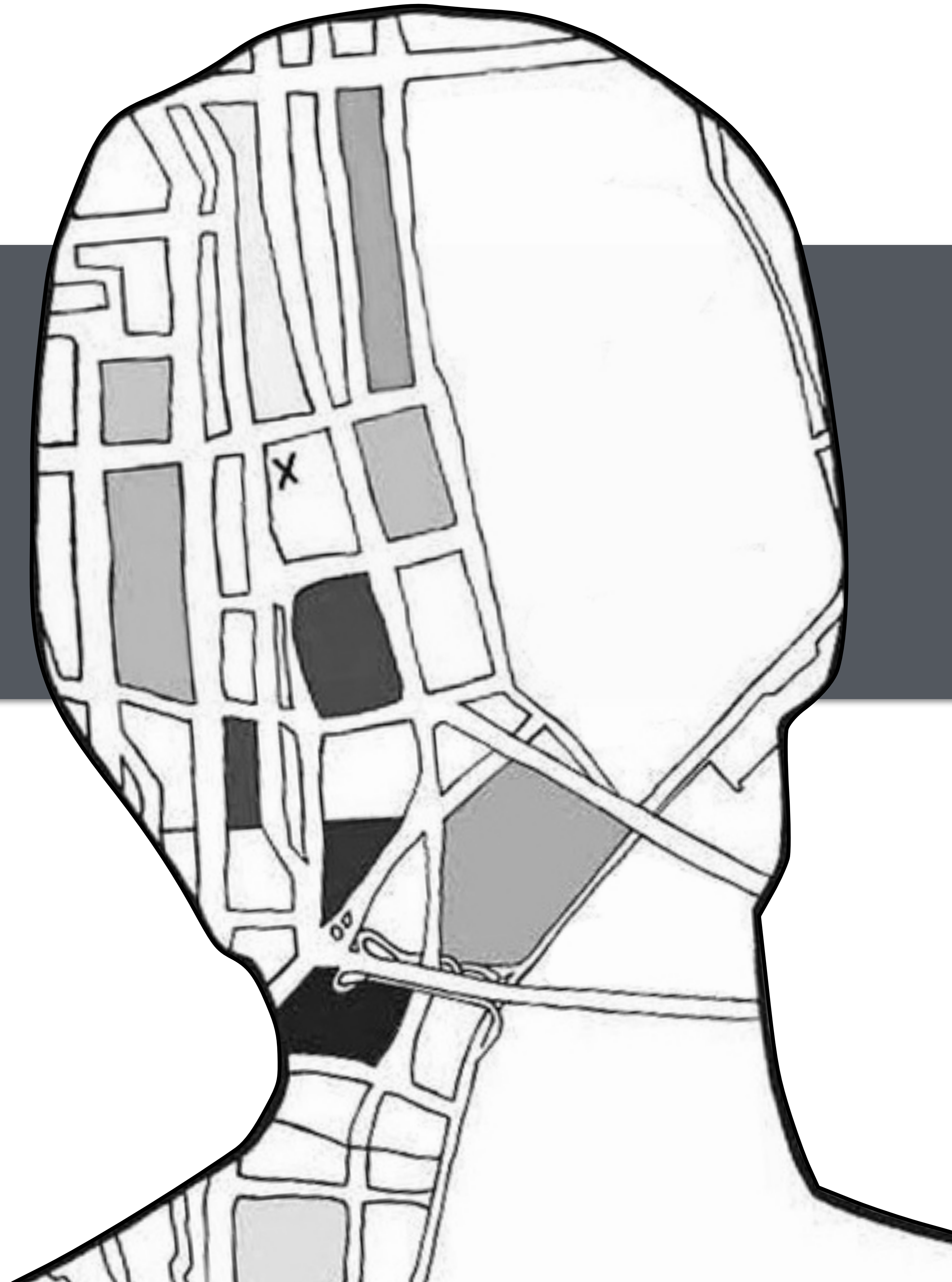
A Modified Neural Network For Adaptive User Systems

presented by **katie zaback**

let's start with a
question

how can we use computers to improve human learning?

traditional human-to-human teaching models are focused on understanding the student and how they learn best



Examples of this include…

- the study of different learning styles or multi-model learning techniques (i.e. kinesthetic, auditory, etc.)
- community-based educational reform efforts (especially in high-need or urban school settings)
- differentiated learning techniques to more adequately match student task to ability level

human-to-human teaching models focus on providing an environment tailored to a student's unique needs

the goal of these environment modifications is to achieve some pre-defined performance objective

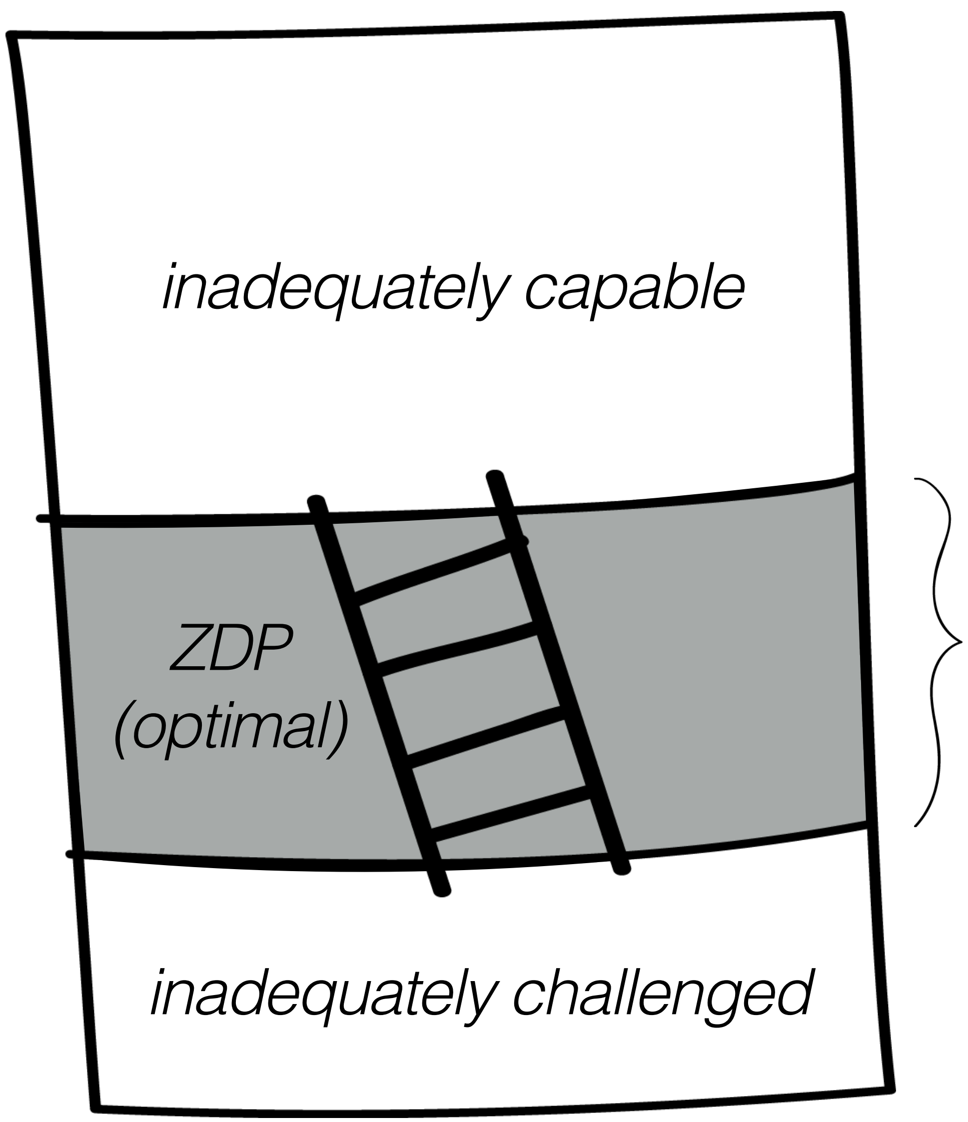
how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

what will these "unique needs" look like?

how can we dynamically change a user's environment?

Lev Vygotsky & The Zone of Proximal Development[1]



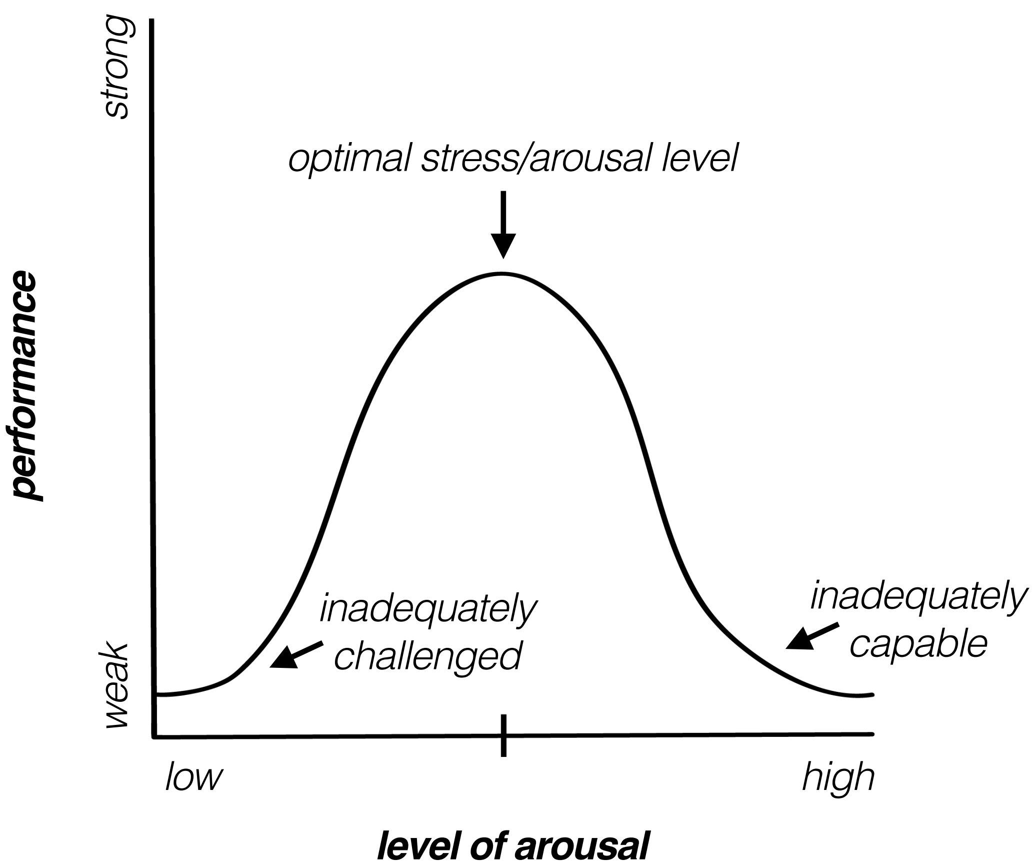
The Concept
There exists some “zone” in which a child can learn difficult material with the assistance of a teacher or peer with a higher skill set

Social Learning Theory
Vygotsky developed the social learning theory, arguing that there is a clear distinction between what a student can do on their own (independently), and what they can do with help (socially)

Scaffolding is a term related to the ZDP, which describes what assistance a student may need from a teacher or peer in order to succeed in mastering some concept or skill

1. Vygotsky, L. (1978). Interaction between learning and development. Readings on the development of children, 23(3), 34-41.

Yerkes-Dodson Law[2] & Optimal Stress Levels[3]



The Concept

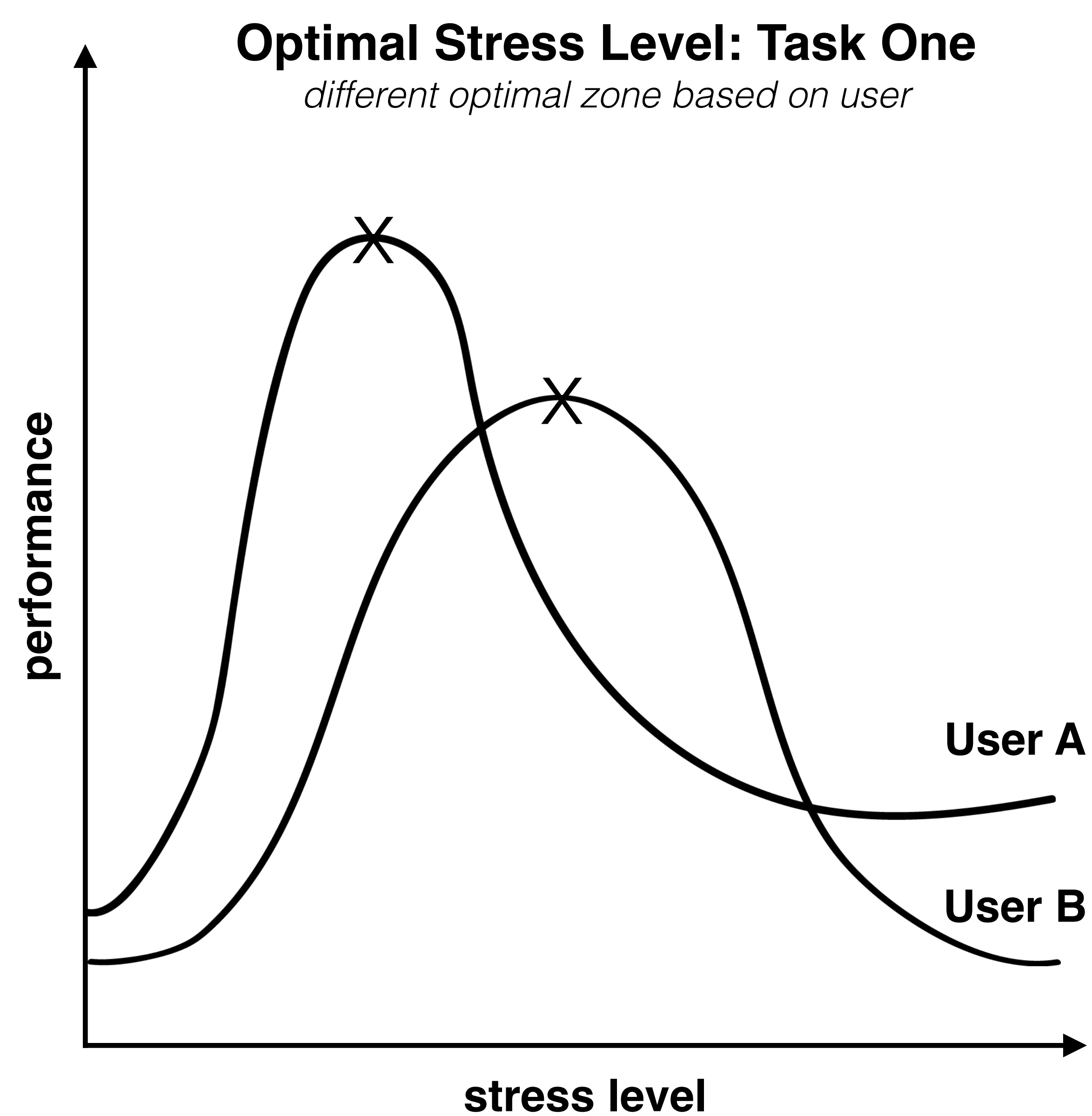
Tasks require some varied level of "arousal" in order to induce optimal performance. Dodson studied the "relation of strength of stimulus to rapidity of learning"[1] arguing that there is some optimal level of stimulus or arousal that will elicit the highest performance.

Stress Hormones and Neurology

A recent study[2] has drawn a connection between the Yerkes-Dodson curve and the production of glucocorticoids in the body. Lupin et al. observed that the level of stress hormones produced in the brain corresponded to an optimal level of memory performance, shown in an upside-down U curve similar to the Yerkes-Dodson curve.

2. Yerkes, R. M., & Dodson, J. D. (1908). The relation of strength of stimulus to rapidity of habit-formation. Journal of comparative neurology and psychology, 18(5), 459-482.
3. Lupien, S. J., Maheu, F., Tu, M., Fiocco, A., & Schramek, T. E. (2007). The effects of stress and stress hormones on human cognition: Implications for the field of brain and cognition. Brain and cognition, 65(3), 209-237.

this “optimal zone” is dynamic — it is different for every user & task and changes over time



Dynamic Optimal Zones — The User

Each user will have their own unique optimal "zone" in order to achieve maximum performance.

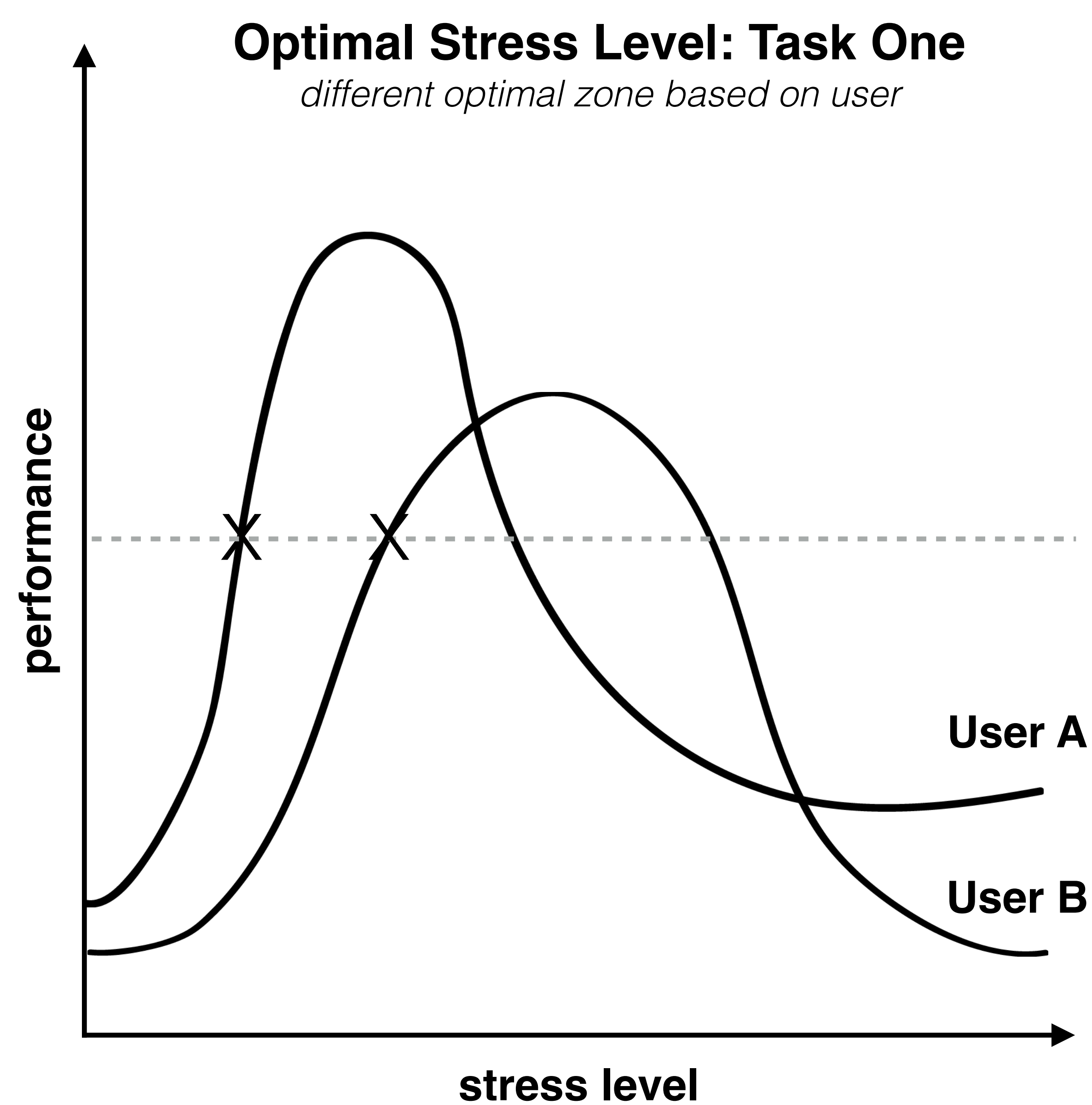
As shown here, User A and B require different stress levels in order to achieve their maximum performance. This graph also tells us that User A and B would require *different levels of stress in order to achieve the same performance objective*.

1. Yerkes, R. M., & Dodson, J. D. (1908). The relation of strength of stimulus to rapidity of habit-formation. Journal of comparative neurology and psychology, 18(5), 459-482.

Dynamic Optimal Zones — The User

Each user will have their own unique optimal "zone" in order to achieve maximum performance.

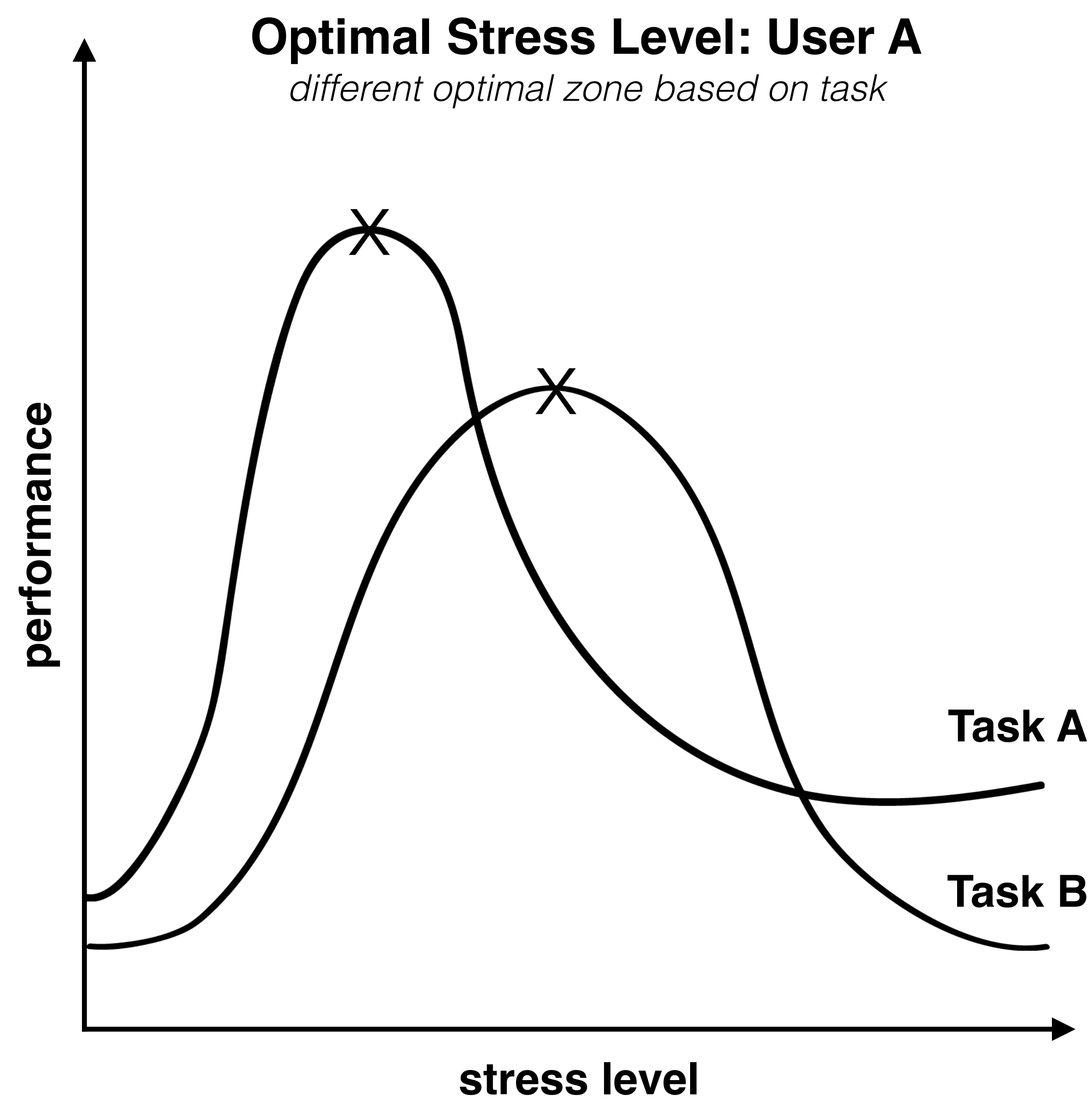
As shown here, User A and B require different stress levels in order to achieve their maximum performance. This graph also tells us that User A and B would require *different levels of stress in order to achieve the same performance objective*.



1. Yerkes, R. M., & Dodson, J. D. (1908). The relation of strength of stimulus to rapidity of habit-formation. Journal of comparative neurology and psychology, 18(5), 459-482.

Dynamic Optimal Zones — The Task

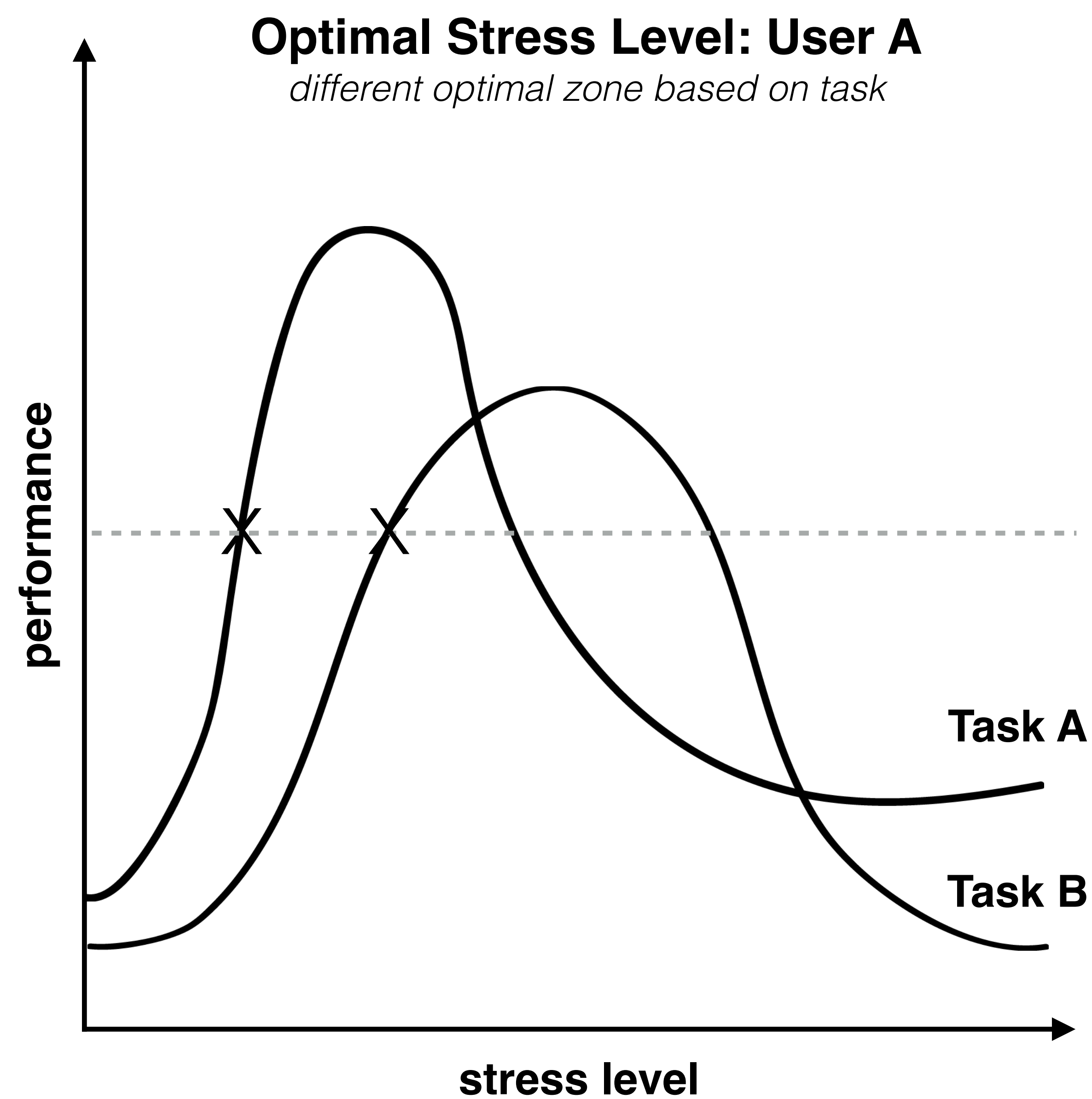
These graphs will not only differ by user, but by task. As shown here, User A requires different stress levels in order to achieve their maximum performance on two different tasks. This graph also tells us that User A would require *different levels of stress in order to achieve the same performance objective on two different tasks*.



1. Yerkes, R. M., & Dodson, J. D. (1908). The relation of strength of stimulus to rapidity of habit-formation. Journal of comparative neurology and psychology, 18(5), 459-482.

Dynamic Optimal Zones — The Task

These graphs will not only differ by user, but by task. As shown here, User A requires different stress levels in order to achieve their maximum performance on two different tasks. This graph also tells us that User A would require *different levels of stress in order to achieve the same performance objective on two different tasks*.



1. Yerkes, R. M., & Dodson, J. D. (1908). The relation of strength of stimulus to rapidity of habit-formation. Journal of comparative neurology and psychology, 18(5), 459-482.

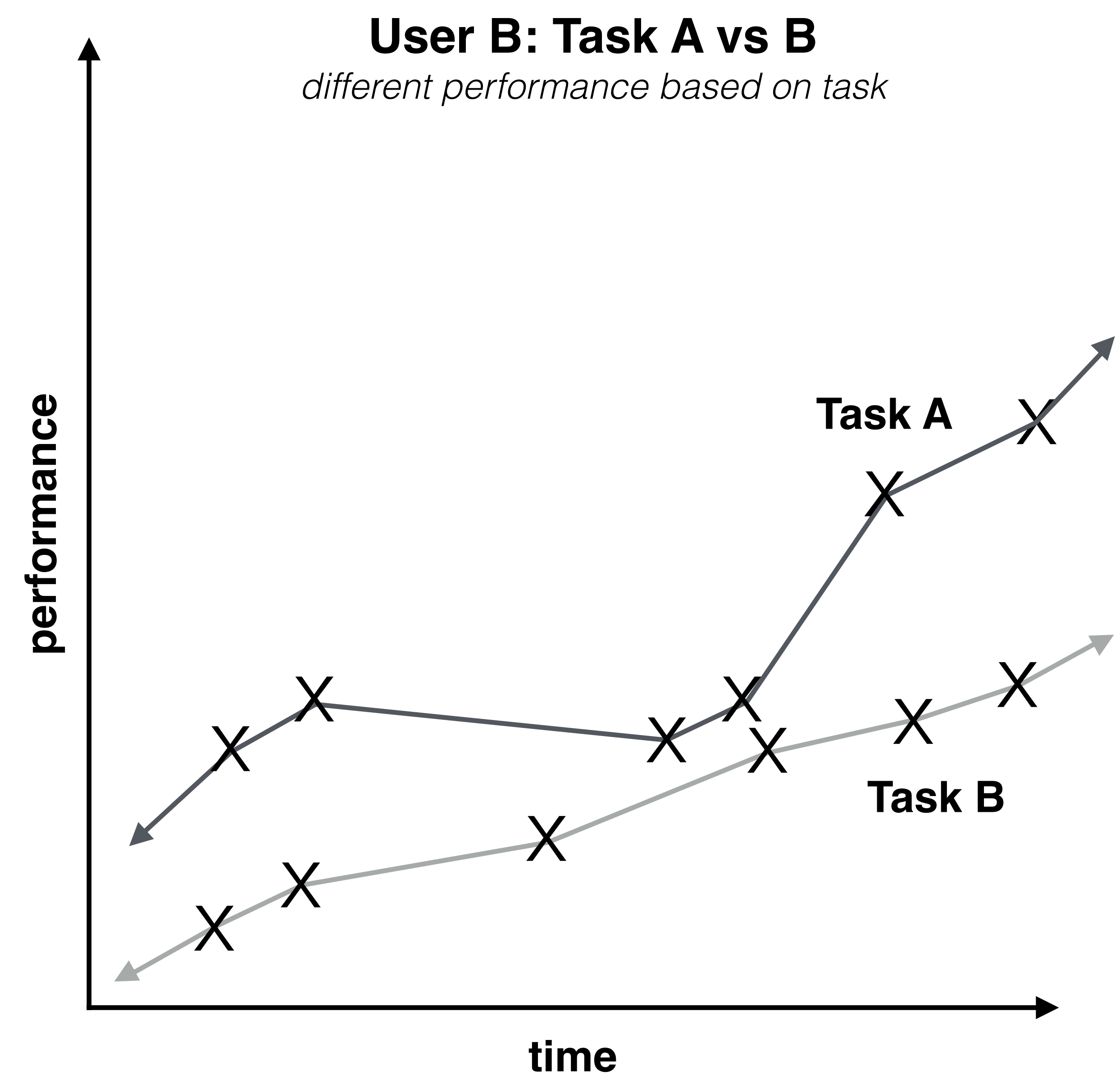
Dynamic Ability Over Time

If we graph user performance in relation to time, performance will rise and fall depending on the current ability of that user. This graph will be unique to each user for each task.

Factors that may influence performance include:

- existing knowledge related to the task
- amount of sleep or task preparation
- user's activity prior to the task (i.e. mental fatigue)
- amount of practice the user has logged for a task
- numerous other declarative user characteristics

Therefore, a single user will require *<u>different stress levels to achieve the same performance objective on the same task depending on when the task is being performed</u>*.



how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

what will these "unique needs" look like?

how can we dynamically change a user's environment?

how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

what will these "unique needs" look like?

dynamic
user-dependent
environment-dependent
time-dependent

how can we dynamically change a user's environment?

how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

what will these "unique needs" look like?

dynamic
user-dependent
environment-dependent
time-dependent

how can we dynamically change a user's environment?

Randall Davis, KR & The Intelligent "Surrogate"[4]

```
user(mary).
environment(game).
threshold = n / time.

has_slow_reaction_time(X, Y) :-
      user(X),
      game(Y),
      threshold > 5.

make_environment_easier(X, Y) :-
      user(X),
      game(Y),
      has_slow_reaction_time(X).
```

The Concept

Davis describes knowledge representation as playing the fundamental role of "surrogate" — an entity that serves to bridge the gap between internal reasoning and the external (usually physical) world. He argues that "any intelligent entity" must deal with this fact: the dichotomy of the internal and the external, and that one key role knowledge representation must serve is that of a surrogate for physical world in order to allow internal reasoning to persist.

Applying KR to the Problem Space

In order to use the traditional KR approach, we would start by obtaining and representing declarative information about the user and environment (i.e. the external world). From there, we would establish sets of rules about what actions to take for that specific user/environment (i.e. internal reasoning). What information we gather, and what we would do with that information, would be decided by the individual programmer before launch.

4. Davis, Randall, Howard Shrobe, and Peter Szolovits. "What is a knowledge representation?." AI magazine 14.1 (1993): 17.
5. Gelfond, Michael, and Yulia Kahl. Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach. Cambridge University Press, 2014.

Randall Davis, KR & The Intelligent "Surrogate"[4]

user knowledge base

psychographic factors
values, attitudes, interests
demographic information
SES, ethnicity, age
behavioral practices
# of jumps/sec, past quizzes completed
other personal information
user-generated text snippets, etc.

environment knowledge base

structure/layout of task space
size of the interactive room, available quizzes
constraints of the system
speed cannot exceed 50 m/s, etc.
behavioral context
replied in 15 sec, text posted @ 11:48 p.m.
other system information
current size of obstacles, speed of enemies, etc.

Rejecting The Traditional Approach

This implementation has several attributes that make it an unfit solution to this problem:

1) *Lack of programmer omnipotence*
   - the programmer must define the rules of the system before launch, and the complexity of the user and their interactions with the environment would be incredibly difficult (if not impossible) to predict
2) *Domain/Environment specific*
   - the rules/system logic (in order to be effective) must be specific to the environment, requiring each environment to essentially have its own unique system that must be programmed by hand
3) *Storage & Memory problems*
   - the program would by necessity need to constantly store large amounts of information about the user & environment, and would need to perform computationally intensive search-space algorithms in order to run

4. Davis, Randall, Howard Shrobe, and Peter Szolovits. "What is a knowledge representation?." AI magazine 14.1 (1993): 17.
5. Gelfond, Michael, and Yulia Kahl. Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach. Cambridge University Press, 2014.

reframing the question

how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

what will these "unique needs" look like?

how can we dynamically change a user's environment?

dynamic
user-dependent
environment-dependent
time-dependent

how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

what will these "unique needs" look like?

dynamic
user-dependent
environment-dependent
time-dependent

how can we dynamically change a user's environment?

knowledge representation is not a good solution to this problem because it fails to accommodate these attributes of the user's unique needs

how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

what will these "unique needs" look like?

dynamic
user-dependent
environment-dependent
time-dependent

how can we dynamically change a user's environment?

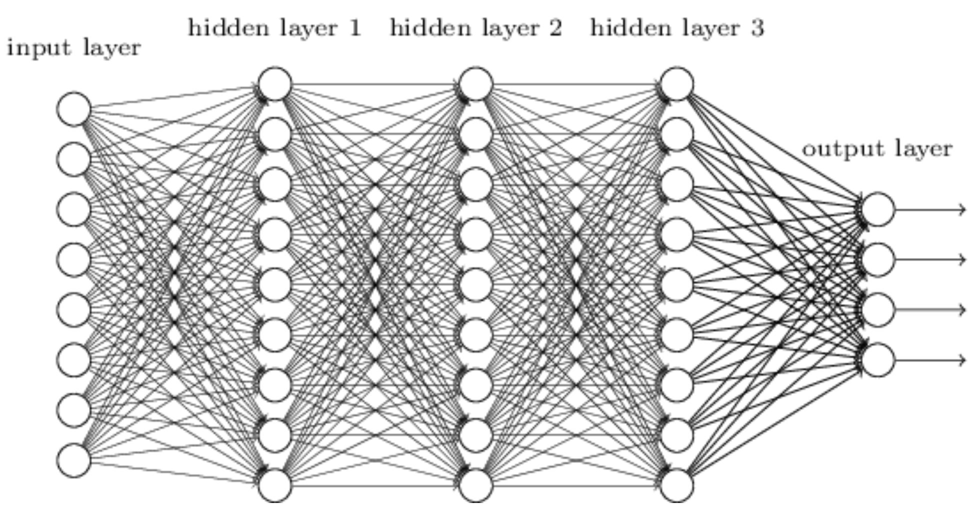
we need a system that can adapt to these dynamic specifications.

what model might be a good fit?

this “optimal zone” is dynamic — it is different for every user & task and changes over time

we can view this “optimal zone” as a point on a graph (i.e. performance as a function)

LeCun[6], Sutskever[7] & Friends: Deep Learning



http://neuralnetworksanddeeplearning.com/images/tikz36.png

The Concept

A network of computational “neurons” can be combined with error-minimizing algorithms in order to approximate complex functions not possible through simple imperative programming. This network can “learn” through supervised learning — by giving the network inputs and expected output and then using back propagation in order to adjust the values of the network matrices.

Matrix Representation

These networks can be represented as a series of matrix operations, making it feasible to engage in “deep learning” (i.e. a network that contains a large amount of layers and neurons). There exist different learning libraries that are designed to efficiently implement these network operations (such as Theano or Tensorflow).

6. Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov 1998.
7. Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems. 2014.

how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

what will these "unique needs" look like?

dynamic
user-dependent
environment-dependent
time-dependent

how can we dynamically change a user's environment?

we need a system that can adapt to these dynamic specifications.

would a neural network be a good fit?

how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

what will these "unique needs" look like?

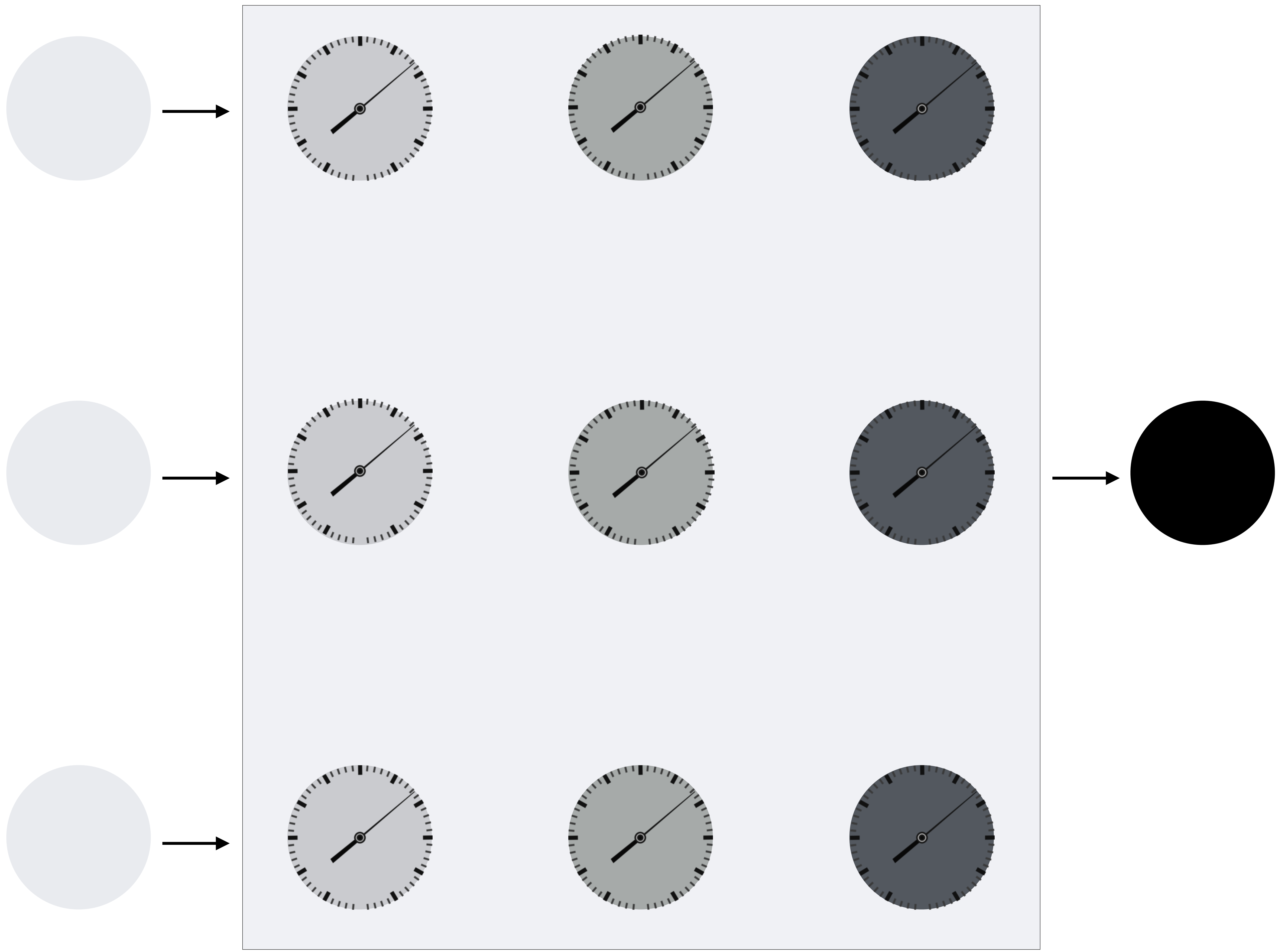
dynamic
user-dependent
environment-dependent
time-dependent

how can we dynamically change a user's environment?

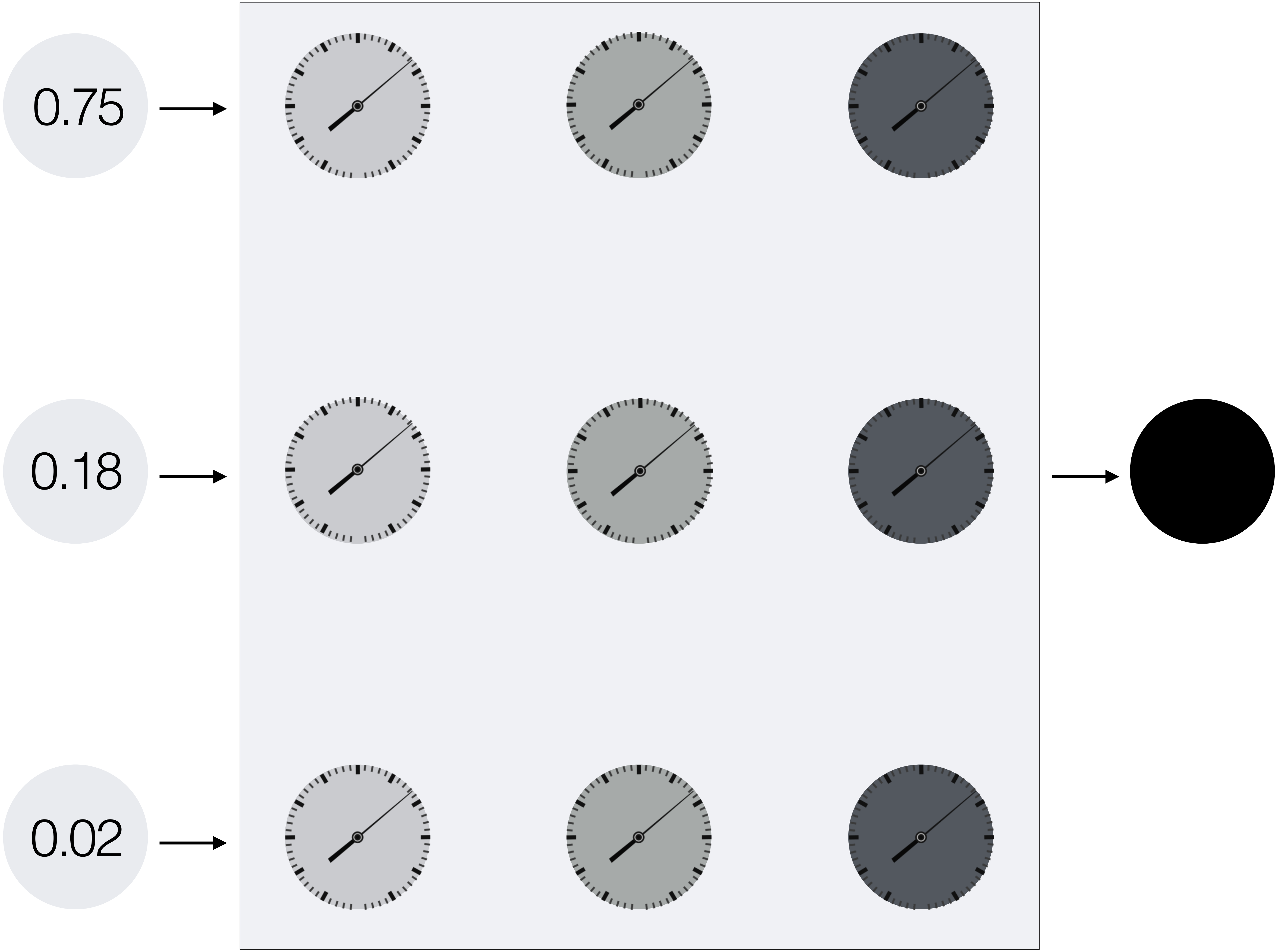
we need a system that can adapt to these dynamic specifications.

how does a NN work?

