10 - ¡Red Neuronal usando PyTorch! PyTorch tutorial.

En vídeos previos utilizamos Python y nonpy para desarrollar modelos de redes normales con un número definido o con un número arbitrario de capas.

Para esto primero utilizamos funciones y después desarrollamos nuestras clases que nos permitieron instanciar nuestros propios modelos de capas de red normal, así como la clase sequential layers que nos permitió definir un modelo con un número arbitrario de capas.

Además implementamos todo, es decir, implementamos a mano el proceso de back propagation el cual es comúnmente omitido en la mayoría de los vídeos instruccionales de deep learning y redes neuronales.

Sin embargo, nosotros ya sabemos cómo implementarlo en código.

Para esto tuvimos que implementar los gradientes y esto nos ha permitido tener un gran entendimiento de cómo funciona el proceso de entrenamiento de una red neuronal.

Ahora vamos a utilizar PyTorch, el cual es uno de los frameworks más populares de deep learning que nos va a permitir implementar modelos mucho más complejos.

Además nos vamos a dar cuenta de que la forma de implementar modelos con PyTorch haciendo uso del módulo sequential va a ser muy parecida a la forma en la que implementamos nuestro modelo en el vídeo donde desarrollamos nuestra clase sequential layers.

Entonces a partir de este vídeo vamos a basar nuestros modelos en PyTorch.

Vamos a programar.

Y aquí tenemos la libreta de Jupyter que vamos a utilizar en este proyecto para implementar nuestro modelo de red neuronal utilizando PyTorch.

Empezamos como siempre importando non-py y matplotlib como hemos venido utilizando habitualmente.

En esta ocasión además vamos a importar los módulos que corresponden de PyTorch.

Vamos a importar Torch completo que es el módulo que contiene todas las funciones y todos los módulos o submódulos de PyTorch.

Y además también vamos a hacer referencia a torch.neuralnetworks.nn como nn.

Esto para facilitar nuestra implementación y que tengamos que escribir menos código cuando hacemos referencia a los módulos que pertenecen al módulo de neural networks de PyTorch.

Y asimismo importamos functional que pertenece a neural networks.

Y para este hacemos referencia como f. f y nn son los nombres que usualmente se utilizan en proyectos que utilizan PyTorch.

Entonces ejecutamos esta celda y ahora continuamos importando las imágenes.

Para importar las imágenes vamos a hacer uso de la base de datos MNIST nuevamente y así vamos a utilizar la función getimages que nosotros mismos implementamos en un vídeo anterior.

Aquí dejo la liga a ese vídeo.

Es importante notar que PyTorch tiene funciones que nos permiten importar la base de datos MNIST y que nos permite crear el set de datos de entrenamiento, validación y pruebas de forma nativa utilizando las funciones de PyTorch.

Sin embargo para hacer esto de forma más genérica y mostrar que podemos trabajar con nuestros propios datos sin hacer uso de librerías de PyTorch y aún así utilizar las funciones de PyTorch para entrenar nuestro modelo de red neuronal y por supuesto crear nuestro modelo de red neuronal.

Entonces voy a importar getimages aquí mandamos a llamar la función getimages donde vamos a generar los datos de entrenamiento y prueba y después vamos a dividirlos en entrenamiento, validación y pruebas como hemos hecho en vídeos previos.

Por lo tanto no voy a detenerme aquí mucho tiempo.

Convertimos aquí los datos de entrenamiento las imágenes, las convertimos a vectores.

Eso hacemos con Sturvy Shape y los ponemos como punto flotante.

Entonces importamos imágenes o bueno más bien convertimos imágenes que ya tenemos guardadas directamente de la página de MNIST en esta ruta.

Normalizamos las imágenes como hicimos en el vídeo donde creamos nuestras propias clases para implementar nuestro modelo de red neuronal.

Esta es la función normalize que nos va a permitir tener una media de cero y una desviación estándar de uno y normalizamos nuestros datos.

Podemos checar que efectivamente los datos ahora tienen una media de cero aproximadamente de cero y una desviación estándar aproximadamente de uno.

Si queremos ver las imágenes bueno primero aquí estamos viendo esta instrucción nos permite ver que la cual es la forma de los datos que realmente los datos las imágenes en el caso de los datos de prueba tienen son 10.000 imágenes y todas ya están en forma de vector como vimos en líneas anteriores y si quisiéramos ver por ejemplo aquí los datos de entrenamiento pero para el caso de las etiquetas para las 10 vemos que son 50.000 datos de entrenamiento y solamente una etiqueta para cada uno de los datos es decir 0123456789 dependiendo de qué dígito estamos reconociendo y si queremos ver los datos bueno esta función ya la conocemos también de vídeos anteriores nos va a permitir aquí mostrar la imagen sale aquí podemos cargar algunas imágenes aleatorias y ver qué cómo son las imágenes de el ministro.

