**Visual Symptom Checker Trained using Reinforcement Learning**

**Zahra Hamedi**

**Vinzenz Uhr**

**MDP Formulation**

The Visual Symptom Checker operates as a Markov Decision Process (MDP), facilitating patient interaction by asking questions about symptoms and the patient’s responses. The MDP formulation encompasses a tuple (X, A, P, R, γ).

Ein Bild, das Text, Schrift, weiß, Diagramm enthält.

Automatisch generierte Beschreibung

Fig. 1: The agent-patient interaction in an MDP.

**States and conditions**

X in tuple (X, A, P, R, γ), represents the set of states, indicating the possible conditions or scenarios during the symptom-checking process.

Each state x ∈ X represents a patient state. Based on the original paper “Visual Symptom Checker” The state x is a composite of the patient's historical responses to the questions about symptoms, which is a vector of |S| elements and the prior probability distribution of diseases obtained from the pre-trained Convolutional Neural Network (CNN) applied to the patient's image, which is a vector of |C| elements.

S denotes the symptom space, encompassing symptoms such as low blood pressure, shortness of breath, dizziness, and weight gain. 0 corresponds to a not yet asked symptom, 1 means the patient has the symptom and -1 means that the patient does not have the symptom. The condition space, denoted as C, corresponds to the set of diseases in the dataset.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Fig. 2: Original State design from the paper

We encountered the problem that the agent asks the same symptom multiple times. We can imagine that this happens because the agent didn’t visit all the possible states and therefore rely on the state-action-values which he has already encountered.

Based on the formulation above we can’t tell from a state how many symptoms already have been asked. Therefore, we extended the state formulation by a number of asked symptoms N(S).

Our final state representation is shown by figure 3.

Ein Bild, das Text, Schrift, Reihe, Screenshot enthält.

Automatisch generierte Beschreibung

Fig. 3: Our state design

**Action Space**

A in the tuple (X, A, P, R, γ), denotes a finite set of actions, representing the available symptoms to ask for the agent (e.g. “has the patient low blood sugar?”).

The action space A aligns with the symptom space S, where each action a ∈ A represents a possible symptom s to inquire about. Each action in the action space corresponds to selecting a specific symptom to ask the patient during the interaction when an agent interacts with a simulated patient.

**Transition Probability P**

When the agent asks the simulated patient if he has a symptom, the patient answers with yes or no. The patient does this based on the conditional probabilities p(s|c) in HealthKnowledgeGraph.csv. The simulated patient answers s by generating a random number k from a uniform distribution between 0 and 1. If k <= p(s|c), the answer is yes, and no otherwise.

**Reward R**

The reward function as proposed in the original “Visual Symptom Checker”-Paper is equal to the posterior p(c\*|S), the probability of correct condition c\* given the asked questions S.

However, we encountered the problem that the agent started asking symptoms which are not related to the correct condition c\*. This happens because the posterior as described in Appendix A of the original Paper is calculated by .   
After each asked symptom which is correlated to the correct condition c\* the conditional probability p(s|c\*) will be multiplied to the current posterior. Because 0 <= p(s|C) <= 1, a correlated Symptom will usually decrease the posterior probability p(c\*|S). Symptoms which are not correlated are skipped and therefore don’t decrease the posterior.

As solution to this problem, we suggest adding the conditional probabilities p(s|c\*) instead of multiplying them.

An alternative would be to use p(s\_j=Yes|C)=0 for symptoms which are not correlated, but we think this is not a good idea, because a patient can have other symptoms which are not correlated (e.g. coughing). To be consistent we also would need to use p(s\_j=No|C)=1, leading to the same effect that the agent asks symptoms which are not correlated to prevent decreasing the posterior.

As described in the section “States and conditions” we also encountered the problem, that the patient asks the same symptom multiple times. To facilitate the prevention, we introduced a punishment after every step.

The final reward function is the addition of the correlated conditional probabilities p(s|c\*), the prior and the punishment.

**Dataset**

The original dataset of the paper is not freely available. Therefore, we used the Slake dataset consisting of 642 annotated medical images, covering 12 diseases and 39 organs throughout the body. Diseases include various cancers (e.g., brain, liver, kidney) and thoracic diseases (e.g., atelectasis, effusion, pneumothorax). Image modalities comprise 140 head CTs or MRIs, 41 neck CTs, 219 chest X-rays or CTs, 201 abdomen CTs or MRIs, and 41 pelvic cavity CTs. The dataset includes 282 CTs, 181 MRIs, and 179 X-Rays. All CTs and MRIs are axial single-slice, and the dataset offers a comprehensive view of medical imaging modalities and complexities.

