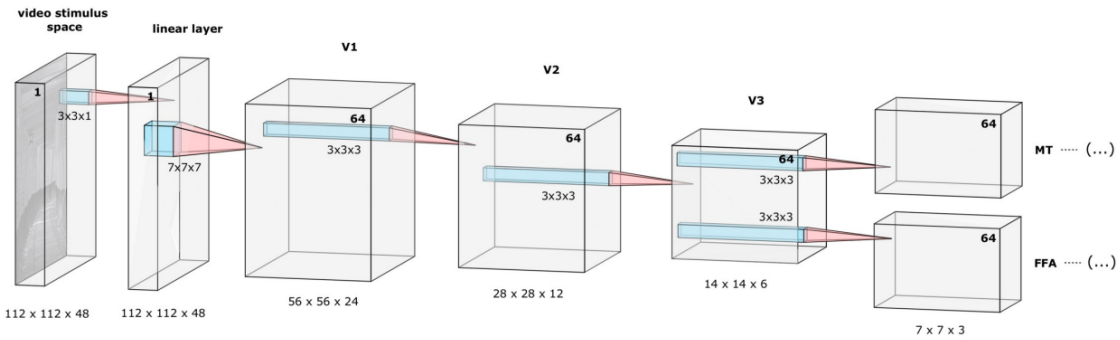


Figure 4: Architecture of NIF model. This figure was extracted form [48]

independently, they considered the information flow in different regions. The connectivity between brain regions and temporal information are captures through applying 3D convolution(X, Y, T) layers. Furthermore, the proposed brain encoding model was trained directly to predict brain responses. In other words, there is no frozen layer based on pre-trained networks in the proposed architecture. To improve the results two suggestions are presented in the following:

1) Adding the LSTM layers to the NIF model architecture. The idea behind of this suggestion is that the LSTM model by leveraging memory gates is able to capture temporal information rather than stats brain encoding.

2) In this work authors considered only the information flow in the forward pass. We suggest applying the bidirectional model [47] to capture the flow information in both forward and backward pass.

In the following, we will review another study which proposed a CNN-based brain model to address the overfitting issue. Overfitting issue is one of the common challenges in brain encoding and decoding. In [9], the authors leverage unlabeled fMRI data (no information about stimuli image) and unlabeled images (no information about brain activity) in the training process. The training process includes four steps (Fig.5):

a) The encoder model (E) was trained using the supervised data (pairs of (Image, Label)).

b) The encoder model was used to train the decoder model (D) in such a way that the encoder and decoder were concatenated back to back (ED, DE).

c) The ED model was trained using both supervised data and unsupervised test fMRI data.
d) The DE model was trained using both supervised data and unsupervised natural images.
The decoder architecture includes three blocks, and each block consists of convolution, ReLU activation, x2 up-sampling, and batch normalization. The encoder consists of pretrained AlexNet conv1 weights, batch normalization and three convolution blocks (Fig.5).

In this work, stimuli dataset were included 1250 images. These images were extracted from 200

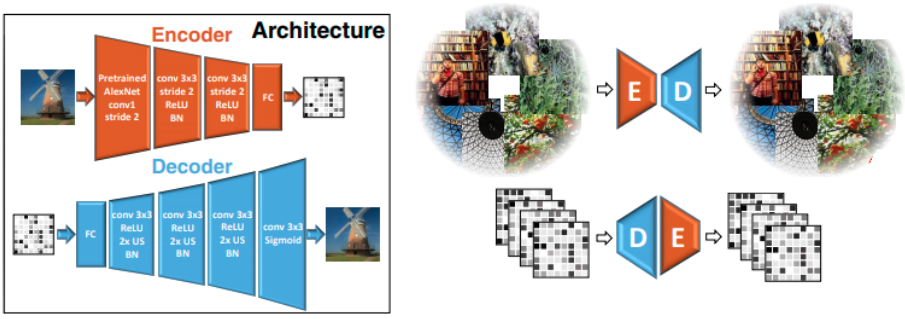


Figure 5: Back to back training system using supervised and unsupervised data. This figure was extracted from [9]

categories of ImageNet dataset [16]. For the brain representation, approximately 4500 voxels from the visual cortex were extracted. The ED model was trained using 50k unlabeled images from ImageNet [16].
The main contribution of this paper is that the brain decoder was adapted by leveraging the statistics of both fMRI and stimuli feature space. The statistics of feature spaces were provided by using unsupervised data. The beck to back training (ED and DE) can be applied to extend and improve other brain encoding and decoding approaches.
The limitation of proposed approach is that the pre-trained network (AlexNet) was applied instead of training the decoder model directly. To improve the model, we suggest that the proposed model should be trained by large scale dataset. The main idea behind this suggestion is that the large scale dataset provides an opportunity for end-to-end brain encoding (E) and decoding (D) training. Also, another limitation of this work is that the authors only used the vision ROIs in the training process and ignored the signal from other regions while these regions carry the important information.

2.3 Adversarial-based brain encoding and decoding models

In [50], authors proposed an approach to train a decoding model directly instead of using the pre-trained network weights in the image reconstruction. The core of this approach is a generative adversarial network (GAN) which generator and discriminator are CNN based models. The generator aims to reconstruct the image from fMRI data and the discriminator aims to distinguish the reconstructed image from the real image. The generator reconstruct an image as similar as possible to the original image by leveraging a pre-trained network. In this process, the comparator (Alexnet-based network) extracts and compare features based on the last convectional layer. Fig 6 demonstrates the combination of three sub-models. The generator architecture includes three fully connected layers followed by six deconvolution layers (upsampling), the discriminator architecture includes five convolutional layers, an average pooling layer and followed by two fully connected layers. The activation function of the last fully connected layer is 2-way softmax to classify the input image.

The proposed model was trained using 6,000 training pairs of (natural images and fMRI responses). The voxels of the low level visual regions (V1, V2, V3,V4) and high level visual regions (lateral occipital complex, fusiform face area, and parahippocampal) were used to represent the brain activity. In this work, the brain response to the stimuli image was presented as a $v \in R^D$ where D is the number of voxels in the vision area of the brain. To improve the model, we suggest another approach to befit high level information of fMRI data such as spatial information and correlation between regions. To consider the spatial information of fMRI data we suggest applying 3D convolve layers [52] instead of the fully connected layers in the generator. Also Graph Convolution Network (GCN) [60] can be used for the "generator" instead of the architecture which proposed by the authors. The input of the GCN model is an irregular brain graph in the non-Euclgidean space. In the brain graph the nodes and the edges represent the parcels and the connections between parcels respectively. The weights of the edges are defined based on the correlation between two regions in terms of functions and topography [60]. The GCN models can be trained in the spatial domain using adjacency matrix or spectral domain using eigenvalues and eigenvectors of graph Laplacian [60].

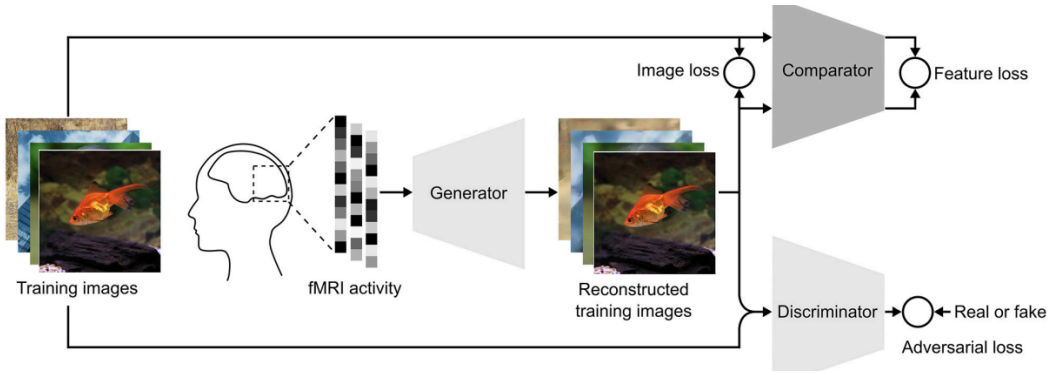


Figure 6: End to End training for image reconstruction using fMRI data. This figure was extracted from [50]

2.4 RNN-based brain encoding an decoding models

In [47], the authors proposed a bidirectional approach to classify the visual stimuli categories using fMRI data. The proposed approach was inspired by the information flow in feedforward and backward passes of the human visual system. In this study, the bidirectional LSTM module was applied to extract the features from five visual areas (V1, V2, V3, V4, and LO). Fig. 7 shows two main steps of the proposed method. In the first step, through the encoding task, 100 voxels were selected from each of the visual area based on the correlation results. In the second step, using selected voxels the bidirectional LSTM was trained.

In this study, 1870 images were presented to the subjects to record fMRI data. These images were selected from Berkely computer vision dataset [33] and three levels of categories (5, 10, and 23 categories) were considered in the experiments. The experiment results showed that tracking the information flow between the visual regions improved the decoding accuracy significantly.

The main limitation of the proposed approach is that the selected voxels (voxels with high correlation in the brain encoding step) are not reliable. The first reason is that the natural image feature space is very expanded and 1840 images are not able to cover the feature space comprehensively. Consequently, the trained model works only for the presented image categories. Training the model

on a large-scale dataset will be able to meet this challenge.

The second reason why the selected voxels are not trustable is that a fixed number of the voxels were selected for each visual area (100 voxels per region). Due to the hierarchical brain visual system, the activation of the voxels in different regions depend on the stimuli image. To improve the performance of the proposed approach, we suggest that non-equal number of the voxels can be selected for different regions. The number of the voxels in each region can be a function of the image features. It is worth to note that the total size of the feature representation for fMRI data should be fix (for this study the total size of selected voxels should be 500).

As conclusion, although previous brain encoding and decoding approaches have made significant advances via DL models, existing approaches still require improvement. Current approaches can be improved using advanced DL techniques. To train such models a large scale dataset is essential. In a nutshell, large scale training process provides opportunity for more accurate predictions.

