DESIGNING MODULAR NEURAL NETWORKS FOR USE IN A VIRTUAL ECOSYSTEM GAME

Jonathan Baker
National Centre for Computer Animation
Bournemouth University

DESIGNING MODULAR NEURAL NETWORKS FOR USE IN A VIRTUAL ECOSYSTEM GAME

Jonathan Baker
National Centre for Computer Animation
Bournemouth University

Abstract

This report describes several possible designs of modular neural networks for use in the AI system of a Virtual Garden Game. Neural networks have a lot potential for creating intelligent, entertaining behaviour because they can be used to simulate learning. This report will show how simple neural networks could be used to do complex tasks such as problem solving, while still addressing the need for clarity, and ease of debugging which has so far has worked against neural networks in the games industry.

Introduction

Neural networks are considered by many to be a dark art, they have hundreds of uses from weather forecasting to pattern recognition, yet underneath it can be extremely difficult to interpret how it is working, never mind what to fix when something goes wrong.

The initial motivation for this report is to explore designs of neural networks to control animals in a virtual garden game. The game is focused on maintaining an ecosystem, and so it relies on some basic adaptable animal behaviour to operate. This will establish an understanding of the fundamentals of neural networks for use in games which will then be extended by exploring some more complex designs that achieve useful intelligent behaviours, like visualisation of future events and problem solving.

The virtual ecosystem game

Any AI system will have to be tailored to the game it is in. So before talking about the development of the AI, it will be useful to know the game, its rules that the AI will exist by, and the desired behaviours of the animals.

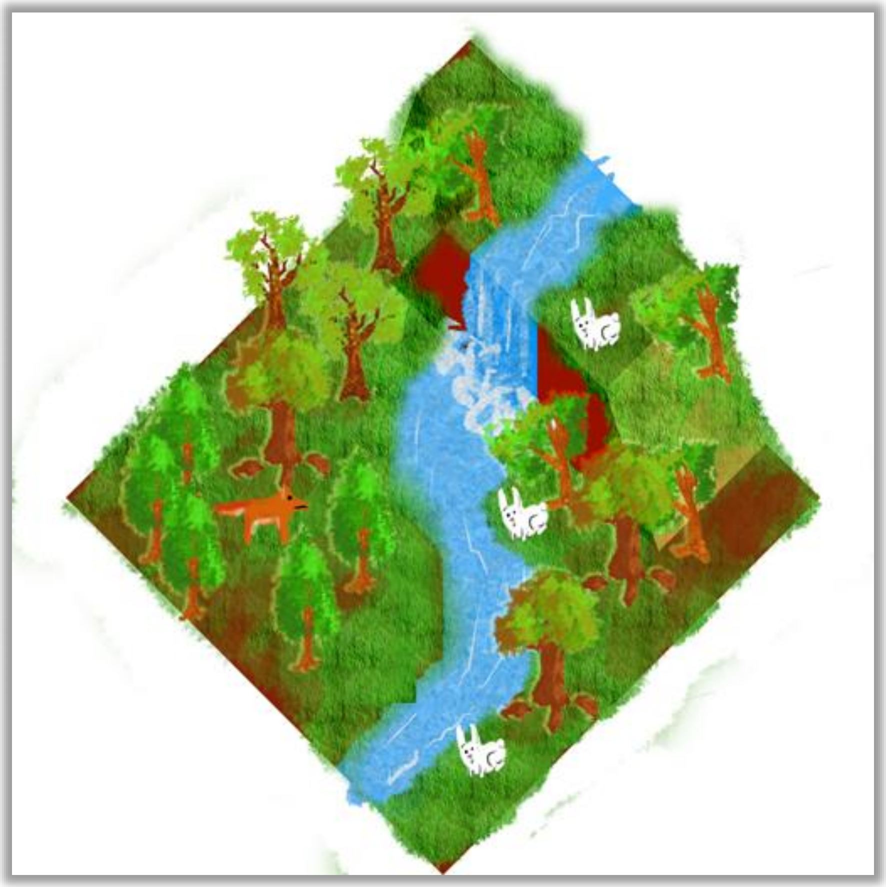


Figure 1. A concept drawing of the ecosystem game. The only parts visible to the AI for the purposes of this report is the grass, rabbits and foxes.

The user constructs and maintains an eco-system. Adding plants and animals will have adverse or positive effects as they try to build a beautiful garden. It will be similar to the Sim City games where you must create the right environment for whatever you want to live there. The user will be rewarded by feedback from the garden, letting them know how well things are doing visually, through vibrancy of colour, bouncy animation, and pleasing sound.

Players will select the right plants for certain situations. The most beautiful will require very strict, stable conditions, while there will be others that may be more dull but will grow more easily, and facilitate some of the more beautiful plants. In addition to plants that rely on other plants, there will be some that rely on animals, for example to cross pollinate and multiply. Because plants don't have brains, and their behaviours don't really change with varying environments, the plants will be governed by rules of a finite state machine, not to be decided here.

Some animals can be introduced to help control certain plants, by eating tree saplings, or pollinate others. But animal populations will work slightly differently, population numbers may increase when food is plentiful, but this will eventually cause the food population to decrease, resulting in a reduction in animal population, in turn this will allow the food population to grow again, and so on. These population swings will make it harder to create a stable environment for more difficult and beautiful plants and animals. The solution to this will be variation, to include animals that have a range of food sources, like having animals that eat animals. By having a big food chain, you increase stability of the garden. It is this stability that allows the beauty in the garden to grow, so this will be what players play for.

Achieving this stability will be a delicate process, and will take an understanding of the games rules. This is why the player will get to decide the desires of the animals, what they will eat, what will eat them, to give them the power to make a successful ecosystem. This gives a lot of control for the player to experiment, and explore the game. Understanding what makes a game tick, learning its rules, and pushing them to their limits is a big factor of what makes games fun.

AI Requirements

The AI that controls the animals needs to be able to adapt to a changing environment, where lack of food may require more desperate measures in order to survive. Or where a predator may need to learn how to take down a new prey.

For the simplification of development, I will be focusing on two animals that wouldn't be named specifically in the game, but you would expect the user to need to create animals with similar characteristics in order for the garden to grow. These are the rabbit, and the fox. One herbivore, one carnivore. One prey, one predator.

Creating correct behaviours for these two animal archetypes should be the minimum required for the game to work. This report will refer to these two animals in following examples.

Behaviours

It should be fairly obvious that the game mechanic is heavily reliant on the AI system. The absolute minimum behaviours required to have a working game are :

Rabbits eat grass Foxes eat rabbits

More advanced behaviours:

Rabbits move away from foxes
Foxes move towards rabbits
Rabbits find shelter bellow trees
Rabbits communicating, alerting of nearby danger

Rabbits grouping together
Foxes hunting together

Why Neural Networks?

Neural networks work on similar principles to animal brains, they facilitate learning, and so are inherently adaptable. Any AI system used for this game type governed completely by hard coded rules is going to be a complex one, and won't produce as entertaining behaviours.

Some understanding of what are neural networks and how they work is required to continue through the development of the AI system design.

A neural network is a function approximator.

You give it inputs, it will work out relationships between the inputs according to its training, and give an output.

For example, the virtual ecosystem game will have rules about how the animals will survive. If you eat grass, you will survive. If you are eaten by a fox, you will die. The game mechanic is the function that the neural network will try to approximate/understand. Training through feedback from the game will build an interpretation of the relationships between the variables, and the network's understanding of the game will grow, it will also adapt as the player changes rules in the environment .

A little more of a description on how they work internally is required; A neural Network constitutes of nodes called neurons. One neuron may have many weighted input connections, and require a certain threshold of activation from these input connections to fire itself. Connection weights can be inhibitory as well as excitory, so can contribute to achieving the neurons activation threshold or be restrictive. This gives the neuron some computational power.

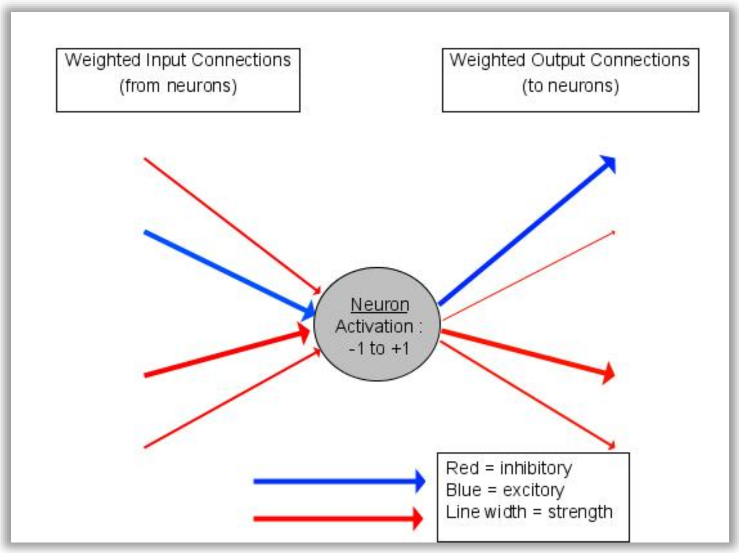

Weighted Input Connections (from neurons)
Weighted Output Connections (to neurons)
Neuron
Activation : -1 to +1
Red = inhibitory
Blue = excitory
Line width = strength

Figure 2. An isolated neuron with multiple input and output connections. These will be connected to other neurons, and activation is passed through the inputs.

So the neurons sensitivity is controlled by adjusting the weights of the incoming connections. Weight adjustment is done in training of the network. This is how a network will learn.