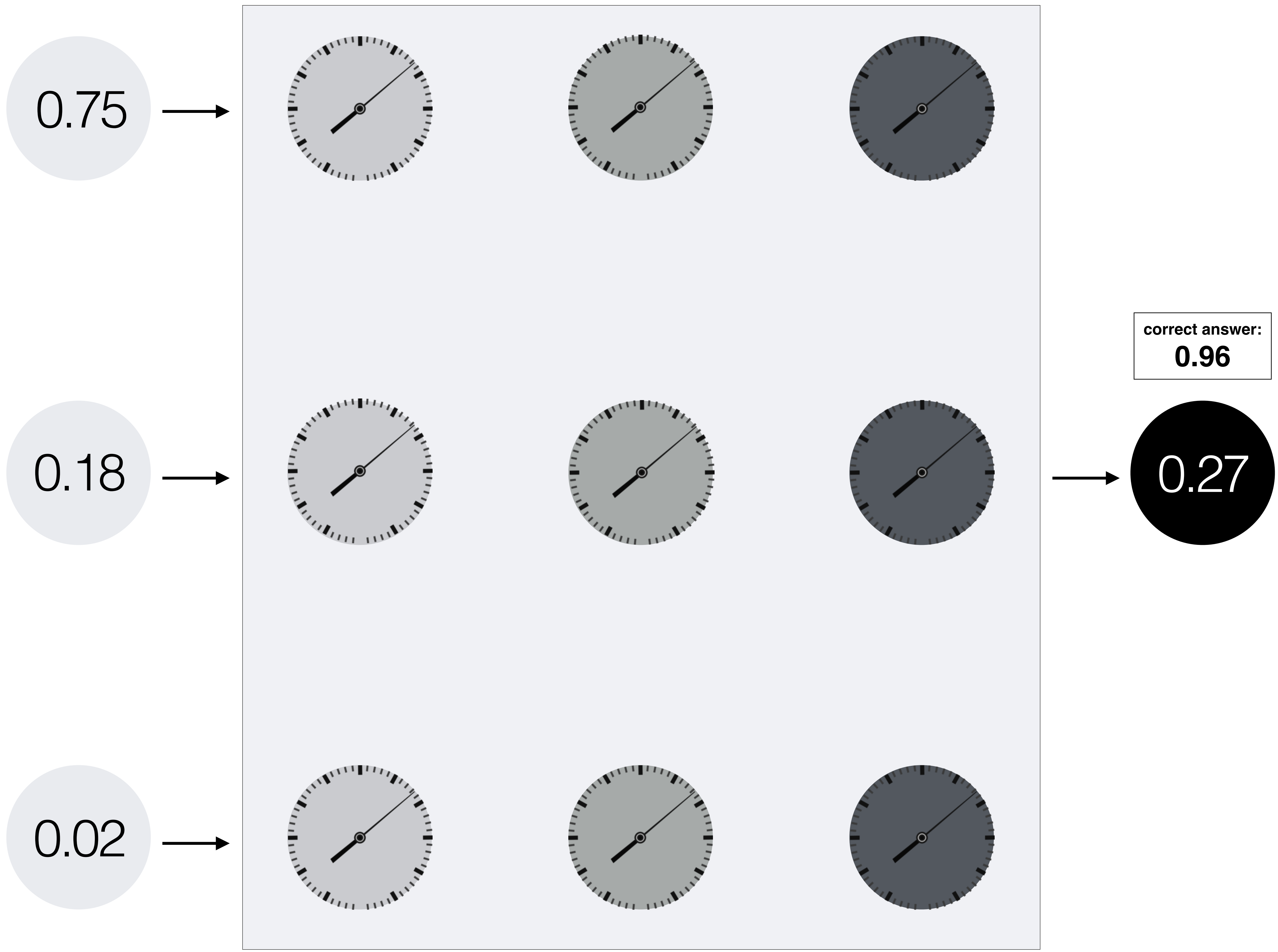
concept: neural networks & a simple metaphor



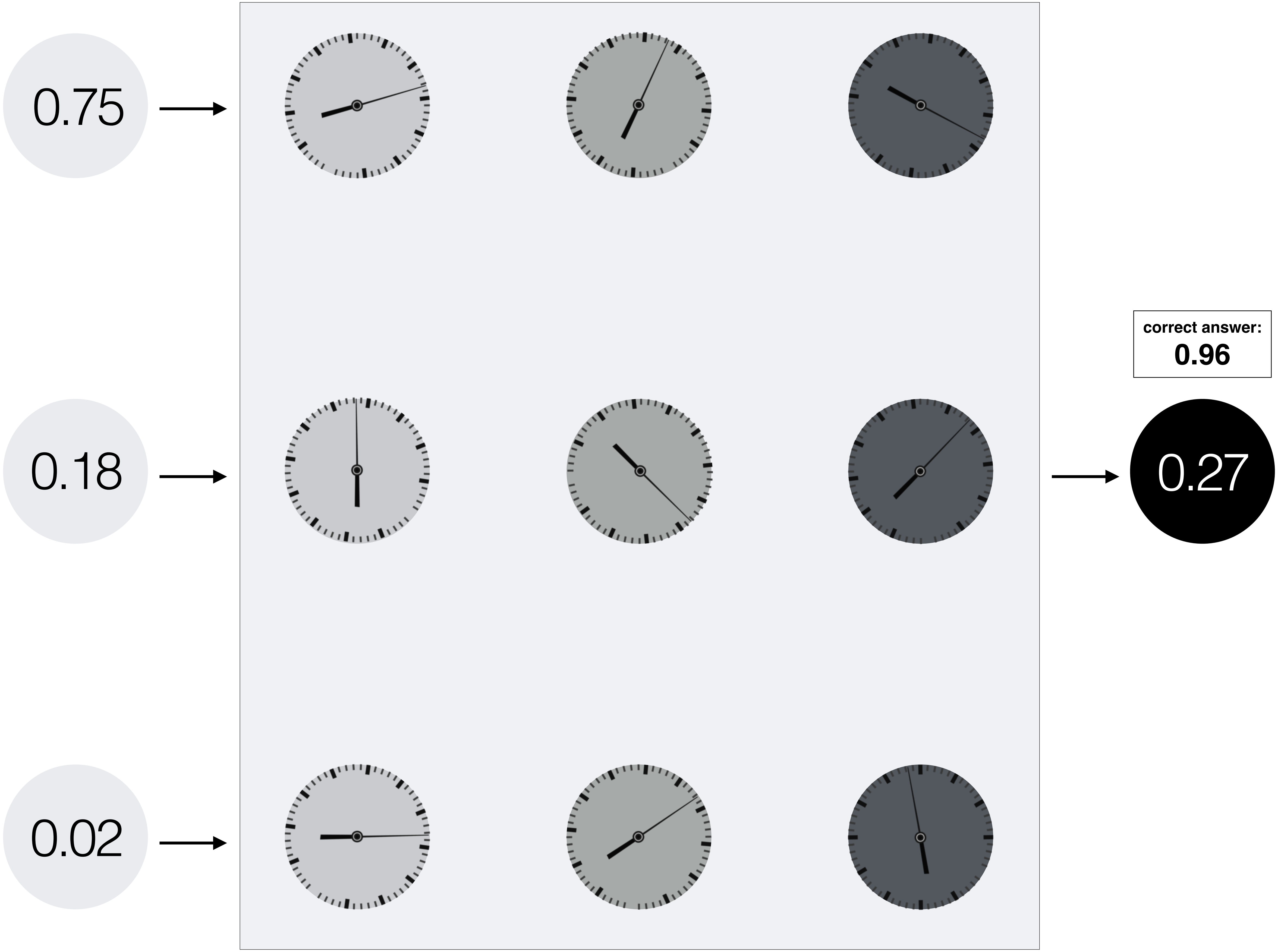
#1 Feed some inputs into the machine and see what it outputs



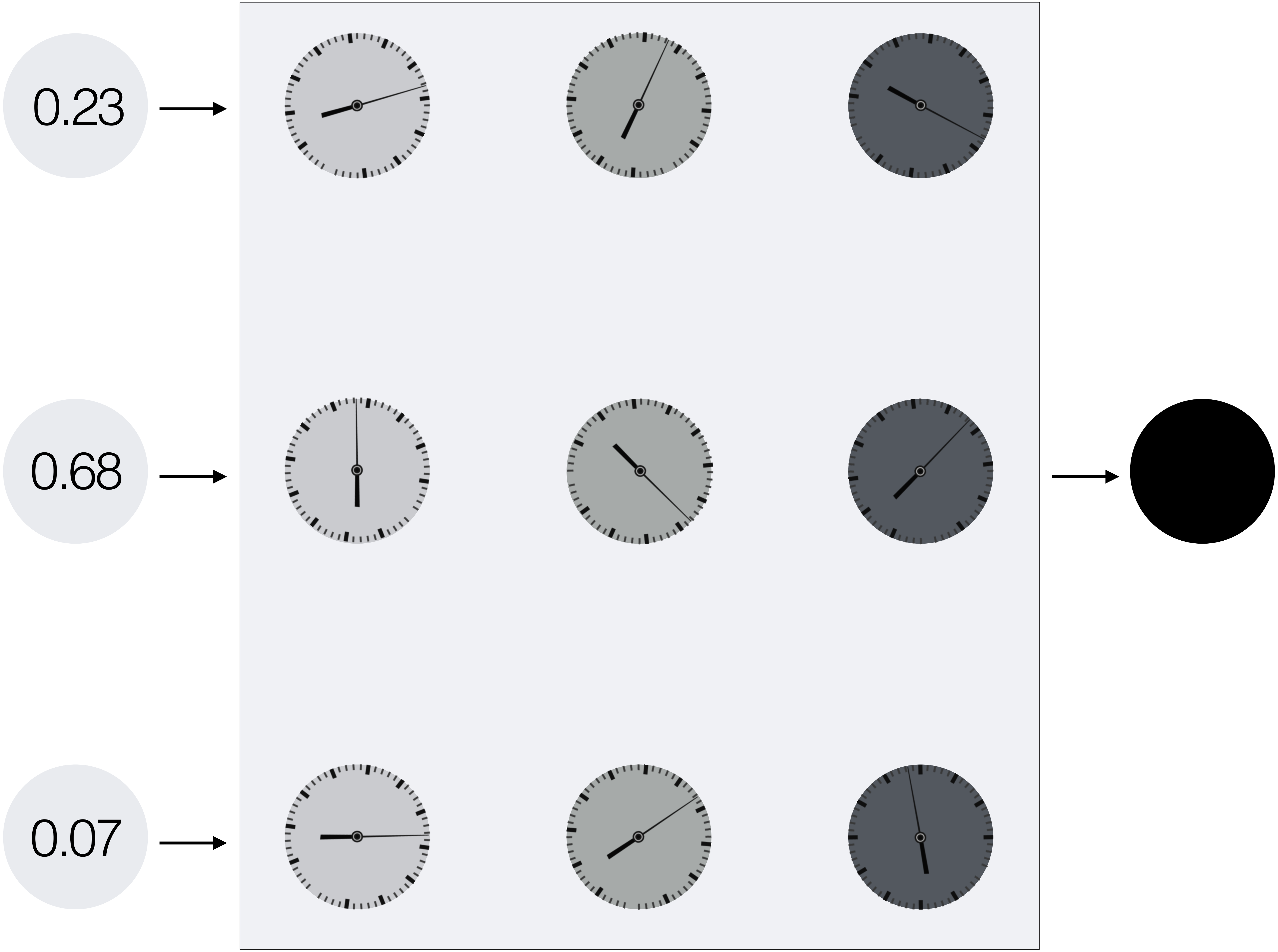
#1 Feed some inputs into the machine and see what it outputs



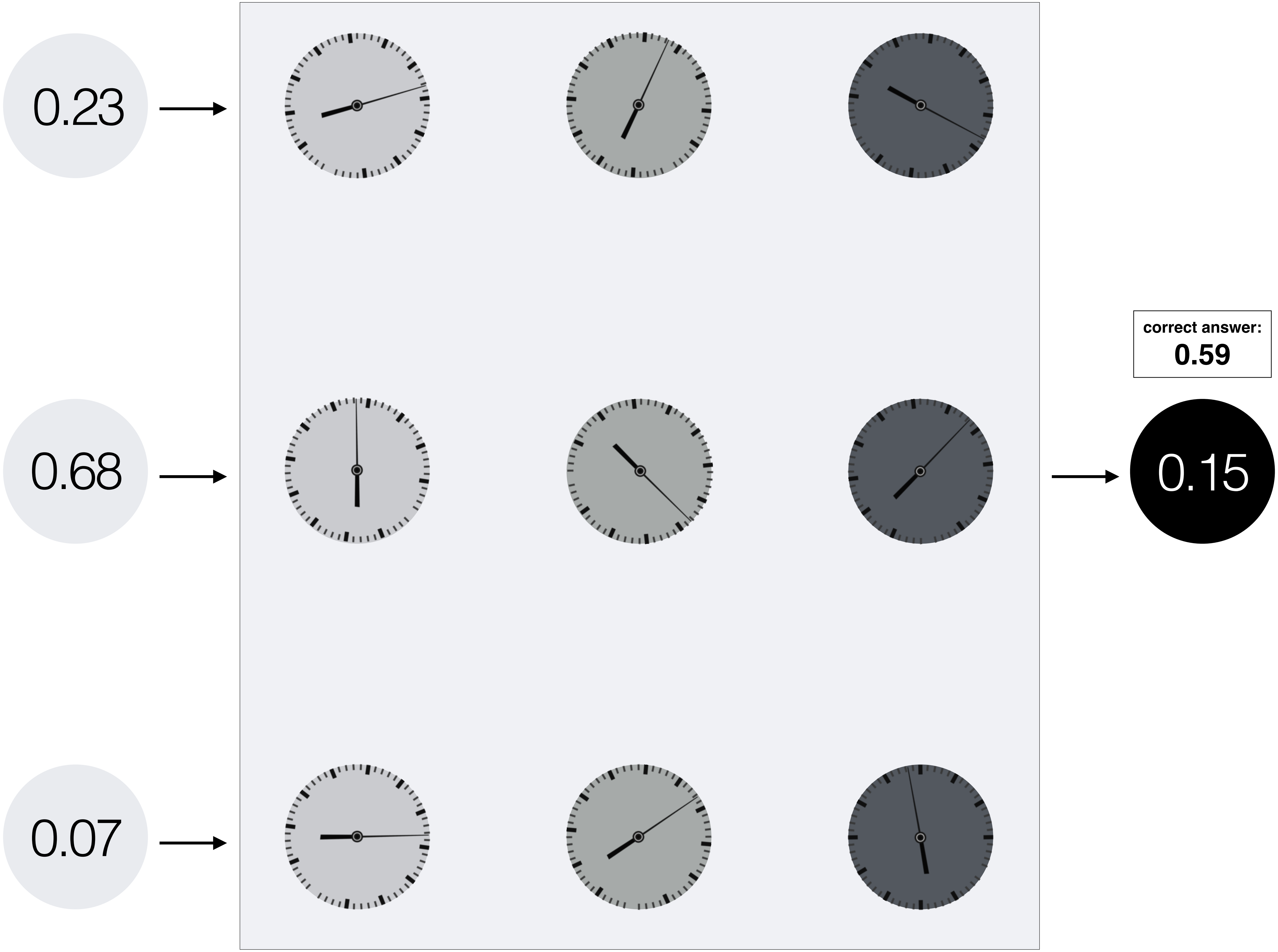
#2 Adjust the dials based on how far off the prediction was from the correct answer



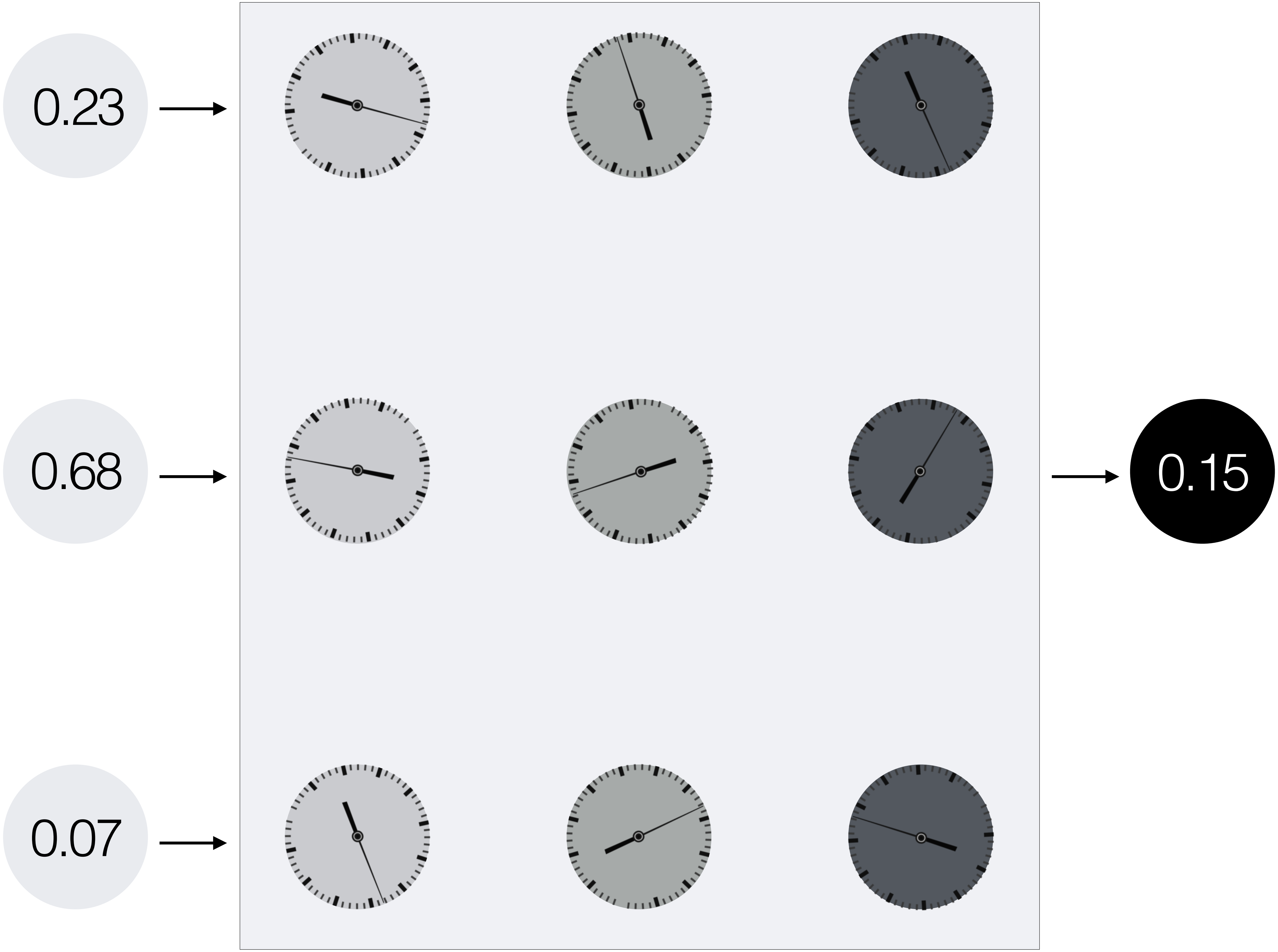
#3 **Feed a new set of inputs to the machine, and see what it outputs**



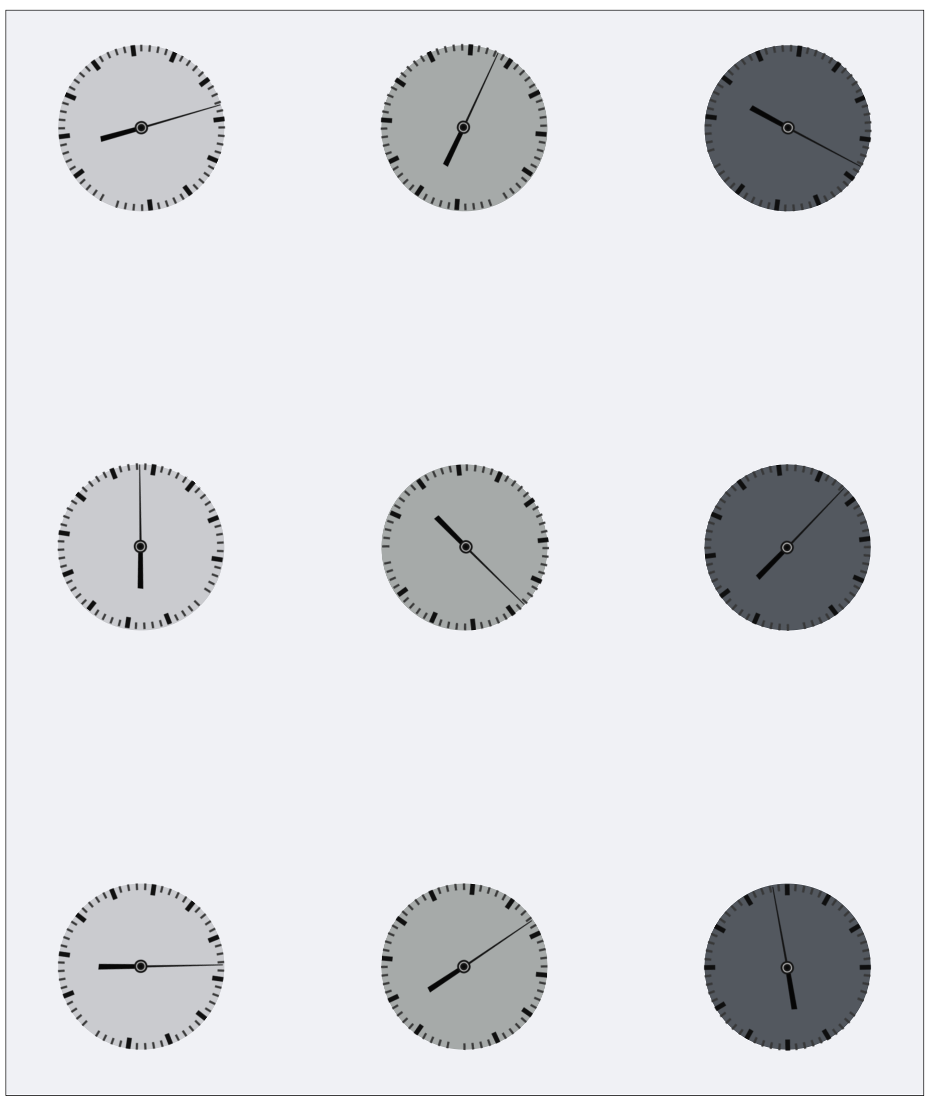
#3 Feed a new set of inputs to the machine, and see what it outputs

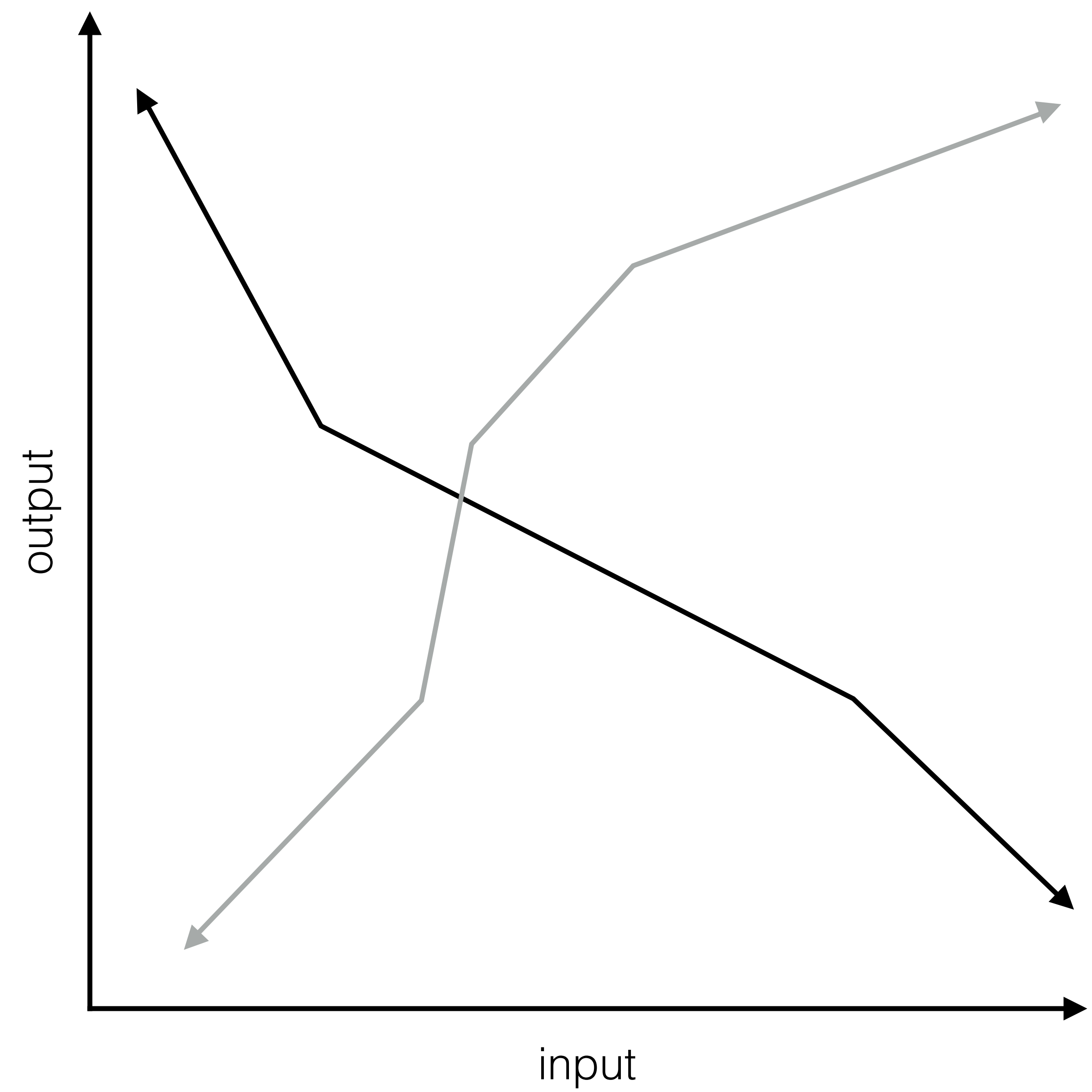


#3 **Adjust the dials again, and then continue this process until the machine is producing correct output**

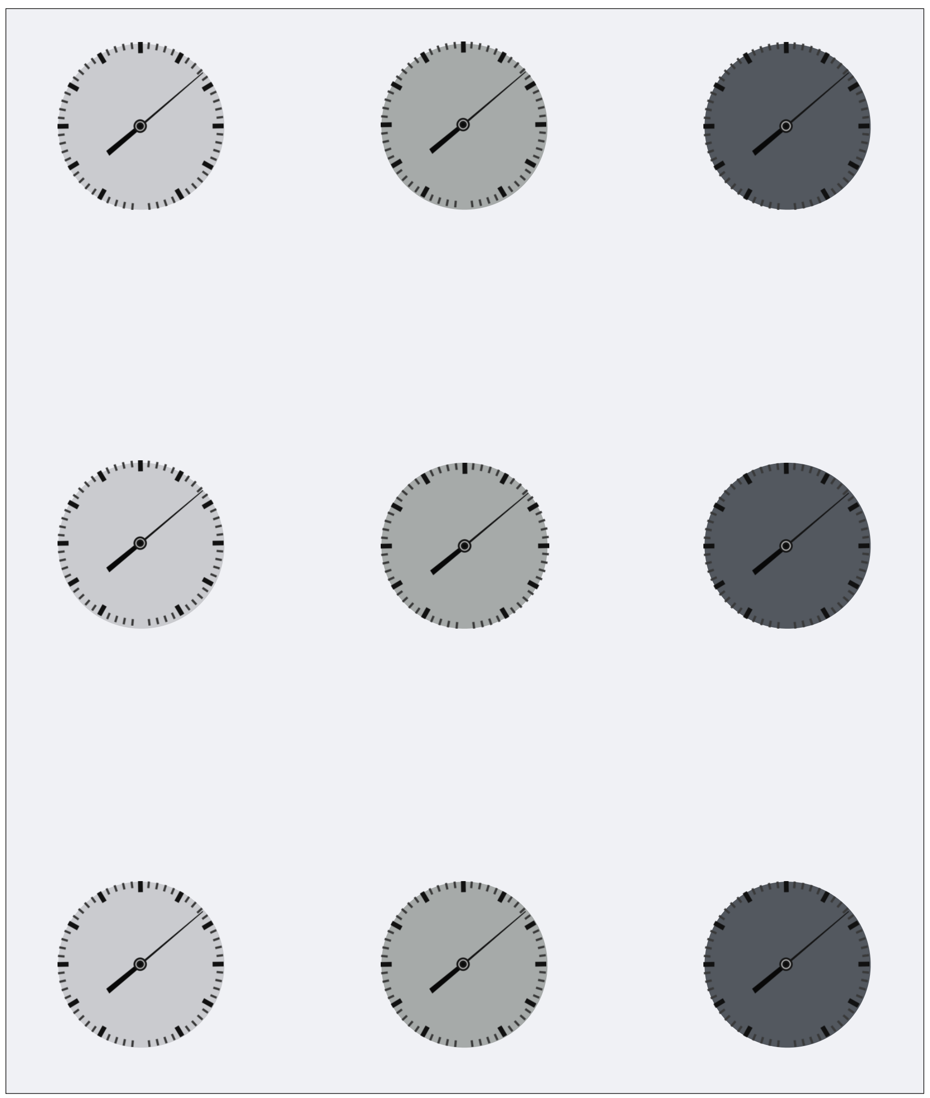


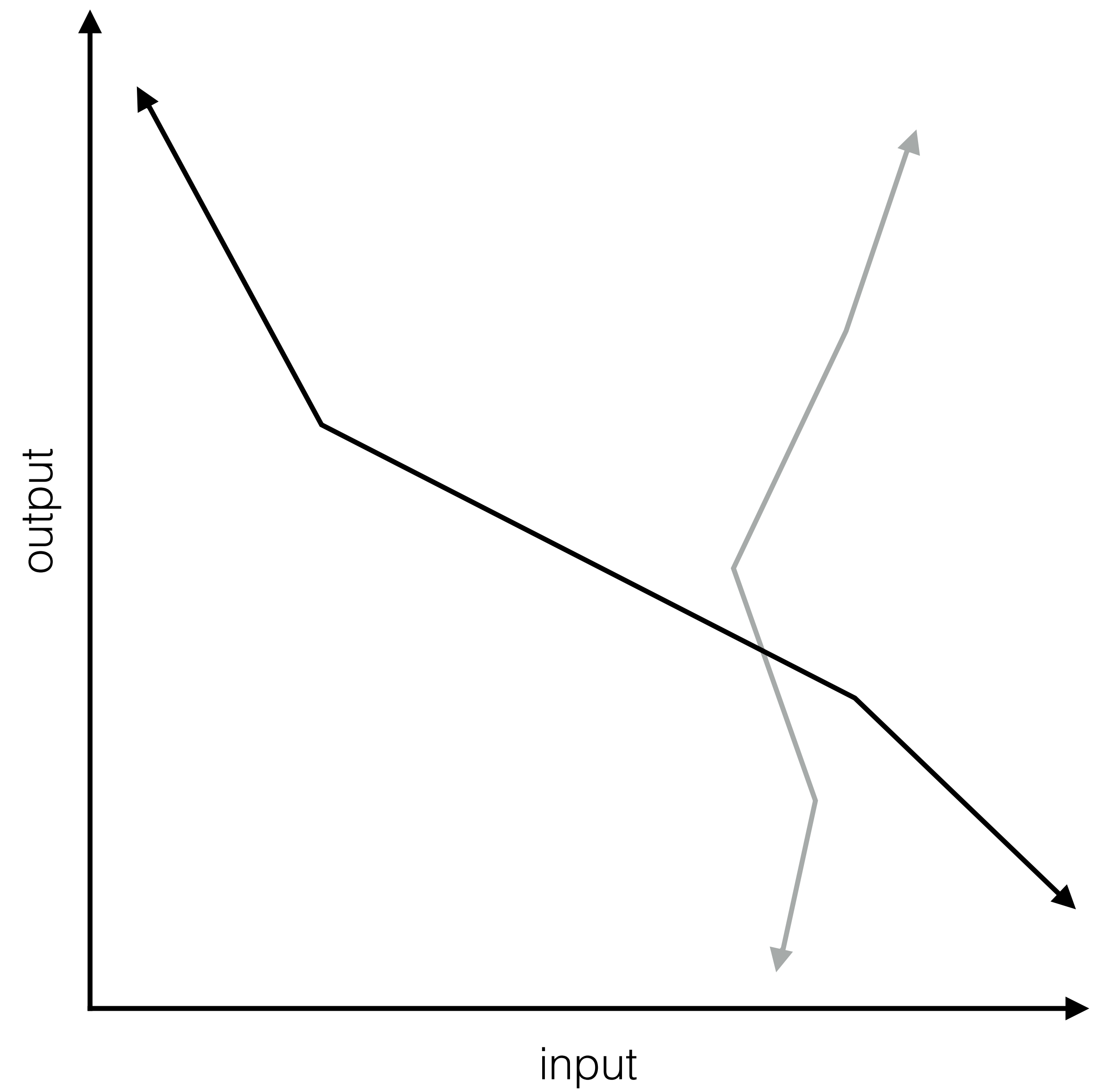
What this metaphorical “machine” is doing is creating a mapping between inputs and outputs (i.e. a function)



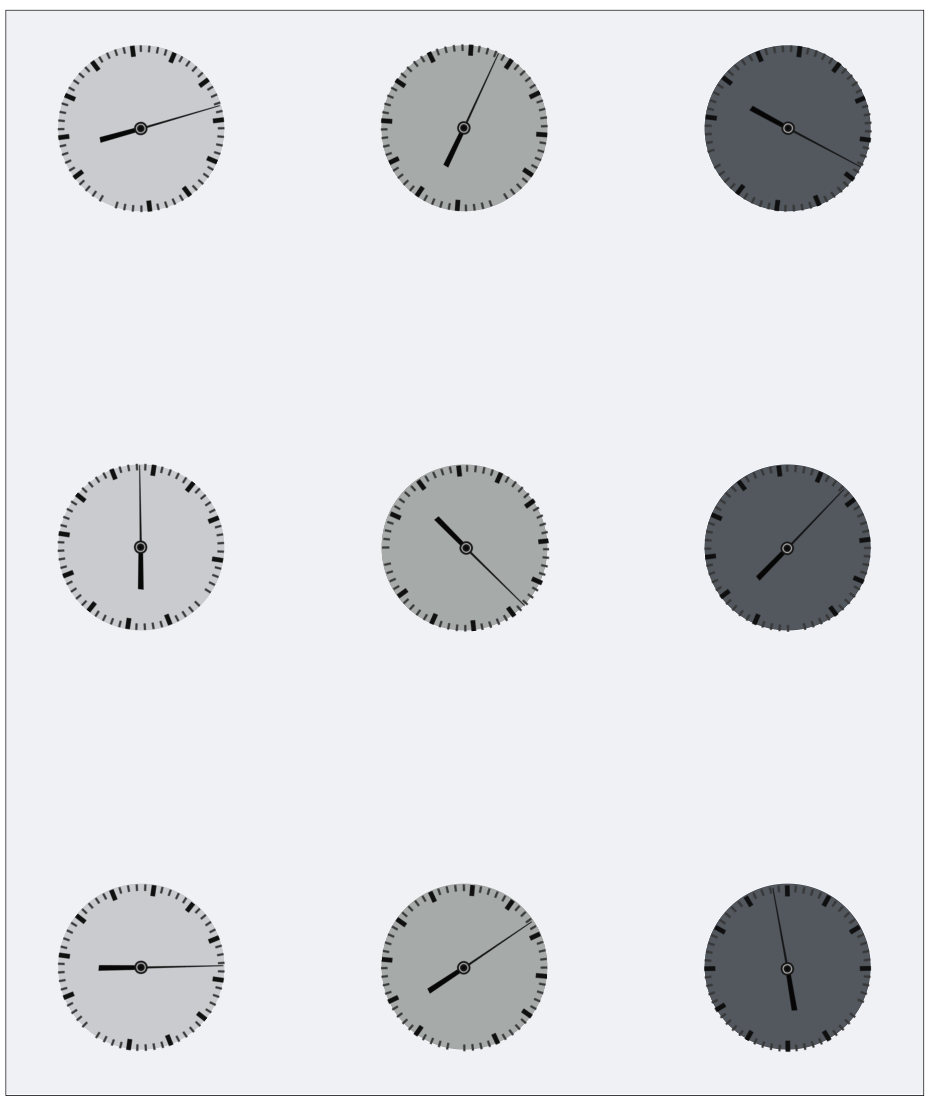


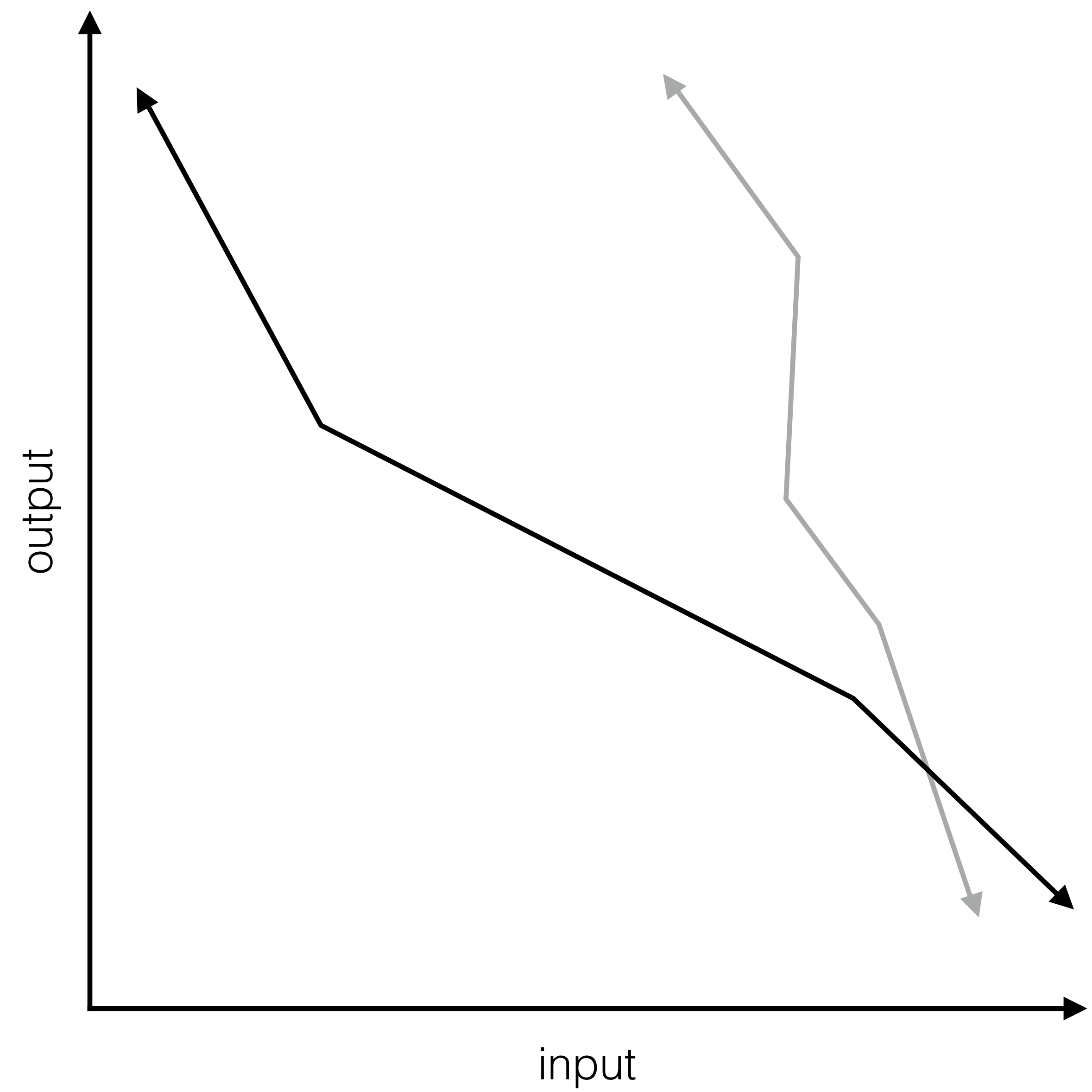
As the dials adjust, the input → output mapping changes…