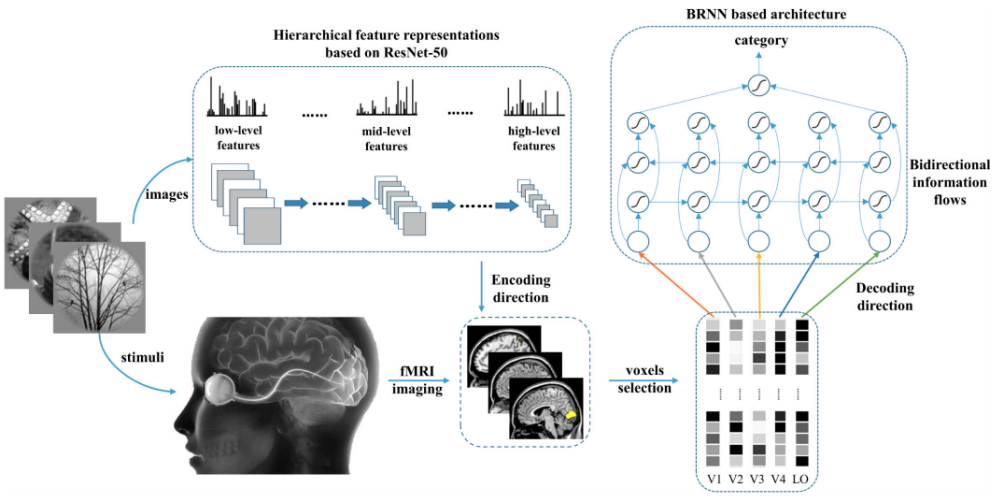


Figure 7: Bidirectional LSTM-based brain decoding. This figure is extracted from [47]

3 The importance of scalability for brain encoding and decoding models

3.1 Increasing the fMRI dataset

Most of the aforementioned brain encoding and decoding models suffer from lack of a comprehensive dataset. The traditional datasets (pairs of fMRI and stimuli) have three limitations: covering limited feature space of vision stimuli, poor spatio-temporal resolution, and low Signal-to-Noise Ratio (SNR). These defects lead to poor modeling of brain mechanism and functions. Recently, researchers in cognitive neuroscience have focused on a collective comprehensive dataset to yield significant progress toward the ideal model. Two main trends of the comprehensive datasets are: 1) increasing the number of the subjects, 2) expanding the stimuli feature space. Fig. 8 shows these two trends where x-axis and y-axis are the number of fMRI scanning hours and the number of the subjects respectively. It is worth to mention that the number of scanning hours indicates the variation of the stimuli and the conditions during the experiments. In Fig. 8, the traditional fMRI datasets are illustrated with gray box where the number of the subjects is almost 20 and the scanning time is almost 1 hour. In contrast, the top right corner of the figure illustrates the comprehensive dataset to achieve the ideal brain encoding and decoding model which does not exist in practice. The ideal model requires the generalization between the subjects and the experimental conditions. However, due to the resource limitations (e.g. scanning time for each subject) datasets are able to grow in one dimension rather than both directions that restrict the generalization power. It means that there is a trade-off between the number of subjects and the scanning hours per subject. In the following we provide more details of fMRI data trends to achieve high accuracy in brain encoding and decoding models.

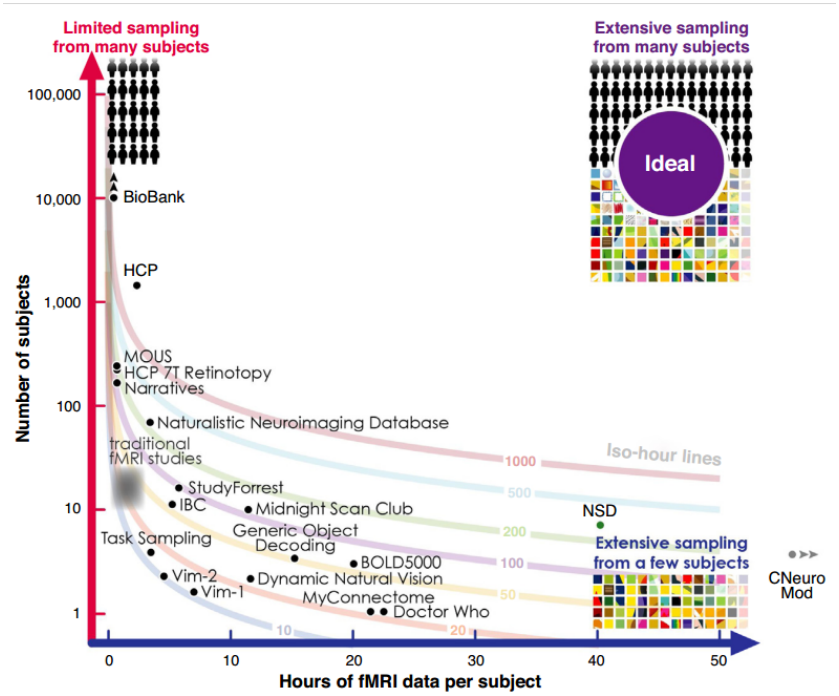


Figure 8: fMRI data sets increases in two trends: number of the subjects and scanning hours [42]

3.1.1 Increasing the number of subjects

Some of the large-scale fMRI datasets provide a large number of subjects while the number of stimuli (conditions) is small. The main idea behind these datasets is increasing the statistical power of brain encoding and decoding models. The ideal brain model must be generalized between different subjects because the general principles of brain mechanism are the same for all subjects. Two outstanding fMRI datasets that have focus on subject generalization are the Biobank dataset [5] and the Human Connectome Project (HCP) [51] in a population of more than 100,000 and 1,200 subjects respectively.

3.1.2 Increasing the fMRI scanning time per subject

Finding the correlation between the huge feature space of natural images and brain activity is complicated. Therefore, understanding the brain hierarchical visual system requires an intensive stimuli set of outdoor and indoor images. For instance, it is possible that the voxels that are correlated with the yellow color were incorrectly correlated with the fruit banana. It is a common issue in the current brain encoding and decoding approaches because of the lack of comprehensive training dataset.
Datasets [11, 6, 10, 1] provide long fMRI scanning time for few subjects. The motivation behind these works is that the ideal brain encoding and decoding models must be trained using extensive stimuli space. More details about these datasets are provided in the following:
BOOld 5000 [11]: In this dataset 4,916 images were presented to the subjects and $\sim$20 hours of fMRI scanning per each of four participants was recorded. These images were extracted from three state of the art computer vision datasets to cover large diversity across image categories: 1) 1,916 images extracted form ImageNet [16], these images contain only one object from specific category such as desk, dog, and etc. 2) 2,000 images from the COCO dataset [31] which each image contains of multiple objects. 3) 1,000 indoor and outdoor scene images from SUN dataset [56] cover 250 categories.
Dr Who [49]: This dataset is a densely sampled large fMRI dataset (TR=700 ms) in a single subject exposed to 30 episodes of BBC's Doctor Who ($\sim$23 h). Dr Who dataset includes of 120,830 whole-brain volumes as a training set and 1,178 volumes as test set.
Natural Scenes Dataset (NSD) [6]: This dataset contains 30–40 hours fMRI data acquisition for each of 8 subjects with high spatial resolution (1.8 mm). The stimuli set includes 73,000 natural scenes images that are extracted from COCO computer vision dataset [31] and stimuli image set was presented 3 times to each subject. In the experiments, for each subject 9,000 unique images and 1,000 shared images between subjects were presented. This sampling strategy provide the opportunity to maximize the number of unique images in NSD dataset. Fig 9 shows that the performance of brain encoding significantly increases by increasing the training samples of NSD data set.
CNeuroMod dataset [1]: The CNeuroMod research group has released the largest individual fMRI dataset to date. CNeuroMod dataset provides a great opportunity to train complex DL-based brain encoding and decoding. In this part, we will review the this dataset focusing on Movie-10 and Friend dataset.
The Movie-10 dataset includes about 10 hours of fMRI data for 6 healthy subjects, 3 men and 3 women. The participants watched the Bourne supremacy, Wolf of wall street, Hidden figures and Life movies with duration times of 100, 170, 120 and 50 minutes respectively. Hidden figures and Life were presented two times. The recorded fMRI data in the first time was considered as a train dataset and the second one was collected as a test dataset.
In the Friends dataset, fMRI data for 40-60 hours per subject was collected while episodes of the

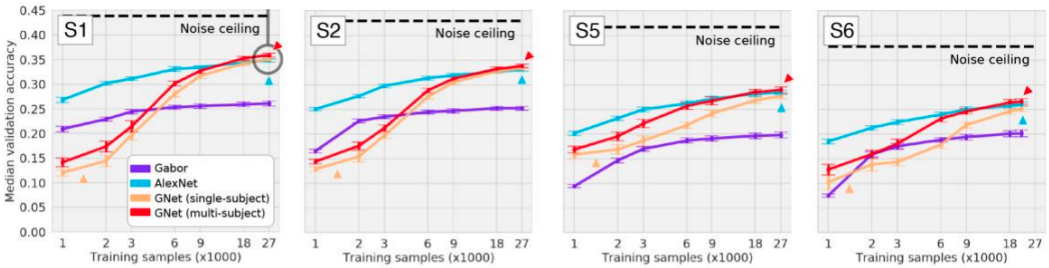


Figure 9: Performance of brain encoding is a function of training data size for different size [6]

Friends TV series were presented to subjects. Each episode of seasons 1 and 2 was cut down in two segments. For both of the Friends and Movie-10 datasets each movie was divided into almost 10 minutes segments then each segment was presented to subjects in a separate run of recording fMRI data. These segments were edited twice: firstly, the end of each video segment and the beginning of the next video segment were overlapped for a couple of seconds; secondly, a black frame was attached at the end of each segment. After the preprocessing step, this intensive fMRI dataset is ready to be used for DL-based brain encoding and decoding.

3.2 Increasing the model complexity

In addition to increasing the size of dataset, there are other techniques to improve the brain encoding and decoding models. These technique increase the computational and memory overhead significantly. Consequently, to apply these techniques the scalability of DL systems is essential. In the following we present three of this approaches:

1) Whole-brain data instead of ROI data: In most neuroscience studies, the voxels in ROIs were used for brain decoding, which neglect fMRI signals out of the ROIs. Missing the significant part of the information in the first step leads to inefficient training. To address this challenge, training the DL model by leveraging whole-brain feature space has attracted more attention. The whole brain provides more discriminant information in brain decoding without needing specific assumptions about the brain activity.