Entonces aquí tenemos la función de crear mini batches esto es para que durante el proceso de entrenamiento no mandemos todos los datos completos porque esto puede causarnos problemas de memoria sin embargo tampoco es conveniente mandar un dato a la vez ya que sería muy ineficiente de tal forma que vamos a crear mini batches como hemos explicado previamente podemos indicar el tamaño del mini batch y vamos a indicar que por default ordene de forma aleatoria los datos que pertenecen al mini batch.

También no voy a tener aquí mucho tiempo ya que hemos explicado esta función varias veces en vídeos previos.

Lo más importante es esta parte de aquí donde se regresa un iterador un objeto iterable de python donde lo que hacemos es tomar los datos desde y hasta y más el tamaño del mini batch tanto para las imágenes como para sus correspondientes etiquetas entonces eso es todo lo que tenemos que hacer aquí entonces voy a ejecutar esa función y la forma de llamar esta función va a ser haciendo uso de un for donde en cada iteración del for vamos a tener un nuevo mini batch y ahorita lo vamos a ver en la función de entrenamiento de nuevo dejo por aquí referencias a vídeos donde hemos explicado esto a más detalle y ahora sí vamos a implementar todas las funciones y los ajustes necesarios para poder correr nuestro nuestro código en pytorch.

Lo primero que tenemos que hacer es convertir todos los arreglos de non-py que contienen los datos de entrenamiento validación y pruebas a tensores de pytorch para esto extrane estos datos que tenemos acá arriba extrane y train x val y val y así los de prueba también tenemos que convertir los atensores y para esto podemos utilizar la función torch.tensor entonces lo que vamos a hacer es tomar el extrane guion bajo lo vamos a poner de nombre tensor para notar que este es el tensor de pytorch y simplemente hacemos torch.tensor y aquí mandamos a llamar ex guion bajo train ahora es posible que simplemente hacemos esto arroje un warning donde nos dice que se está tratando de que los datos de non-py no son writable o no pueden ser escritos y lo más seguro para para evitar este tipo de warning aunque funcionaría el código funcionaría es que convierta una copia de los datos no convertir los datos directamente esta es la forma más correcta de hacerlo entonces vamos a hacerlo aquí vamos a hacer lo mismo pero ahora para los datos de entrenamiento y train perfecto ahora vamos a copiar esto vamos a copiar esto dos veces más para los datos de validación y de pruebas entonces aquí sería val y sería también y ahora lo que vamos a hacer aquí es que tiene que ser llegaval y aquí también tiene que ser y repetimos también para los datos de pruebas y entonces notamos que ahora tenemos un tensor que se llama extrane tensor que contiene los datos originales pero convertidos ahora tensor los datos extrane lo mismo para y train checamos que validación y validación está correcto y validación y validación x test y test está correcto entonces ejecutamos esta celda y así continuamos lo que vamos a hacer a continuación es determinar si tenemos un GPU disponible y esta es una de las ventajas principales de pytorch y que nos permite simplemente importando las librerías que mostramos al principio podemos utilizar los GPU en caso de tenerlos disponibles para esto lo que vamos a hacer es checar si tenemos un GPU disponible esto lo hacemos con torch dot a CUDA punto is available esto nos da a arrojar verdadero en caso de que tengamos un un GPU disponible una tarjeta de video entonces vamos a poner que el nombre usual es device device va a ser igual a torch torch punto device en caso de que no esté disponible device va a ser igual a privates y aquí pondríamos CPU y vamos a imprimir estamos usando y lo que tengamos no en este caso vamos a imprimir device y aquí podemos ver que en nuestro caso estamos utilizando un GPU es decir estamos utilizando CUDA perfecto y ahora sí vamos a implementar las funciones que si van a tener algunos cambios con respecto a las funciones que implementamos previamente no obstante estos cambios van a ser muy muy sutiles y solamente se refieren a cómo utilizamos aquí el modelo por ejemplo la función de acura así que nos va a permitir evaluar el desempeño de nuestro modelo recibe un parámetro que se llama modelo motel en este caso este es nuestro modelo de pytorch que vamos a utilizar por el momento aún no hemos instanciado el modelo de pytorch lo vamos a utilizar aquí utilizando sequential para esto lo que vamos a hacer es crear un modelo le podemos poner modelo uno va a ser igual a el módulo neural networks nn de pytorch que importamos como nn es decir este que importamos aquí podríamos hacer lo mismo utilizando toch punto nn y aquí punto sequential pero no lo vamos a hacer así lo que vamos a hacer es como lo referimos como nn simplemente utilizamos nn punto sequential y aquí vamos a poner nuestro modelo sale aquí vamos a poner las capas lineales las capas que vayamos a utilizar en nuestro modelo sin embargo todavía no