Multi-graph classification with discrete stochastic graph neural networks

Yinan ZHANG, Beril Beşbınar
EPFL

Abstract

In cancer immunotherapy, highly-multiplexed immunohistochemistry (mIHC) is a promising tool which enables accurate evaluation of tumor and immune cells based on staining results. In this study, we are interested in classifying cancerous patients represented by a bag of cell-graphs constructed from mIHC data. Yet, each multi-graph bag not only contains discriminative graphs useful for classification, but also redundant graphs whose characteristics are shared among all or most entities in the dataset. In order to solve this problem, we propose an algorithm for jointly learning graph embedding and graph sampling. Our framework introduces stochastic layers with discrete random variables into traditional graph neural networks. Experiments results show that this algorithm is effective no matter one graph or multiple graphs need to be selected.

1 Introduction

Thanks to the development of highly-multiplexed immunohistochemistry (mIHC) technology, we can analyze the tumor microenvironment with single-cell resolution. Existing algorithms often assess tumor-infiltrating lymphocyte(TIL) distribution patterns in patients via image-based deep learning methods[Swiderska-Chadaj et al. 2018] or graph-based feature analysis[Lu et al. 2018]. Though graphs are ideal for representing spatial arrangement and interactions of different cells, these graph-based methods often requires manual feature extraction, thus leading to inaccuracies. Here, we will make use of discrete stochastic graph neural networks to classify cancerous patients represented by bags of cell graphs.

In mIHC images, only some regions of interest (ROIs) are scanned at high-resolution, all of which are not always important for decision making. Therefore, we would like to investigate the feasibility of characterizing a patient with a single or few selected patches. We can make use of a software called inForm by PerkinElmer to process those ROIs, and output files with locations of cell centers and cell phenomaps. Each ROI will be processed into a graph, with cells as nodes, and interactions between them as edges. Edges are drawn based on Euclidean distance between node pairs plus thresholding, and node features will be their phenomaps encoded as one-hot vectors. Each bag of graphs representing a patient is a mixture of discriminative graphs useful for classification, and other redundant graphs whose characteristics are shared among all or most of patients. Therefore, the problem of classifying patients based on discriminative graphs boils down to learning graph embeddings and a selection mechanism based on the embeddings jointly.

In my master's thesis, we have already justified the architecture of a graph embedding module combing GraphSAGE[Hamilton et al. 2017] and DiffPool[Ying et al. 2018]. So the focus of this study will be to develop a differentiable graph sampling module which can be trained jointly with the embedding module. In fact, the process of sampling graphs can be viewed as the process of sampling from categorical distributions. Since categorical variables cannot be used in neural networks due to the inability to do backpropagation, we replace them with Gumbel-Softmax variables, whose samples are differentiable[Jang et al. 2016]. First, the sampling module learn probabilities of sampling each graph in the bag from original graph embeddings, then Gumbel-Softmax samples are generated using the probabilities, which can tell us how to do graph selection. Experimental results show that this algorithm not only achieves satisfactory classification accuracy, but can choose discriminative graphs as well, either in the scenario of single-graph selection or multi-graph selection. Our contribution is three-fold: 1) we present a framework for classifying multi-graphs with automatic graph selection, inspired from the characteristics of mIHC data; 2) The whole framework is end-to-end differentiable, so we can learn graph embedding and sampling jointly; 3) Our learning scheme can adapt to different scenarios, from single-graph selection to multi-graph selection.

The rest of the report is organized as follows. Section 2 summarizes the multi-graph classification model implemented in my master's thesis. Section 3 explains the architecture of the new model which enables automate graph selection, and the algorithm for jointly learning graph embedding and sampling. Section 4 introduces the datasets and discusses experimental results. Lastly, we conclude in Section 5.

2 Early Work

In my master's thesis, we proposed a multi-graph classification model to learn hierarchical cancerous patient representations in a self-supervised way, where patient IDs are served as class labels. A graph is expressed as $G = (\mathbf{A}, \mathbf{X})$, where $\mathbf{A}$ is the binary adjacency matrix, and $\mathbf{X} \in \mathbb{R} \in^{\mathbb{N}\times\mathbb{D}}$ is the node feature matrix. Given a set of multi-graph bags representing patients $\mathcal{S} = \{S_1, S_2, \cdots\} = \{\{G_1, G_2, \cdots\}, \{G_1, G_2, \cdots\}, \cdots\}$ and the corresponding labels $\mathbf{y}$, the model aims to learn a mapping $f : \mathcal{S} \rightarrow \mathbf{y}$ that maps multi-graph bags to their labels. Here, multi-graph bags are expected to only contain discriminative graphs. In other words, every graph is equally important for classification. Using a graph embedding module combining GraphSAGE and DiffPool, graph structure could be encoded into low-dimensional vectors. After that, representation of a patient will be generated by concatenating all graph embeddings in the bag, which will be used later for classification via a fully connected layer using softmax activation. This framework is schematically depicted in Figure 1. The forward pass of model takes the form below:

$$\hat{\mathbf{Y}} = f_c(\text{concat}(f_e(\mathcal{S}), dim = 1) \tag{1}$$

where f_e denotes the graph embedding module, f_c denotes the classification layer. While training, parameters are updated using stochastic gradient descent, and cross entropy loss is denoted in Equation 2 over the batch.

$$\ell_1 = -\sum_{b=1}^{B}\sum_{m=1}^{M} Y_{bm} \ln \hat{Y}_{bm} \tag{2}$$

where B is the batch size, M is the number of classes, and $\mathbf{Y}$ is class labels encoded in one-hot vectors.

Though this model is easy to train and has proven to be effective, it is needed to select distinctive graphs manually for each patient, which is tedious and time-consuming. In the next section, we will talk about improving the model so as to automate the process of graph selection.

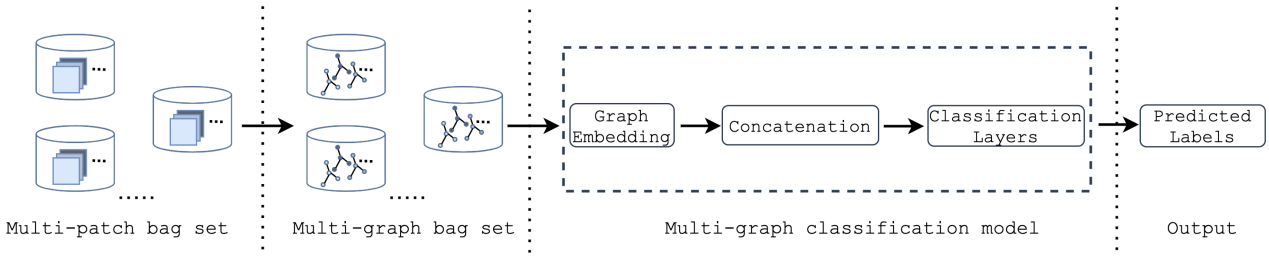


Figure 1: Self-supervised learning of multi-graph embeddings by imitating a classification task

3 Proposed Method

To address the challenging situation where each bag of graphs for classification is a mixture of discriminative graphs and redundant ones, we design a multi-graph classification model with discrete random variables that is capable of training the parameters of GNN layers and learning how to sample discriminative graphs simultaneously in an end-to-end fashion. Our proposed model, is mainly composed of three sub-blocks: a) a graph embedding module exploiting exactly the same structure as introduced in our early work, b) a newly added graph sampling module which can approximate sampling mechanism for categorical variables using the Gumbel-Softmax trick and thus be trainable using backpropogation, c) a fully connected classification layer. Figure 2 shows the schematic illustration of this model. In the rest of this section, we introduce the graph sampling module in detail first, as this part does not exist in our early model. Afterwards, we discuss about our joint learning algorithm under different scenarios, where the number of graphs for selection will differ.

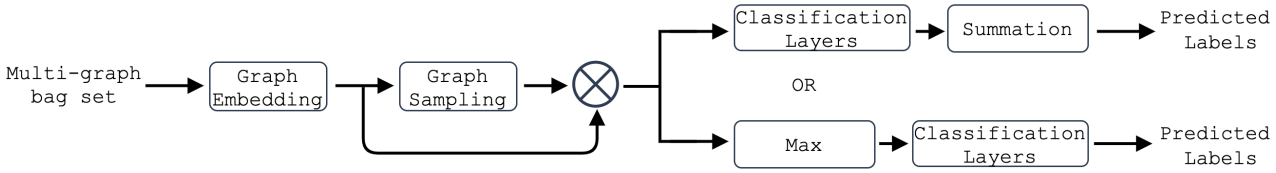


Figure 2: Multi-graph classification model with automatic graph selection

3.1 Graph Sampling with the aid of the Gumbel-Softmax estimator

As shown in Figure 2, the graph sampling module takes a bag of graph embeddings $\mathbf{E} \in \mathbb{R}^{K \times H}$ as input, where K is the number of graphs in the bag and H is the embedding dimension,

and outputs L K-dimensional Gumbel-Softmax samples $\mathbf{Z} \in \mathbb{R}^{L \times K}$, where L is the predefined number of discriminative graphs. Firstly, we adopt a multi-layer perceptron (MLP) to obtain unnormalized log probabilities $\mathbf{U} \in \mathbb{R}^{L \times K}$, which will serve as parameters of Gumbel-Softmax distribution. After that, L K-dimensional samples from categorical distribution with class probabilities

$$P_{ij} = \frac{e^{U_{ij}}}{\sum_{j=1}^{K} e^{U_{ij}}} \qquad \begin{array}{l} \text{for } j = 1, \ldots, K, \\ \qquad\qquad i = 1, \ldots, L \end{array} \tag{3}$$

can be approximated by L Gumbel-Softmax samples using the following formulation:

$$Z_{ij} = \frac{\exp\left(\left(U_{ij} + g_{ij}\right)/\tau\right)}{\sum_{j=1}^{k} \exp\left(\left(U_{ij} + g_{ij}\right)/\tau\right)} \quad \text{for } j = 1, \ldots, K, \quad i = 1, \ldots, L \tag{4}$$

where τ is the softmax temperature, and $g_{11} \ldots g_{LK}$ are i.i.d samples drawn from the $Gumbel(0, 1)$ distribution. During training, we start with a high temperature value where samples are uniform and anneal to a small temperature where samples are close one-hot vectors. By left multiplying graph embeddings $\mathbf{E}$ with Gumbel-Softmax samples $\mathbf{Z}$, we obtain new embeddings of the selected graphs $\mathbf{E}' \in \mathbb{R}^{L \times H}$ to be fed into the final classification layer. If using soft Gumbel-Softmax samples as derived in Equation 4, the new embedding of a selected graph is actually a weighted combination of all graph embeddings in the bag. Otherwise, we could utilize the Straight-Through (ST) Gumbel-Softmax estimator and discretize samples using $argmax$ as one-hot vectors, In this way, original embeddings of the selected graphs are directly used in $\mathbf{E}'$. In the backward pass, ST Gumbel-Softmax uses gradients obtained from the former relaxed version.

In short, by replacing non-differentiable categorical samples with differentiable gumbel-softmax samples, we could use backpropogation to compute gradients and automate the process of graph selection. The whole framework is end-to-end trainable.

3.2 Jointly learning graph embedding and graph sampling

Suppose that multi-graph bags not only contains discriminative graphs that are helpful for label assignment, but also similar redundant graphs. Hence, we propose a joint learning algorithm to classify multi-graph bags, as well as identify discriminative data in those bags. Though our framework is end-to-end differentiable, training is not that easy and straightforward, even in the simplest scenario where each bag only has one discriminative graph. When generalizing to multi-graph selection, graph embeddings in $\mathbf{E}'$ need to be aggregated in a permutation-invariant way, since the graph sampling order should not affect classification results.

3.2.1 Single-graph selection

Assume there is only one discriminative graph to be chosen and $L = 1$, $\mathbf{E}'$ actually becomes a H-dimensional vector, so the final fully connected layer from $\mathbb{R}^H$ to $\mathbb{R}^M$ can directly use it to predict the class label, where M is the number of classes. The simplest way is to set different learning rates for different sub-blocks of the model and train everything at one time

using cross-entropy loss with random initialization of parameters. However, this method has a high chance of failure due to the complexity of our architecture. Therefore, we have tried the following ways to improve model robustness and performance:

1. Use pre-training. Initially, omit the graph sampling module from our model. Seeing that the final classification layer requires a vector of dimension H as input, we could either adopt mean or max aggregator after obtaining K H-dimensional embeddings while pre-training. Once having pre-trained the graph embedding module and the classification layer, start training the whole model with a larger learning rate for the graph sampling module and a smaller one for the rest.

2. Combine supervised cross-entropy loss ℓ_1 for measuring the quality of classification results with other types of unsupervised auxiliary losses. According to our initial assumption, there exist similar graphs in different bags, which are irrelevant to classification and thus should not be sampled. Since parameters will be updated using stochastic gradient descent, we could define a batch-wise loss ℓ_2 as shown below to minimize the average Euclidean distance between every two unselected graphs:

$$\ell_2 = \frac{1}{B_u^2} \sum_{i=1}^{B_u} \sum_{j=1}^{B_u} d\left(\mathbf{e}_i, \mathbf{e}_j\right) \tag{5}$$

   where B_u is the number of discarded graphs in a batch of training data, $\mathbf{e_i}$ and $\mathbf{e_j}$ are two graph embeddings lying in $\mathbf{E}$ but not chosen by $\mathbf{Z}$. The average distance is minimal when all the noisy graphs in the batch are discarded and all the discriminative graphs are kept.

3. Lastly, it should be noticed that when feeding data to the model, we should shuffle graphs in each multi-graph bag at every iteration. Otherwise, the graph sampling module would memorize positions of discriminative graphs, instead of learning a sampling mechanism based on graph embeddings.

Experimental results shown in Section 4.3.1 could help us understand the role of pre-training and auxiliary loss. Our joint learning algorithm is summarized in Algorithm 1. When it comes to single-graph selection, just close auxiliary loss ℓ_3 by setting $\alpha_3 = 0$.

3.2.2 Multi-graph selection

Single-graph selection can be viewed as a special case of multi-graph selection. While keeping the existing settings, we have tried improving the algorithm from the following aspects to adapt to this more sophisticated scenario:

1. Aggregate embeddings of sampled graphs in a permutation-invariant way. It is obvious that changing sampling order or rearranging rows of matrix $\mathbf{E}'$ should not affect model output. For simplicity, a max aggregator could be used before the classification layer. Another approach is that, we feed sampled graphs to a fully connected layer from $\mathbb{R}^H$ to $\mathbb{R}^M$ first, then use a summation operator followed by a softmax activation operator to predict the class label. The main difference between these two approaches is where to apply the aggregator, either before or after the fully connected layer.

2. Avoid sampling the same graph multiple times. We make use of inner product to access the similarity of two Gumbel-Softmax samples. By minimizing the batch-wise auxiliary loss designed in Equation 6, we could encourage the model to sample different graphs in a multi-graph bag.

$$\ell_3 = \frac{1}{B}\sum_{b=1}^{B}\left(\frac{1}{L^2}\sum_{i=1}^{L}\sum_{j=1}^{L}\mathbf{z}_{bi}\cdot\mathbf{z}_{bj}\right) \tag{6}$$

where B is the batch size, $\mathbf{z}_{bi}$ and $\mathbf{z}_{bj}$ are corresponding Gumbel-Softmax samples.