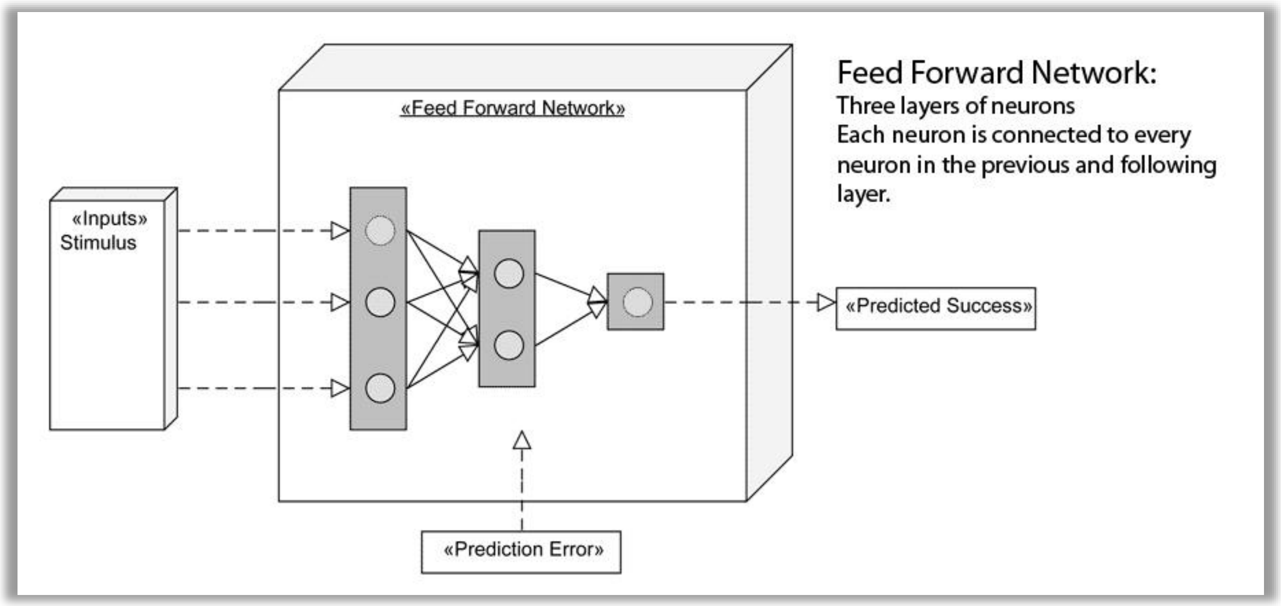


Figure 3. A Feed Forward Neural Network
The most fundamental network design, yet complex enough to handle all the tasks contained in this report.

A simple neural network can be created from layers of neurons. Three layers are enough to classify any nonlinearly separable pattern (Minsky and Papert 1969 cited Rabin 2002). For example, a three layer network can learn to replicate a XOR [exclusive OR] circuit (Masters 1993).

Adaptability is why neural networks are so suitable for this project, the rules of how to survive in the game will change, any AI written with finite rules, would be more traditional, and easier to debug partly because of the tried and tested techniques developed over the years, but the behaviour would not be as interesting. Seeing AI systems learn gives them something of an endearing quality, as if as humans we suddenly see a part of ourselves in them, and empathise with them. This adds a great deal of entertainment, and attachment to the creature, and also allows for special moments, unique to players. For an example, Barnes and Hutchens (2002) talk about Lionheads real time strategy game Black and White, in the book AI Game Programming Wisdom where you can train a giant creature;

> *"One player raised a Creature that eats only sleeping male villagers over the age of 30. The fact that a flexible learning AI enabled the player to pursue his goal with a discerning palette resulted in immense satisfaction and pride on his behalf."*

The ecosystem game is an excellent platform to experiment and explore the undiscovered, unapplied potential of Neural Networks. Even though they have been around for a long time, used in many games, companies in the video games industry are still wary of using their main advantage to its full potential.

Why Modular Neural Networks?

A neural network can be like having a little black box that you put things in, and get outputs that are sometimes expected, but sometimes are unpredictable.

So Neural Networks best feature, happens to also make it pretty unwieldy and scary for developers, which is why we still haven't seen the game AI revolution much documented raving about for the last ten years.

In discussion for this report of the viability of a Neural Networks controlling the decision making in an AI system Hilton (2008), the director at the at UK games company Free Radical Design helps to explain the industries restricted enthusiasm for Neural Networks;

> *"Being 'sub symbolic' they are hard to control / predict, and thus hard for designers to fix bugs with etc. In my opinion they're best used in games where you can explicitly give feedback and expect that the entity may choose to disobey you or not learn like you expect, e.g. god games. NN have been used in Creatures and in Black and White for exactly this* (Hilton 2008)*."*

> *"If I were to do a learning system for a non-player-character PC, I would prefer a system that is easier to debug and monitor. E.g. a rule based system which weights attached to rules affected by previous experience. That way they can always be overridden* (Hilton 2008)*."*

So from this it is important to recognise the need of debugging and visualisation of a system, not just for the development of the game project, but also with regards to its possible application in other game types, which may have a stricter margin of error and unpredictability, and so require tighter control of any learning AI system.

Previously used AI systems

Black and White previously mentioned, used a Belief-Desire-Intent based AI system for the learning of its creatures, using a mixture of AI devices. Desires were defined by single neurons with user defined inputs, Opinions were constructed by the creature on the fly as a decision tree, and Beliefs were held as attribute lists. Intentions were defined by a possible list of actions, e.g. "Attack Enemy Town", which could be broken down further into a more specific plan; "Throw stone at house".

The desires defined the animals personality, and could be changed by the player through rewards or punishments in response to its actions. The neurons that defined this are very simple neural networks, e.g. hunger was represented by one neuron, with weighted connections to game controlled variables; Low energy, tasty food, and unhappiness. The weights would then be controlled by the players feedback to the animal (Evans 2002).

This is a very good example of how a modular learning system can be constructed, in a controlled and readable way. But with this control, certain limitations are apparent;

1) The creature AI in Black and White required it to have *goals* set by the player (Goal Orientated AI). The behaviour demanded for the animals in the virtual ecosystem game are less complex, so all that is needed is for them to simply *react* to stimulus(Reactive AI). A goal orientated AI system is more complex to debug, and so Lionhead used a language orientated approach using decision trees, and specific lists of actions.
2) Evans (2001), lead designer of the AI in Black and White comments on its limitations:

   "The creature plans at the goal level, but once he has chosen a goal and a
   suitable object, he finds a suitable action for satisfying that goal by
   looking in a precompiled plan library.
   (For example, the creature knows that
   there are various ways of being destructive to an object: throwing it,
   throwing something at it, kicking it, casting a spell at it). It would be
   nice if the creature could plan dynamically how to satisfy a goal, rather
   than just looking in a list of suitable actions (Evans 2001).*"*

So returning to the question why modular neural networks, it is because any AI system has to be easily understood to be controlled. By using many small neural networks for specific tasks, and connecting them in a strict hard coded architecture, will make neural networks much more easy to understand, and so make them more controllable.

Neural Network implementation

I will be using neural networks in the decision making process to rate possible actions with a predicted success. The real success or feedback of an animals action, will be determined by the game code as a value from -1 to +1, and fed back to the animal at the beginning of the next AI cycle.

-1 dictating strong negative feedback, 0 being neutral and +1 dictating strong positive feedback.