lo hemos implementado le podemos poner aquí un comentario que diga implementar modelo instanciar a instanciar modelo modelo uno no y aquí vamos a dejarlo así vamos a comentarlo ese es el modelo que va a recibir acura así cuando mandemos a llamar la función x y y van a ser los datos o el set de datos que vamos a utilizar para calcular dicho acura si y aquí el tamaño del minibatch ya que para calcular los acura si vamos a tener que ir generando llamando nuestro modelo corriendo haciendo el forward pass de nuestro modelo pero para esto vamos a utilizar minibatch es la misma manera que utilizamos para el entrenamiento entonces de la misma manera que anteriormente vamos a hacer el número de elementos correctos vamos a llevar este conteo en la variable en un correct y luego vamos a hacer un total y aquí también va a ser igual a cero y lo que sigue es que vamos a poner nuestro modelo en modo de evaluación esta es una particularidad de paitos y esto lo que nos permite es que aquellas capas que se comportan de manera diferente en modo entrenamiento que en modo de evaluación se comportan la forma correcta porque simplemente estamos evaluando el modelo después vamos a ver cuáles son estas capas si por el momento no nos vamos a preocupar simplemente vamos a ponerlo en modo en modo evaluación y lo siguiente va a ser que pongamos nuestro modelo vamos a recibir nuestro modelo y vamos a hacer que nuestro modelo lo sea igual a nuestro modelo pero lo vamos a poner vamos a indicarle que el modelo corra en el dispositivo que estamos utilizando el modelo corra en este caso en gpu con el punto 2 indicamos en dónde queremos cargarlo todo esto para que haga uso del gpu que se cargue en la memoria del gpu y ahora vamos a hacer algo diferente a la implementación de la función accuracy previa lo que vamos a hacer es ejecutar lo que sigue del código sin calcular el gradiente si entonces hacemos un touch y un bajo no grad y esto lo que nos va a permitir es que para todo el código que continúa no se va a calcular el gradiente derivado de que paitos hace el auto grad con respecto a todas las variables que corresponden a los parámetros de nuestro modelo de red normal lo que va a pasar con esta al hacer esto es que en este caso aunque estemos ejecutando el modelo y el modelo tiene parámetros que tienen habilidad la opción de que se calcula el gradiente de forma automática esto por cierto lo hace directamente sequential en videos futuros vamos a hacer lo vamos a mostrar cómo se puede hacer de forma manual pero utilizando paitos entonces en este caso lo que vamos a hacer aquí con no grad es que no calcula el gradiente con respecto a las variables o parámetros la red normal y ahora sí vamos a hacer nuestro for donde vamos a mandar a llamar o vamos a ponerle x y y en y en create vamos a mandar a llamar a función de create mini batches y create mini batches recibe si vemos aquí x y mb size entonces vamos a mandarle aquí en este caso x y y vemos mb size vamos a ponerle mb size y en este caso vamos a dejarle shufl por default en verdadero que reorden los datos de forma aleatoria esta es la forma en la que vamos a iterar el objeto que devuelve el objeto iterable que devuelve la función create mini batches en cada iteración de este for va a arrojarnos un nuevo mini batch en x y y ahora lo que tenemos que hacer es cargar estas variables x y y y a la memoria de gpu de la misma forma en la que cargamos el modelo en la memoria de gpu entonces haríamos x y 2 y aquí ponemos device y hacemos lo mismo para un comentario aquí lo que quiero es copiarlo hacemos lo mismo para y entonces aquí las pusimos en memoria gpu y lo siguiente va a ser correr el modelo de la misma manera que como implementaríamos el loop de entrenamiento de aquí ya no tenemos cambios de aquí ya realmente los cambios la función acura si con respecto a nuestro a nuestras implementaciones anteriores son estas partes que incluyen poner el modelo en modo de evaluación cargar el modelo en gpu y ejecutar todo este código sin calcular el gradiente esto se hace porque es mucho más eficiente hacer esto y si estamos utilizando haciendo uso de una parte de código que no necesita calcular gradientes es mucho mejor si hacemos esto de esta manera así nos ahorramos bastantes recursos de nuestro gpu lo siguiente entonces sería calcular los scores para esto vamos a mandar a llamar al modelo y pasar los datos x y que les pasamos aquí en la función acura si pueden ser los de validación pueden ser los de pruebas o incluso podríamos calcular a que hubas y con los datos de entrenamiento si así lo deseáramos una vez que calculamos ya los scores ahora lo que vamos a hacer es calcular cuál es el score más alto para cada uno de los resultados que arroja que arroja mandar a llamar nuestro modelo para esto ahora lo que vamos a hacer es una función que nos calcule cuál es el índice que contiene el score mayor recordemos que score si ponemos aquí un comentario score va a tener una forma que si vemos un solo para un solo un solo elemento del mini