In Section 4.3.2, we will compare these two aggregation methods as well as discuss the necessity of adding the auxiliary loss mentioned above.

Algorithm 1: Jointly learning embedding and sampling for multi-graph classification

Input : Multi-graph bag set $\mathcal{S} = \{S_1, S_2, \cdots\}$; batch size B, the number of graphs in each bag K; the number of discriminative graphs in each bag L; the number of classes M; weights for different losses $\alpha_1, \alpha_2, \alpha_3$; graph embedding module f_e graph sampling module f_s final classification layer f_c

Output: Predicted class labels $\mathbf{y} \in \mathbb{R}^B$

`// Pre-training Phase`

$\hat{\mathbf{Y}} = f_c(\text{max or mean}(f_e(\mathcal{S}), dim = 1)$

backpropogating with cross-entropy loss ℓ_1

`// Joint Training Phase`

$\mathbf{E} = f_e(\mathcal{S}) \in \mathbb{R}^{\mathbb{B}\times\mathbb{K}\times\mathbb{H}} \rightarrow$ graph embedding

$\mathbf{Z} = f_z(\mathbf{E}) \in \mathbb{R}^{\mathbb{B}\times\mathbb{L}\times\mathbb{K}} \rightarrow$ Gumbel-Softmax samples

$\mathbf{E}' = bmm(\mathbf{Z}, \mathbf{E}) \in \mathbb{R}^{\mathbb{B}\times\mathbb{L}\times\mathbb{H}} \rightarrow$ batch matrix multiplication

if *aggregating before* f_c **then**

  $\hat{\mathbf{Y}} = f_c(max(\mathbf{E}', dim = 1)) \in \mathbb{R}^{\mathbb{B}\times\mathbb{M}}$

else

  $\hat{\mathbf{Y}} = sum(f_c(\mathbf{E}'), dim = 1) \in \mathbb{R}^{\mathbb{B}\times\mathbb{M}}$

$loss = \alpha_1\ell_1 + \alpha_2\ell_2 + \alpha_3\ell_3 \rightarrow$ do backpropogation

4 Experimental Results

In this section, we evaluate different training methods of our joint learning algorithm on two synthetic datasets corresponding to the scenario of single-graph selection and multi-graph selection respectively. Experimental results show that the proposed algorithm has the potential of automating graph selection in multi-graph classification tasks if trained properly.

Experimental set-up. Table 1 shows some common hyperparameter settings. There are 10 different multi-graph bags in total, each containing 5 graphs, so the number of classes is 10. Setting batch size = 50 means that parameters are updated using batch gradient descent. The graph embedding module stacks two layers of GraphSAGE and two layers of DiffPool, and the graph sampling module is a three-layer MLP. Lastly, a fully connected layer from $\mathbb{R}^{64}$ to $\mathbb{R}^{10}$ is implemented for predicting the label.

batch size	# of classes	# of graphs	hidden dimension	# of layers
50	10	5	64	7

Table 1: Common hyperparameter settings

4.1 Dataset

We creat two kinds of datasets to meet the needs of different scenarios. Dataset A is designed for single-graph selection, while dataset B is designed for multi-graph selection. Table 2 show their data statistics.

	avg nodes	avg edges	node categories	classes	graphs in each bag	discriminative graphs in each bag	bags
dataset A	250	902	5	10	5	1	10
dataset B	250	908	5	10	5	2	10

Table 2: Data statistics

In total, there are 10 discriminative graphs and 40 similar graphs in dataset A. Each bag formed by 1 discriminative graph and 4 redundant graphs, and there are 10 bags. Figure 3 visualizes the graphs in one of the bags as an example.

There are 10 discriminative graphs generated from 5 kinds of distributions, and 4 similar graphs in dataset B. Each bag picks 2 discriminative graphs from different distributions and 3 similar graphs, there are also 10 bags. Figure 4 gives an example of dataset B.

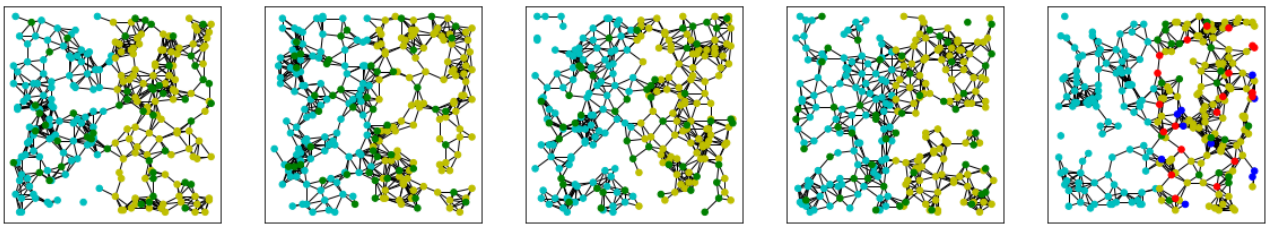


Figure 3: Visualization of a bag in dataset A

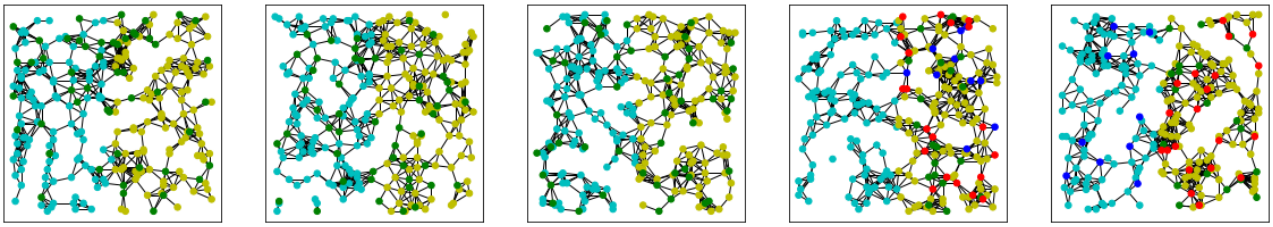


Figure 4: Visualization of a bag in dataset B

4.2 Pre-training

In the pre-training phase, parameters of the embedding module and the classification layer are updated with a learning rate of 1e-2. Validation data is generated using the same metrics as

training data. We plot loss and accuracy curves to infer where to stop training, and exhibits confusion matrices and softmax model outputs at the end point. Results for both datasets using a max graph aggregator are shown from Figure 5 to 8. We can reach 100% accuracy on both training and validation data when there is only one discriminative graph for each patient as shown in Figure 5 and Figure 6. When there are two graphs to be selected, accuracy on the validation data drops to 80% as shown in Figure 7 and Figure 8. The results are similar for the mean aggregator.

4.3 Joint training

When performing end-to-end training, parameters of the sampling module are updated with a larger learning rate of 1e-4, while the rest are updated with a smaller learning rate of 1e-6. Graphs inside each bag are shuffled at every epoch. For simplicity, validation data is set to be the same as training data, since this study mainly focuses on justifying the design of our joint learning algorithm. For those successful experiments, confusion matrices and softmax model outputs of validation data are plotted to analyze classification results. We also visualize Gumbel-Softmax samples and related unnormalized log probabilities to measure the quality of the sampling module.

4.3.1 Single-graph Selection

Table 3 summarizes the experiments done on dataset A, in the scenario of single-graph selection. For all these experiments, the temperature is annealed using the schedule $\tau = \max(0.5, 10\exp(-0.01t))$, where t is the number of iterations.

Initially, we train our model from scratch using the hard Gumbel-Softmax estimator in Experiment 1 and the soft Gumbel-Softmax estimator in Experiment 2. However, from Figure 9 and Figure 10 we can see that the model doesn't learn. Training loss only drops a little, and training accuracy doesn't improve at all. As our model combines embedding module and sampling module, it's too ambitious to train them together with random initialization of weights.

Next, let's see if pre-training is helpful, and which graph aggregator has better performance. If using a mean aggregator in the pre-training phase, and the soft Gumbel-Softmax estimator in the joint training phase, training loss and accuracy start from a good point and then become worse as the number of epochs increases (see Figure 11). This is because soft Gumbel-Softmax samples are nearly uniform with a high initial temperature, and by multiplying those soft samples with graph embeddings, it's like we are averaging them. When changing to the hard estimator, training loss goes down and training accuracy goes up when increasing the number of epochs as shown in Figure 12. It's tolerable that loss and accuracy curves are not smooth owing to the existence of stochastic layers. When using a max aggregator, the model can be trained successfully with both soft and hard estimators, shown in Figure 15 to 20. During validation, classification accuracy and sampling accuracy are the highest when using the max aggregator together with the soft estimator as recorded in table 3. Hence, the max aggregator is adopted for all the following experiments.

Afterwards, we want to investigate if adding auxiliary loss ℓ_2 will further improve model performance. When training with ℓ_2 alone, the model still has acceptable classification performance, and the sampling accuracy can reach 60% in the validation phase (see Table 3, Experiment 7). When combining ℓ_1 and ℓ_2 in Experiment 8, the performance is better than

that of Experiment 7 but worse than that of Experiment 6. Adding auxiliary loss ℓ_2 will harm model performance, though it is effective to training with it alone.

In general, pre-training is the key to the success of subsequent joint training, and the max aggregator should be adopted to ensure that both soft and hard estimator can work later. We should train with only ℓ_1 without adding auxiliary loss ℓ_2. Under the same condition, the soft estimator often gives better results , as we generate a new embedding based on a weighted sum of all candidate graph embeddings, which is more reliable than relying on one graph embedding. The best training strategy for single-graph selection is the same as that of Experiment 6.

No.	aggregator in pre-training	Gumbel-Softmax samples in training	weight for ℓ_1	weight for ℓ_2	classification acc		sampling acc
					soft	hard	
1	\	soft	1	0	\	\	\
2	\	hard	1	0	\	\	\
3	mean	soft	1	0	\	\	\
4	mean	hard	1	0	90%	90%	80%
5	max	soft	1	0	100%	100%	90%
6	max	hard	1	0	90%	90%	80%
7	max	hard	0	1	80%	70%	60%
8	max	hard	1	1	90%	80%	70%

Table 3: Experimental settings and results on dataset A

4.3.2 Multi-graph Selection

Table 4 summarizes the experiments done on dataset B, in the scenario of multi-graph selection. We use a smaller annealing rate and train more epochs this time, since this scenario is more complex. The annealing schedule is written as: $\tau = \max(0.5, 10\exp(-0.001t))$.

We want to figure out which way to combine selected graphs is better in the joint training phase, a max aggregator before the classification layer, or a summation aggregator after it. The max aggregator only works with a soft estimator, as shown in Figure 33 to 36, while the summation aggregator works with both estimators as shown in Figure 27 to 32. Yet, sampling accuracy is higher when using max aggregator (see Table 4).

Sampling inaccuracies mainly come from choosing the same graph twice, so we combine auxiliary loss ℓ_3 with ℓ_1 to see if it can help. However, as shown in Figure 37, adding ℓ_3 actually deteriorates the performance when using the hard estimator. If forcing the model to choose different graphs in one bag, we sacrifice classification accuracy a lot.

Overall, in the scenario of multi-graph selection, how to aggregate graphs has a crucial influence on the experimental results. Additionally, auxiliary loss increases the difficulty of training thus leading to poor results. Soft Gumbel-Softmax estimator in training works better than the hard one in terms of classification accuracy, but worse in terms of sampling accuracy (see Table 4). Lastly, if we want to infer discriminative graphs in each bag, sampling probabilities for each class is more reliable than Gumbel-Softmax samples, as they are stochastic.

No.	aggregator in pre-training	Gumbel-Softmax samples in training	weight for ℓ_1	weight for ℓ_3	aggregator in joint training	classification acc		sampling acc
						soft validation	hard validation	
9	max	hard	1	0	summation	90%	80%	80%
10	max	soft	1	0	summation	100%	70%	70%
11	max	hard	1	0	max	\	\	\
12	max	soft	1	0	max	100%	70%	85%
13	max	hard	1	1	summation	\	\	\

Table 4: Experimental settings and results on dataset B

5 Conclusion

In this study, we introduce stochastic discrete layers into graph neural networks to do multi-graph classification by jointly learning graph embedding and sampling. Either in the scenario of single-graph selection or multi-graph selection, our proposed algorithm shows satisfactory classification and sampling accuracy. From experimental results, we can see that pre-training is necessary, while adding auxiliary losses will confuse our model. When selecting multiple graphs, training with the hard Gumbel-Softmax estimator will result in higher sampling accuracy, while training with the soft estimator will result in higher classification accuracy. The choice of graph aggregator in the joint training phase directly affects the quality of classification results and sampling results. For future work, we could figure out other permutation-invariant methods to aggregate graphs, in order to further reduce information loss.

References

[Hamilton et al. 2017] Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034.

[Jang et al. 2016] Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

[Lu et al. 2018] Lu, C., Wang, X., Prasanna, P., Corredor, G., Sedor, G., Bera, K., Velcheti, V., and Madabhushi, A. (2018). Feature driven local cell graph (fedeg): Predicting overall survival in early stage lung cancer. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 407–416. Springer.

[Swiderska-Chadaj et al. 2018] Swiderska-Chadaj, Z., Pinckaers, H., van Rijthoven, M., Balkenhol, M., Melnikova, M., Geessink, O., Manson, Q., Litjens, G., van der Laak, J., and Ciompi, F. (2018). Convolutional neural networks for lymphocyte deftection in immunohistochemically stained whole-slide images.

[Ying et al. 2018] Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. (2018). Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems*, pages 4800–4810.