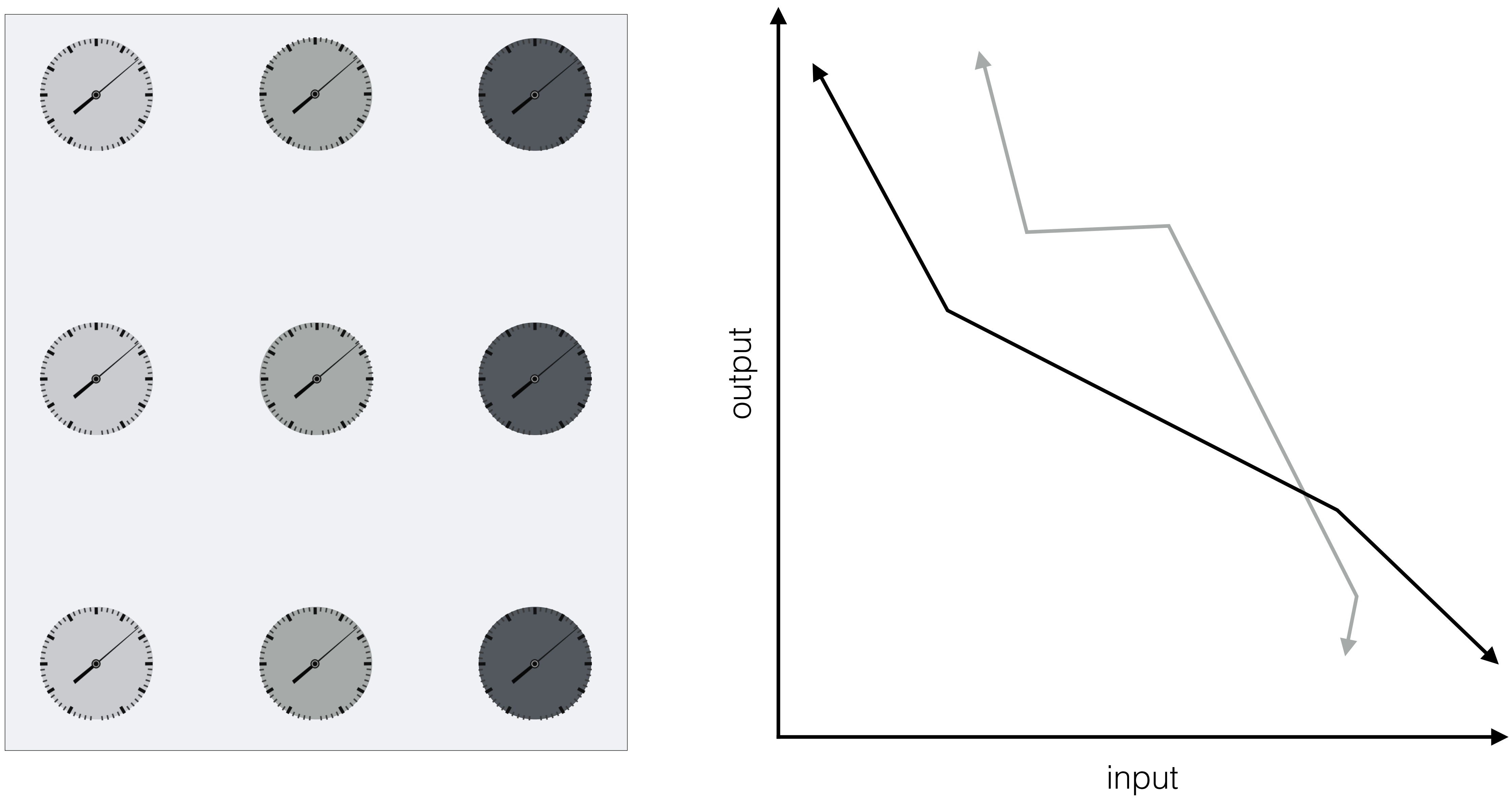


As the dials adjust, the input → output mapping changes…

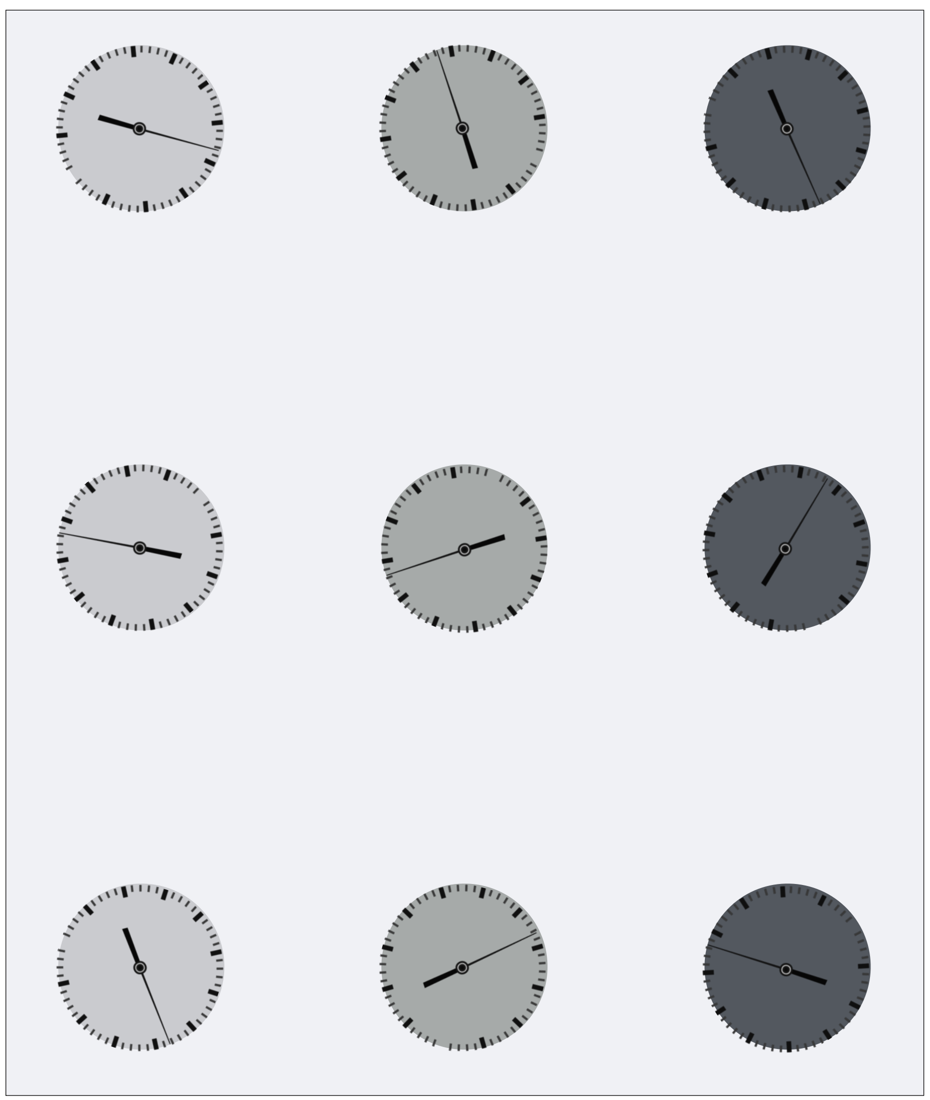


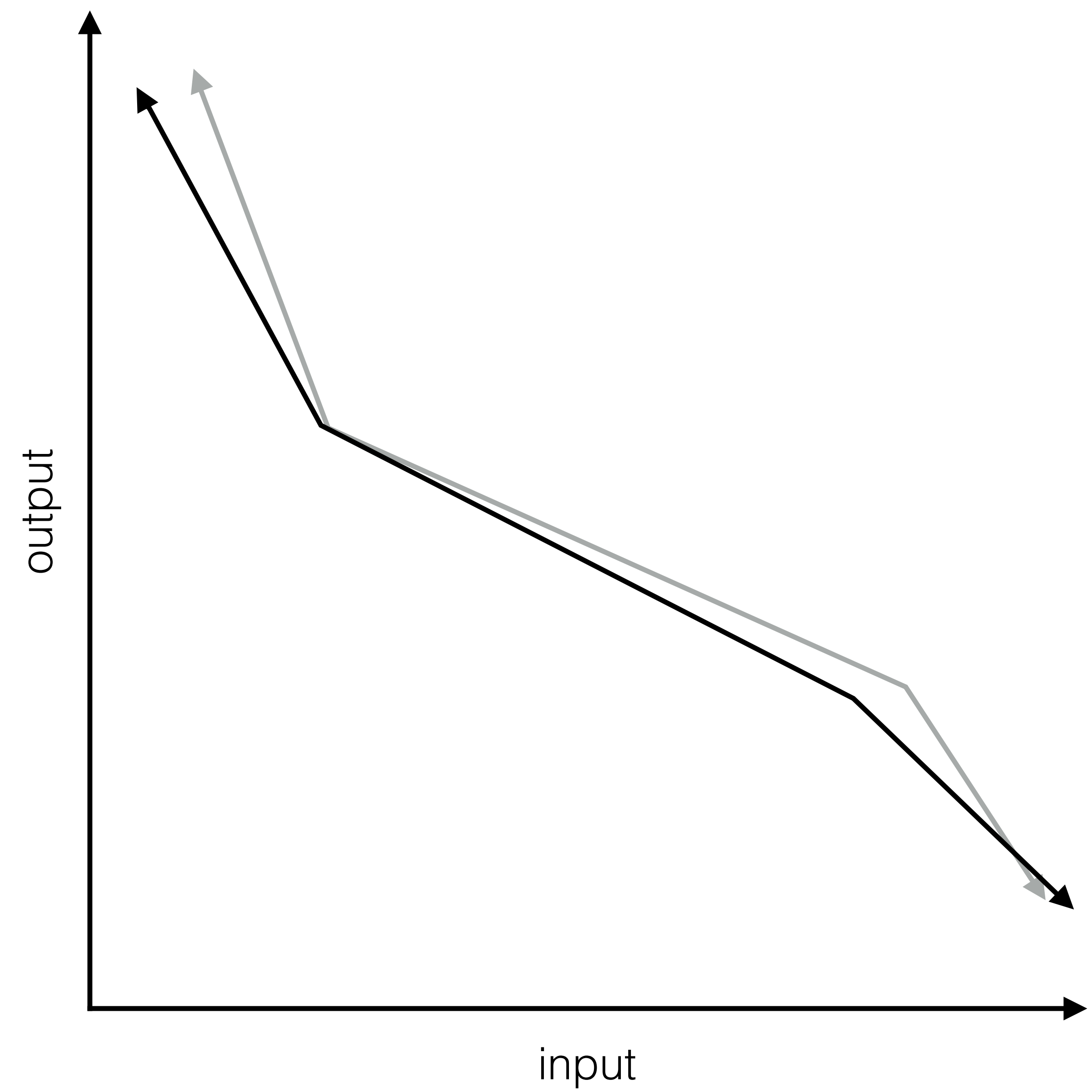


As the dials adjust, the input → output mapping changes…



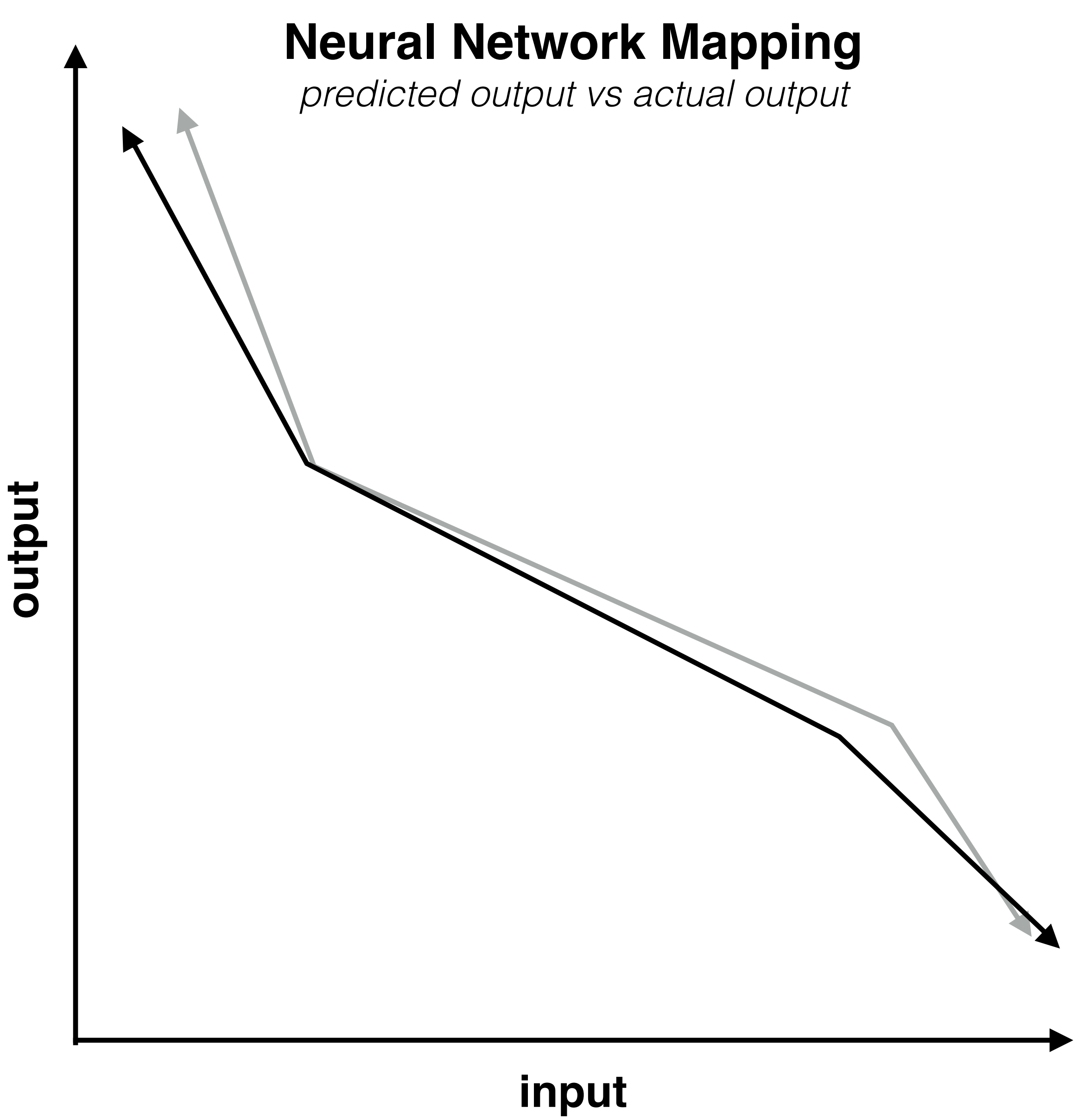
As the dials adjust, the input → output mapping changes…

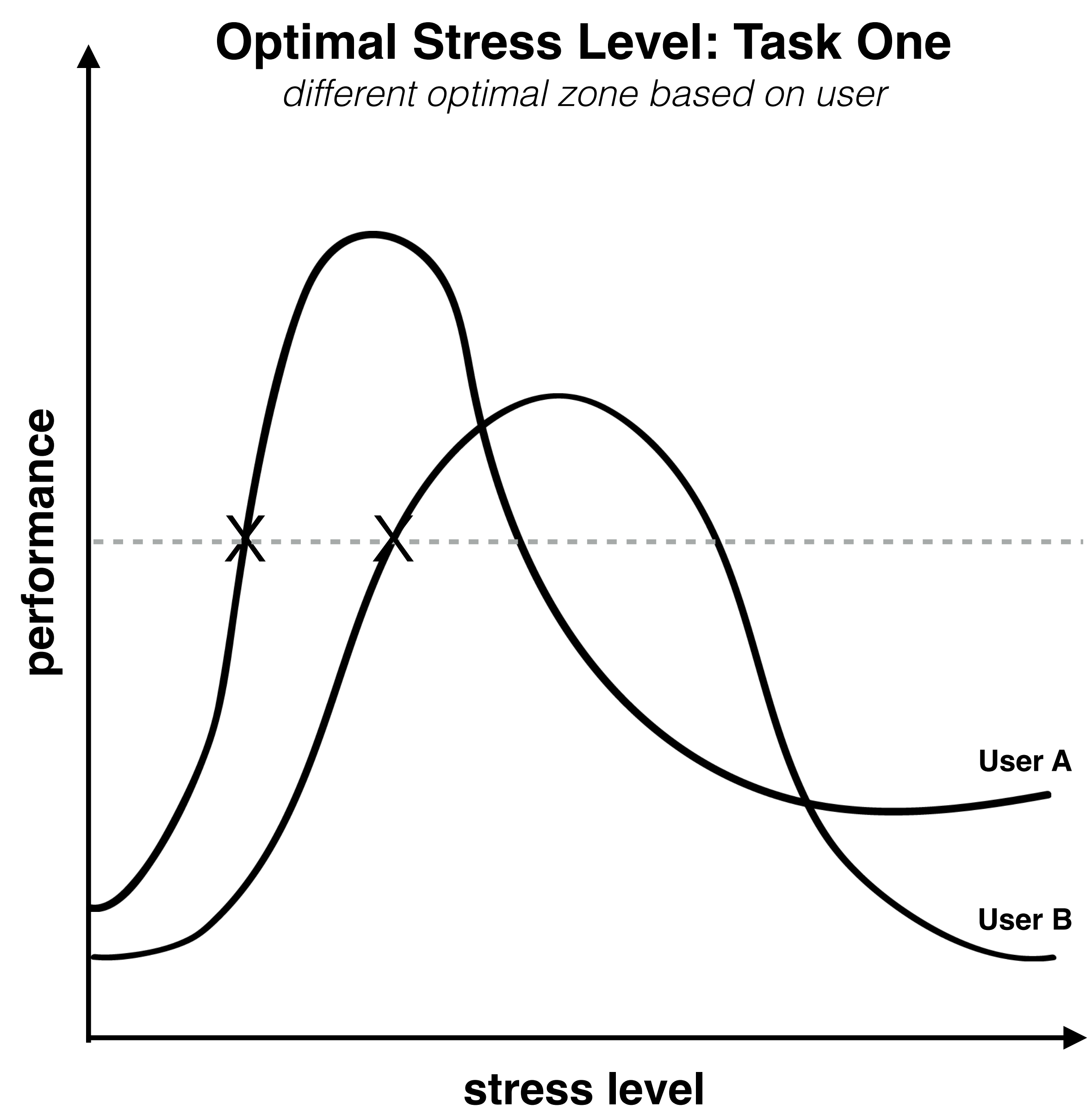




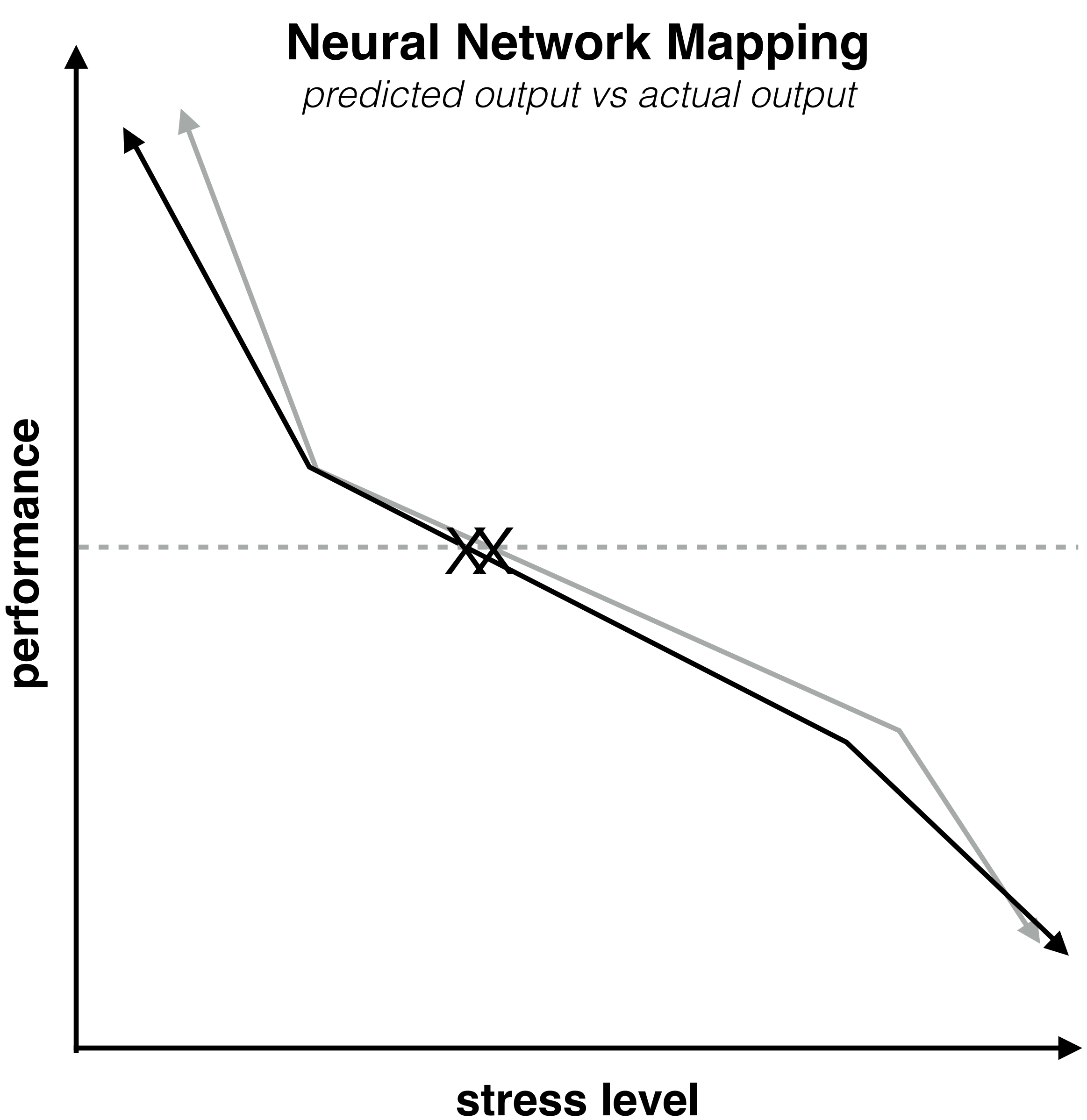
we can view this “optimal zone” as a point on a graph (i.e. performance as a function)

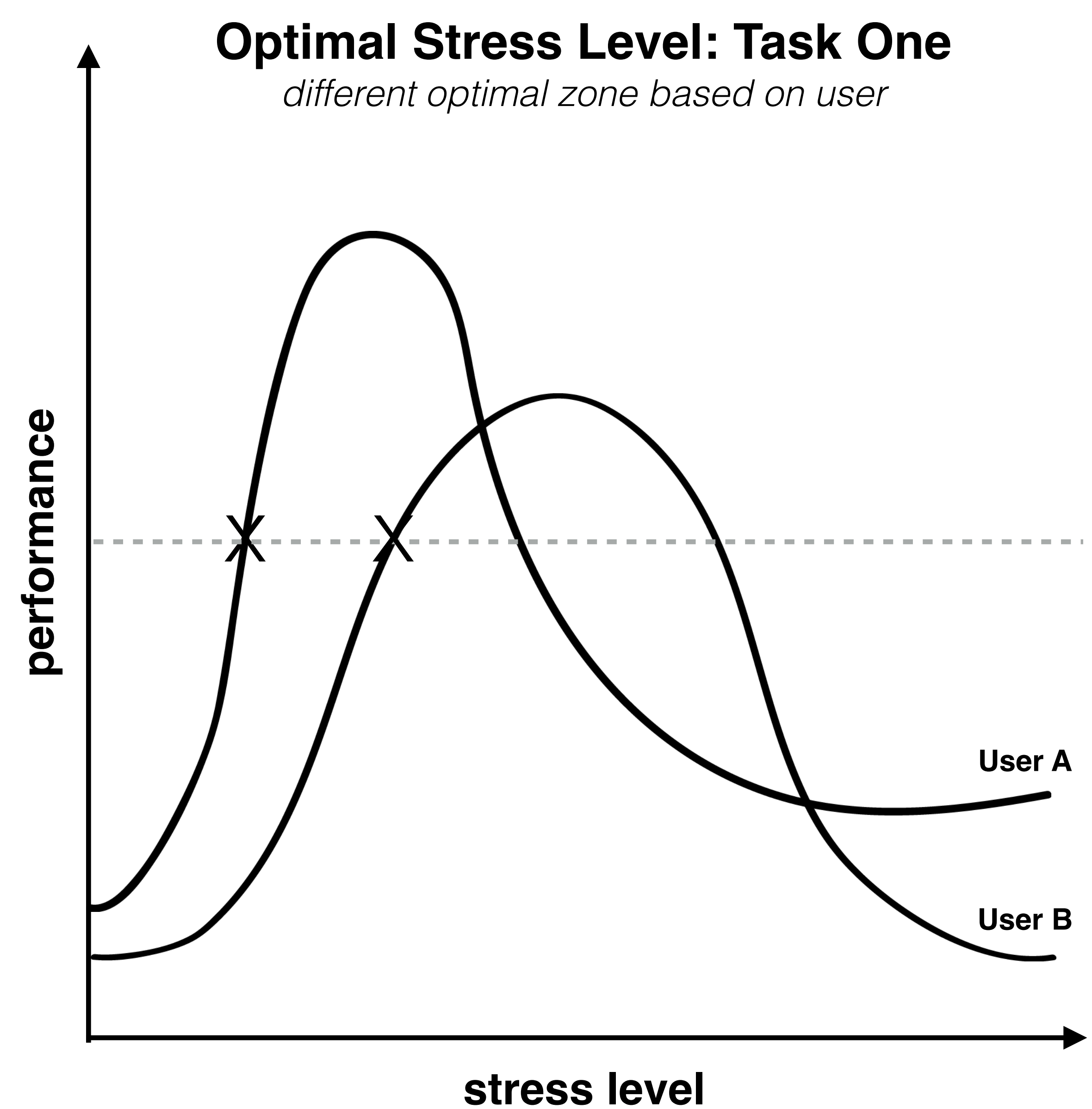
If we frame this problem as a function approximation problem, we can use a neural network to learn the performance function for a unique user in time





If we frame this problem as a function approximation problem, we can use a neural network to learn the performance function for a unique user in time





how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

what will these "unique needs" look like?

dynamic
user-dependent
environment-dependent
time-dependent

how can we dynamically change a user's environment?

we need a system that can adapt to these dynamic specifications.

would a neural network be a good fit?

how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

what will these "unique needs" look like?

dynamic
user-dependent
environment-dependent
time-dependent

how can we dynamically change a user's environment?

we can use a neural network that learns to predict a user's performance based on stress level

how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

what will these "unique needs" look like?

dynamic
user-dependent
environment-dependent
time-dependent

how can we dynamically change a user's environment?

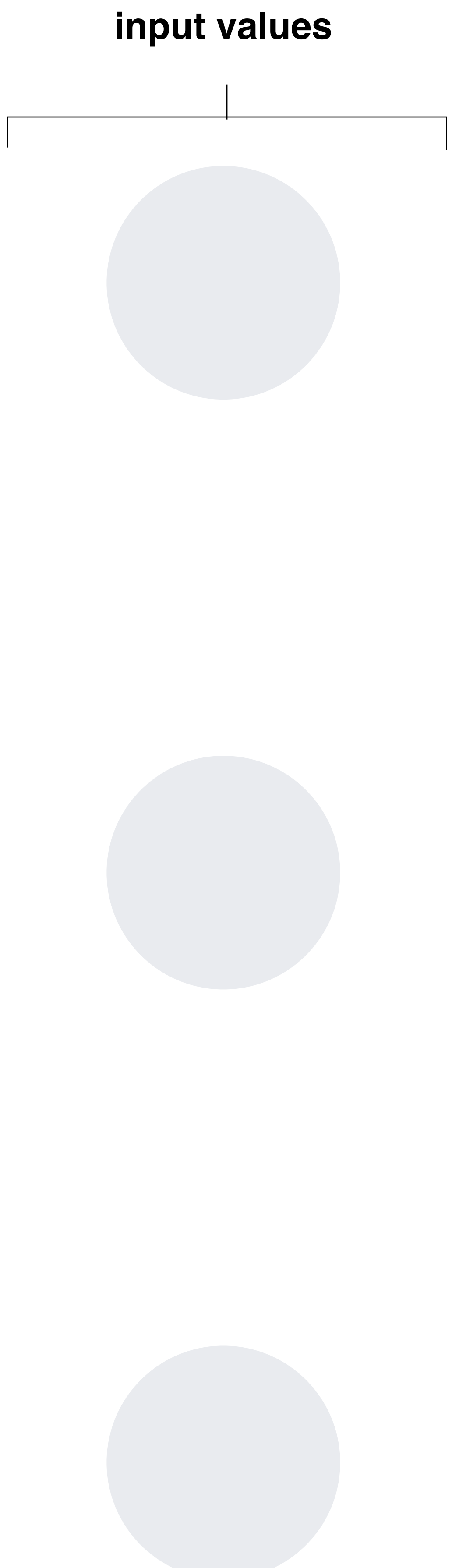
we can use a neural network that learns to predict a user's performance based on stress level

how can we measure stress level?

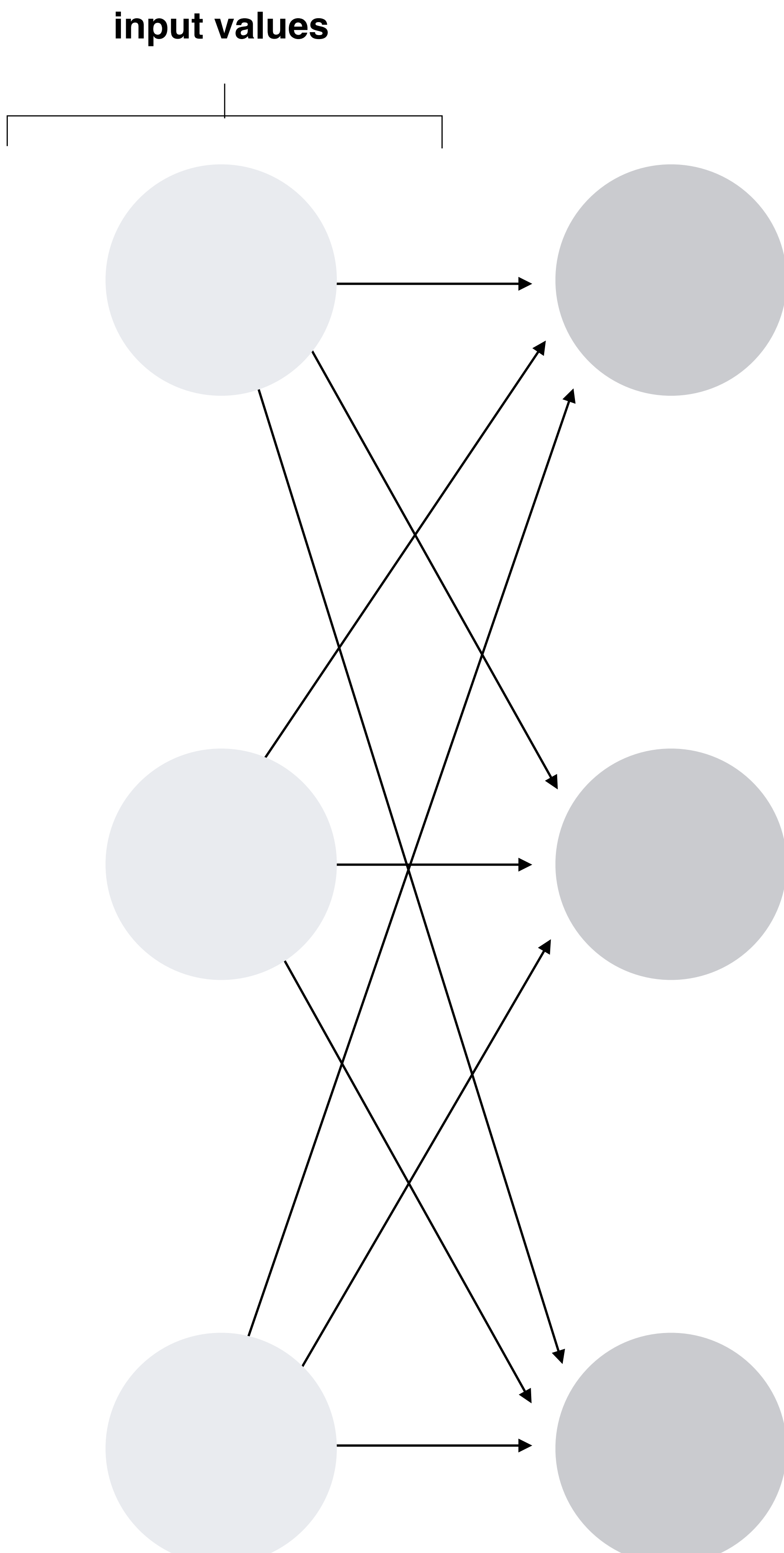
background research: basic NN architecture

in order to determine how we will represent stress level, we should first look at the architecture of a neural network

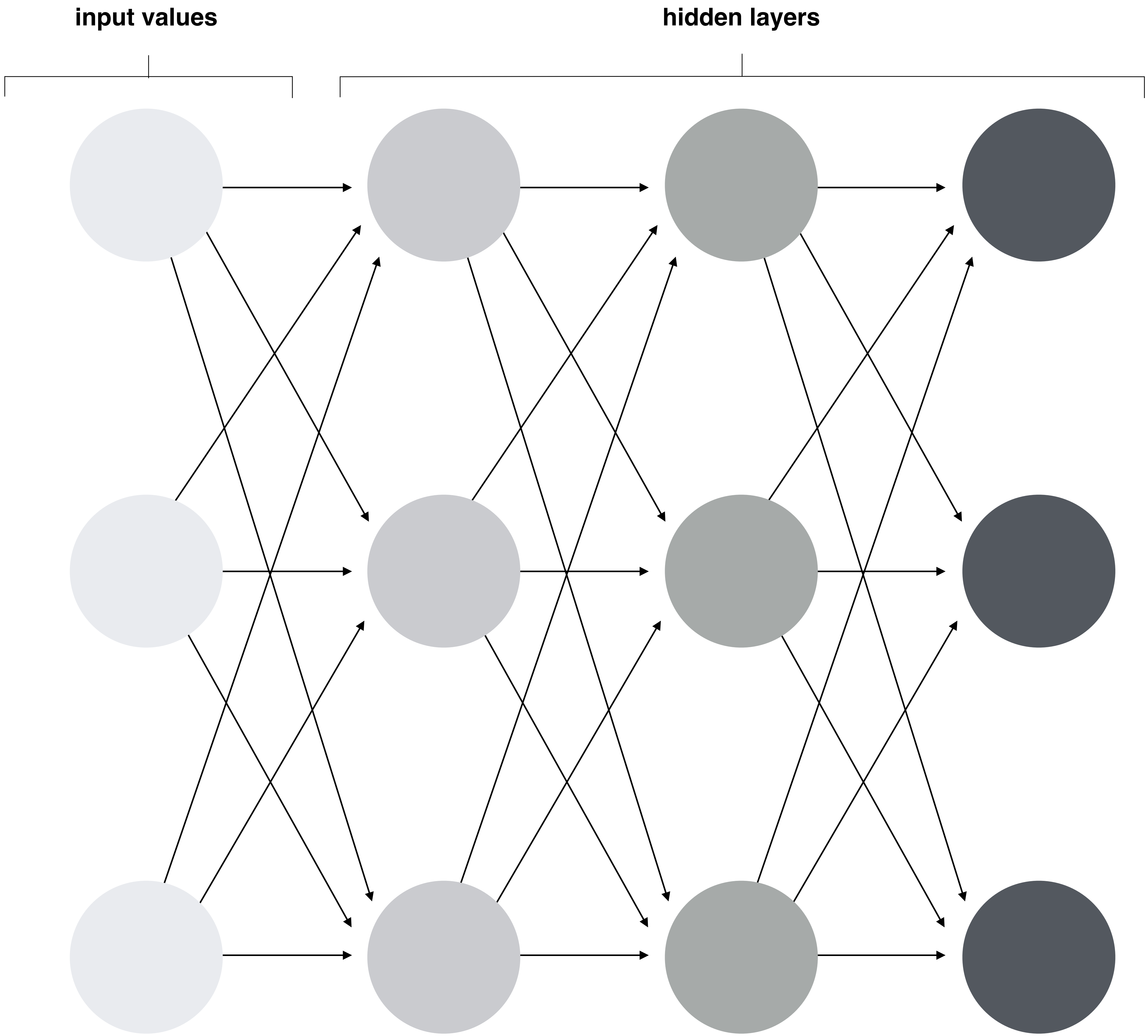
8. Michael A. Nielsen, “Neural Networks and Deep Learning”, Determination Press, 2015.
9. Welch Labs. *Neural Networks Demystified [Part 4: Backpropagation]*. Retrieved from https://www.youtube.com/watch?v=GlcnxUlrtek
10. Multilayer Perceptron — DeepLearning 0.1 documentation. (n.d.). Retrieved May 10, 2017, from http://deeplearning.net/tutorial/mlp.html



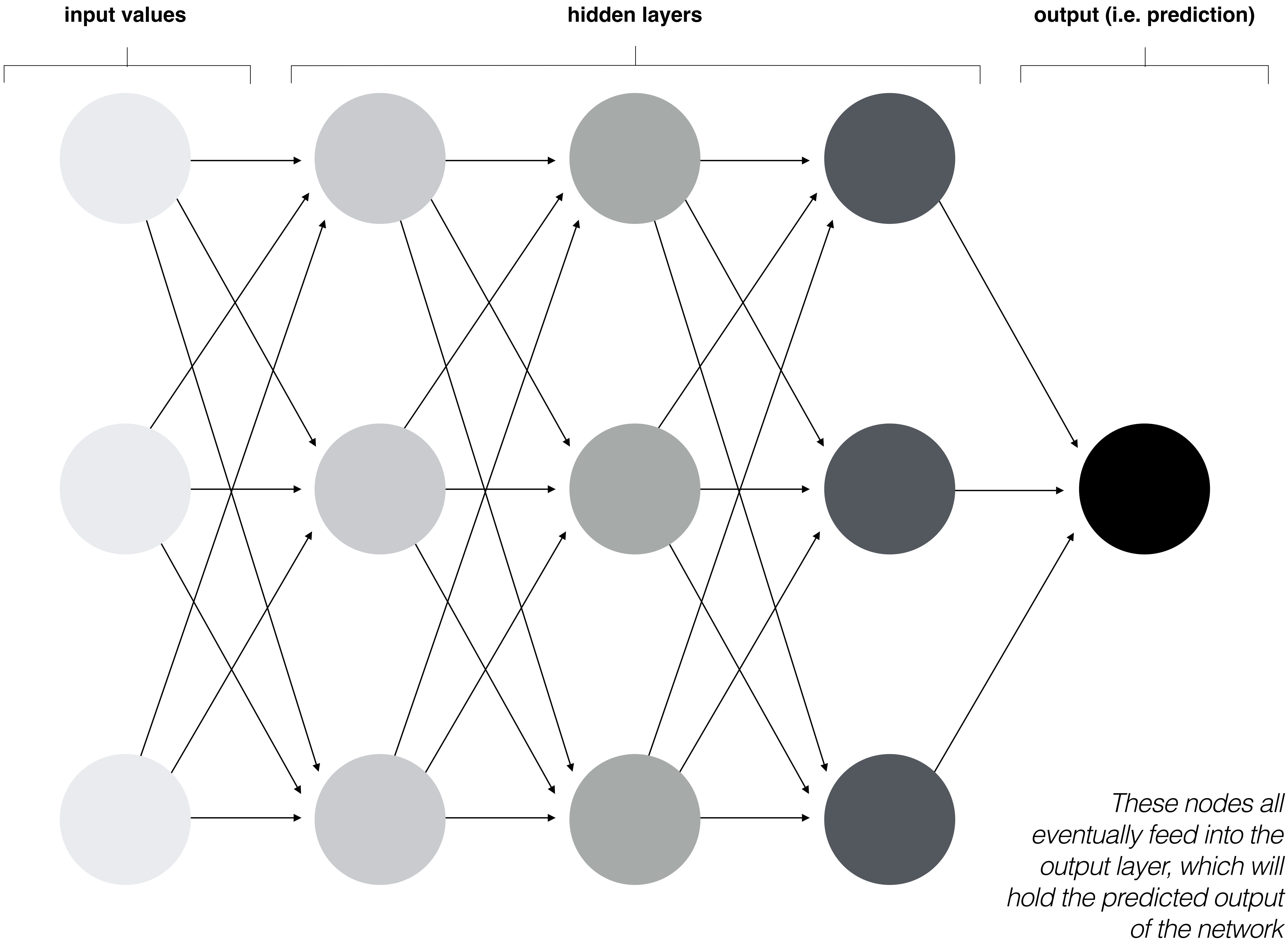
The input to a neural network is a series of values (either discrete or continuous)

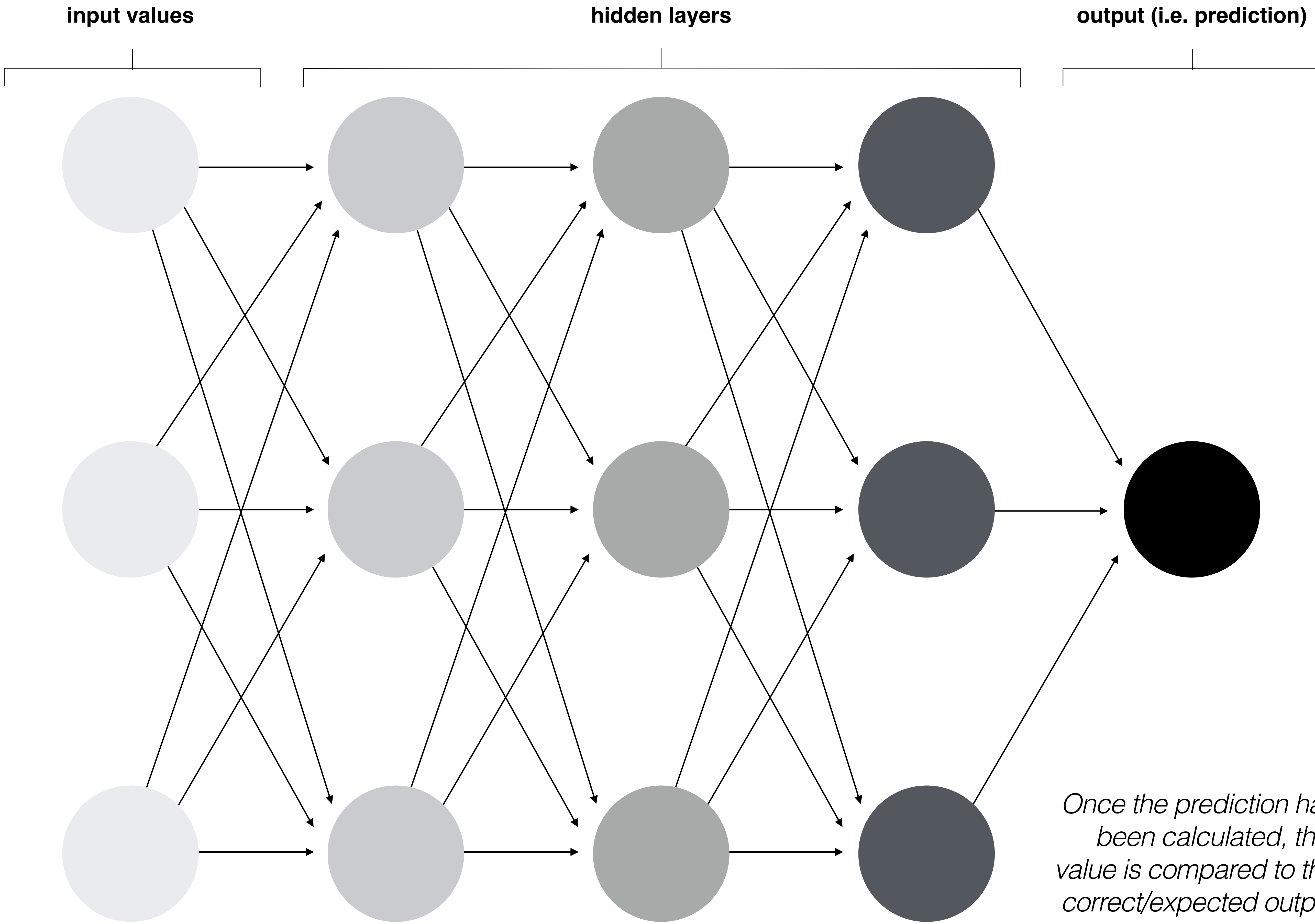


Each of these input values will feed into a series of hidden layer nodes (which hold a weight value between 0 and 1)

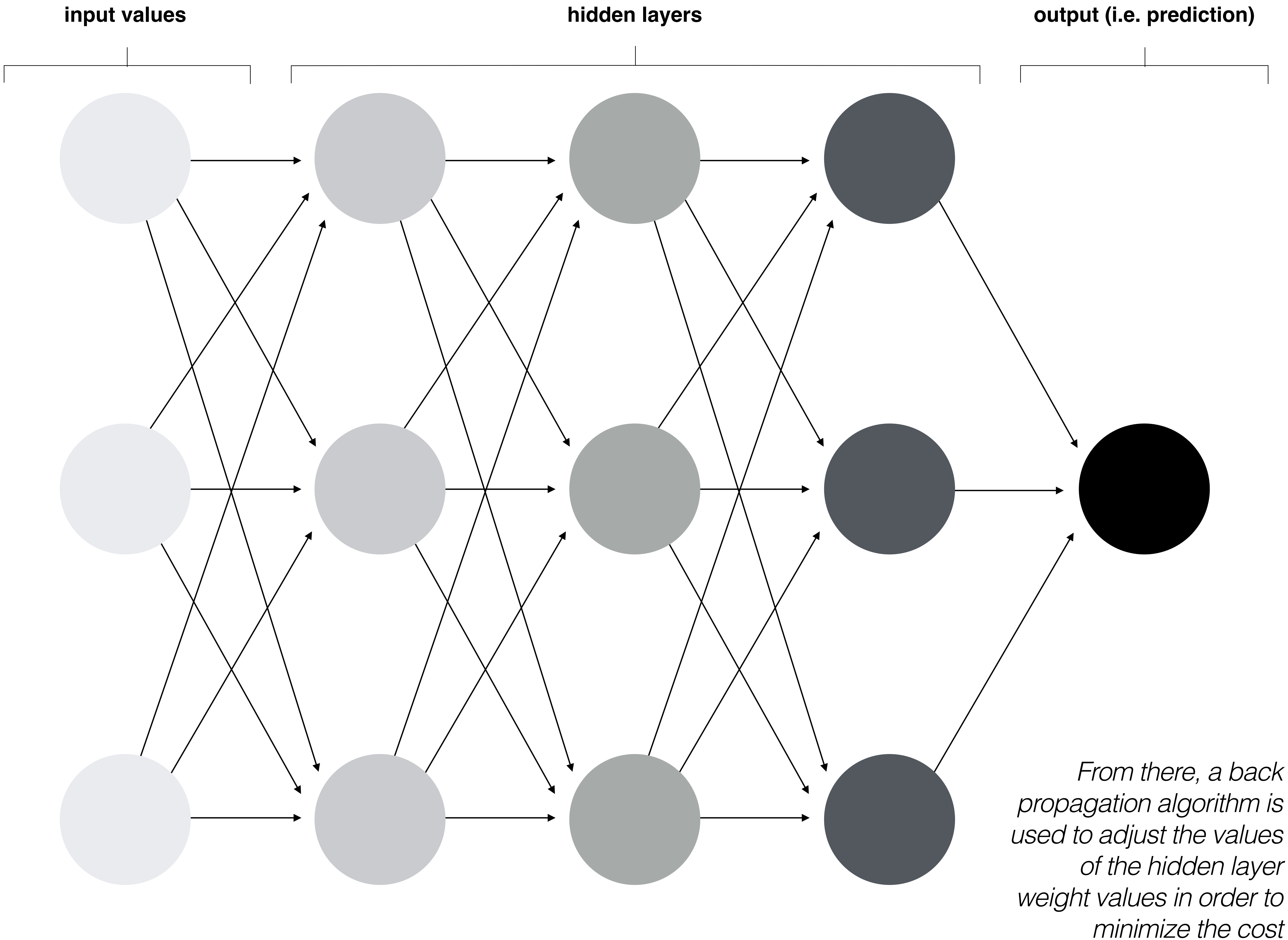


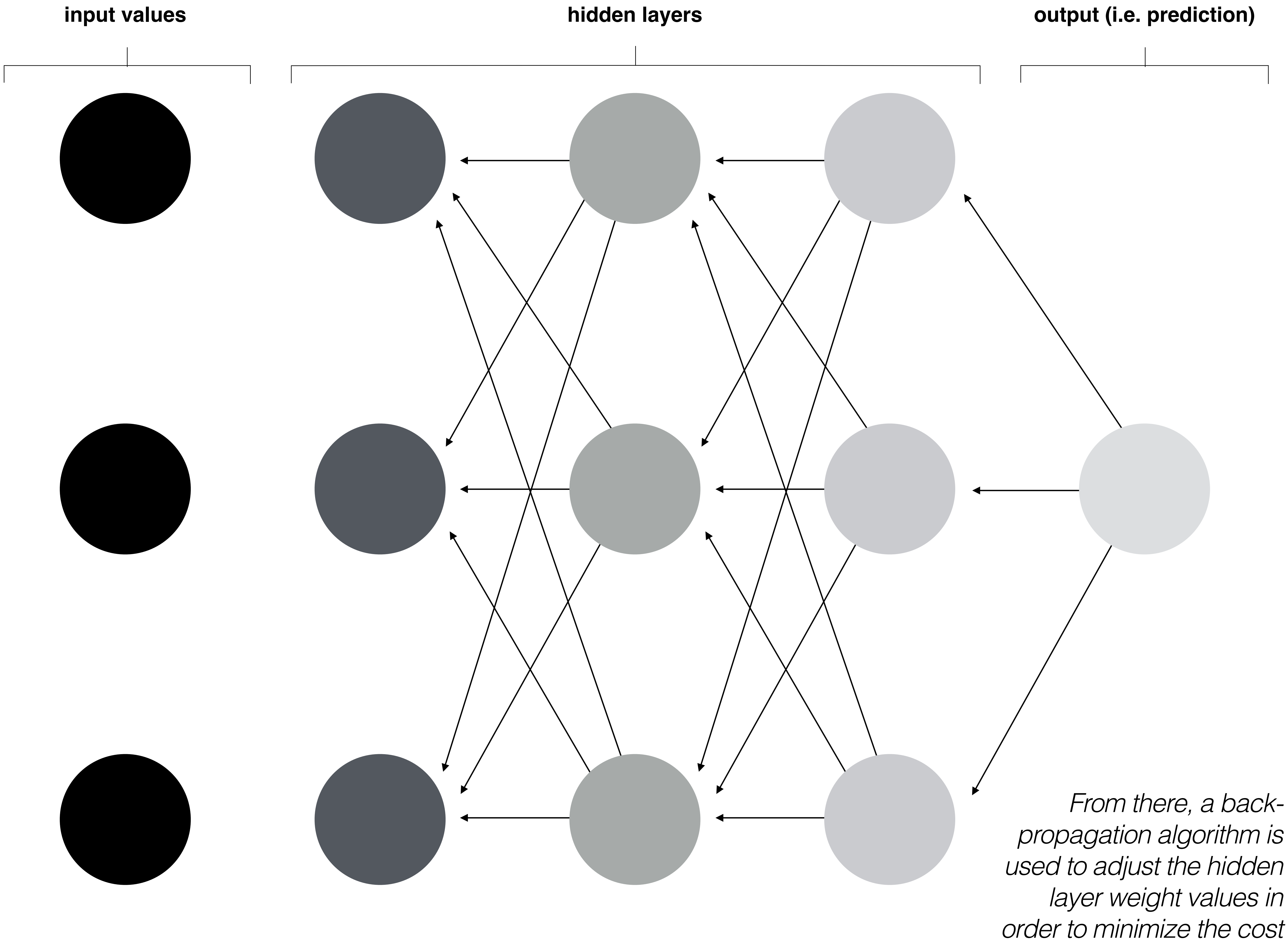
The number of weight nodes in a single hidden layer & the number of total hidden layers is typically dependent on the complexity of the problem being solved

