Apuntes IA Semana 7 21/3/2024l

1st Gilberth Rodríguez Mejías
2020061179
Tecnológico de Costa Rica
Cartago, Cartago

I. Redes Neuronales

A. Repaso clase pasada

Se volvió a ver el data set de MNIST, el cual es un conjunto de números escrito a mano en imágenes transformadas de 28x28 píxeles todas en blanco y negro. La regresión logística solo sirve para identificar si una imagen es un numero en especifico, el ejemplo de clase es con el numero 5, donde la regresión logística solo puede identificar si la imagen es un 5 o no lo es. Para pasar esta imagen como entrada a una regresión logística se convierte en un vector plano (Flatten). Dando como resultado 784 entradas para la regresión logística, una por cada píxel de la imagen.

Para crear una regresión logística para múltiples opciones se utilizan varias regresiones loísta, donde el resultados de ellas seria un One-hot vector. Se retomo ejemplo de la clase pasada donde 10 personas tenían que identificar los números del 0 al 9, si la persona se encarga de identificar el 5 y la imagen tiene un 5 indica con un 1 que si es, en caso contrario un 0. Dando como resultado un vector de 10 de largo.

class	one-hot codification
0	1 0 0 0 0 0 0 0 0 0
1	0 1 0 0 0 0 0 0 0 0
2	0 0 1 0 0 0 0 0 0 0
3	0 0 0 1 0 0 0 0 0 0
4	0 0 0 0 1 0 0 0 0 0
5	0 0 0 0 0 1 0 0 0 0
6	0 0 0 0 0 0 1 0 0 0
7	0 0 0 0 0 0 0 1 0 0
8	0 0 0 0 0 0 0 0 1 0
9	0 0 0 0 0 0 0 0 0 1

Fig. 1. One-hot Vector

Si enviamos la primera entrada a las 10 regresiones logísticas al mismo tiempo se agiliza el análisis, pero porque no enviar toda la imagen de 784 píxeles al mismo tiempo a las 10 regresiones. Esto agiliza en gran medida los cálculos haciendo que las 10 regresiones analicen la imagen completa al mismo tiempo. Y devolviendo como salida un One-hot vector.

En lugar de calcular cada regresión lineal de forma vectorial porque no combinar todos los vectores en una matriz para hacer una sola operación. Así se crea la matriz de pesos W. Como se muestra en la imagen 2 donde m seria el tamaño de la capa y n seria el tamaño de los features. En el caso de la imagen esta matriz seria de 784x10, m = 10 neuronas de la capa (el tamaño de la capa) y n = 784 la cantidad de features.

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix}$$

Fig. 2. Matriz de Pesos

Con respecto al bias pasa de ser un escalar a un vector, este vector almacenaría los 10 bias de las 10 regresiones.

Lo siguiente es un ejemplo del calculo de dos funciones primero de manera normal y luego con matrices

2 operaciones

$$x = [3, 4, 5, 6]$$

$$W_1 = [3, 2, 4, 5]$$

$$b_1 = 2$$

$$W_1 * x + b_1 = [3, 4, 5, 6] * [3, 2, 4, 5] + 2 = 67 + 2 = 69$$

$$sigmoid(W_1 * x + b_1)$$

$$x = [3, 4, 5, 6]$$

$$W_2 = [4, 3, 2, 1]$$

$$b_2 = 3$$

$$W_2 * x + b_2 = [3, 4, 5, 6] * [4, 3, 2, 1] + 3 = 40 + 3 = 43$$

$$sigmoid(W_2 * x + b_2)$$

Ahora con matrices

$$x = [3, 4, 5, 6]$$

$$W = [[3, 2, 4, 5], [4, 3, 2, 1]]$$

$$b = [2, 3]$$

$$xW^T + b$$

$$sigmoid(xW^T + b)$$

$$\begin{pmatrix} 3 & 4 & 5 & 6 \end{pmatrix} * \begin{pmatrix} 3 & 4 \\ 2 & 3 \\ 4 & 2 \\ 5 & 1 \end{pmatrix} + \begin{pmatrix} 2 & 3 \end{pmatrix}$$

Solución

$$\begin{pmatrix} 69 & 43 \end{pmatrix}$$

Y si se hace lo mismo para las X. Las convertimos en una matriz para computarlas todas. Lo que llevaria a:

- X: Representa una matriz de inputs. Permitiendo computar múltiples samples a la vez.
- W: Matriz de pesos (feature, regresion). Una fila por cada entrada (feature) y una columna por cada regresión.
- b: Contiene un bias por cada regresión.

Nuevos conceptos de redes neuronales.

- Capa de entrada: es la primera entrada a la red neuronal. En el caso del ejemplo del MNIST serian los píxeles.
- Capa intermedia: Serian las salidas de las diferentes capas antes de la salida final.
- Capa de salida: es la salida de toda la red neuronal, sus entradas son las salidas de las capas intermedias.

Esto nos permite enlazar funciones no lineales con otras funciones no lineales, lo que lleva a resolver problemas mas complejos dado a la no linealidad. Permite hallar funciones mas complejas y que sean lineales, hiper-planos.

Esto nos llevaria a tener una matriz de peso y un vector de bias por cada capa de la red.

Caracteristicas de las redes neuronales:

- No linealidad nos permite atacar problemas comlpejos.
- Compuesta por capas (Hiper parámetros)
- Importante que cada capa sea diferente.
- Si se puede derivar, se puede optimizar.
- Y en cada capa hay neuronas.