Appendices

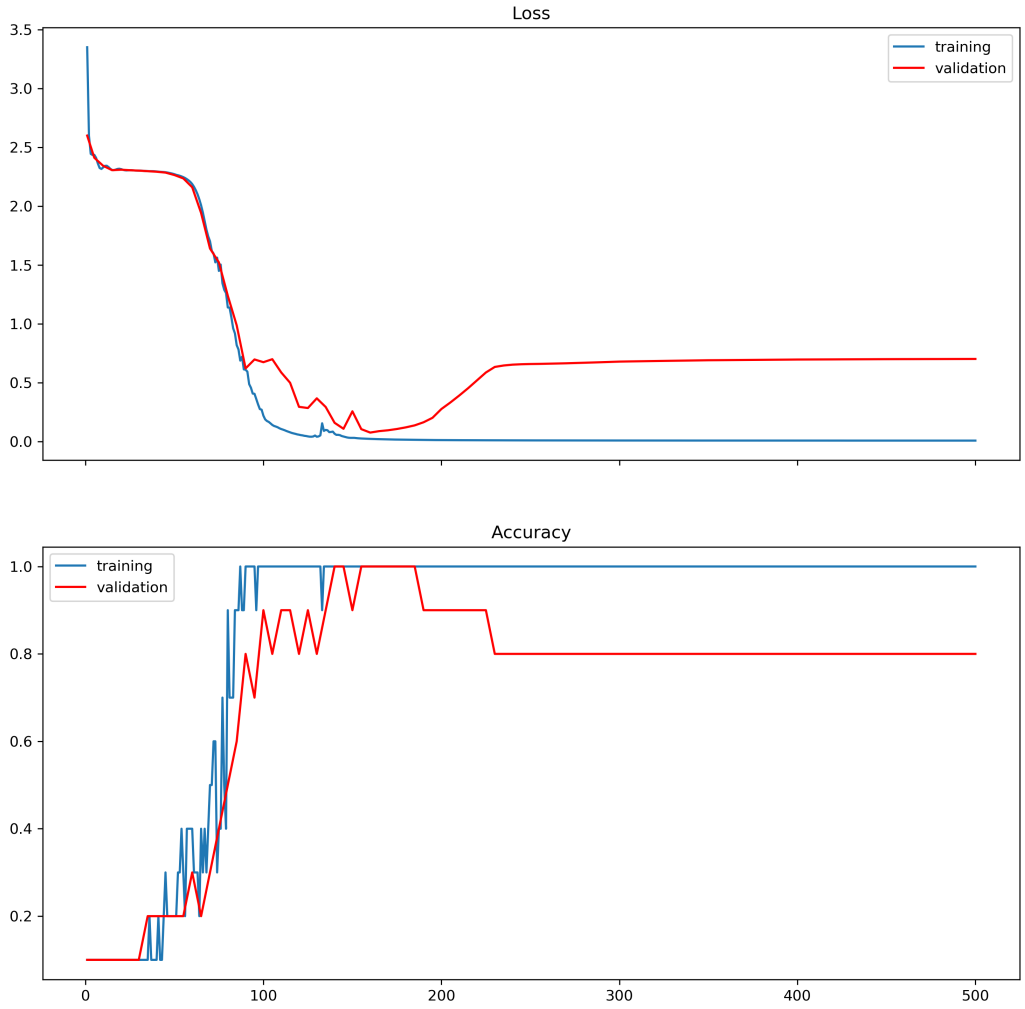


Figure 5: Pre-training, dataset A, max aggregator: loss/accuracy vs. epochs

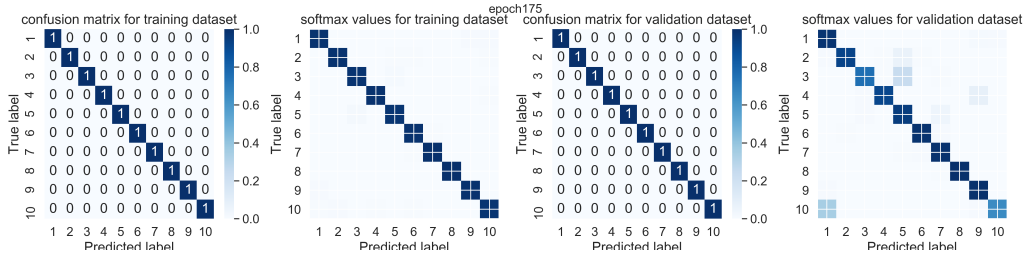


Figure 6: Pre-training, dataset B, max aggregator: loss/accuracy vs. epochs

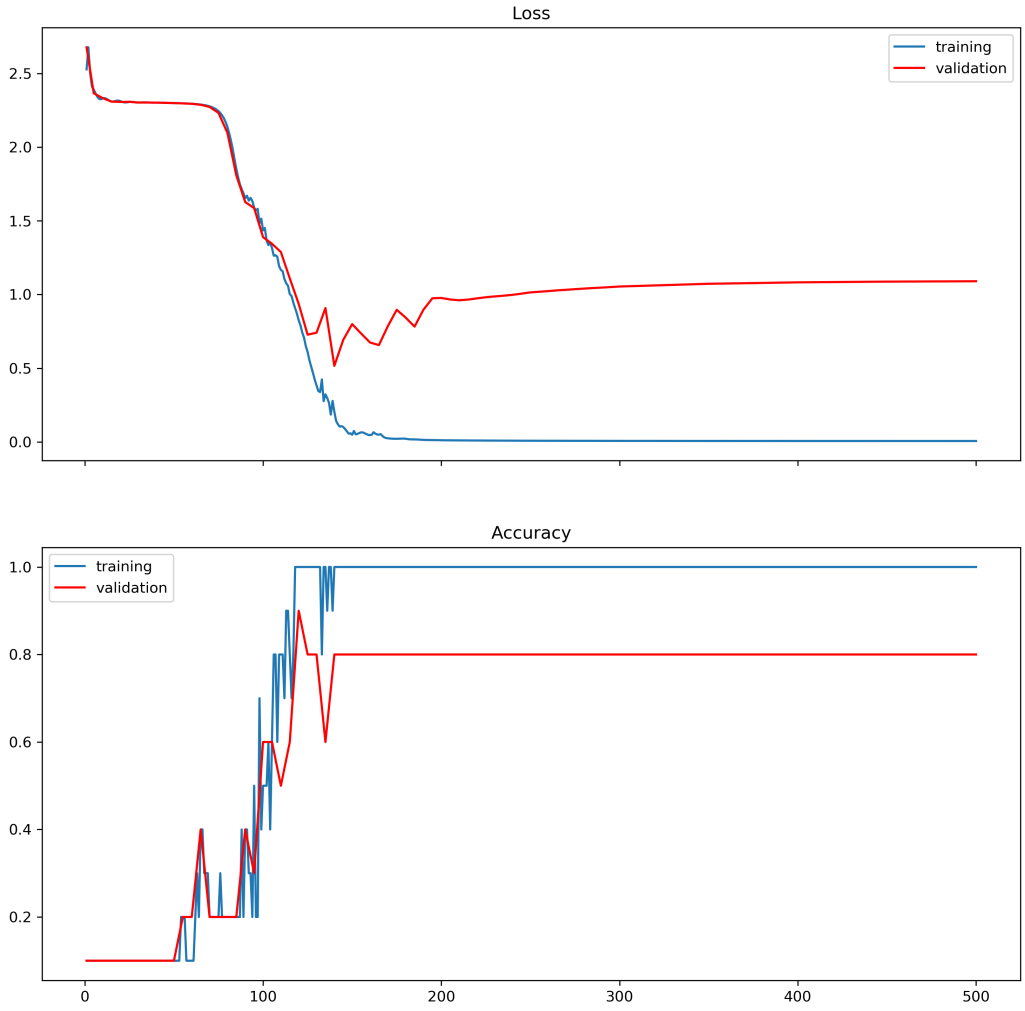


Figure 7: Pre-training, dataset B, max aggregator: loss/accuracy vs. epochs

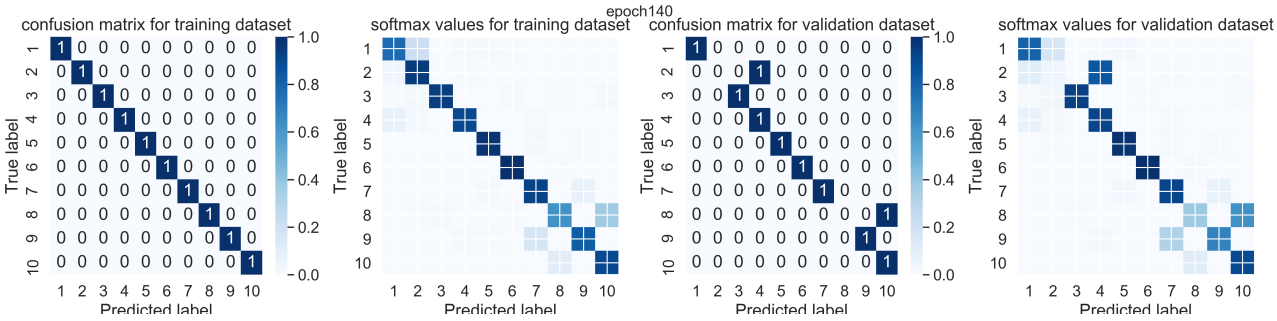


Figure 8: Pre-training, Dataset B: confusion matrices and heatmaps of softmax model outputs at epoch 140

