

Sesión 1:

Conceptos básicos de PyTorch, Redes Neuronales y el MultiLayer Perceptron (MLP)

Elvin Mark Munoz Vega

1. Objetivos
2. Introducción a PyTorch
3. Redes Neuronales
4. MLP: MultiLayer Perceptron

Objetivos

1. Aprender a usar la librería PyTorch
2. Aprender los fundamentos básicos de Redes Neuronales

Introducción a PyTorch

Frameworks más utilizados en Machine Learning

Entre los frameworks más utilizados tenemos:

1. TensorFlow (Google)
2. PyTorch (Facebook AI's Research Lab)
3. MXNet (Apache)
4. caffe (Berkeley)
5. scikit-learn

Frameworks más utilizados en Machine Learning

Entre los frameworks más utilizados tenemos:

1. TensorFlow (Google)
2. PyTorch (Facebook AI's Research Lab) ← Utilizaremos este framework!
3. MXNet (Apache)
4. caffe (Berkeley)
5. scikit-learn

Porque PyTorch?

- Es uno de los frameworks más utilizados en la investigación de redes neuronales. Y esta tendencia sigue creciendo.


Porque PyTorch?

- Es uno de los frameworks más utilizados en la investigación de redes neuronales. Y esta tendencia sigue creciendo.
- Es mucho más fácil de utilizar que otros frameworks. Permite el uso de GPUs de una manera muy sencilla entre otras cosas.

Porque PyTorch?

- Es uno de los frameworks más utilizados en la investigación de redes neuronales. Y esta tendencia sigue creciendo.
- Es mucho más fácil de utilizar que otros frameworks. Permite el uso de GPUs de una manera muy sencilla entre otras cosas.
- La generación dinámica de grafos lo hace más Pythonista. Contrario con otros frameworks, en los que primero tienes que generar todo el grafo de operaciones.

Todo son tensores. escalares, vectores, matrices ...



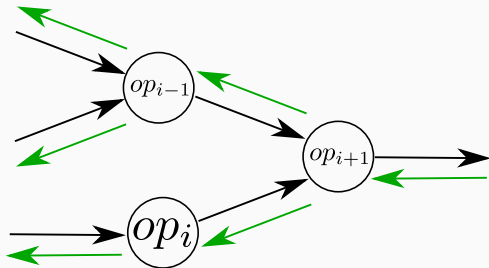
```
a = torch.tensor(2.0)           # Rango 0
v = torch.tensor([0.0,1.0])     # Rango 1
M = torch.tensor([[0.0,1.0],[2.0,3.0]]) # Rango 2
# ...
```

Autograd: Auto diferenciación

El corazón de la librería PyTorch: Calcular derivadas de manera automática.

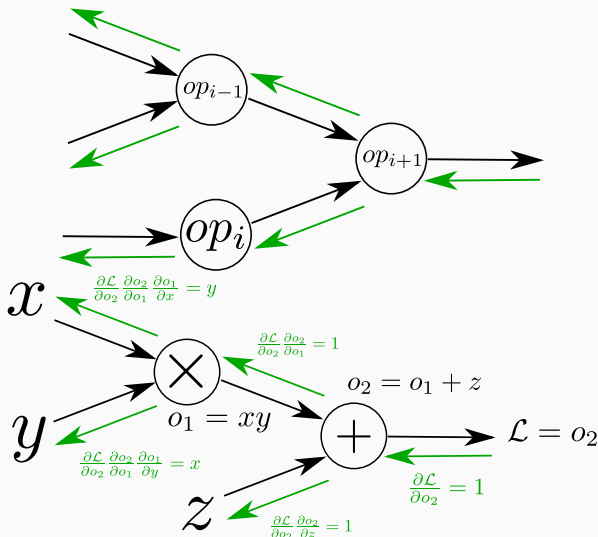
Autograd: Auto diferenciación

El corazón de la librería PyTorch: Calcular derivadas de manera automática.



Autograd: Auto diferenciación

El corazón de la librería PyTorch: Calcular derivadas de manera automática.



Tiempo de practicar!