2) Time series instead of the peak signal: Mostly, decoding models were trained using the peak signal of HRF which shows the strongest response to the stimuli. Recently, it was shown that temporal information (signals before and after the peak) contains discriminant information which can be used to improve the model training. In other words, considering the dynamic process of fMRI is essential to improve the decoding performance.

3) Dual modeling: Typically, encoding and decoding models are trained separately which leads to overfitting. Recently, to address this problem, training the encoder and decoder models simultaneously attracted more attention. Dual learning will be opening the door to achieve high performance in cognitive science.

Last but not least, due to large scale computations and memory requirements, big data techniques are essential to train an efficient brain model. In the next section, we will see how big data approaches are able to provide efficient and practical computing frameworks to train large scale brain encoding and decoding models.

4 Parallelism-based approach

In the DL training process, the computations are involved many matrix multiplications that leads to many opportunities for parallelization. In this section, we review the main parallelism approaches to scale up DL models. Furthermore, the strengths and limitations of these methods with respect to DL-based brain encoding and decoding models will be investigated.

4.1 Data parallelism

In data parallelism, each core loads a copy of the DL model and each copy is trained by a part of data. In other words, the intensive data is divided into some non-overlapping blocks to train only one of the copy of models. In this parallel approach, each core communicates with other cores in order to update the model parameters. During the DL training process, two steps are considered to synchronize the parameters. Firstly, the gradient of the loss function is computed for each core. Then the average of gradients is computed via communication between the cores to update the parameters. However, to do so we should answer two questions.

4.1.1 Which strategy should be selected to synchronize?

The parameters can be synchronized through two main strategies: centralized and decentralized. For models with large-scale of parameters, a central server can be applied to prevent direct communications between cores during the parameters updating process. While in the decentralized approaches, there is no central server and the cores communicate with each other directly to update the parameters. A fully connected communication between the cores leads to the bottleneck due to the large numbers of the communication cost. Applying the ring-allreduce strategy reduces the overhead significantly [34]. In this approach each core have communication only with two core. It mean that each core sends the data to left neighbor, and receives data from the right neighbor.
One of the challenges in the communication between cores is that the number of parameters in the different layers are different. For instance, the size of the gradient tensor in the Convolution layers is less than FC layers. Consequently, updating the parameters with small size of gradient tensors leads to overhead and latency in the communications. In [27], the authors proposed an approach to address this problem by collecting the small gradient tensors together. After that when the size of the collection is greater than a specific threshold then the communication is occurred (sending the collection). This approach by increasing the bandwidth utilization decreases the overhead.

4.1.2 What is the best schedule to update the parameters?

The parameters can be updated after each batch. In this case, the batch size plays an important role in the amount of communication overhead. In [29] authors showed that the large batch size leads to instability in the DL models training. On the other hand, if the batch size is too small then the cores could not be completely utilized (Fig.10). Consequently, considering an efficient updating schedule to trade off between speed up and convergence is essential. In [29], the authors proposed an updating schedule to select the batch size freely. In the proposed approach, a core computes the gradients; then the parameters are updated locally (without communication with other cores). The core continues the forward and backward pass for the next batch. To avoid diverging, the parameters are updated based on gradient averaging of other cores (Fig.11).
In [32] the authors proposed a new scalable Autoencoder model to analyze massive fMRI dataset. In this work, the DL model is trained to extract hierarchical abstraction of HCP dataset. This DL

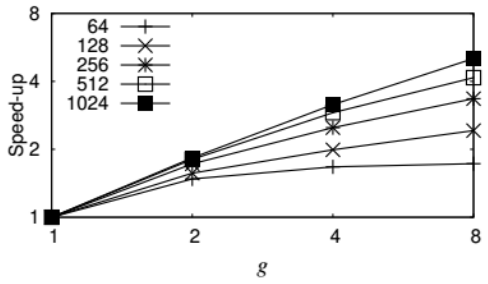


Figure 10: The effect of batch size on hardware efficiency. This figure was extracted from [29]

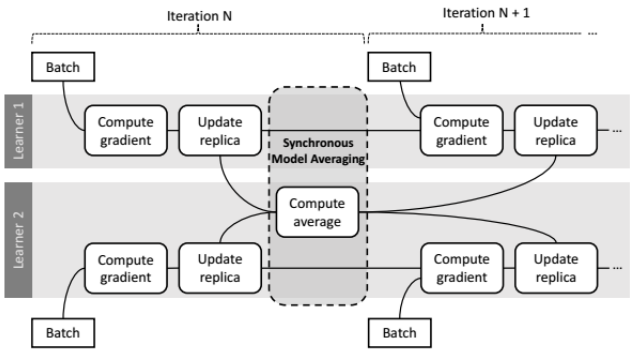


Figure 11: Parallel training with two cores.This figure was extracted from [29]

model is trained in a distributed infrastructure by applying data parallelism strategy. To do so, they applied a combination of Apache Spark and Tensorflow on a large cluster of GPUs. In this work, Hadoop was used as distributed file systems and it allowed Spark to be involved with data partitioning and task scheduling. Spark scheduler aims to avoid reloading the same data which is essential for complex DL systems. Furthermore, Spark allows the direct tensor communication between cores in the data parallelism. In this study, the researchers used both direct communication and parameter server to share the local gradients.

Fig.12 shows the acceleration of the training process of the Autoencoder model relative to the

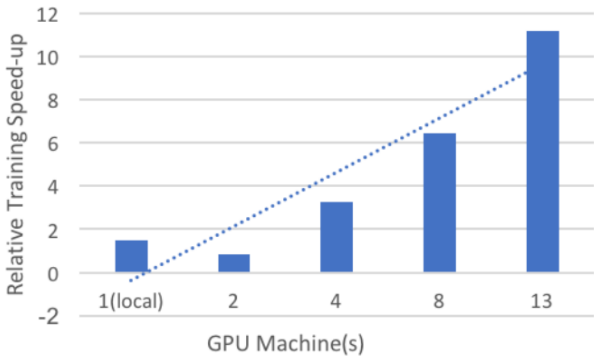


Figure 12: Acceleration of fMRI DL model training using Saprk and Hadop. This figure was extracted from [32]

number of GPUs. Training with one GPU machine is faster than two GPU machines since the communication overhead in one GPU machine is significantly less than two GPU machine. By adding more GPUs the training time significantly speeds up. For instance, the training time with 13 GPUs

is seven times faster than one GPU. Experiments showed that by increasing the number of GPUs more than 13 the performance started to decrease because of the communication overhead.

4.2 Model parallelism

In model parallelism, the DL model is divided into several parts then each part is trained on the specific core. The training data is presented to the core which has the input layer. In the forward pass, each core computes the activation of the assigned layer(s). This result is considered as the input of the next core which holds the next layer. In contrast, the backpropagation starts with the core which holds the output layer. The gradients are computed in each core and the results are propagated toward the core which holds the input layer [34].

Partitioning a DL model is a time-consuming task because it should be done in a case-by case manner. In [38, 39], the authors proposed a fitness - based approach for partitioning the DL model. This approach starts with computing the fitness score for a random partition. In the next step, the fitness score is evaluated for permutations of the initial partition. If the fitness score increased, the permutation is kept. This process continues until the fitness score starts to converge.

In the training of the too large model on the GPU, the part of memory which is involved with the program is called memory footprint and the part of the memory which is involved with the parameters is named memory load. In the model parallelism the memory footprint is reduced significantly which is very beneficial for GPUs or TPUs [59].

In this step, there is a challenging question: to scale-up a brain encoding and decoding system, which of the parallelism technique (model or data) should be selected for more efficient training process? In the following the model and data parallelism techniques will be compared.

The main advantage of the data parallelism is that we can apply this approach to models without providing details about the model architecture. However, to apply the model parallelism it is essential to provide an efficient partitioning model. This is a crucial step because without efficient partitioning, the cores could be in the stall mode due to the long delay in the communications. Therefore, without efficient partitioning, even by increasing the parallelism degree (number of cores) the training process will not be accelerated.

Both model parallelism and data parallelism need communication between cores. In the case of

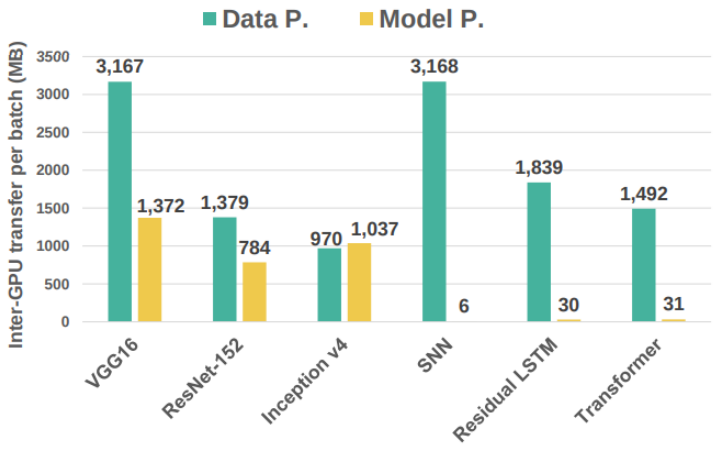


Figure 13: Comparison of the communication overhead between model parallelism and data parallelism. This figure was extracted from [35]

data parallelism, the cores share the local gradients with each other to update the weights. In the model parallelism, cores communicate the temporal results in the forward and backward passes.

Fig.13 shows the comparison of the communication cost between these two approaches. In the Spiking neural network (SNN), residual long short term memory (LSTM), and Transformer models there are negligible model parallelism communications between the GPUs. Because the less intermediate data between layers leads to low communication cost. While the huge number of parameters increases the cost of the parameter synchronization in data parallelism. On average, the inter-GPU transfer per batch (MB) in the data model is 13 times greater than model parallelism communications.

Based on this discussion, there are limitations and strengths of both parallelism techniques. However, we can benefit from both techniques by combination of these techniques which is known as a Pipeline technique.

4.3 Pipeline parallelism

Pipeline parallelism is a combination of data parallelism and model parallelism. In this technique, both data and model are divided between cores. Each core trains a specific part of the DL model by using a non-overlapping part of the dataset. In comparison with data parallelism and model parallelism, this approach is able to increase the scalability noticeably. It compensates for the bottlenecks of data parallelism since a single core is in charge of holding a part of the model instead of the complete model. Also, this technique reduces the communication delays between nodes by benefiting from multi- batches in the training process [34]. Training different parts of the model with multi-batches leads to increasing the utilization of cores significantly. For instance, Fig.14 shows a pipeline parallelism to train a DL model with multiple- batches across four cores. $F_{n,k}$ and $B_{n,k}$ represent the computations of kth batch size on nth core in the forward and backward pass respectively. It is worth mentioning that, in this process, $B_{k,n}$ is dependent on both $F_{k,n}$ and $B_{K,n+1}$ [26].