batch va a tener una forma que va a ser de una fila por 10 columnas donde cada columna va a corresponder al score para cada clase entonces lo que queremos es detectar encontrar cuál es el score más alto y el índice de ese score pertenece al número que estamos identificando por ejemplo si el score más alto está en el índice 5 esto significa que la clase que está prediciendo nuestro modelo es 5 es decir que el número la imagen que estamos analizando pertenece a un dígito 5 no esa sería nuestra predicción y aquí tendríamos el tamaño del mini batch no realmente no sería uno sino sería el resultado de scores corresponde al tamaño del mini batch si entonces tenemos esto para cada uno de los elementos del mini batch entonces ahora lo que vamos a hacer es tomar scores punto max y esto es porque torch scores va a ser un tensor de pytorch y entonces a diferencia de como hacíamos con non-py donde lo que hacíamos era hacer uso de la función np punto arc max aquí simplemente vamos a hacer uso de max y vamos a indicarle que sea en la dimensión 1 que corresponde en vez de axis con pytorch vamos a utilizar dim y que sea la dimensión 1 es decir que nos elija el índice del elemento mayor a través de las columnas nos va a devolver esto un valor para cada uno de los elementos en el mini batch utilizamos aquí guion bajo porque el método max para un tensor devuelve dos valores devuelve el primer valor que es el valor del número máximo es decir si el score mayor tiene un valor arbitrario de vamos a decirle score entonces devolvería ese valor y además el segundo parámetro que devuelve es el índice de ese valor en este caso no nos interesa saber cuál es el valor simplemente nos interesa saber cuál es el índice de dicho valor entonces por eso nos quedamos simplemente con el segundo valor que arroja el max y ahora lo que vamos a hacer es proceder de la misma manera que hicimos para nuestra implementación previa donde vamos a tomar los números correctos y los vamos a aumentar con los valores de pred que sean iguales los valores en este vector que sean iguales a los valores en ye y de nuevo a cuenta vamos a hacer un punto squeeze como hicimos en el vídeo previo y esto es porque si recordamos ye si vemos aquí cuando mandamos a llamar a la forma de ye hicimos el punto shape ye tiene una forma de de matriz aunque sea una matriz de una sola columna pero pero es así no el número de filas y una columna y el número la forma de los elementos que son arrojados por pred que están arrojados y guardados aquí en pred simplemente es un vector que corresponde a una sola dimensión es decir es un vector de la forma vamos a ponerlo acá pred tiene la forma pred shape shape es de la forma simplemente si tenemos mb size no tiene coma uno entonces para poder hacer y hacer la comparación de pred con ye ye tiene vamos a poner el comentario nada más este es la forma de pred pero la forma de ye es shape es así mb size coma 1 entonces hacemos el punto squeeze para que elimine este este uno y quede de la forma simplemente mb size y se puede comparar directamente con pred con el vector contiene pred y ahora continuamos y lo que vamos a hacer es acumular el número de elementos que sean que sean procesado entonces aquí vamos a hacer nom total nom y abajo total total sería igual a el número de elementos en el minibatch que en este caso sería pred punto size de 0 como pred es un tensor hacemos uso del método size en vez del shape como haríamos normalmente si se tratara de una regla de non pi y con esto lo que vamos a hacer ya finalmente va a hacer regresar en valor flotante el número de correctas dividido entre el número de total o me acabo de dar cuenta aquí tenemos que poner puntos son para que recordemos que aquí cuando hacemos pred y comparamos con ye lo que nos va a devolver es un true para los elementos que son iguales y un false para los elementos que no son iguales que se pueden interpretar como un 1 o un 0 pero lo que queremos hacer es sumar todos estos elementos entonces los true se interpretan como uno y los sumamos y esos son los que acumulamos si entonces me faltaba aquí este esta función afortunadamente nos dimos cuenta a tiempo y no nos va a dar errores al menos de los errores que tal vez nos mande cuando corramos todo nuestro modelo este no va a ser uno de ellos y obviamente aquí me acabo de dar cuenta también nos llama total se llama nom total si esa variable total no sé por qué le puse total y ahora si vamos a implementar nuestra función de entrenamiento para esto vamos a definir la función train que va a recibir un modelo de pytorch va a recibir el optimizador el optimizer que es la forma en la que se van a actualizar los parámetros de nuestra red normal va a recibir también el tamaño del mini batch y va a recibir un número de hip hooks en este caso por default se va a poner que sean 100 pero esto se puede pasar como un parámetro adicional entonces notemos que de forma muy similar a la función acura si el primer parámetro que se recibe es el modelo de pytorch por lo tanto