# Topic 15: Learning to play "20 Questions"

*COMP9417, Assignment 2, Jake Fitzgerald z5019627*

# 1 Introduction

The task was to build an intelligent bot to play the traditional "20 questions" game. The game involves the player selecting an object, animal, person, etc. The bot is then able to ask up to 20 yes or no questions to determine which object the player selected. In this case an artificial neural network (ANN) written in Python was used to implement the bot, following Robin Burgener's original design. Due to limited data and the high computational cost of training large neural nets, the game was simplified to guessing a subset of animals with a varying but smaller question limit (less than 20), with a broader goal of creating a scalable network.

# 2 Method

With 20 questions, Burgener's bot could distinguish over 1 million answers ($2^{20}$) with 80% accuracy (jumping to 98% accuracy with 21 questions). His methodology, as outlined in the 20Q patent, makes up the basis of my bot.

## 2.1 Data

Sourcing training sets for 20Q was challenging. Traditionally, user input trains the game by teaching it new objects and reinforcing their relationships with question/answer combinations. Due to time restrictions this was impossible, so instead I used a public zoo dataset provided by University of California, Irvine (58 unique animals and 28 attributes used as questions) to create a second 'Zoo' bot simulating a user. The training and validation sets are simply interactions between the 20Q bot (which asks questions) and the Zoo bot (which answers questions from the dataset). For debugging / experimental purposes, datasets of varying size were created: micro, small, medium and big (original).

## 2.2 Network

The ANN is the intelligence behind the 20Q bot and the focus of this report. In layman's terms, the network's goal is to translate a question and answer vector and into a vector of animal probabilities, where the correct animal has the highest probability. *Figure 1* outlines the network's high level functionality. Given n questions and m animals, the ANN feeds forward and propagates backwards through three layers:

- **Input layer:** A vector of n questions / answers (q/a). All questions are referred to by indexes from Zoo. Questions asked have their index in the input vector set to +1 for a yes response, -1 for a no response and 0 if the question isn't asked.
- **Output layer:** A target vector of m animal probabilities. Given a q/a vector, the ideal target vector is all 0's with one 1 for the correct animal index (i.e. 100% probability for the correct guess, 0% probability for any incorrect animal).
- **Hidden layer:** Captures non-linear dependencies between q/a vectors and their corresponding animal vectors.

The network has a bias of 1 (for input and hidden layers) to ensure outputs are % probabilities between 0-1 and uses a sigmoid squashing function for nonlinearity.

## 2.2.1 Algorithm selection

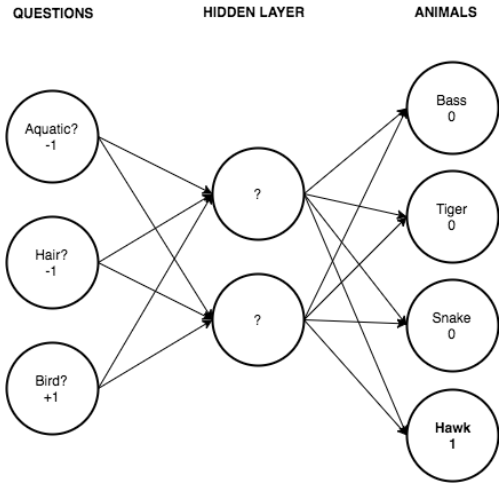


*Figure 1: High level network structure*

ANN was chosen due to its inherent adaptiveness, flexibility and ability to capture non-linear characteristics, all of which are critical in the 20Q game. The relationship between Boolean questions/answers and animals is non-linear and has many underlying factors that an ANN can fit to. Moreover, the game is based on human input which is subjective and error-prone. For example, if the animal was a cow, the answer to "Can it be eaten?" would vary depending on religious background. ANN provides high flexibility for incorrect / subjective user input that other algorithms like decision trees (the obvious alternative) cannot. Moreover, ANNs are less susceptible to the curse of dimensionality since they compress data in hidden layers, unlike decision trees which are highly susceptible. Since each possible question adds another dimension, a decision tree would not be appropriate here.

# 2.3 Smart question selection

The 20Q strategy is to ask questions which cut out as many animals as possible. Since the probability of a yes-no response is 50/50 from the bot's perspective, it is best to select questions that cut out closest to 50% of remaining animals. Half of the bot's functionality is identifying relationships between q/a's and animals, while the other half is selecting the best next question.

