08 - ¡Red Neuronal Desde Cero estilo FRAMEWORK de DEEP LEARNING! - Transcript

La primera vez que utilicé un framework de deep learning, que en mi caso fue TensorFlow, me pareció increíble cómo simplificaba la complejidad de implementar el algoritmo de retro propagación a través de las diferentes capas del modelo.

Sin embargo, cuando provequé las dudas, me pareció que era prácticamente mágica la forma en la que abstraía esta complejidad.

Sin embargo, una de las dudas que siempre me surgía es que no estaba yo totalmente cierto, no entendía completamente cómo es que estas funciones que abstraían tanta complejidad funcionan.

Por lo tanto, en este vídeo vamos a mostrar cómo implementar una red normal desde cero.

Sin embargo, a diferencia del vídeo anterior, donde utilizamos una función que nos permitía entrenar una red normal de un tamaño determinado, en este caso vamos a implementar nuestras propias clases de la capa lineal y la capa reloj, donde cada una además de saber cómo implementar el forward pass para la operación que realiza, también sabrán cómo implementar el backward pass.

De tal forma que utilizando instancias de estas clases, vamos a poder crear un modelo de una red neural de una longitud de un tamaño arbitrario.

Además, para poder formar este modelo de tamaño arbitrario, lo que vamos a hacer es también programar una clase que voy a llamar sequential layers, que va a permitir recibir objetos o instancias de la clase lineal y de la clase reloj para formar un modelo de red normal con más capas mucho más complejo.

Esta clase sequential permitirá hacer el forward pass para todos los elementos del modelo y el backward pass para los mismos.

De tal forma que tendremos una gran cantidad de abstracción y un modelo sumamente elegante que funciona de forma muy similar a cómo funcionan estos frameworks de deep learning, como son pytorch o tensorflow.

La estructura de este video es para que puedan ir programando conmigo lo que yo voy haciendo, es un tipo de programamos juntos, por lo tanto es un video que se extendió un poco más lo habitual por lo que he decidido partirlo en dos partes.

En la primera parte vamos a programar la capa de linear, la capa de reloj, así como la clase sequential que va a permitir agrupar o recibir estos objetos y crear nuestro modelo.

En la segunda parte del video vamos a programar la función de costo, la función softmax y también el modelo de entrenamiento, es decir el loop de entrenamiento haciendo uso del modelo que instanciemos a partir de las clases que vamos a crear.

Es un video que me parece sumamente entretenido, quizá es el video que más me ha gustado porque me parece muy elegante la forma en la que queda este resultado y por lo tanto espero que a ustedes también les agrade.

Bueno, programemos.

En la parte derecha de la pantalla podemos ver que tenemos la libreta de yupiter que vamos a utilizar para implementar nuestro modelo de red neuronal multicapa utilizando o programando nuestras propias clases, es decir vamos a crear nuestros objetos de la clase de la capa lineal de la capa reloj, así como una clase que vamos a llamar sequential o sequential layers y lo que nos va a permitir es implementar un modelo que combine diferentes clases o diferentes tipos de capas permitiéndonos así tener nuestro modelo neuronal multicapa con un número arbitrario de capas.

De una manera similar a cómo funcionan los frameworks como son pytorch y tensorflow, sin embargo aquí lo vamos a hacer haciendo uso exclusivamente de nonpy.

Para esto vamos a empezar entonces importando las librerías que vamos a utilizar como son nonpy y matplotlib.

Matplotlib simplemente la vamos a utilizar para graficar los datos de la base de datos mnist.

Vamos a importar también la función getimages que realizamos en un video previo que nos va a permitir formatear los datos provenientes de la base de datos de mnist y convertirlos a arreglos de nonpy que van a ser fácilmente manipulados en nuestro modelo de red neuronal.

Notemos que en este caso, a diferencia de la red neuronal anterior donde simplemente dividimos los datos de entrenamiento, las imágenes para los datos de entrenamiento, validación y pruebas entre 255 para tenerlas entre un rango de 0 y 1, ahora no lo estamos dividiendo, no las estamos dividiendo entre 255.

Lo que vamos a hacer es normalizar restando la media de los datos y dividiendo entre la desviación estándar.

De esta forma vamos a tener una media de 0 y una desviación estándar de 1 en nuestros datos.

Si vemos actualmente la media y la desviación estándar pues van a ser valores que están en un rango de 0 a 255 como esperaríamos.

Así definimos la función normalize que nos va a permitir hacer la operación que ya describí.

Notemos que lo que vamos a hacer es calcular la media y la desviación estándar para pasar a la función normalize que recibe la media de los datos, la desviación estándar, así como los datos que van a ser normalizados.