vamos a empezar la función de entrenamiento haciendo que nuestro modelo esté cargado en memoria de gpu por lo tanto vamos a ser un model punto 2 a device como hicimos previamente en la función acura si después lo que vamos a hacer es recorrer todo nuestro modelo por todo nuestro proceso de entrenamiento si entonces vamos a hacer un for y poc y range y poc para que entrenemos por el número de hip hooks que hayamos establecido y después vamos a hacer un paso de entrenamiento para cada uno de los mini batches entonces sabíamos for y vamos a ponerle x y como a y es exactamente el mismo for que utilizamos para calcular la cura si en create mini batches y aquí si notamos vamos a vamos a ver crea y mini batches recibe x en este caso x sería x los datos de entrenamiento pero en formato de tensor luego los datos de entrenamiento también pero para el caso de las etiquetas en formato de tensor y luego vimos que sería el tamaño del mini batch mv size y el último parámetro es shuffle que por default vamos a dejarlo en tru entonces no cambiamos eso y hacemos así el recorrido para cada mini batch que se genere en nuestro set completo de entrenamiento notemos que a diferencia de la función acura si aquí no queremos hacer with torch punto no grat no lo queremos hacer porque definitivamente como es el entrenamiento aquí queremos calcular el gradiente de la función de costo con respecto a nuestros parámetros entonces queremos que se calcule el gradiente y ahora lo que vamos a hacer es poner nuestro modelo en modo de entrenamiento y vamos a ver ahorita por qué porque dentro del for vamos a mandar a llamar a la función acura si que va a poner el modelo en modo evaluación y si luego cuando regresamos no lo ponemos otra vez en un entrenamiento se va a quedar en modo evaluación por lo tanto en este caso es necesario ponerlo dentro del for y pasamos los datos a aquí sería perdón x y los pasamos x y punto 2 device de igual forma hacemos lo mismo para las etiquetas notamos que realmente se parece mucho al código que utilizamos para que hubo así y no sería una sorpresa que el siguiente paso va a ser precisamente calcular los scores de model para x y y si nos damos cuenta de nuevo muy parecido lo que acabamos de hacer en acura sin sin embargo aquí vienen las partes son diferentes tenemos que calcular la función de costo y vamos a poner un comentario aquí que sea costo función función costo vamos a ponerlo así y entonces lo que vamos a hacer es que nuestra función de costo nuestro costo va a ser igual y aquí es donde mandamos a llamar a f si nos acordamos dijimos que f es la forma en la que vamos a identificar los módulos de functional dentro del módulo de neural network de pytorch entonces hacemos f y aquí hacemos cross entropy que es la función de costo que hemos venido utilizando siempre es la que ya explicamos y aquí la función de costo recibe directamente los scores entonces si vemos aquí lo que puede recibir recibe input vemos que recibe target y recibe input en este caso lo que tiene que recibir son los scores input serían los scores aquí dentro de esta función se calcula el softmax entonces no lo tenemos que implementar de forma muy parecida a como hicimos en el vídeo donde creamos nuestra propia clase nuestro propio objeto y nuestra propia clase para sequential entonces se implementa directamente la función de softmax directamente dentro de cross entropy y el siguiente sería los el target vale entonces sería le ponemos aquí target que sería y y así ya con la función de costo calculada tenemos que proceder de forma muy similar a como hemos hecho en vídeos anteriores tenemos que calcular los gradientes la función de costo con respecto a los parámetros y después actualizar los parámetros con dichos gradientes sin embargo dado que estamos utilizando pytorch pytorch acumula el gradiente si nosotros no lo ponemos en cero entonces queremos que en cada iteración el gradiente se ponga en cero porque de lo contrario estaría acumulando el gradiente con respecto al gradiente de la iteración previa que sería del mini batch anterior y eso es algo que no queremos por lo tanto un paso que tenemos que hacer adicional es optimizer punto cero grad y así ponemos los gradientes en cero y ya de nueva cuenta que sería entonces la función de costo vamos a calcular el backward pass donde se calcula el gradiente de la función de costo con respecto a todos los parámetros y finalmente hacemos un optimizer recordemos que el optimizer es la forma en la que vamos a actualizar los parámetros optimizer punto este con esto tenemos ya implementado nuestro loop de entrenamiento nos damos cuenta que es muy parecido a lo que ya hemos hecho en el pasado ahora si queremos hacer algo que nos permite imprimir en cada hip hop por ejemplo podemos hacer un print y vamos a ponerle aquí vamos a ponerle hip hop ponemos dos puntos y aquí ponemos el hip hop y para este hip hop la función de costo es costo el valor de la función de costo sería 2 punto ups aquí como es tenemos que mandar el atributo item esta