Notebook para la Clase 1:

https://colab.research.google.com/drive/1_ZegHWY5N3HepfAA9y-2l60P41eQ_HnK?usp=sharing

Tambien lo pueden encontrar aqui:

<https://www.google.com>

Redes Neuronales

Que son las redes neuronales?

Que son las redes neuronales?

Podemos definir una red neuronal como una función $\mathcal{M}(\mathbf{x}|\mathbf{W})$ que tiene (muchos) parámetros (\mathbf{W}) que son “entrenados” para que realice una tarea específica. Por ejemplo, Reconocimiento de imágenes, generación de imagenes, detección de objetos, etc.

Que son las redes neuronales?

Podemos definir una red neuronal como una función $\mathcal{M}(\mathbf{x}|\mathbf{W})$ que tiene (muchos) parámetros (\mathbf{W}) que son “entrenados” para que realice una tarea específica. Por ejemplo, Reconocimiento de imágenes, generación de imagenes, detección de objetos, etc.

Las operaciones entre los parámetros \mathbf{W} y la entrada \mathbf{x} son de diversos tipos. Puede ser una simple multiplicación matricial, convolución entre tensores, etc.

Que son las redes neuronales?

Podemos definir una red neuronal como una función $\mathcal{M}(\mathbf{x}|\mathbf{W})$ que tiene (muchos) parámetros (\mathbf{W}) que son “entrenados” para que realice una tarea específica. Por ejemplo, Reconocimiento de imágenes, generación de imagenes, detección de objetos, etc.

Las operaciones entre los parámetros \mathbf{W} y la entrada \mathbf{x} son de diversos tipos. Puede ser una simple multiplicación matricial, convolución entre tensores, etc.

Lo que hace tan poderosa a una red neuronal son principalmente 3 cosas:

Que son las redes neuronales?

Podemos definir una red neuronal como una función $\mathcal{M}(\mathbf{x}|\mathbf{W})$ que tiene (muchos) parámetros (\mathbf{W}) que son “entrenados” para que realice una tarea específica. Por ejemplo, Reconocimiento de imágenes, generación de imagenes, detección de objetos, etc.

Las operaciones entre los parámetros \mathbf{W} y la entrada \mathbf{x} son de diversos tipos. Puede ser una simple multiplicación matricial, convolución entre tensores, etc.

Lo que hace tan poderosa a una red neuronal son principalmente 3 cosas: la cantidad de parámetros,

Que son las redes neuronales?

Podemos definir una red neuronal como una función $\mathcal{M}(\mathbf{x}|\mathbf{W})$ que tiene (muchos) parámetros (\mathbf{W}) que son “entrenados” para que realice una tarea específica. Por ejemplo, Reconocimiento de imágenes, generación de imagenes, detección de objetos, etc.

Las operaciones entre los parámetros \mathbf{W} y la entrada \mathbf{x} son de diversos tipos. Puede ser una simple multiplicación matricial, convolución entre tensores, etc.

Lo que hace tan poderosa a una red neuronal son principalmente 3 cosas: la cantidad de parámetros, las operaciones dentro de la red neuronal,

Que son las redes neuronales?

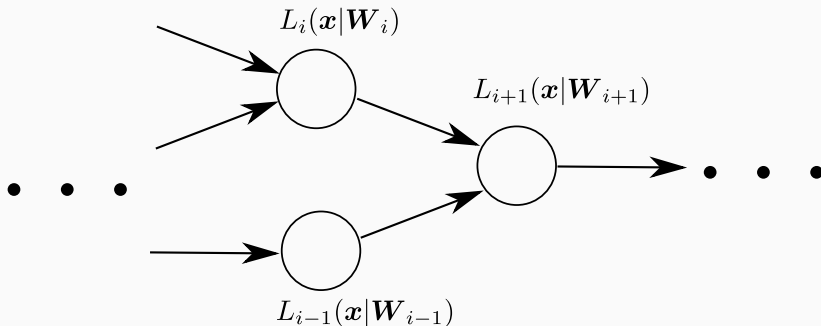
Podemos definir una red neuronal como una función $\mathcal{M}(\mathbf{x}|\mathbf{W})$ que tiene (muchos) parámetros (\mathbf{W}) que son “entrenados” para que realice una tarea específica. Por ejemplo, Reconocimiento de imágenes, generación de imagenes, detección de objetos, etc.