Following 12 disease labels are available: ['Mass', 'Pneumothorax', 'Brain Enhancing Tumor', 'Pneumonia', 'Nodule', 'Atelectasis', 'Liver Cancer', 'Pleural Effusion', 'Cardiomyopathy', 'Brain Edema', 'Lung Cancer', 'Infiltrate', 'Brain Non-enhancing Tumor']

However, for only five of these diseases ('Pneumothorax', 'Pneumonia', 'Pleural Effusion', 'Cardiomyopathy', 'Lung Cancer'), symptom probabilities are available in the 'HealthKnowledgeGraph.csv.' These diseases are specifically related to lung and heart conditions. Consequently, this project focuses on 111 chest images from the Slake dataset, concentrating on diseases associated with the lungs and heart.

We split the dataset into a train and test set using stratification and an 80/20 split.

**CNN Network for Visual Prior**

We employed the AlexNet convolutional neural network (CNN) architecture to achieve probability distributions of the five target diseases: Pneumonia, Pneumothorax, Lung Cancer, and Effusion based on chest medical images. Before training, data preprocessing was conducted to enhance the quality of the dataset. This included image resizing, normalization, and augmentation techniques to ensure the model's robustness and generalization capabilities. In this project, we employed rotation, x and y shift, contrast adjustment, and crop augmentations for training medical image classification models. Medical images often exhibit variability in terms of patient anatomy and imaging conditions. Therefore, these augmentations help the model learn invariant representations, making it more robust to these variations. In our dataset, we have imbalanced distribution of classes.

The label Cardiomegaly has 33 samples, Pneumonia has 28 samples, Pneumothorax has 15 samples, lung cancer has 18 samples, and Effusion has 17 samples. The training process involves minimizing the Focal loss function. We adopted a weighted loss approach using focal loss. Focal loss is particularly well-suited for imbalanced classification problems, as it introduces a modulating factor that down-weights the contribution of well-classified examples, effectively focusing the model on the more challenging instances, and is tailored to address class imbalances

We utilized stochastic gradient descent (SGD) as the optimization algorithm. Moreover, to alleviate the impact of class imbalance, we employed the WeightedRandomSampler during the training phase. This sampler ensures that batches are composed in a way that gives more importance to under-represented classes by assigning higher probabilities to samples from these classes.

In our fine-tuning process, we optimized hyperparameters, settling on a learning rate of 0.0045 and a weight decay of 5\*1e-2. These choices were made after iterative experimentation to achieve a balance between effective model training and preventing overfitting.

**Summary of the AlexNet model:**

A screenshot of a computer code

Description automatically generated

Fig. 4: Architecture of CNN model

**Evaluation CNN**

Ein Bild, das Screenshot, Rechteck, Text, Diagramm enthält.

Automatisch generierte Beschreibung

Fig. 5: Evaluation Confusion Matrix of the CNN

A graph of training and validation loss

Description automatically generated

Fig. 6: Train and Validation Loss and Accuracy of the CNN

**RL environment**

The visual symptom checker agent learns an estimate of the optimal action-

value function Q\*. We used the Deep Q-Network (DQN).

Ein Bild, das Text, Schrift, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

Fig. 7: Architecture of the DQN network

We created a DataLoader class that loads the image and calculates the visual prior for each image using the trained CNN network. This speeds up the RL training process because it doesn't have to compute the same visual prior repeatedly.

For the MDP environment we created the Env class. We didn’t use any RL specific libraries like OpenAI gym.

As agent we used -greedy with a linear decay.

**RL training**

Each episode consists of ten training steps. A training step consists of asking the patient about a symptom. Each step transition is stored in a replay buffer of size 10,000. A step transition consists of . is the current state, is the action took at state , is the next state and is the received Reward.

After one episode, we ran ten training iterations of the DQN network. In each iteration we used a batch of 128 transition samples from memory. Based on the DQN paper, we clipped the gradients to the maximum norm of 100. After each iteration we performed a soft update of the target network.

During the training process we tried different epsilon decays. Other parameters were not tuned.

**RL Results**

We used Tensorboards to track different metrics during the training process. Following metrics we documented: Training loss, norm of the gradients, reward at each step, final reward at the end of the episode, total reward per episode (sum of all rewards), top-1 accuracy, top-2 accuracy, top-3 accuracy and the number of selected actions during evaluation. The rewards are measured on the training set and the top-K accuracy and the number of actions on the validation set.

We did out final evaluation with two models: One uses our reward approach with punishments after each action and addition instead of multiplication to combine the conditional probabilities. We call it “Addition-Model”. The second model uses the reward from the paper with multiplication and without punishments. We call this “Multiplication-Model”.

|  |  |
| --- | --- |
| Number of epochs | 12000 |
| Training Time | ~18 hours |
| Epsilon | 3000 epochs, for 3000 epochs linear decay, for the last 6000 epochs |
| Gamma (future reward discount factor) | 0.9 |
| TAU (soft update of target network) | 0.005 |
| Learning rate | 1e-5 |

Table 1: Used parameters for both models

Ein Bild, das Text, Diagramm, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

Fig. 8: Value curve of Epsilon

Ein Bild, das Text, Diagramm, Screenshot, Design enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Diagramm, Screenshot enthält.