GPipe [26], is a scalable pipeline parallelism library for training DL models. Two important prop-

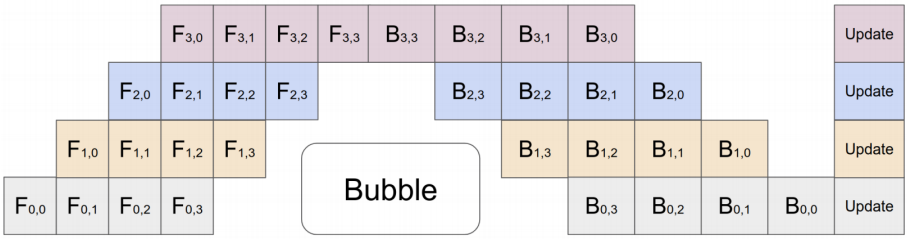


Figure 14: Forward and backward steps in the pipeline parallelism for four cores. This figure was extracted from [26]

erties of GPipe are efficiency and reliability. In terms of efficiency, the DL training process speeds up linearly with the number of cores. Furthermore, GPipe provides consistent DL training without considering the number of partitions.

The GPipe is evaluated on two DL models, AmoebaNet-D (L, D) and Transformer-L. The AmoebaNet model has L normal cell layers with filter size D and the Transformer-L model has L encoder and decoder layers, with 8192 hidden dimensions. Table.1 presents the maximum size of these models which are supported by GPipe under different scenarios. For these results, it was assumed that each model parameter requires 12 bytes. The experiments showed that when the number of the data micro-batches is at least four times more than the number of model partitions, the delay of the communication is almost negligible. For the Transformer model, the scaling up is linear in terms of the number of cores.This scalability comes from the balanced computation through the cores. However, AmoebaNet model has less scalability because in the distributed training some

Table 1: Maximum size of AmoebaNet and Transformer-L models in different scenarios [26]

NVIDIA GPUs (8GB each)	Naive-1	Pipeline-1	Pipeline-2	Pipeline-4	Pipeline-8
AmoebaNet-D (L, D)	(18, 208)	(18, 416)	(18, 544)	(36, 544)	(72, 512)
Number of Model Parameters	1.05GB	3.8GB	6.45GB	12.53GB	24.62GB
Peak Activation Memory	6.26GB	3.46GB	8.11GB	15.21GB	26.24GB
Cloud TPUv3 (16GB each)	Naive-1	Pipeline-1	Pipeline-8	Pipeline-32	Pipeline-128
Transformer-L	3	13	103	415	1663
Number of Model Parameters	282.2M	785.8M	5.3B	21.0B	83.9B
Total Model Parameter Memory	11.7G	8.8G	59.5G	235.1G	937.9G
Peak Activation Memory	3.15G	6.4G	50.9G	199.9G	796.1G

of the cores involved more computations than others, which leads to delay in the communications (bubble overhead).

4.4 Mixture of experts

Mixture of experts (MoE) is one of the machine learning approaches which divide a problem space into intermediate problems in terms of complexity. In other words, it is beneficial to use multi models with intermediate size instead of using a single complex model. The reason is that, during the training each model is specialized in a sub problem. Specifically, for complex DL models, MoE plays an important role to reduce the memory. Because instead of extracting the same features for all of the training samples, only the specific features are extracted. Hence, in [17] the authors proposed a sparsely-activated technique to reduce the communication overhead. Also, the proposed method improved the training process in terms of stability and fine-tuning.

In Fig.15- right, each partition represents one core. In the standard data parallelism training system

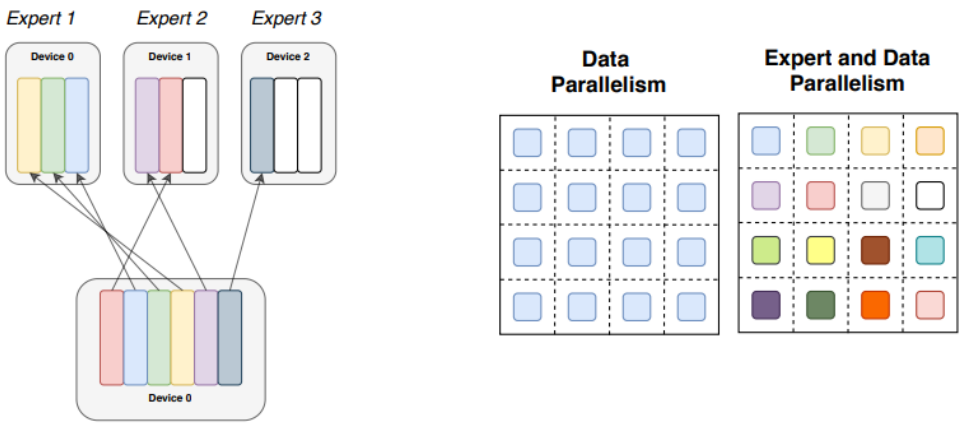


Figure 15: Sparse activating in the MOE parallelism system. This figure was extracted from [17]

all the cores have the same weight matrix while in the MoE systems each core has unique parameters. Different colors represent different weight matrices. In Fig.15- left, shows the schematic diagram of the sparsely-activated technique in MOE systems. Assuming each core holds an expert model, the communication overhard is reduced because the cores need to synchronize the common parameters instead of all the parameters.

5 Memory-based scaling up

In the forward pass the activations of layers are computed and then stored because in the backpropagation step these values are used to update the parameters. For instance, training ResNet50 on the ImageNet dataset needs about 40GB of memory [22]. One modification of this model, GPIPE model, increased the baseline accuracy 10% by consuming 4.6 time more memory [26]. One of the factors that has main effects on scalability is storing these large amounts of temporary data in the training process. To reduce the temporal data overhead in this chapter we review some memory management techniques.

5.1 Mixed precision

Data parallelism as a state of the art approach is applied in training large-scale DI systems. But there are two main challenges. Firstly, although the larger batch size increases the platform utilization, it may decrease the accuracy on the test data set. For instance, in [57] authors trained ResNet-50 with different batch sizes. The results show that when the batch size is 64K, the test accuracy decreases to 73.2% where the baseline accuracy is 75.4%. The reason is that larger batch size increases the generalization by decreasing the gradient variance. In other words, updating the parameters based on gradient averages leads to bigger step size in the training process [27]. The second challenges is that increasing the number of processors increases the communication overhead. To address these challenges, in [37] a mixed precision based approach was proposed to scale up DL systems. The proposed approach trained the ResNet-50 without any loss accuracy (75.8%) in 6.6 minutes where the batch size was 64K.

The main idea behind this approach is using 16-bit numerical formats instead of full precision float points (32 bit). Three advantages of this method are: reducing the memory usage, reducing the bandwidth, and accelerating the math computations. In the training process, parameters, activations and gradients values are stored in the memory with FP16 format. Then, all the computations in the forward and backward pass are performed in FP16 format. To avoid the trade-off between speed and accuracy it is necessary to maintain a copy of parameters in FP32 format. Consequently, in the end of each iteration parameters are update based on FP32 format. NVIDIA research group in [46] reported that training a large scale natural language model (RNN-NLP) using mixed precision approach is 4.2 times faster than FP32 on a same platform. Also Facebook research group in [44] showed that RNN model can be converge 5 times faster than baseline.

Representing the gradient values in FP16 has a main challenge which is called Gradient Underflow. Due to gradients values are too small (negative exponents), they are represented with 0 value in FP16 format while a large range of FP16 remain unused. In [37] authors proposed a solution to address this problem by shifting the gradient values based on specific scaling factor. The scaling factor can be selected among 8, 32, 64, 128 values or it can be tuned automatically during the training process. In the automatically approach, the scale factor is initialized with a large number. Then in the training process if the gradient values are Inf or NaN the scaling factor is decreased other wise after a while the scaling factor value is increased (Alg.1).

Algorithm 1: Mixed Precision Algorithm [37]

```
Result: Training Parameters
Initialize S to a large value;
while model convergence do
    Make an FP16 copy of the weights;
    Forward Propagation ;
    Multiply the loss values with the scaling factor S;
    Backward propagation;
    if there is Inf or NaN in the gradient values then
        Reduce S;
        Skip the weight update and go to the next iteration;
    else
        Multiply the weights with 1/S;
        Complete the weight update (FP32);
        if there was not any Inf or NaN in the last N iterations then
            Increase S;
        else
        end
    end
end
```

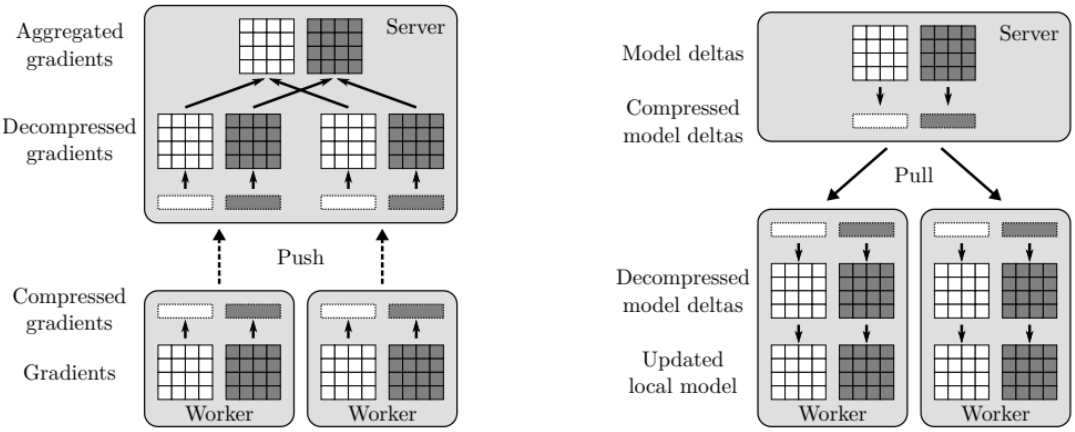
5.2 Data compression

Memory and bandwidth bottlenecks are main challenges to scale up DL models. One solution can be applying lossy compression technique to reduce the memory storage without any change in the accuracy.

