Student-Teacher Curriculum Learning via Reinforcement Learning: Predicting Hospital Inpatient Admission Location

Rasheed el-Bouri [1]   **David Eyre** [1 2]   **Peter Watkinson** [2]   **Tingting Zhu*** [1]   **David A. Clifton*** [1]

Abstract

Accurate and reliable prediction of hospital admission location is important due to resource-constraints and space availability in a clinical setting, particularly when dealing with patients who come from the emergency department. In this work we propose a student-teacher network via reinforcement learning to deal with this specific problem. A representation of the weights of the student network is treated as the state and is fed as an input to the teacher network. The teacher network's action is to select the most appropriate batch of data to train the student network on from a training set sorted according to entropy. By validating on three datasets, not only do we show that our approach outperforms state-of-the-art methods on tabular data and performs competitively on image recognition, but also that novel curricula are learned by the teacher network. We demonstrate experimentally that the teacher network can actively learn about the student network and guide it to achieve better performance than if trained alone.

1. Introduction

A major problem that many hospital wards face is the timely preparation of beds for patients from the emergency department (ED) who are due to be admitted as inpatients (Staib et al., 2017). The average time between initial examination (triage) in the ED and a request for a bed in the hospital is 4 hours (Francis, 2013) and the only information available to staff about the patient is that collected at triage. This information can be helpful to deduce the patient's condition, which in turn can be used to predict which of the relevant inpatient wards needs to prepare space. Often, the ward is

requested after the 4 hours have passed in a reactive manner. Predictively allocating patients to wards however, should allow for the timely care and admission of the patient as well as allocation of resource to the relevant departments. Furthermore this may contribute to reducing crowding of the ED as well as patient waiting times. With this in mind, we present a method of predicting where in a hospital emergency patients will be admitted after being triaged using a combination of reinforcement and curriculum learning.

Curriculum learning was introduced to train a neural network in a similar manner to that in which humans are educated. The approach stems from the observation that children in schools learn by beginning with simple ideas and progressing on to more complex topics; it is believed that neural networks may also benefit from this structured approach to learning. By initially presenting the network with data that are 'easier' to fit, the optimisation surface (of network prediction error vs. network parameters) is more likely to be convex (Bengio et al., 2009).

In this paper we propose using neural networks for classification by exploiting this notion of a curriculum and extending the teacher-student curriculum learning method of (Matiisen et al., 2017). We use two networks, where one (the teacher) is trained with information about the state of the other (the student) in order to guide training and maximise performance. A teacher's observations of many previous students provides them with an idea of how a student can best learn (Papay & Kraft, 2015) using a pre-defined curriculum. This work aims to mimic this action of learning from an agent (that is itself learning), with the aim of creating a policy (the function that maps a state to an action) to guide a future agent to achieve successful performance.

The contributions of this work include the development of a student-teacher learner, where the teacher is trained using reinforcement learning which, to the best of the authors' knowledge, is the first work to propose such a student-teacher configuration. It is also believed to be the first work to use a representation of the weights extracted from the student as inputs to the teacher. This method has demonstrably better performance for our healthcare application than all others to which it was compared. We believe that the resulting classifier will prove a useful tool for hospitals in

---

*Equal contribution   [1]University of Oxford [2]Oxford University Hospitals Trust.   Correspondence to: Rasheed el-Bouri <rasheed.el-bouri@eng.ox.ac.uk>.

order to predict the resources required ahead of time and improve the movement of patients out of the ED and into the wards.

Section 2 discusses related work and Section 3 outlines the setup of the student-teacher network. Results are in Section 5 and we discuss the curricula generated by the teachers in Section 6 before outlining the limitations of this approach in Section 7.

2. Related Work

In this work, we create a teacher agent that learns to train a feedforward student network by selecting the data to present to the student. We organise our data into batches according to a curriculum. We define the 'easiness' of a batch of data according to the entropy within the batch; i.e, a batch B_i is said to be easier than batch B_j if $H[B_i] < H[B_j]$, where H is the entropy of the batch. It has been shown in (Wu & Tian, 2016) and (Pentina et al., 2015) amongst others that training neural networks in a structured fashion by presenting batches from easiest to hardest leads to improvement in performance of the network, as well as increasing the speed of training. However, we also see in the works of (Hacohen & Weinshall, 2019) and (Bengio et al., 2009) that training in an 'anti-curriculum' (hardest to easiest data) can also lead to improved performance. A curriculum may also be defined for specific problem sets, with results from (Zaremba & Sutskever, 2014) and (Bengio et al., 2015) showing that a curriculum tailored for the task at hand provided better results than a general curriculum. This is indicative that training according to some type of structure is important for learning, and that structure may be task specific.