Las operaciones entre los parámetros \mathbf{W} y la entrada \mathbf{x} son de diversos tipos. Puede ser una simple multiplicación matricial, convolución entre tensores, etc.

Lo que hace tan poderosa a una red neuronal son principalmente 3 cosas: la cantidad de parámetros, las operaciones dentro de la red neuronal, la cantidad de datos utilizados en su entrenamiento.

Capas de una red neuronal

Podemos dividir nuestra red neuronal en pequeños bloques llamados capas o “layers” (L_i). Cada uno de estos layers L_i utiliza parte de todos los parámetros de la red neuronal para realizar cierto tipo de operación. En otras palabras dividiremos nuestros parámetros \mathbf{W} en pequeños subgrupos $\{\mathbf{W}_1, \mathbf{W}_2, \dots, \mathbf{W}_n\}$, donde cada subgrupo de parámetros \mathbf{W}_i es utilizado por la capa L_i .

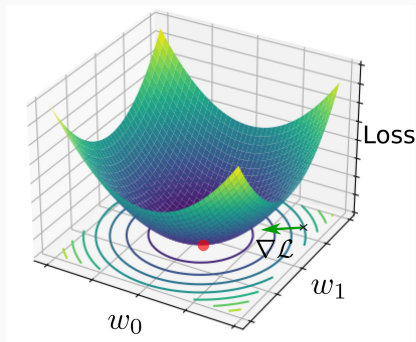


Forwarding simplemente significa evaluar nuestro model en los inputs que tengamos.

Importante: Es durante este proceso donde tambien se va generando el grafo que guarda información de las operaciones hechas.

Función de Perdida (Loss Function)

- El loss function es simplemente una medida que nos indica que tan bien entrenado está nuestro modelo.
- Nuestro objetivo: encontrar los parámetros que minimicen el loss function.
- Diferentes tipos de loss function dependiendo de la tarea resolver: Mean Square Error Loss (Regresión), Cross Entropy Loss (Clasificación).



Usando el grafo con el registro de las operaciones hechas durante el forwarding, realizamos el backpropagation del loss para poder calcular la gradiente de cada parámetro.

Estas gradiente serán usadas por los optimizadores más adelante para poder minimizar el loss function.

Optimizadores (Optimizers)

La tarea del optimizador es de minimizar el loss function. Tenemos diferentes tipos de optimizadores

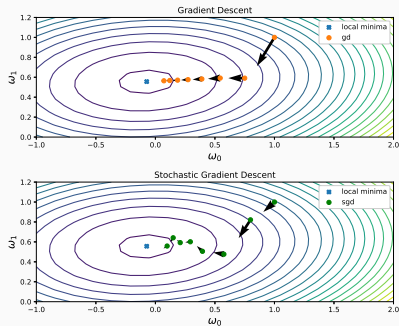
- Optimizadores de primer orden (First Order Optimizers)

 - SGD, Adam, AdaGrad, ...

- Optimizadores de segundo orden (Second Order Optimizers)

 - KFAC, Natural Gradient, ...

SGD: Stochastic Gradient Descent



- Stochastic: probabilidad y aleatoriedad.
- Tenemos un límite de memoria en los computadores.
- Entrenar por lotes o “batches”.
- Generar los batches **aleatoriamente**.

Otros (variaciones) optimizadores de primer orden

- SGD + Momentum
- AdaGrad
- Adam
- SAM
- y muchos más ...

Optimizadores de segundo orden

- Mientras que los optimizadores de primer orden utilizan la gradiente, los optimizadores de segundo order utilizan el Hessiano. (Natural Gradient Method)
- O aproximaciones del Hessiano (Fisher Information Matrix). (KFAC)

Básicamente el algoritmo para entrenar una red neuronal en general sigue el siguiente pseudocódigo.

preparar datos de entrenamiento $input, target$

for $epoch \leftarrow 0, N$ **do**

$output \leftarrow M(input|W)$

$loss \leftarrow \mathcal{L}(output, target)$

$W \leftarrow W - \lambda \nabla \mathcal{L}$

MLP: MultiLayer Perceptron

Que es un MLP?