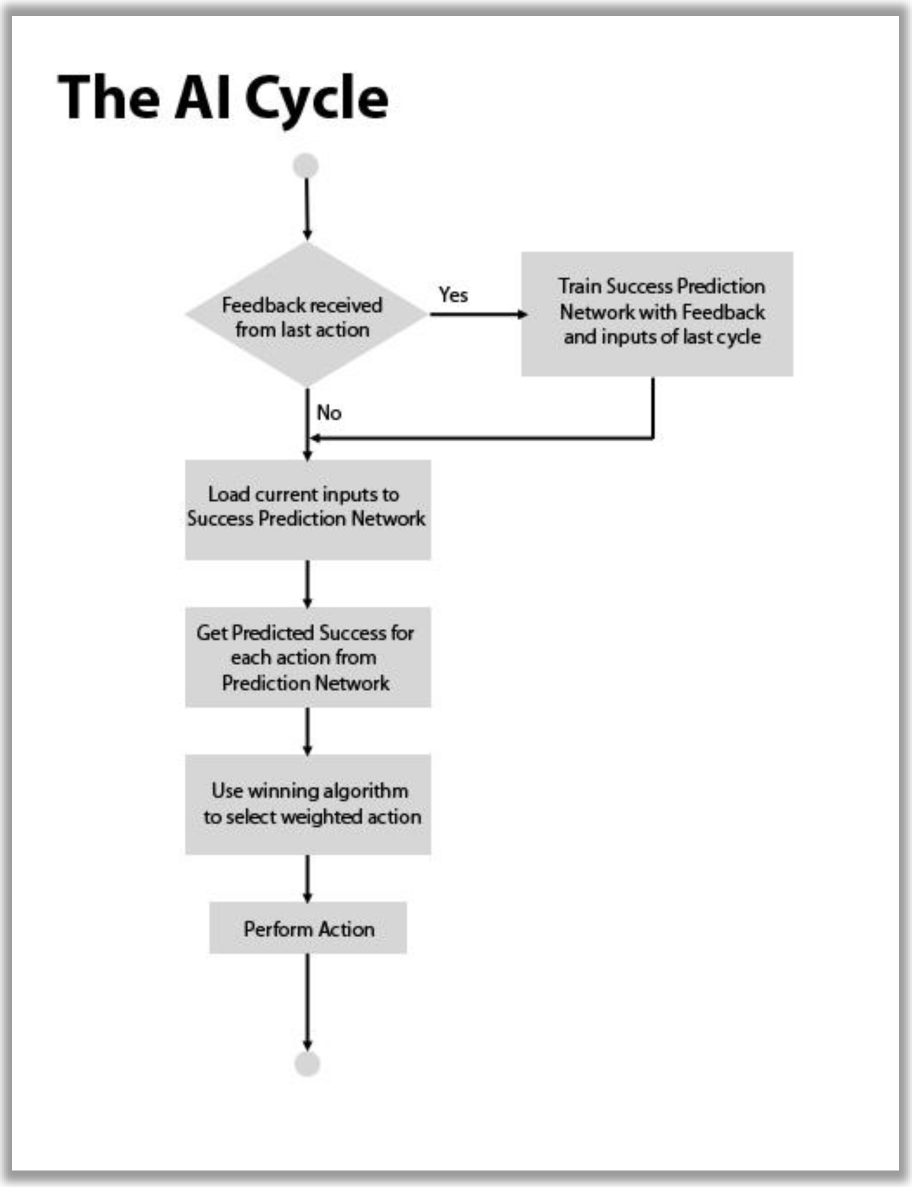

The AI Cycle
Feedback received from last action
Yes
Train Success Prediction Network with Feedback and inputs of last cycle
No
Load current inputs to Success Prediction Network
Get Predicted Success for each action from Prediction Network
Use winning algorithm to select weighted action
Perform Action

Fig 4. The AI cycle used for rating actions

This keeps the neural networks as isolated as possible, and allows you to control how much influence the network has on the decision making process.

Presenting information to the network

Because the neurons have a limited activation range -1 to +1 dictated by the activation function used, information about the environment has to be processed in a specific manner for it to contain all the information you desire important to achieving certain behaviours.

Finite Range
Convert to a percentage.
E.g. An animals health could have the range 0 to 100, so this would be normalised, 0.0 to 1.0.

Infinite Range
Normalise, so that it becomes a relative value.
E.g. Distance of an object to an animal. You still have to keep within the activation limits of -1 to +1 so it would be best to represent the distance as a relative amount, normalised to the object the largest distance away.

Boolean Values, Presence and Absence
Here is where the designer of the network has to have understanding of how the network will work, normally the presence of an object would be 1.0, but do you set absence of an object to -1.0 or 0.0? This decision will be based on the behaviour you want to see and a bit of trial and error. Generally true or false values you will want to use -1.0 and +1.0 because they are opposite, and for presence and absence you would want to use 0.0 and 1.0 because then the network will be reactive to presence, where as an activation of 0.0 does can have no excitory or inhibitory effect on the network.

Neural Network design

Minimum Behaviours

Behaviours :		Eating desirable things Moving towards desirable things
Actions available :		Eat Move will happen automatically if it doesn't eat
Environment Inputs : **(describes one square of environment)**		Grass Fox Rabbit

Using a Success Prediction Network

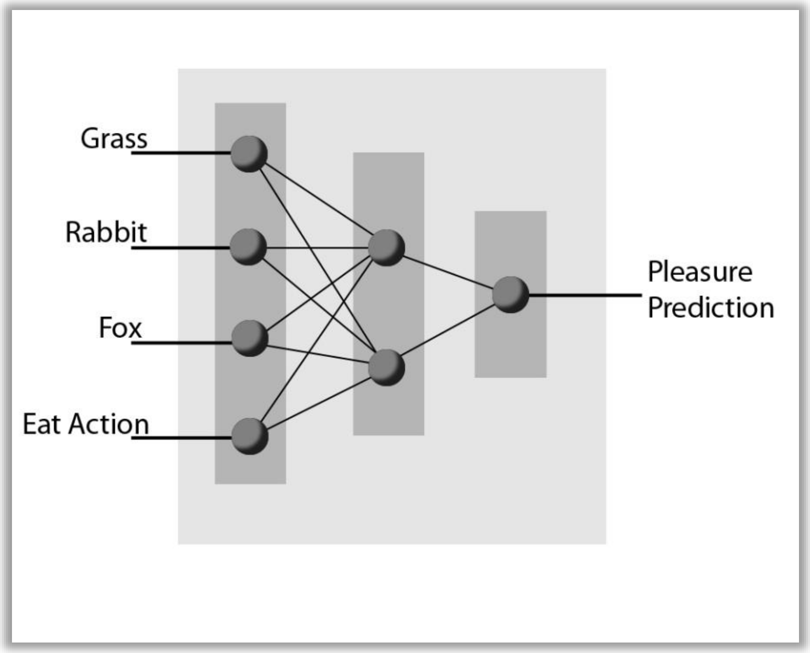


Figure 5. Diagram of the Success Prediction Network for a rabbit. Note that the number of neurons in the hidden[central] layers are only suggestive. Trial and error will dictate the best structures for each task.

Grass, rabbit and fox are all represented by 0.0 for absence, and 1.0 for presence. Eat action is represented by 0.0 for not eating, 1.0 for eating.

Which action?

There will be a Success Prediction Network trained to predict the pleasure associated with the situation. Each game cycle the squares around the animal, and the one it is currently on will be passed through the Success Prediction Network which will calculate its rating. If the winner is the current square, it will perform the eat action on that square. If a surrounding square is determined the winner, it will move to that square.

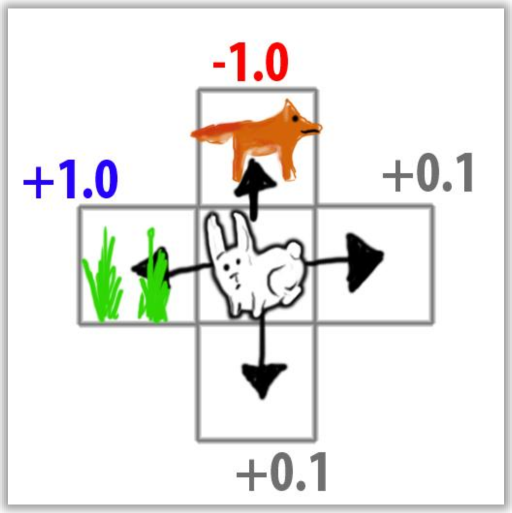


Figure 6. How the Neural Network is used to predict success for each adjacent square. This will result in a move left to the grass.

Eat what?

If it eats, it needs to decide what to eat in the square; grass/rabbit/fox. So using the same Success Prediction Network, each of the environment inputs is cycled through with the eat action input to rate the individual environment items. The animal then eats the winning environment item, and the game will return the feedback to the animal in the form of pleasure.

Learn

This pleasure is then fed back to the Success Prediction Network for a learn cycle, with the input of the action, and the item activated only (eat, grass). So the network will adjust itself to associate that item, and action with the feedback. Here I could supply the inputs of the whole square, but as the designer, I know this information is irrelevant I can remove these inputs, which will speed up learning. If I was aware of subtle relationships of the environment with how pleasurable eating grass would be, I would choose to include all the information, compromising speed of learning for accuracy, e.g. grass with mushrooms are very painful, grass without is pleasurable.

Determining winning outputs

How we determine the winning output is important. A basic highest value will suffice, though chances of getting stuck in learning ruts are almost definite. A learning rut is where one action is enforced early on, and like a human prejudice, restricts the learning of new things. This demonstrates the importance of a variety of learning methods, which will be explained later in the report.

Lack of curiosity killed the cat

We need to encourage the animal to learn new things, or it may not learn enough about its environment to survive, especially if it clings to one action linked to high amount of stimulation. So to improve on the winning algorithm stated above, investing a little curiosity in deciding the winning output will help the animal to learn faster. This is done by weighting unknown actions higher than known actions. Unknown actions would be outputs of the Success Predictions that are close to zero. As these suggest that no substantial amount of pain or pleasure is associated. The curiosity value will decide how close to zero an unknown action is.

Empathy is a survival trait

So the method described above is reactive learning, the animal performs an action, he gets feedback and associates it with his actions. This "carrot and stick" method is prone to learning ruts, where one action is enforced so strongly that he'll never have the chance to learn differently. A simple solution of adding a weighted random element to action selection was suggested above, a better solution is to learn from different methods. Evans (2002) talks about the game Black and White, where in addition to the "carrot and stick" method, creatures learned from other NPCs and from commands directed from the player to prevent the animal learning lessons that inhibit future learning.

So for this reason it is important that the animals are empathetic. Their understanding of the environment and so their survival relies on them being able to feel other animals pain, and pleasure, to broaden the number of ways they can experience things.

This would be implemented quite simply; Whenever an animal receives feedback and learns nearby to another, one can be said to be observing the other learning. So the learning animal's input, and resulting feedback would be fed into the observing animals Success Prediction Network to be learnt from.

A move towards realism here could be determining the range of learning by the extremity of feedback. E.g. A painful death of a rabbit by a fox would be seen by many rabbits from a distance, but a rabbit discovering that eating a tree is slightly unpleasant is not overly dramatic or important, and would go unnoticed at a distance.

You can argue, if you are going to have a community that learns each others lessons, why not just have one centralized Success Prediction Network, which would increase performance, and speed up learning. But this networking method allows for pockets of knowledge in the environment, which creates something interesting to watch and build on when developing AI that allows more realistic communities. In addition it also allows for misinterpretation of the world, and deception, which may not be important in my game example, but is a very intriguing and potentially useful topic.