At a low level, the bot must select the question which creates the greatest difference from the current probability vector *for the worst case of its yes or no response*. When selecting the next question, every unasked question is fed forward through the network for both its yes and no response to get two hypothetical probability vectors. These hypothetical vectors are subtracted from the current vector and their absolute sum is calculated (despite its simplicity this algorithm performed better than many others using standard deviations, averages, etc). The worse of the two sums represents a question's 'score', where the highest scoring question is the next best question. This worst-case-scenario method avoids selection of non-descriptive questions, as outlined in the example in *Figure 2*.

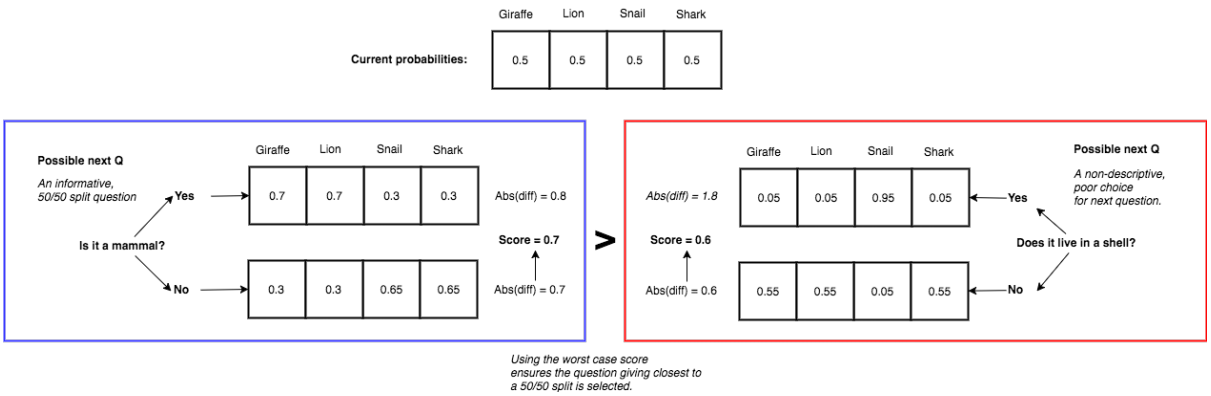


*Figure 2: Smart question selection using worst-case-scores*

## 2.4 Cross validation

Typically, a neural net is trained with a large training set, validated on a subset of those training sets and then tested using an unseen testing set. Due to the unique nature of the problem (and limited computing power), an atypical cross validation method was used:

- **Training set:** For each animal, a random q/a vector was fed into the network for each animal. This was repeated n times. The training sets *did not* use smart question selection as it was too costly to feed forward $(q\_limit)^2$ times over 1000's of sets (especially at scale). It was also hypothesised that smart question selection could cause the network to overfit to certain question paths in early training, limiting its exposure to different question combinations.
- **Validation set:** n/10 games with random animals and smart question selection. These sets were not used to train the network, simply to validate its progress for certain epoch.
- **Testing set:** Games with real user input. Guesses are included as questions. This is the ultimate measure of the bot's correctness and is suitable. Correct guesses from the bot were used to further train the network.

## 2.5 Python program

The network and bot were written in Python. At a very high level, the program was broken up into three classes:

- **Zoo:** Answering questions from the dataset relating animals to questions.
- **NN:** The neural network (abstracted).
- **20Q:** Has all game mechanics, Zoo feeds data into 20Q which bases its responses on a "brain" from NN, essentially connecting the Zoo and NN classes.

For more details, view the README.txt of the submitted files.

# 3 Experimentation

To create a scalable network, parameters must be optimised with precision. In the case of an ANN, overfitting is a big issue while high dimensionality is *less* problematic. To avoid overfitting, a series of experiments relating to the following parameters were undertaken:

- **Learning rate:** How aggressively each training set is propagated backwards (0 to 1).
- **Hidden layer size:** Number of neurons in the hidden layer.
- **Epoch:** Maximum number of training sets before overfitting.

Mean square error and accuracy of validation sets were used as measures for parameters.