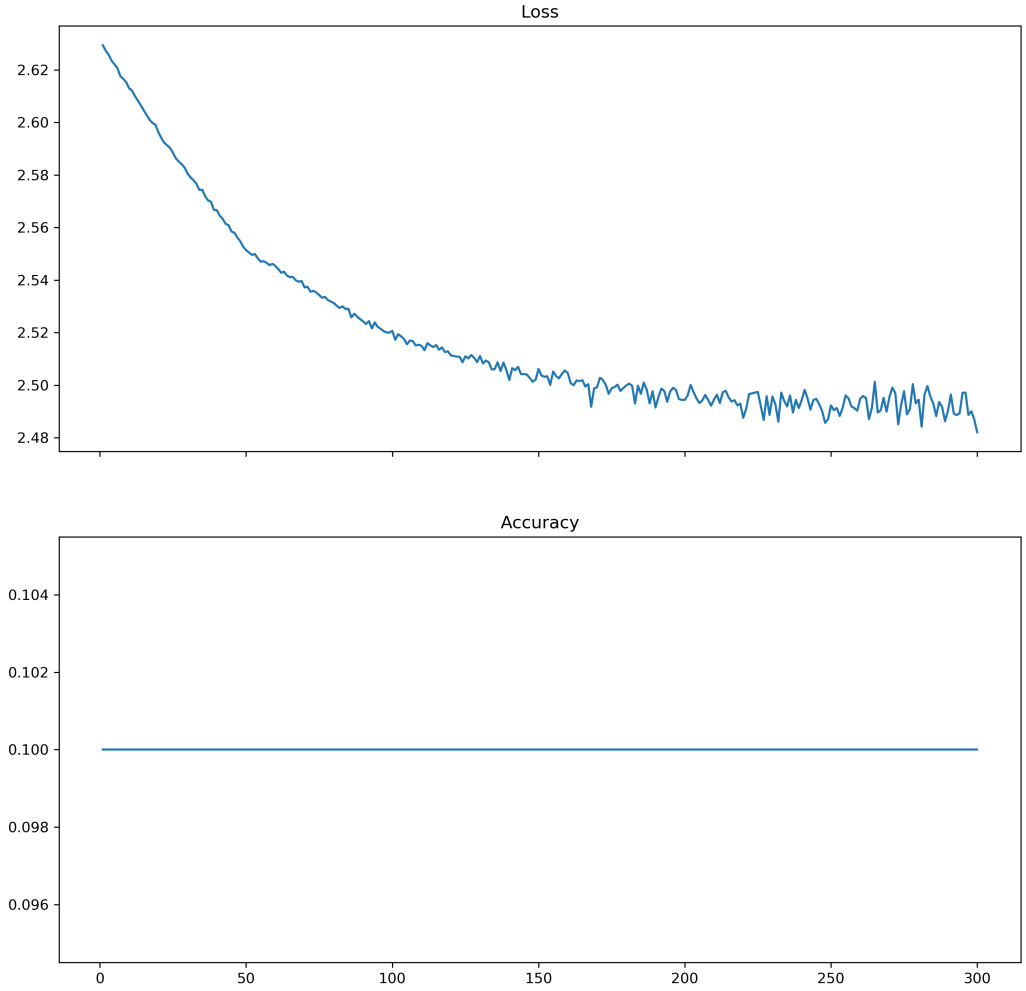


Figure 9: Experiment 1: loss/accuracy vs. epochs

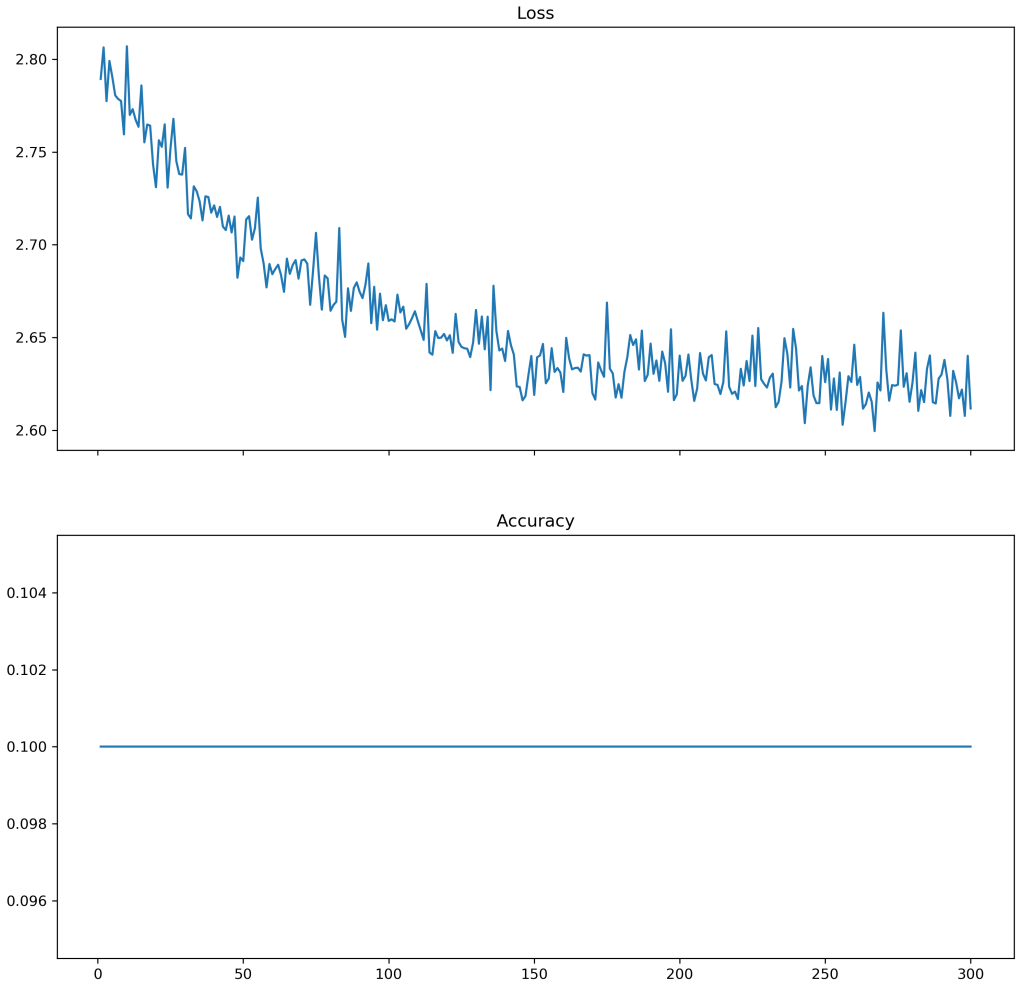


Figure 10: Experiment 2: loss/accuracy vs. epochs

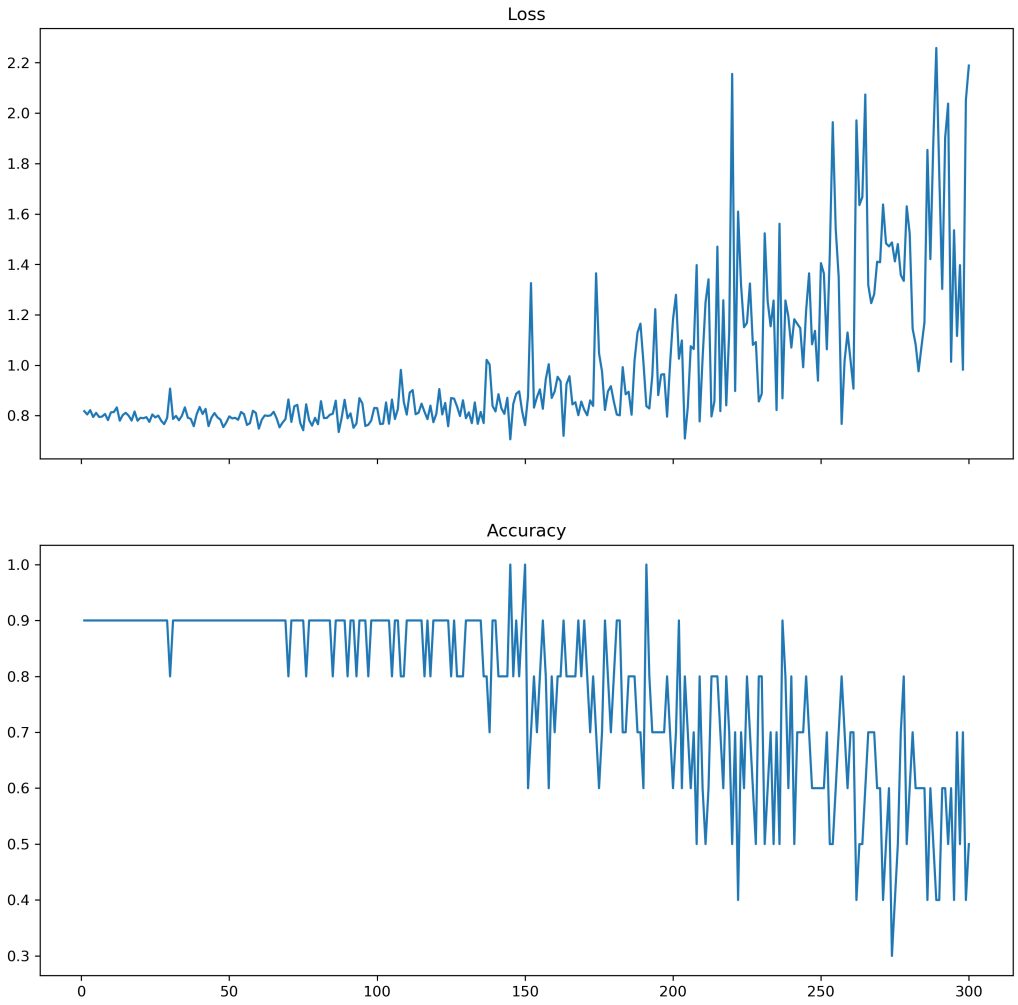


Figure 11: Experiment 3: loss/accuracy vs. epochs

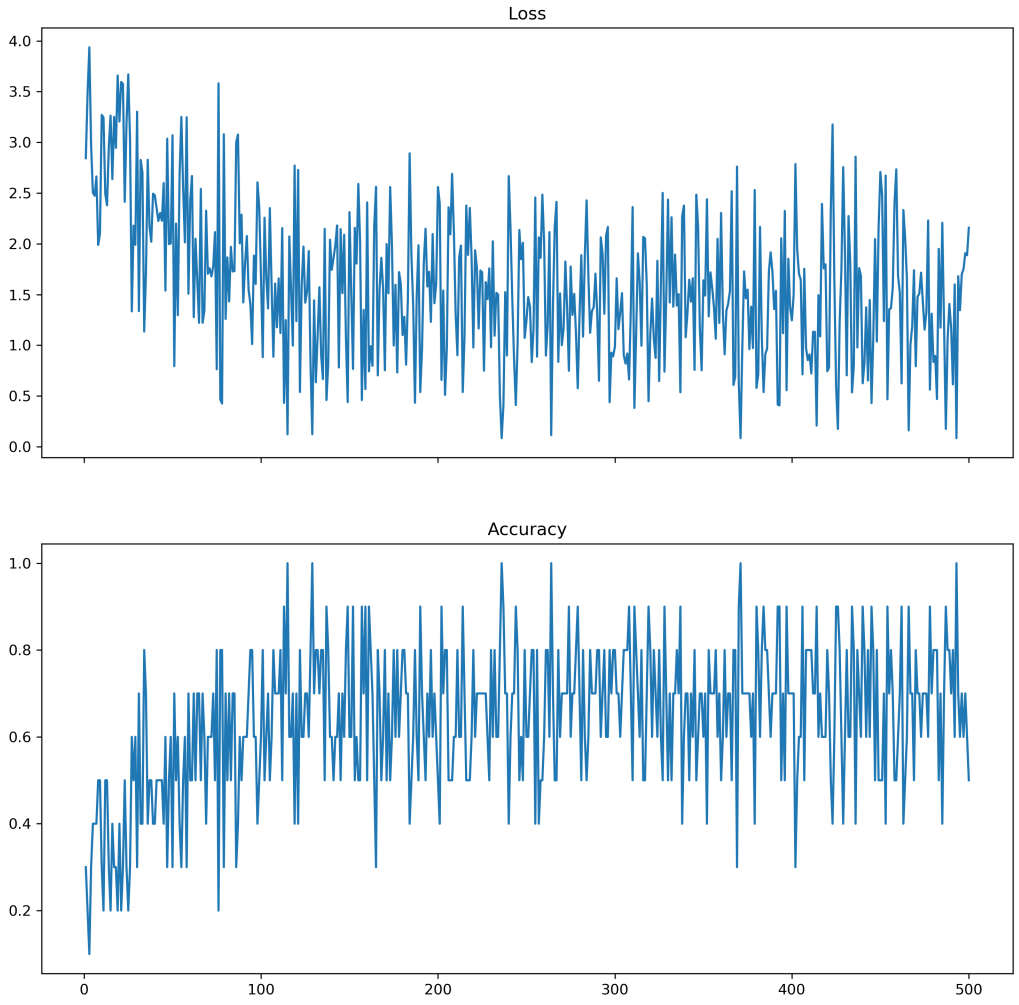


Figure 12: Experiment 4: loss/accuracy vs. epochs

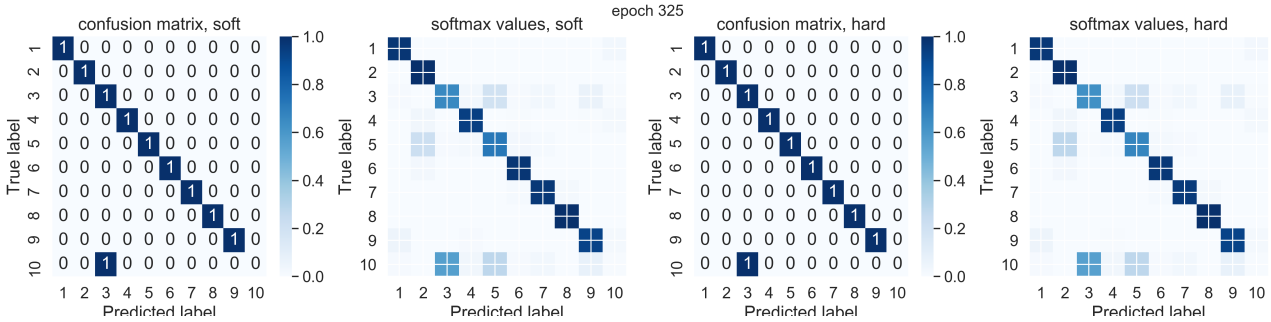
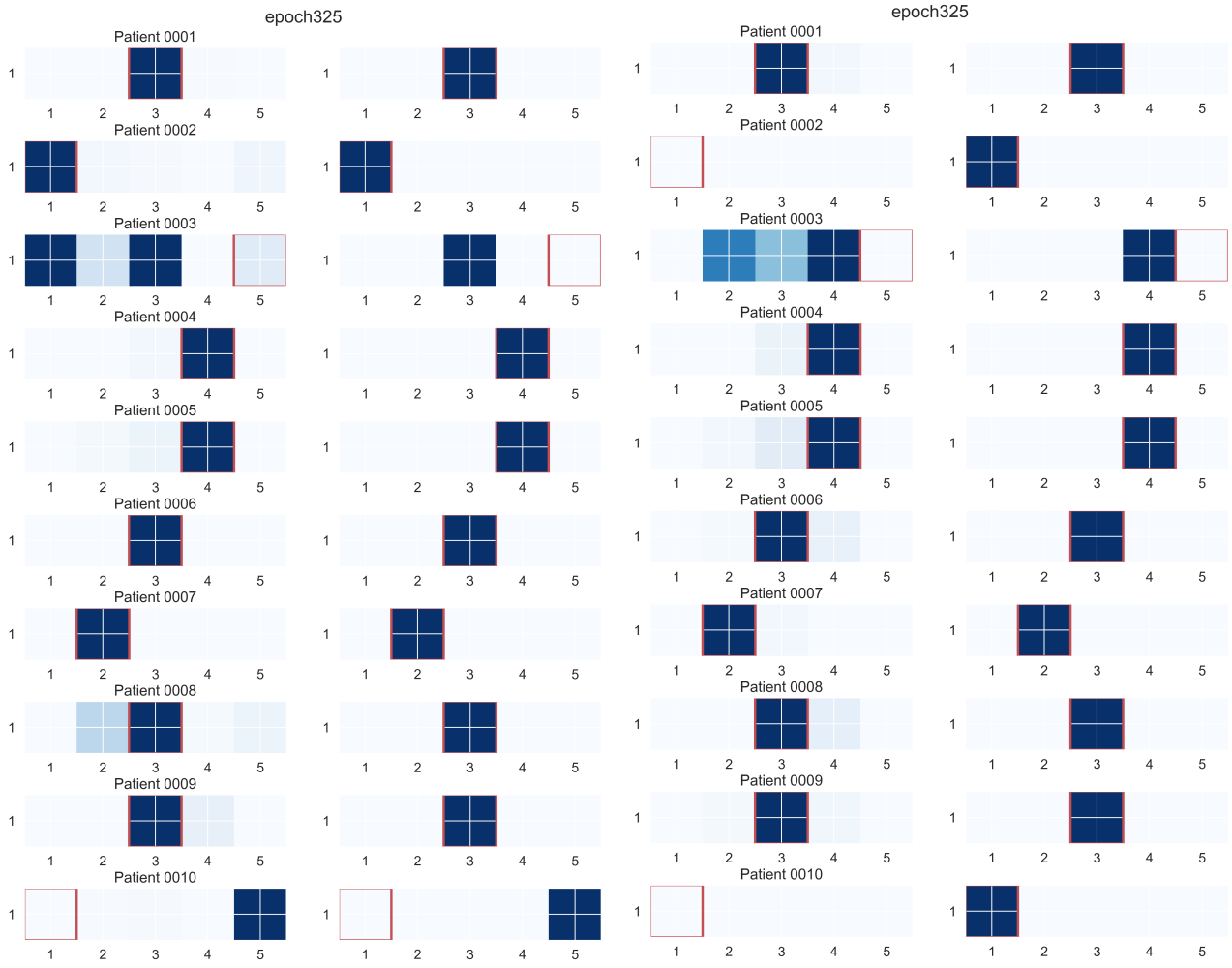


Figure 13: Experiment 4: confusion matrices and heatmaps of softmax model outputs at epoch 325



(a) Gumbel-Softmax samples (b) Unnormalized log probabilities

Figure 14: Experiment 4, epoch 325, left: soft version right: hard version

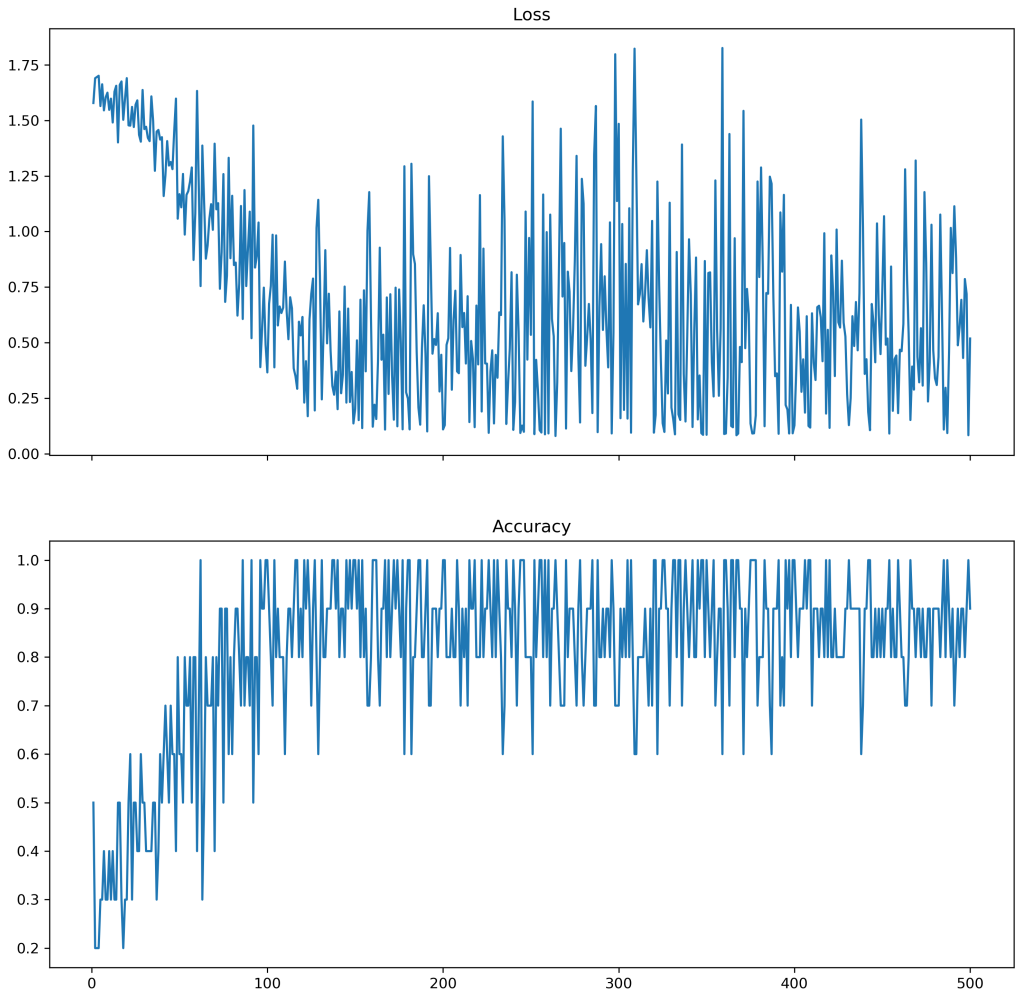


Figure 15: Experiment 5: loss/accuracy vs. epochs

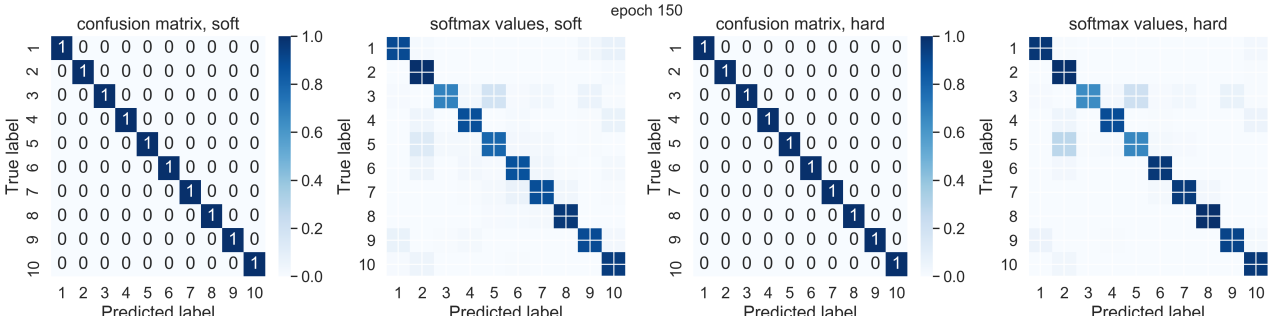


Figure 16: Experiment 5: confusion matrices and heatmaps of softmax model outputs at epoch 150