Debugging

Debugging, an important aspect, but with so few inputs it would be very simple. At this stage understanding what is going on at low levels is fairly easy. Test inputs can be created and passed in, and even looking at the weights of the connections in the Success Prediction Network will be fairly revealing because there is only one output. You should be able to spot certain associations to pleasure and pain as long as information is returned in a graphical way, making erroneous behaviours easy to spot. However the winner selection algorithm will complicate things, so it will be necessary to be able to separate hard coded parts from the network, or restrict their complexity when debugging.

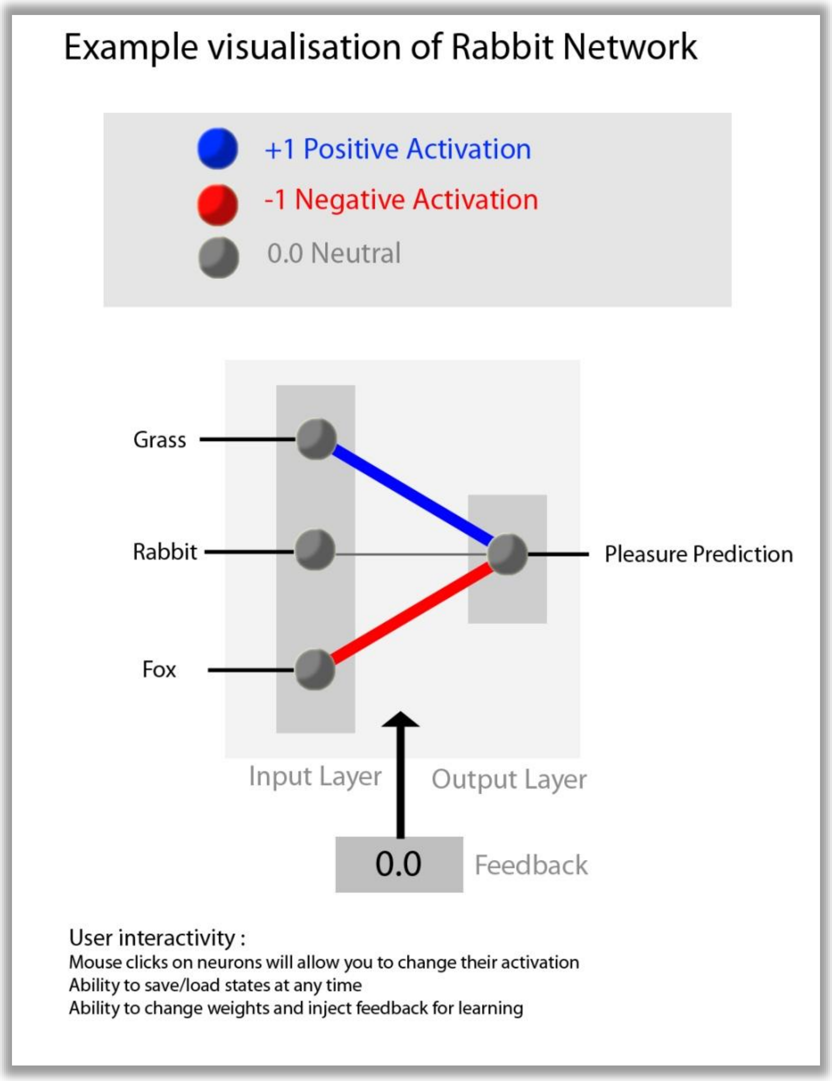

Example visualisation of Rabbit Network
+1 Positive Activation
-1 Negative Activation
0.0 Neutral
Grass
Rabbit
Fox
Pleasure Prediction
Input Layer
Output Layer
0.0
Feedback
User interactivity :
Mouse clicks on neurons will allow you to change their activation
Ability to save/load states at any time
Ability to change weights and inject feedback for learning

Figure 7. Example of visualising network, with features to ease debugging and analysis

Summary

Behaviours to expect :

Animals will move around in a random looking fashion, until they move next to something they deem desirable, which they will then move onto, and try to eat. The restricted view they have of the world may mean they go round in circles, and may get stuck in parts of the map if there is nothing desirable adjacent to them to move towards.

Positives:
-Uses more than one learning method to avoid learning ruts.
-Will move toward pleasure.
-Will be curious about new experiences from the winning algorithm.
- Easily debugged

Negatives:
-It is reactive. It is only aware of present time, so it cannot associate past actions with current situations.
E.g. If a fox is adjacent to a rabbit, it will not know to move away until fox is on his square.
-Movement is limited . It wont move away from potential pain.
-Environment awareness is limited, it may get stuck in one place.

So overall it wont look very intelligent.

Improving Movement

If the system was more advanced, and was able to form goals, this would govern the movement . But as my system is reactive, its only goal is pleasure, so the movement can be autonomous from the main Success Prediction Network as long as its design shares the same goad, to seek out pleasure .

By increasing the animals field of view, the extra information will help the animal gauge where to go more accurately, and to help prevent him going round in circles where there is no stimulation. There is also the hope that with the addition of time delayed learning introduced later on, the animal will begin to associate situations with impending pain, and attempt to avoid them, say if a fox was stalking the rabbit.

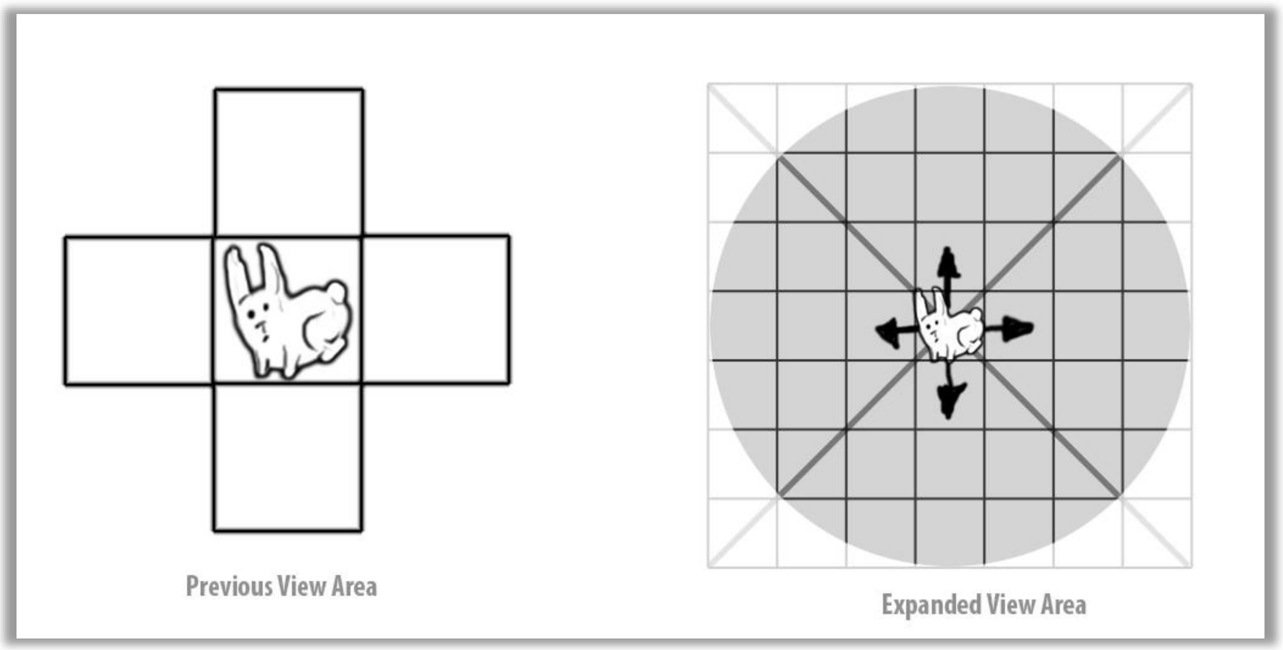


Fig 8. New expanded viewing area, you can see how much extra information needs to be presented to the network.

So the AI will work as before, comparing the current square to the potential pleasure of moving in each direction , deciding to eat if current square is rated the winner.

But the calculation of potential pleasure of moving each direction will be now have to take into account the animals extended field of view. The problem with adding extra information, is that there would be a big increase of the number of inputs if each square had to be described to the network separately, one for each stimulus : grass, fox, rabbit. This will slow learning, maybe even reduce intelligence potential because of extra local minima added because of increased complexity of the network.

The best solution in this case is to combine all the information that I think is important into one group of inputs, describing the information in a relative form rather than absolutely.

Important information for each direction:

-Type of objects

-Quantity of objects (relative)

-Distance of objects from animal (absolute)

This way when the rabbit is evaluating the four directions. If the up direction had a fox far away, but had loads of grass compared to the other directions, the pain associated with the fox will be scaled down by its distance, and wont prevent it from taking advantage of the pleasurable grass.

This method feels like a natural way of visualising the world.