## 3.1 Learning rate

Learning rate is a % ratio representing the rate at which a network propagates backwards, i.e. the degree to which it reinforces each training set. A learning rate that is too high results in aggressive learning and overfitting. In a game subject to erroneous input, overfitting to incorrect q/a sets could establish incorrect relationships. This is evident in *Figure 3*, where a lower learning rate of 0.1 results in a smooth learning curve, whereas a learning rate of 1.0 overfits (in this case, every guess beyond 60k epoch was 'chicken').

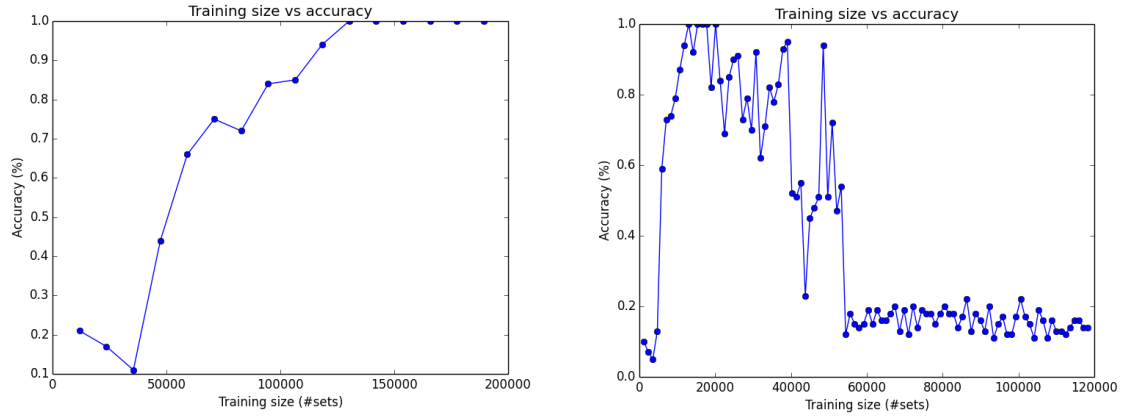


*Figure 3: Epoch vs accuracy for a learning rate of 0.1 (left) versus 1.0 (right) on the same 'medium' dataset*

## 3.2 Hidden layer size

Number of neurons in the hidden layer is a critical parameter. Too few neurons and the network cannot fit all the predictive factors, but too many and the network overfits to the training sets and cannot generalise well. As shown in *Figure 4*, a hidden layer size 38 gives the smallest error / highest accuracy, with overfitting occurring past 42 hidden neurons.

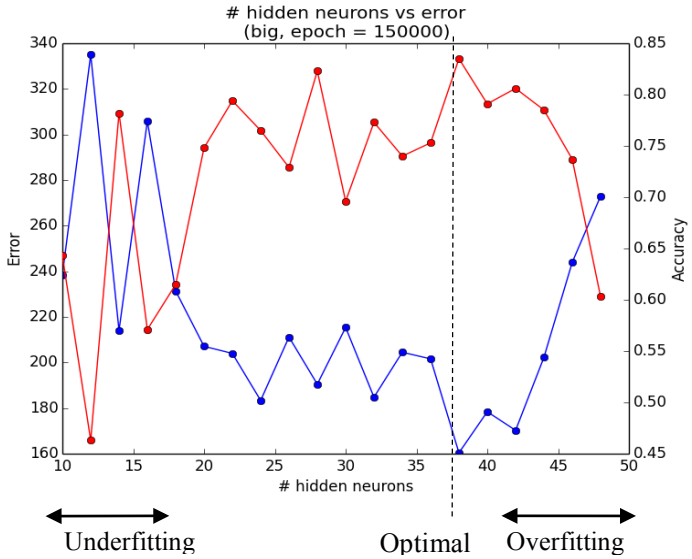


*Figure 4: Hidden layer size against error for 'big' dataset*

## 3.3 Epoch and error

Too many training sets overfit the network, especially with a high learning rate. To overcome this, the network was validated after certain epoch intervals. Training ended once a validation set error hit below an upper bound (error of less than 10% of the set size). *Figure 5* shows cross-validation *without* the error break for a network with 38 hidden neurons and a 0.1 learning rate (left) versus 0.9 learning rate (right). Despite the smaller epoch required for adequate training, a 0.9 learning rate caused the error to bounce between 20 and 40 after 240k epoch, in contrast to the 0.1 learning rate which produced a smoother learning curve. As such, a learning rate of 0.1 was selected.