Student-teacher curriculum learning has been explored in (Matiisen et al., 2017) where a similar initial configuration to that proposed here is used. Progress of the student is measured through its improvement in task performance, which in turn affects the probability of selecting a curriculum batch on which to train. The authors proposed four different algorithms to monitor the progress of the student. These are (i) the use of a non-stationary bandit to select the next training batch, (ii)-(iii) using linear regression and a windowed linear regression on the task accuracy to predict the batch most likely to provide the greatest improvement in performance, and (iv) using Thompson sampling to select the next batch. While these approaches have an effective performance on those specific problems presented in the existing studies, the selection of data is not based on information pertaining to the state of the student (but instead is based on prior information of the student in its previous state, and from previous performance of the algorithms). In this work, we propose using knowledge of the current state of the student to develop a policy for the teacher to train future students. Another work which uses a similar approach is that of (Graves et al., 2017)

where a curriculum is also generated and a non-stationary bandit is used with the EXP3.S algorithm (Auer et al., 2002). However, again ignoring the state of the student, this latter algorithm is conditioned on the previous performance of the student.

Reinforcement learning has proven to be highly effective in many applications, canonically within the domain of games (Oh et al., 2015; Kelly & Heywood, 2017). The training of a neural network can also be seen as a game where the task is to choose the right combination of training data to maximise a score, which in this case is the prediction accuracy. This makes reinforcement learning an attractive choice for training the teacher network. Our approach has parallels with the field of hierarchical learning, in which a higher-level agent guides the training of a lower-level agent (Kulkarni et al., 2016). However to the best of the authors' knowledge, none of the methods used in hierarchical learning operate directly on the weight space of the lower-level agent.

3. Methodology

3.1. Data Pre-Processing

The first task for our model is to organise data into batches according to their complexity. As previously mentioned this can be achieved according to the entropy of the data (among other methods not described here for brevity). For tabular data (categorical and numerical data such as the information recorded at triage), we choose to represent the entropy through the use of the Mahalanobis distance. The Mahalanobis distance is defined as:

$$d\left(\mathbf{x}_n\right) = \left(\left(\mathbf{x_n} - \boldsymbol{\mu}\right)^T \mathbf{S^{-1}} \left(\mathbf{x_n} - \boldsymbol{\mu}\right)\right)^{\frac{1}{2}} \qquad (1)$$

where $\mathbf{x_n}$ are the input features of datapoint n, $\boldsymbol{\mu}$ is the vector of the mean value of each feature, and $\mathbf{S}$ is the covariance matrix. As certain inputs to a neural network may be of different data types we encode the input data prior to organising into a curriculum using a denoising autoencoder for the Mahalanobis distance to be defined in a scalar space.

Let $\mathcal{D}$ be a training dataset consisting of inputs and labels x_n and y_n respectively. We first train a denoising autoencoder such that $\phi\left(x_n\right) \approx x_n$. To construct our curriculum from this latent representation, we calculate the Mahalanobis distance, $d\left(\phi\left(x_n\right)\right)$ for all n. We then sort the data from the lowest Mahalanobis distance to the highest and create N batches by dividing the whole set into N separate batches. Two methods of creating the batches were investigated:

1. All batches are separate and contain separate training data; i.e, if B_0 is the batch with the lowest $d\left(\phi\left(x_n\right)\right)$ and B_N that with the highest, then $B_0 \cap B_1 \cap \ldots B_N = \varnothing$

2. Batches are cumulative supersets; i.e, $B_0 \subset B_1 \subset \ldots B_N$.
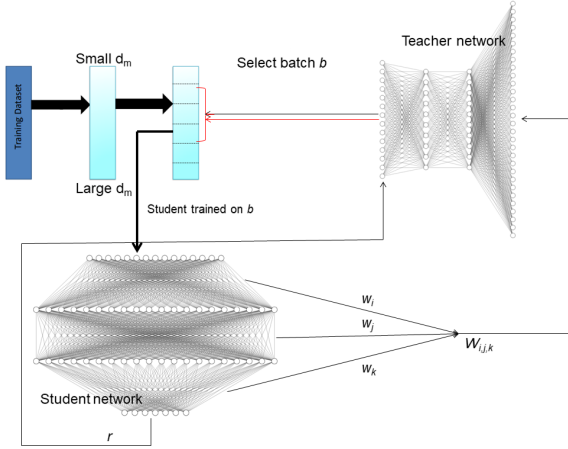

Figure 1. The architecture of the model we are proposing. The data is separated according to d_m, the Mahalanobis distance. The reward from the student, r is fed back to the teacher, as well as a representation of the weights from each layer of the student network. The first output of the teacher (black) is where along the sorted data to sample from, the second (red) is how much data around this to include in the batch.