es una particularidad bar to by torch y podemos calcular el accuracy accuracy sería me faltó aquí terminar de cerrar la llave si y aquí tenemos accuracy sería igual a vamos a poner los puntos que también tenemos esto y sería la función accuracy recibe el modelo luego recibe x y que en este caso vamos a hacerlo con los sets de validación sería x y bajo val y queremos pasar los tensores y guay guay y en bajo val tensor y que más nos dice mb size el tamaño del mini batch vamos a ponerle podríamos y quisieramos pasar un tamaño de mini batch diferente pero en este caso vamos a dejarlo a cómo estaba y esto ahora si ya cerramos nuestro función dejen hacerlo un poquito más pequeña nada más para ver que estamos viendo que está completo f y poc estamos imprimiendo el impo que estamos imprimiendo el costo para esa para notamos que vamos a vamos a imprimir el del último si el del último valor no importa podríamos irlo acumulando si quisiéramos y calcular un promedio de todos los costos para cada mini batch en este caso se va a imprimir el último pero está bien va debe en la tendencia debe ser de que debe ir a la baja entonces lo dejamos así y finalmente hacemos a cura si que recibe el modelo podemos verlo aquí recibe el modelo x y el mini batch entonces x los datos de validación muy importante que sacan los datos de validación no con los de prueba todavía el modelo ahora perdón las etiquetas y el tamaño del mini batch y creo que ya terminamos y con esto terminamos nuestro loop de entrenamiento o aquí hay algo que se me hace que me ha faltado para hacer y es poner de type de type igual a torch punto era float y aquí también nos hace falta de type igual a torch de la misma forma que hicimos aquí en por cierto también teníamos que hacerlo aquí en accuracy no lo hicimos y type igual a torche punto 32 y aquí y type igual a torche excelente escutamos esa salda nuevamente es bueno revisarlo por eso nos damos cuenta de estos detalles este y bueno es un es un dolor de cabeza menos al momento de escutar el código pero son tropicó que está bien vamos a escutar nuestro loop de entrenamiento y ahora lo que sí tenemos que hacer finalmente vamos a instanciar nuestro modelo y para esto ya se va a parecer mucho esto es lo más fácil se va a parecer mucho a la forma en la que instanciamos nuestro modelo cuando creamos nuestra propia clase sequential layers donde creamos nuestro propio framework pequeñito de learning si en este caso vamos a utilizar nn.sequential y aquí lo que va a recibir son las capas que vamos a implementar entonces podemos hacerlo vamos a hacerlo todo aquí en la misma línea vamos a implementar una capa lineal y la capa lineal recibe las features de entrada las features de salida y el parámetro del bias que por default estén true lo vamos a dejar así entonces lo que vamos a hacer es que para infeatures sea el tamaño de parámetros el tamaño que el tamaño de elementos el número de elementos en nuestro vector de entrada es decir en nuestros datos para este caso infeatures va a ser igual a 784 vamos a ponerlo de forma manual vamos a poner aquí out features y vamos a ponerle una variable que se llama hidden y lo que vamos a hacer aquí es crear que nuestra variable hidden para poderlo cambiar de forma fácil sea igual a es más vamos a ponerle hidden 1 sea igual a vamos a decir a 1000 neuronas y aquí sería hidden 1 entonces esta es nuestra capa lineal luego lo que vamos a hacer es agregar una capa de activación relu si no recibe parámetros luego vamos a poner otra capa lineal en punto línea y vamos a poner que infeatures ahora sea igual a es más no voy a poner ahora infeatures para ahorrarme un poco de espacio o lo que podemos hacer mejor aún vamos a poner esto en la siguiente línea para que vayamos aquí perfecto así va a ser infeatures va a ser igual a hidden 1 out features va a ser igual a vamos a crear hidden vamos a ponerle hidden nada más y después vamos a pasar una capa junto relu ahora podemos agregar una última capa vamos a ponerle aquí hidden sea igual también vamos a dejarle mil y finalmente ok tenemos una capa lineal aquí tenemos una capa lineal fully connected seguida de una capa relu tenemos otra capa lineal seguida de una capa relu y vamos a poner la última capa que va a ser la capa de salida que va a ser también lineal lineal va a recibir aquí hidden como input features y como output features va a ser el número de clases que le podemos poner aquí un 10 directamente si queremos ya les sabemos que son 10 clases y así terminamos nuestro modelo sequential vamos a correrlo a ver si nos manda algún error aquí tochen n has no relu ah dónde está esto ah relu aquí debe ser con mayúscula nada más vamos a correrlo para ver si el modelo se implementa bien o que ya parece que no mando errores entonces tenemos el modelo ya creado y ahora lo que vamos a hacer es mandar a llamar llamar este modelo y bueno tenemos el modelo creado ese es un parámetro que se va a pasar el siguiente parámetro es el optimizer entonces vamos a definir nuestro optimizer aquí