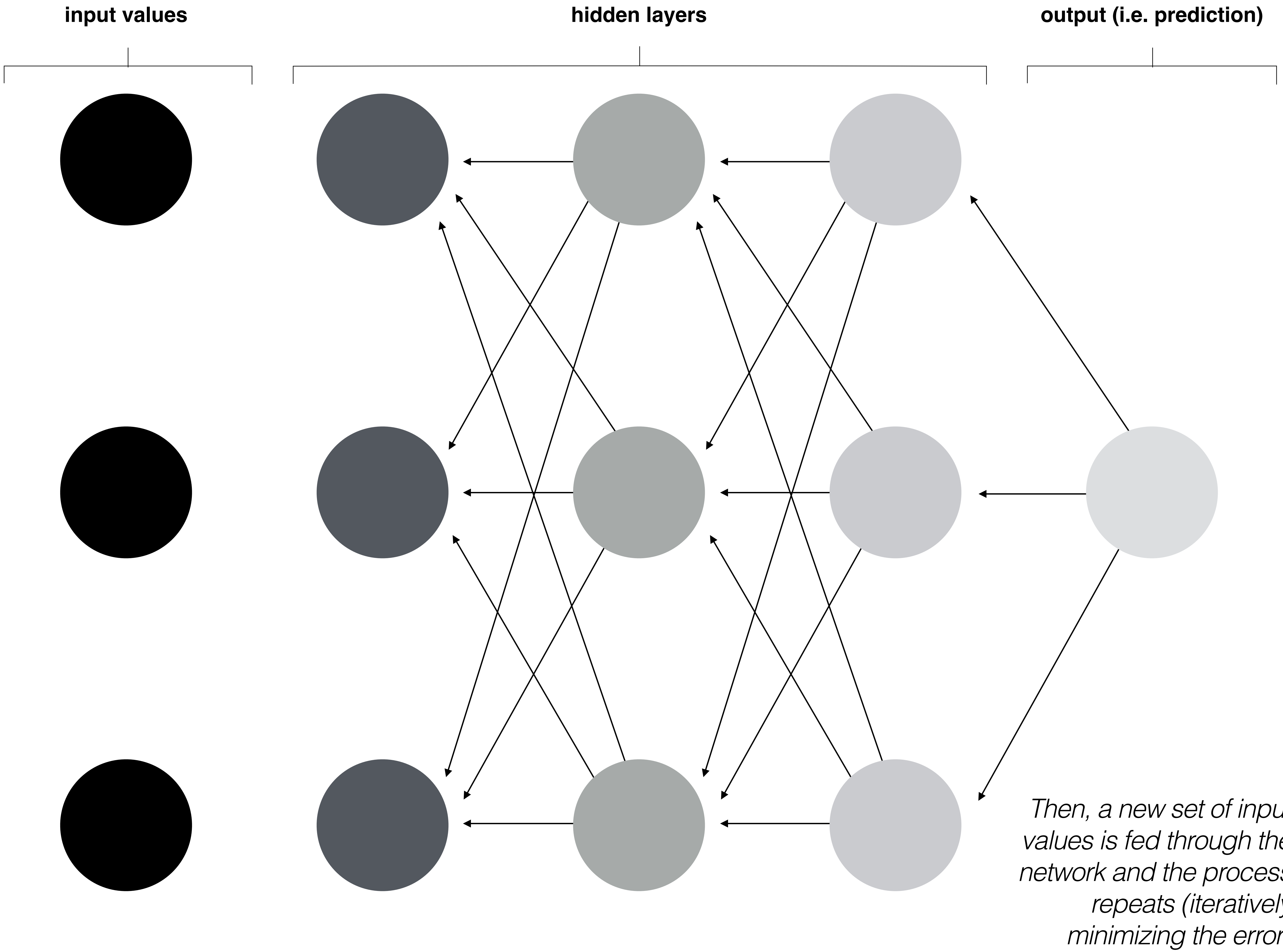

input values
hidden layers
output (i.e. prediction)
These nodes all eventually feed into the output layer, which will hold the predicted output of the network

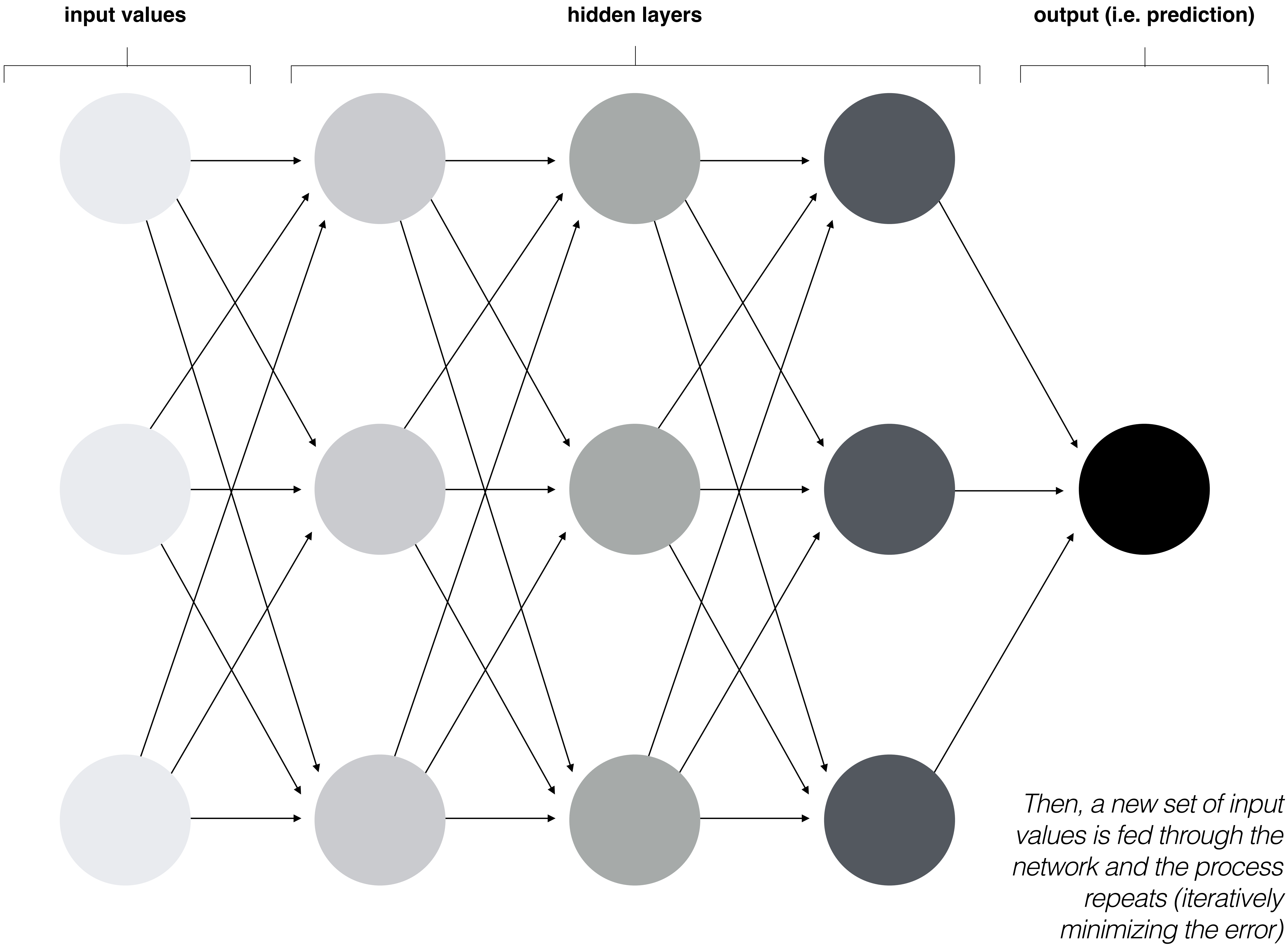


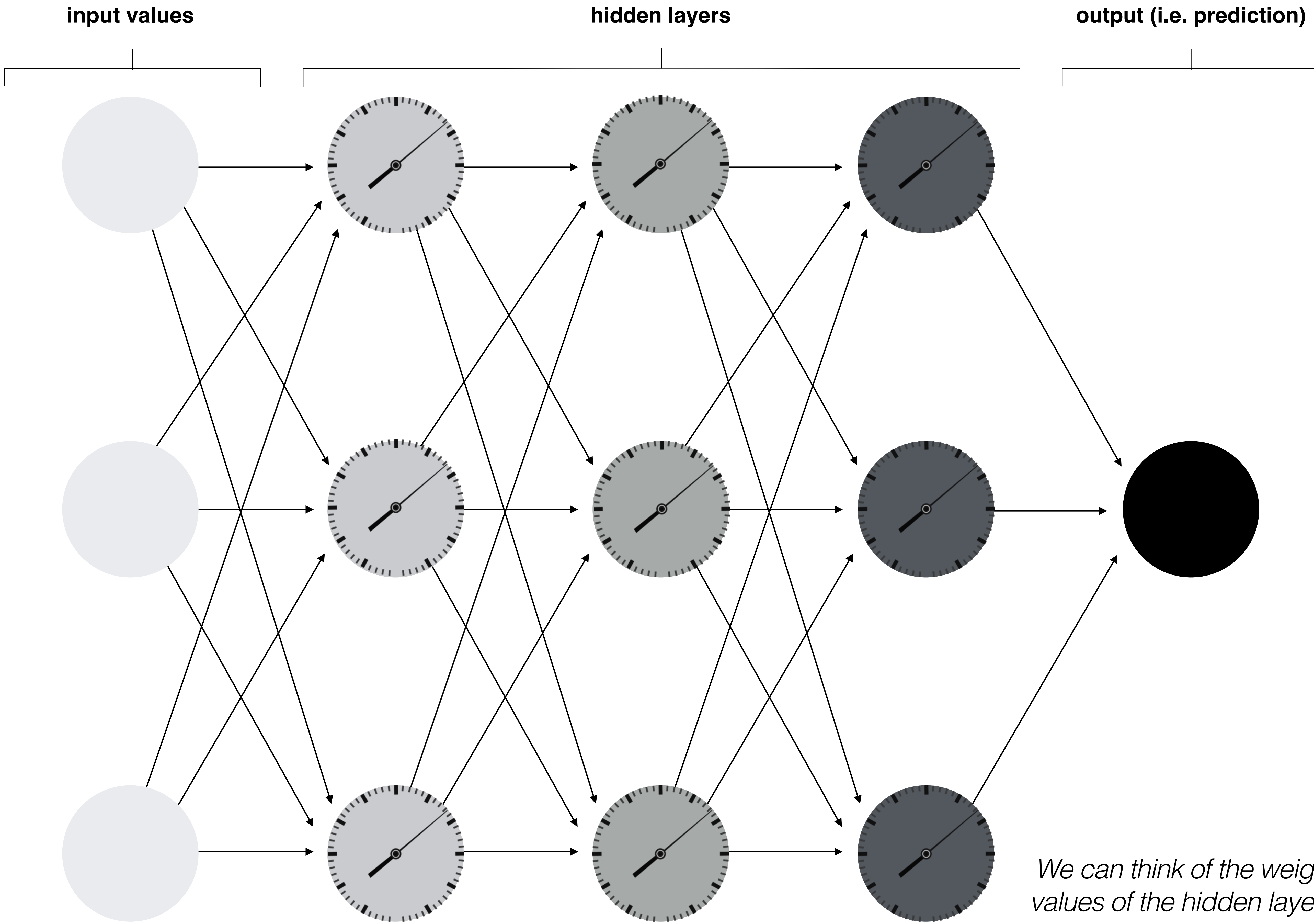
Once the prediction has been calculated, that value is compared to the correct/expected output using something called a <u>*cost (or loss) function*</u>


input values
hidden layers
output (i.e. prediction)
From there, a back propagation algorithm is used to adjust the values of the hidden layer weight values in order to minimize the cost

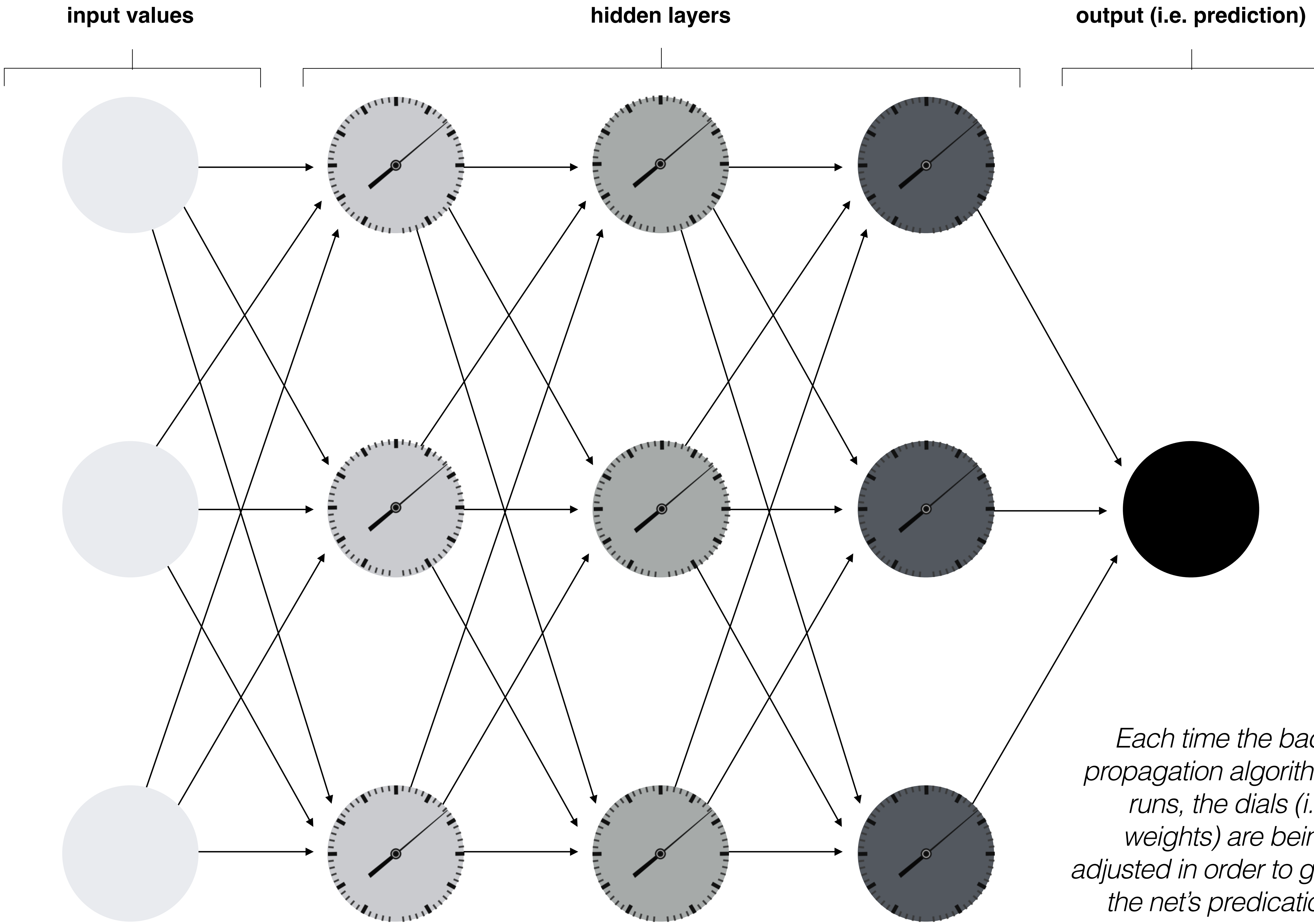

input values
hidden layers
output (i.e. prediction)
From there, a back-propagation algorithm is used to adjust the hidden layer weight values in order to minimize the cost


input values
hidden layers
output (i.e. prediction)
Then, a new set of input values is fed through the network and the process repeats (iteratively minimizing the error)

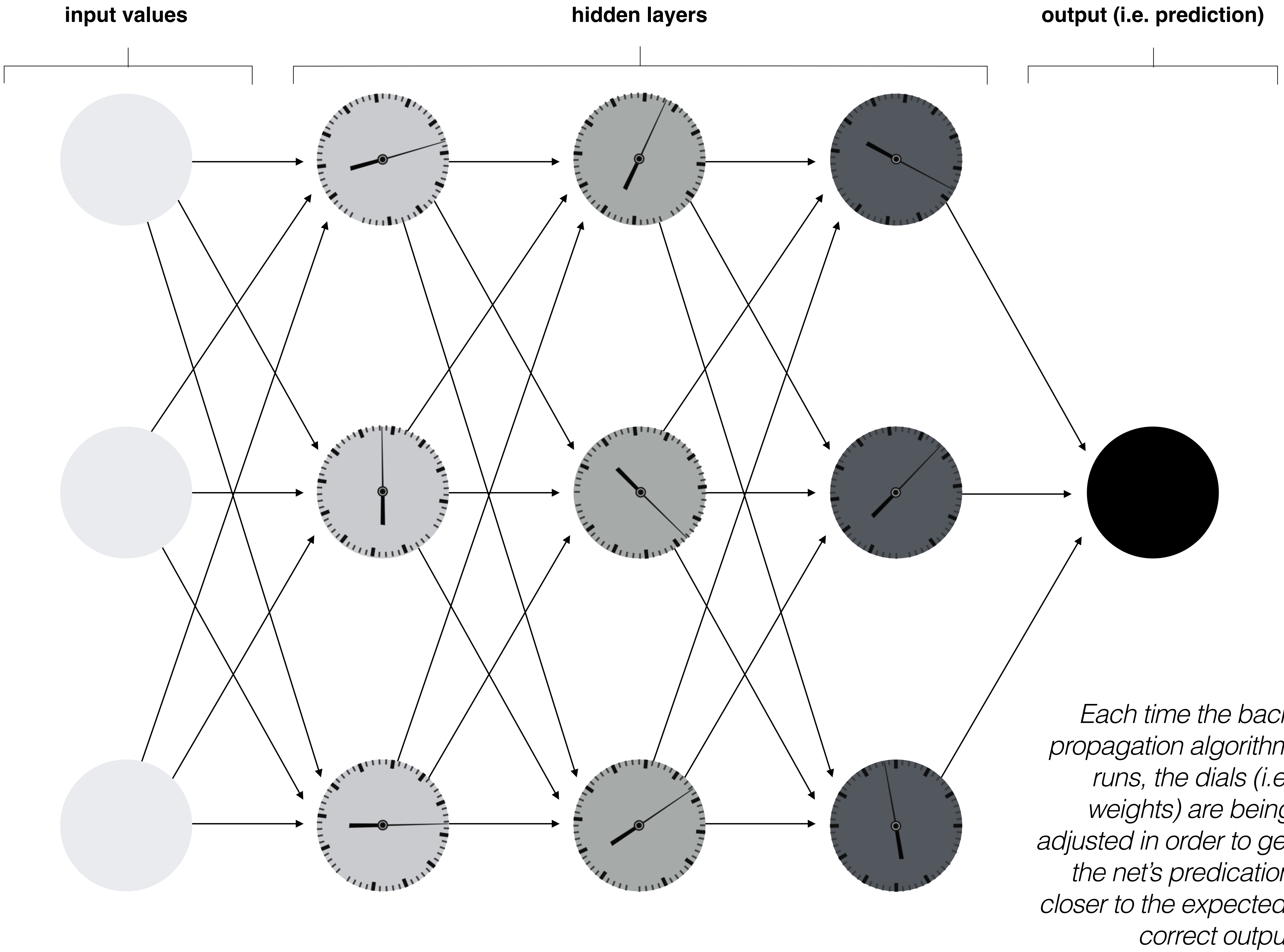

input values
hidden layers
output (i.e. prediction)
Then, a new set of input values is fed through the network and the process repeats (iteratively minimizing the error)

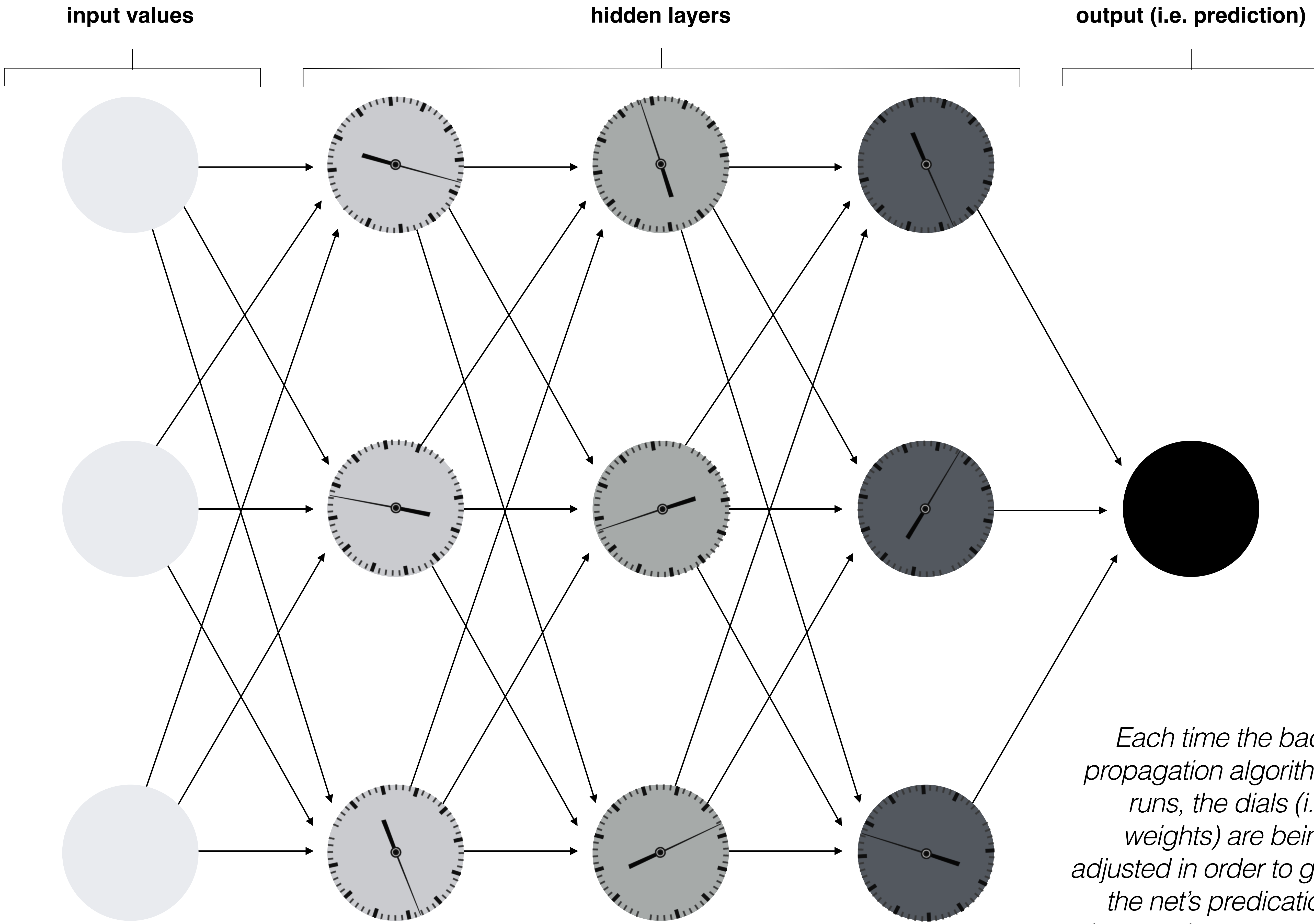


We can think of the weight values of the hidden layers like the dials from our metaphorical "machine"



Each time the back propagation algorithm runs, the dials (i.e. weights) are being adjusted in order to get the net's predication closer to the expected/ correct output


input values
hidden layers
output (i.e. prediction)
Each time the back propagation algorithm runs, the dials (i.e. weights) are being adjusted in order to get the net's predication closer to the expected/ correct output



Each time the back propagation algorithm runs, the dials (i.e. weights) are being adjusted in order to get the net's predication closer to the expected/ correct output

we now know that the input of a neural network is a series of discrete or continuous numeric values

we can represent stress level in our neural network by obtaining a series of numeric values that reflect the level of difficulty in the user environment

how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

what will these "unique needs" look like?

dynamic
user-dependent
environment-dependent
time-dependent

how can we dynamically change a user's environment?

we can use a neural network that learns to predict a user's performance based on stress level

how can we measure stress level?

how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

what will these "unique needs" look like?

dynamic
user-dependent
environment-dependent
time-dependent

how can we dynamically change a user's environment?

we can use a neural network that learns to predict a user's performance based on *environment parameters* that capture difficulty/stress level

how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

what will these "unique needs" look like?

dynamic
user-dependent
environment-dependent
time-dependent

how can we dynamically change a user's environment?

we can use a neural network that learns to predict a user's performance based on *environment parameters* that capture difficulty/stress level

how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

what will these "unique needs" look like?

dynamic
user-dependent
environment-dependent
time-dependent

how can we dynamically change a user's environment?

once we have that model, we can use it to <u>modify the environment parameters</u> because we have successfully approximated the user performance function

how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

what will these "unique needs" look like?

dynamic
user-dependent
environment-dependent
time-dependent

how can we dynamically change a user's environment?

once we have that model, we can use it to modify the environment parameters because we have successfully approximated the user performance function

how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

what will these "unique needs" look like?

dynamic
user-dependent
environment-dependent
time-dependent

how can we dynamically change a user's environment?

once we have that model, we can use it to modify the environment parameters because we have successfully approximated the user performance function

how can we use computers to dynamically change a user's environment to fit their unique needs in order for them to achieve some goal?

what will these "unique needs" look like?

dynamic
user-dependent
environment-dependent
time-dependent

how can we dynamically change a user's environment?

we now have two distinct objectives that the neural network must meet

objective #1

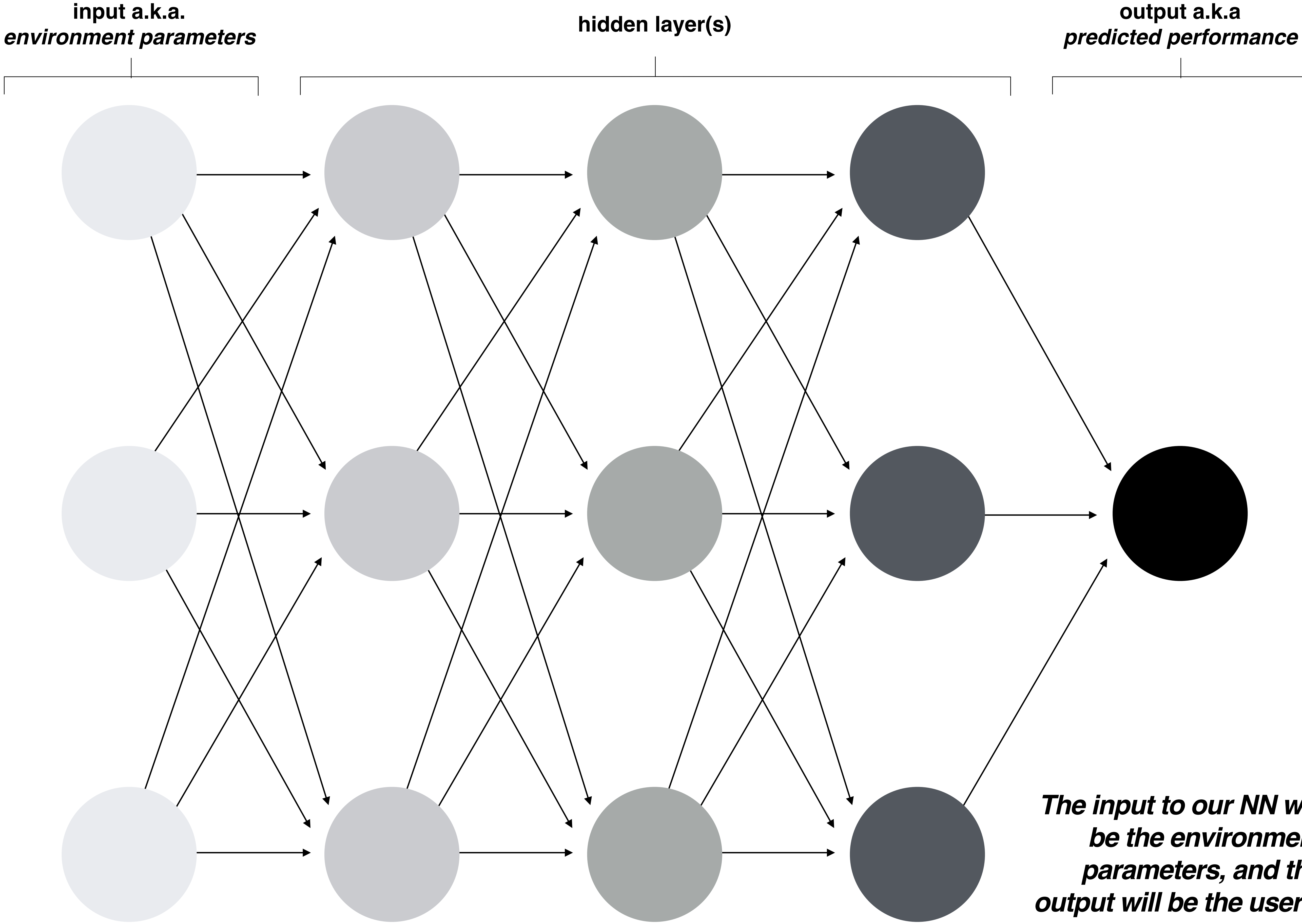
predict a user's performance based on *environment parameters*

objective #2

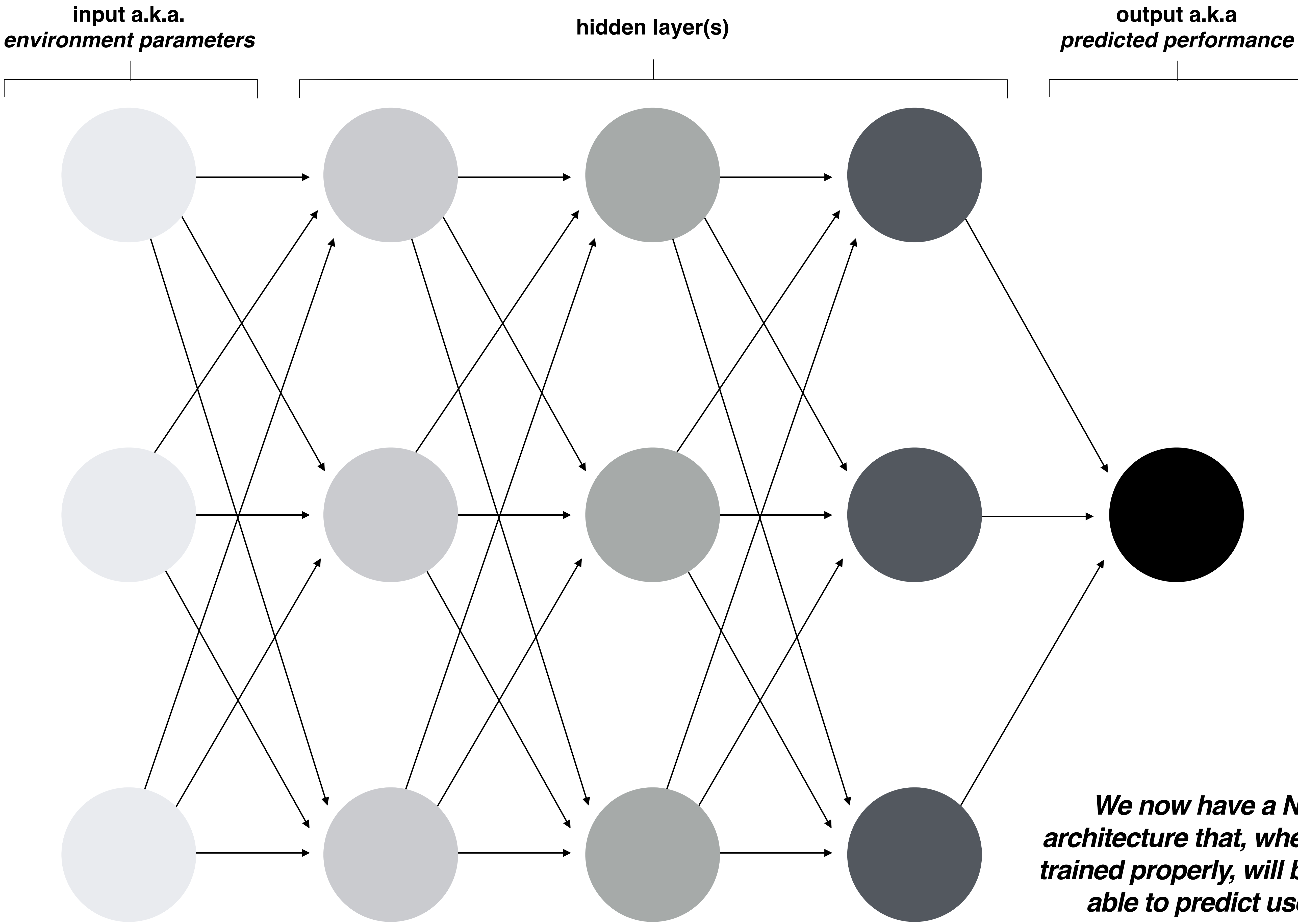
modify the environment parameters based on our *performance objective*

objective #1

predict a user's performance based on environment parameters



The input to our NN will be the environment parameters, and the output will be the user's performance under those environment conditions



We now have a NN architecture that, when trained properly, will be able to predict user performance given a set of environment parameters

objective #2
modify the environment parameters based on a performance objective

how can we use a NN to modify the environment parameters in pursuit of some performance objective?

Backpropagation: The NN Powerhouse

The Concept

We previously looked at the metaphorical "machine" and how we could think about NN weights simplistically like a bunch of dials. You may have been asking: how do we know which way to turn the dials, and how much? The answer is backpropagation and gradient descent.

In order to adjust the NN weights to values that achieve higher accuracy (i.e. lower cost), we use partial derivatives to iteratively take "steps" (i.e. weight adjustments, dial "turns") toward some goal. In the case of our NN, we want to adjust the weight values to iteratively lower our cost, thus achieving greater predictive accuracy.

11. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

Backpropagation: The NN Powerhouse

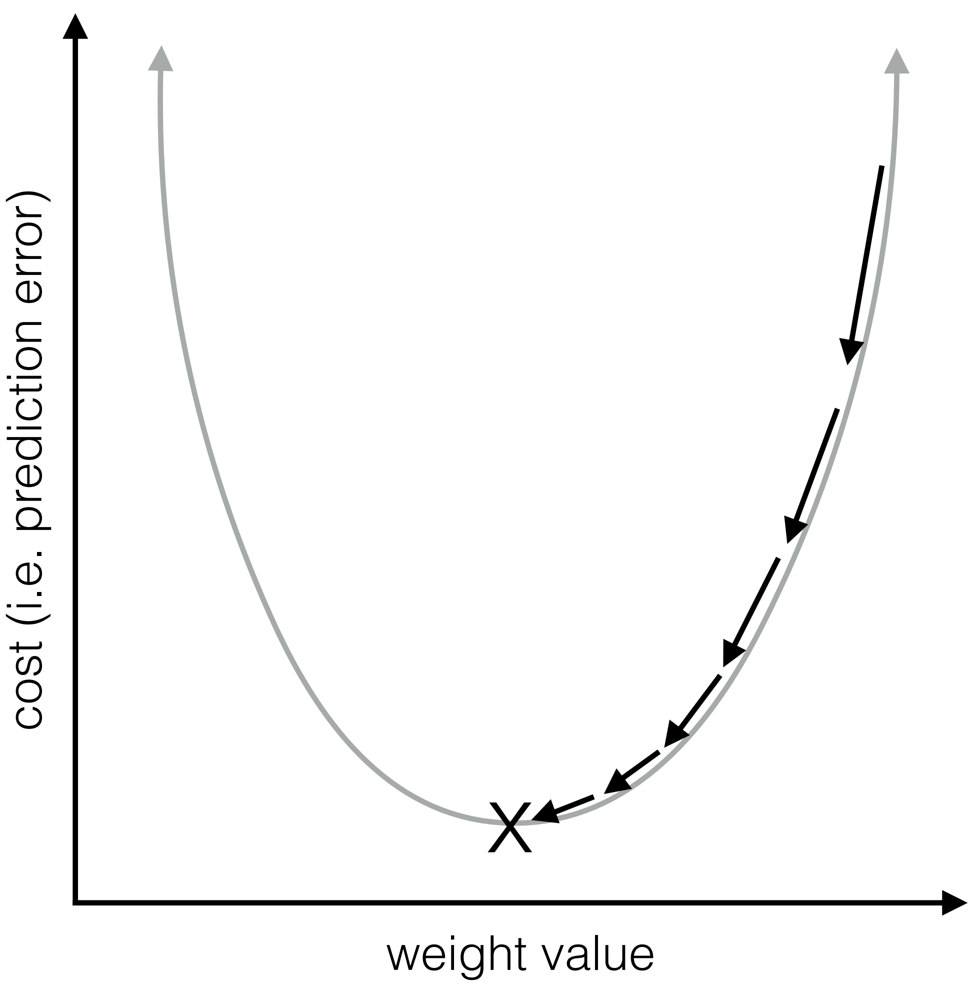
The Concept

Hinton et. al (1986) describe backpropagation:

"The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units."[11]

11. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

Backpropagation: The NN Powerhouse



Gradient Descent & Cost Minimization

Each time the backpropagation algorithm updates the weights of the net, the following occurs:

1) calculate the gradient for a hidden layer (i.e. the partial derivative of a single weight value w.r.t to the cost function)
2) multiply each gradient value by some static learning rate value (a pre-determined constant number)
3) subtract each weight by its corresponding gradient value

This process constitutes a single "step" of gradient descent. The goal of this process is to take each weight from some randomized starting value to a global minima in the cost function. The learning rate of the network determines the "step size" taken at each iteration.

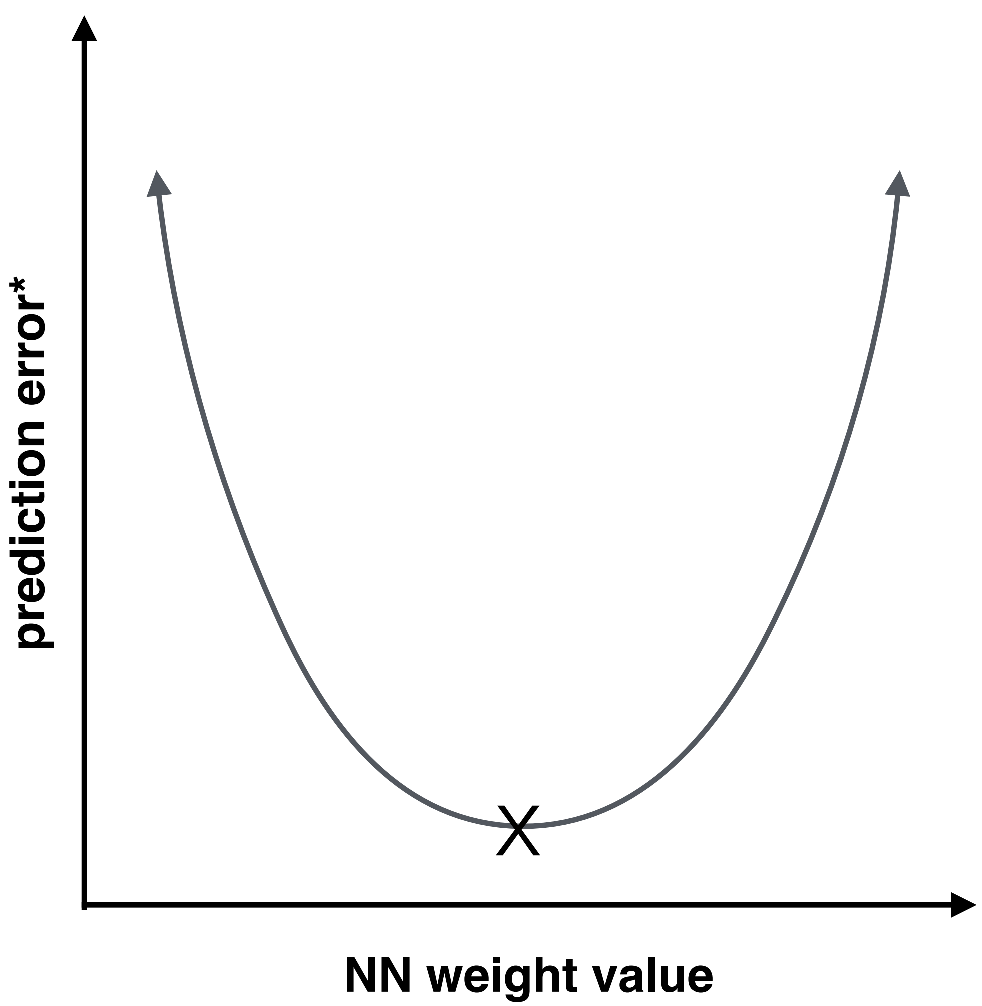
11. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

we can use this same method of gradient descent to modify the environment parameters

NN Weight Values	Environment Parameters
cost definition (NN prediction - expected output)	**cost definition** (NN prediction - performance goal)
gradient calculation partial derivative of each weight value in the hidden layer(s) w.r.t the cost function	**gradient calculation** partial derivative of each input value (i.e. environment parameter) w.r.t to the performance delta
optimization goal iteratively decrease the difference between the NN output and the actual user performance	**optimization goal** iteratively decrease the difference between the NN output and the user performance goal

NN Weight Values

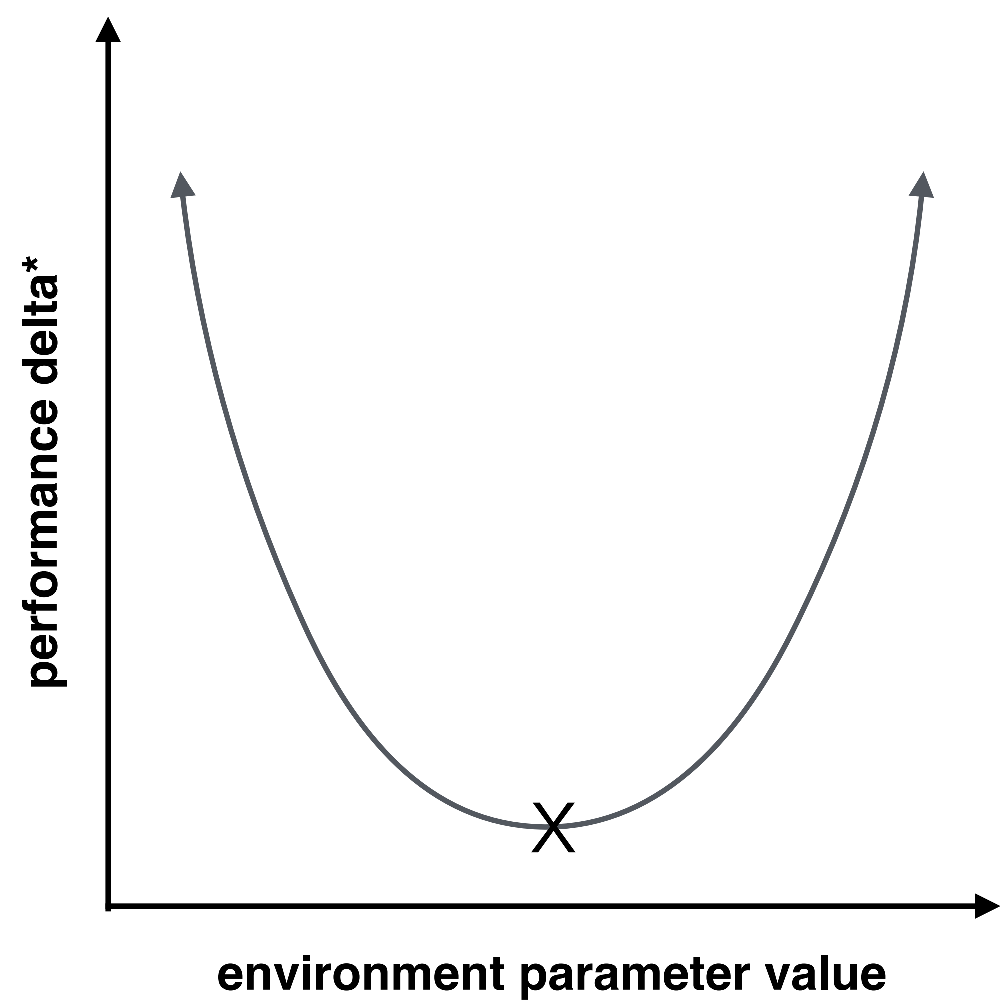
X = global minimum



* where the prediction error is defined as the difference between the NN's prediction and the correct/actual user performance

Environment Parameters

X = global minimum



* where the performance delta is defined as the difference between the NN's prediction of the user's performance and the performance goal