3.2. Teacher Network

Figure 1 shows the overall architecture for the student-teacher network being proposed in this work. The teacher network, as most reinforcement learning agents, is characterised by the state, action, reward, next state tuple *(s,a,r;s')* (Sutton et al., 1998). Using these we are able to discern the value of the student being in a given state and taking a certain action (being trained on certain data) for a given task.

We seek to discover a curriculum that will improve the performance of the student over alternative training methods. As previously mentioned, we aim to condition on the current configuration of the student and so we define the *state* therefore to be the weights of the student network, i.e, a representation of the weights between all of the nodes for all layers as described in Section 3.4.

The *actions* of the teacher are selecting the ideal batch of training data that are organised by a curriculum, to train the student network. This is done through the policy by selecting the appropriate data to train with based on the output as described in Section 3.3. Note that in this problem, the number of actions can change depending on how many batches we define our curriculum to have, or the method with which we select data to create a batch.

The *reward* is given by the improvement in accuracy of the student compared to the previous accuracy achieved before the training data batch was trained, i.e., $r_t = \delta_t - \delta_{t-1}$ at step t where δ is the accuracy of the student on the training set. However, this would lead to overfitting if only trying to maximise the training accuracy. As a result we measure the students accuracy on the training set and the validation set jointly by multiplying them as $(\delta_t - \delta_{t-1}) A_v$ where A_v is the validation accuracy to formulate the new reward. This way if overfitting occurs, the validation accuracy reduces thereby reducing the overall reward.

Our *next state*, s' is the updated weights of the student network after training on the batch chosen by the teacher.

To train the teacher we must make use of the Bellman equation. This describes the value of an agent (any model or actor which has a state and the capability to take actions) being in a certain state for a given task. It can also describe the value of an agent being in a certain state and taking a certain action for that same task (Q-value) (Wiering & Van Otterlo, 2012). The equation is described by:

$$Q^*(s,a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^*(s',a') \mid s,a] \quad (2)$$

where Q^* is the optimal action-value function, r is the reward of the action that was taken, γ is the discount factor, s, a and s', a' are the current state and action and the next state and action respectively. The discount factor is a way of weighting whether future or immediate rewards are important, with high values of γ favouring a long-term reward and low values favouring immediate reward. The discount factor is generally treated as a hyper-parameter.

The intuition behind Equation 2 is that, if we have a function Q^* which tells us the value of being in a certain state and taking a certain action, then naturally, optimal behaviour is given by choosing the next action which maximises this.

For large state spaces such as the weight space of the student, a function for the Q-values must be approximated (Gaskett et al., 1999). This can be done using function approximation methods such as deep neural networks. Now that $Q(s,a)$ is dependent upon network parameters, θ, it becomes $Q(s,a;\theta)$ (note that Q here is the action-value function for the current policy and not Q^*, the optimal policy) and the loss function to train our teacher network is given by:

$$\mathcal{L}(\theta_i) = \left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2 \quad (3)$$

where θ^- are the parameters of the target network (the version of the teacher that is held constant for K steps to stabilise training as described in (Mnih et al., 2015)).

Experience replay is used to decorrelate the data being trained on with the current policy. This is required to stabilise the learning of the Q-function. A buffer, R is kept of

the previous (s, a, r, s') tuples that the teacher has encountered. A random sample of R of size m is then taken and is presented to the teacher as a batch for training. We train the teacher on the students most recent experience for m steps and then train the teacher on the random sample of historic experience of size m.

The architecture used for the teacher network is a fully-connected feedforward neural network with 3 hidden layers and 50 nodes in each layer. The hidden layer nodes are activated by the ReLU function and a dropout rate of 20% is used to prevent overfitting. Stochastic gradient descent with momentum (Sutskever et al., 2013) is used to train the network weights with a momentum value of 0.9. A discount factor of 0.95 is used and the target network is updated after every 20 times the non-target network is updated. An experience replay batch size of 10 is used after every 10 updates on the non-target network.

3.3. The Teacher Action Space

Maximum flexibility in data selection would be to allow for a continuous action space, such as making every entry of data in the training set a batch of data in itself. We use deep deterministic policy gradients as described in (Lillicrap et al., 2015) which allows us to parameterise the action selection. With this approach, the teacher, g, consists of two networks, an actor and a critic. The actor is a network which selects the actions to take and the critic evaluates the 'value' of being in a particular state. The actor, parameterised by θ^μ (of which there is also a target actor), takes the encoded state of the student, f, as input and has two outputs. The first is the index along the curriculum (after ranking our training data according to the curriculum) at which we will centre our batch and the second is the width around this datapoint that we will expand our batch to. For example if our first output gives us the value 30 and the second output gives 50, then we will centre our curriculum batch on the datapoint in the curriculum indexed at 30 and select everything 25 below and 25 above this datapoint as a batch of data to train on.