(a) Gumbel-Softmax samples

(b) Unnormalized log probabilities

Figure 17: Experiment 5, epoch 150, left: soft version right: hard version

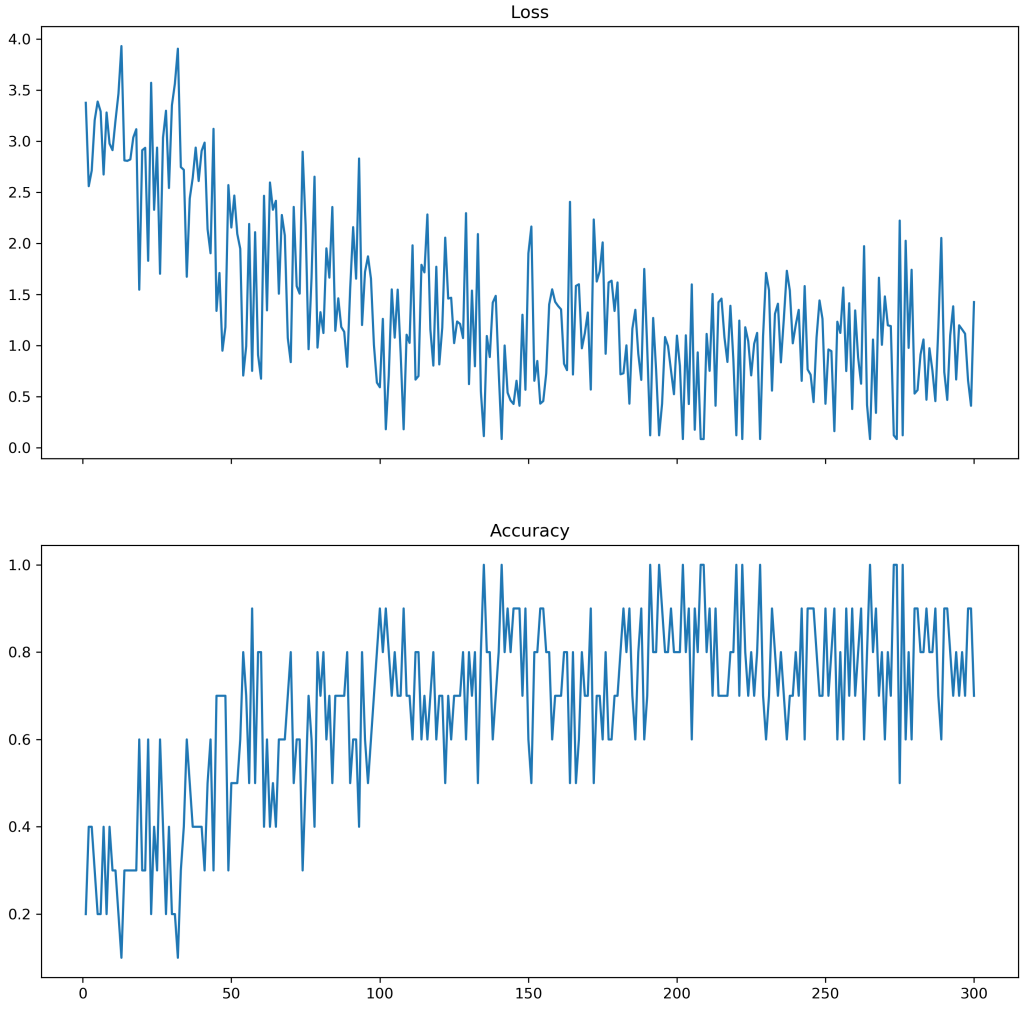


Figure 18: Experiment 6: loss/accuracy vs. epochs

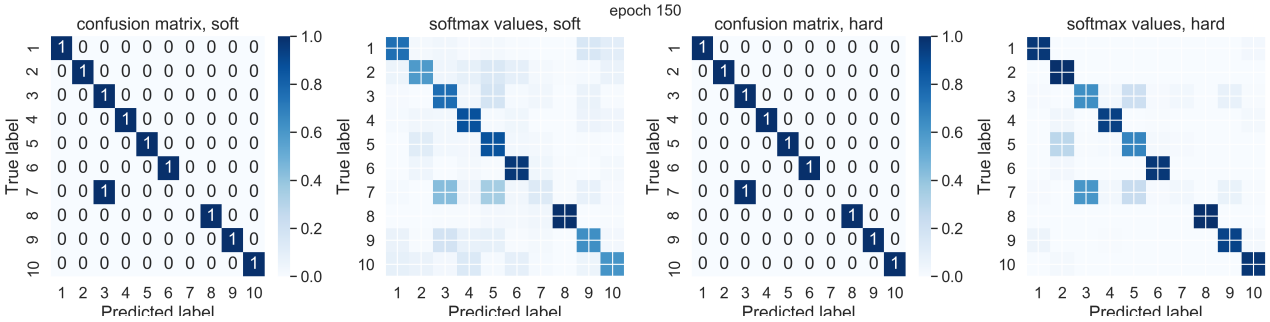
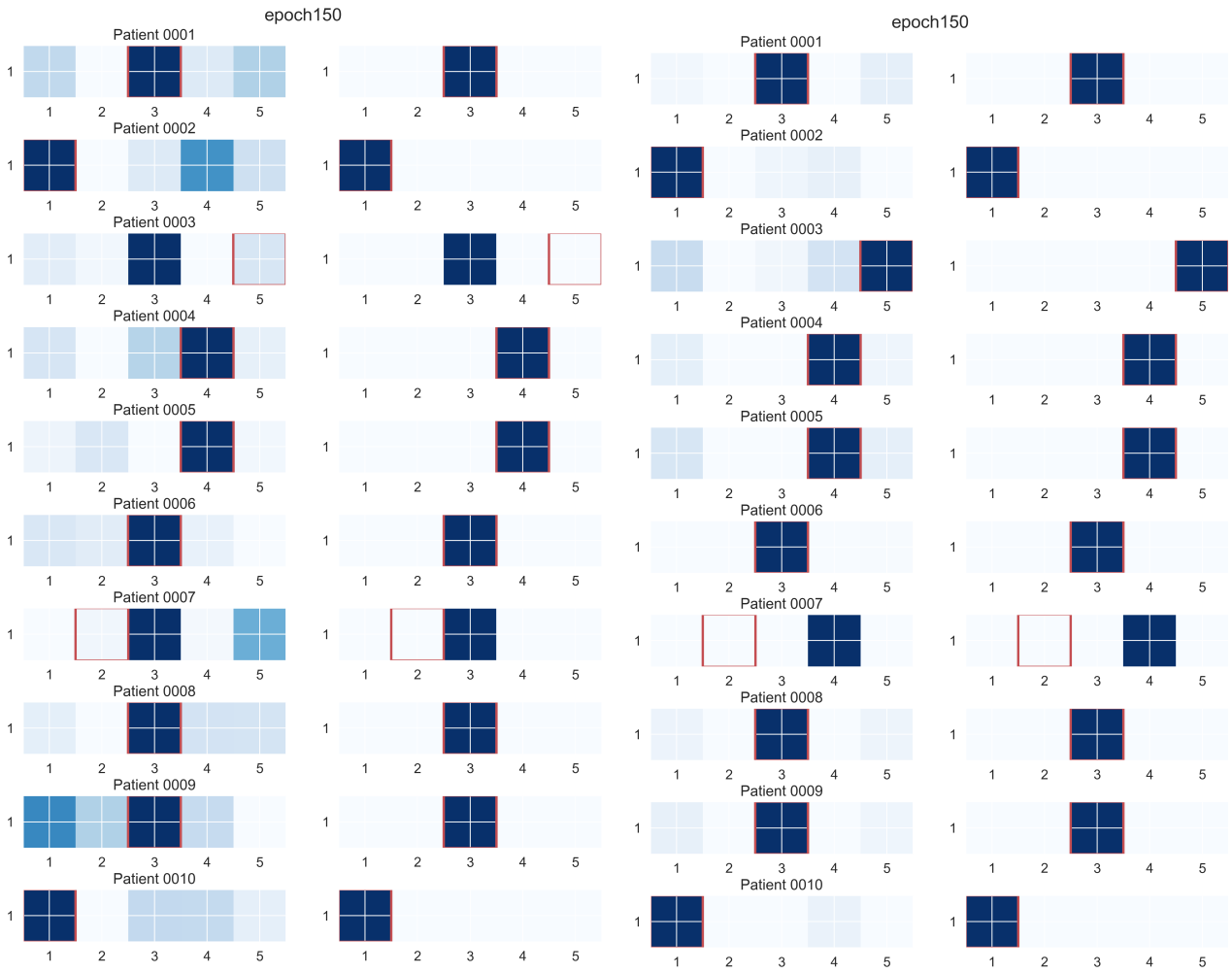


Figure 19: Experiment 6: confusion matrices and heatmaps of softmax model outputs at epoch 150