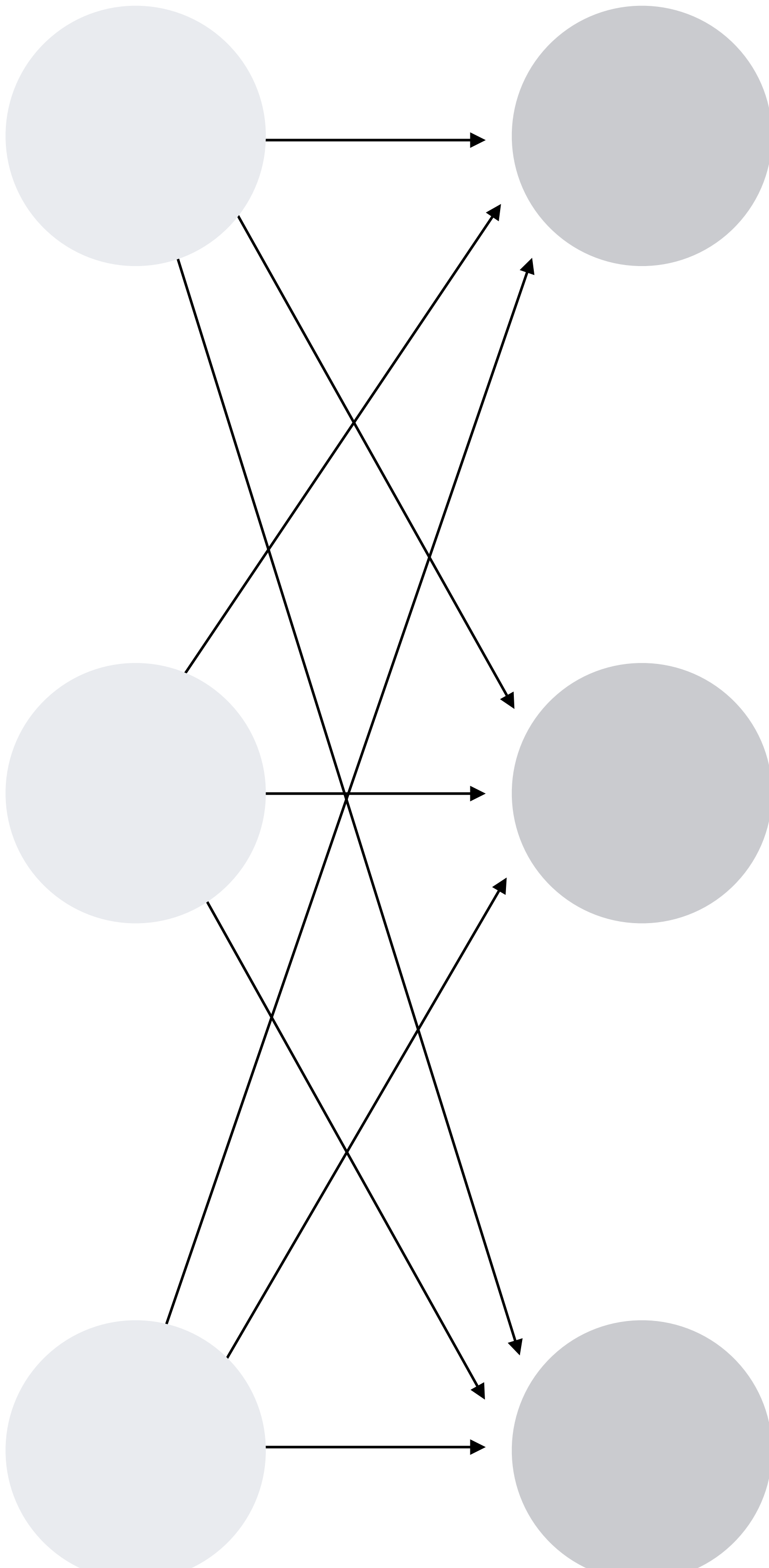
now that we have met both our objectives, we can see what our full model will look like

ModelArchitecture

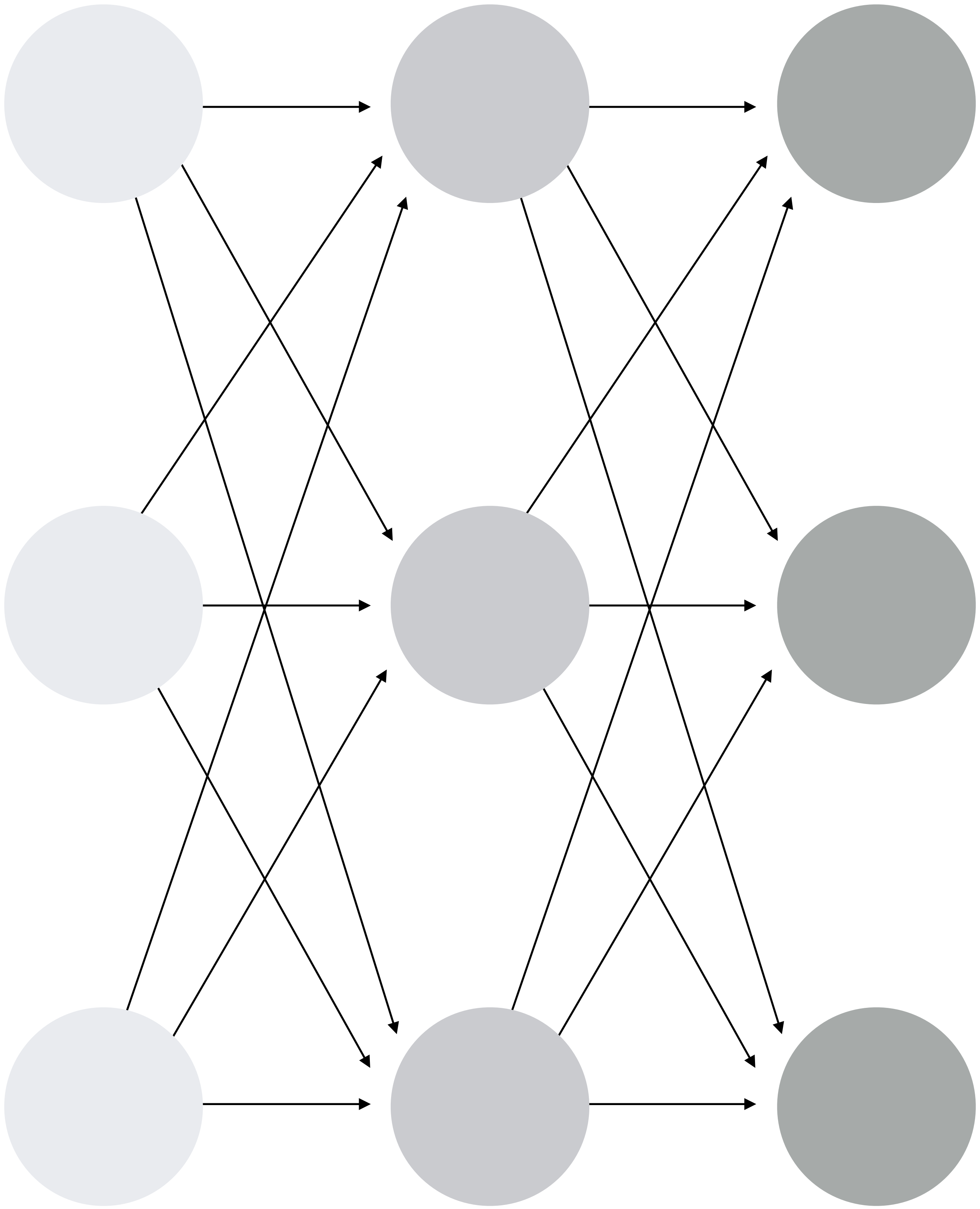
#1 Collect a new dataset that contains the current environment parameter settings & corresponding user performance

#2

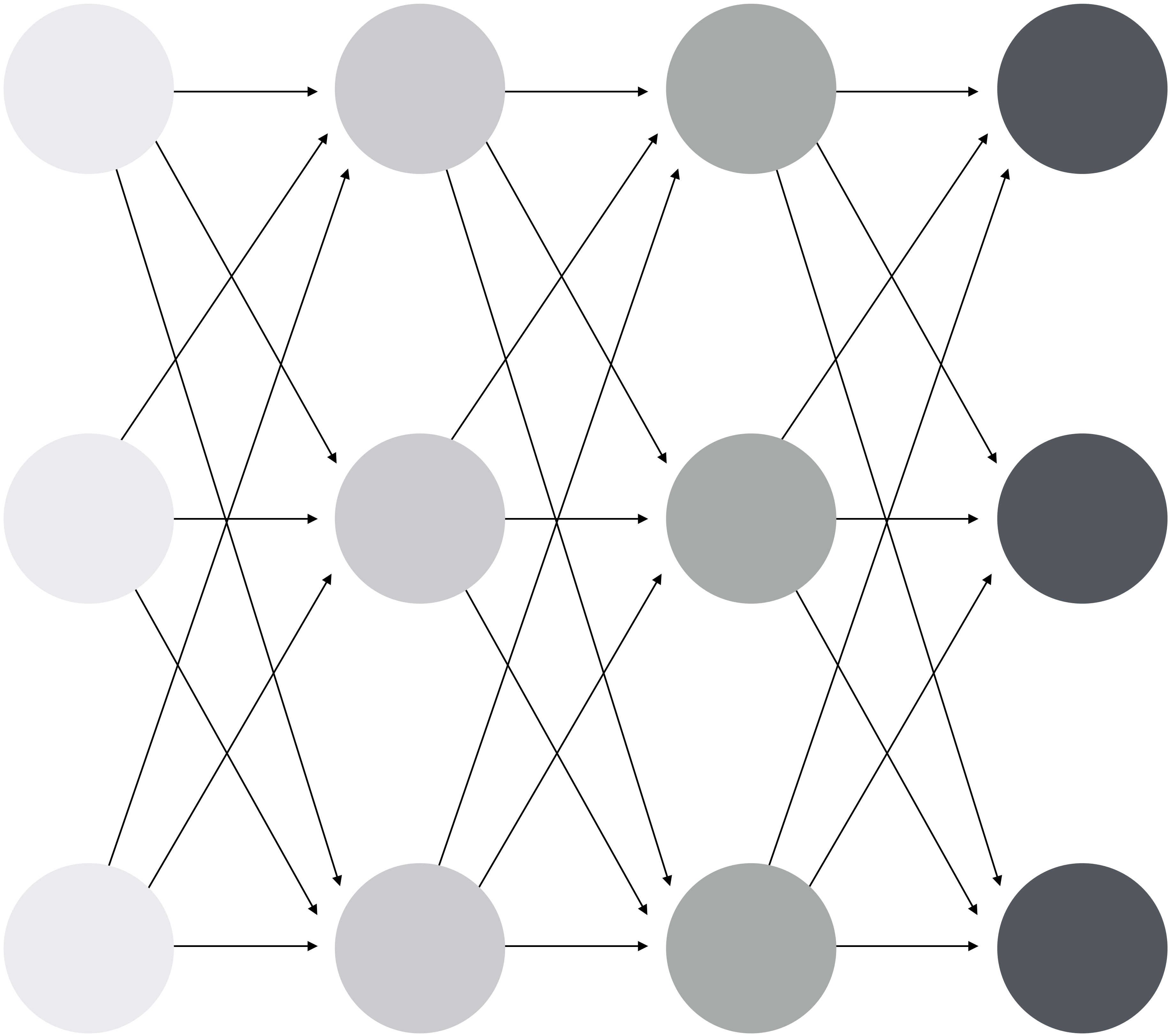
Feed this input through the neural network, using the current weight values (initially random values between 0 and 1)



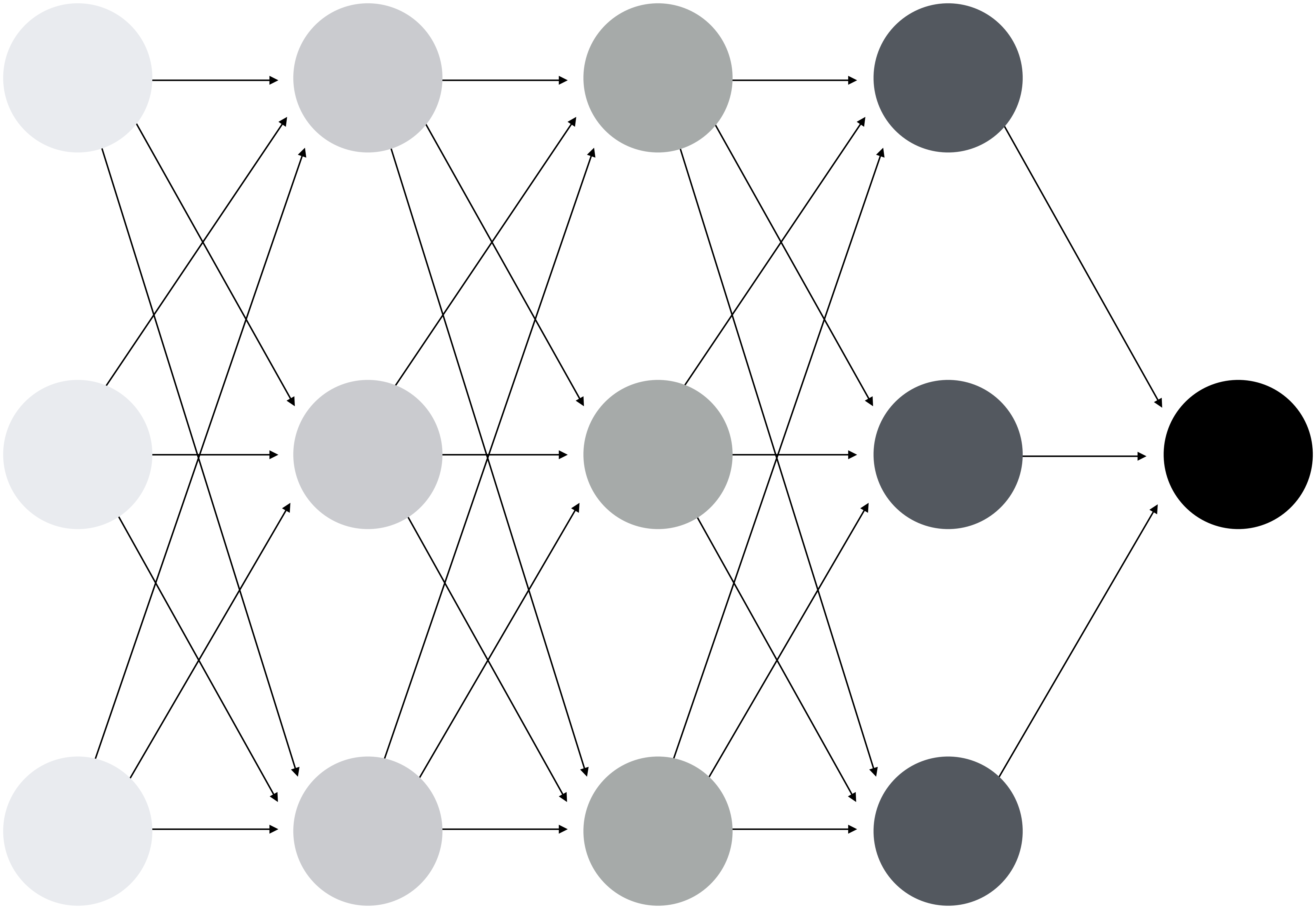
#2 Feed this input through the neural network, using the current weight values (initially random values between 0 and 1)



#2 **Feed this input through the neural network, using the current weight values (initially random values between 0 and 1)**

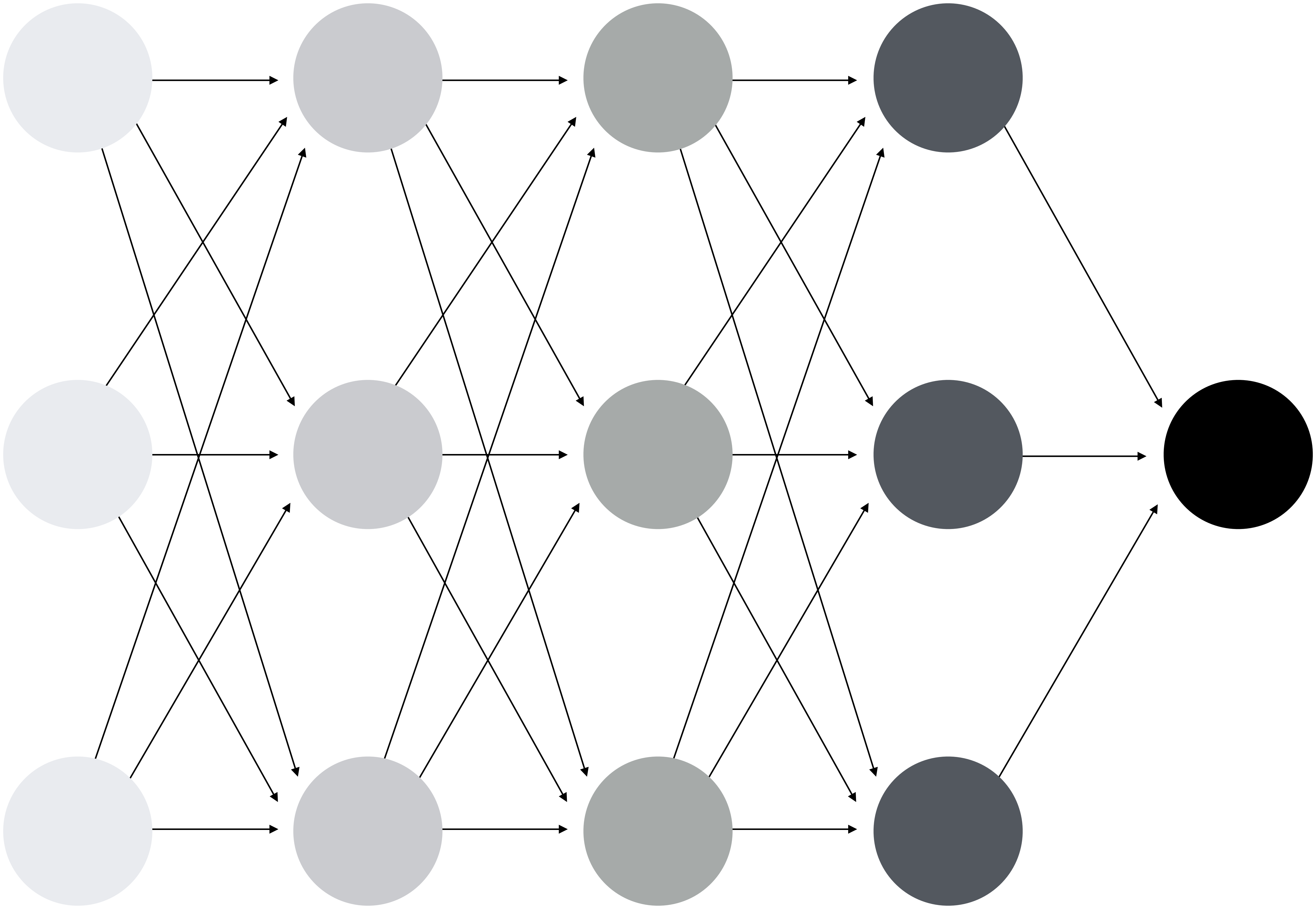


#3 Retrieve the predicted output from the NN (for the environment parameters it was given)



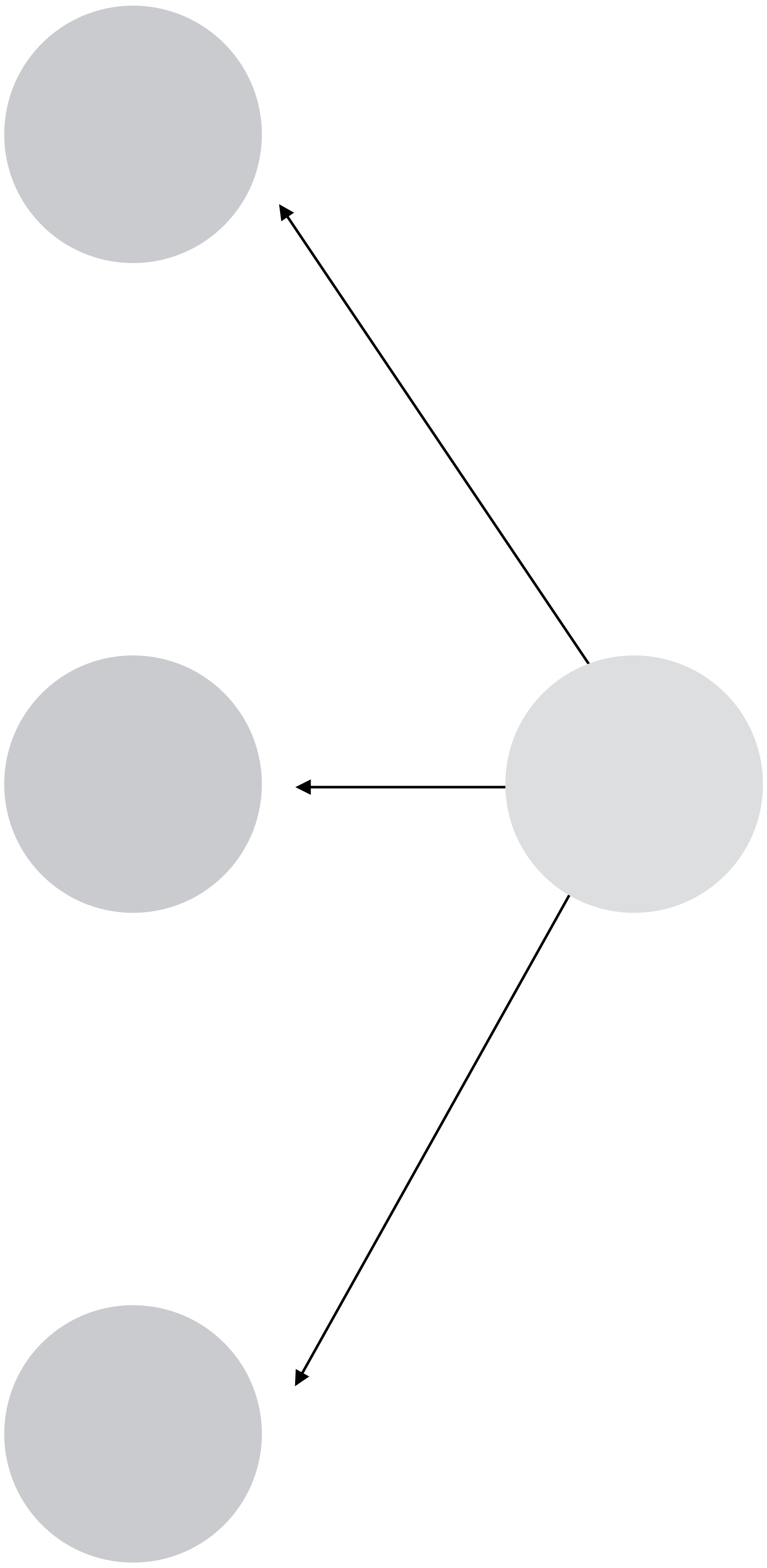
#4

Calculate the error w.r.t to the weight values of the network
Calculate the performance delta w.r.t to the environment parameters

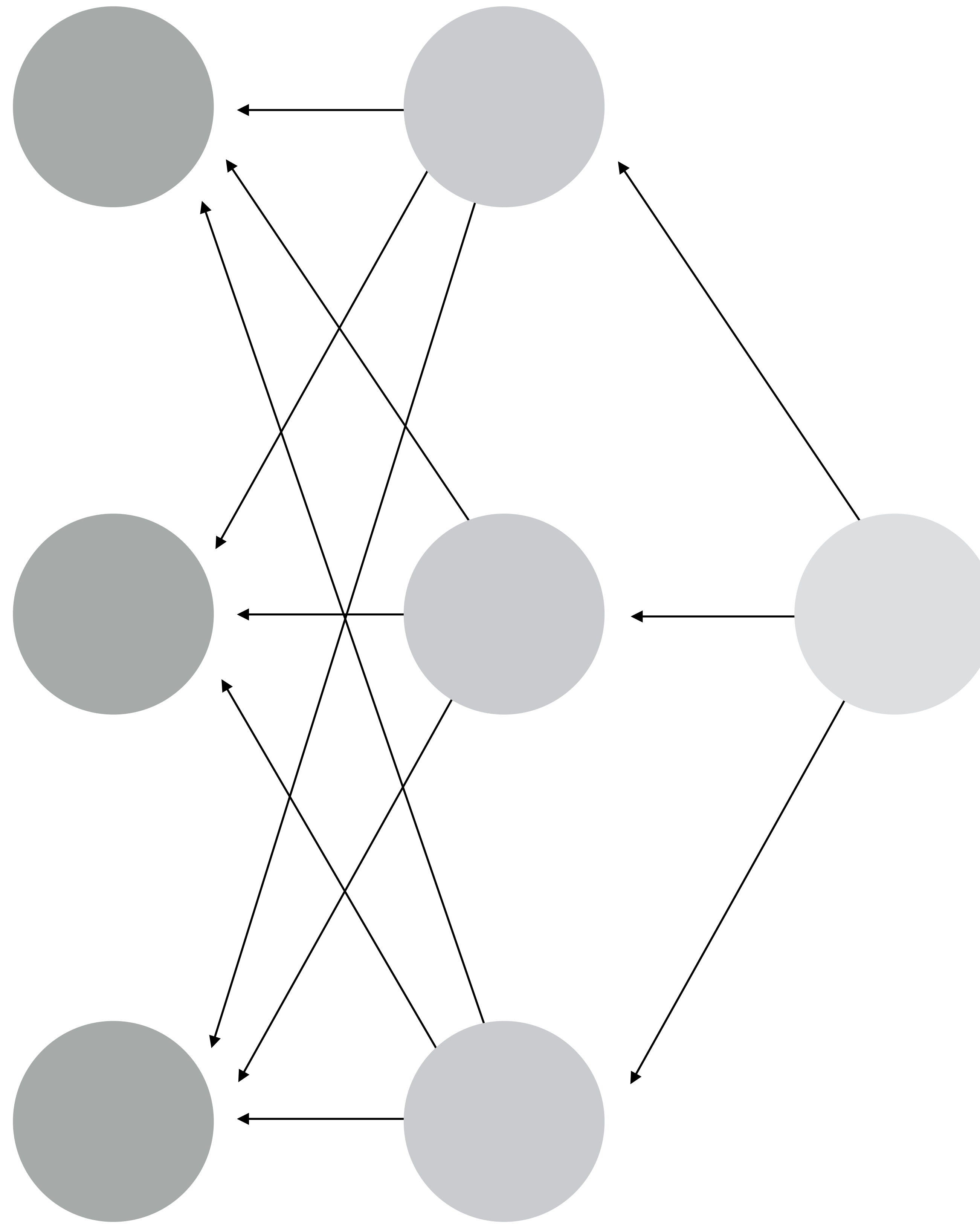


#5

Backpropagate the error of the weight values, modifying each value by the (gradient * learning rate)

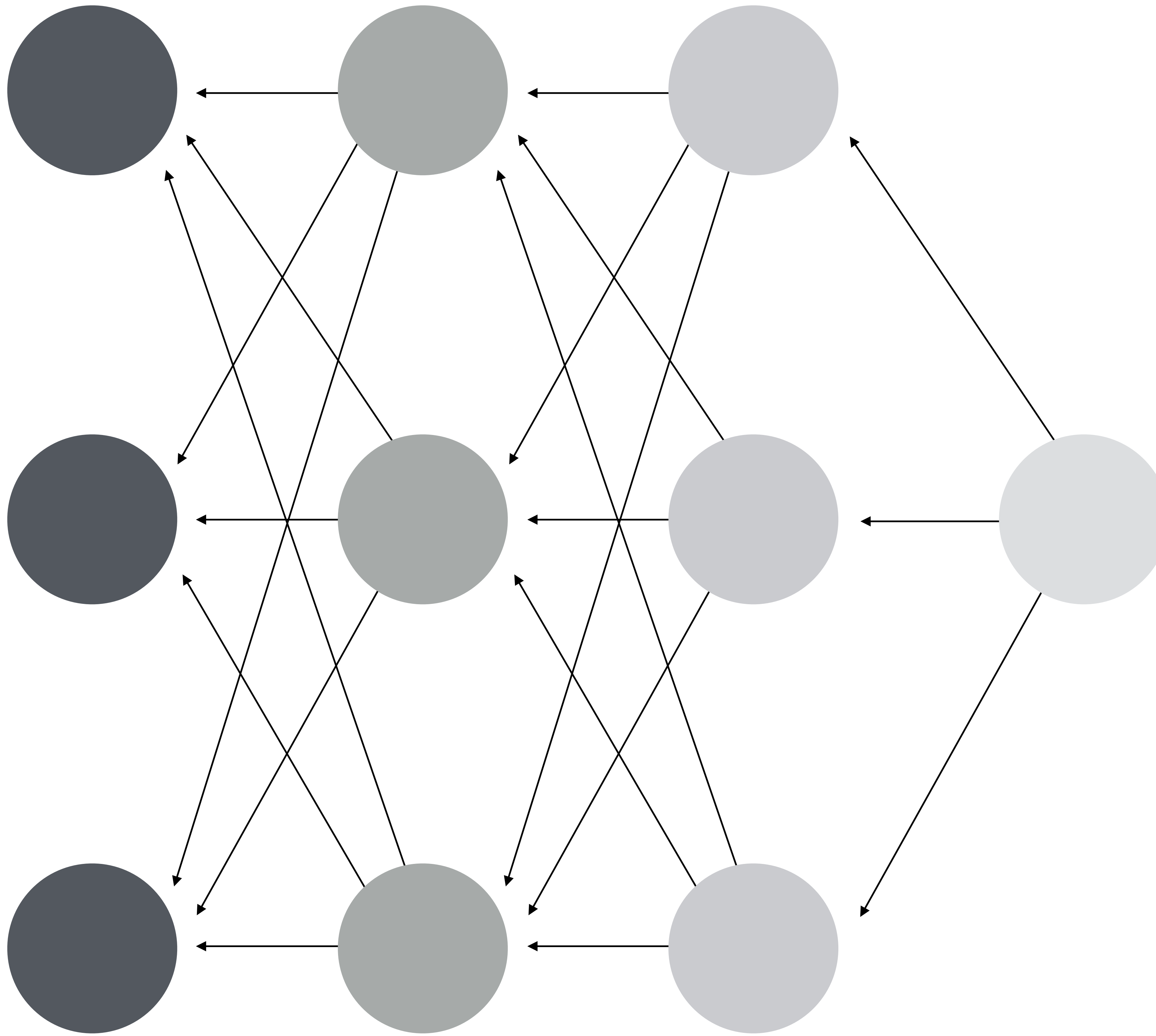


#5 Backpropagate the error of the weight values, modifying each value by the (gradient * learning rate)



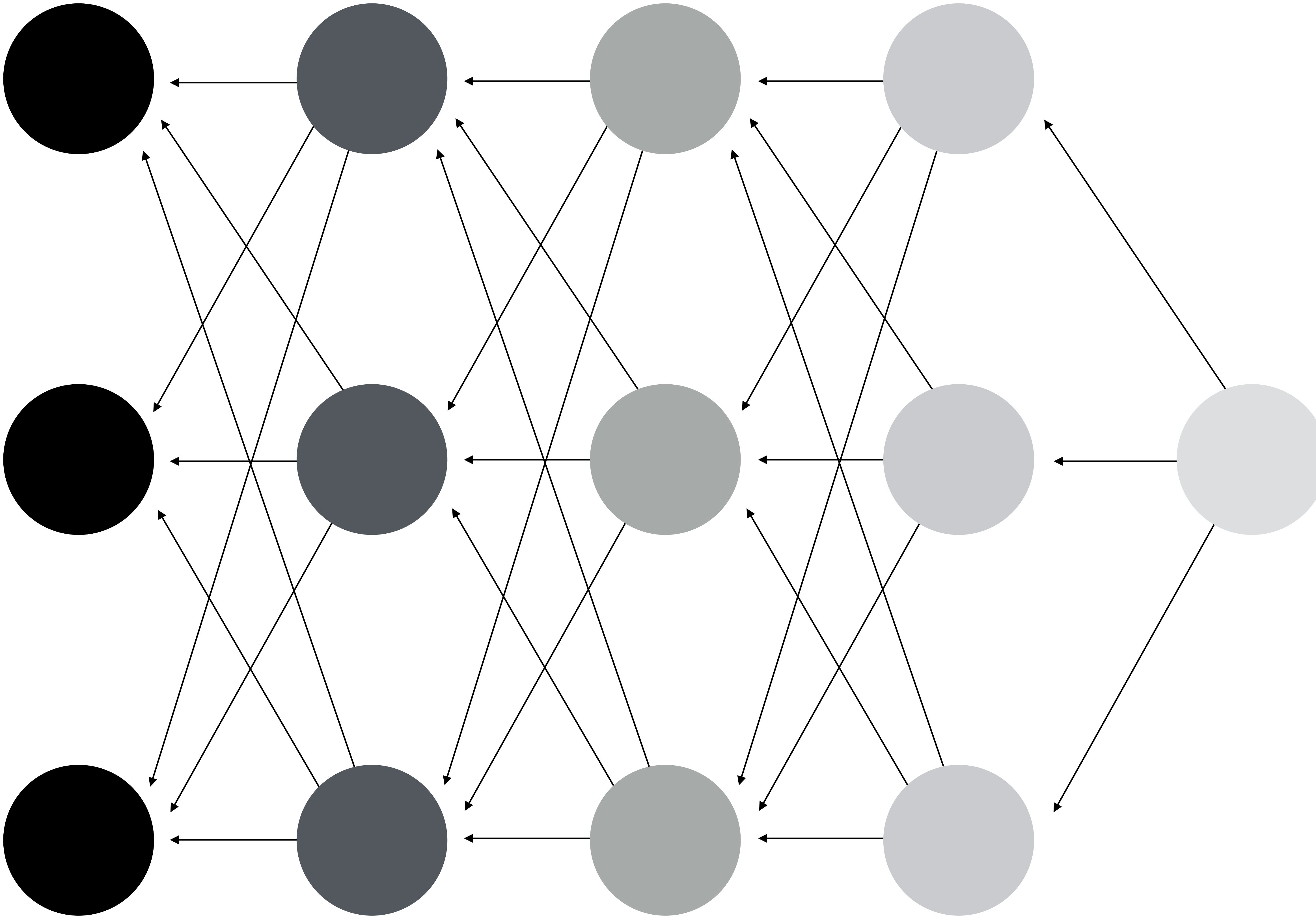
#5

Backpropagate the error of the weight values, modifying each value by the (gradient * learning rate)



#6

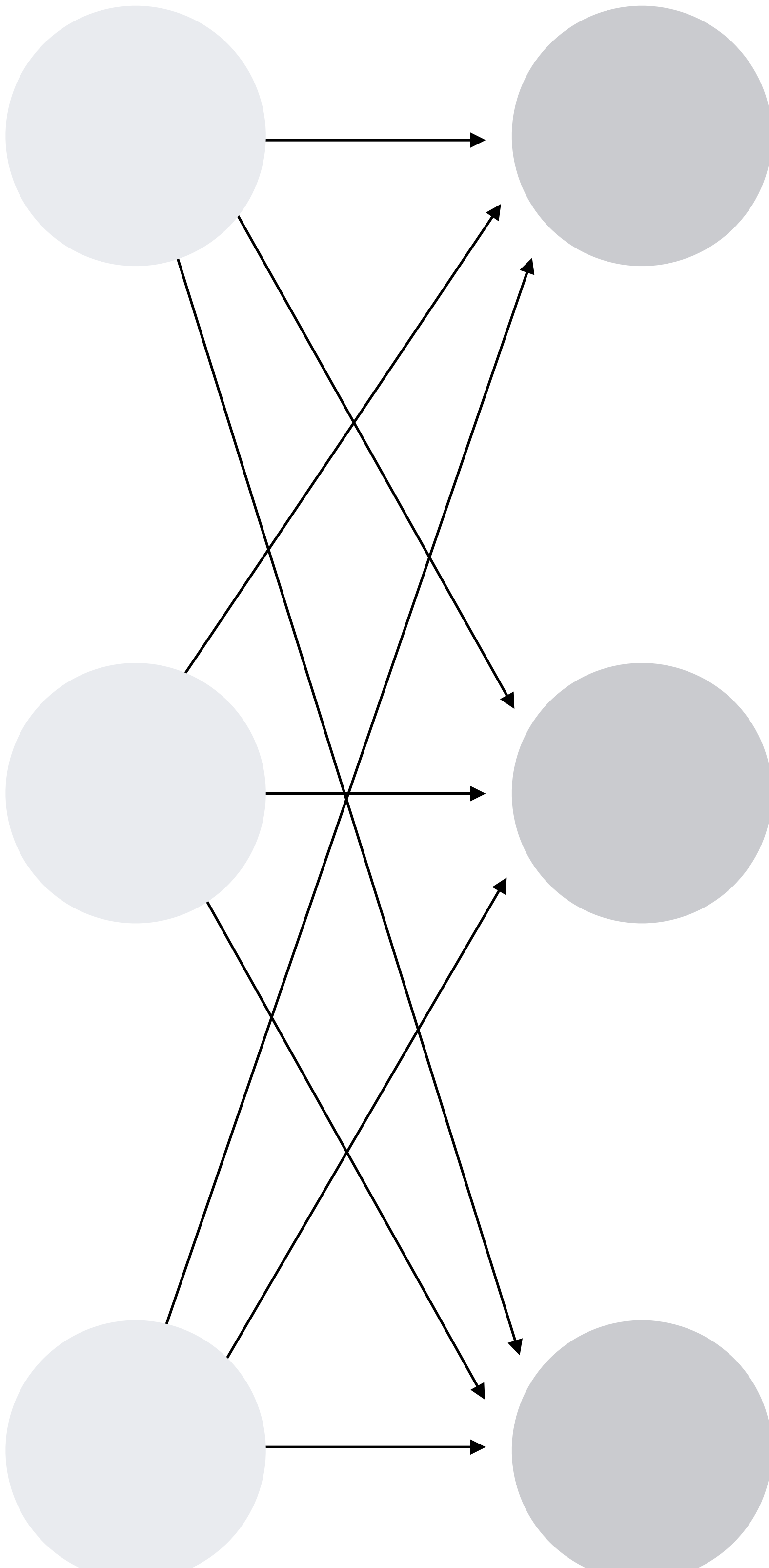
Modify the environment parameter values provided to the net via the gradients calculated from the performance delta



#7 We can now repeat this process by collecting a new dataset with the modified environment parameters and repeating the process

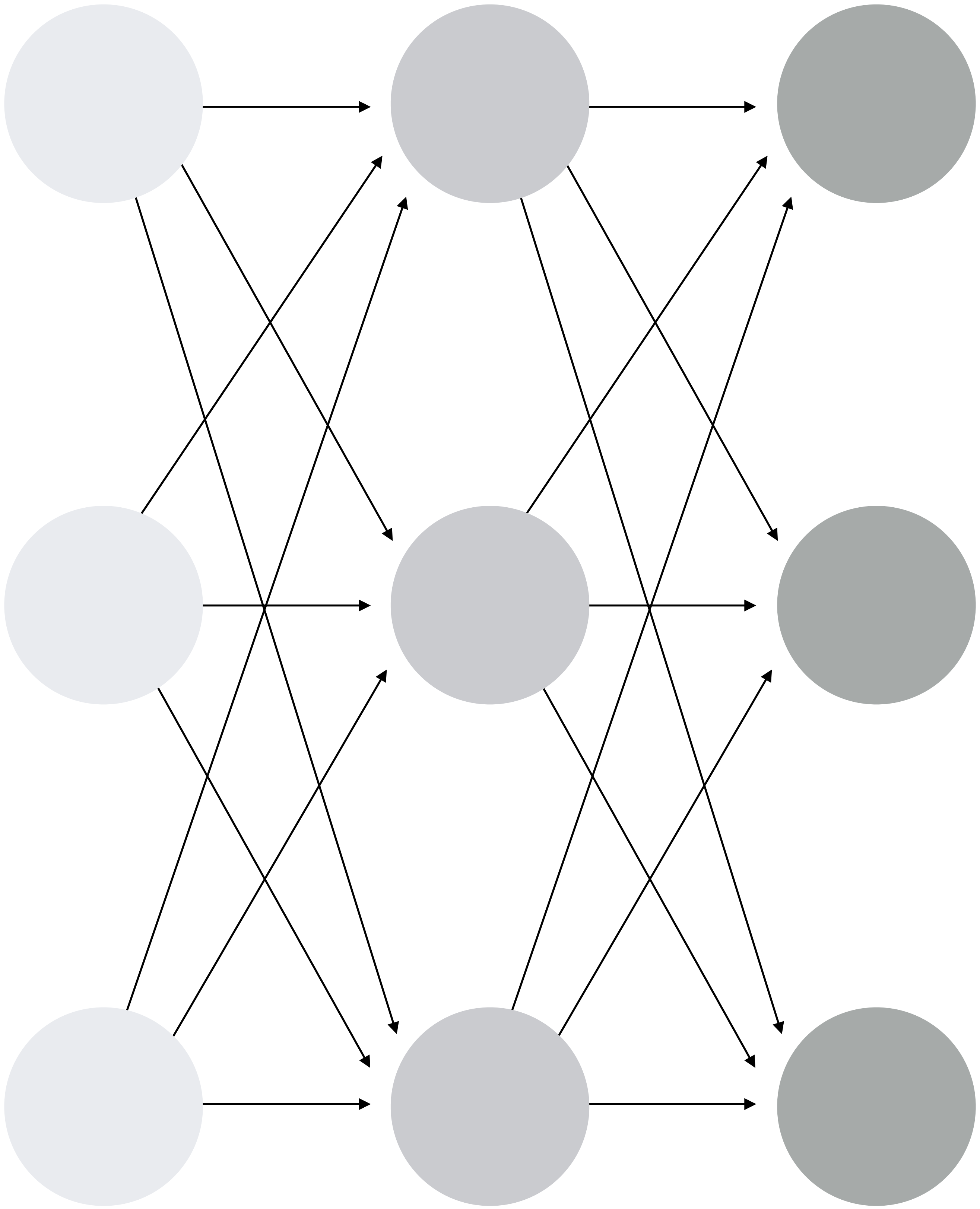
#7

We can now repeat this process by collecting a new dataset with the modified environment parameters and repeating the process

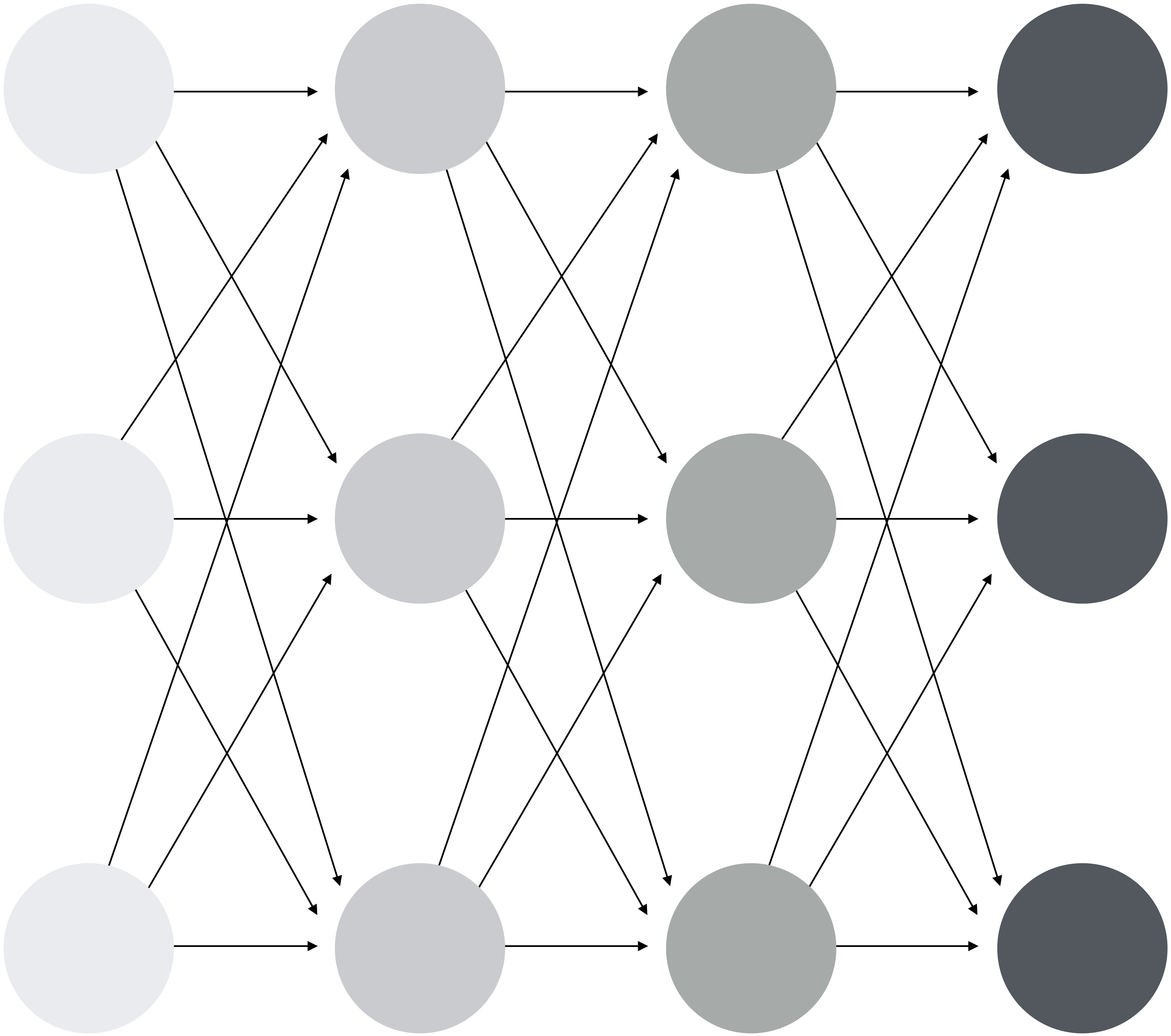


#7

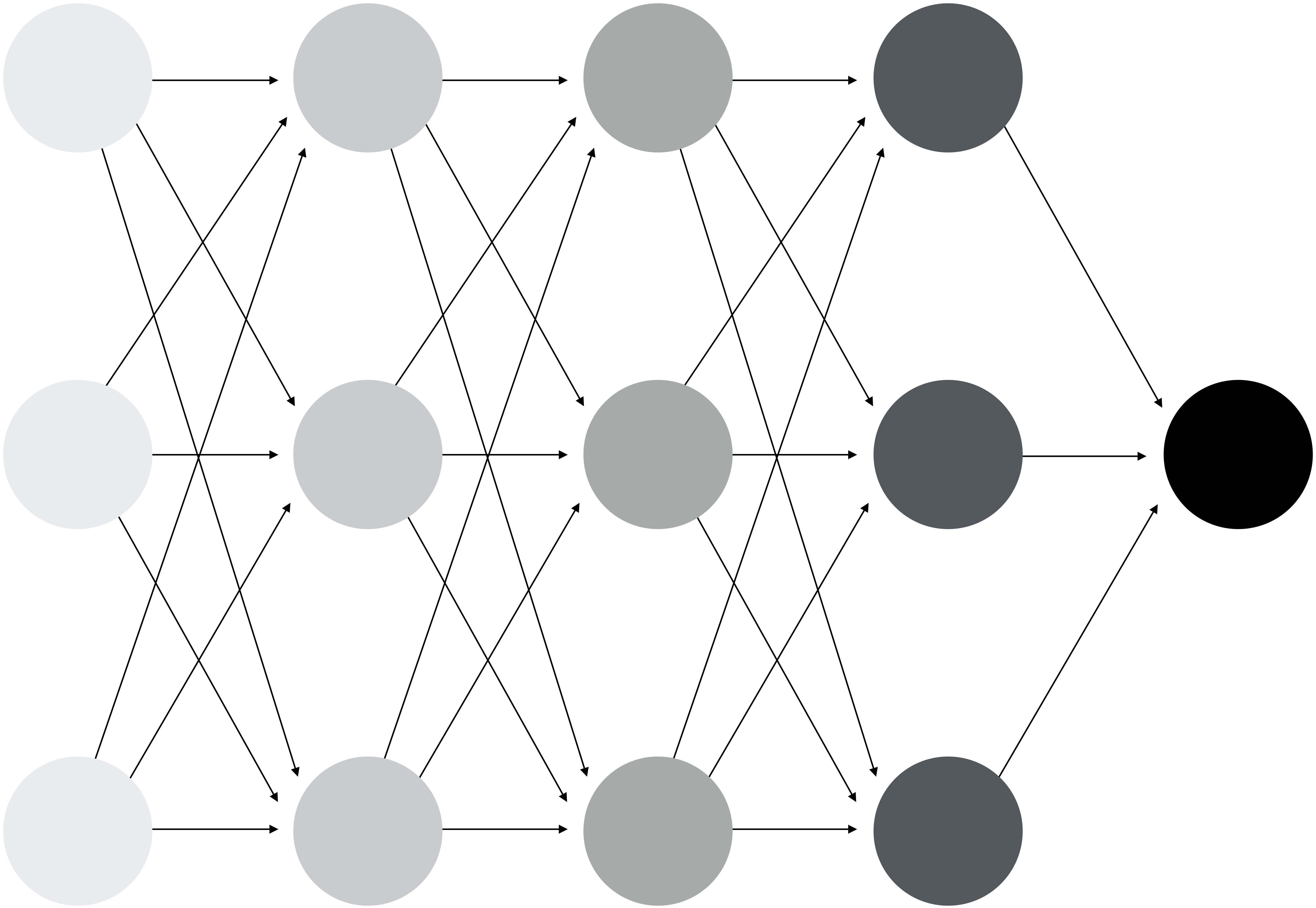
We can now repeat this process by collecting a new dataset with the modified environment parameters and repeating the process