que se llama optimizer va a ser igual toch vamos a hacer uso de toch punto el módulo optim y aquí podemos indicar cualquier optimizador que queramos el único que hemos cubierto es esto castigrate en descent es el que hemos estudiado y es el que vamos a utilizar por este momento sin embargo podríamos utilizar adame o algún otro otro otro modelo entonces es más podemos ver aquí rápido la lista de posibilidades adelta adacrat adame es quizá el más común y lo vamos a estudiar en el futuro por lo pronto vamos a quedarnos simplemente con esto castigrate en descent que es el que hemos estado estudiando y ahora los parámetros que se recibirían es indicar qué es lo que queremos optimizar para esto lo que vamos a hacer es decir modelo 1.0 parámetros esto es lo que queremos optimizar y podemos checar acá que se requiere un learning rate entonces vamos a poner learning rate igual a vamos a poner lr y ahorita creamos una variable aquí arriba que se llame lr se da igual a y vamos a empezar con un parámetro relativamente pequeño 1 e a la -3 por poner un parámetro por default y ahora ya tenemos nuestro optimizer y finalmente estamos listos a bueno otra otra variable que se va a recibir es el número de hipox y pox vamos a decir que sean 20 y pox para no tardarnos tanto y ya vamos a mandar a llamar nuestra función train que ahora sí va a recibir el modelo va a ser modelo 1.0 el optimizador optimizer va a ser model 1.0 que va a ser optimizer y vemos que recibe el mini batch size esa es otra variable que tenemos que poner vamos a ponerle mv y bajo size y era el número de hipox y así sería simplemente vamos a crear también mv size que no le hemos definido y ésta vamos a ponerle que sean 2048 y si todo está bien en este momento vamos a ver que nuestro modelo que pasó a nuestro modelo va a entrenar y contamos siempre es emocionante la primera vez que vamos a implementar un loop de entrenamiento porque creemos que todo está bien a veces no funciona a veces sí vamos a ver mando un error no nos vamos a preocupar optimizer touch touch optin que nos dice in it got an expected learning rate ok el parámetro no se llama learning rate se llama lr ok lr igual a lr ok ahora sí parece que está corriendo mando otro error vamos a buscarlo la función de costo que nos dice f input igual scores target igual a ge ok que es esto ok ya sé qué es lo que está pasando pusimos ge pero recordemos que lo tenemos que convertir a un vector si es quiz sale si no tendría la forma número de elementos coma 1 queremos que simplemente sea un vector de una dimensión número de elementos intentamos ahora y volvemos a correr y vemos qué pasa y está entrenando ahora no hay que ponernos tan contentos porque empezó muy bajo y terminó también muy bajo entonces nos hace falta entrenar más este modelo para esto vamos a ver qué qué podemos hacer que nos permite hacer un ajuste y decir ok por ejemplo el learning rate está un poquito bajo vamos a empezarlo con 1 punto uno por diez al menos dos y definitivamente pongamos más más grande el mini batch vemos que entrenó sumamente rápido vamos a hacerlo de 4096 los mini batches siempre es bueno ponerlos en potencias de dos ya que los frameworks de deep learning están eficientados para esto entonces 2048 por 2 sería 4096 y vamos a hacer 50 y pox sale y vamos a ver qué pasa y vemos que mucho más rápido está llegando a vamos a dejarlo entrenar y llegó a 90% ok estamos llegando 90% relativamente rápido sin embargo podemos hacer más y pox o sea estamos viendo que que lo está haciendo muy rápido si está llegando 50 y pox muy rápido vamos a hacer un entrenamiento más donde hagamos es más déjenme comentar esta parte y lo que voy a hacer aquí es que entrene por otro 50 y pox y vamos a entrenar con un learning rate un poquito más pequeño en estos otros 50 y pox a partir de ese punto entonces podemos ver que ahora empieza ya desde 90 va en 91 por ciento tal vez podríamos hacerlo entrenar un poquito más pero creo que voy a hacer algo diferente lo que voy a hacer vamos a generar un entrenamiento más un último entrenamiento nada más para ver qué pasa vamos a dejar esto hacia cómo está y lo que voy a hacer ahora es algo diferente vamos es un modelo vamos a hacer un modelo aún más sencillo con una sola capa de entrada y una capa de salida con mil neuronas en cada en cada una si vamos a quitar esta y a veces un modelo más grande es más difícil de entrenar no quiere decir que un modelo más pequeño es mejor pero es más difícil de hacer tuning y para no tardarnos mucho vamos a hacer una última iteración con este modelo vamos a hacer uno si vamos a hacer uno a la menos dos y vamos a correrlo y vemos que rápido está llegando a 90 por ciento excelente está entrenando nuestro modelo está entrenando llegó empezó en 60 por ciento y fue rápido llegando a 90 por ciento si quisieramos hacer otros 50 y pox ahora sí comentando esta parte vamos a probar con 5 a la menos dos más grande ver qué pasa va llegando 92 93 94 ok nos hacía falta un learning rate más grande es más vamos a hacerlo