Here is an example of how I will approach this:

<u>Distance :</u>

Triangle$_1$ = 0.8

Triangle$_2$ = 0.5

Triangle$_3$ = 0.5

Square$_1$ = 0.5

Diamond$_1$ = n/a (out of view)

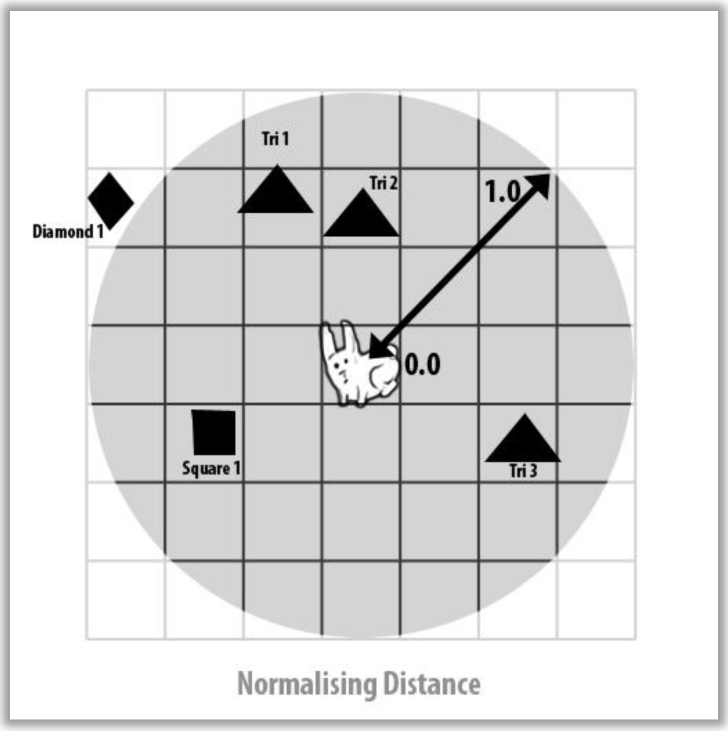


Figure 9. Normalising the distances for processing the world in a relative form. Note the diamond which will be excluded from calculation as it is out of view.

Here the distance is an absolute value, which I will explain shortly, the distance has been normalised in terms of the animals field of view, 1.0 being at the border of the viewing area, and 0.0 being the animals position. Note the diamond is outside the view, so it is not counted.

The choice of making this absolute or relative comes with benefits and drawbacks for each. It could be relative, worked out by normalising with 0.0 as the animal, and 1.0 as the object furthest away within the viewing area(rather than the view boundary itself), which would allow for greater distinction of distances if they were all close to each other, but then the distance wouldn't be a constant value from which to make associations with. For example, a fox could be dangerously close, but this would be recognised as 1.0 if the animal was surrounded by lots of other close objects and the fox happened to be the furthest out. By making it absolute, the range of view is constant, and associations on object distances can be made and relied upon, so if the fox is on the boundary of view, the animal may know that it need not act. But if the fox is on an adjacent square, the animal will know his immediacy regardless of other objects around him.

Quantity of objects:

The quantity of objects will be described relatively. This is the only approach without using numerous inputs. By finding the ratio of each object type in each direction, then multiplying each value by the average distance of each of the objects in that area, we can give a clear view of the environment, that describes quantity and distance with just one set of inputs.

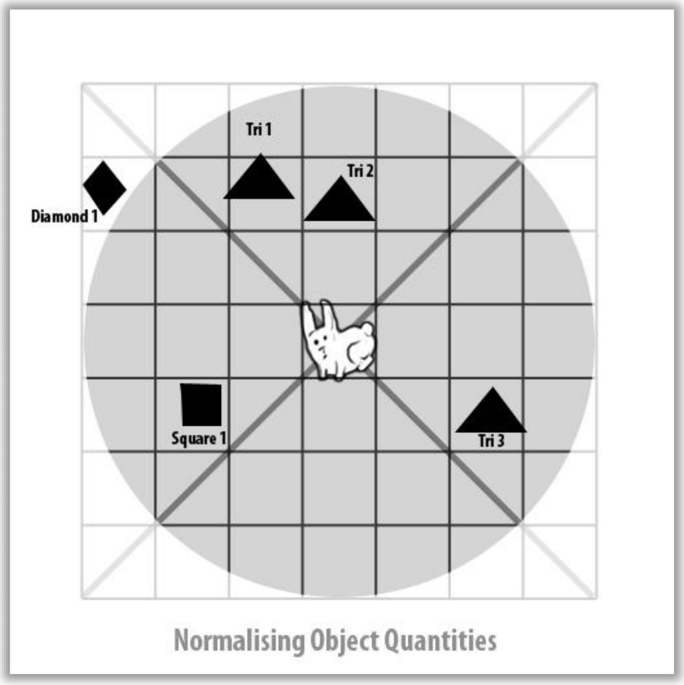


Figure 10. Normalising quantities according to possible movement direction.

Input = (no. of same objects in area)/(total no. of same objects in view) *

(1- (average distance of same objects in area $)^2$)

The average distance needs reversing because we want closer objects to have stronger activation than further away ones. The squared average value is used because it helps increase activation of objects closer to the animal, than ones further away.

Up direction

Triangles	=	$2/3 * (1 - 0.422^2)$	=	0.468 activation
Squares	=	$0/0 * 0.0$	=	0.0 activation
Diamonds	=	$0/0 * 0.0$	=	0.0 activation

Right direction

Triangles	=	$1/3 * (1 – 0.5^2)$	=	0.2475 activation
Squares	=	$0/1 * 0.0$	=	0.0 activation
Diamonds	=	$0/0 * 0.0$	=	0.0 activation

Down direction

Triangles	=	$0/3 * 0.0$	=	0.0 activation
Squares	=	$0/1 * 0.0$	=	0.0 activation
Diamonds	=	$0/0 * 0.0$	=	0.0 activation

Left direction

Triangles	=	$0/3 * 0.0$	=	0.0 activation
Squares	=	$1/1 * (1 – 0.5^2)$	=	0.75 activation
Diamonds	=	$0/0 * 0.0$	=	0.0 activation

Each direction would then be rated by passing the triangle, square, and diamond inputs through the Success Prediction Network with the animals current state(hunger, health etc) to get a predicted pleasure rating for that direction. Or in this case substitute triangle, square, diamond for grass, rabbit fox.

If one of these ratings beats the rating given for the current square, the animal will move in that direction. Otherwise it will eat on the current square.

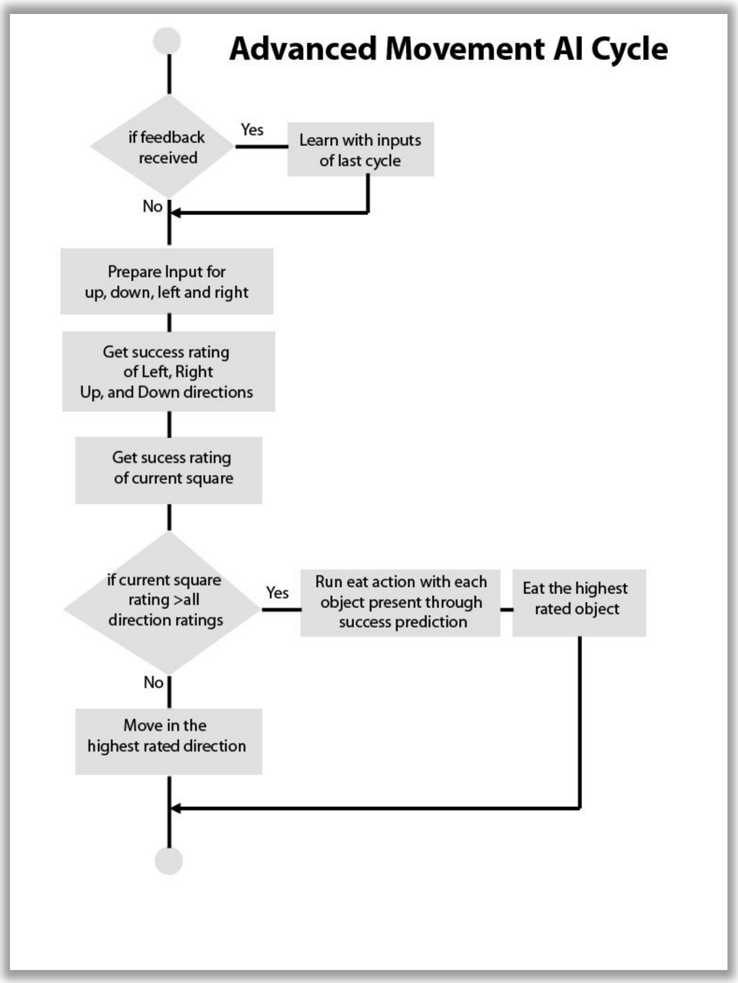

Advanced Movement AI Cycle
if feedback received
Yes
Learn with inputs of last cycle
No
Prepare Input for up, down, left and right
Get success rating of Left, Right Up, and Down directions
Get sucess rating of current square
if current square rating >all direction ratings
Yes
Run eat action with each object present through success prediction
Eat the highest rated object
No
Move in the highest rated direction

Figure 11. The new AI cycle with advanced movement. Note the extra step reuired to prepare the data.

Summary

The extended view should help prevent the animal getting stuck in places with no stimulus. It shouldn't inhibit the animal from eating either, (which relies on the current squares pleasure prediction beating the four directions) due to the quantities of the objects being dealt with in ratios, so activation of surrounding squares would be sub 1.0 and the current square will always be 1.0 or 0.0, making the current square much more stimulating than surrounding ones could possibly be, resulting in the animal always having a preference for it if there is something of worth.

Taking this idea further it would be interesting to plug an overview of the animals view into the Success Prediction Network for general decision making, just so it would be able to make associations between the general surroundings and the feedback it receives. This is less useful at the moment, as I haven't implemented any delay learning to enable the network to associate seeing a fox stalking in the distance with the pain of a fox attack several cycles later. But it would be useful in another game where feedback can come from something a distance away. E.g. a real time strategy game where archers can fire arrows from a distance. The AI would then need to be able to associate a distant view of an archer with immediate pain.