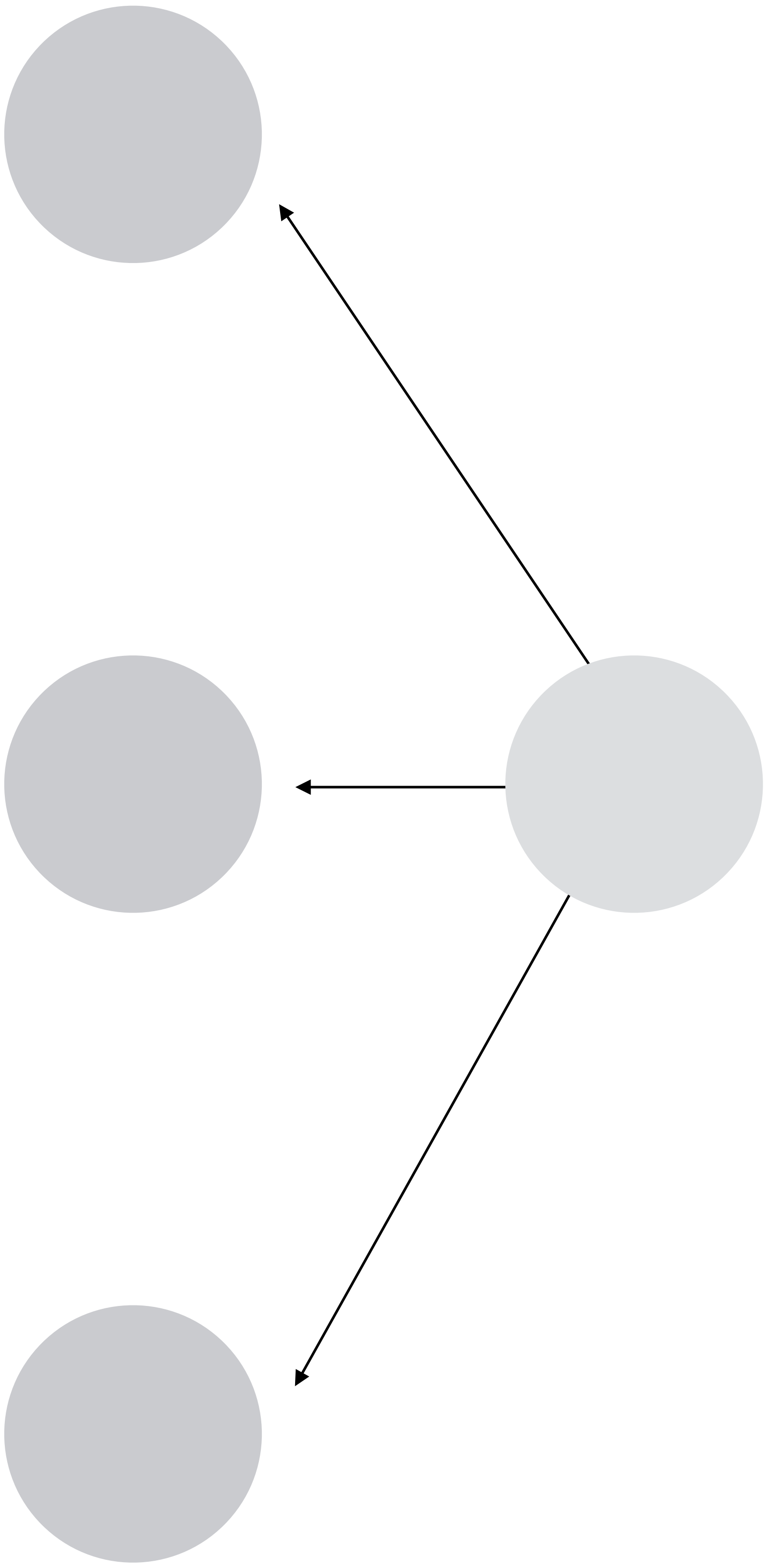
#7 **We can now repeat this process by collecting a new dataset with the modified environment parameters and repeating the process**



#7 **We can now repeat this process by collecting a new dataset with the modified environment parameters and repeating the process**

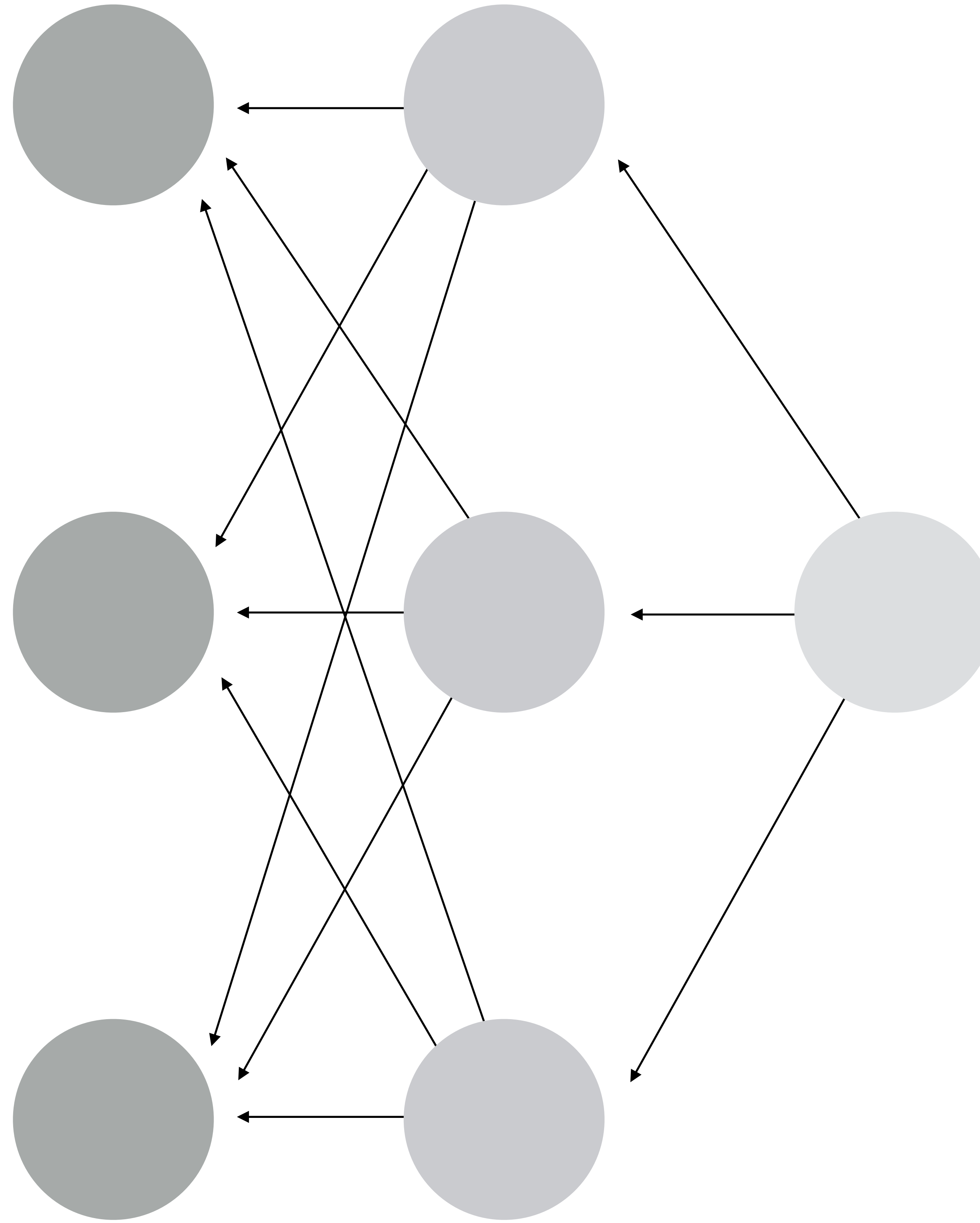


#7 **We can now repeat this process by collecting a new dataset with the modified environment parameters and repeating the process**

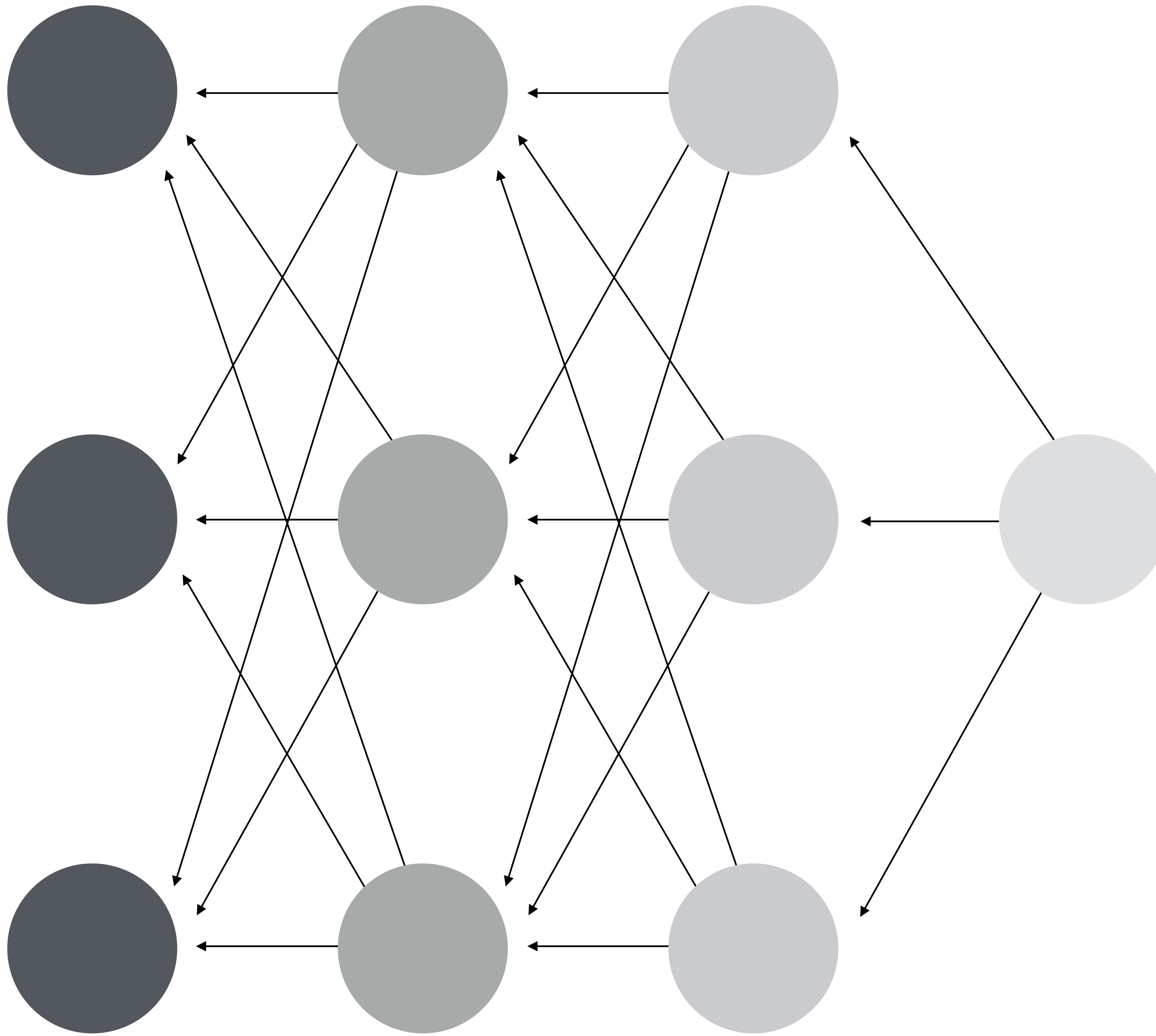


#7

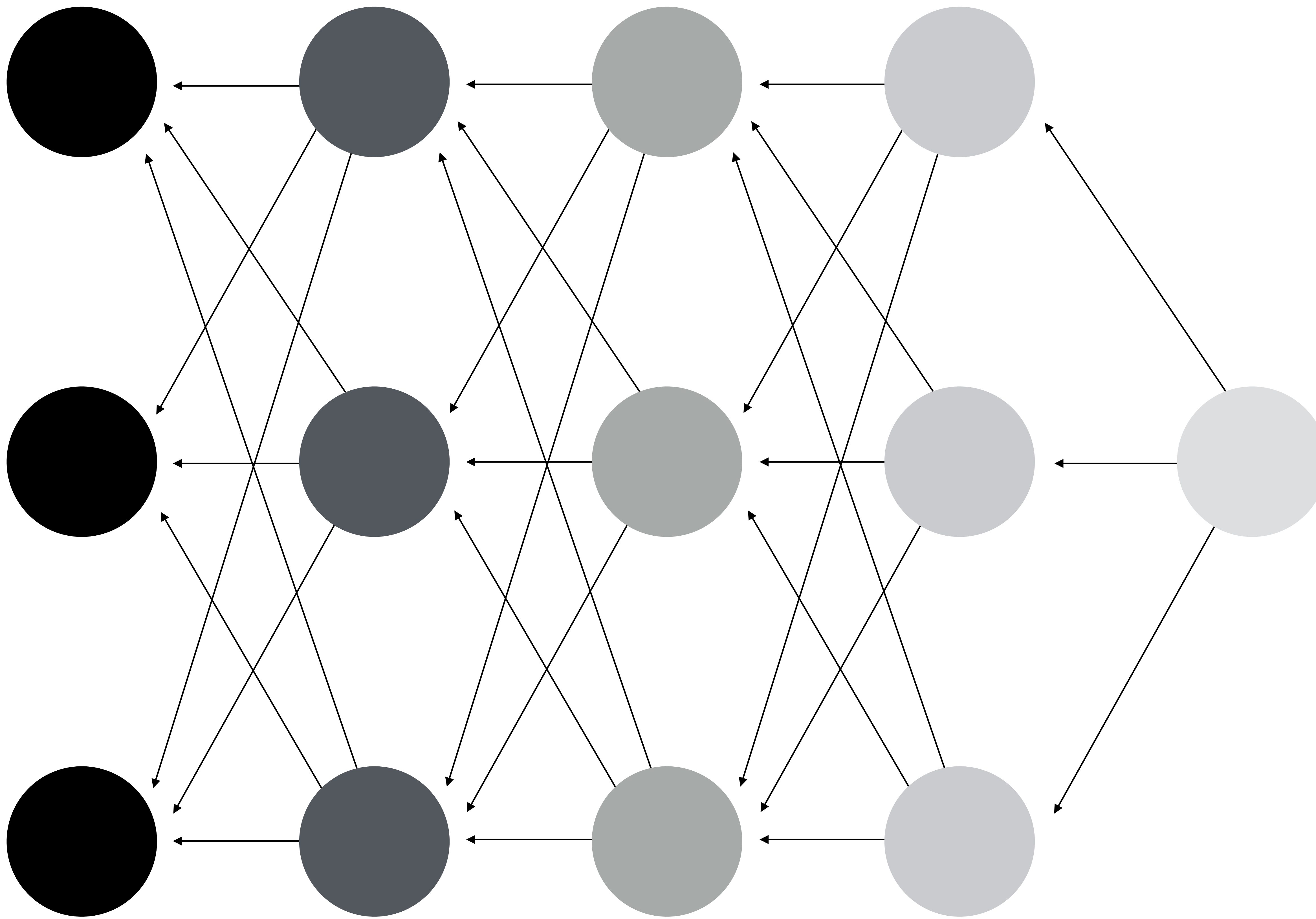
We can now repeat this process by collecting a new dataset with the modified environment parameters and repeating the process



#7 **We can now repeat this process by collecting a new dataset with the modified environment parameters and repeating the process**

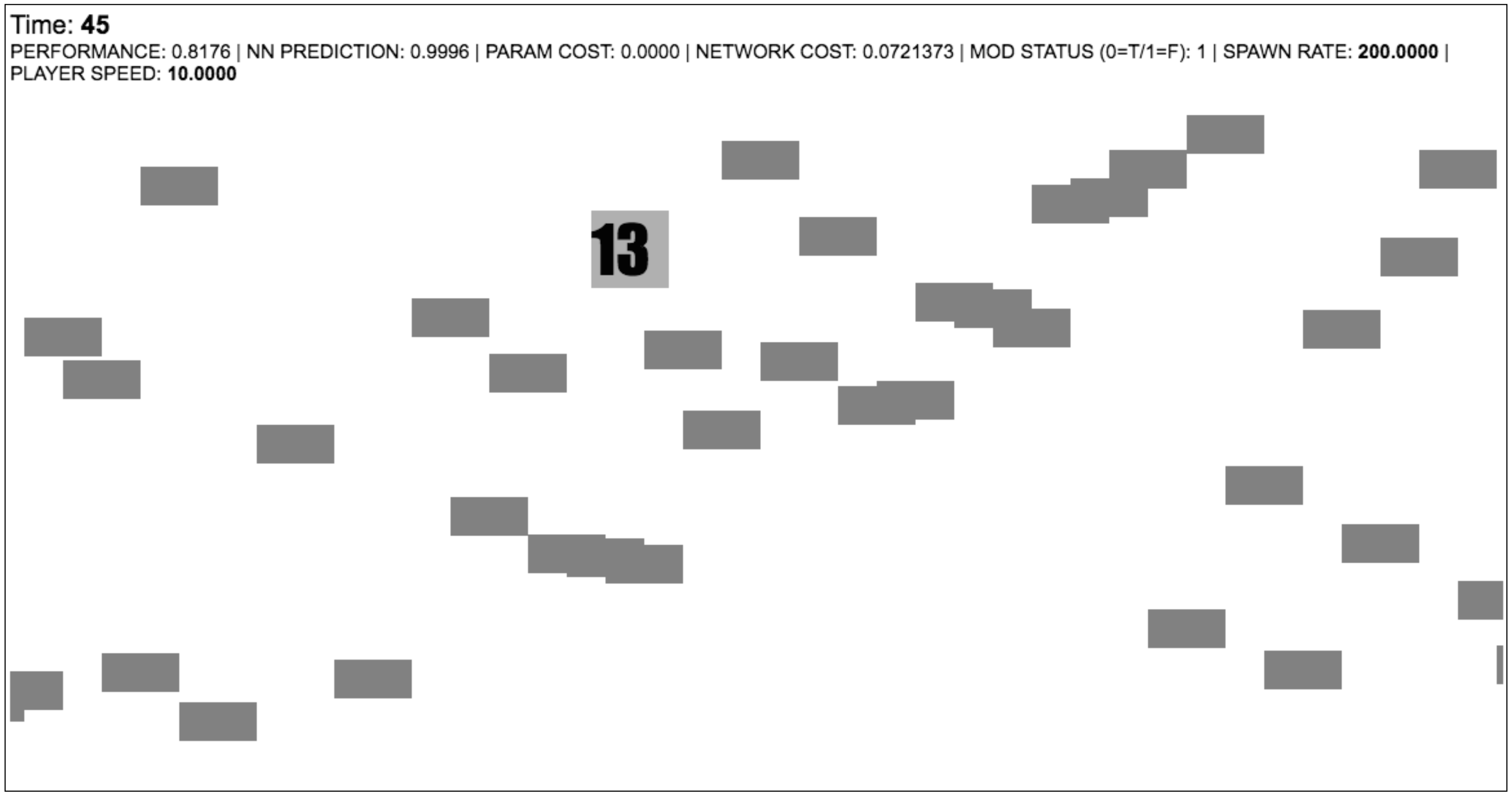


#7 **We can now repeat this process by collecting a new dataset with the modified environment parameters and repeating the process**



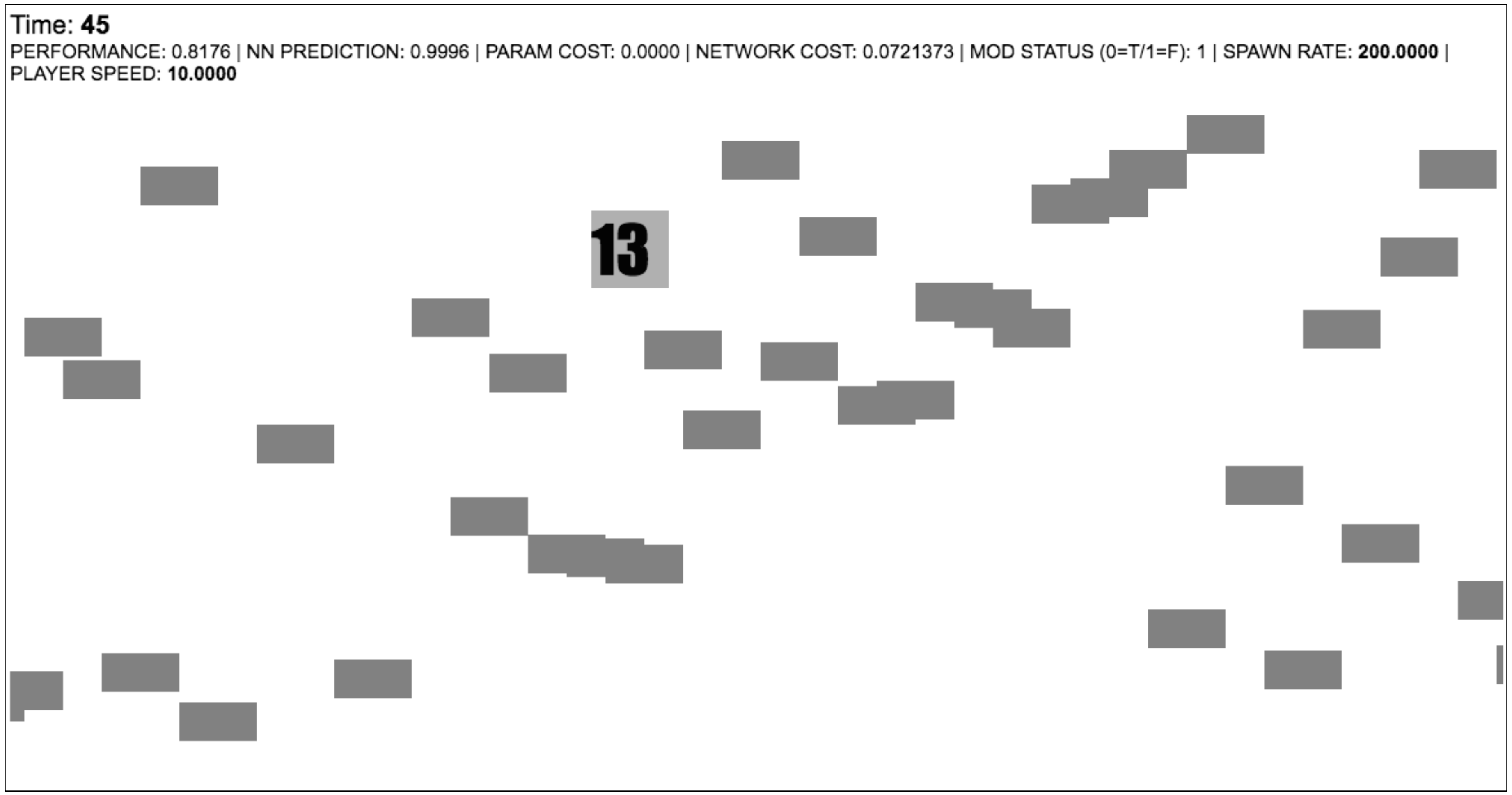
now that we have our model, let's implement it in code and see how it performs!

but first, we need a testing environment…



Avoidance: A Simple Testing Environment

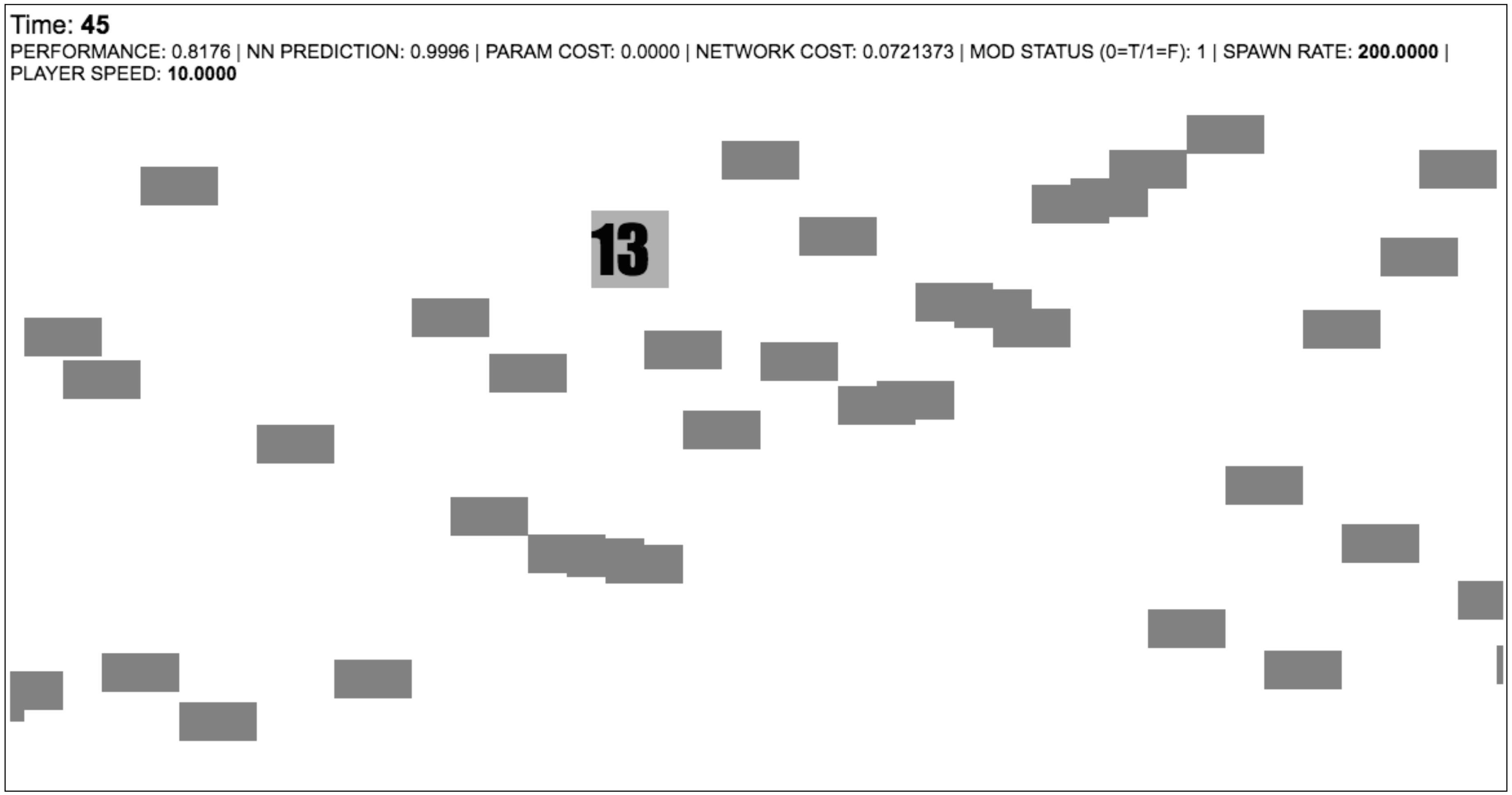
For the purposes of testing, I created a game called "Avoidance." The object of the game is to collide with as few enemy blocks as possible. The player controls a small block using the UP, DOWN, LEFT and RIGHT keys.



Avoidance: Client-Side Tools

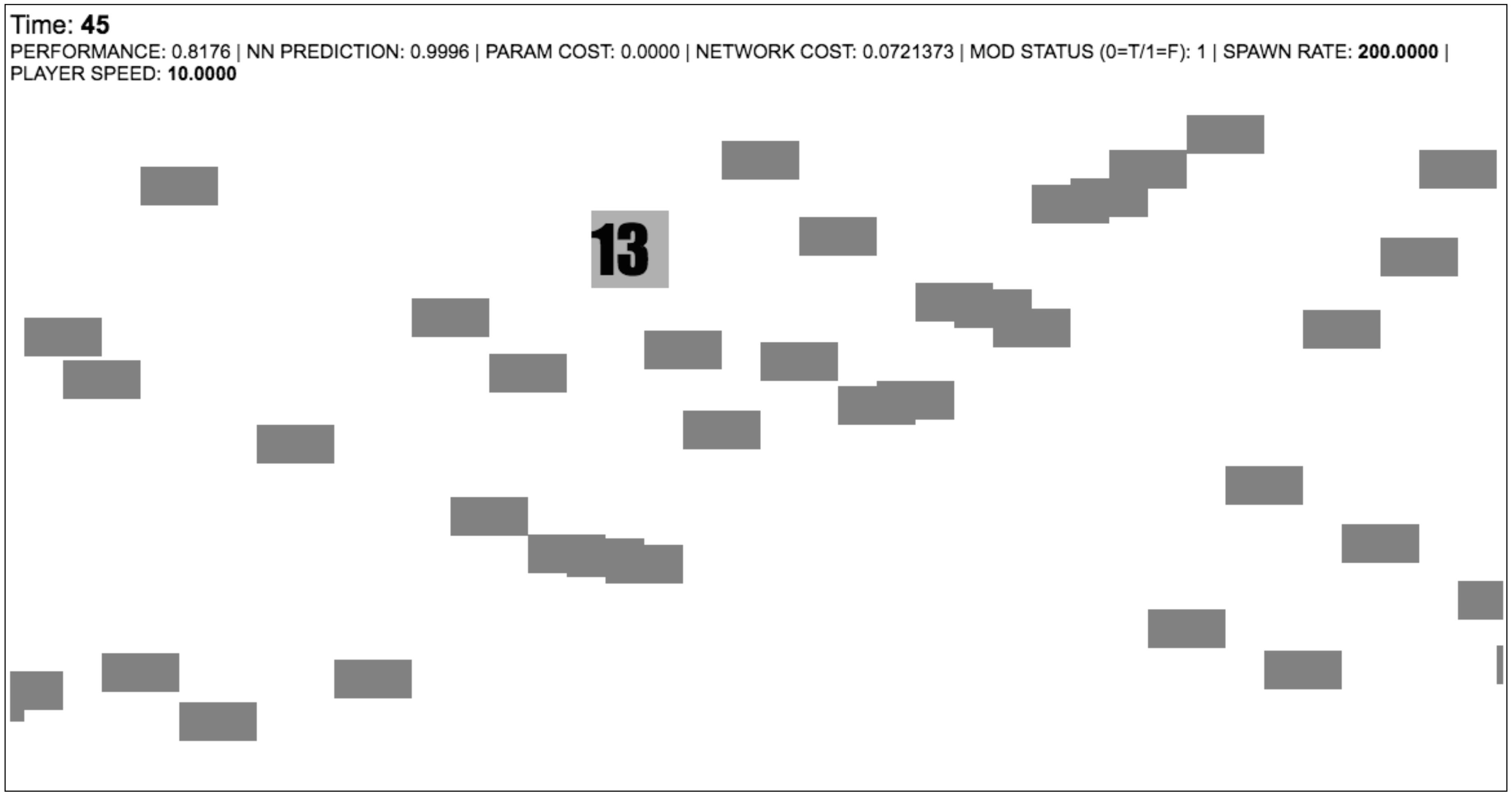
This game runs in the browser. The game logic is written in Javascript, using an open-source library called GameQuery[12]. All the entities are GameQuery sprites, which are essentially wrappers for HTML elements that can be modified as the game plays.

12. gameQuery - a javascript game engine with jQuery. (n.d.). Retrieved from http://gamequeryjs.com/documentation/



Avoidance: Server-Side Tools

The server-side code is written in Node JS. All of the code necessary to run the game is stored in a directory on an Amazon AWS EC2 instance, from which the server can be launched.



Avoidance: Neural Networking

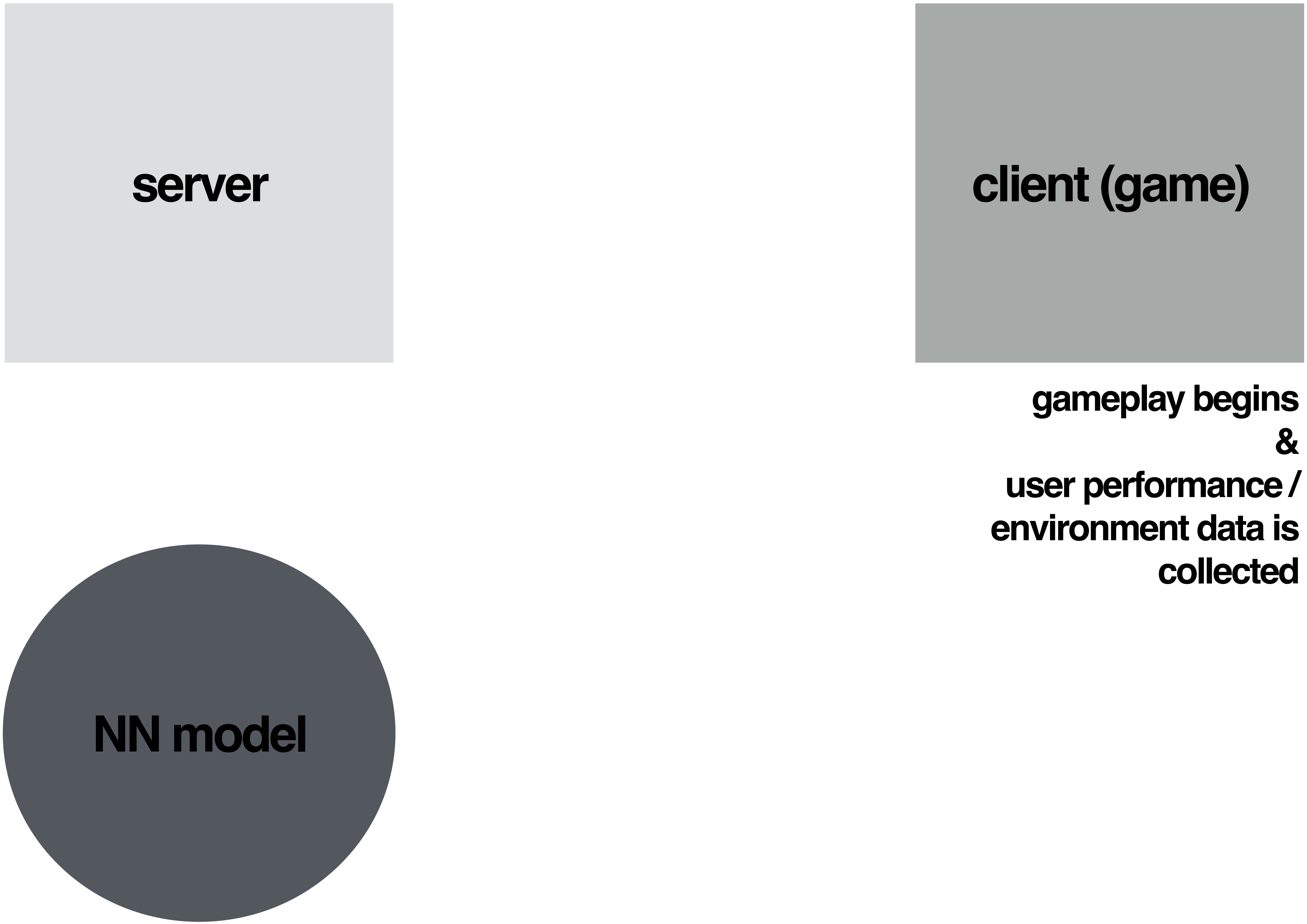
The actual NN code is written in Python, using a popular machine learning library called Theano[13]. Theano uses dynamic C code generation so it performs integral NN tasks (such as gradient calculation) fast and efficiently.

13. The Theano Development Team, et al. "Theano: A Python framework for fast computation of mathematical expressions." (2016).

now that we have our model, let's implement it in code and see how it performs!

GameArchitecture

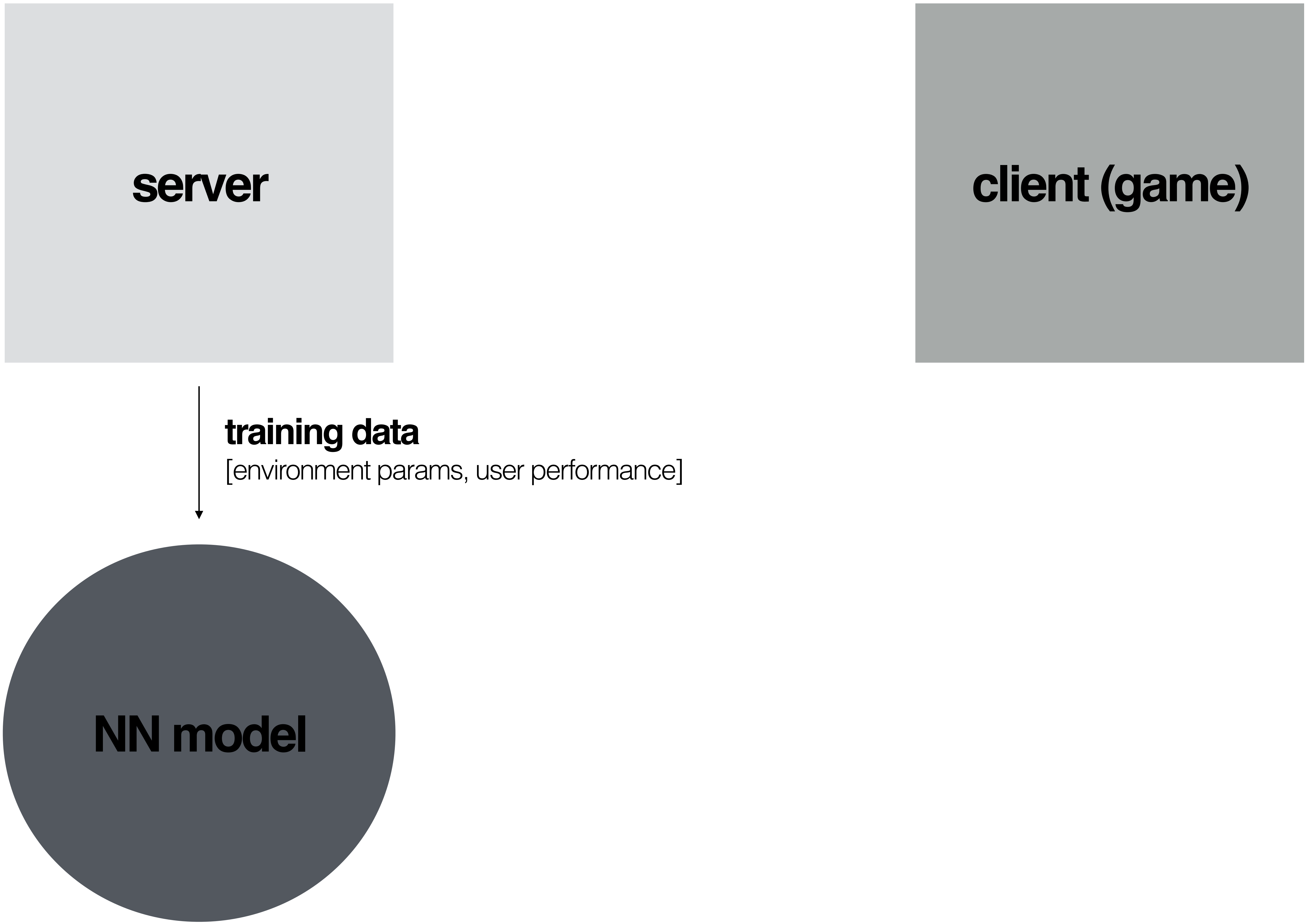
avoidance: learning loop & architecture



avoidance: learning loop & architecture


server
client (game)
training data
[environment params, user performance]
NOTE: this constitutes one set of environment parameter values
and the corresponding user performance given that environment
NN model

avoidance: learning loop & architecture



avoidance: learning loop & architecture

server

client (game)

NN model

feed input data into NN to get predicted performance
&
backpropagate error / update weight values

avoidance: learning loop & architecture

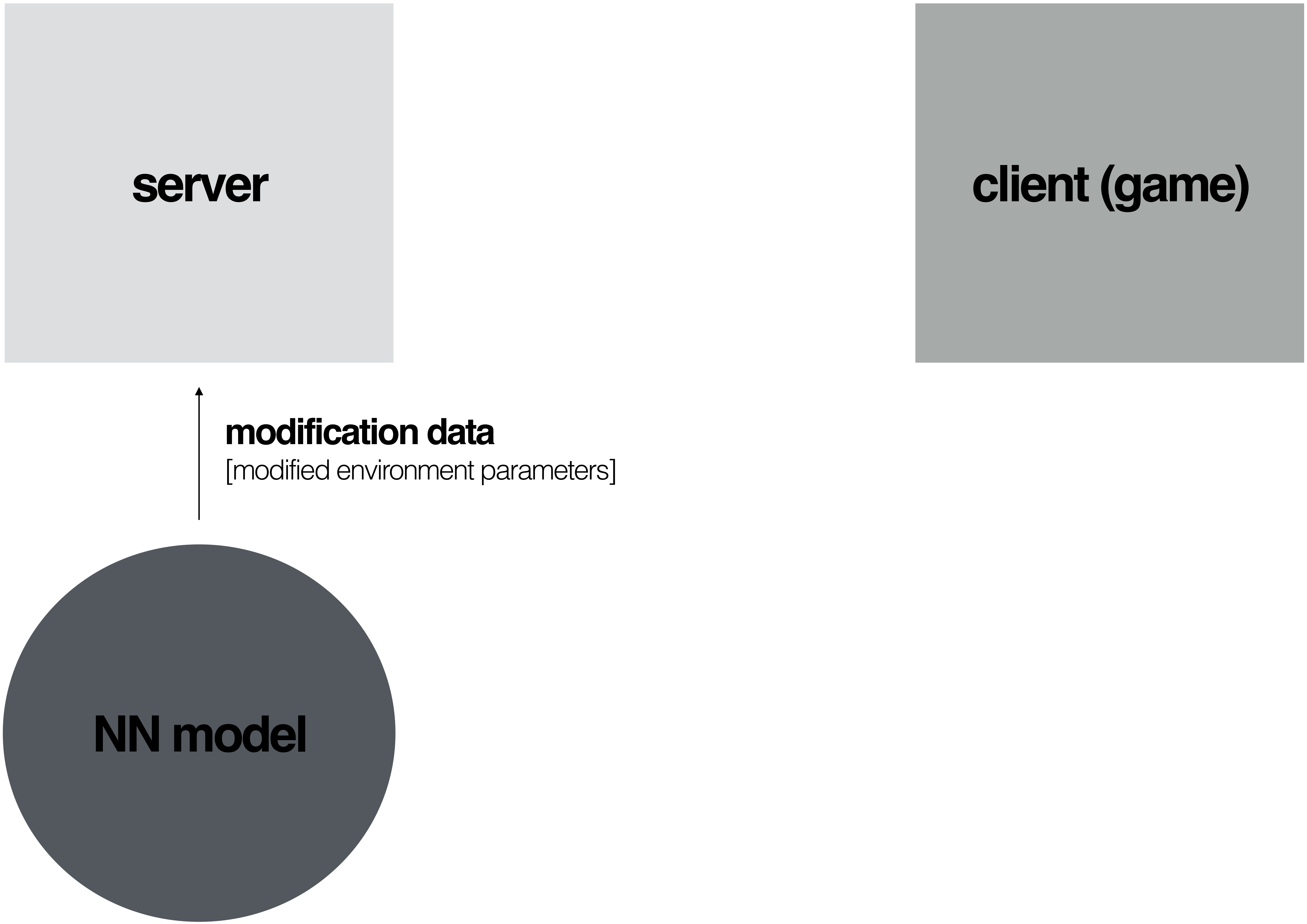
server

client (game)