Notemos que la media y la desviación estándar son para los datos de entrenamiento en el set extrane y así esta media y esta desviación estándar van a ser utilizadas para normalizar tanto los datos de entrenamiento, validación y pruebas.

La lógica de esto es que necesitamos normalizar los datos de validación y de pruebas con los datos estadísticos, con los valores de las estadísticas de los datos de entrenamiento.

De lo contrario podría ser que nuestro modelo no funcione también si quisiéramos normalizar directamente con la media o la desviación estándar de los datos de validación o de pruebas.

Ahora podemos ver que la media y la desviación estándar, los datos de entrenamiento son prácticamente 0 y 1.

Tenemos aquí la función que ya hemos visto varias veces que nos permite graficar nuestras imágenes.

Es simplemente graficar de forma aleatoria un número pertenciente a MNIS para conocerlos.

Estas son las ecuaciones que vamos a implementar, también las conocemos.

Hemos visto ya varios vídeos tanto teóricos como prácticos de cómo implementarlas y aquí tenemos la función que nos permite crear los mini batches para hacer un entrenamiento en mini batches, donde vamos a mandar una sección de datos a la vez para hacer un entrenamiento más eficiente que enviar una muestra a la vez.

También hemos estudiado esta función en vídeos previos, por lo tanto dejo la liga por si quieren más detalles acerca de esta función.

Vamos a empezar programando nuestra red neuronal que va a estar totalmente basada en Python y non-pipe.

Recordemos que la vez pasada creamos una red neuronal como estaba como está aquí donde el forward pass era simplemente tenía los datos de entrenamiento.

Teníamos esta capa que esta capa oculta que serían 200 neuronas, si no me equivoco.

Después pasábamos por una función de activación reloj y posteriormente esto salía a la última capa donde ya de aquí pasaba la función softmax y la función de pérdida y así haciamos el back propagation.

En este caso vamos a hacer el mismo procedimiento, pero lo vamos a hacer de una forma general que nos va a permitir realizar nuestros propios objetos que van a saber cómo implementar el forward pass y el backward pass.

De esta forma lo que vamos a hacer para generar nuevas capas es crear instancias del objeto de la capa que queremos crear.

En este caso vamos a crear una clase que se llame linear que va a saber cómo hacer la función lineal, es decir la multiplicación de w por las activaciones de la capa anterior más el bias, así como cómo calcular el backward pass.

De igual forma vamos a implementar una clase que se llame reloj y que nos permita aplicar la función de activación reloj a los datos de entrada que en este caso van a ser los datos de entrada provenientes de la función de activación de la capa previa.