In [30], the authors developed a tensor compression approach which includes three main steps: the compresses, transmits, and decompresses. In the first step, each core is computing the local gradient and pushing the compressed gradient tensor to the server. In the server, the tensors are decompressed and after aggregating the gradient values, the compressed results are sent to the cores again. Finally, after decompressing the data, the cores update the local models. Fig.16 demonstrates these steps.

In this work authors presented 3LC lossy compression which in this type of compression algorithms the large data is compressed with losing some data however this lost data is not noticable. 3LC approach benefits from the strengths of 3-value quantization, quartic encoding, zero-run encoding techniques.

In this study, the compression process contains four steps: firstly, the gradient tensor and a local buffer were added. In the second step 3-value quantization is applied to the sum in a way that each floating-point number in the gradient tensor is represented as three values -1, 0, 1. In this step to control the compression level, the quantization error is calculated by subtracting the quantized values from accumulation. Based on this remaining the local buffer is updated. In the third step quartic encoding algorithm is applied to compress five 3-values matrix into a single byte. In the last step, to remove the replicated values, zero-run encoding techniques is applied. Fig.17 demonstrates the gradient tensor compression steps.



(a) Gradient pushes from workers to servers. **(b) Model pulls from servers to workers.**

Figure 16: Tensor compression in deep learning. This figure was extracted from [30]

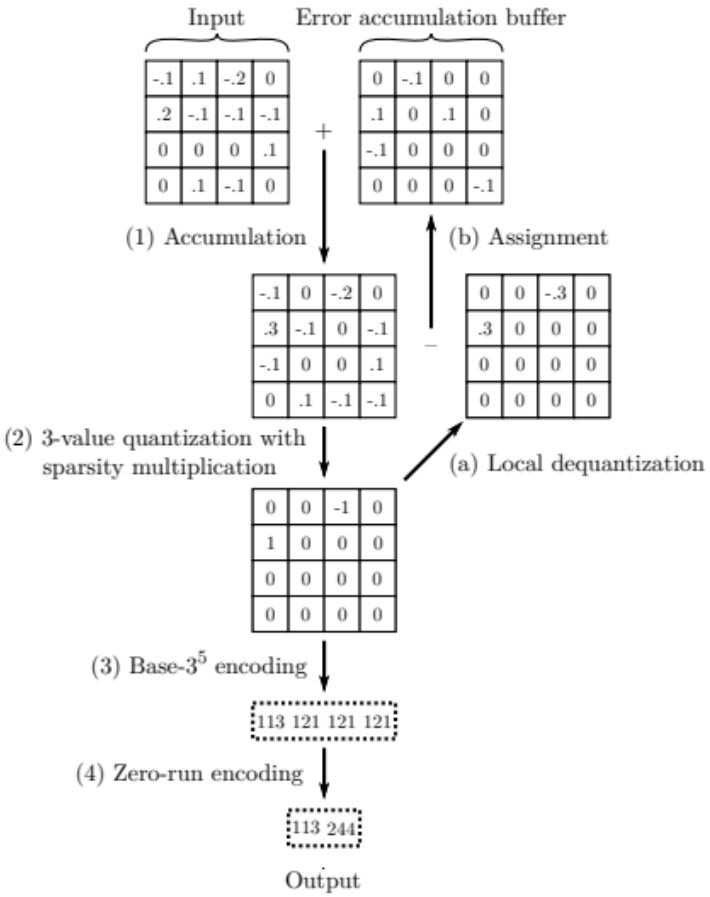


Figure 17: The gradient tensor compression steps. This figure was extracted from [30]

6 Infrastructure for Deep Learning

6.1 Comparison of TPU, CPU and GPU

Parallel computing plays an important role during the DL training process. Multi-core processors allow parallel computing; therefore the graphics processing units (GPUs) typically are used for DL training tasks. In the case of the intensive dataset, there are two main bottlenecks for GPUs: bandwidth and capacity limitation of memory. To address these problems, DL models can be scaled up on high technology multi-core central processing units (CPUs) or distributed infrastructures [34].
One of the state of the art technologies is Intel® Xeon® Scalable processors [3]. Two key properties of this technology are large core count and large mid-level caches for each core. These properties provide high computational and memory capacity. In this technology the basic DL operations such as convolution and batch normalization are parallelized over the cores. In contrast, with other technologies, which fetch the same data multiple times this advanced technology reuses the data in the cache. Furthermore, each core is involved in an equal number of computations since this multi-core CPU technology enhanced the load balance.
Training complex DL models with intensive data can be trained on a distributed infrastructure. Distributed platform includes multiple nodes and each node includes several GPUs. Different companies such as Facebook, Google, and Microsoft build very large DL systems on the specialized distributed infrastructure. Facebook utilizes both CPU-based and GPU-based resources to serve their specific tasks of training and inference [21]. Microsoft performs large scale DL on a massive number of CPU-servers [15]. In the proposed infrastructure, the DL model was divided into layers such that each layer can fit in the L3 cache. In this design, training efficiency improved significantly. Google employed an infrastructure based on multiple communication protocols and Tensor processing units (TPUs) [4]. TPUs are new technology that are proposed by Google to accelerate DL training and inference. Comparison of TPU, CPU and GPU are presented in Tables.2 and Fig.18.

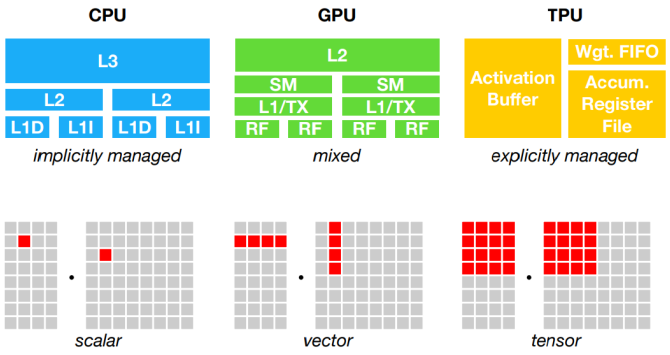


Figure 18: First row: Memory subsystems Architecture, Second row: Compute primitive. This figure was extracted from [13]

6.2 Benchmarking TPU, GPU and CPU platforms

In [8], the authors presented the benchmarking results on Google's Cloud TPU v2/v3, NVIDIA's V100 GPU, and an Intel Skylake CPU for ParaDnn model. ParaDnn is a sequential model which includes Fully Connected (FC), Convolutional Neural networks (CNN), and Recurrent Neural Networks (RNN).
In addition to the data size and the batch size, also the model hyperparameters are considered in

Table 2: Comparison of TPU, CPU and GPU [13]

Processing Unit	Hierarchical Memory	Compute Primitive (data unit)	Performance (per cycle)	Purpose	Manufacturers
CPU	Three general cache levels: L1, L2 and L3	1X1	Tens of operation	Solve every computational problem in a general	Intel
GPU	Mixed architecture: L2 cache, Shared Memory, L1 cash, and Register File	1 X N	Tens of thousands of operation	1)Accelerate the rendering of graphics 2)Training ML	NVIDIA
TPU	Explicitly managed: Based on registered account file	N X N	Up to 128000 operations	Training and inference ML models	Google

Table 3: The ranges of the variables in the benchmark process [8]

Model type	Variable	Min	Max	Inc
Fully Connected Models	Layer	4	128	×2
	Units	32	8192	×2
	Input	2000	8000	+2000
	Output	200	1000	+200
	Batch Size	64	16384	×2
Conv.Neural Nets	Block	1	8	+1
	Filter	16	32	64
	Image	200	300	+50
	Output	500	1500	+500
	Batch Size	64	1024	×2
Recurrent Neural Networks	Layer	1	13	+4
	Embed Length	100	900	+400
	Length	10	90	+40
	Vocab	2	1024	×4
	Batch Size	16	1024	×4

the benchmarking experiments as well. The FC model's hyperparameters are: the number of the layers, the number of units in each layer. The hyperparameters of the CNN model are: the number of the filters and the number of blocks, where blocks contain convolutional layers and batch normalization layers. The RNN model contains RNN, LSTM, or GRU layers. In this model the number of the layers and the embedding size (length of input vector) are considered as the hyperparameters. Table 3 presents the range of these hyperparameters for these DL models.

The number of Floating point operations per second (FLOPS) evaluates the computation efficiency in different scenarios. The effect of CNN, RNN and FC parameters on TPU utilization is shown in the heatmap (Fig.19). In the benchmarking, two of the model variables are selected while the other variables were kept fixed. The TPU utilization was improved by increasing the batch size, the number of filters in CNN, the number of units in FC and embedding size in RNN.

Another element which has a noticeable effect on the training performance is communication overhead between the cores. Fig.20 demonstrates the communication overhead where x-axis and y-axis are FLOPS values for one-core TPU and eight-core TPU respectively. In the ideal scenario (without any overhead), the points should be placed in the y=x line. In the case of FC, when the batch size is less than 1024 the performance of 1-core GPU is more than 8-core TPU. Even though, by increasing the batch size the scalability was increased, the batch size should be around 16k (at least) to achieve 50% utilization. It is worth mentioning that the communication overhead of CNN is less than FC because the number of the training parameters (the communication to share the local gradients) in the CNN model is less than FC.