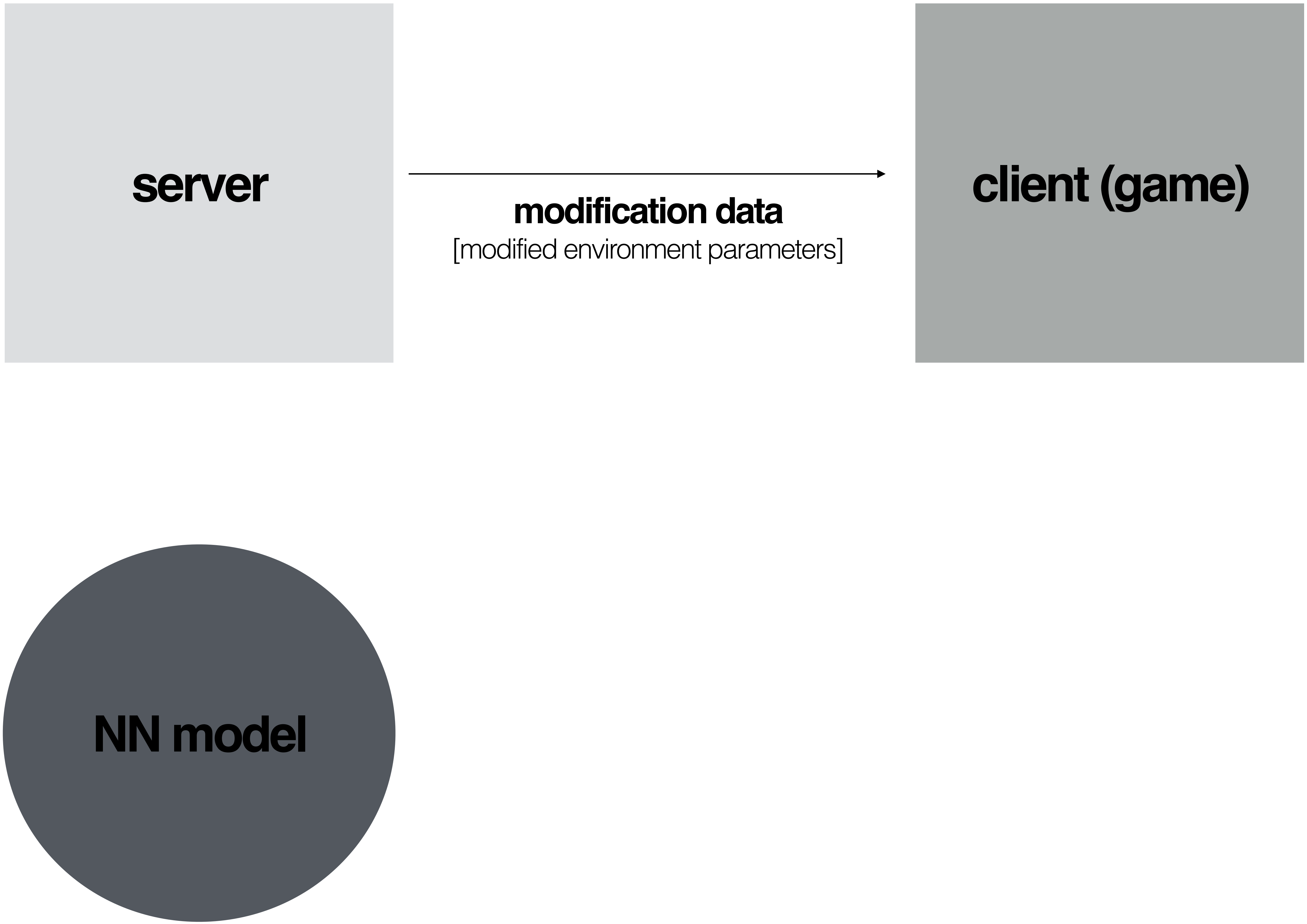
NN model

calculate modified environment parameters based on performance delta

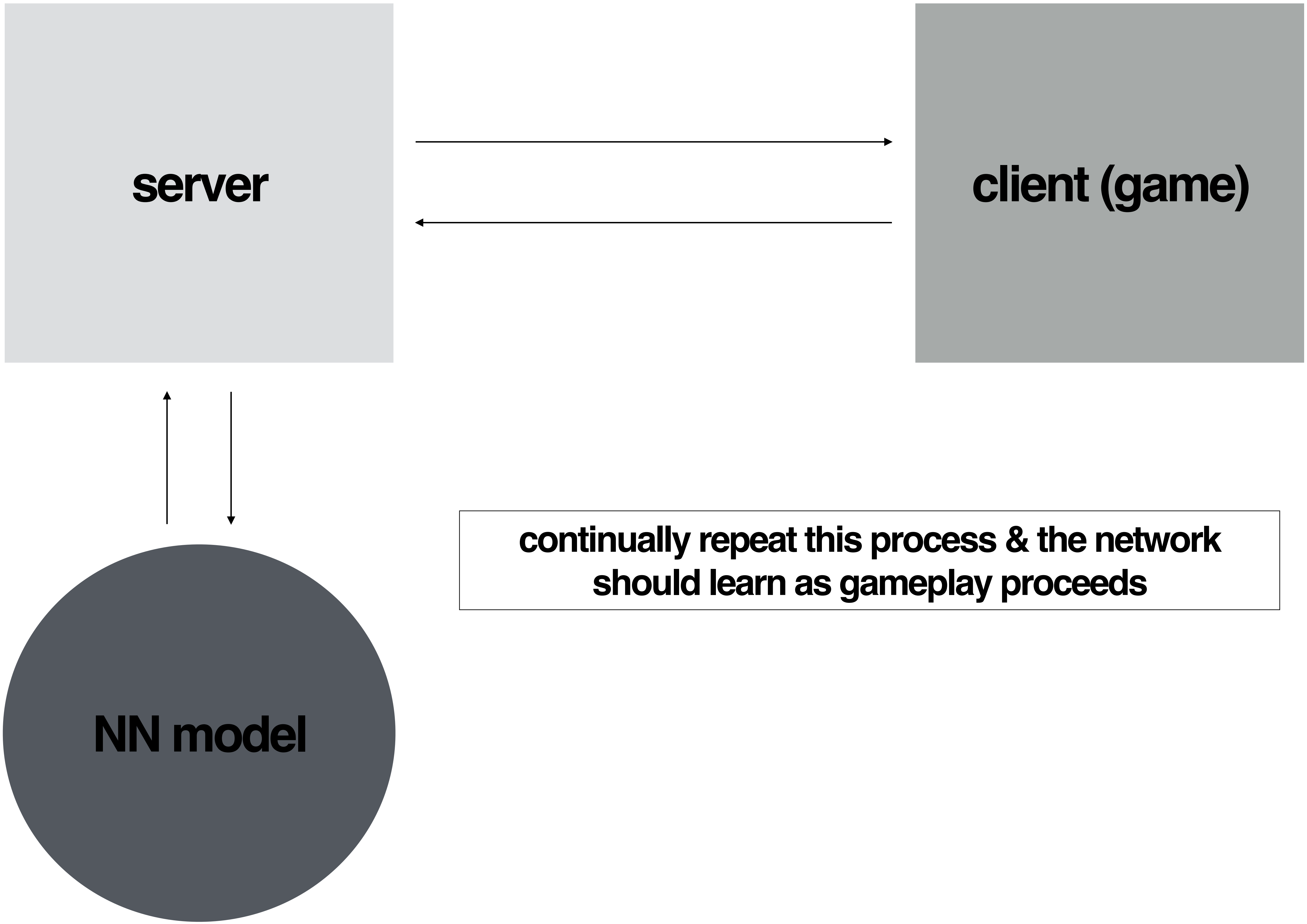
avoidance: learning loop & architecture



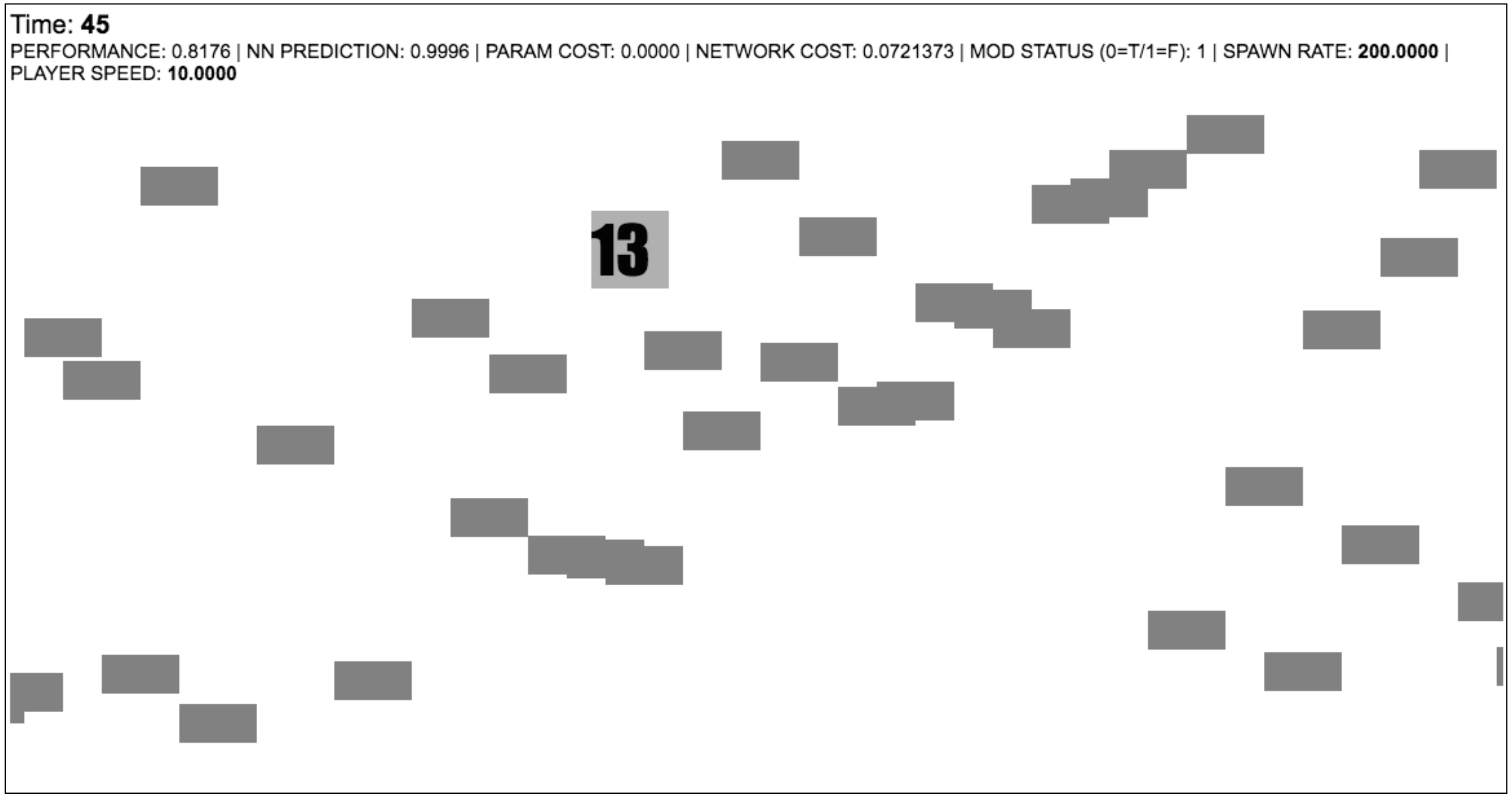
avoidance: learning loop & architecture



avoidance: learning loop & architecture



avoidance: game environment specifics



performance measure

(# of collisions / time lapsed since last measurement)

environment parameters to modify

- enemy spawn rate
- player speed

example data sample

[spawnRate = 300.0, playerSpeed = 20.0, performance = 0.60]

we’ve outlined how our NN will operate in our test environment, so how does it actually perform?

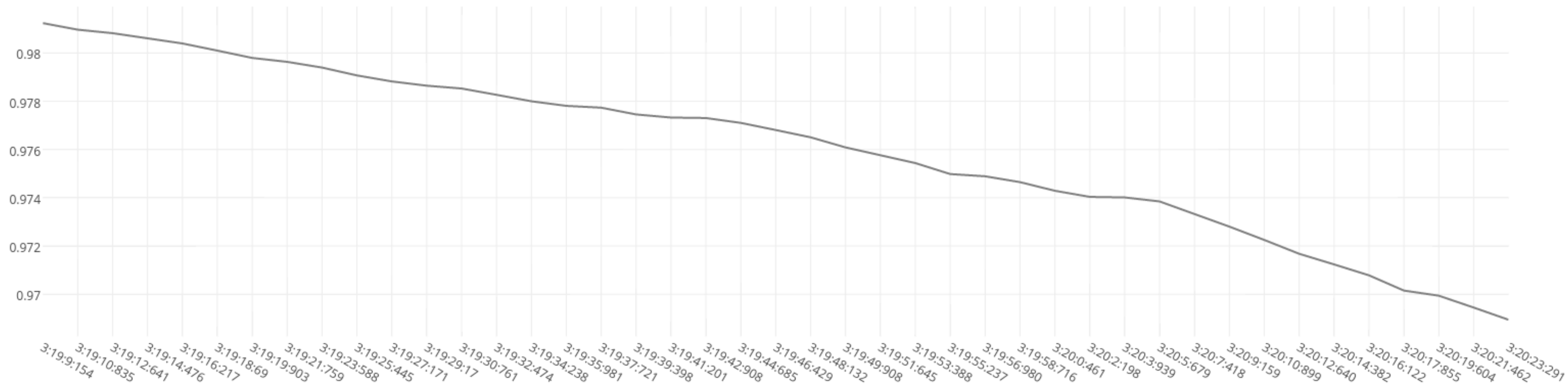
we’ll start by just testing the NN’s prediction capabilities

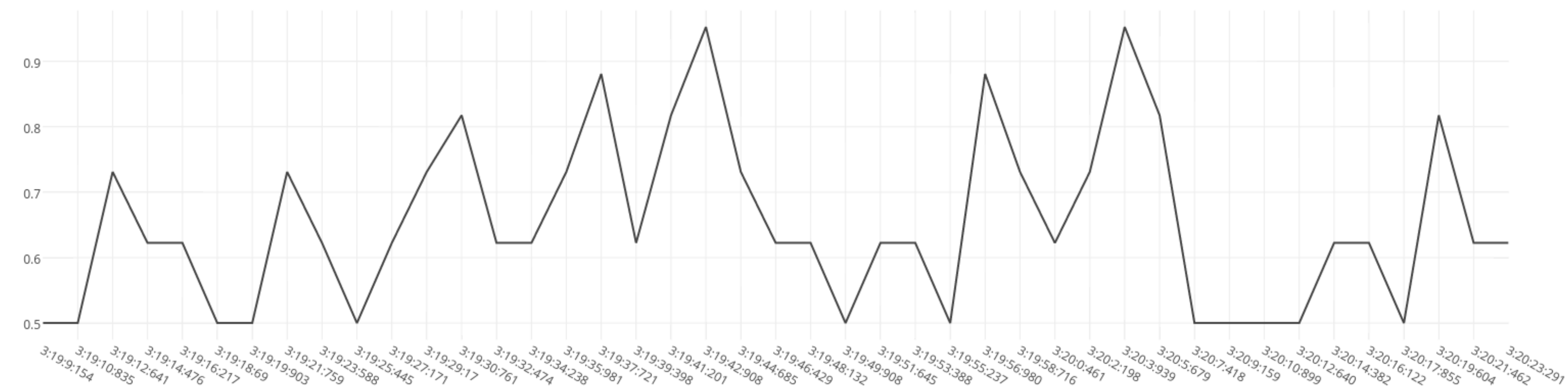
can it accurately predict a user's performance, given the environment parameters?

TestResults

initial trial

prediction v.s. performance

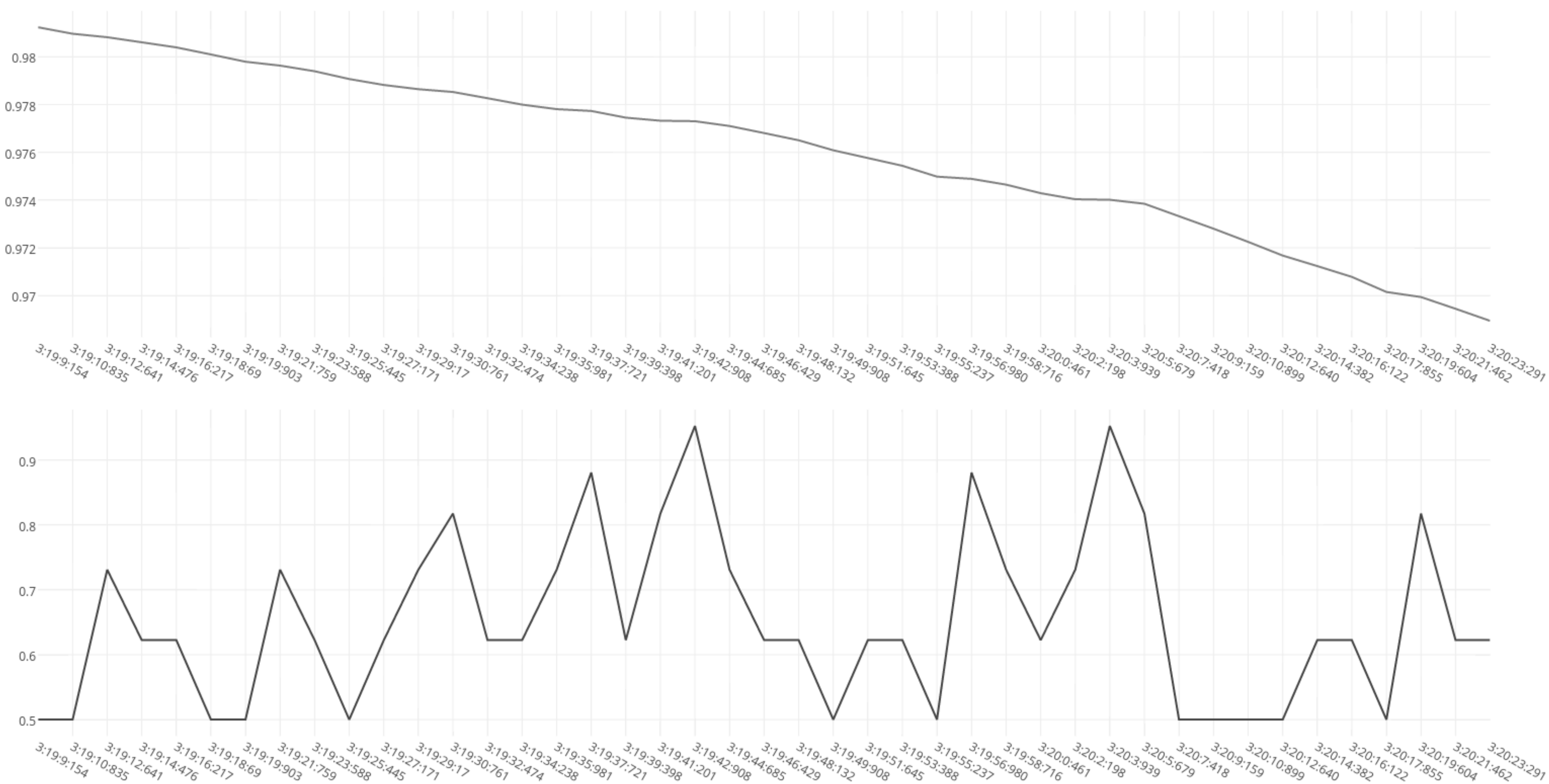




wow…that really sucks.

initial trial

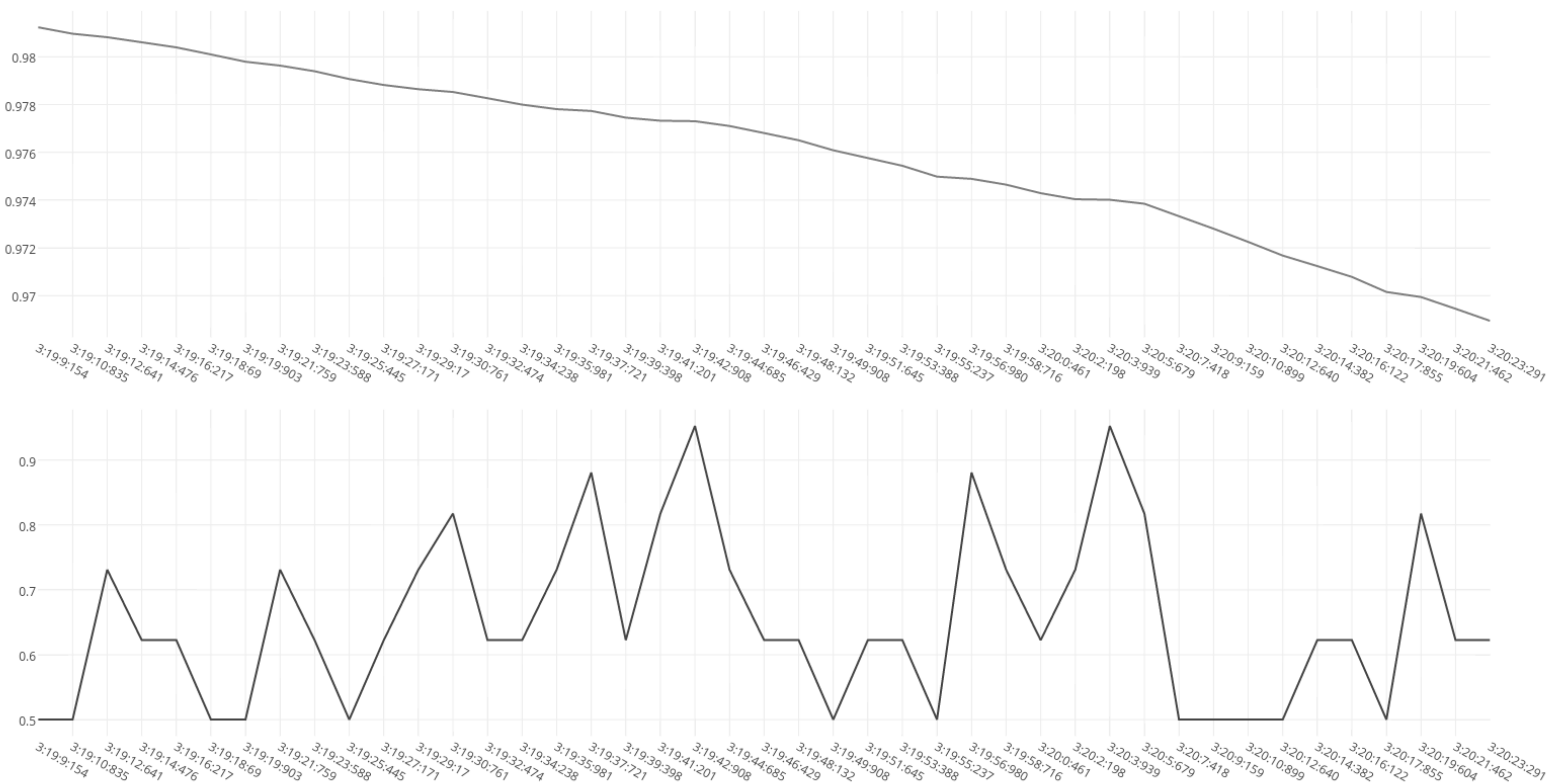
prediction v.s. performance

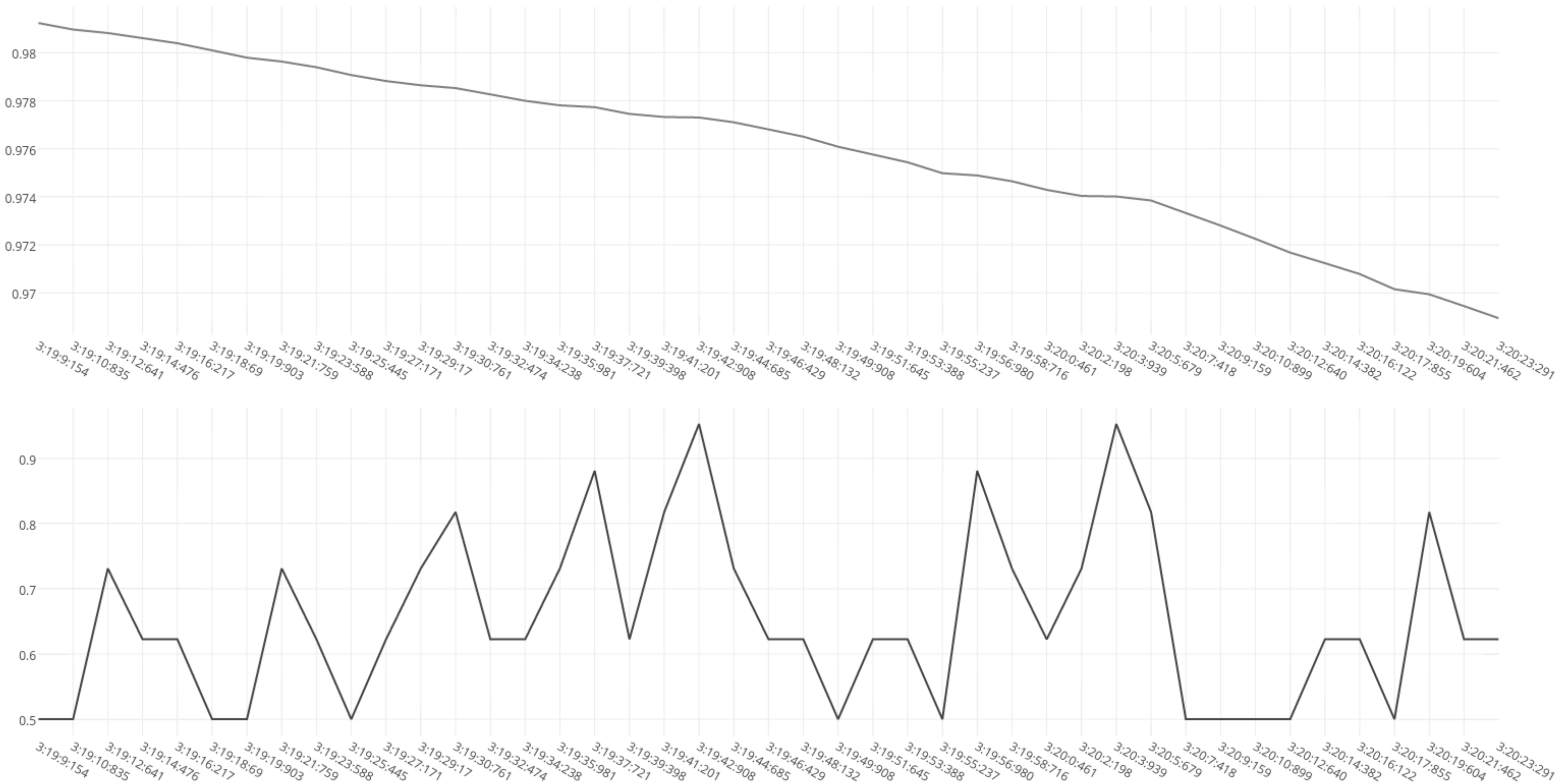


what went wrong?

initial trial

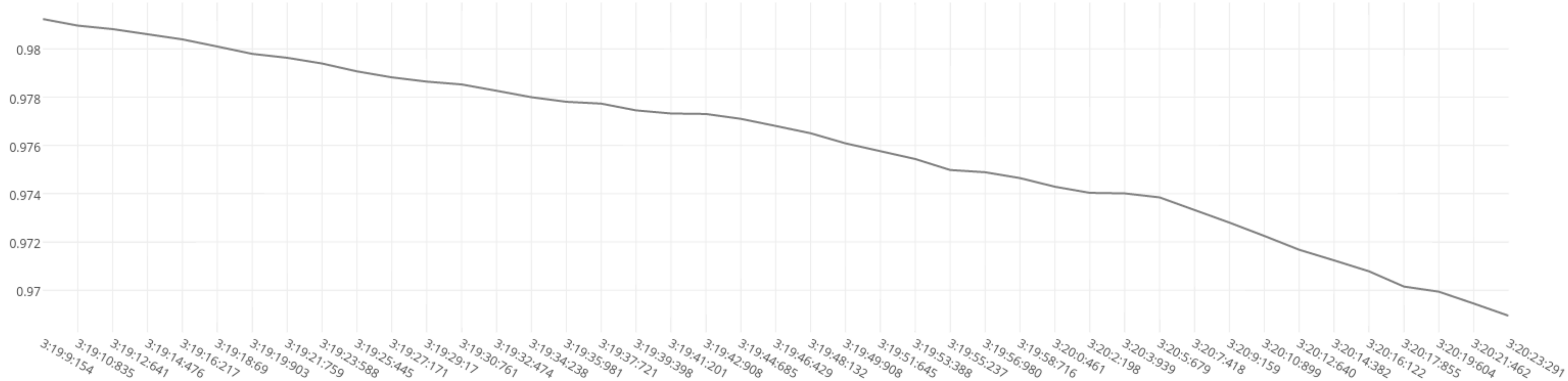
prediction v.s. performance

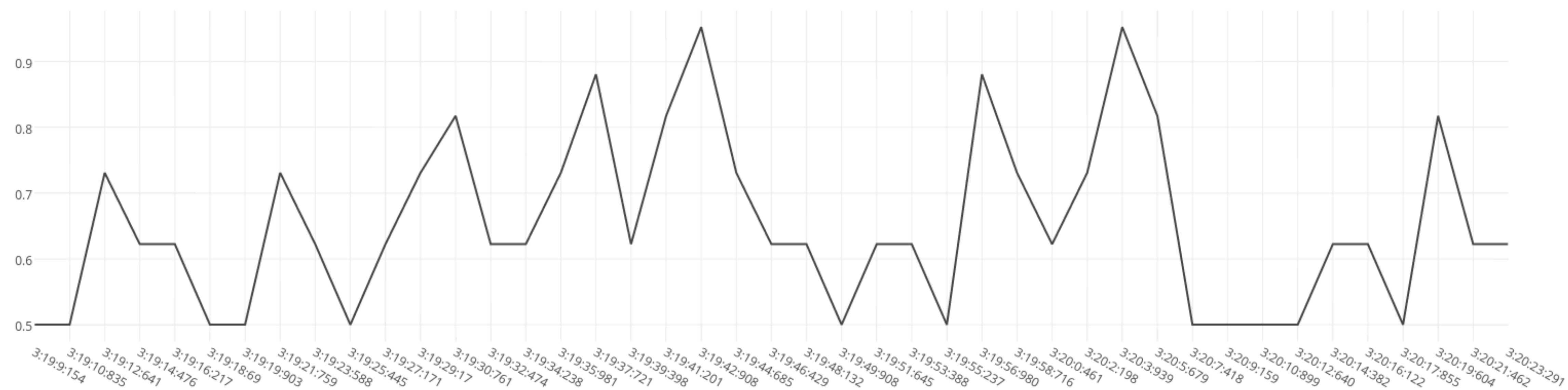




the actual performance of this user fluctuates very frequently — this entire graph only represents about a minute and a half of gameplay

*** in our original game architecture, the NN model is only learning on one piece of data sent from the server at each pass — we should augment that data with artificially created batches that will reflect the probable user performance***





the actual performance of this user fluctuates very frequently — this entire graph only represents about a minute and a half of gameplay

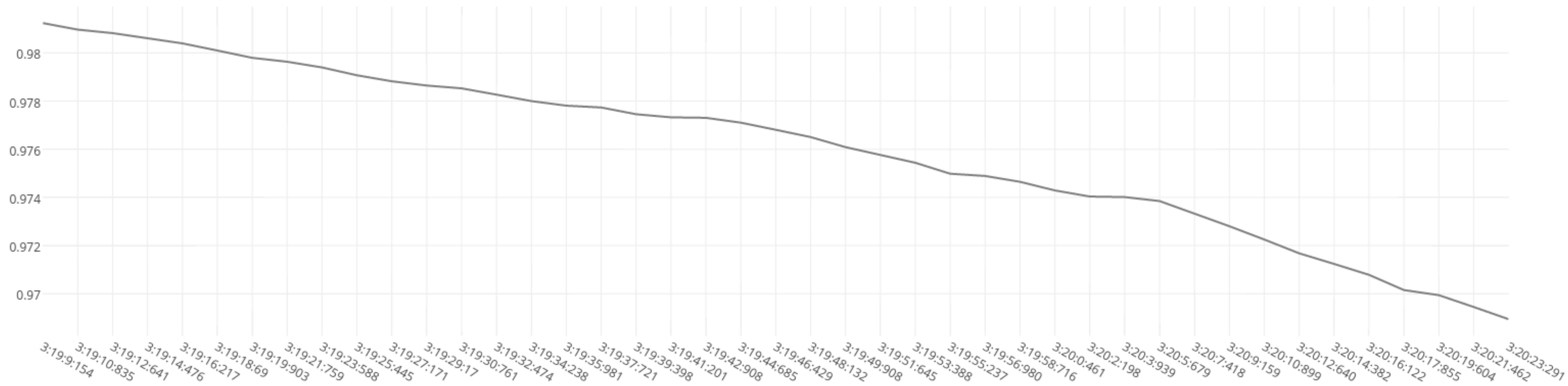
Augmenting Data Samples

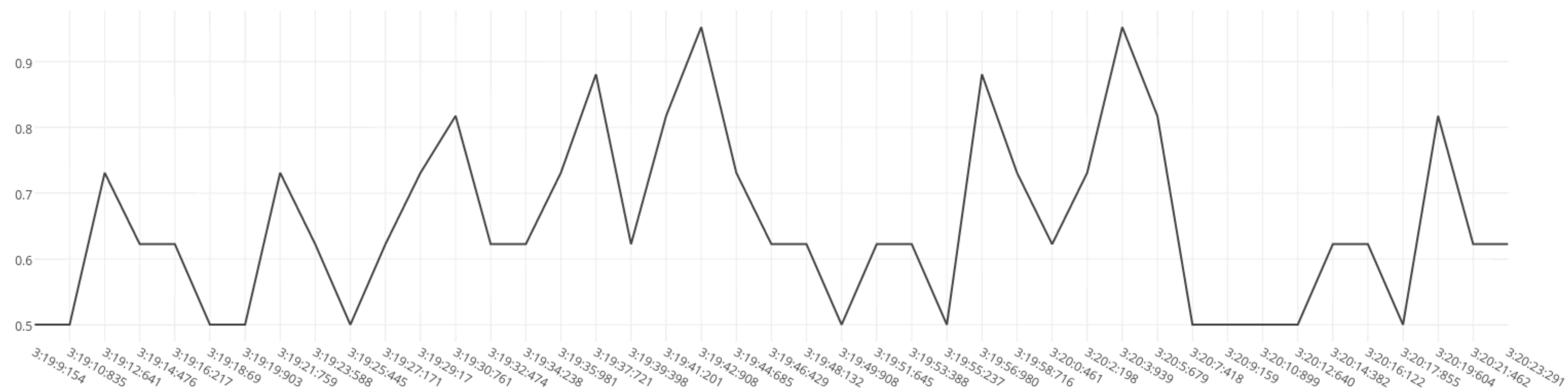
```
# where params[] is an array that contains
#     the current environment parameter settings

batch_size = 1000

for x in range(0, batch_size):
    for y in range(0, num_params):
        train_data[x][y] = random.uniform(params[y]-5, params[y]-5)
```

this code snippet shows how additional training data was artificially created before passing through the network

the assumption here is that if a given user's performance was X when the environment parameters were set to Y, then the user will probably perform relatively similarly for Y-5 and Y+5

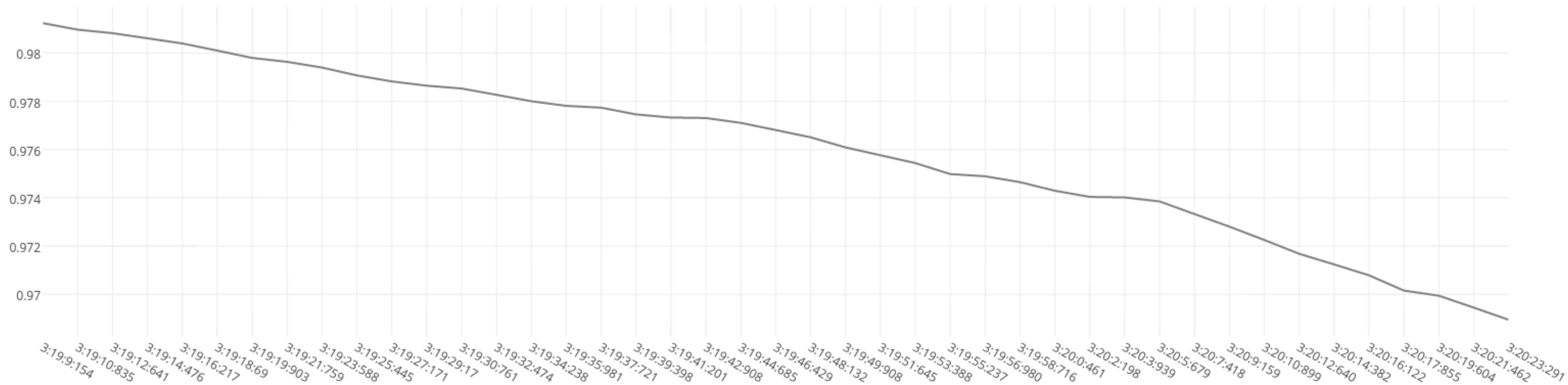
this not only gives the NN more data to learn on, which will allow it to minimize error more quickly (because there will be more weight updates per pass) but will also keep the NN from overfitting (as opposed to if we just created a batch with 1000 identical data points) *the NN needs to adapt more quickly as the user's performance ebbs and flows*

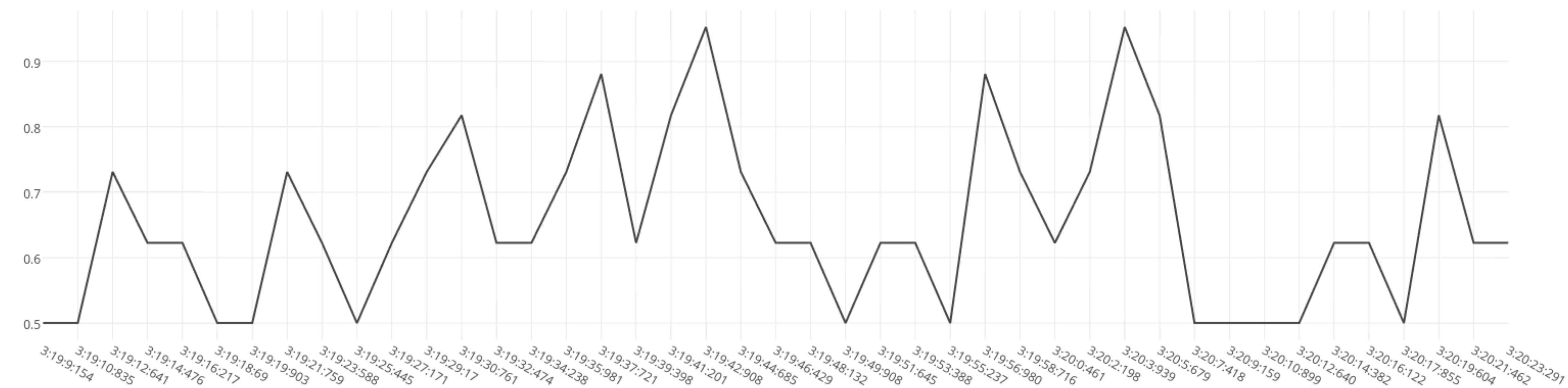




<u>*we need to adjust the learning rate so the NN can learn & adapt more quickly*</u>

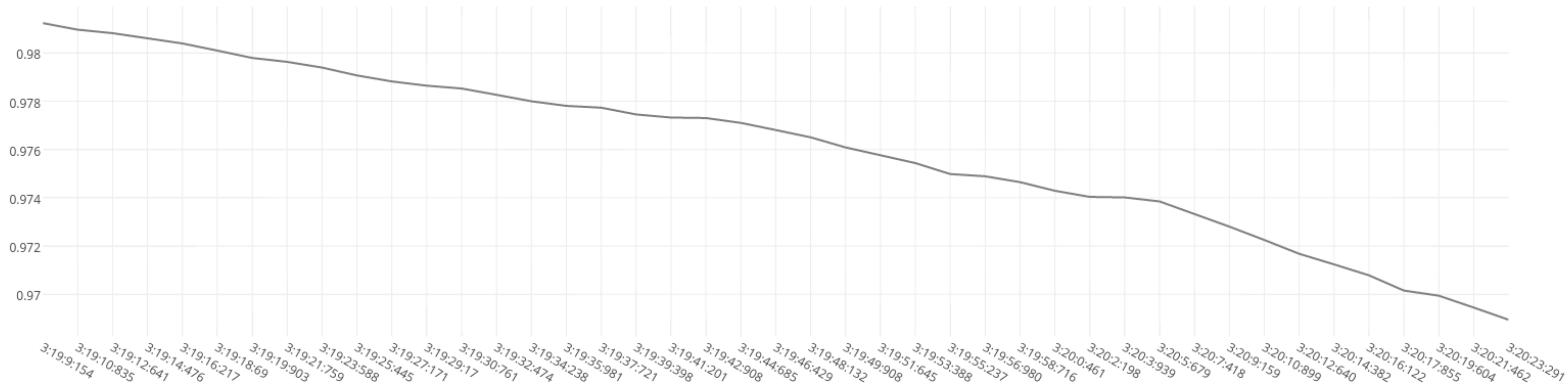
the NN needs to adapt more quickly as the user's performance ebbs and flows