Vamos a empezar haciendo algo muy interesante, vamos a definir una clase nueva que sea una subclase de los non-pies arrays, es decir los arreglos de non-pies y esto nos va a permitir que al generar nuestros parámetros la red normal w, recordemos que la red normal tiene los parámetros w que va a ser multiplicado por x más un bias, entonces todos los datos, todos los valores arreglos numéricos con los que trabajemos, en vez de hacerlos como un arreglo de non-pies, los vamos a hacer como esta nueva clase que va a heredarte el arreglo de non-pies, esto nos va a permitir generar atributos a estos parámetros, por ejemplo vamos a poder generar el atributo w punto grat en donde vamos a poder guardar el gradiente de la función de pérdida con respecto a dicho parámetro, para esto vamos a hacer una nueva clase que se va a llamar non-pies, vamos a poner non-pies tensor y vamos a decir que no va a hacer nada, si, simplemente van a hacer objetos que van a ser van heredar de la clase arreglo de non-pies y ya pero al hacer esto nos va a permitir agregarle atributos y en este caso va a ser el atributo grad de gradiente, esta es una funcionalidad muy útil como en estos casos que python nos permite, sin embargo es una funcionalidad que también tenemos que usar con cuidado porque nos permite agregarle de forma arbitraria prácticamente nuevos atributos a ciertas clases que tal vez no queremos modificar, por lo tanto lo recomendable en general es siempre utilizar tipos de getters y setters, no voy a entrar a detalles con esto, esto quizá sería para un vídeo de programación orientado a objetos en python pero en este caso nos va a facilitar mucho la implementación del gradiente de nuestra función de costo con respecto a los parámetros, entonces los objetos de w y b que creemos van a ser de este tipo y ahora sí vamos a implementar nuestra clase linear y esta clase va a ser muy parecida a la clase linear de python y lo que vamos a hacer es esto, vamos a definir nuestra clase y como dije brevemente la clase linear va a tener las funciones, debe de saber cómo inicializarse obviamente para esto vamos a hacer un "init", debe de saber cómo hacer su "forward pass" y debe de saber cómo calcular su "backward pass", entonces vamos a definir nuestra clase linear que va a ser "class linear", luego podemos ir poniendo los comentarios pero ahorita para hacerlo más rápido vamos a escribirla directamente, vamos primero a poner la función de inicialización de la clase, obviamente recibimos "self" que es la instancia o es el nombre con el que comúnmente se refiere a la instancia que vamos a generar de cada clase, vamos a hacer los parámetros que recibe de forma muy similar a cómo funciona por ejemplo pytorch donde cuando instanciamos un objeto de la clase linear recibimos el tamaño de elementos a la entrada, es decir si vemos por ejemplo este modelo de red normal, recibiremos cuántos elementos hay en la capa previa, la entrada se refiere a la capa previa, por ejemplo si quisiéramos instanciar los elementos para esta capa lineal para la capa de salida pues pensaríamos que los elementos de entrada son el número de neuronas de la capa anterior, si nos referimos a la capa de entrada el número de elementos de entrada van a ser el número de píxeles que entran en este caso 784, dado que estamos trabajando con MNIST y por otro lado el otro parámetro va a ser el número de elementos de salida output size que realmente se refiere al número de neuronas en la capa y lo que vamos a hacer es que entonces vamos a poner un parámetro que se llame W y este lo vamos a inicializar ahora de la forma ya sabemos vamos a llamar inicializar parámetros vamos a poner un comentario rápido aquí, init parameters, init parameters utilizando el tipo de inicialización kaming que explicamos en el vídeo anterior por aquí también dejo la liga a este vídeo donde explico los detalles y por qué esta es la mejor forma de inicializar cuando tenemos redes neuronales profundas ya que la forma de inicializar como hicimos la vez anterior con valores aleatorios pequeños no funciona bien para redes neuronales profundas entonces vamos a empezar creando nuestros parámetros y para esto lo que vamos a hacer es en pie vamos a hacer la función random run n y vamos a generar una matriz que tenga de dimensiones el tamaño output size el tamaño de neuronas por el número de elementos de la capa previa de la misma manera que hicimos en nuestra red normal anterior pero ahora utilizando kaming lo que vamos a hacer es dividir entre la raíz cuadrada de el número de elementos de la entrada entre dos y aquí viene una parte interesante porque hasta ahora esto lo estamos generando como si fuera un tipo de arreglo numérico de python pero dijimos que vamos a utilizar la clase en p tensor que nosotros acabamos de crear y dado que es una su clase de un arreglo de python va a heredar todas las propiedades que tengan los arreglos de python pero nos va a permitir generar atributos adicionales en este caso vamos a hacer una vista de este arreglo como si fuera un objeto en p y abajo tensor punto viúlo que nos permite es generar una vista nueva de este de este objeto realmente está apuntando al mismo objeto no lo está cambiando sin embargo nos permite generarlo y visualizarlo como si fuera a crear una vista como si fuera un arreglo del tipo en p tensor de hecho podemos ver la documentación y aquí como vemos una de las formas de utilizar non pi viú es donde se pasa una subclase de ende array y aquí lo que hace es regresar una instancia