**Explicar Min Max**

Utilizado en juegos usuario vs computadora, este algoritmo consiste en elige el mejor movimiento para la computadora suponiendo que el contrincante (un usuario) escogerá su mejor movimiento, el cual perjudicará a la computadora para hacerla perder.

Para escoger la mejor opción:

Se genera un árbol de búsqueda con todos los posibles movimientos a partir de un estado dado (estado en el que se encuentra el juego). Se recorre cada nodo terminal (jugadas que se pueden hacer) aplicando una función de utilidad, la cual evalúa si es beneficiosa con respecto a la computadora o al usuario para asignar un valor numérico positivo o negativo.

Cuando se vaya regresando hacia atrás (subir hasta la raíz del árbol), calculamos el valor de los nodos superiores a partir del valor de los inferiores; por lo que en cada llamada recursiva analizamos si es el turno de la computadora ya que nos interesa MAXIMIXAR el resultado; o bien, si es el turno del usuario nos interesa MINIMAR el resultado.

Al final nos devolverá la jugada que debe realizar la máquina para maximizar sus posibilidades de ganar ante el usuario.

Todo depende la función de utilidad si es en favor de la computadora (debemos tener la situación menor) o del usuario. En cuanto más generemos el árbol hacia abajo, obtendremos una mejor decisión.

**Parámetros de un algoritmo genético y explicarlos**

Tamaño de la población de soluciones candidatas (individuos). El tamaño de la población depende de la naturaleza del problema. Una población es un grupo de individuos.

Número de generaciones. En cada generación se selecciona una proporción de la población para preparar una nueva generación (próxima generación).

Tamaño del torneo. Este tamaño indica el número de individuos que compiten en un torneo ya que se selecciona un individuo de una población mediante un método de selección de torneo. El individuo ganador del torneo está seleccionado para crossover.

Tasa de mutación. La mutación mantiene una diversidad genética de una generación a la siguiente. La tasa de mutación representa la probabilidad de mutación, lo que implica el incremento del espacio de búsqueda (Las tasas de mutación deben estar entre 0,015 - 0,020)

Establecer el elitismo. Si algunos muy buenos individuos aparecen en una generación, es conveniente considerarlos para la siguiente generación. Esto se llama "elitismo"

**Cómo funciona un algoritmo genético:**

Primero determinamos los genes y el genoma de acuerdo a la naturaleza del problema.

* **Evolución**

Construimos la primera generación de acuerdo al tamaño de la población que especificamos, donde cada individuo es una solución.

De acuerdo al número de generaciones indicado, se realiza la evolución (crear las siguientes generaciones), por lo cual el tamaño de la siguiente será el mismo que el de la primera. Primero mantenemos el mejor individuo de la generación previa si está activado el elitismo. Después sigue el proceso de crossover, el cual itera sobre el tamaño de la nueva población para crear los individuos, por lo que en cada iteración se seleccionan los padres mediante el método de selección de torneo, se realiza crossover entre los padres ganadores y se añade el hijo dentro de la nueva generación. En caso de presentarse una tasa de mutación, se muta la nueva población con base en la tasa para añadir nuevo material genético y expandir un poco el espacio de búsqueda.

* Torneo

Se crea el torneo con base en el tamaño especificado. Para cada lugar disponible en el torneo se obtiene de manera random un individuo como candidato y se añade a la población del torneo. Después se consigue el individuo más apto (se itera sobre los individuos, comparándolos con el valor de su fitness function para encontrar el mejor invividuo)

* Crossover

Para crear un nuevo hijo, se obtiene una posición random inicial y final del padre para establecer sus genes dentro del hijo. Después se establecen los genes de la madre dentro del hijo, por lo que en caso de que el hijo no tenga el gen de la madre, lo agregamos en una posición vacía que encontremos en el hijo individuo.

* Mutation

Obtenemos una segunda posición random en el individuo. Obtenemos los genes que se encuentran antes y después de la segunda posición obtenida (dos bloques de genes) para intercambiarlos de posición entre ellos.

Después de obtener las generaciones especificadas, de la última generación seleccionamos el mejor individuo y sospechamos que es nuestra solución. Volvemos a correr el algoritmo para verificar. Si son resultados muy variados, no será la solución y tendré que ajustar mis parámetros hasta obtener un valor frecuente.

**Explicar a detalle cómo se detectan las letras**

Redes neuronales. Reconocer patrones (de audio o una imagen) que el ser humano no ve a simple vista; o bien para predecir.

* Entrenamiento

Definimos el número de input (7x5) y output (lista de letras) neurons.

Construimos e inicializamos el training set. Para ello se determina los datos de muestra con base en el número de letras indexadas de letterListIndex; y para cada letra se asigna un valor correspondiente (0,5 ó -0,5) al elemento de la posición especificada con un índice dentro del conjunto de entrenamiento.

Después construimos la red neuronal de Kohonen, que toma como parámetros el número de inputNeurons y outputNeurons.

Asignamos el conjunto de entrenamiento a la red neuronal de Kohonen.

Iniciar el proceso de aprendizaje de la red neuronal con el conjunto de entrenamiento asignado.

* neuralNetwork.learn()

crea e inicializa el error y el error cuadrático medio.

Confirma que la normalización será posible

Crea una red neuronal Kohonen con base en el número de inputNeurons y outputNeurons.

Crea la matriz de recuento de ganadores y la matriz de corrección

Si el método de aprendizaje es aditivo, crea un vector de área de trabajo

Borra, aleatoriza y normaliza el vector outputWeights

Realiza un proceso iterativo en el cual:

Obtiene el vector de error y el área de trabajo

Si el error cuadrático medio es menor que el error quit threshold, la iteración se detiene.

Actualiza el error máximo cuadrado y copia los pesos de la mejor red en esta red.

Determina el número de ganadores

Si el número de ganadores es menor que el número de neuronas de salida y más pequeño que el número de conjuntos de entrenamiento, ninguna neurona gana. Entonces forzamos a un ganador y regresa a la parte inicial de la iteración (evaluar errores)

Ajustamos pesos

Actualizar las estadísticas para mostrar en la vista

Si el error máximo es menor que 0,00001; se evalua si el proceso iterativo se termina o continua.

Sino ajustamos el LearningRate al multiplicarlo por el factor de reducción.

Una vez terminado el proceso iterativo, copiamos las conexiones de la red neuronal a la mejor red neuronal y normalizamos los pesos de las conexiones.

* GetWinningNeuron

Normalizar la entrada y obtener el factor de normalización y la última entrada sintética.

Para cada output neuron:

Calcular el producto punto de las neuronas de entrada y sus pesos de conexión; y normalizar la neurona de salida.

Realizamos el mapeo bipolar.

Determinamos la neurona ganadora.

Realizamos la cuenta para el redondeo.

Finalmente proporcionamos el índice de la neurona ganadora

* Self-Organizing Map (Vector de posibles soluciones en el orden que dice el algoritmo de Kohonen). Elemento complementario para obtener la solución.

Construye el mapa vacío como un vector o caracteres

Define el patrón de entrada (asignar 0,5 o -0,5)

Obtener la neurona ganadora basada en el patrón de entrada

Asigne la letra correspondiente de los datos de muestra en el mapa en el índice de neuronas ganadoras