The critic, parameterised by θ^Q (of which there is also a target critic) has two outputs (for our two independent actions, index selection and batch width selection) which are the Q-values to which the reward is added for training. The input to the critic is the same as the input to the actor (the state of the student) except it is altered by concatenating onto this the actions selected by the actor which also have some exploration noise applied. The critic is then trained with loss as in Equation 3. The actor is updated using the following loss:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q\left(s, a \mid \theta^Q\right)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu\left(s \mid \theta^\mu\right)$$

(4)

where Q is the Q-function and μ is the policy. For a deriva-

tion of this please see (Lillicrap et al., 2015). A pseudocode is included in Algorithm 1 for a detailed description of how the teacher network is trained. We also experiment with

---

Algorithm 1 The student-teacher training routine for discrete batches using the DDPG algorithm

---

Data: Training dataset organised into N batches of Mahalnobis curriculum

  initialise teacher critic network, Q, actor network, μ

  initialise target teacher and actor, Q_T, μ_T

  initialise random process $\mathcal{N}$ for action exploration

  initialise replay buffer, R, select batchsize of replay, m

  select update frequency value, U

  select stable update value, τ

for x in X students **do**

    initialise student network, f_x

    for i in I iterations **do**

     -Extract state of f_x, s_i

     -Select action $a_i = \mu\left(s_i \mid \theta^\mu\right) + \mathcal{N}_i$ according to current policy and exploration noise

     -Execute action a_i and observe reward r_i based on improvement in performance on validation set and observe new state s_{i+1}

     -Store transition (s_i, a_i, r_i, s_{i+1}) in replay buffer, R

     -Sample random minibatch of n transition tuples from R

     -Set $y_i = r_i + \gamma Q_T\left(s_{i+1}, \mu_T\left(s_{i+1} \mid \theta^{\mu_T}\right) \mid \theta^{Q_T}\right)$

     -Update critic by minimising:

     $L = \frac{1}{n} \sum_i \left(y_i - Q\left(s_i, a_i \mid \theta^Q\right)\right)^2$

     -Update actor using sampled policy gradient:

     $\nabla_{\theta^\mu} J \simeq \frac{1}{N} \sum_i \nabla_a Q\left(s, a \mid \theta^Q\right) \nabla_{\theta^\mu} \mu\left(s \mid \theta^\mu\right)$

     if i mod U = 0 **then**

       Update target networks:

       $\theta^{Q_T} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q_T}$

       $\theta^{\mu_T} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu_T}$

     else

       continue

     end

    end

end

---

teachers with discrete action spaces and compare the performance. The discrete action space teachers are trained using more appropriate algorithms such as deep Q-networks (DQN). A pseudocode for this type of teacher can be found in the supplementary material.

3.4. Student Network

The state of the student network is defined to be the matrix of weights of each layer in the network i.e, W^{ij}, where W^{ij} indicates the matrix of weights between layers i and

j. W^{ij} is of size $M_i \times M_j$ where M_i and M_j are the number of nodes in layers i and j respectively. As the size of these matrices is dependent upon the number of nodes in each layer we seek a more compact representation of these matrices. We do this by comparing each row in a weight matrix to a fixed reference vector. For example, $W^{ij}_{1:} \cdot a$ would compare the first row of matrix W^{ij} with our fixed reference, a. The purpose of the reference vector is to allow us to compare the vectors of weights and allow us to identify them relative to the same reference. We represent each row using two scalars, $|\langle W^{ij}_{n:}, a \rangle|$ and $\angle \left(W^{ij}_{n:}, a \right)$ for $n = 1, 2, \ldots, M_i$. These are concatenated to give a vector representing a weight matrix between two layers, denoted by $\mathbf{v}_i \in \mathbb{R}^{2M_i}$. This is repeated for the next layer, giving $\mathbf{v}_j \in \mathbb{R}^{2M_j}$ and the vectors for each layer are then also concatenated together as a single vector, $\mathbf{v} \in \mathbb{R}^{2(M_i + M_j)}$. This vector $\mathbf{v}$ is then used as the representation of the state of the student. As described in Section 3.2, the performance of this network on a separate validation set as well as the training set is used as the reward which is used to update the teacher.

We define the student to be a fully-connected feedforward neural network with 2 hidden layers, with nodes $M_i = M_j = 50$. Each node in the hidden layer is activated by the ReLU function apart from the final layer where a softmax function is used to classify. The student is trained using stochastic gradient descent with a fixed learning rate of 0.001. We initialise a student and allow it to be trained using $N = 100$ separate batches sorted by the Mahalanobis distance. We then initialise another student and repeat the procedure. We repeat this for 10,000 students, therefore providing the teacher with 1,000,000 tuples in the experience replay buffer, R, (the collection of the agents experiences) to sample and train from. At test time we initialise another student and allow the teacher to implement the learned policy.