de la subclase de ende array de del arreglo de non pi que observo que apunta al mismo arreglo no causa reinterpretación de la memoria o sea es el mismo objeto aún entonces vamos a hacer esto y lo mismo vamos a hacer para los parámetros correspondientes al ballas solamente que los ballas es como recordamos se inicializan con cero y el tamaño va a ser del número de neuronas es decir de output size y recordemos que queremos especificar que tengamos una columna vamos a tener el número de neuronas como un parámetro de filas pero especificamos que sea un vector es decir un vector columna y también decimos que queremos crearlo como una vista de en p y en bajo tensor y con esto terminamos la inicialización de nuestro de los elementos de los parámetros de nuestra clase lineal o de la instancia de nuestra clase lineal el siguiente punto va a ser vamos a definir la función forward que va a implementar esta ecuación si recordamos aquí de la del vídeo donde implementamos la red normal previamente aquí definimos una función que se llama scores donde lo que hacíamos era hacer todas estas operaciones pasábamos hacíamos la función lineal parámetros por x más el ballas luego la activación y luego la salida de la última capa que eran los scores aquí lo que vamos a hacer es simplemente la operación para una sola capa lineal es decir wx más b es todo lo que vamos a implementar otra cosa que vamos a hacer aquí es que en vez de llamar a la función forward que podríamos hacerlo lo que vamos a hacer es hacer uso de guión bajo guión bajo col que lo que nos permite es utilizar la instancia de la función lineal como si fuera una función y pasar los parámetros directamente esta este tipo de función este tipo de funciones que llevan guión bajo guión bajo se llaman donder y en el caso de col está hecha para para esto para que la instancia de la del objeto que creemos opere como una función y es más vamos a hacer muy rápido una función tipo domí una función muy sencilla y un tanto trivial que nos permita entender mejor qué es lo que pasa cuando utilizamos la función col vamos a definir una clase que se llame suma por decir algo y lo único que va a hacer es sumar dos números lo que voy a hacer es un def guión bajo col es más primero vamos a decir que tenemos un def de suma que se llama así la función de un def que se llama sum norms vamos a decirlo así y que reciba dos números x y lo que haga sea return x más ya ahora si quisiéramos mandar a llamar esta crea crear una instancia de este objeto vamos a decir a es igual a suma así entonces hayas una instancia de este objeto de hecho lo podemos checar a es un objeto tipo suma está bien entonces ahora si hacemos a de vamos a decir a punto son nom y mandamos dos y tres nos debe de volver cinco acuerdo funciona bien pero tuvimos que mandar invocar la función a punto som si ahora lo que hacemos es cambiar a y le ponemos la el nombre de la función que un bajo que un bajo col que un bajo que un bajo y volvemos a instanciar a ahora obviamente nos va a mandar un error porque ya no está definida esta función pero si ahora nada más llamamos a y le pasamos directamente los valores dos y tres directamente va a llamar esta esta función con bajo y bajo y vemos que efectivamente funciona esto es lo que hace la función guion bajo guion bajo col es decir se llama de forma automática cuando mandamos a llamar el nombre de la instancia de la clase el nombre del objeto que creamos y simplemente le pasamos los parámetros de tal forma que cuando creemos nuestra instancia del de la clase lineal cuando mandemos a llamar ese nombre que asignemos a la instancia y le pasemos como parámetro los datos de entrada o los datos de la capa previa se va a ejecutar esta función por lo tanto aquí recibiríamos obviamente self y los datos de entrada x y lo que va a pasar acá es que vamos a hacer la operación lineal como habíamos dicho z va a ser igual vamos a llamarlos z porque recuerden que decimos que z es a esta operación w x más b es más dejen hacerlo aquí z sería igual a w por x más b y no nos preocupemos como hemos dicho ahorita por las dimensiones simplemente esto es lo que vamos a implementar ya hemos explicado también a detalle cómo hacer que las matrices las dimensiones de las matrices coincidan para que el producto sea válido entonces esto sería self de w recordemos que podemos utilizar la roba para hacer el producto por x más self de b es la única operación que va a realizar aquí nuestra función en forward en este caso el forward por default va a ser en col podemos poner un comentario que diga que diga este es la función este es el forward forward de la función lineal de la clase lineal y finalmente vamos a regresar z y lo único que nos hace falta es implementar la función backward esta sí la tenemos que definir backward y lo que vamos a hacer como técnica de implementación de la función backward para todas nuestras nuestras clases es que recibamos una entrada y una salida es decir por ejemplo en el caso de una capa lineal aquí vamos a recibir en una capa lineal siempre una entrada que va a ser la salida de la capa previa puede ser después de pasar por la función de activación es uno nos importa porque no cambia el número de entradas entonces va a ser lo que sea que viene de la de la capa previa y la salida va a ser esta comúnmente al entrada vamos a usar un nombre genérico y le vamos a llamar x y a la salida le vamos a llamar z simplemente para seguir con la nomenclatura que hemos venido utilizando en estas ecuaciones la