The number of samples per second is another factor to evaluate the performance of large-scale

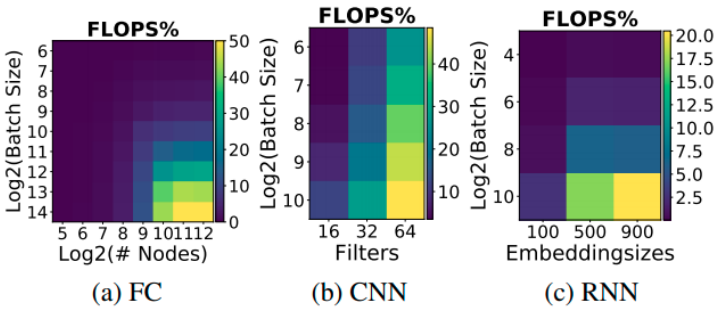


Figure 19: The effect of hyperparameters on the TPU utilization [8]

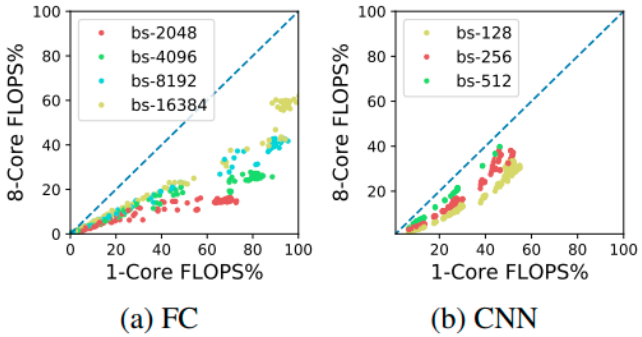


Figure 20: Correlation between scalability of the DI models and communication overhead. This figure was extracted from [8]

DL systems. Fig.21 shows the effects of batch size and the number of units in the Fully Connected network on the TPU, GPU and CPU utilization. In these experiments, the number of layers is fixed. Only in the case of the TPU and GPU increasing the batch size improves the utilization. The reason is that these platforms provide parallel computing better than CPUs. However, due to memory capacity only the CPU is able to perform the training process when the number of the units is greater than 12.

Recently, accelerator design has made significant progress to improve the scalability of DL sys-

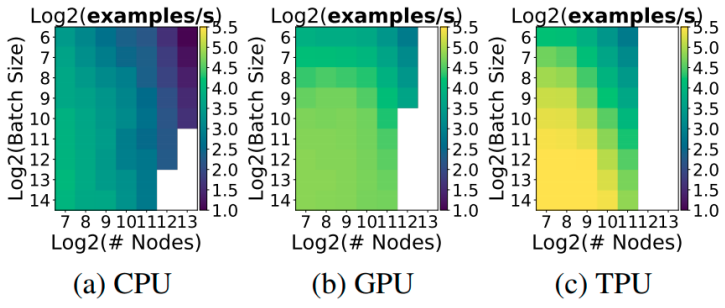


Figure 21: The effect of batch size and number of units on platforms utilization. This figure was extracted from [8]

tems. However, the data shuffling and reshaping during the training process which affect on the performance have received less attention. For instance, in [55], the authors break down the time of training a DL system into compute, data transfer, software stack time. The benchmarking results show that 24%, 34% and 42% training time was spent on compute, data transfer, software stack time respectively. Obviously software stack time is a bottleneck of the large scale DL systems,

specially, in the depth CNN networks (e.g. ResNet-50) due to tiling operations. Each tile has three parts: a compute unit, caches and a switch. The data shuffling and reshaping during the training process causes switching between cores (each core contains compute and caches units) to be time consuming.

6.3 Intel® Optane™ DC persistent memory

Intel® Optane memory is a super fast memory module that acts as a bridge between RAM and flash-based storage. This technology keeps the most frequently used data near to the processor. The main advantage of Optane memory over DRAM is that Optane memory keeps the data when the system is turned off.

In [20] authors presented the advantages of the Intel® Optane™ DC persistent memory [2] for the parallel computing of the intensive neuroimaging. The evolution was performed in two cases: Memory mode and App Direct mode. In the first case, Optane was considered as an extension of volatile main memory which needs power to maintain the stored data. The main advantage of Memory mode is that accessing and reading the large data is fast. In App Direct mode, Optane was used as a non-volatile memory storage device which keeps the stored data even after the power is off. In this case, the Operating System (OS) can distinguish the DRAM from Optane and consider them as two different levels of memory.

For the experiments, a standard neuroimaging application (adding one unit to each voxel value) was applied on 76 GiB and 603 GiB BigBrain [7]. The main property of this application is that the CPU time can be ignored compared to the I/O time. Furthermore, GNU Parallel and Apache Spark were applied to parallelize the application. The experiments were performed using 25 and 96 parallel processes. In the following, we will review the results briefly.

Fig.22 A and C show the parallel incrementation application processing on BigBrain using 25 processes and 96 processes respectively. These results are the average of three time repetitions on 4 storage technologies: DRAM, Optane, Local Disk, and Isilon. Furthermore, the anticipated read and write duration values of these four storage devices are obtained based on bandwidth (MB/s) and these values are compared for two cases: Memory mode, and App Direct mode. Fig.22.A shows DRAM has the best results in terms of makespan. Optane has the second rank with slight difference. Optane in the Memory mode was faster than App Direct mode due to utilizing DRAM as a cache for writing. Isilon was the slowest storage device with a very significant difference compared to the other. It is worth mentioning that Optane in Memory mode was significantly faster than expected.

Fig.22 B and D show the read, increment and write breakdown results for different storage devices. Clearly, the execution time for these tasks are significantly different. The efficiency of Optane in the Memory mode is slightly less than DRAM, while it is noticeably higher than other Local SSD and Isilon.

Fig.22 shows that, unexpectedly, increasing the number of the processes from 25 to 96 did not improve the performance even made the results worse.

In the training process of DL-based brain models, transferring intensive fMRI dataset into a distributed infrastructure is one of the main challenges. Intel® Optane™ DC persistent memory can be considered to address this challenge as it is efficient in terms of speed, capacity and cost. Based on the aforementioned paper, Optane (specifically, in the memory mode) is able to accelerate the data movement due to using DRAM as a cache for writing. In other words, storing data close to compute resources enhances scaling (increasing core utilization).

However it should be noted that choosing the number of the processors is really crucial as the large

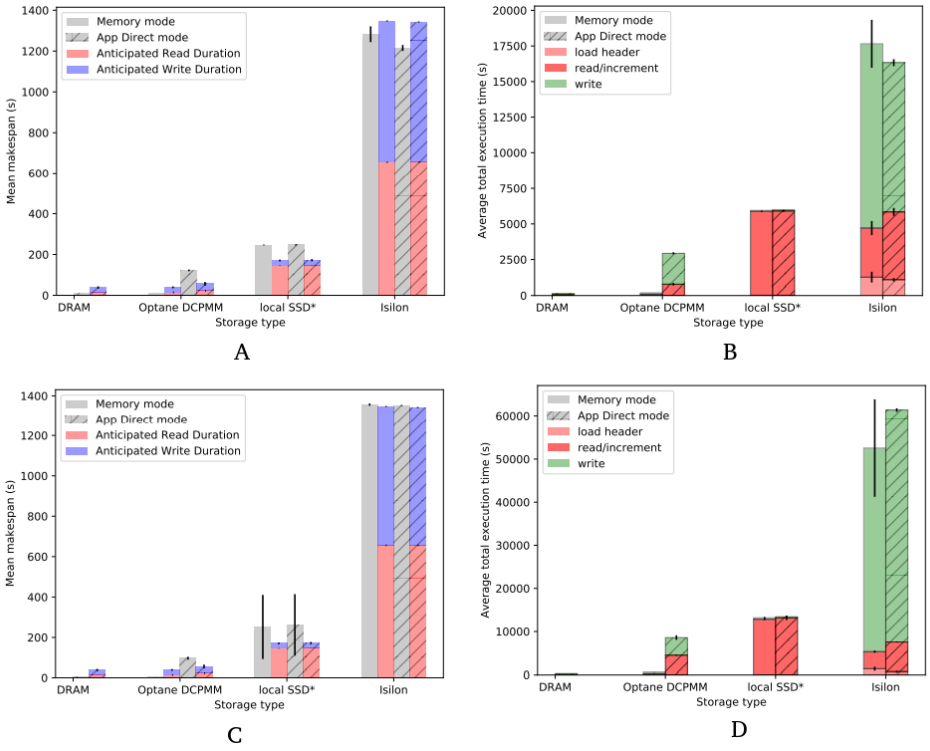


Figure 22: The parallel incrementation application processing on BigBrain using A: 25 processes, C: 96 processes. The breakdown results for different storage devices B: 25 processes, D: 96 processes. This figure was extracted from [20]

number of the processors may increase the training time due to the increasing I/O time. Therefore to use this technology firstly, the optimum number of the processors should be selected.

7 Application of big data approaches for fMRI brain models

7.1 Parallelism approaches for fMRI brain models

The performance of different parallelism approaches depends on their ability to avoid bottlenecks. Therefore, based on DL system architecture and characteristics the parallelism strategy must be selected. For instance model parallelism is more flexible for RNN-based models since the huge number of parameters increases the cost of the parameter synchronization in data parallelism. For CNN-based models applying data parallelism is more efficient due to compute complexity and few number of parameters.

Pipeline and Hybrid parallelism benefit from the strengths of both Model and Data parallelism. Pipeline parallelism combines the data parallelism and model parallelism; it means that both data and model are divided between cores. While the hybrid method designs manually in a way that for convolution and pooling layers, data parallelism is applied and for FC layers model parallelism is applied.

In the following, we discuss how the brain encoding and decoding models can be scale-up by leveraging the parallelism approaches.

In [50], the authors proposed a large image reconstruction model with pairs of $\{$fMRI, stimulus images$\}$ based on generative adversarial network (GAN). The fMRI data contains voxels from the VC region. The proposed model contains three DL models: 1) a generator neural network, 2) a comparator neural network, and 3) a discriminator neural network (Fig. 6). In this study, the experiment was limited to only 6000 images and results showed that by increasing the number of images the accuracy increased significantly (Fig.23).

To increase the accuracy it is necessary to increase the data size, but there is a challenge here;

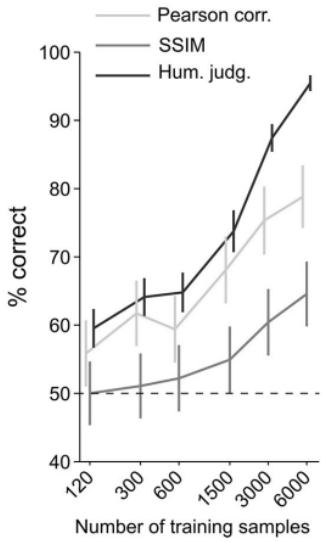


Figure 23: For the image reconstruction with increasing the number of fMRI data the accuracy increases noticeably [50]

how to manage the complex model (huge number of parameters) in terms of memory requirements. In this case, pipelines parallelism could be a potential option to train three sub-models efficiently in the distributed infrastructure. The aim is leveraging both model and data parallelism benefits.