4. Datasets

Ward Admission Dataset

In this study we considered the patient data collected in the electronic health records (EHR) of the Oxford University Hospitals Trust between January 2013 and April 2017. Specifically, we use the Infections in Oxfordshire Research Database (IORD). This includes administrative (e.g. date and time of arrival), demographic (e.g. age, gender and so on), as well as physiological and medical information (e.g. vital sign measurements and medical tests ordered during the patient's visit). We only consider patient arrivals to the emergency department. Any historical data stored about the patient will also be available in the EHR upon their next arrival to the emergency department. As there are over 100 potential locations of admission within the hospital, we

group these into seven 'ward functions' where wards in a single 'function' are capable of treating patients with similar conditions making this a seven-class classification. Only patients who were admitted in an emergency were considered providing a dataset of 14,324 patients. A training set of 60% of the dataset was used and was balanced, leaving 8,589 patients for training on. The validation set was 20% of the dataset and testing was also 20% and the classes were kept in the same distribution as the original dataset.

MIMIC-III Dataset

To validate the efficacy of the methodology we implement the algorithm on another classification problem from the MIMIC-III dataset in the next section (Johnson et al., 2016). The patients for this dataset are also emergency patients only and have all features available. This provides us with a dataset of 8806 patients. These were split into the same train-validation-test proportions as before with only the training set being balanced as before. As MIMIC-III is an ICU focused dataset, replicating the experiment we have carried out with the ward admission dataset is not possible. As a result we create a new problem of classifying the mortality of patients (binary classification) based on 11 features that are available early in the patient's admission.

CIFAR-10

To further validate our methodology we also report results on the CIFAR-10 image recognition dataset (Krizhevsky, 2009) to assess the model's performance on a non-tabular dataset. Due to our model having the capability of selecting large batches and memory constraints, we take a random, stratified sample from the training set of 12,000 images of which 1,200 are made into a validation set. We then take a random, stratified sample of 3000 images from the test set as our held-out test set. We repeat this three times essentially creating three separate datasets which we identify as sample 1, sample 2 and sample 3 respectively in Table 1.

5. Results

To make a comparison between our student-teacher network and other state-of-the-art methods for classification, we show normal stochastic mini-batch training (batchwise), vanilla curriculum training (curriculum) and the method proposed by (Matiisen et al., 2017) (ST + curric). We report the (ST + curric) result for the student-teacher method that worked best for each dataset out of the four approaches discussed in Section 2. The performances for both methods of batch formation (as discussed in Section 3.1) were compared. It was found that for direct curriculum training (curriculum) method 2 was best whereas for the student-teacher methods (ST + curric, DQN student-teacher, DDPG student-teacher), method 1 worked best.

Table 1. Average classification accuracies and standard deviations for various baseline and state-of-the-art methods on the Ward Admission (tabular), MIMIC-III (tabular) and CIFAR-10 (image) datasets. All models are averaged over the same five seeds apart from those highlighted with * which indicates that the accuracy reported from the cited text is quoted.

METHOD	WARD ADMISSION ACC (SD)	MIMIC-III ACC (SD)	CIFAR-10 SAMPLE 1 ACC (SD)	CIFAR-10 SAMPLE 2 ACC (SD)	CIFAR-10 SAMPLE 3 ACC (SD)
BATCHWISE	0.45 (0.01)	0.63 (0.01)	0.65 (0.02)	0.65 (0.02)	0.65 (0.02)
CURRICULUM	0.48 (0.02)	0.63 (0.02)	0.68 (0.01)	0.68 (0.01)	0.68 (0.01)
ST + CURRIC	0.53 (0.03)	0.63 (0.02)	0.68 (0.02)	0.68 (0.02)	0.68 (0.02)
DEEPFM	0.59 (0.01)	0.66 (0.01)	N/A	N/A	N/A
DEEP+CROSSNET	0.58 (0.02)	0.68 (0.02)	N/A	N/A	N/A
AUTOINT	0.57 (0.02)	0.67 (0.01)	N/A	N/A	N/A
DENSENET*	N/A	N/A	0.96 (0.01)	0.96 (0.01)	0.96 (0.01)
GPIPE*	N/A	N/A	**0.99** (0.01)	**0.99** (0.01)	**0.99** (0.01)
DQN STUDENT-TEACHER	0.58 (0.01)	0.66 (0.02)	0.86 (0.04)	0.88 (0.02)	0.86 (0.03)
DDPG STUDENT-TEACHER	**0.62** (0.02)	**0.70** (0.01)	0.90 (0.01)	0.90 (0.01)	0.89 (0.01)