entrada es x y la salida z entonces aquí lo que hacemos es que vamos a recibir una x la salida va a ser z esto va a ser genérico para cualquier capa es decir no importa que sea la primera capa oculta la segunda capa oculta o la última capa de la red normal al entrada de forma genérica le vamos a llamar x y a la salida le vamos a llamar z vamos a definir cómo calcular el gradiente y recordemos el gradiente por ejemplo de esta capa recordemos que aquí va a llegar en esta capa el gradiente de la función de pérdida con respecto a esta z de alguna manera ese gradiente se ha calculado y nos va a llegar a esta capa es decir el gradiente que se recibe aquí esto también lo hemos explicado muy a detalle en otros vídeos es decir por ejemplo si nos imaginamos que esto es un son todas las neuronas de una capa aquí recibimos x la salida de z por aquí se calcula la función de vamos a decir que fuera la final la última capa aquí se calcula softmax más cross entropy y aquí sale la función de pérdida y ya como ya hemos explicado podemos calcular el gradiente de la función de pérdida con respecto a z y aquí llega ese gradiente de regreso se retro propaga el gradiente de la función de pérdida con respecto a z y es este gradiente el que vamos a utilizar primero para calcular el gradiente que se propaga aquí de regreso que sería el gradiente de l con respecto a x recordemos que aquí pueden haber otras neuronas conectadas otras capas de neuronas conectadas entonces este gradiente se retro propaga para quien para las capas previas puede ser una capa de activación no puede ser otra capa lineal en el vídeo anterior o en el vídeo anterior donde implementamos la red neuronal vamos a verlo aquí lo tengo es la libreta que implementamos en ese vídeo podemos ver que calculamos el gradiente por ejemplo este gradiente de a1 es ese gradiente que se está retro propagando para las capas previas y lo que hacíamos era la multiplicación de los parámetros con el gradiente que recibimos z2 de z2 y utilizábamos los nombres de las variables la deriva de a1 y de z2 ahora lo que vamos a hacer es que el gradiente lo vamos a estar recibiendo en esta variable z pero recordemos que z es el resultado de cfw por x más b y al hacerse al ser w y b objetos de la clase que definimos en p tensor z también va a convertirse en este tipo de objeto por lo tanto nosotros vamos a recibir el gradiente de z ya no en una variable que se llame de z sino en el atributo punto grad de dicha variable por lo tanto aquí lo que vamos a hacer es que vamos a guardar el gradiente de x también el atributo gradiente de x para esto vamos a tener que asegurarnos que x lo convirtamos a un objeto en p tensor eso lo vamos a hacer cuando mandemos a llamar el modelo esto va a ser igual a la misma ecuación que programamos aquí los parámetros por la derivada de la función de pérdida con respecto a z y esto va a ser vemos que recordemos que es la transpuesta de los parámetros para que las dimensiones coincidan por z punto grad recordemos el gradiente de z va a estar guardado en ese atributo grad quizá en este momento puede ser un poco confuso que no vemos cómo este z y este z donde se calcula el gradiente pero cuando creemos nuestro modelo secuencial vamos a ver que vamos a interconectar una capa después de la otra y así al tenerlas en forma consecutiva como una cadena cuando mandemos a llamar la función back guard de la instancia del objeto lineal de la clase lineal ya este valor se va a haber calculado y ya se va a haber calculado su gradiente por lo tanto al momento de pasar este objeto z el atributo punto grad ya va a contener el gradiente ahora lo que vamos a hacer es calcular el gradiente de la función de pérdida con respecto a x y lo vamos a guardar en el atributo punto grad de la variable x vamos a repetir el proceso ahora pero para la los parámetros de peso sinápticos y para los parámetros de vayases recordemos que esta neurona o esta capa recuerden que es una capa de renorna también recibe los parámetros w y los parámetros b y necesitamos calcular las derivadas la función de pérdida con respecto a w y con respecto a b para poder actualizar dichos parámetros ya también hemos explicado con bastante detalle cómo se realizan estas operaciones tanto de forma teórica como de forma práctica de igual forma es más aquí lo tengo de esta libreta que tengo abierta vamos a ver que podemos calcular la derivada de w como el gradiente de la función de pérdida con respecto a z por las entradas que son las salidas de la capa previa transpuestas entonces vamos a implementar exactamente esa opción vamos a empezar con cdf de w punto grad vamos a guardarlo en el atributo gradiente en el atributo punto grad de w y esto va a ser igual a el gradiente de z por x punto t y x recuerden que es el nombre genérico que estamos utilizando para las entradas y es exactamente esta ecuación notemos que ahora no vamos a dividir entre el batch size como hicimos la vez anterior que dividíamos entre el número de elementos la lógica de que realmente no es necesario dividir entre el batch size es porque lo que estamos calculando acá es el gradiente vamos a llamarlo vamos a ver aquí por ejemplo vamos a calcular vamos a decir que esto es el gradiente este es w y estamos calculando el gradiente con respecto a w entonces cuando actualizamos los parámetros haciendo uso de gradient descent y utilizamos w y esto castigating descent que es lo único que hemos hecho w menos el gradiente de la función de pérdida con respecto a w aquí estamos