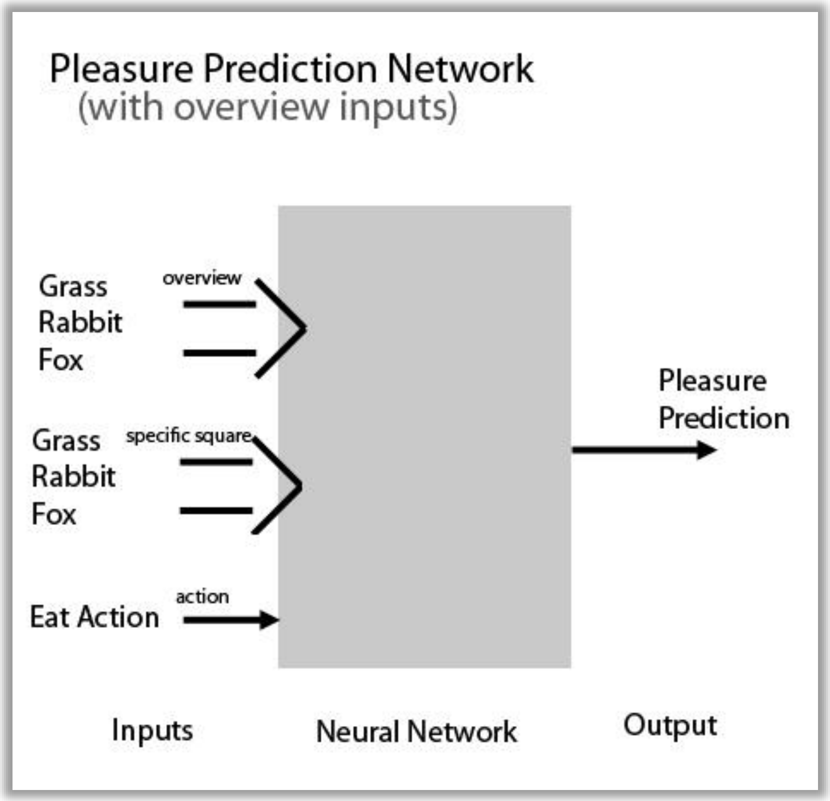


Figure 12. You can see here the overview input set would give the information of the animals viewing area. As discussed earlier, its activation values will be weaker than the specific square values which would be on or off due to the formula used for calculating the overview. This is good because we want the specific square to be more controlling of the pleasure prediction.

A limitation of this is that there is no information of objects quantity in relationship to objects of different types, the alternative is calculating the ratio of one object in a direction to the total number of objects of any type. The issue with this is that if there is a large number of one object, and a small number of another, the smaller numbered object activation will get washed out. E.g. 100 grass objects to one fox. The huge amount of grass objects will reduce the activation of the fox input, making it almost undetectable by the network. The way you could get this to work would be to have a separate network for movement, which learnt separately from the main Success Prediction Network. This could then be trained to be ultra sensitive to the fox input, and to create inhibitory weights on the grass object as it would be deemed less important.

Time delayed learning

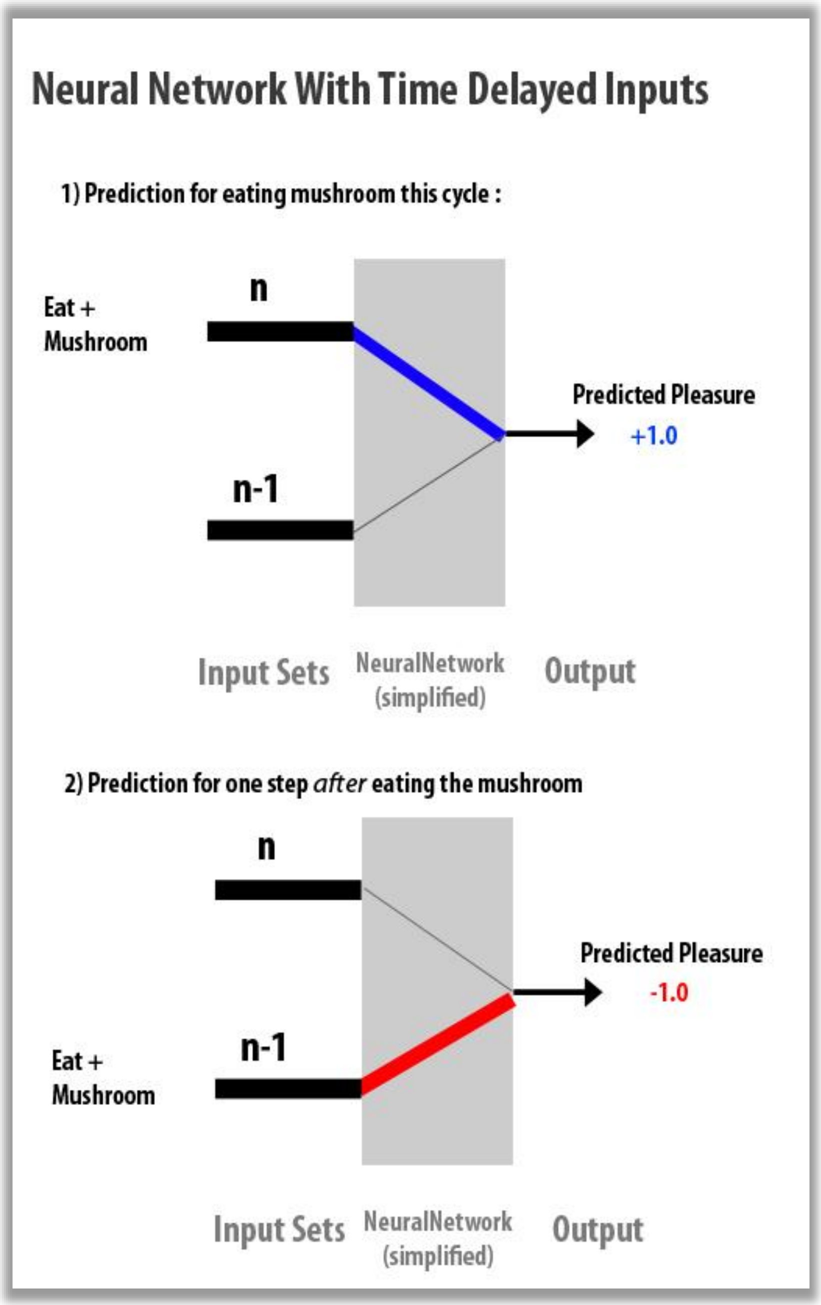
At the moment, the animal has no concept of time. If the animal encountered a wild mushroom, that when eaten tasted tasty, but on the following cycle made it sick, would the animal stop eating it before it killed itself through vomiting?

No, it would vomit a lot.

The input of “wild mushroom” would be activated when the animal is over it. It would have a neutral prediction of pleasure/pain for the item having never encountered it before. The curiosity factor would kick in, the animal would eat the mushroom. Mmmm tasty mushroom, the game will give the animal positive feedback, which the animals Success Prediction Network will use to learn, and as the “eat” action input and “wild mushroom” environment input are activated, the network will associate this with the pleasurable feedback.

The next cycle begins. The game rules say after eating a mushroom, your going to feel sick. The game communicates this to the animal with a severe dose of painful feedback. The Success Prediction Network uses this feedback to learn, but this time, its current inputs are different, the wild mushroom has been eaten, it is not present, so it is not activated, maybe the animal is eating grass this cycle. So in learning, the network associates the grass with pain. This is the wrong association. But the animal still thinks wild mushrooms are pleasurable, so we have succeeded in giving it a suicidal nature as he will now avoid grass, and seek out mushrooms.

What it needs is to be able to associate current feedback with past inputs. This is done, by duplicating the number of inputs so we can include the input set of the last cycle. This new set of inputs will be inputs_{n-1}. When learning, inputs_n will be the current cycle’s inputs, and inputs_{n-1} will be last cycles inputs. The feedback will be used as before to adjust weights, but now associations can be made between inputs_n and inputs_{n-1}.



Neural Network With Time Delayed Inputs
1) Prediction for eating mushroom this cycle :
Eat + Mushroom
n
n-1
Predicted Pleasure
+1.0
Input Sets
NeuralNetwork (simplified)
Output
2) Prediction for one step *after* eating the mushroom
n
n-1
Eat + Mushroom
Predicted Pleasure
-1.0
Input Sets
NeuralNetwork (simplified)
Output

Figure 13. Example of predicting the consequences, predicting the success one step into the future. Weights show the associations to pain(red) and pleasure(blue) that the network would learn. Input sets are just groups of inputs.

To take advantage of this, the eat and mushroom inputs are applied to the input$_n$ set as usual, and the predicted pleasure is recorded. Then the input sets are moved down one, so the eat and mushroom inputs are going in at n-1, and nothing is going into n. This will give use the predicted pleasure two steps in the future.

So if we put the “mushroom” and “eat” inputs in input$_n$, we will get a prediction of pleasure, because it was tasty. Then we look to see if it has any side effects in the future, by putting the “mushroom” and “eat” inputs into input$_{n-1}$, and it should predict pain because he was sick the last time he ate them.

The hard coded part of the AI can now evaluate these two ratings, of how it will effect the animal now, and how it will effect it in the future, and decide whether all that vomiting is worth the super tasty mushroom. Another neural network could be used, but it may be better for clarity and debugging to use hard coded methods, like averaging the estimates.

The next learning cycle the inputs have to be reset to the actual inputs he experienced and actions he took, ready for the feedback that will allow him to build on the relationships.

But this is interesting, because by having many of these delayed inputs, in theory you could use it to look many more steps into the future, and you would have the beginnings of an AI that can imagine future situations.

Summary

By adding many delayed inputs to a network you are going to slow down learning, adding inputs linearly increases the complexity of the network exponentially.