For the Ward Admission and MIMIC-III datasets we also compare to state-of-the-art methods for classifying tabular data, namely DeepFM (Guo et al., 2017), Deep+CrossNet (Wang et al., 2017) and AutoInt (Song et al., 2019). For the CIFAR-10 dataset we compare with models that achieve state-of-the-art performance on image datasets, namely DenseNet (Huang et al., 2017) and GPipe (Huang et al., 2019). For our non-medical dataset, CIFAR-10, we sort data using cosine similarity as entropy. We must use a different student for this dataset and so we use a student that has 4 convolutional layers with 32 filters of size 3x3 in the first two layers and 64 of these in the second two. These are all activated by ReLU and maxpooled and are followed by 3 feedforward layers of size 50 nodes each. We use the encoding of the weights of the feedforward component of the network as the state of the student. From Table 1, we see that our method (DDPG trained student-teacher network) outperforms at least 3 state-of-the-art methods for classifying tabular data. It is also capable of providing a competitive performance on image classification problems, approaching state-of-the-art performance. This demonstrates the robustness of this method as a training methodology for various datasets. In Section 6 and the supplementary material we go on to investigate the policies that have been generated for training and discuss them in the context of the curriculum.

6. Investigation of Policies Learned

Learned Curricula

In this section we investigate the policies learned by the teachers through their experience of training many students. We implement the student in such a way that the configuration of the student at which the highest accuracy is achieved is saved. From Figure 2 we see that the student accuracy curve does not follow the usual smooth training arc seen during stochastic mini-batch training. We can see areas

where the teacher aids the student in escaping local minima. The teacher swinging from training the student with low entropy data to high entropy data just before iteration 4000 in Figure 3 corresponds to a degradation in performance of the student in Figure 2. This may indicate that the teacher has used the high entropy data to push the student out of a local minimum hence why the performance degrades. This turns out to be an unsuccessful attempt as with further training the model does not perform any better than it had done previously. However, just prior to iteration 7000, the teacher once again performs a 'high entropy spike' (shown by the orange crosses rising). This again deteriorates the performance by escaping from a local minimum, but with further training the student achieves a performance of 62% (indicated by the red dotted line in Figure 2, the highest it has achieved so far. We show policies learned for the MIMIC-III and CIFAR-10 datasets in the supplementary material.

Constrained Policy Learning

To demonstrate that the curriculum is indeed guided by the teacher and not simply providing an alternative optimisation trajectory, we train two identical teachers, one with no constraints and one that has been constrained in some way and we compare the policies learned. We use a DDPG teacher where the constrained teacher uses 1 datapoint at a time with probability 0.999 (i.e., the teacher's second output (batch width) is almost always 0) and uses the real teacher output otherwise. The students of these teachers are trained on the CIFAR-10 dataset. From Figure 4, we see a fairly typical curriculum with a relatively noisy starting phase going from low to high entropy then rapidly back to low. However the general trend is that low entropy data is used and trained with before progressing onto the higher entropy data. In Figure 5 we see that we begin in a similar way as in Figure 4 (rapidly going from low to high entropy) but after this the smooth transition from low entropy to high entropy is not
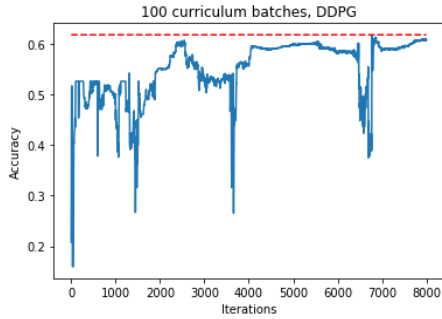
Figure 2. The performance of the student on the held-out test set of the Ward Admission prediction while it is trained by the teacher. The red dashed line is the best performance achieved by this student. Notice that it does not follow the typical smooth learning arc usually seen.
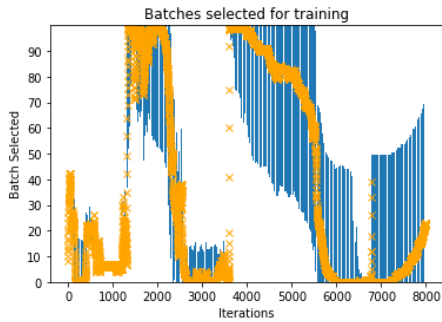

Figure 3. Actions generated by the policy of the teacher that has led to the performance of the student shown in Figure 2. Orange crosses are the first output (where to select data from) and blue bars are the second output (how much data around the central selection point to include in the batch for training). If the batch selected is near zero then this is low entropy data and if it is near the top of the batch selection then this is high entropy data.