(a) Gumbel-Softmax samples

(b) Unnormalized log probabilities

Figure 20: Experiment 6, epoch 150, left: soft version right: hard version

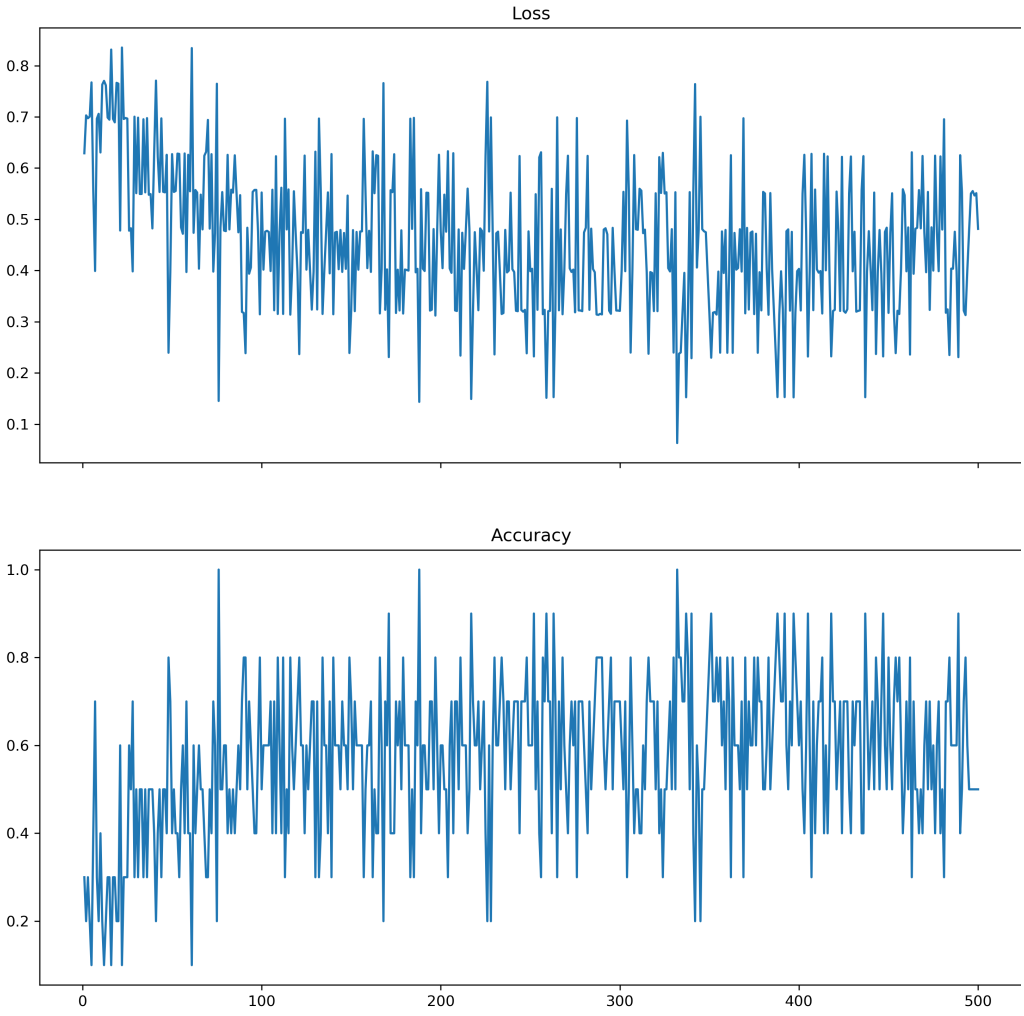


Figure 21: Experiment 7: loss/accuracy vs. epochs

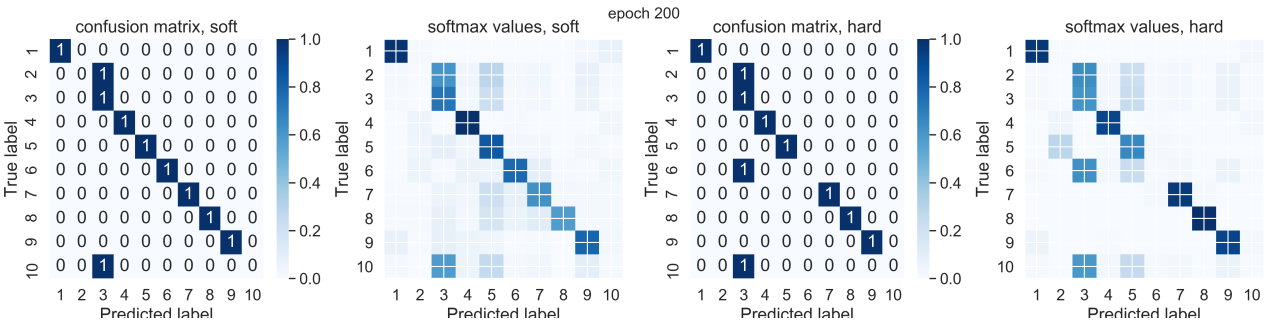
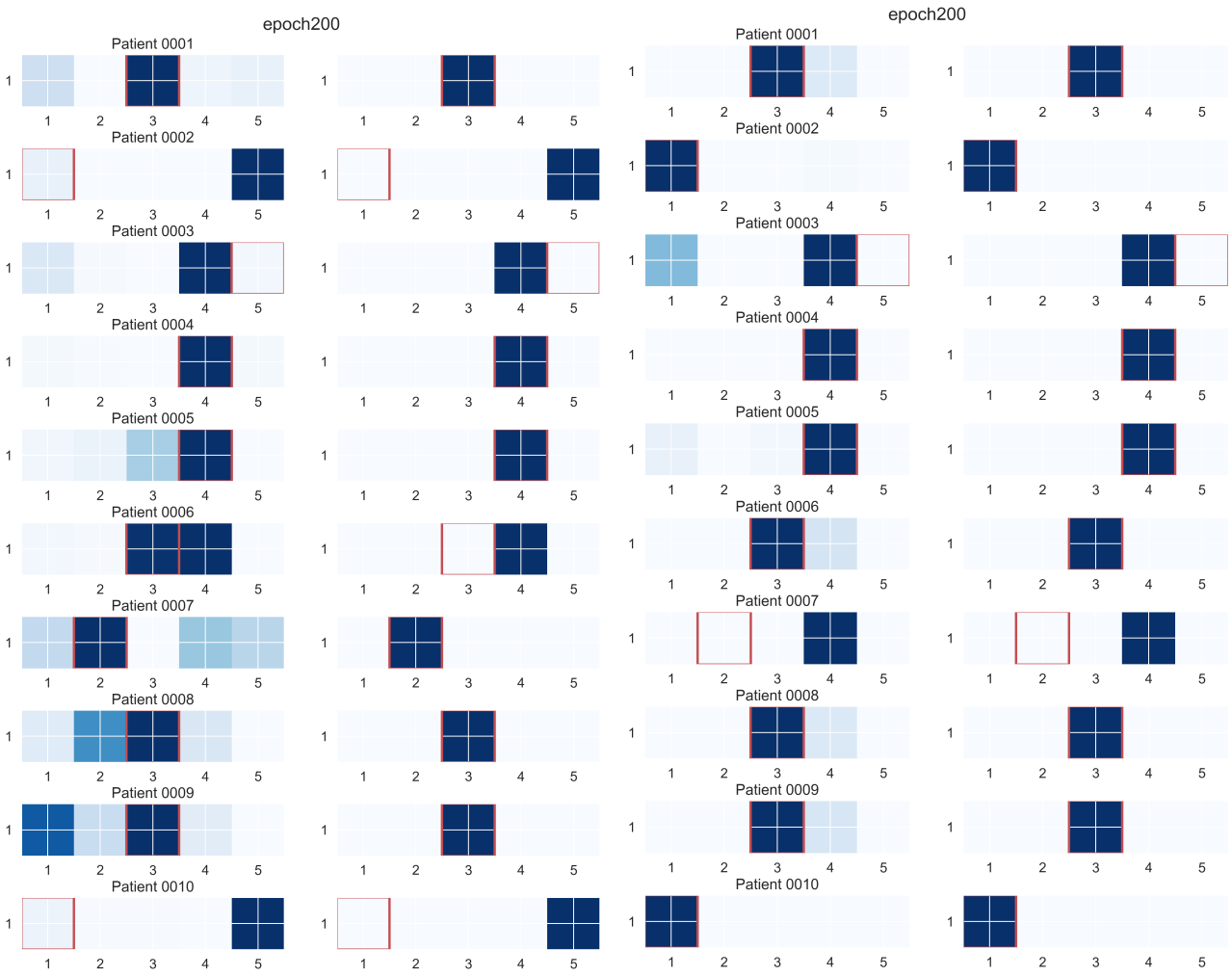


Figure 22: Experiment 7: confusion matrices and heatmaps of softmax model outputs at epoch 200



(a) Gumbel-Softmax samples (b) Unnormalized log

Figure 23: Experiment 7, epoch 200, left: soft version right: hard version

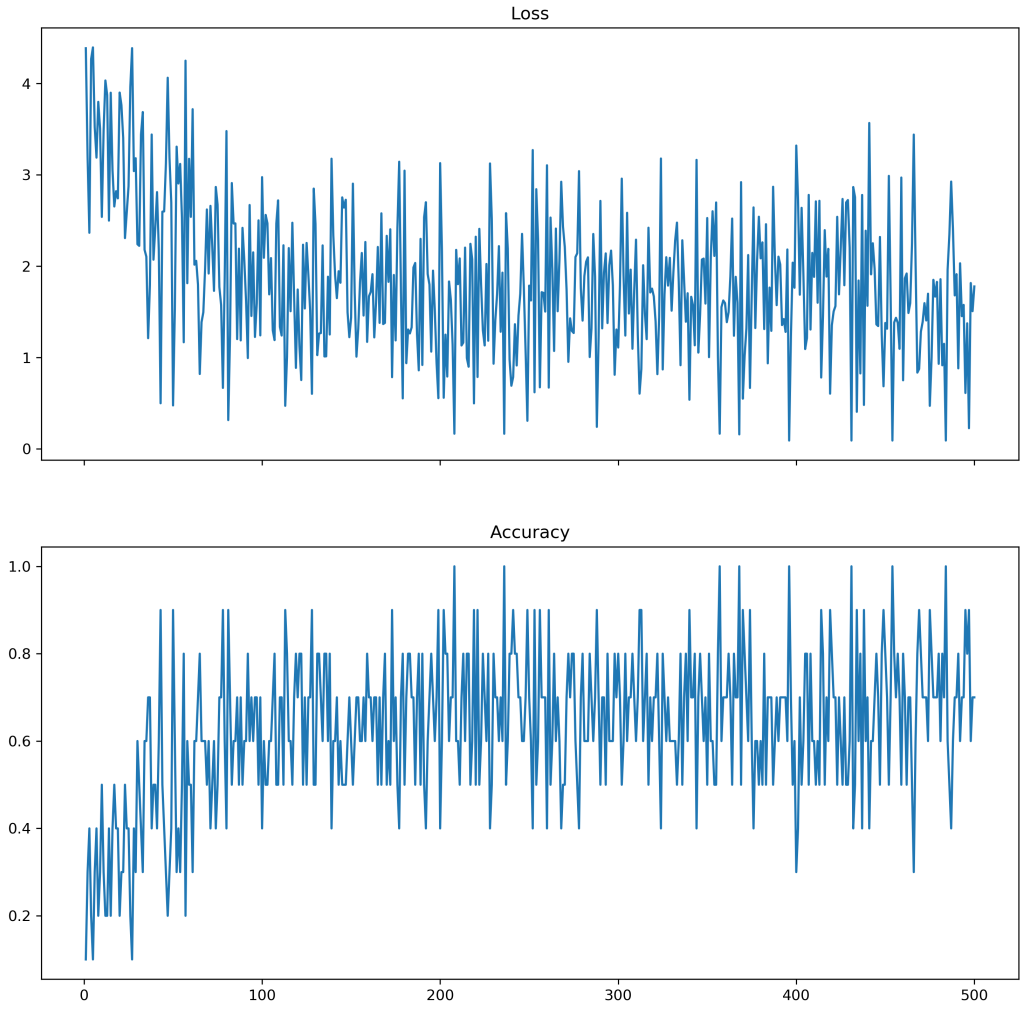


Figure 24: Experiment 8: loss/accuracy vs. epochs

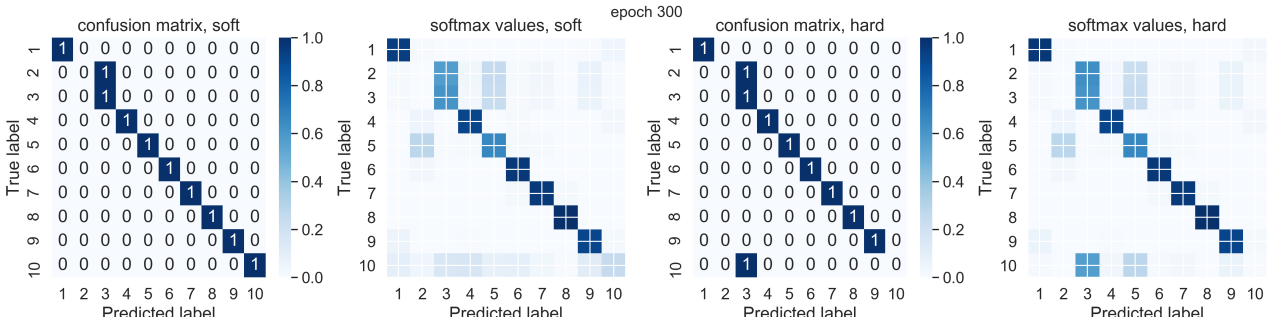
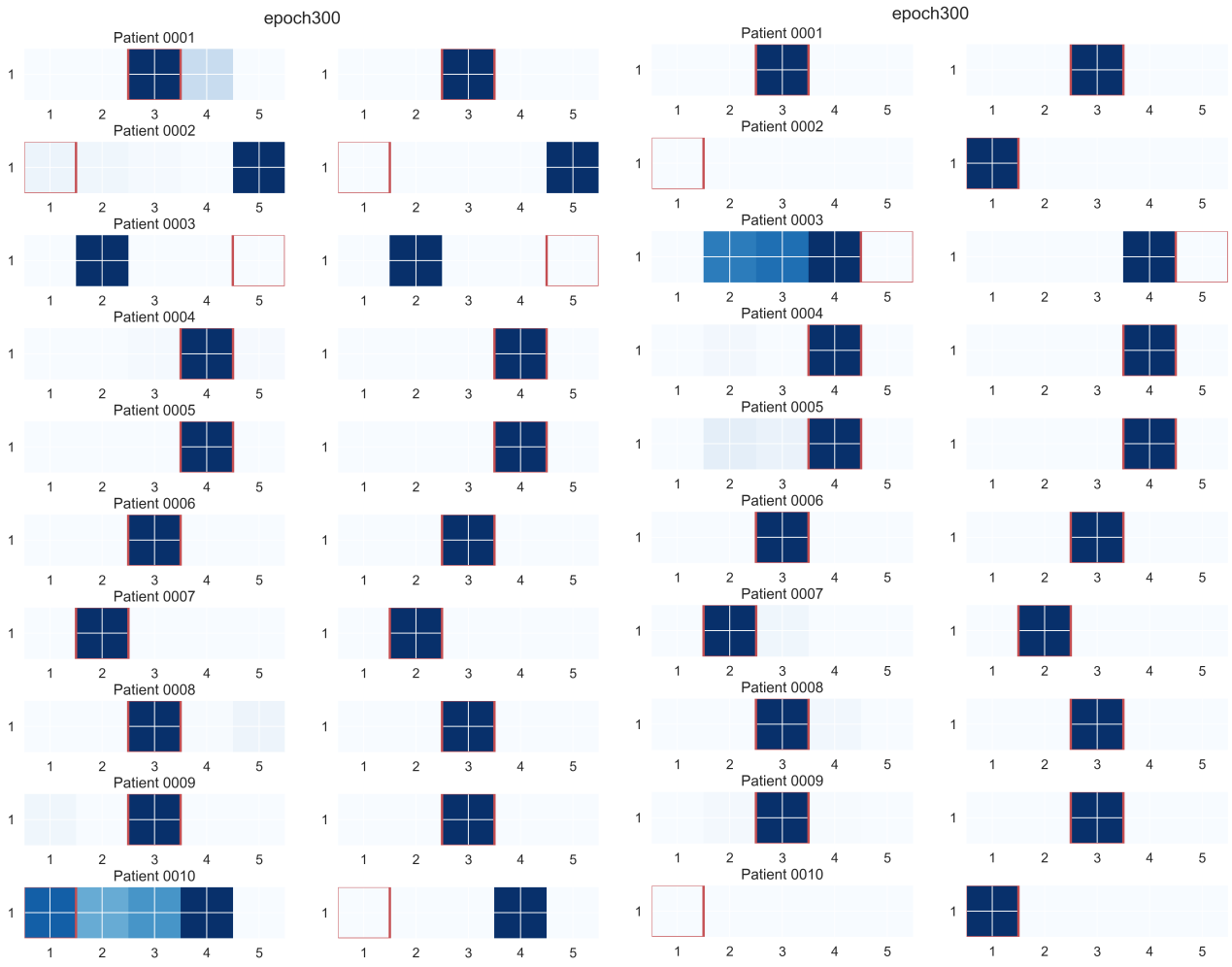


Figure 25: Experiment 8: confusion matrices and heatmaps of softmax model outputs at epoch 300