1)Model parallelism: The image reconstruction architecture in [50] contains three sub-models: generator, comparator and discriminator. This implicit property of model provides reasonable model partitioning. Consequently, computing cores can be divided into three subsets respect to each

sub-model. In this case the model parallelism leads to the balance computing in the distributed infrastructure which is crucial step in the pipeline technique.
2)Data Parallelism: The sub-models are CNN-based models therefore a few number of parameters and low level computing complexity make it possible, to apply data parallelism for training each of the sub-models.
However, by applying these models the communication overhead issue comes up. In this case, to reduce the communication overhead, we can apply the hierarchical All-reduce approach [27]. In the first level of communication, the computing cores in each subgroup (cores that are involved with training the generator, comparator or discriminator) are communicated together. Then the gradient result of each group is stored in a master core (here we will have totally three master cores). In the next level of communication, the master cores share the gradient tensors with each other. In the final step, master core of each group propagates the final result to the other cores of group to update the parameters. Fig. 24 demonstrates the communication process to manage the locally and globally shared model parameters. Three cluster of machines can be utilized to reduce I/O latency with synchronous updates. In the proposed distributed training infrastructure, TPUs can be the best hardware to support the data parallelism on CNN models inside the clusters. The reason is that compare to CPU and GPU, TPU has higher computation efficiency (number of Floating point operations per second (FLOPS)). In other words, in the large scale of the data and computation, TPU utilization improves by increasing the batch size, the number of filters and the number of units (see more details in section 6.2).

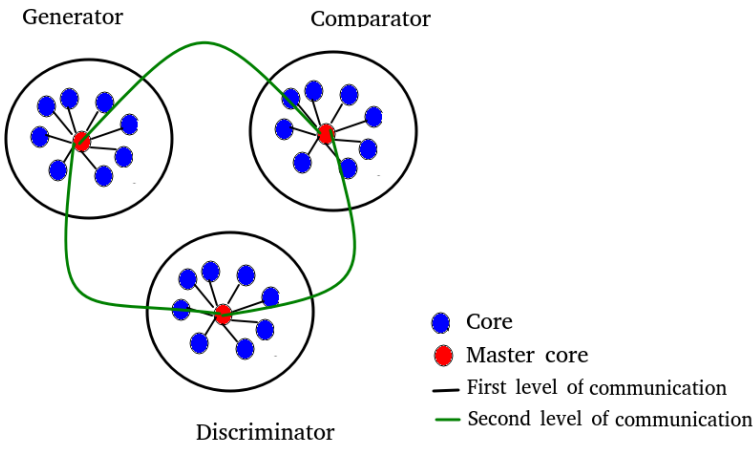


Figure 24: Hierarchical communication between cores

To achieve high performance of scalability, the number of cores in each group can be considered as a tunable parameter.
In the following, we present another study that has the potential to scaling up through model parallelism. In [23], the authors trained a decoder model to map the VGG16 model on the hierarchical brain activity. The decoder consisted of a set of independent linear regression models to predict the activation of VGG16 units. Table 4 shows the data size of different layers of VGG16.
With respect to aforementioned study, we raise a question: how can we decrease the training time of proposed model on a large scale dataset? We suggest a model parallelism approach because it provides an efficient distributed training through multi cores. In this study, as we have a set of individual regression models, the decoder model can be broken efficiently. The model partitioning is very crucial to reduce the communication overhead in the model parallelism.
In the following, we present two other brain encoding task that has potential to scale up using MOE

Table 4: Decodability dataset [23]

Layer	Data size
$conv1_1$	224 × 224 × 64
$conv1_2$	224 × 224 × 64
$conv2_1$	112 × 112 × 128
$conv2_2$	112 × 112 × 128
$conv3_1$	56 × 56 × 256
$conv3_2$	56 × 56 × 256
$conv3_3$	56 × 56 × 256
$conv3_4$	56 × 56 × 256
$conv4_1$	28 × 28 × 512
$conv4_2$	28 × 28 × 512
$conv4_3$	28 × 28 × 512
$conv4_4$	28 × 28 × 512
$conv5_1$	14 × 14 × 512
$conv5_2$	14 × 14 × 512
$conv5_3$	14 × 14 × 512
$conv5_4$	14 × 14 × 512
fc6	1 × 1 × 4096
fc7	1 × 1 × 4096
fc8	1 × 1 × 1000

approach:

1) Recently, authors in [43] applied the mixture of voxel-wise regression model to brain encoding. In this study, the words (stimuli) were represented as 25-dimensional feature vectors to predict the brain response. In this training system, the whole brain was considered as a problem space which was divided into some sub-problems. The sub-problems were ROIs which were encoded with a group of regression model experts. The brain encoding results were compared with two other models, Multilayer Perceptron and Ridge regression. The results showed that MoE has greater R^2 rather than two other models.

Here, there is a question about the aforementioned paper: how can we reduce the training time in a distributed system? We can answer this question by taking the advantage of the MoE approach, in which the experts can be trained independently on the different cores. It is worth to mention that, in this study, there is no communication overhead between the cores because the experts are trained with the different features of fMRI data (ROIs).

2) In another study [25], authors proposed the MoDEN (Mixture of Deep Expert Network) approach to classify the fMRI data. In this approach, an encoder and a discriminator were trained at the same time. An encoder predicts the brain activity for specific tasks; and a discriminator predicts the task label. This unified approach was applied on three HCP tasks: working memory tasks, language and social. As Fig25 shows the encoding problem space is divided between three experts in a way that each of the experts is trained for a specific task. To scale up this model, applying MoE parallelism approach is more efficient because each expert (autoencoder deep model) needs unique weight matrices. This sparsely-activated technique is able to reduce the communication overhead significantly.

The new research direction of cognitive neuroscience is moving toward complex cognitive processes instead of individual elementary cognitive tasks (visual object classification, working memory or auditory tasks etc). For instance, video game dataset [10] have been designed to engage subjects with multi cognitive tasks, in parallel. Consequently, the MOE system is essential to scale up complex cognitive processes by leveraging deep learning approaches and large-scale neuroimaging datasets.

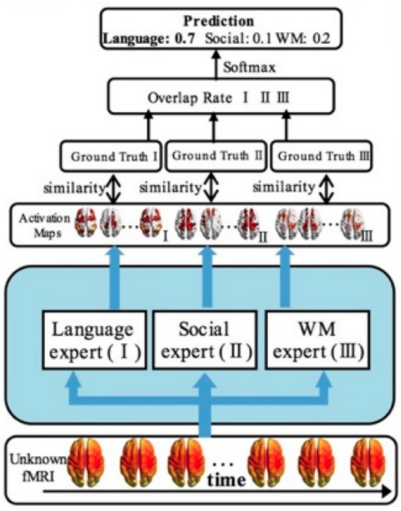


Figure 25: Mixture of three deep expert networks (MoDEN) for brain decoding model. This figure was extracted from [25]

7.2 Memory- based scaling up for fMRI brain models

We reviewed two memory based approaches: compression based approaches and mixed precision. In the following we discuss efficiency and drawbacks of these two approaches with respect to the scaling up fMRI brain models.

Most of the compression based approaches provide few memory consumption reduction (traffic reduction) while sacrificing accuracy. Some of the reasons for low performance of memory based approaches can be attributed to: low rank matrix approximations, manual setup requirements, no theoretical and mathematical support, time consuming to leverage, extra expensive computational overhead [14]. Furthermore, these approaches suffer from implicit properties of the weight matrix, because many elements of the weight matrix are close to zero [14]. On the other hand, the efficiency of compression based approaches are highly dependent on the architecture and model size. Here we provide two examples to make it more clear. Example1: applying compression based techniques in end-to-end deep models leads to more instability of the results in comparison to models which use pre-trained models. Example2: compression based scaling up is not suitable for the large CNNs such as GoogleNet since the accuracy decreases significantly [14]. Consequently, compression based approaches are more beneficial to apply on the small size of the fMRI data models.

Mixed precision generally showed reasonable memory reduction (almost half) without any change in the accuracy. Mixed precision is highly trust-able since this approach was tested on a noticeable number of projects with different model size, model architecture, data type, data size, etc [36, 41]. Therefor, based on these results, mixed precision can be high potential technique to scale up brain encoding and decoding DL systems with large size of data and model complexity.

7.3 Selecting efficient platform for fMRI brain models

Scalability restrictions lead to developing special-purpose distributed infrastructures. Because each CPU, GPU and TPU offer advantages based on DI system architecture and characteristics. For instance, TPU platform shows better flexibility for CNN models and large batch sizes. As we discussed in chapter 4, these two properties (CNN based models and large batch sizes) meet the

requirements for efficient data parallelism. TPU can be the best hardware to support the data parallelism on CNN based models.
As we discussed in chapter 4, model parallelism is the best option to scale up RNN based models which are involved with large number of the fully connected layers(large number of the parameters). Even though the model parallelism improves scalability, applying this approache bring some challenges such as communication delay and overhead. On the other hand, CPU and GPU can provide more memory and bandwidth for commutation and intermediate results therefore these hardwares satisfy the model parallelism approach requirements.

8 References

[1] Cneuromod dataset. https://www.cneuromod.ca/gallery/datasets/, 2021.

[2] Intel® optane™ dc persistent memory. https://www.intel.ca/content/www/ca/en/architecture-and-technology/optane-dc-persistent-memory.html, 2021.

[3] Intel® xeon® scalable processors. https://software.intel.com/content/www/us/en/develop/articles/intel-processors-for-deep-learning-training.html, 2021.

[4] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, and M. Kudlur. A system for large-scale machine learning. In 12th USENIX symposium on operating systems design and implementation, 16:265–283, 2016.

[5] F. Alfaro-Almagro, M. Jenkinson, N.K. Bangerter, J.L. Andersson, L. Griffanti, G. Douaud, S.N. Sotiropoulos, S. Jbabdi, M. Hernandez-Fernandez, E. Vallee, and D. Vidaurre. Image processing and quality control for the first 10,000 brain imaging datasets from uk biobank. Neuroimage, 166:400–424, 2018.

[6] E.J. Allen, G. St-Yves, Y. Wu, J.L. Breedlove, L.T. Dowdle, B. Caron, F. Pestilli, I. Charest, J.B. Hutchinson, T. Naselaris, and K. Kay. A massive 7t fmri dataset to bridge cognitive and computational neuroscience. bioRxiv, 2021.