Figure 4. Actions generated by the policy of an unconstrained DDPG teacher on the CIFAR-10 dataset. Orange is the first output (where to select data from) and blue is the second output (how much data to include in the batch). If the batch selected is near zero then this is low entropy data and if it is near the top of the batch selection then this is high entropy data.

Figure 5. Actions generated by the policy of a constrained DDPG teacher on the CIFAR-10 dataset where it is highly probable that only one datapoint will be selected at a time for training. We see that the selection is significantly more chaotic. Batch numbers near zero are low entropy data and batch number 100 contains the highest entropy data.

observed. It would seem that due to individual datapoints providing much noisier gradients than batch updates, the teacher must oscillate rapidly between high and low entropy data in an attempt to provide effective training. It is interesting to observe that the teacher in both cases samples along the entire entropy axis before converging to high entropy. We also show in the supplementary material the effect of reducing the learning rate of the student as another form of constraint.

Policy Stability for Similar Students

To investigate the stability of the policies learned by the teacher we save the state of the student at every iteration and corrupt this signal with zero-mean Gaussian noise with
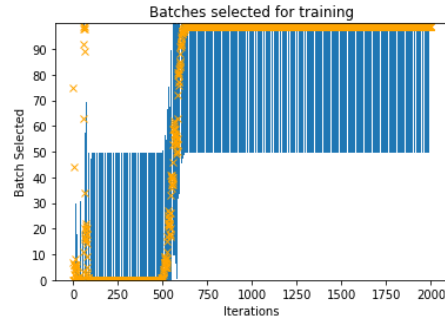
standard deviation 0.1. We see from Figures 6 and 7 that whilst the policies have changed slightly, the overall strategy of the policy is the same. This is encouraging as this is indicative of learned behaviour conditioned on the state as opposed to providing an alternative form of stochastic training. We provide further examples of stable policies for the other datasets in the supplementary material.

Policy Transfer between Tasks

We also investigated how transferable the trained teachers are between tasks. A teacher was trained on the Ward Admission dataset and transferred to the MIMIC-III prediction problem with no fine-tuning. The policy the teacher generated on the 'as before unseen' training set is shown in
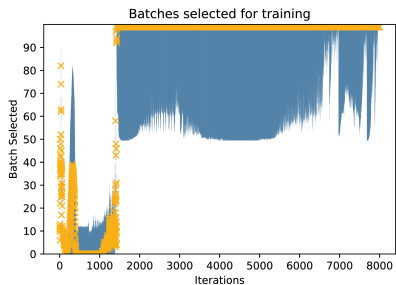
Figure 6. Actions selected by the policy of the teacher for a student to train on the Ward Admission dataset.
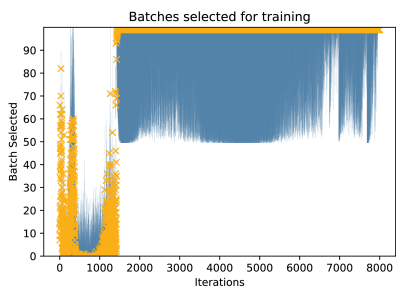

Figure 7. How the actions of the teacher shown in Figure 6 have changed by applying Gaussian noise to the states of the student.

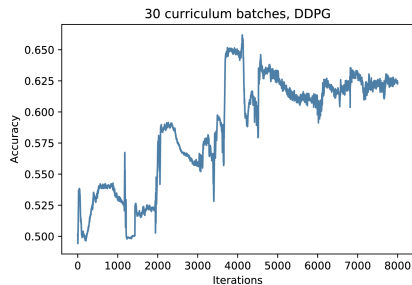*Figure 8.* Performance of student on MIMIC-III dataset when trained by a transferred teacher taught to train students on the Ward Admission dataset.
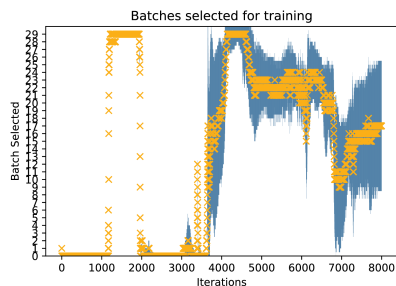

Figure 9. Actions selected by the transferred policy for the randomly initialised student which yielded the performance shown in Figure 8.

Figure 9 and the corresponding result is shown in Figure 8.

Here we see that the teacher once again uses the 'entropy spiking' technique to escape possible local minima resulting in the waves of improvement in accuracy shown in Figure 8, characteristic of curriculum learning. The performance attained is less than it would be when training the teacher directly using the MIMIC-III dataset (averaging $67\% \pm 1\%$ versus $70\% \pm 1\%$, however it still outperforms the baseline methods and performs equivalently to some of the state-of-the-art methods (shown in Table 1). It is not currently clear why the transfer of teachers seems to be successful but this is being investigated for various datasets. More examples of policies generated by transferred teachers can be seen in the supplementary material.