(a) Gumbel-Softmax samples (b) Unnormalized log probabilities

Figure 26: Experiment 8, epoch 300, left: soft version right: hard version

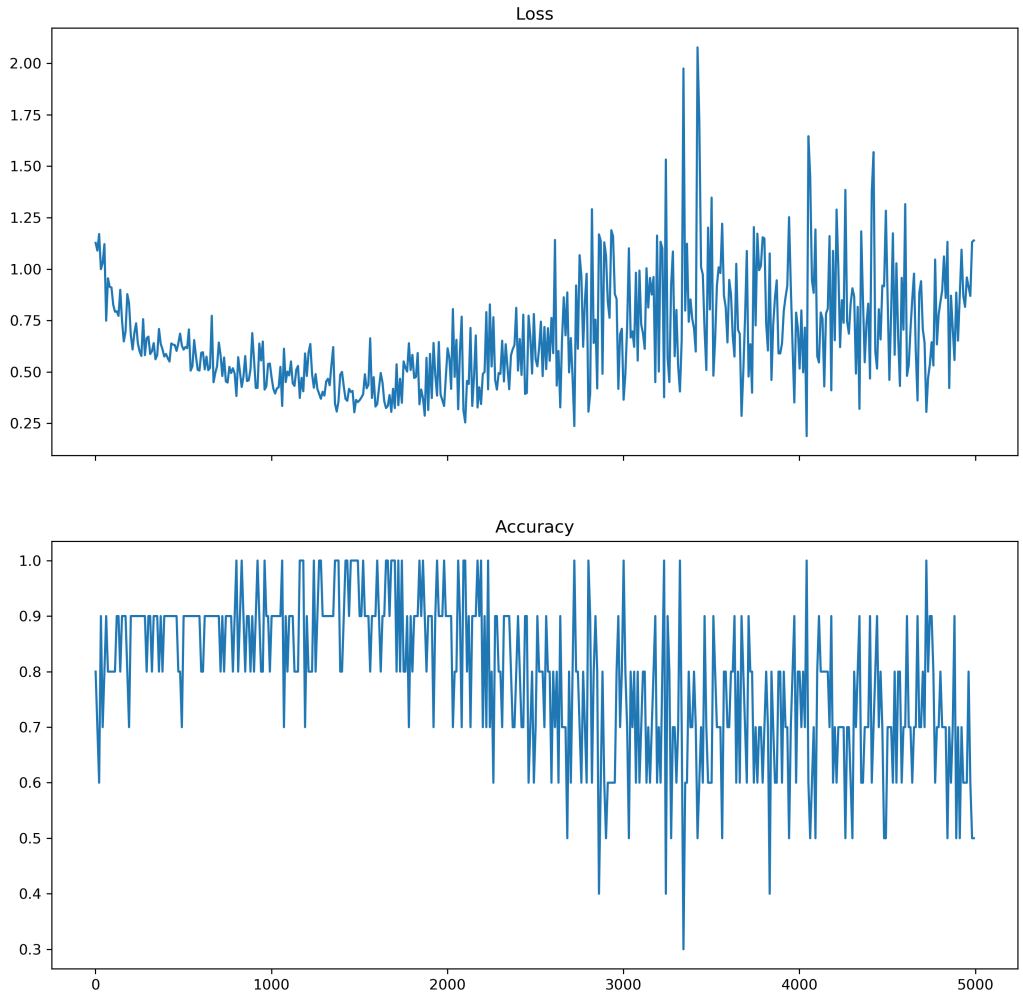


Figure 27: Experiment 9: loss/accuracy vs. epochs

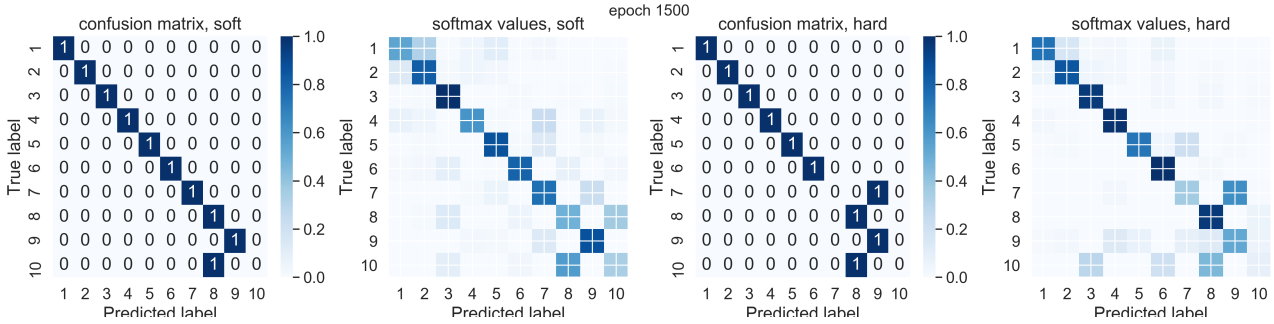
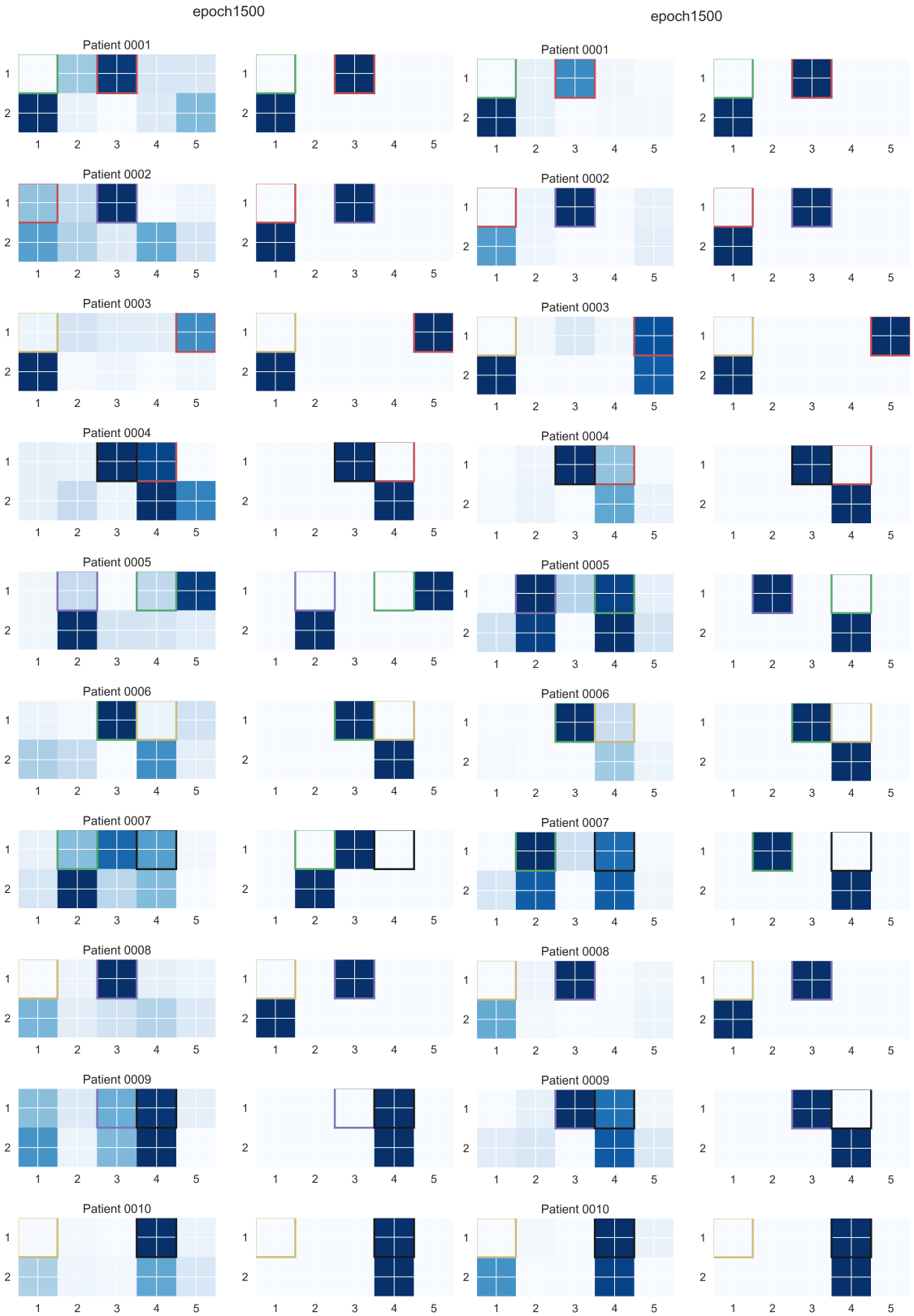


Figure 28: Experiment 9: confusion matrices and heatmaps of softmax model outputs at epoch 1500



(a) Gumbel-Softmax samples (b) Unnormalized log probabilities

Figure 29: Experiment 9, epoch 1500, left: soft version right: hard version

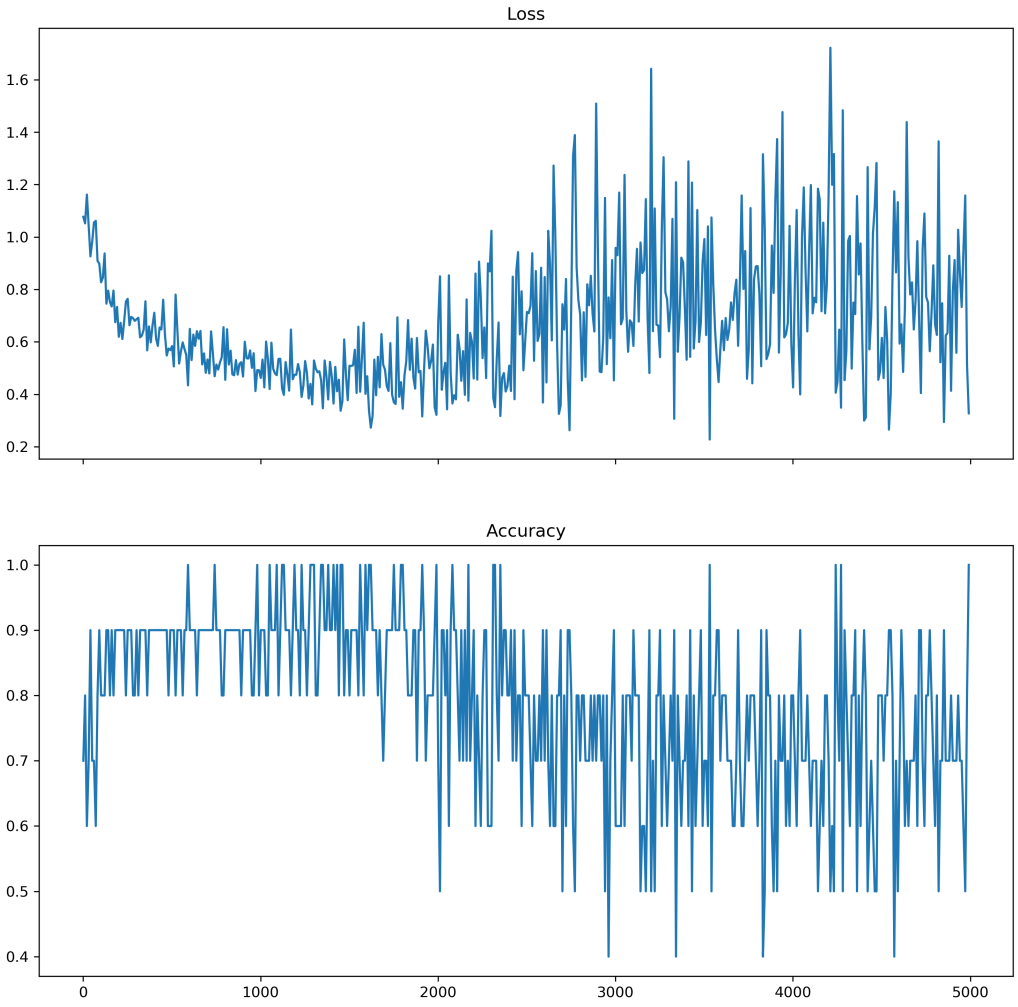


Figure 30: Experiment 10: loss/accuracy vs. epochs

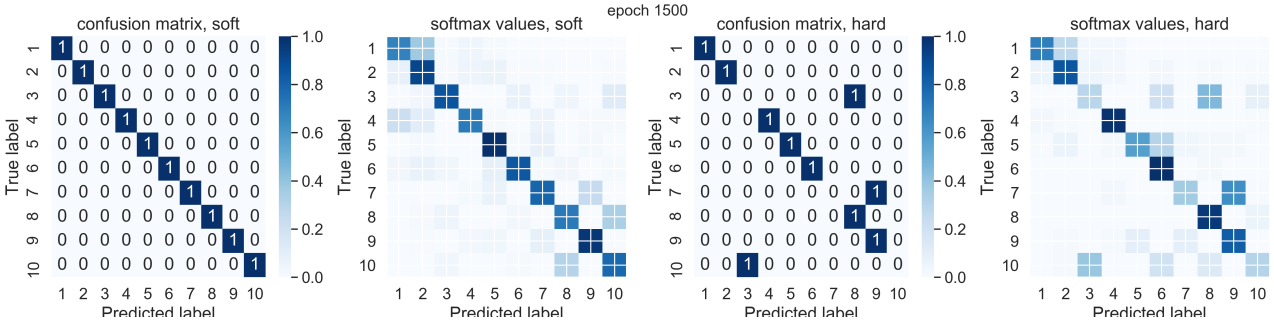
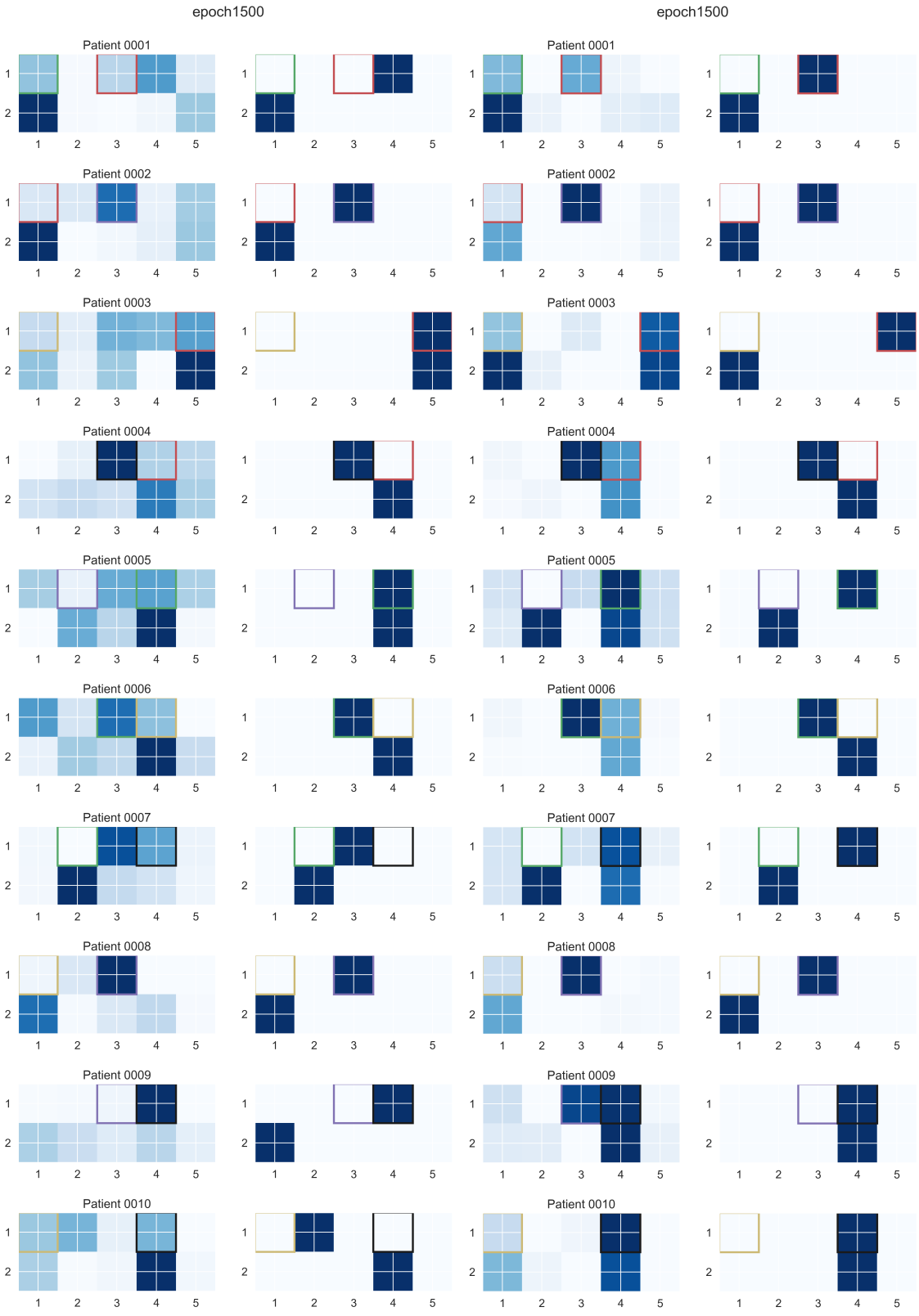


Figure 31: Experiment 10: confusion matrices and heatmaps of softmax model outputs at epoch 1500