A possible solution to help would be to use neural networks to compress the data of input sets, reducing the number of inputs for each input set. But as soon as you do this, it becomes impossible to read as a human, and debugging gets a lot more difficult as a result.

It would be better to degrade the impact of the delayed input sets the bigger the time step. By degrading the activation of the input sets as they got further away from current time it would lessen their effect on the pleasure prediction, and also slow the modification of weights connected to them.

Overall, time delayed learning is invaluable device for learning. Deciding where to use this would be dictated by the game rules, for any behaviours that require associations with current time, to actions performed in the past.

The effect of using the time delayed learning with added inputs to describe the animals extended view (as explained above) will mean the rabbit’s Success Prediction Network would be able to associate an approaching fox with the pain of an attack in the future, and so move away.

Time delay learning for AI that can visualise future steps

So far the neural networks have been used to produce a scalar value, a prediction of success. This is has been used for assigning weights to proposed actions, but also, it makes the network much easier to read because there is only one output neuron, so all the weights affect it in someway and a general idea for the inputs relationships can be made just by looking at a visualisation of the network-as you can see from previous figures of example networks.

But a neural network can produce more complex outputs, and there can be some big benefits from increasing the outputs that outweigh the cost of losing readability.

For example, a network could be trained to predict the next cycles inputs, rather than just the next cycles feedback.

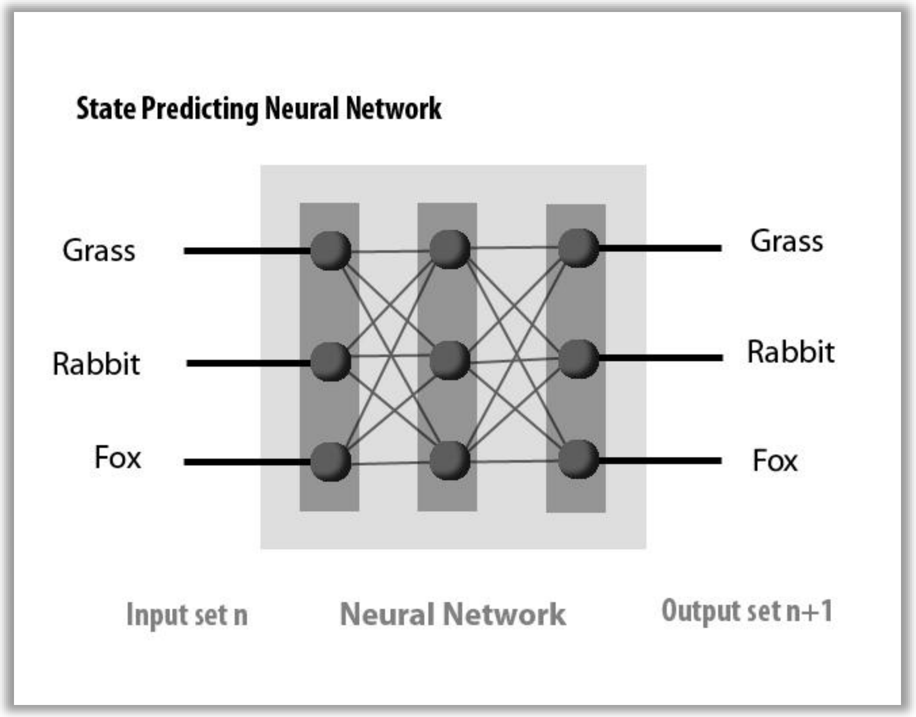


Fig 14. Predicting the next cycle's inputs, feedback would be calculated by the AI on the next cycle from how close the prediction was.

Now this is not probably going to be very accurate, or very useful with this limited input set. But there are some things you could expect to see, like when there was the presence of food, the next cycle, it would be eaten. E.g. A rabbit moving onto grass, the grass would be activated, and feedback from past experiences will have trained the network to predict that in the next step, the grass would be absent, suggesting the rabbit has eaten the grass.

Adding inputs informing the network of the animals surroundings may make these predictions more accurate, if we can supply the relative information[as described earlier regarding advanced movement]. Then it will have more information to make a prediction from. For example, if currently the rabbit was not on grass, but the surrounding area had high quantities of grass, we could assume that in the next step, there is a good chance of grass being present, as the rabbit will be likely to move onto it, and the network would be able to create these associations as it had done for pleasure prediction. It is a case of supplying it with enough related information.

We could also increase its accuracy by giving it more information about what has happened in the past :

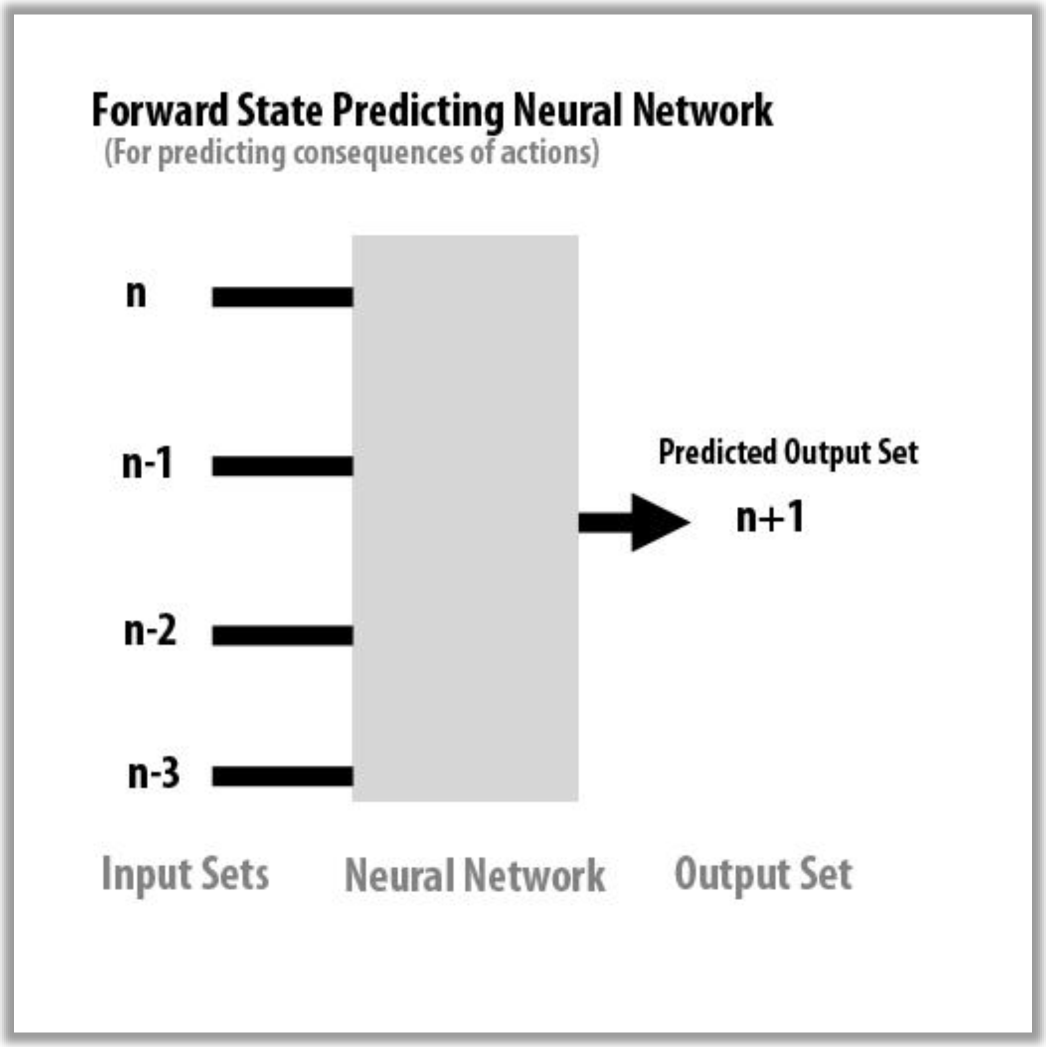

Forward State Predicting Neural Network
(For predicting consequences of actions)
n
n-1
n-2
n-3
Predicted Output Set
n+1
Input Sets
Neural Network
Output Set

Figure 15. Increasing the accuracy of a state predicting network. Note that it outputs a set of outputs the same size as the input set n.

Now we can visualise the future. We can visualise step by step, putting the current state into n, and all past states into n-1 to n-3, we get a predicted state n+1. Now we can put this new predicted state into the input n, and move the other sets down to get the prediction for n+2. If we do it again we get a prediction for n+3. And so on.

This is the animals imagination, the more it looks into the future, the more unlikely it becomes, but now the creature can plan!

E.g. A rabbit is under the shelter of a tree, it wants to move to the grass to eat, but it can see a fox a long way off. The AI has been constructing this Forward state predicting neural network along side the aforementioned Success Prediction Network which it still uses for assessing the success of immediate possible actions. Now the hard coded part of the AI system will use both these networks to assess possible moves.

The AI uses the state predicting network with the current state, and previous states to predict state$_{n+1}$. This state is then passed through Success Predicting network with all the possible actions, and the winning action is recorded, along with the predicted success recorded. Remember, no actions are carried out. This is all in the animals head.

State$_{n+1}$ is then passed through the state predicting network as per the example above, with input$_n$ moving to input$_{n-1}$, and input$_{n-1}$, moving to input$_{n-2}$ etc. Now by evaluating this network we will get state$_{n+2}$ and we can repeat to get state$_{n+3}$ and so on, each time recording the predicted success for each, and calculating the desired action at each stage.