[7] K. Amunts, C. Lepage, L. Borgeat, H. Mohlberg, T. Dickscheid, M.É. Rousseau, S. Bludau, P.L. Bazin, L.B. Lewis, A.M. Oros-Peusquens, and N.J. Shah. Bigbrain: an ultrahigh-resolution 3d human brain model. Science, 340:1472–1475, 2013.

[8] Y.E. AWang, G.Y. Wei, and D. Brooks. Benchmarking tpu, gpu, and cpu platforms for deep learning. arXiv preprint arXiv:1907.10701, 2019.

[9] R. Beliy, G. Gaziv, A. Hoogi, F. Strappini, T. Golan, and M Irani. From voxels to pixels and back: Self-supervision in natural-image reconstruction from fmri. In Advances in Neural Information Processing Systems, pages 6517–6527, 2019.

[10] P. Bellec and J. Boyle. Bridging the gap between perception and action: the case for neuroimaging, ai and video games. Psy,ArXiv, 2019.

[11] N. Chang, J.A. Pyles, A. Marcus, A. Gupta, M.J. Tarr, and E.M. Aminoff. Bold5000: a public fmri dataset while viewing 5000 visual images. Scientific data, pages 1–18, 2019.

[12] D. Changde, L. Jinpeng, H. Lijie, and H. Huiguang. Brain encoding and decoding in fmri with bidirectional deep generative models. Nature communications, 2019.

[13] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, and C. Guestrin. An automated end-to-end optimizing compiler for deep learning. In 13th USENIX Symposium on Operating Systems Design and Implementation, 18:578–594, 2018.

[14] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. A survey of model compression and acceleration for deep neural networks. arXiv preprint arXiv:1710.09282, 2017.

[15] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. Symposium on Operating Systems Design and Implementation, 14:571–582, 2014.

[16] J. Deng, W. Dong, R. Socher, L.J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. IEEE conference on computer vision and pattern recognition, pages 248–255, 2009.

[17] W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. arXiv preprint arXiv:2101.03961, 2021.

[18] Ashby FG. An introduction to fmri. An introduction to model-based cognitive neuroscience, pages 91–112, 2015.

[19] J. Goense, Y. Bohraus, and N.K. Logothetis. fmri at high spatial resolution: implications for bold-models. Frontiers in computational neuroscience, 10:p.66, 2016.

[20] V. Hayot-Sasson, S.T. Brown, and T. Glatard. Performance benefits of intel® optane™ dc persistent memory for the parallel processing of large neuroimaging data. 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), pages 509–518, 2020.

[21] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, and J. Law. Applied machine learning at facebook: A datacenter infrastructure perspective. International Symposium on High Performance Computer Architecture (HPCA), pages 620–629, 2018.

[22] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[23] T. Horikawa, S.C. Aoki, M. Tsukamoto, and Y. Kamitani. Characterization of deep neural network features by decodability from human brain activity. Scientific data, page 190012, 2019.

[24] X. Hu, L. Guo, J. Han, and T. Liu. Decoding power-spectral profiles from fmri brain activities during naturalistic auditory experience. Brain imaging and behavior, 11(1):253–263, 2017.

[25] H. Huang, X. Hu, Q. Dong, S. Zhao, S. Zhang, Y. Zhao, L. Quo, and T. Liu. Modeling task fmri data via mixture of deep expert networks. 15th International Symposium on Biomedical Imaging, pages 82–86, 2018.

[26] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q.V. Le, and Y. Wu. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In Advances in neural information processing systems, pages 103–112, 2019.

[27] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, and T. Chen. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. arXiv preprint arXiv:1807.11205, 2018.

[28] K.N. Kay, T. Naselaris, R.J. Prenger, and J.L. Gallant. Identifying natural images from human brain activity. Nature, pages 352–355, 2008.

[29] A. Koliousis, P. Watcharapichat, M. Weidlich, L. Mai, P. Costa, and P. Pietzuch. Crossbow: scaling deep learning with small batch sizes on multi-gpu servers. arXiv preprint arXiv:1901.02244., 2019.

[30] H. Lim, D.G. Andersen, and M. Kaminsky. 3lc: Lightweight and effective traffic compression for distributed machine learning. arXiv preprint arXiv:1802.07389, 2018.

[31] T.Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C.L. Zitnick. Microsoft coco: Common objects in context. In European conference on computer vision, pages 740–755, 2014.

[32] M. Makkie, H. Huang, Vasilakos A.V. Zhao, Y., and T. Liu. Fast and scalable distributed deep convolutional autoencoder for fmri big data analytics. Neurocomputing, 325:20–30, 2019.

[33] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In Proceedings Eighth IEEE International Conference on Computer Vision, 2:416–423, 2001.

[34] R. Mayer and H.A. Jacobsen. Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools. ACM Computing Surveys (CSUR), pages 1–37, 2020.

[35] C.C. MChen, C.L. Yang, and H.Y. Cheng. Efficient and robust parallel dnn training through model parallelism on multi-gpu platform. arXiv preprint arXiv:1809.02839, 2018.

[36] N. Mellempudi, S. Srinivasan, D. Das, and B. Kaul. Mixed precision training with 8-bit floating point. arXiv preprint arXiv:1905.12334, 2019.

[37] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu. Mixed precision training. arXiv preprint arXiv:1710.03740, 2017.

[38] A. Mirhoseini, A. Goldie, H. Pham, B. Steiner, Q.V. Le, and J. Dean. A hierarchical model for device placement. In International Conference on Learning Representations, 2018.

[39] A. Mirhoseini, H. Pham, Q.V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, and J. Dean. Device placement optimization with reinforcement learning. arXiv preprint arXiv:1706.04972, 2017.

[40] T.M. Mitchell, S.V. Shinkareva, A. Carlson, K.M. Chang, V.L. Malave, R.A. Mason, and M.A. Just. Predicting human brain activity associated with the meanings of nouns. science, 320(5880):1191–1195, 2008.

[41] S.R. Nandakumar, M. Le Gallo, C. Piveteau, V. Joshi, G. Mariani, I. Boybat, G. Karunaratne, R. Khaddam-Aljameh, U. Egger, A. Petropoulos, and T. Antonakopoulos. Mixed-precision deep learning based on computational memory. Frontiers in Neuroscience, 14:406, 2020.

[42] T. Naselaris, E. Allen, and K. Kay. Extensive sampling for complete models of individual brains. current opinion in behavioral sciences. bioRxiv, 40:45–51, 2021.

[43] S.R. Oota, A. Avvaru, N. Manwani, and R.S. Bapi. Mixture of regression experts in fmri encoding. arXiv preprint arXiv:1811.10740, page 21, 2018.

[44] M. Ott, S. Edunov, D. Grangier, and M. Auli. Scaling neural machine translation. arXiv preprint arXiv:1806.00187, 2018.

[45] R.A. Poldrack, J.A. Mumford, and T.E Nichols. Handbook of functional mri data analysis. Cambridge University Press, 2011.

[46] R. Puri, R. Kirby, N. Yakovenko, and B. Catanzaro. Large scale language modeling: Converging on 40gb of text in four hours. 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), pages 290–297, 2018.

[47] K. Qiao, J. Chen, L. Wang, C. Zhang, L. Zeng, L. Tong, and B. Yan. Category decoding of visual stimuli from human brain activity using a bidirectional recurrent neural network to simulate bidirectional information flows in human visual cortices. Frontiers in neuroscience, 2019.

[48] K. Seeliger, L. Ambrogioni, Y. Güçlütürk, L.M. van den Bulk, U. Güçlü, and M.A.J. van Gerven. End-to-end neural system identification with neural information flow. PLOS Computational Biology, 17(2), 2021.

[49] K. Seeliger, R.P. Sommers, U. Güçlü, S.E. Bosch, and Van Gerven. A large single-participant fmri dataset for probing brain responses to naturalistic stimuli in space and time. bioRxiv, page 687681, 2019.

[50] G. Shen, K. Dwivedi, K. Majima, T. Horikawa, and Y. Kamitani. End-to-end deep image reconstruction from human brain activity. Frontiers in computational neuroscience, 13:21, 2019.

[51] D.C. Van Essen, S.M. Smith, D.M. Barch, T.E. Behrens, E. Yacoub, K. Ugurbil, and Wu-Minn HCP Consortium. The wu-minn human connectome project: an overview. Neuroimage, 80:62–79, 2013.

[52] X. Wang, X. Liang, Z. Jiang, B.A. Nguchu, Y. Zhou, Y. Wang, H. Wang, Y. Li, Y. Zhu, F. Wu, and J.H. Gao. Decoding and mapping task states of the human brain via deep learning. In Proceedings of the ieee conference on computer vision and pattern recognition, pages 1505–1519, 2020.

[53] X.W. Wang, D. Nie, and B.L. Lu. Emotional state classification from eeg data using machine learning approach. Neurocomputing, 129:94–106, 2014.

[54] H. Wen, J. Shi, Y. Zhang, K.H. Lu, J. Cao, and Z.L Liu. Neural encoding and decoding with deep learning for dynamic natural vision. Cerebral Cortex, 28(12)):4136–4160, 2018.

[55] S. Xi, Y. Yao, K. Bhardwaj, P. Whatmough, G.Y. Wei, and D. Brooks. Smaug: end-to-end full-stack simulation infrastructure for deep learning workloads. ACM Transactions on Architecture and Code Optimization, 17(4):1–26, 2020.

[56] J. Xiao, J. Hays, K.A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. computer society conference on computer vision and pattern recognition, pages 3485–3492, 2010.

[57] Y. You, Z. Zhang, C.J. Hsieh, J. Demmel, and K. Keutzer. Imagenet training in minutes. Proceedings of the 47th International Conference on Parallel Processing, pages 1–10, 2018.

[58] S.D. Yun and N.J. Shah. Whole-brain high in-plane resolution fmri using accelerated epik for enhanced characterisation of functional areas at 3t. PloS one, 12(9):p.e0184759, 2017.

[59] J. Zhang, S. H. Yeung, Y. Shu, B. He, and W. Wang. Efficient memory management for gpu-based deep learning systems. arXiv preprint arXiv:1903.06631, 2019.

[60] Y. Zhang, L. Tetrel, B. Thirion, and P. Bellec. Functional annotation of human cognitive states using deep graph convolution. bioRxiv, 2020.