finalmente vamos a hacer un último entrenamiento 100 y pox más a partir de ese modelo de ese mismo o sea que ya llevamos 50 50 100 y esto sería 200 y pox vemos que ya llega a 96 por ciento llegamos a 97 por ciento en este momento sería un buen un buen punto para intentar con un learning rate más pequeño pero lo voy a dejar así vemos que después de estos últimos 100 y pox llegamos a 97 por ciento de accuracy y realmente para llegar al 99 por ciento o a un porcentaje más alto tendríamos que hacer un proceso de tuning de sintonización de nuestro modelo que nos va a llevar un poco más de tiempo sin embargo para fines didácticos para fines de este vídeo podemos ver que logramos entrenar nuestro modelo muy parecido a lo que veníamos implementando previamente pero ahora ya lo hicimos con no tenemos que aquí comente este modelo comente estas líneas para que el entrenamiento se realizara a partir de este mismo modelo si o sea ya cuando ya habíamos llegado a 90 por ciento lo que hice fue probé con otro learning rate y aumenté el número de hipoks pero esto es hacer 100 hipoks adicionales a los que ya veníamos haciendo como si hubiéramos hecho si hicimos 50 primero y 100 más sería hacer 150 hipoks si hubiera dejado estas líneas sin comentar el modelo habría sido creado nuevamente desde cero y entonces no habíamos visto esta mejora desde 95 por ciento vamos a hacerlo una última vez vemos que ahora empieza en 82 por ciento y pronto va a llegar a 90 seguramente va a llegar a 96 por ciento sin problemas pero ahí empezó otra vez como desde un punto de partido inicial porque ya porque volvimos a generar el modelo a crear el modelo otra cosa que es importante le quité aquí el número de parámetros le quité algunos parámetros vamos a probar nuevamente con otra capa pero es importante notar que no quiere decir que por tener un modelo más complejo este va a ser no va a ser tan bueno al contrario simplemente un modelo más complejo es más difícil de sintonizar por eso tal vez pueda parecer que no se por comportó también pero vamos a probar lo vamos a probar con este modelo lo estamos generando nuevamente con 100 hipoks y un learning rate de 5 por 10 la menos 2 empiecen 74 pero llega 90 por ciento sin problemas y ahora va en el hipok 30 también va a llegar a 97 por ciento sin problemas definitivamente hacía falta que el learning rate sea mayor en este caso estamos simplemente designando el learning rate al de dazo prácticamente llegó a 96.5 por ciento no obstante la forma correcta de hacerlo que le explicaremos en un vídeo futuro es hacer una búsqueda dentro de un espacio de rangos posibles y determinar en ese dentro de ese espacio cuál es el learning rate que mejor funciona y después hacer una búsqueda aún más fina en un espacio muy pequeño dentro del learning rate que hayamos encontrado en este caso vamos a dejarlo así 96.5 por ciento está bien para fines de este vídeo y lo último que vamos a hacer es ahora sí checar la cura así con nuestros datos de prueba entonces lo que haremos será x guion bajo test guion bajo tensor lo primero es que el modelo x y ok entonces pasamos el modelo se llama modelo 1 x y en bajo y en bajo y lo que vamos a hacer ahora es a pasarle un tamaño del mini batch y vamos a calcularlo y nos da que el acura si final es 95.9 por ciento si es sería el resultado que reportaríamos 95.9 por ciento casi 96 por ciento en un proceso de entrenamiento sumamente sencillo donde no hicimos ningún tipo de túnel entonces terminamos con esto hemos terminado un vídeo donde lo que estamos haciendo es entrenar nuestro modelo utilizando pytorch les agradezco mucho la atención prestada este vídeo en este vídeo creamos un modelo de red neuronal haciendo uso de pytorch pudimos ver que la forma de declarar nuestro modelo es muy parecida a la que realizamos en el vídeo donde creamos nuestros propios objetos nuestras propias clases para formar nuestra red neuronal además pudimos utilizar gpus y notamos que el entrenamiento es mucho más rápido por lo tanto a partir de ahora dado que ya entendemos muy bien cómo funciona red neuronal porque la programamos desde cero vamos a utilizar pytorch para crear modelos más complejos en vídeos futuros lo que vamos a ver son redes neuronales convolucionales las cuales nos permiten resultados muy muy buenos principalmente para imágenes o para datos que tienen cierta relación espacial vamos a estudiarlas vamos a crear modelos con redes neuronales convolucionales y también vamos a hacer algunas cosas interesantes con ellas como por ejemplo analizar cómo se comportan las neuronas a través de las diferentes capas de una red neuronal convolucional donde vamos a generar y observar ahí imágenes muy interesantes más adelante también veremos redes neuronales recurrentes y sus variantes como son el estiems o grus vienen temas muy muy interesantes y si han llegado hasta este momento y han visto estos vídeos les agradezco enormemente su tiempo su atención y su dedicación nos vemos en el siguiente vídeo muchas gracias continuamos y [MÚSICA]