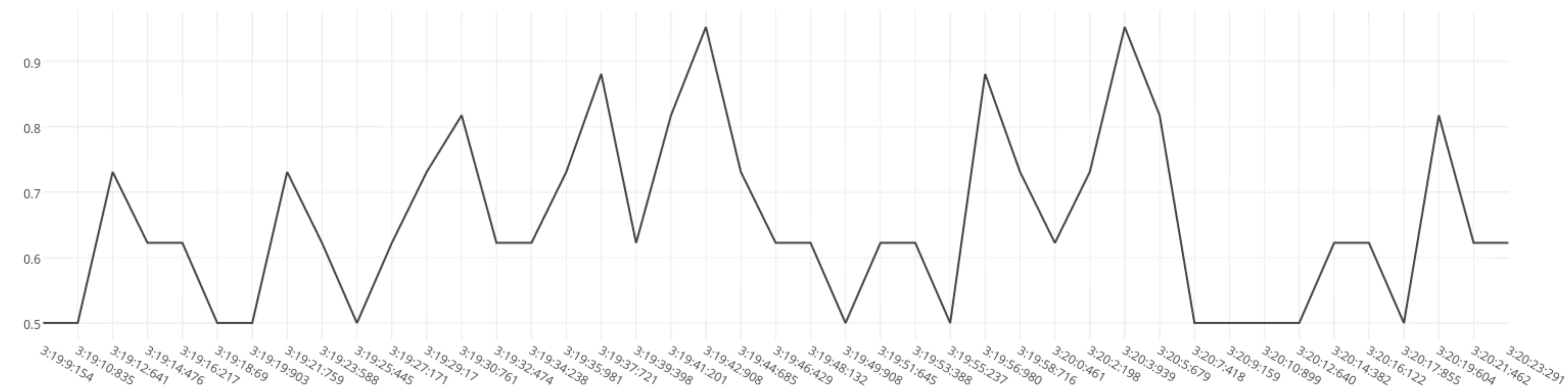




we need to adjust the learning rate so the NN can learn & adapt more quickly

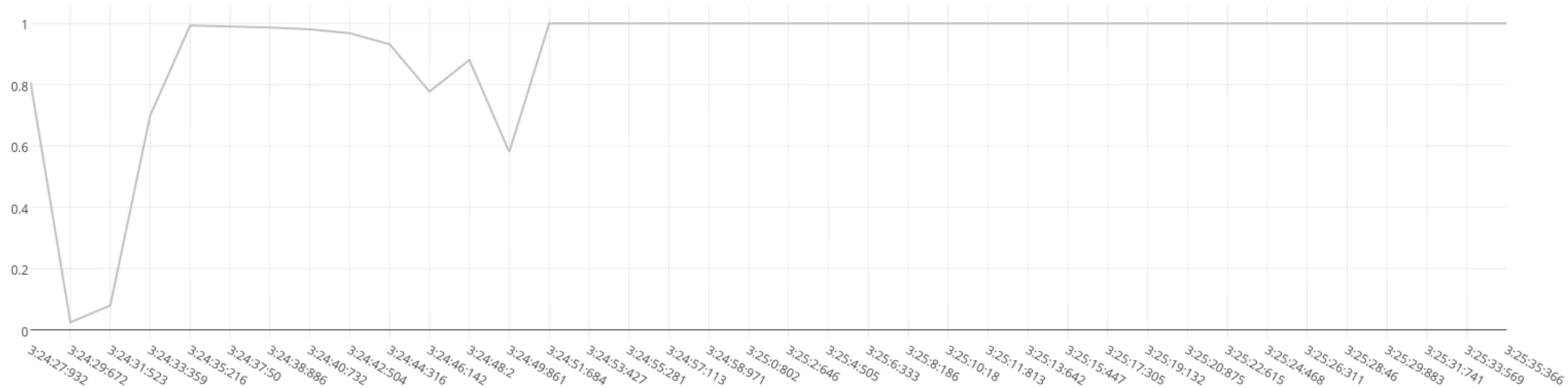
learning rate: 0.15

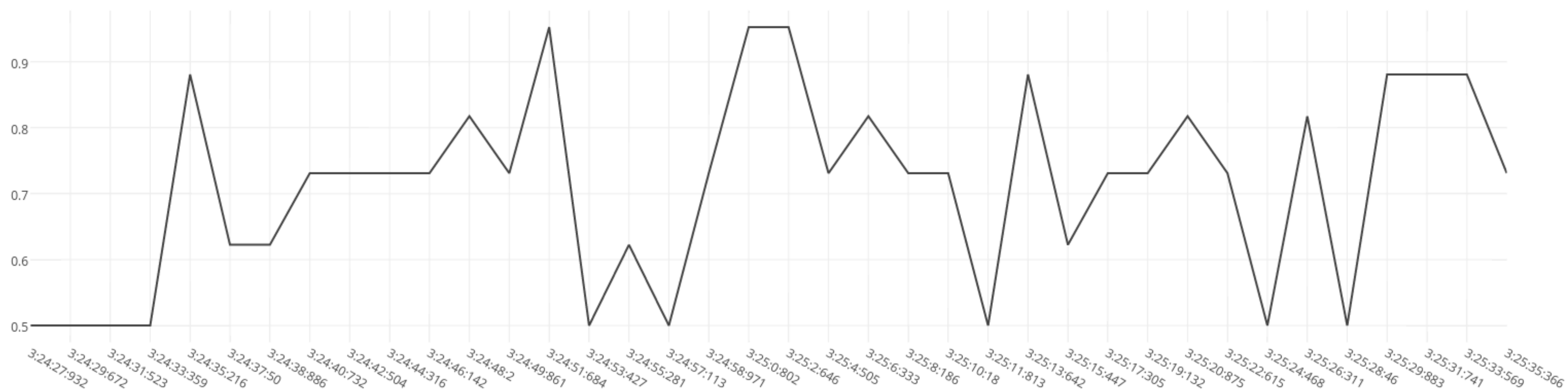




we need to adjust the learning rate so the NN can learn & adapt more quickly

learning rate: 9.5

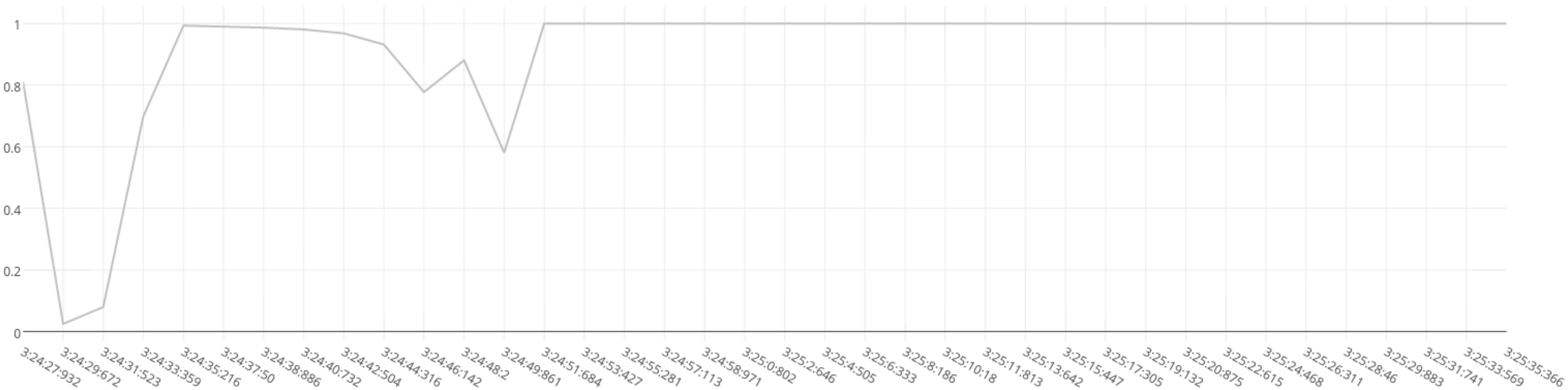


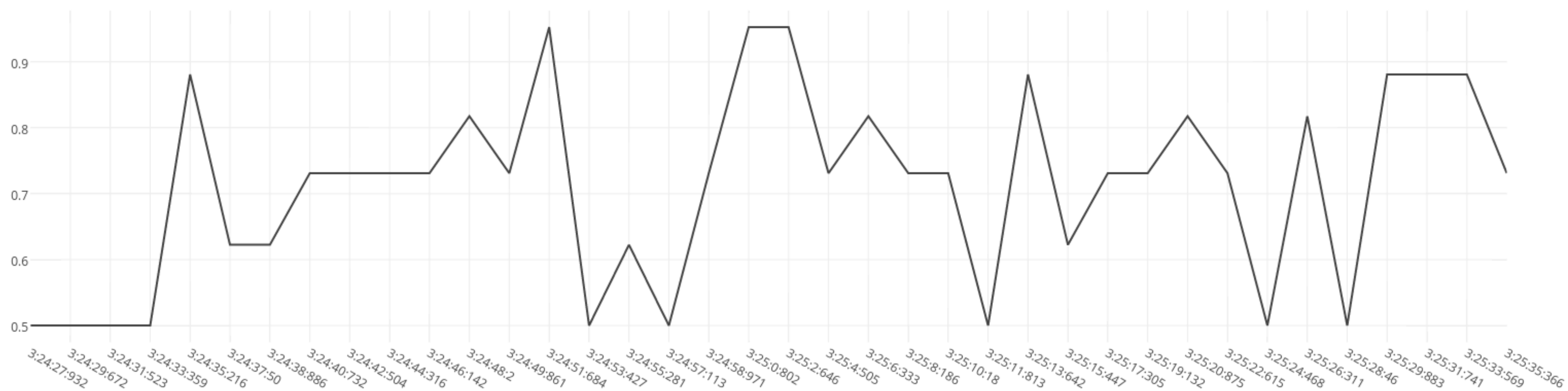


we need to adjust the learning rate so the NN can learn & adapt more quickly

that's a bit better, but we can see when the prediction flatlines because it's consistently putting out values extremely close to 1 — the learning rate is too high

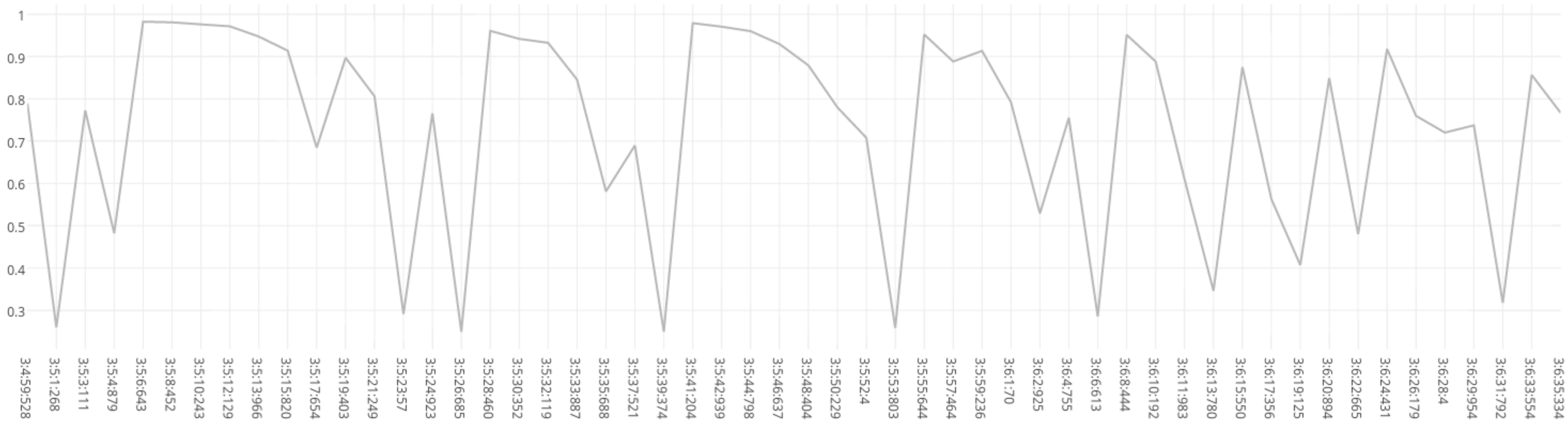
learning rate: 9.5

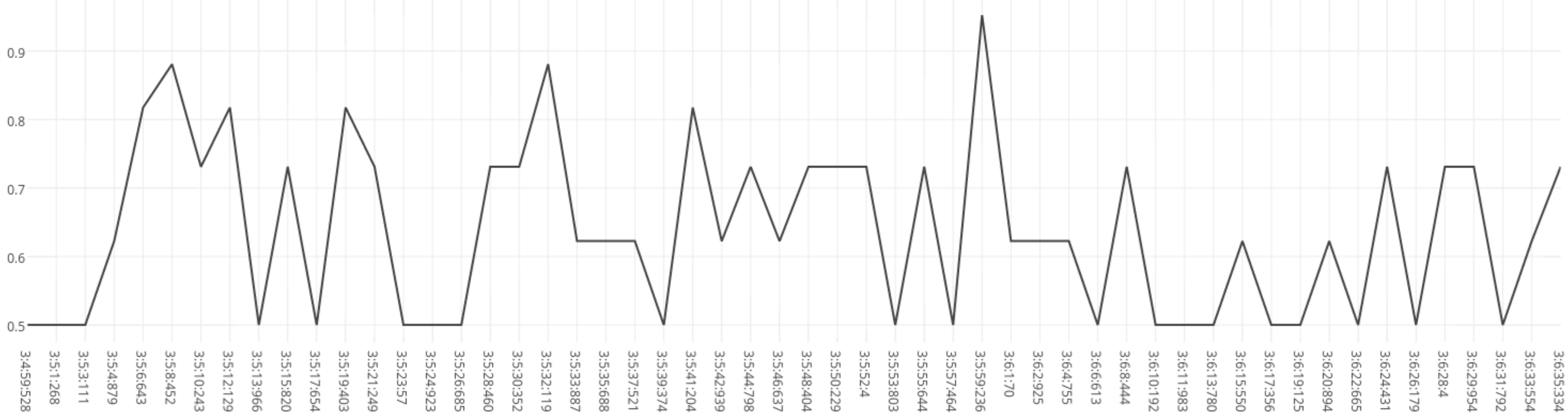




we need to adjust the learning rate so the NN can learn & adapt more quickly

learning rate: 4.5

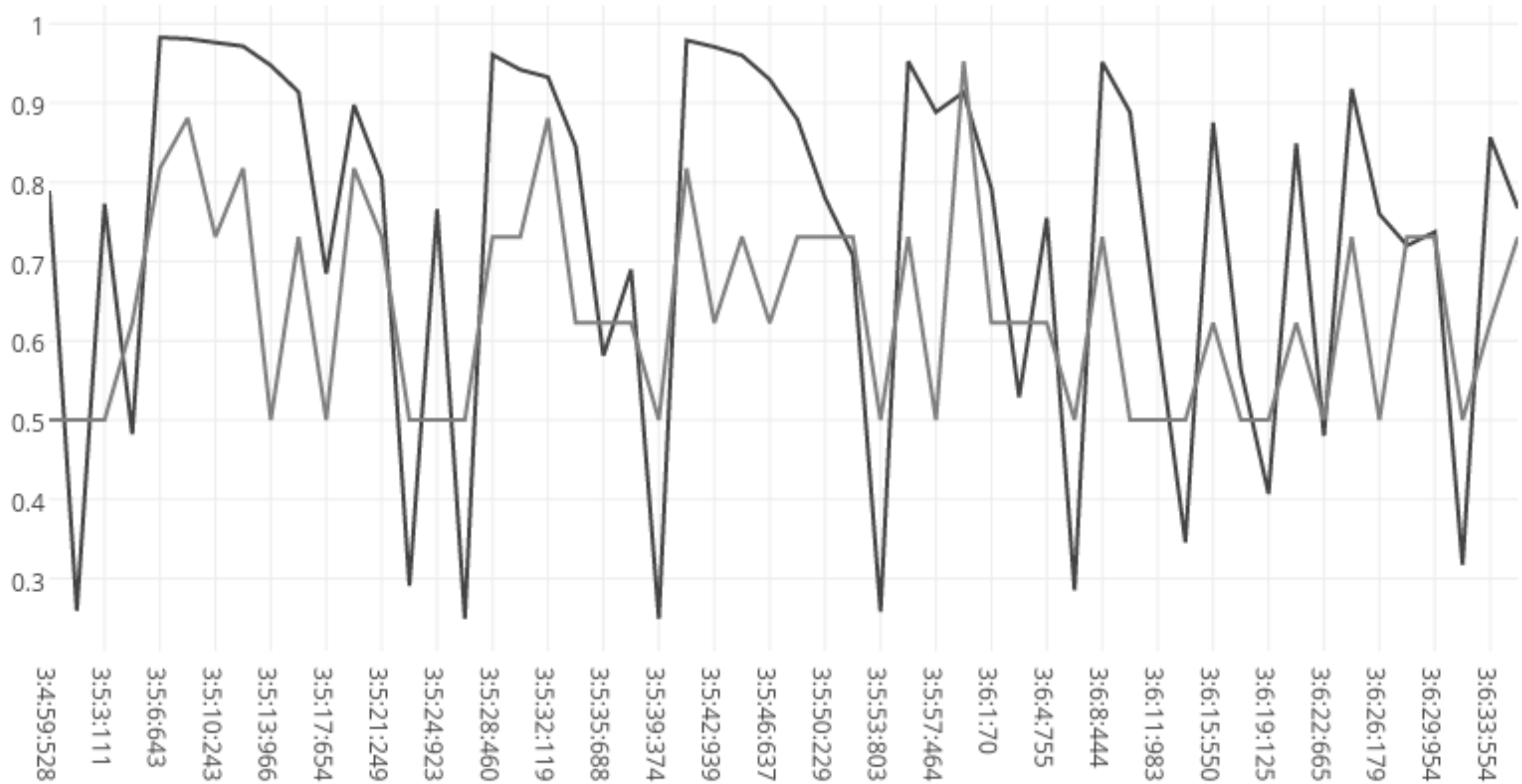




these look a lot more similar! let’s compare them more closely…

we need to adjust the learning rate so the NN can learn & adapt more quickly

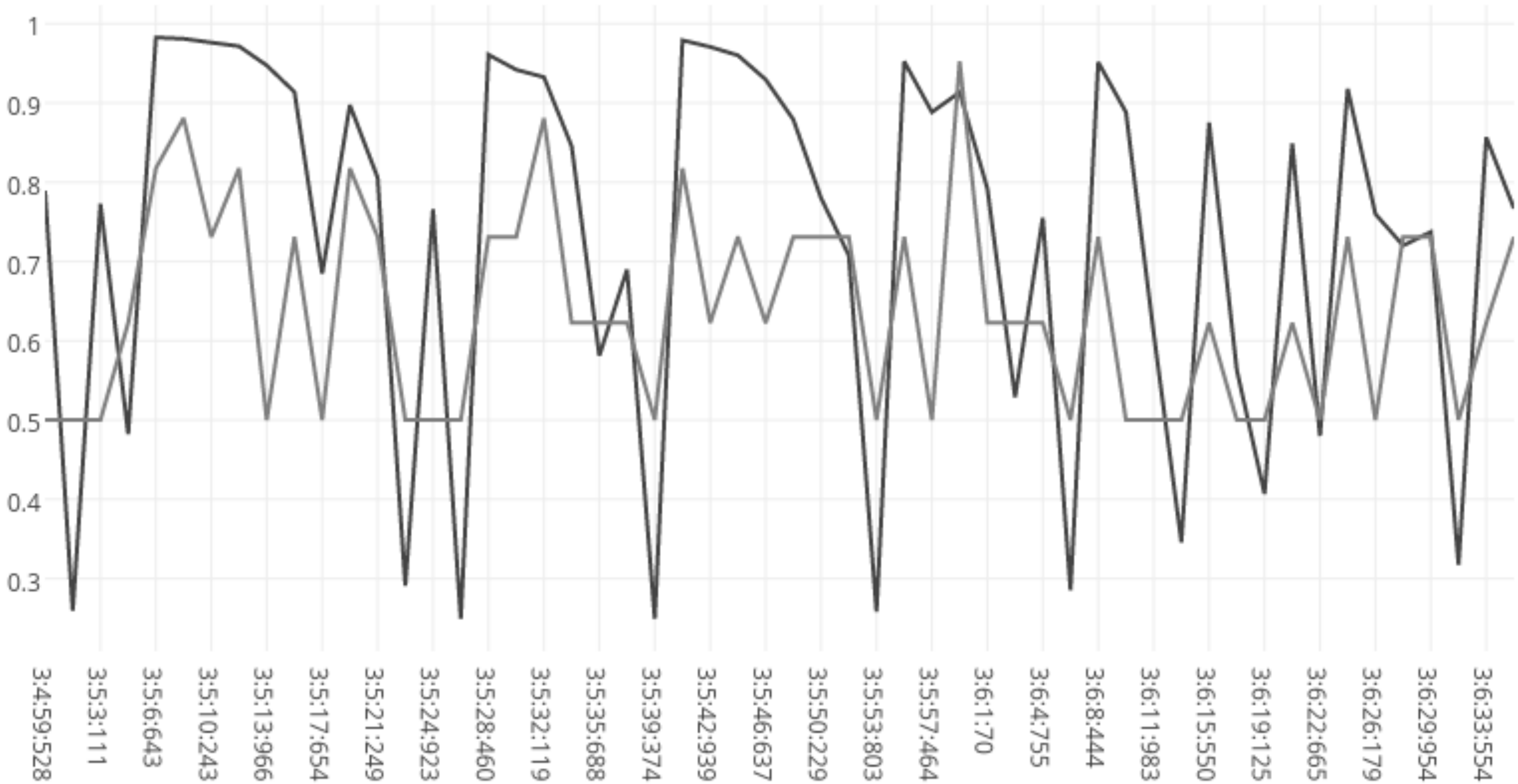
learning rate: 4.5



these look a lot more similar! let's compare them more closely…

we need to adjust the learning rate so the NN can learn & adapt more quickly

learning rate: 4.5



great! now, let's see how well the network can dynamically modify the environment!

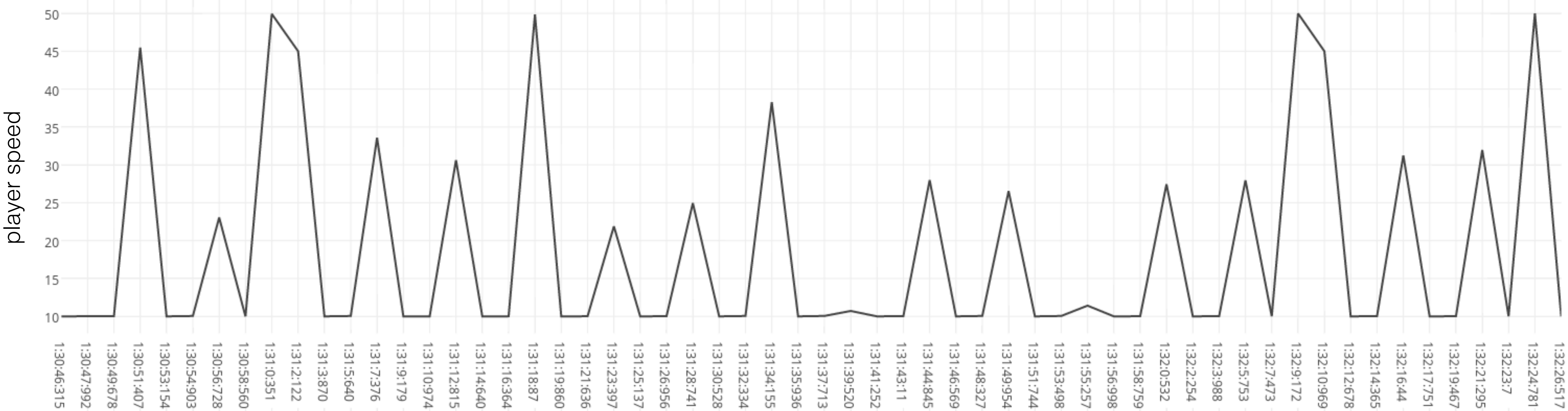
```
network_cost = self.layers[-1].network_cost(self)
layer_gradients = T.grad(network_cost, self.params)
network_updates = [(param, param-learning_rate1*grad) for param, grad in zip(self.params, layer_gradients)]

performance_delta= self.layers[-1].input_cost(self, [50.0])
performance_gradients = T.grad(input_cost, self.x)

environment_updates = [(train_x, (train_x+learning_rate2*input_gradients))]

train = theano.function([i],
        [network_cost],
        updates=network_updates,
        givens={self.x: train_x,
                self.y: train_y,
                self.goal: perf_goal},
        on_unused_input='ignore')

modify_environment = theano.function([i],
        [performance_delta, performance_gradients],
        updates=environment_updates,
        givens={self.x: mod_x,
                self.y: mod_y,
                self.goal: perf_goal},
        on_unused_input='ignore')
```

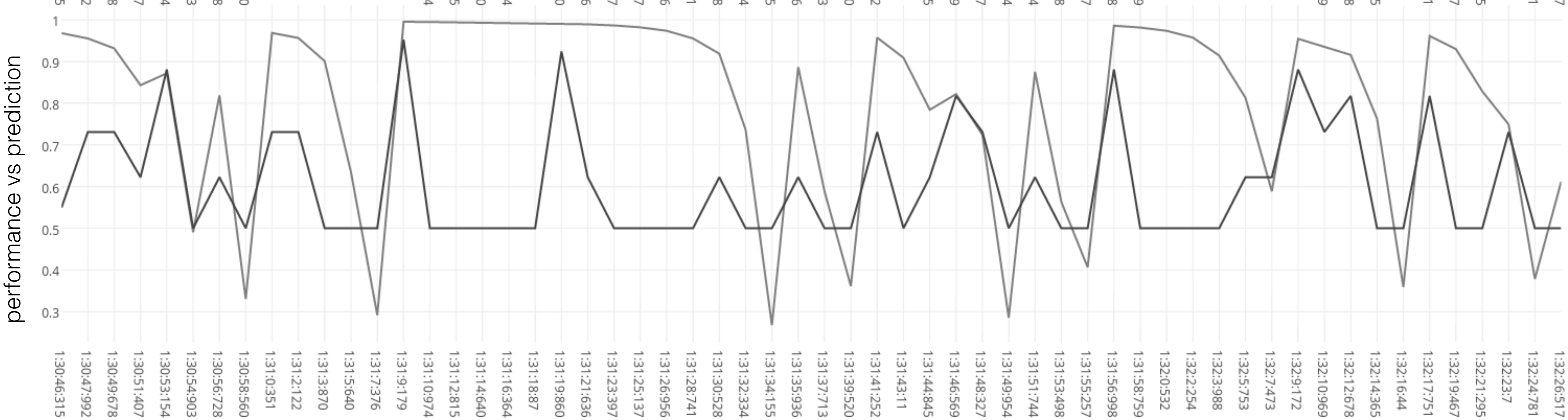
Two Different Learning Functions

These code snippets show how the gradients for both the weight values and the environment parameters are calculated. There are also two separate functions — one that performs the backpropagation for the network weights, and the other that uses just the single provided data sample to calculate how to update the environment parameters.

These graphs show approximately a minute and a half of gameplay, and how how the environment, performance and prediction changed over time. The graph on the bottom shows how the user's performance never dips below the performance goal, but the prediction accuracy fluctuates quite a bit. The graph on the top shows how the net updated the environment parameters over the course of gameplay.

learning rate = 4.3
performance goal = 0.5





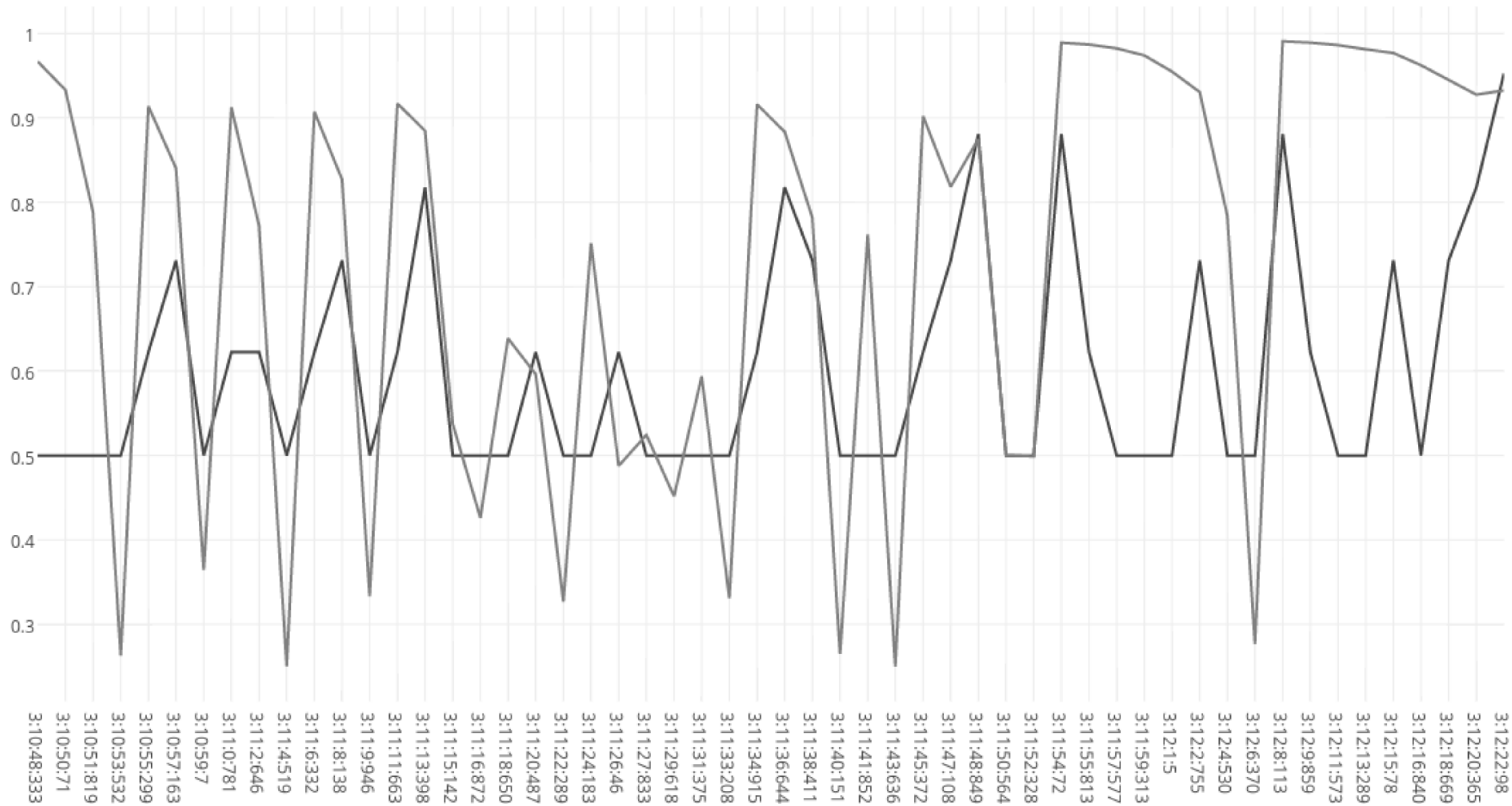
After observing gameplay, I saw that the environment parameters fluctuated wildly when the network prediction was far-off from the actual user performance. So, as a modification to the learning model, the NN now only performs the modify() function when the difference between the net's prediction and the actual performance is less than 0.0100.

More testing was performed with this updated model.

After observing gameplay, I saw that the environment parameters fluctuated wildly when the network prediction was far-off from the actual user performance. So, as a modification to the learning model, the NN now only performs the modify() function when the difference between the net's prediction and the actual performance is less than 0.0100.

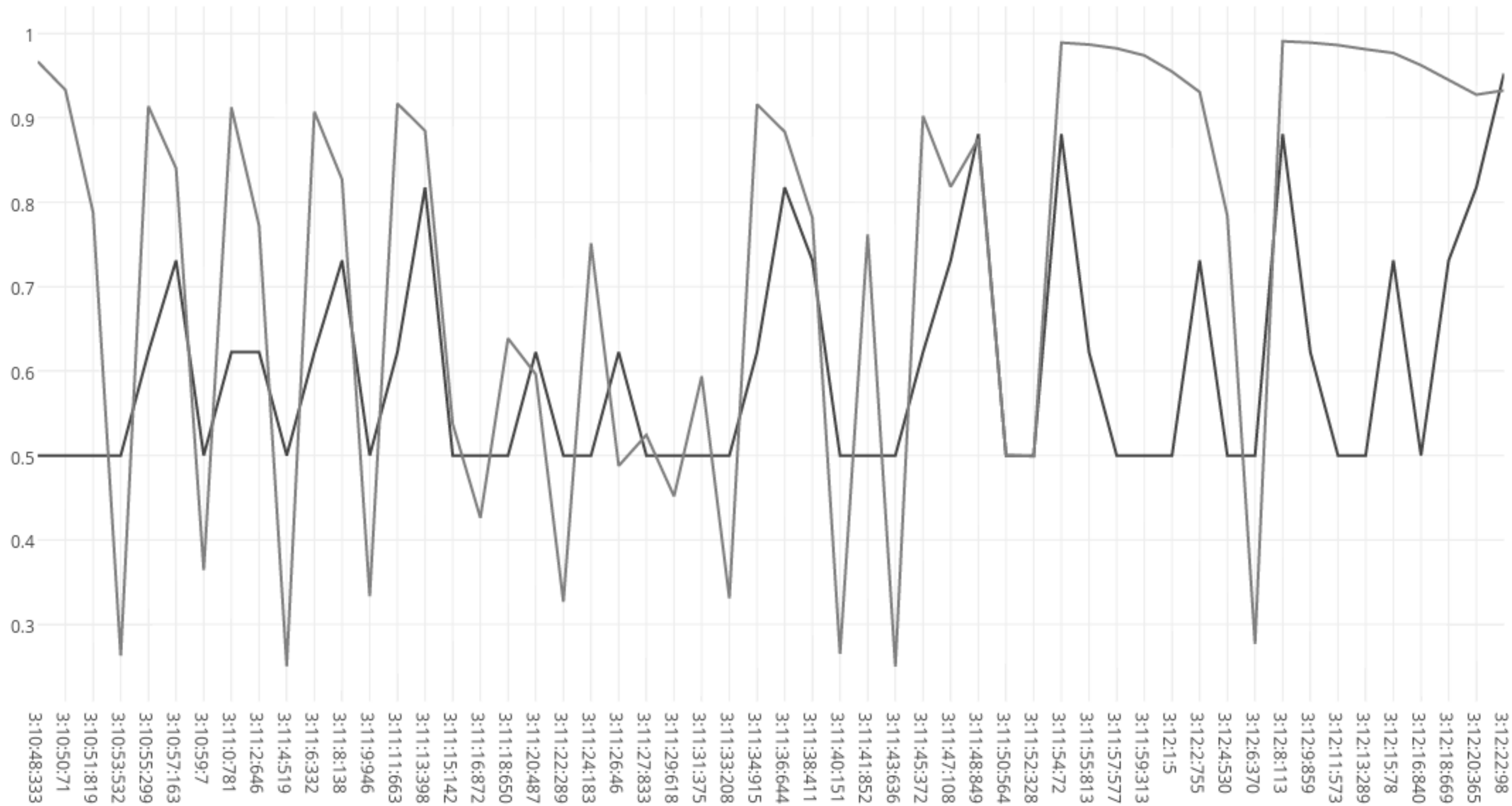
More testing was performed with this updated model.

learning rate = 4.3
performance goal = 0.5



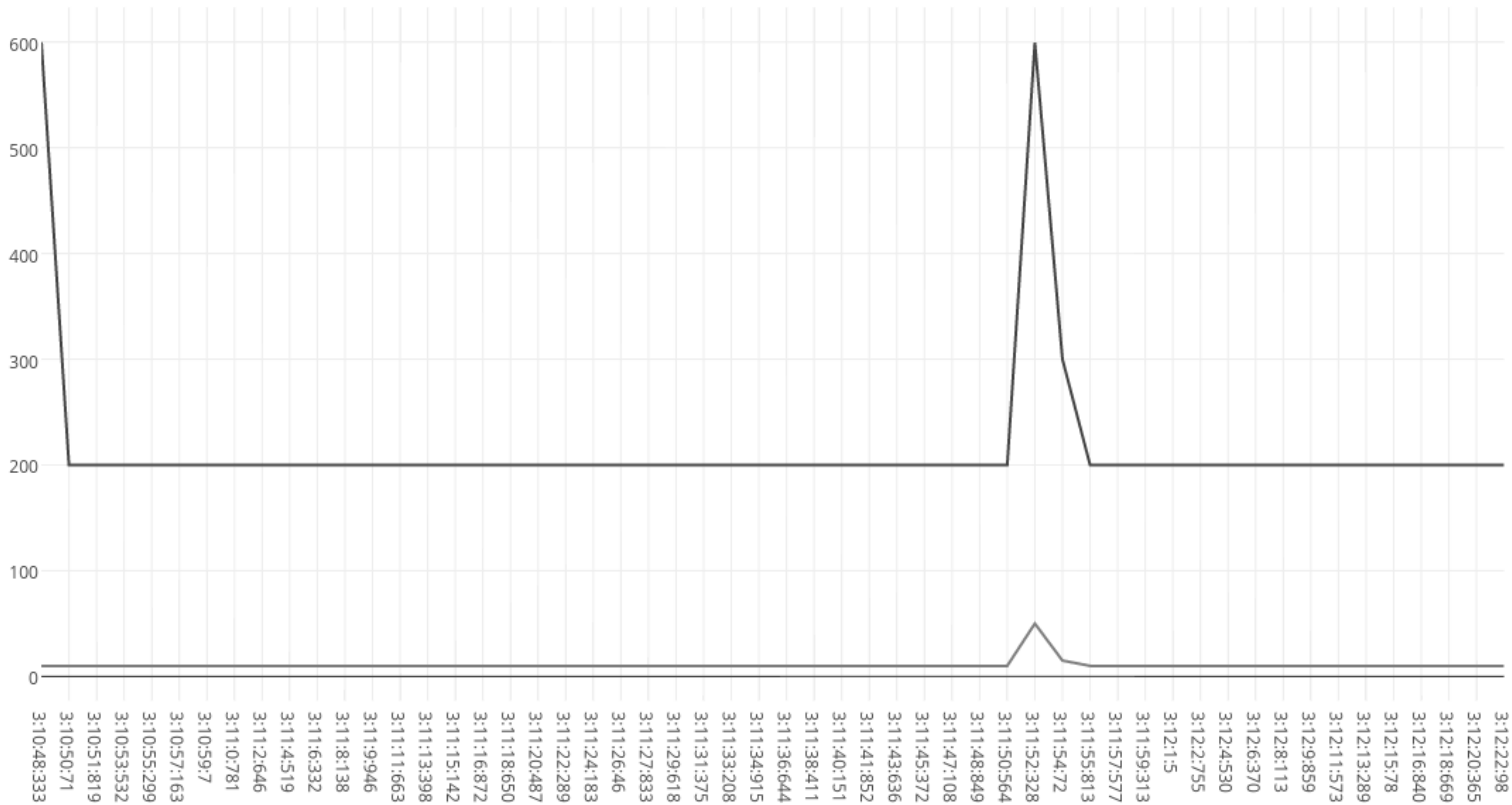
Next, we'll look at how the parameters were modified with this new learning model.

learning rate = 4.3
performance goal = 0.5



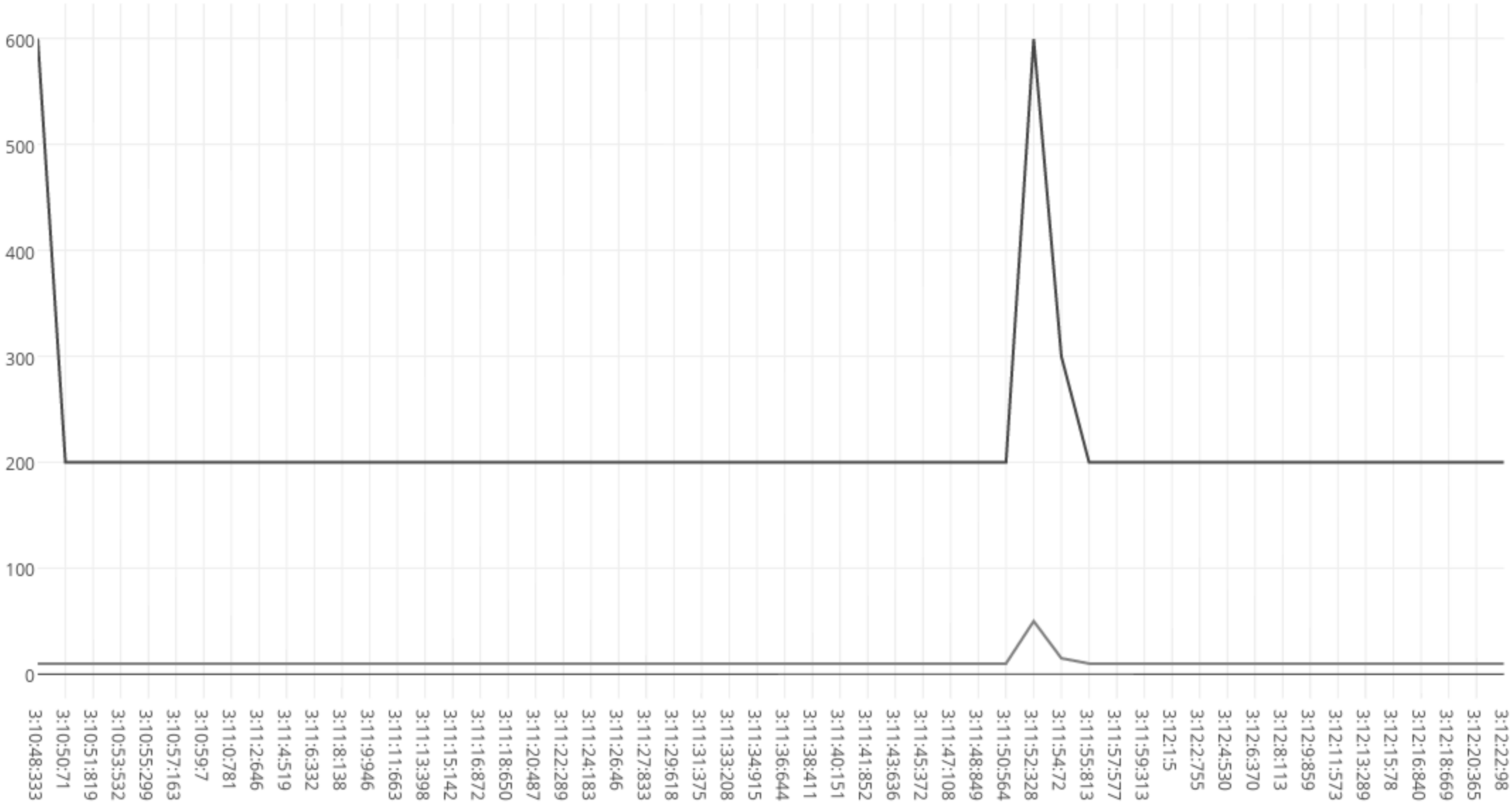
Next, we'll look at how the parameters were modified with this new learning model.

learning rate = 4.3
performance goal = 0.5



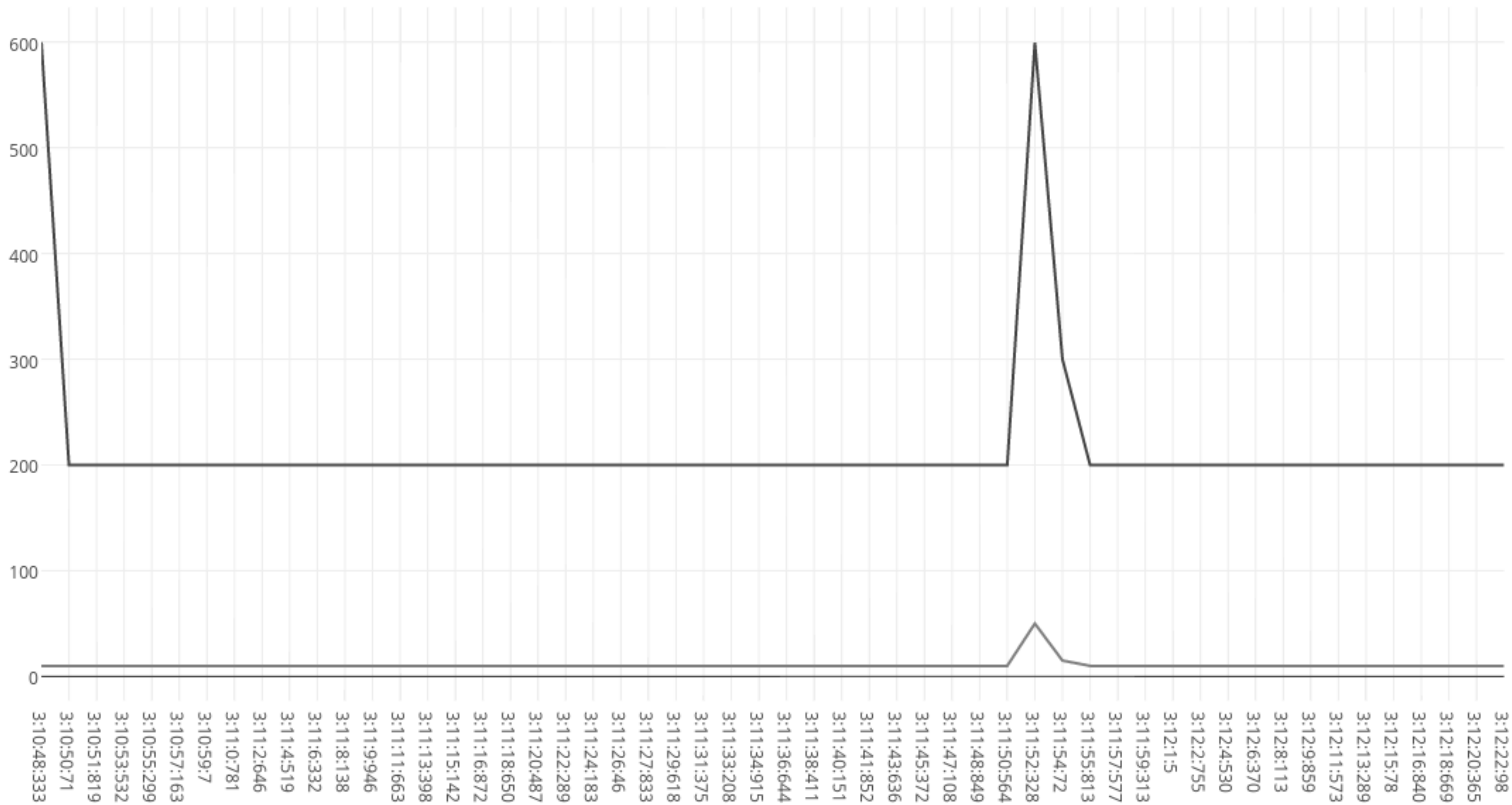
The top line shows how the enemy spawn rate changed over the course of gameplay. The bottom line shows the player speed changes.

learning rate = 4.3
performance goal = 0.5



It's obvious from this graph that the modifications were only occurring at specific moments of gameplay, so couldn't account for the user performance being maintained through the course of gameplay. This mean that the performance was changing as a result of the performance modifications, but the net wasn't always keeping the user at the performance goal.

learning rate = 4.3
performance goal = 0.5



so, some aspects of the NN worked very well and others fell short…

what modifications could be made?
what conclusions can we draw?

Potential Modifications & Future Work

Performance Measurement

The performance measurement function obviously plays a big role in the efficacy of this model. Essentially, the modification ability will only be as good as the performance function. Because this game was relatively trivial, but also very fast-paced, the performance function difficult to come up with. Further research into what types of performance functions work best with this learning model might improve modification ability.

Interval Mapping & Constraints

Another difficult aspect of applying this model to a real-world user system was how to interpret the gradients when applied to the environment parameters.

Different Environments

Obviously a key aspect of future work would be testing this model using different and more complex systems. With some improvements and modifications, this learning model could be applied to all sorts of different user systems: online stores, different games, social media sites. All of these interactive environments could potentially benefit from the ability to modify the environment for a specific user in pursuit of a performance goal.

To Modify or Not To Modify?

While I tested a few different thresholds for when to perform the modification step of the learning model, more research could be done into what threshold produces the best results.

References

1. Vygotsky, L. (1978). Interaction between learning and development. Readings on the development of children, 23(3), 34-41.
2. Yerkes, R. M., & Dodson, J. D. (1908). The relation of strength of stimulus to rapidity of habit-formation. Journal of comparative neurology and psychology, 18(5), 459-482.
3. Lupien, S. J., Maheu, F., Tu, M., Fiocco, A., & Schramek, T. E. (2007). The effects of stress and stress hormones on human cognition: Implications for the field of brain and cognition. Brain and cognition, 65(3), 209-237.
4. Davis, Randall, Howard Shrobe, and Peter Szolovits. "What is a knowledge representation?." AI magazine 14.1 (1993): 17.
5. Gelfond, Michael, and Yulia Kahl. Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach. Cambridge University Press, 2014.
6. Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov 1998.
7. Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems. 2014.
8. Michael A. Nielsen, “Neural Networks and Deep Learning”, Determination Press, 2015.
9. Welch Labs. *Neural Networks Demystified [Part 4: Backpropagation]*. Retrieved from https://www.youtube.com/watch?v=GlcnxUlrtek
10. Multilayer Perceptron — DeepLearning 0.1 documentation. (n.d.). Retrieved May 10, 2017, from http://deeplearning.net/tutorial/mlp.html
11. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.
12. gameQuery - a javascript game engine with jQuery. (n.d.). Retrieved from http://gamequeryjs.com/documentation/
13. The Theano Development Team, et al. "Theano: A Python framework for fast computation of mathematical expressions." (2016).
14. Russell, Stuart, Peter Norvig, and Artificial Intelligence. "A modern approach." Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs 25 (1995): 27.

questions?