multiplicando siempre el gradiente de la función de pérdida con respecto a w que es este gradiente lo estamos multiplicando por un valor un learning rate que es un número pequeño entonces si dividimos aquí entre un número como el número de elementos que puede ser 512 por ejemplo o 1024 lo único que está pasando es que esa división entre el número de elementos se puede de alguna manera decir que está integrada en este valor alfa en este learning rate tal vez si lo único que va a pasar es que si no dividimos aquí entre el número de elementos tenemos que utilizar un learning rate un poco menor para para que de alguna manera de forma integrada implemente esa operación y luego lo mismo vamos a hacer para ahora el vallas self punto b punto grad va a estar dado por la suma de de la misma es la misma ocasión que hicimos la vez pasada set a grad y keep teams y cual si recordamos z tiene dimensiones que es del número de neuronas por el número de elementos en el mini batch y al hacer axis 1 lo que está haciendo es que está sumando todos los elementos a través de las columnas es decir todos los ejemplos del mini batch resultando simplemente en números que van a coincidir con el tamaño de b por ejemplo si z fuera una matriz de 10 por 64 donde las 64 columnas representan un ejemplo diferente lo que está haciendo es sumar todos los ejemplos y eso es lo que se va a agregar al vallas de nueva cuenta no dividimos entre el número de elementos y así tenemos nuestra clase lineal ya tenemos lista es todo lo que hay que hacer para instanciarla ahora que terminemos vamos a vamos a mostrar cómo hacerlo y ahora lo que vamos a hacer es crear nuestra clase relu para esto vamos a hacer nuevamente clases y ahora se va a llamar relu y no vamos a inicializar parámetros para esta clase porque relu como función de activación no tiene ningún parámetro simplemente recibe los elementos de la capa anterior y los pasa después de haber aplicado la función que aplica rel entonces lo único que vamos a hacer es nuevamente utilizar la misma estrategia de crear la función con col para que podamos llamar directamente al instancia de la clase y pasarle los valores como si fuera una función y especificar qué es lo que va a recibir y si pensamos en la función de activación la función de activación relu por ejemplo esta lo que recibe es z es decir el producto de w x más b y saca aquí la función de activación que le vamos a llamar a entonces la entrada que va a recibir va a ser z si recordamos relu tiene esta forma entonces realmente lo único que está haciendo es que está eliminando en cero todos los números negativos los está convirtiendo en cero entonces es simplemente nada más con vamos a hacer return non pi maximum entre cero y la entrada que sería z eso es lo único que va a ser el forward pastor relu en esta parte de cero pues el gradiente va a ser cero y en esta parte el gradiente va a ser uno entonces realmente lo único que se hace es que es que se deja pasar el gradiente tal cual para los elementos de entrada z que son mayores a cero y para los elementos que son menores a cero pues se ponen en cero no hay gradiente que se retro propague y entonces vamos a verlo aquí dependiendo de los elementos estos elementos de entrada los elementos que tienen un valor positivo el gradiente se va a dejar pasar tal cual a las neuronas que correspondan con un valor que hayan arrojado un z positivo mientras que aquellos valores con que son negativos pues no no recibirán un gradiente de regreso entonces simplemente lo único que vamos a hacer es de backward si nos recibimos la entrada y recordemos que los vamos a definir de forma que siempre digamos recibamos una entrada y una salida en este caso decíamos que la salida de relu se llama a entonces le ponemos z y a y lo que vamos a hacer es calcular el gradiente que va a regresar acá este gradiente de aquí que sería el gradiente de él con respecto a z que ahora lo que vamos a hacer es llamarlo z punto grat y este va a ser es decir lo estamos guardando en el atributo grat de la variable z y ahora lo que hacemos es que bueno lo primero que vamos a hacer es crear una copia de los elementos que estamos recibiendo en el caso de relu recordemos va a estar tenemos la función de activación aquí relu aquí arroja la activación recibe z y aquí recordemos que está recibiendo el gradiente de la función de pérdida con respecto a esa activación y lo primero que vamos a hacer es crear una copia de esa activación porque lo único que va a hacer es distribuir este mismo gradiente dejarlo pasar tal cual en los elementos donde z es mayor mayor que 0 y convertirlo a 0 en los elementos donde z es 0 creamos una copia de a el gradiente de de a nuevamente tiene un atributo grat y se va a hacer el primer valor y ahora vamos a actualizar este valor de z grat lo que vamos a decir es que los valores donde z sea menor o igual a 0 ponerlos en 0 realmente el gradiente cuando en z igual a 0 no está definido pero vamos a poder definirlo y no hay ningún problema decir que si es 0 el gradiente se igual a 0 y esto es todo lo que tenemos que hacer para la clase relu y estamos llegando ya a la parte más entretenida a la parte más interesante que nos va a permitir implementar un modelo con un número de capas arbitrario vamos entonces a implementarlo vamos a decir que va a ser una clase le vamos a poner nombre sequential vamos a poner sequential lejos para hacerlo diferente por ejemplo de pytorche que tiene un módulo que se llama precisamente sequential entonces la intención