7. Conclusion and Limitations

In this work we have presented a method of predicting which type of hospital ward a patient will be admitted to after being triaged in the ED. We believe that this can prove a useful tool for hospitals to improve the flow of patients out of the ED and into the hospital. We have also shown that a 'teaching' neural network can learn from an encoding of the state of another network. In this way an appropriate curriculum can be generated by the former in order to maximise the performance of the latter on the task at hand. It is hoped that the insights gained from using this approach will help with curriculum design as well as provide insight into how learning occurs in neural networks.

One of the limitations we face with this approach is that the representation of the weights of the student is a form of dimensional reduction and therefore may not be the best representation of the weights of the network for learning from. To improve upon this we propose in future works to introduce an attention mechanism to identify which nodes in the network are predominantly responsible for the prediction. We will then use these weights as inputs to the teacher to remove redundancies from the input. There is also the problem that the number of inputs to the teacher relies on the size of the student where a student with M nodes in L hidden layers results in an input size of order M for the teacher. While this is better than explicit representation of the weights which would result in an input size of order M^2, we actively seek for better representations of the student's state. One method of doing this is potentially through the use of distillation networks as seen in (Hinton et al., 2015), allowing for a more compact representation of the student network and then using the student-teacher methodology proposed in this work. We are encouraged however by the

performance of the student-teacher setup on relatively small student networks achieving state-of-the-art performances. We also encourage experimentation with various curricula to rank data according to the dataset being considered. This changes the way data are ranked and therefore provides different combinations of data that can be grouped into batches. Alternatively we could investigate having different outputs for the teacher networks and investigate methods of combining these outputs to select data for training.

Acknowledgements

References

Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. The nonstochastic multiarmed bandit problem. *SIAM journal on computing*, 32(1):48–77, 2002.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 1171–1179, 2015.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48. ACM, 2009.

Francis, R. *Report of the Mid Staffordshire NHS Foundation Trust public inquiry: executive summary*, volume 947. The Stationery Office, 2013.

Gaskett, C., Wettergreen, D., and Zelinsky, A. Q-learning in continuous state and action spaces. In *Australasian Joint Conference on Artificial Intelligence*, pp. 417–428. Springer, 1999.

Graves, A., Bellemare, M. G., Menick, J., Munos, R., and Kavukcuoglu, K. Automated curriculum learning for neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1311–1320. JMLR. org, 2017.

Guo, H., Tang, R., Ye, Y., Li, Z., and He, X. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

Hacohen, G. and Weinshall, D. On the power of curriculum learning in training deep networks. *arXiv preprint arXiv:1904.03626*, 2019.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems*, pp. 103–112, 2019.

Johnson, A. E., Pollard, T. J., Shen, L., Li-wei, H. L., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Celi, L. A., and Mark, R. G. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3:160035, 2016.

Kelly, S. and Heywood, M. I. Multi-task learning in atari video games with emergent tangled program graphs. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 195–202. ACM, 2017.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, 2009.

Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pp. 3675–3683, 2016.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Matiisen, T., Oliver, A., Cohen, T., and Schulman, J. Teacher-student curriculum learning. *arXiv preprint arXiv:1707.00183*, 2017.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529, 2015.

Oh, J., Guo, X., Lee, H., Lewis, R. L., and Singh, S. Action-conditional video prediction using deep networks in atari games. In *Advances in neural information processing systems*, pp. 2863–2871, 2015.

Papay, J. P. and Kraft, M. A. Productivity returns to experience in the teacher labor market: Methodological challenges and new evidence on long-term career improvement. *Journal of Public Economics*, 130:105–119, 2015.

Pentina, A., Sharmanska, V., and Lampert, C. H. Curriculum learning of multiple tasks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5492–5500, 2015.

Song, W., Shi, C., Xiao, Z., Duan, Z., Xu, Y., Zhang, M., and Tang, J. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 1161–1170, 2019.

Staib, A., Sullivan, C., Prins, J. B., Burton-Jones, A., Fitzgerald, G., and Scott, I. Uniting emergency and inpatient clinicians across the ed–inpatient interface: The last frontier? *Emergency Medicine Australasia*, 29(6):740–745, 2017.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pp. 1139–1147, 2013.

Sutton, R. S., Barto, A. G., et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

Wang, R., Fu, B., Fu, G., and Wang, M. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*, pp. 12. ACM, 2017.

Wiering, M. and Van Otterlo, M. Reinforcement learning. *Adaptation, learning, and optimization*, 12:3, 2012.

Wu, Y. and Tian, Y. Training agent for first-person shooter game with actor-critic curriculum learning. 2016.

Zaremba, W. and Sutskever, I. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.