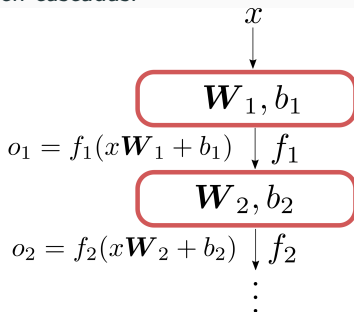
El MultiLayer Perceptron (o Perceptron Multicapas), consiste principalmente de 2 tipos de capas puestas en cascadas.

Transformación Lineal:

$$L(x|\mathbf{W}) = y = x\mathbf{W}$$

Función de Activación:

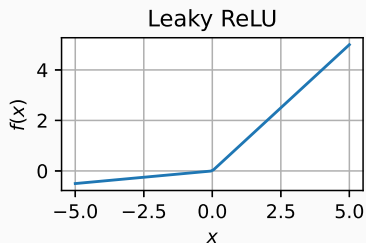
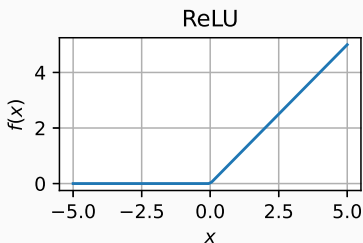
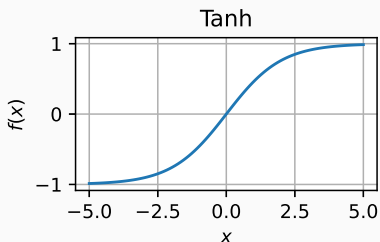
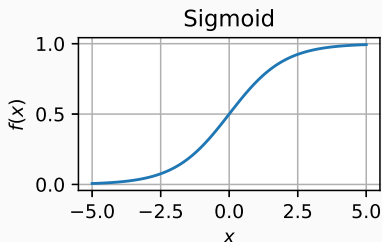
$$L(x) = f(x)$$



Importante notar que las funciones de activaciones son las que le dan las propiedades no lineales a nuestro modelo.

Funciones de Activación

Diferentes tipos de funciones de activación: Sigmoid, Tanh, ReLU, LeakyReLU, ...



Usemos el siguiente MLP de 2 capas
(sin función de activación en la
ultima capa):

$$y = f_1(x\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2$$

$$o_1 = x\mathbf{W}_1 + b_1$$

$$o_2 = f_1(o_1)\mathbf{W}_2 + b_2$$

$$y = o_2$$

MLP: Backwarding

Calculamos la perdida $\mathcal{L} = \mathcal{L}(y, t)$.

Luego calculamos la gradiente de la perdida con respecto al output

$g = \frac{\partial \mathcal{L}}{\partial y}$, para luego propagar esta gradiente en la red.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial o_1} &= \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial o_2} \frac{\partial o_2}{\partial o_1} \\ &= f'_1(o_1) g \mathbf{W}_2^T\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial x} &= \frac{\partial \mathcal{L}}{\partial o_1} \frac{\partial o_1}{\partial x} \\ &= \frac{\partial \mathcal{L}}{\partial o_1} \mathbf{W}_1^T\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{W}_2} &= \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial o_2} \frac{\partial o_2}{\partial \mathbf{W}_2} \\ &= f_1(o_1)^T g\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} &= \frac{\partial \mathcal{L}}{\partial o_1} \frac{\partial o_1}{\partial \mathbf{W}_1} \\ &= x^T \frac{\partial \mathcal{L}}{\partial o_1}\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial b_2} &= \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial o_2} \frac{\partial o_2}{\partial b_2} \\ &= 1 \cdot g\end{aligned}$$

$$\frac{\partial \mathcal{L}}{\partial b_1} = \frac{\partial \mathcal{L}}{\partial o_1} \frac{\partial o_1}{\partial b_1} = 1 \cdot \frac{\partial \mathcal{L}}{\partial o_1}$$

Tiempo de practicar!

Notebook para la Clase 1:

https://colab.research.google.com/drive/1_ZegHWY5N3HepfAA9y-2l60P41eQ_HnK?usp=sharing

Tambien lo pueden encontrar aqui:

<https://www.google.com>