es que esto trabaje un poquito parecido vamos a hacer un def realmente inicializamos parámetros vamos a recibir y vamos a recibir un número de capas no elementos de capas y aquí las layers es una lista con que contienen objetos de tipo kapa y en este caso nada más tenemos objetos de tipo linear y relu entonces son los únicos objetos que podemos pasar y ahora si lo primero que vamos a hacer es tener una lista propia que incluye a las capas que sea una instancia del objeto que creemos donde va a recibir simplemente las capas la lista con las capas que le pasemos layers se llama vamos a ponerle layers aquí para que sea plural y lo siguiente que vamos a tener es un objeto que vamos a poner x y esta esta variable va a almacenar las salidas z o las activaciones a de cada capa cuando estemos haciendo el forward pass de cada capa esto nos va a permitir simplemente de forma temporal almacenar todos los elementos de esta capa si por ejemplo son 100 neuronas pues va a almacenar los elementos las activaciones de esas 100 neuronas y vamos a crearlo por lo pronto con no no le vamos a crear ningún valor y lo siguiente sería también vamos a tener un diccionario que se va a llamar outputs y este diccionario lo que va a guardar son las activaciones de cada capa pero ahora sí como de forma un poco más permanente para poder implementar también el proceso de retro pagación entonces en cada iteración de cada capa el valor que se almacena en self luego se va a almacenar en este diccionario y lo siguiente sería vamos a llamar nuevamente crear una función que se va a llamar call siguiendo la misma metodología que hemos venido trabajando va a recibir self y va a recibir x inicialmente lo x van a ser los valores las imágenes es decir los valores de entrada a la capa estos valores acá serían los que inicialmente se van a recibir porque recordemos que este modelo incluye todas las capas entonces va a recibir la entrada la primera entrada la red neuronal vamos a hacer primero que los primeros valores de self x va a ser igual a x y luego lo siguiente sería que outputs y esto a cada capa vamos a llamar la l más el número de la capa vamos a decir simplemente que la capa cero sea igual a los elementos de entrada la capa 1 ya va a ser la primera capa de la red neuronal que va a tener parámetros pero para poder hacer literación de forma eficiente nos conviene decir que la capa cero sean los elementos de entrada y así cuando hagamos nuestro loop para llamar a las funciones de forer pass el primer elemento va a corresponder a las imágenes de entrada y ahora sí vamos a hacer nuestro loop y lo que vamos a hacer es enumerar todos los elementos que hay en layers y ahora vamos a empezar desde 1 ya que el elemento 0 es el elemento de entrada de las imágenes entonces de forma temporal guardamos en x el layer layer aquí recordemos y esto es lo que nos permite hacer el enumerado es que layer va a contener cada objeto cada elemento de self punto layers y recordemos que layers contienen objetos de tipo linear o reloj es decir de tipo capa de red neuronal como utilizamos todos los objetos creamos todas las clases con este tipo de definición de guión bajo guión bajo colgión bajo guión bajo simplemente con llamar al objeto que está contenido en este en este elemento layer vamos a ponerlo así y poner la entrada nos va a permitir hacer el forer pass para esa capa si fuera una capa de tipo lineal nos va a permitir implementar esa función para esta capa si es una capa de tipo reloj nos va a permitir implementar esta función y lo que va a hacer es que va a implementar la función forward guión bajo guión bajo colg para los elementos que están en self x en la primera instancia self x va a tener los elementos de entrada y posteriormente en cada iteración aquí vamos a guardar esos elementos x y ahora lo que vamos a hacer es guardarlo en autos tal vez esto se pudiera hacer de forma más eficiente directamente utilizando altos pero creo que la forma más ilustrativa es haciéndolo así con una variable separada que nos permita guardar los elementos de forma temporal para cada activación y entonces outputs de layer y vamos a decir que ahora va a ser string de e va a ser igual a self dot x simplemente vamos a regresar el último valor de self dot x y esto es porque en este valor en la última iteración va a tener el valor de la salida de la última capa es decir el valor de los scores y con esto al mandar a llamar el objeto que instanciamos de esta clase nos va a arrojar además de guardarnos todos estos parámetros de cada capa que nos van a permitir calcular el back propagation para cada capa también nos va a arrojar directamente los scores de nuestro modelo es decir los valores a la última capa en la salida de la red normal y ahora vamos a definir una función que va a ser una función que se va a llamar backward y esta función nos va a permitir implementar el backward pass para todas las capas de la red normal va a recibir solamente self y entonces lo que va a ser la función backward es implementar la función backward de cada una de las capas de las instancias de los objetos de capa contenidos en self punto layers para esto es necesario hacer el proceso en reversa recordemos que el proceso de retro propagación se realiza a partir de la última capa hacia atrás hacia la primera capa entonces vamos a hacer un for donde recorramos todas las capas pero a partir partiendo de la última y para esto hacemos un for y ya no utilizaremos en un marito porque no podemos hacer en un número en un