So what ideally we would hope to see in the predictions, is that the fox will get closer as the steps progress, because it will want to eat the rabbit. This relies on the state predicting neural network to have made an association in the past, with the sight of the fox in the distance, with the eventual closing in on the rabbit to eat it. In a perfect world, the rabbit would eventually envisage the fox eating it, and predict the pain induced, and so after this planning, would predict better success staying where he was until the situation changes, e.g. the rabbit would remain in the safety of the tree until the fox had gone before heading out to the grass.

So planning may be possible with neural networks. It is emulating the animal visualising the future, using one state, to predict the next, and each time assessing how it will act itself. Finally it would use some hard coded algorithm to decide if any pain is worth the pleasure of its future actions.

The key to understanding this working is that the network isn't really "remembering" past experiences that followed each other, it is *calculating* potential experiences from the patterns and rules it created from learning the environment.

Limitations are that the number of hidden layers and hidden neurons would be very important to the success of this. Working out the right balance would be a case of trial and error. Increased number of past inputs may increase accuracy if past events have a big effect on future events. Inputs might be better spent on information about the surroundings to give the network enough information to predict events.

Reversing the technique to Problem solve

By having a starting state, we can use a neural network to predict future states, but what if we reverse this method. Instead of training a network to predict the next step, we train it to predict the last. This seems pointless, because we already know what happened in the past. But when problem solving we often start at the destination, and need to work out a way *back* to our current situation. This is where this network would come in useful.

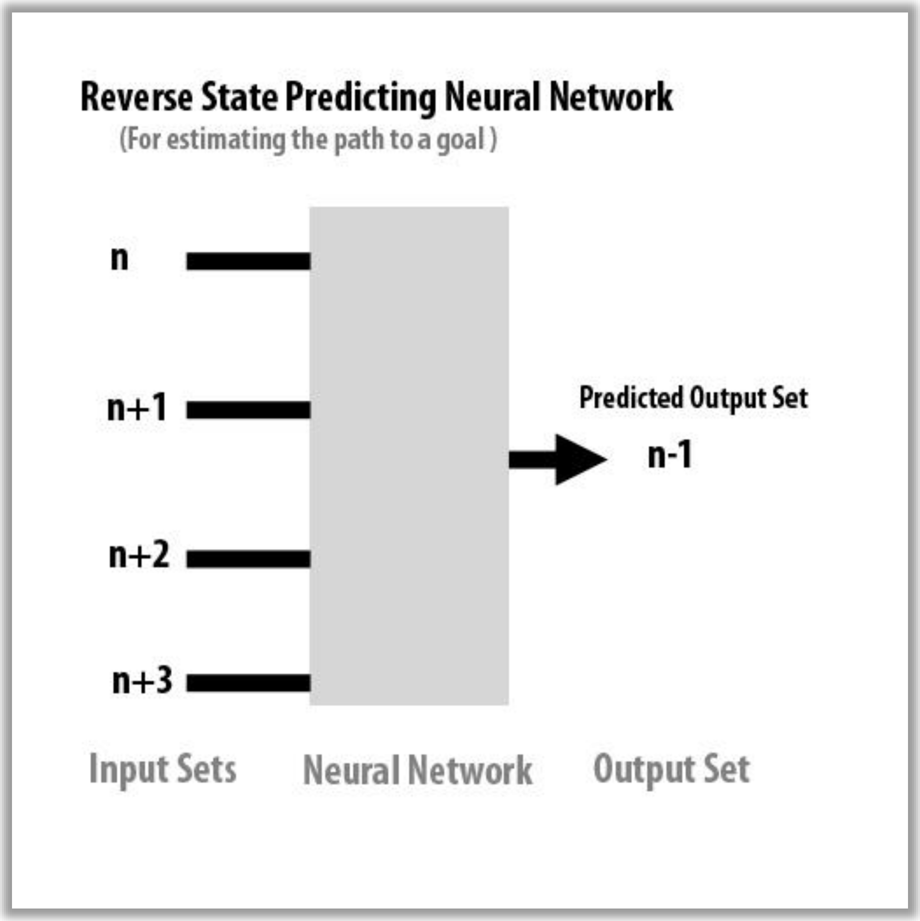

Reverse State Predicting Neural Network
(For estimating the path to a goal)
n
n+1
n+2
n+3
Predicted Output Set
n-1
Input Sets
Neural Network
Output Set

Figure 16. Training this network to predict or remember the previous state could be used in problem solving to work back to the current state. This has applications in other game types, where the user would give the AI a goal, and the AI's task would be to work out the steps to achieve it.

A desired state would be fed into input$_n$, the predicted output set would then be compared to the current state. If they are the same, this is the path to the goal. If not, the input sets would be shuffled down, input$_{n-1}$ which was just calculated would be put into input$_n$ and the output would be 2 steps from the goal instead of 1. This again, is compared to the current state, and repeated if not the same.

Summary

Both these techniques rely upon a neural network being able to be used to predict a state at a different time based on previous states. Which would in theory be possible because the networks internal connections will describe a relationship with time, the environment, and the creatures own choice of actions in the past.

These techniques have the potential to be more accurate than statistical sequential predictions, another method used in AI, where probabilities of events are used to predict. The neural networks could hold the advantage because there will be subtle relationships learnt by the network between inputs that cant be taken into account from a purely statistical model.

This is a very exciting topic, but really requires experimentation, and proof of networks being able to produce these more complex predictions, before being able to confidently expand any ideas here.

Conclusion

I have tried to develop a philosophy for an AI system that includes neural networks. Where the unpredictable parts are contained within hard structure code to tame their unpredictable nature.

The proof is in the pudding, and we shall see how well this system works in practice with my virtual ecosystem game.

There are still many things I have not been able to talk about, like animal's vitals being affecting the feedback, e.g. low hunger making eating more pleasurable, and how neural networks would use that information. I hope I have been able to share my perspective clearly enough for others to be critical of this report. I look forward to hearing other peoples thoughts on the viability of my ideas, because I find the planning and problem solving networks particularly exciting. Richard Evans talked about wanting his AI to recognise goals dynamically, and maybe this is a solution.

I struggled to find many relevant articles that dealt with neural networks on the simple level that I have applied here. The games industry can be quite secretive with regards to sharing knowledge on technology, and even more reserved about the idea of using neural networks. The main complaint comes down to the lack of control people have with them. I have tried to define a clear way of working, to enable this control, but without having tried them out myself yet, these ideas cannot be confirmed as viable. Also a lack of real world experience of working on an AI system, restricts my understanding of the needs of a system, and whether my own ideas would actually work in the end.

If I were to do it again, I would ensure I had the time to run experiments in parallel to the research, and ask more people inside the industry on their views, and reservations.

But overall all these ideas rest heavily on neural networks working as they should, and that is my biggest concern as theory and practice can differ a lot in the world of neural networks. But a more controllable factor for success with neural networks is making sure that you are supplying the network with the relevant information, and presenting it in the correct manner to allow the network to create the logical relationships you want it to understand. This is down to the designer, and his understanding of the problem and neural networks. Yes the neural network can fill in the gaps here and there, but it will an error with the fundamental design or one wrongly normalised input that will throw the whole thing into a world of unpredictability. This will ensure that neural networks instil a equal feeling awe and fear in AI designers for years to come.

REFERNCES:

Barnes, J. and Hutchens, J., 2002. Testing Undefined Behaviour As A Result of Learning. *In:* Rabin, S., ed. *AI Game Programming Wisdom*. USA: Charles River Media, 615-623.

Evans, R., 2001. *AI in Computer Games: The Use of AI Techniques in Black & White*. London: Queen Mary University of London. Available from: https://www.dcs.qmul.ac.uk/research/logic/seminars/abstract/EvansR01.html [Accessed 3 March 2008].

Evans, R., 2002. Varieties Of Learning. *In:* Rabin, S., ed. *AI Game Programming Wisdom*. USA: Charles River Media, 567-578.

Hilton, K. (khilton@frd.co.uk), 4 March 2008. *RE: Quick AI Question*. E-Mail to Baker, J., (jon_m_baker@hotmail.com).

Masters, T., 1993. *Practical Neural Network Recipes In C++*. USA: Academic Press Professional.

Rabin, S., 2002. *AI Game Programming Wisdom*. USA: Charles River Media.

Appendix:

Re: quick AI question

From: **Karl Hilton** (khilton@frd.co.uk)
Sent:
04 March 2008 15:40:38
To:
Jonathan Baker (jon_m_baker@hotmail.com)

```
Hi Jonathan,

Here are some comments from our Lead AI programmer. Hope this is useful
for you.

Karl.

My opinions on neural networks in general:

Being 'sub symbolic' they are hard to control / predict, and thus hard for
designers to fix bugs with etc. IMO they're best used in games where you
can explicitly give feedback and expect that the entity may choose to
disobey you or not learn like you expect, e.g. god games. NN have been
used in Creatures and in Black and White for exactly this.

I'm not aware of any action games that use them and I wouldn't consider
using them because of the lack of direct control we would have.

As for training them, I don't know how long it would take nor how complex
a NN you would need to get realistic and acceptable behavior for a FPS.
However, I suspect the number of times that it would need retraining when
odd behavior presented itself would become a little prohibitive. This
would need prototyping though.

The run time costs in memory and CPU would likely correspond to the size
of the network and the number of interconnections (i.e. their complexity).
As a self contained data structure, it would be ideally suited to running
on a thread or SPU in an iterative manner. It's suitability for quick
responses would be determined by how many iterations are needed before a
result node is triggered, which is again a function of its complexity. It
would probably be OK.

If I were to do a learning system for an NPC, I would prefer a system that
is easier to debug and monitor. E.g. a rule based system which weights
attached to rules affected by previous experience. That way they can
always be overridden.
```