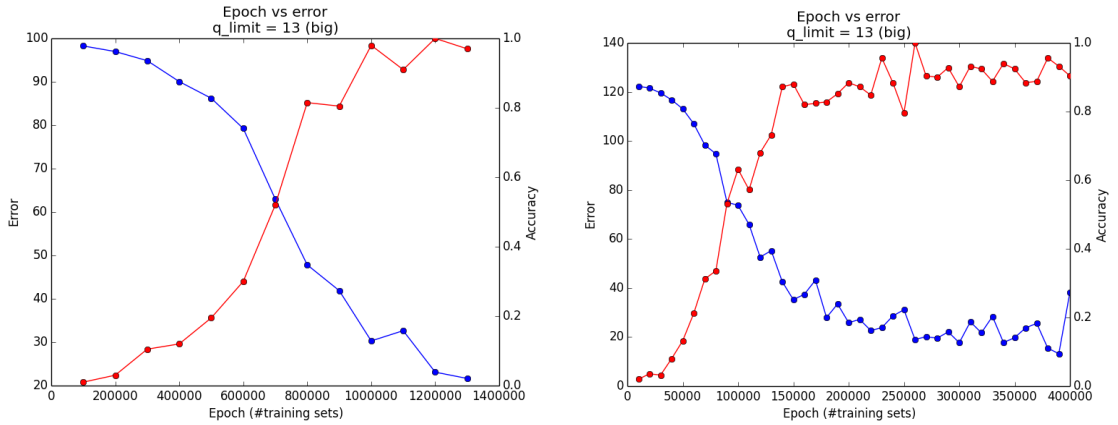


*Figure 5: Learning rate of 0.1 (left) versus 0.9 (right)*

# 4 Results

The bot achieved 100% accuracy and low errors across the micro, small and medium datasets. The bot struggled to differentiate between similar animals in the big dataset, such as ‘toad’ versus ‘frog’ and ‘haddock’ versus ‘bass’, but on the most part was able to guess correctly. It also showed evidence of learning in *Figure 7*, where it was able to adjust from its incorrect ‘pitviper’ guess to correctly guess ‘honeybee’ in the next game. Ultimately, the bot guessed animals in < 13 questions with a relatively high success rate and achieved 100% accuracy in several validation sets (*Figure 6*).

```
win: slug ... 13 qs: 1. domestic? n (aardvark 1.0%) 2. predator? n (wasp 4.0%) 3. venomous? n (wasp 11.0%) 4.
tail? n (housefly 22.0%) 5. catsize? n (housefly 23.0%) 6. breathes? y (housefly 30.0%) 7. 6 legs? n (housefly
 27.0%) 8. backbone? n (slug 46.0%) 9. 0 legs? y (slug 87.0%) 10. airborne? n (slug 98.0%) 11. aquatic? n (slu
g 98.0%) 12. 4 legs? n (slug 97.0%) 13. eggs? y (slug 98.0%)
win: duck ... 13 qs: 1. domestic? n (aardvark 1.0%) 2. predator? n (wasp 4.0%) 3. venomous? n (wasp 11.0%) 4.
tail? y (housefly 10.0%) 5. catsize? n (haddock 18.0%) 6. 4 legs? n (haddock 42.0%) 7. eggs? y (haddock 73.0%)
 8. toothed? n (haddock 35.0%) 9. backbone? y (haddock 54.0%) 10. fins? n (haddock 42.0%) 11. airborne? y (duc
k 56.0%) 12. aquatic? y (duck 81.0%) 13. 0 legs? n (duck 86.0%)
====================================
100.0 % accuracy, error= 23.0975882177
====================================
```