(a) Gumbel-Softmax samples (b) Unnormalized log probabilities

Figure 32: Experiment 10, epoch 1500, left: soft version right: hard version

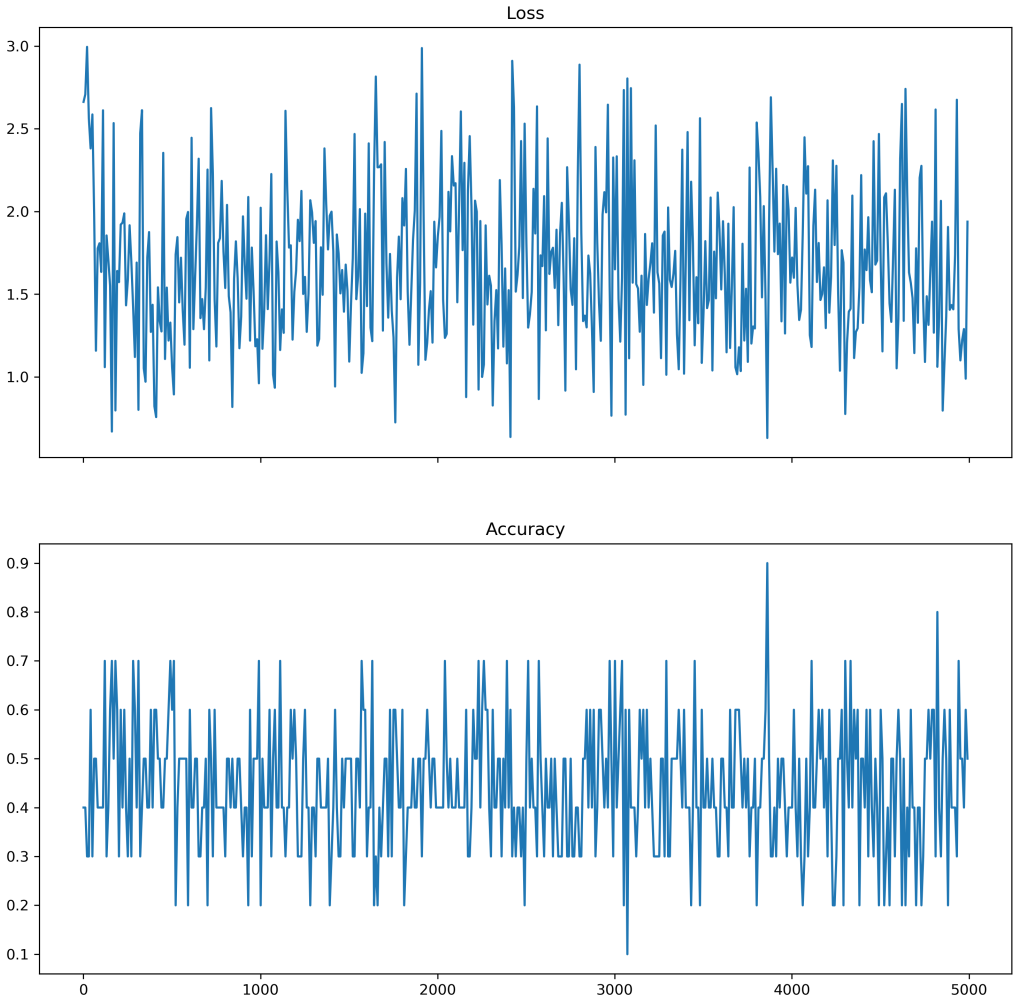


Figure 33: Experiment 11: loss/accuracy vs. epochs

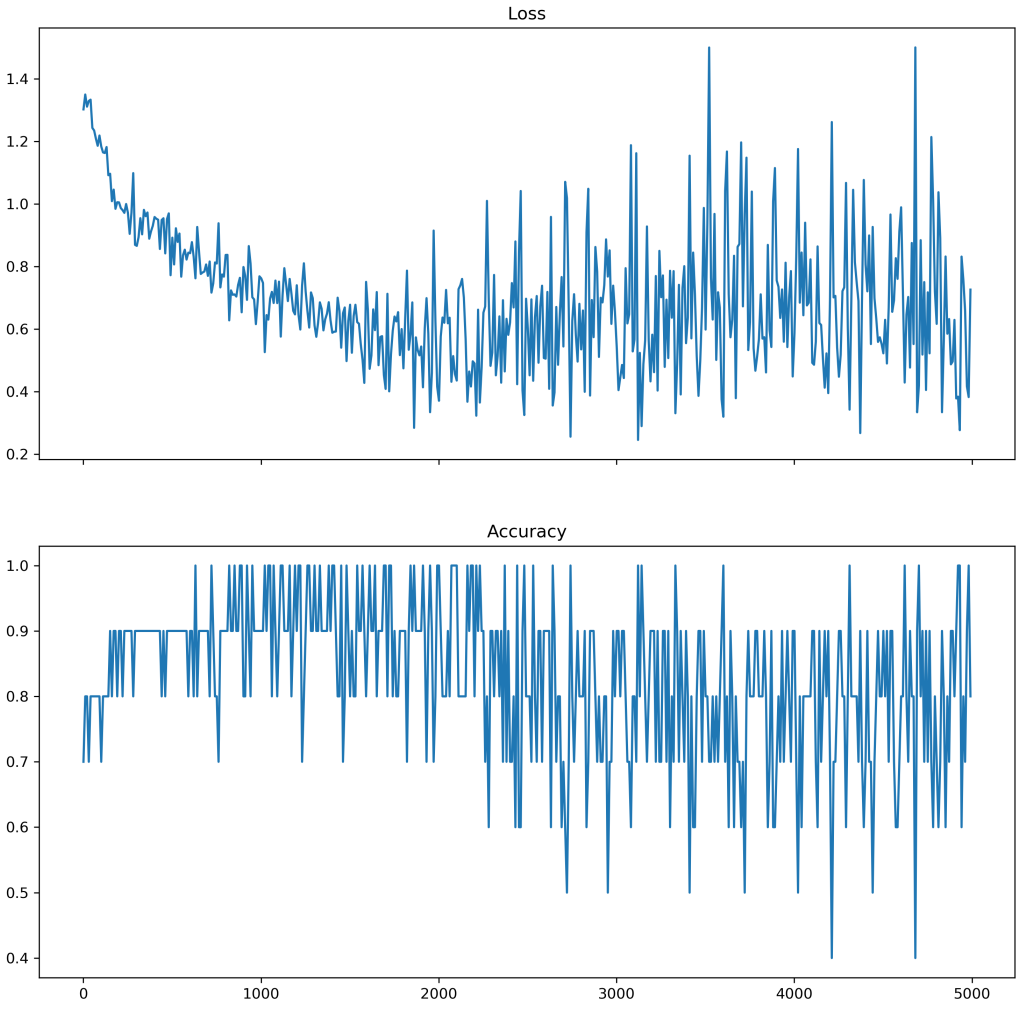


Figure 34: Experiment 12: loss/accuracy vs. epochs

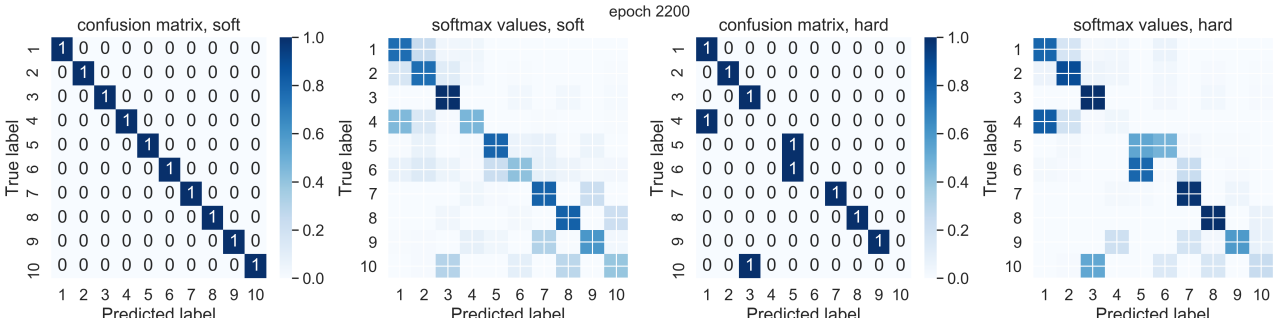
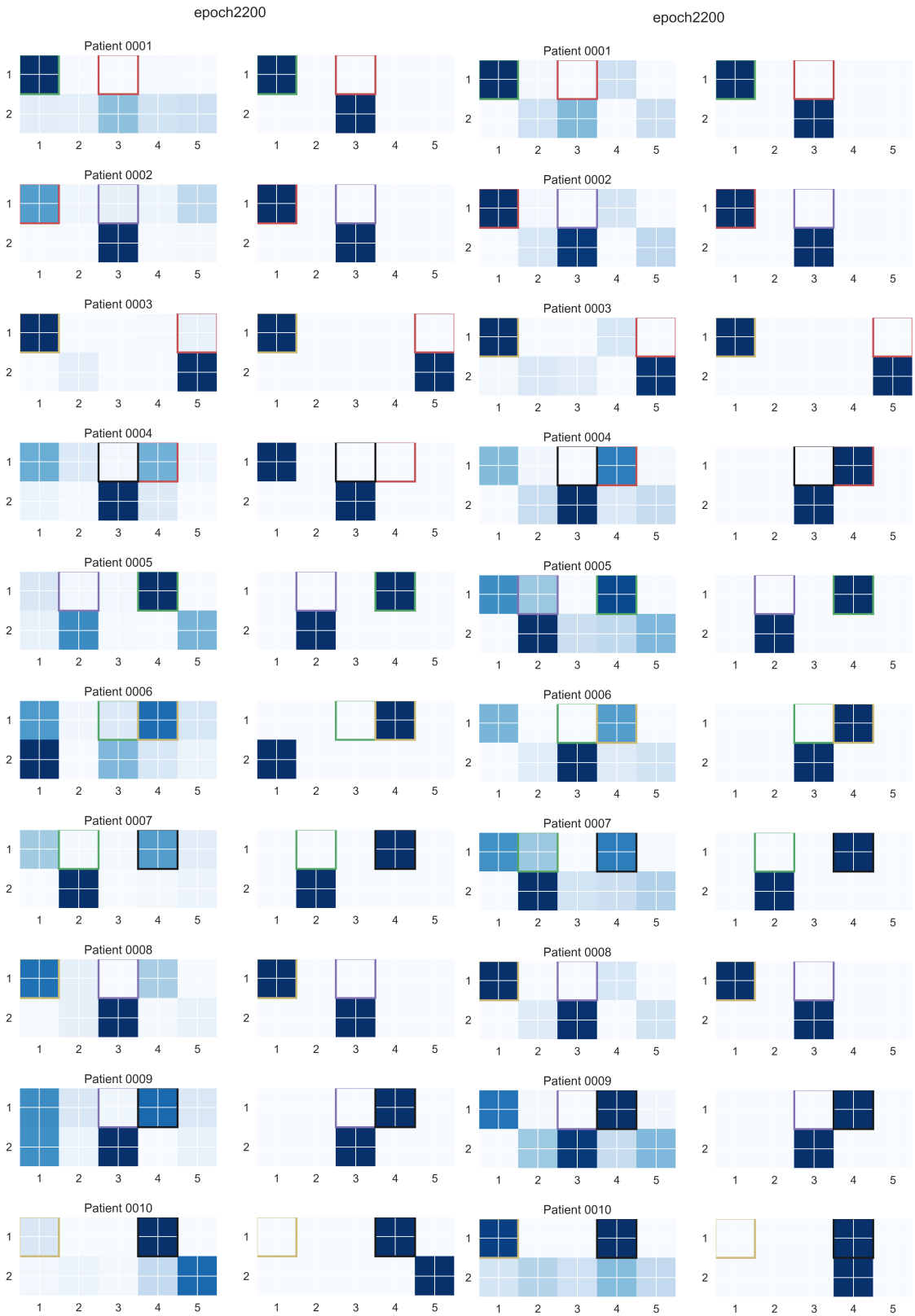


Figure 35: Experiment 12: confusion matrices and heatmaps of softmax model outputs at epoch 2200



(a) Gumbel-Softmax samples (b) Unnormalized log probabilities

Figure 36: Experiment 12, epoch 2200, left: soft version right: hard version

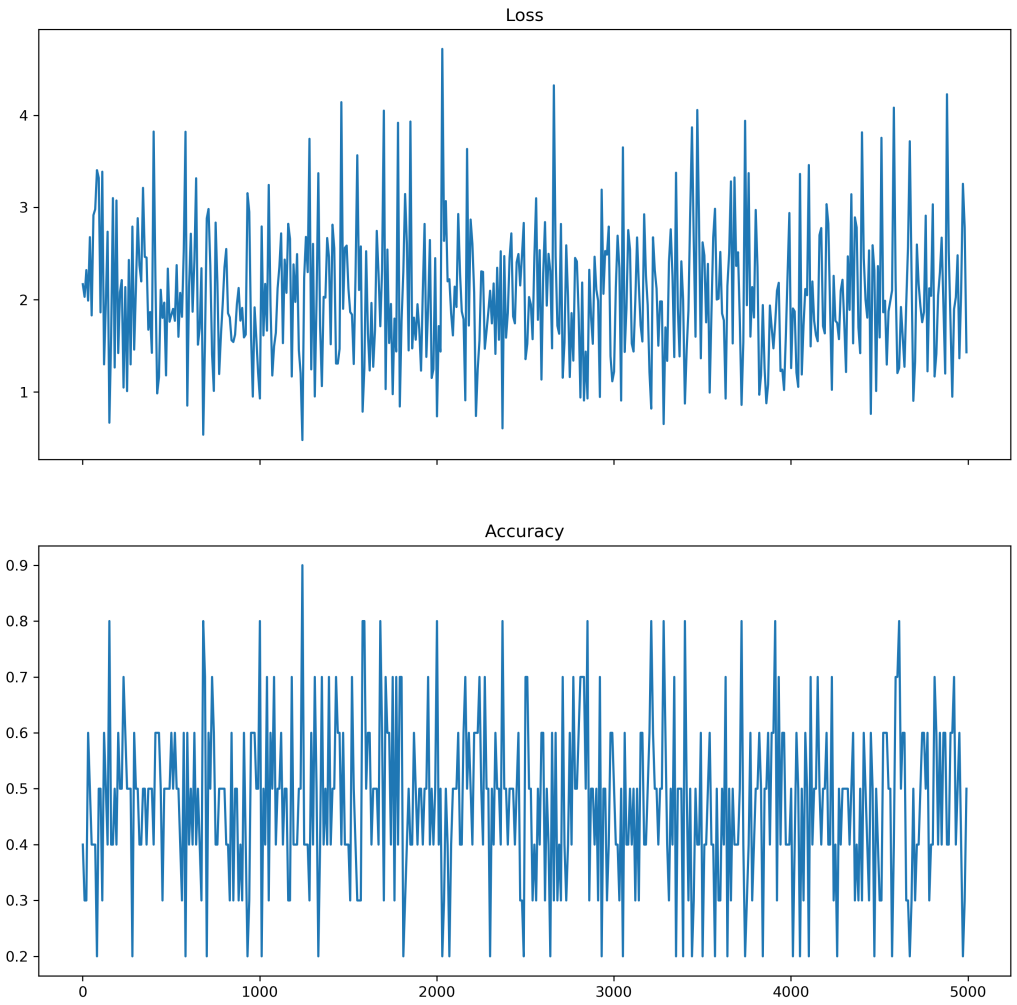


Figure 37: Experiment 13: loss/accuracy vs. epochs