marito pero en reversa lo que vamos a hacer es reversed y para esto vamos a hacer un reverse que esté en el rango de el número de elementos que lo vamos a sacar con len de la lista que contiene las capas de nuestro modelo que sería self punto layers y con esto vamos a hacer un loop que vaya si son tres capas por ejemplo bueno va a ir de 2 1 0 y aquí nada más vamos a implementar de self punto layers del elemento y es decir la capa y y ahora si vamos a llamar a punto backward por eso definimos la función backward que siempre reciba una entrada y una salida vemos que en lineal también recibe una entrada y una salida y entonces así podemos implementar el backward para cada elemento en self punto layers recuerden que en outputs estuvimos guardando cada activación en cada capa por lo tanto self punto outputs y aquí es importante notar qué es lo que estamos recibiendo recordemos es entrada y salida las entradas estarían guardadas en la capa y directamente string de y y luego recibiría los elementos de salida de esa capa que están guardados en outputs pero ahora sería en y más uno de la capa y más uno y esto es lo que estaríamos pasando a la función backward de cada objeto que forma nuestro modelo de reteneronal entonces estos son los elementos que son la entrada de la capa estos son los elementos que son la salida de la capa y así estamos pasando z como a o x como a cita si se trata de una función de una clase un elemento de la clase lineal perfecto y ya con eso implementamos el backward pass para todas las capas de la reteneronal la siguiente función que vamos a implementar es una función que nos permita actualizarlos parámetros de nuestra reteneronal esto lo vamos a hacer para cada capa por lo tanto vamos a definir nuestra función que se llama update va a recibir self y va a recibir también un learning rate y vamos a decir que se igual a 1 y a la -3 por decir algo y vamos a hacer un for que nos permita ir recorriendo todas las capas y ahora van a ver capas que van a tener parámetros es decir todas las que corresponden a línea pero van a ver capas que no tengan parámetros como las que corresponden a reloj entonces para no tratar de actualizar parámetros en una capa que no tiene parámetros como reloj vamos a decir que si la capa y es instants la capa que estamos analizando es decir esta capa de esta iteración pertenece a la instancia reloj a la clase reloj entonces vamos a continuar si nos saltamos esa capa no actualizamos nada en caso contrario vamos a actualizar ahora los valores de los parámetros que corresponden a esta capa para esto lo que vamos a hacer nada más algo que vale la pena notar tomar un unito y notar es que w son w y b son atributos que pertenecen al objeto layer que recibimos en esta lista de capas recordemos que vemos aquí que w pertenece a este a esta clase linear por lo tanto lo que haríamos aquí es decir layer punto w sería igual al valor actual de esa variable w menos learning rate por y aquí recordemos nuevamente que es el gradiente de este valor w que estaría en la y el punto w punto grat recordemos que guardamos los gradientes en el atributo grat de cada w aquí podemos ver las ventajas de utilizar esta este tipo de herencia en nuestra en nuestro modelo donde creamos nuestros parámetros utilizando herencia de un objeto de tipo non pa y realmente no cambiamos nada simplemente esto nos permite poder utilizar el punto grat agregar un atributo punto grat a nuestros parámetros y finalmente el layer punto p que sería igual a la y punto b menos el learning rate por la y de b punto grat simplemente hacemos una actualización basada en esto castigar en descent no hemos visto métodos de optimización más sofisticados posteriormente los vamos a ver pero por el momento sería esto dependiendo de qué kappa estemos analizando se actualizarán los parámetros correspondientes a esa kappa y ya ahora sí ya lo único que nos queda vamos a crear una función una función que se llama predict que esta función la vamos a utilizar simplemente cuando ya tengamos nuestro modelo entrenado para predecir cuál es la salida de nuestro modelo que realmente podríamos utilizar simplemente la misma la misma función no necesitamos tener una función adicional pero podemos hacerla y que de una vez nos calcula cuál es el elemento mayor porque por ejemplo si mandamos a llamar nada más a nuestro modelo recordemos que esto nos arroja los scores no nos arroja cuál es el mayor score cuál es la clase que tiene el mayor score entonces en predico que vamos a hacer es que nos arroje eso se va a recibir los elementos de entrada y lo que va a regresar es el máximo de los elementos en el vector de salida de los scores por lo tanto vamos a decir self punto col de los elementos en la es decir va a mandar a llamar a esta función pero nos va a devolver el índice del elemento que tiene que es el más grande de esos scores por lo tanto esa va a ser ese índice corresponde a la clase de dicha imagen o a la clase que nuestro modelo está prediciendo y con esto terminamos la clase sequential ya casi estamos listos ahora vamos a utilizar la función de costum en este vídeo vimos cómo programar la clase lineal reloj y sequential que nos permiten construir un modelo de red normal con tamaño arbitrario en el siguiente vídeo vamos a utilizar este modelo que ya podemos crear para entrenar nuestra red neuronal por lo tanto vamos a ver cómo programar la función de costo y la función softmax así como como implementar el modelo o el loop de entrenamiento continuamos y y [MÚSICA]