*Figure 6: 100% score in a validation set for 'big'*

```
================ YOUR TURN: SELECT ANIMAL ================
~Your animal will not be shared with the bot until it makes its final guess
Your animal is: honeybee
====================== 13 QUESTIONS ======================
1. 8 legs?
Answer? (y/n) n
( calf 1.0 % )
2. 5 legs?
Answer? (y/n) n
( calf 1.0 % )
3. venomous?
Answer? (y/n) y
( seasnake 54.0 % )
4. aquatic?
Answer? (y/n) n
( pitviper 47.0 % )
5. fish?
Answer? (y/n) n
( pitviper 87.0 % )
6. milk?
Answer? (y/n) n
( pitviper 95.0 % )
Bot guessed ' pitviper ' incorrectly! ( honeybee )
Training network...
```

```
================ YOUR TURN: SELECT ANIMAL ================
~Your animal will not be shared with the bot until it makes its final guess
Your animal is: honeybee
====================== 13 QUESTIONS ======================
1. 8 legs?
Answer? (y/n) n
( calf 1.0 % )
2. 5 legs?
Answer? (y/n) n
( calf 1.0 % )
3. venomous?
Answer? (y/n) y
( seasnake 55.0 % )
4. aquatic?
Answer? (y/n) n
( pitviper 31.0 % )
5. fish?
Answer? (y/n) n
( pitviper 77.0 % )
6. milk?
Answer? (y/n) n
( pitviper 91.0 % )
7. invertebrate?
Answer? (y/n) y
( scorpion 81.0 % )
8. domestic?
Answer? (y/n) y
( honeybee 92.0 % )
9. predator?
Answer? (y/n) n
( honeybee 99.0 % )
Bot guessed ' honeybee ' correctly!
Training network...
```

*Figure 7: Evidence of learning with incorrect guess (left) followed by correct guess (right)*

# 5 Discussion

While the network conquered the Zoo dataset, its cross-validating method had some fundamental flaws due to limited computing power. Subsequently the network has scalability issues which are discussed below.

## 5.1 Scalability

The need for a highly scalable network presented several issues:

1. **Dynamically adding questions and targets:** The biggest limitation of the current network is that it can't add new targets to the network via user input, i.e. if a user selects an animal not in the list, it will not add it for future games. The problem with dynamic addition is that it requires adding new input and output nodes to the network. To avoid catastrophic interference, *the entire network must be re-trained* and the size of the hidden layer may also need to increase. As such, this feature was excluded as it seemed superfluous and potentially harmful.
2. **Sourcing questions:** While new targets can be added via user input on an incorrect bot guess, *questions must be sourced before users interact the bot*, presenting a significant data mining challenge.
3. **Computational cost:** The time complexity of training a network is exponential, particularly when cross-validation is done correctly (see below).

## 5.2 Cross-validation flaws

Due to limited computing power, the training, validation and testing sets run slightly different versions of the game. The training set runs a brutish game of randomly generated questions (non-sequential), the validation set runs a similarly brutish game with smart question selection (sequential) and the testing set (user input) uses smart question selection *with guesses being included as questions*.

Originally this guessing algorithm was hard-coded: once the network was > 95% certain of a target, it used the next question as a guess. Not including this functionality in the training / validation sets meant that this guessing threshold was not naturally embedded in the network, resulting in an under-performing 20Q bot. It was replaced with a last-question guessing policy. When cross-validating, ideally each set plays the same game, however sequential sets with smart question selection and guessing thresholds ups time complexity from linear to exponential. Having exponentially complex training sets is not feasible, especially at scale.

## 5.3 Improvements

1. **Dynamic adding:** A naïve solution would be to include empty input and output neurons, filling them with new q/a's and targets. A better solution would be to use a reinforcement learning algorithm with a Markov decision process.
2. **Less aggressive learning rate:** A smaller learning rate of 0.01 would improve the algorithm's performance at scale when there are far more underlying factors.
3. **Dynamic programming:** Using dynamic programming techniques with a reinforcement learning algorithm could lower complexity enough to allow for consistent training / testing sets with all features (smart question selection, guess thresholds, etc).
4. **Scaled responses:** Similar to the real game, allow for multiple answers with different weights. E.g. no = -2, probably not = -1, unknown = 0, maybe = +1, yes = +2.

# 6 Conclusion

The 20Q bot and its ANN are functional on smaller datasets, but in its current form the bot has limited scalability and high computational cost of training (as well as very limited data). Ultimately, the correct implementation of a 20Q bot would use a reinforcement learning algorithm as opposed to a traditional ANN with a squashing function such that it can dynamically grow as user input becomes more diverse. Nonetheless, the task of building a simple 20Q bot was successfully completed, with a rough framework setup for establishing a scalable, intelligent bot capable of playing a fully-fledged 20Q game.