Automatisch generierte Beschreibung

Fig. 9: Metrics from the Multiplication-Model

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

Fig: 10: Confusion Matrix from the Multiplication-Model

Ein Bild, das Text, Diagramm, Reihe, parallel enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Diagramm, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

Fig. 11: Metrics from the Addition-Model

Ein Bild, das Text, Screenshot, Zahl, Diagramm enthält.

Automatisch generierte Beschreibung

Fig. 12: Confusion matrix from the Addition-Model

|  |  |  |
| --- | --- | --- |
| Multiplication-Model | Top-1 | 0.49 |
| Top-2 | 0.71 |
| Top-3 | 0.88 |
| Addition-Model | Top-1 | 0.51 |
| Top-2 | 0.75 |
| Top-3 | 0.87 |
| Untrained-Model (CNN only) | Top-1 | 0.49 |
| Top-2 | 0.73 |
| Top-3 | 0.87 |

Table 2: Mean Top-K-Accuracy for 100 iterations

**Discussion**

Both models were able to reduce the loss, i.e. learn the action value. But neither model was able to make any progress in top-k accuracy on the validation set. When comparing with an untrained RL model, we see that the accuracy and the confusion matrices are almost the same and that the models depend entirely on the prior coming from the CNN.

The Multiplication-Model could increase the Reward a little bit. We assume the model learns which actions doesn’t decrease the posterior and then just takes these actions. This would align with the number of selected actions.

The Addition-Model could increase the reward significantly, but there was still no progress on the Top-K-Accuracy Evaluation. This surprised us a little. We would have expected that it would increase. This either means that our approach is wrong, that the model can't learn enough from the training data, or that there is a bug in the code that we haven't discovered yet. We could also be unlucky with our train-test-split. A better approach would be cross-validation, but this would increase the training time by a multiple of the number of foldings.

The oscillations of the rewards and the Top-K-Accuracies are also striking. However, we do not think that this is an abnormality, but an expected behaviour.

When we compare the training data there is a significant difference between the number of data from the paper and our dataset. They had 5841 images and we had 111 imgs. This is a factor of x52.

**Encountered problems**

As described in the section “States and conditions” we encountered the problem, that the patient was asked the same symptom multiple times.

We can imagine that this happens because the agent didn’t visit all the possible states and therefore relied on the state-action values which he had already encountered. Based on the Bellmann Equation the state-action-value depends a lot on the discounted value of the successor state. At the beginning of the training the output of the DQN network is initialized to a normal distribution. This leads to a discounted value around 0 for successor states, which have not been visited. This gives lower state-action-values for correlated symptoms which have not been asked a lot.

As described in this report we tried different approaches but none of them were completely satisfying.

Debugging the problems we encountered was also difficult. We had some bugs in our code and it took a long time to find them. For example, we had two bugs regarding asking for the same symptom multiple times. First, we calculated the penalty, i.e. the number of symptoms asked, based on the symptoms in the state representation. In the state representation there is no representation for asking the same symptom twice. The model learned that there was no penalty for asking for the same symptom again. The second error was that we didn't check whether a symptom had already been asked. If the model asked the patient for an unrealistic symptom and the patient answered yes, the model learned that it could ask again, resulting in a better reward most of the time. Both bugs took a long time to discover.

Another problem was the speed of training. As the agent learns the RL environment serially, this slows down the whole training process considerably. We wanted to use multiple threads for this, but we didn't have the time to implement this. Slow training iterations then limited us to shorter training periods.

**Attempted but discarded approaches**

To counteract the discounted value around 0 for unvisited successor states, we tried optimistic initialization. We initialize the bias of the output layer to 30. Unfortunately, it quickly got stuck in a bad local minimum, asking the same questions repeatedly. From there we didn't go any further.

Ein Bild, das Text, Screenshot, Reihe, Diagramm enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Screenshot, Diagramm, Reihe enthält.

Automatisch generierte Beschreibung

Fig 13: Metrics with the optimistic initialization approach

At the beginning we used exponential epsilon decay. After the described problems, we needed a longer exploration period. To make it easier to control, we switched to a linear decay.

**Open Questions**

An open question is how the original paper could use a reward function which multiplies the conditional probabilities. It’s also possible that they used Addition aswell, because in Appendx A it’s not clearly described how the calculcated .

Interesting would also be which parameters they used for Epsilon, Tau etc.

Another open question is how to handle train-test split to train two networks, where one network depends on the other. To train the CNN we split the Dataset into a train und test split. For the RL model we used the same split. This means that the computed visual prior is biased because the CNN could learn on these pictures. An alternative would be to train the RL model only on the test split. In this case we would need another split as test split for the RL model. This would decrease the Trainingset for the CNN and the RL model. If we then add a validation set for both trainings, it gets even worse. In the original paper they mentioned that they used a train-val-test split for the CNN but it’s not clearly described how the split the data for the RL model.