La cantidad de capas que va a tener el modelo se escogen de acuerdo con el problema y las capacidades de hardware que se tenga. Solo la capa de salida es fija, las cantidad de capas intermedias se escogen por experimentación, al igual que la cantidad de neuronas por capa. Cada una de las capaz tiene que ser derivable, porque se pueden optimizar con el descenso del gradiente.

Para definir los pesos y los bias para las neuronas se tiene que realizar la derivada de la función de perdida por cada uno de los pesos y bias de las neuronas. No es la misma función de perdida que en la regresión logística, dado que esa es binaria. La nueva función de perdida es softmax. Se utiliza la técnica de Back Propagation donde se van ajustando los pesos y bias de atrás para adelante, o mejor dicho de la capa de salida a la capa de entrada.

Se usa mayormente para clasificar data, por ejemplo clasificar imágenes. Nos da un porcentaje de que tan probable es que pertenezca a cada una de las clases, como son probabilidades su suma da 1.

B. Perceptron

- Inventado por Frank Rosenblatt.
- Es una regresión logística, con otro loss function el Hinge Loss.
- Se penso que ina a resolver muchos problemas. Pero Minsky y otros señalaron algunos problemas
  - Requerimientos computacionales de la época, estos no eran lo suficientemente avanzados.
  - No puede resolver un XOR

  Estos problemas llevaron al invierno de la AI donde por varios años no se avanzo en ese tema.

1) Biografía de Frank Rosenblatt: Fue un profesor asociado de neurobiologia en la Universidad de Cornell, Nueva York. Es el creador de la teoría del perceptron, desarrollo una maquina experimental que podía entrenarse para identificar automáticamente objetos y patrones, como las letras del alfabeto. Es te concepto lo desarrollo en el libro *"Principles of Neurodynamics"*.

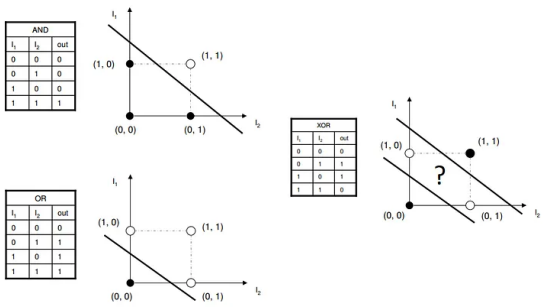

Fig. 3. Ejemplo del problema del XOR

2) Problemas del XOR: Como se muestra en la figura 3 en el caso del and y or con un solo perceptron se puede separar linealmente las clases y asi poder clasificarlas. Pero en el caso del XOR no se puede realizar porque se ocupa una separación no lineal de las clases.

Esto lleva a que no pueda ser modelado por una regresión logística o perceptron. Pero si conectamos una regresión logística con otras estas ya no se comportan linealmente. Esto llevo a la creación de las Redes Neuronales o perceptrones multicapa.

3) Función de activación:

- En regresión logística es llamado función no-lineal (Sigmoid)
- Depende de la señal activa o no la neurona
- Deja pasar, bloquear o transformar la información
- Existen varias funciones de activación

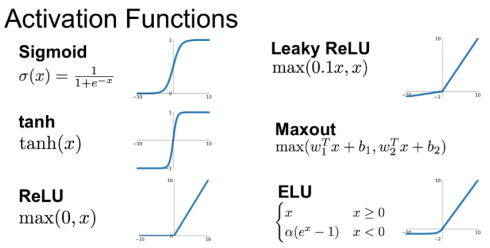

Fig. 4. Ejemplo de funciones de activación

En la figura 4 se muestran varias de las posibles funciones de activación. Algunas características de estas funciones:

- ReLU no es derivable al igual que Leaky ReLU.
- En caso de no es derivable se usa un método llamado work around.
- En las capas intermedias se utiliza mas ReLU y Leaky ReLU.
- Se sufre del problema del **Desvanecimiento del gradiente**.

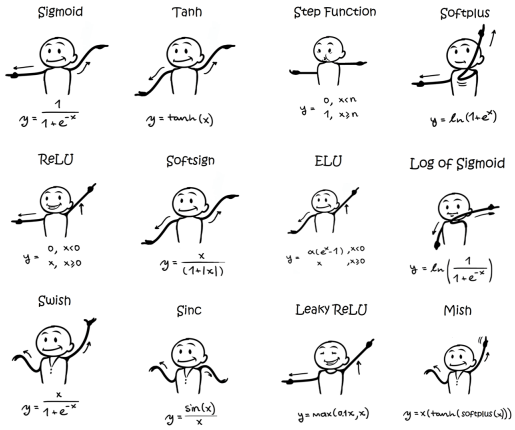

Fig. 5. Ejemplo de funciones de activación cartoon

4) Perceptrón Multicapa (MLP): La notacion puede variar dependiendo de la lectura.

- $W = \theta$
- $\theta = W, b$
- XW
- WX

Para calcular la pasada hacia adelante, de la capa de entrada a la capa de salida. ???

El $h(0)$ es la entrada de la siguiente capa, se multiplica por el siguiente W y se suma el correspondiente b.

$h(0) = sigmoid(XW^0 + b^0))$
$h(1) = sigmoid(h(0)W^1 + b^1))$
$h(2) = sigmoid(h(1)W^2 + b^2))$
$h(n) = sigmoid(h(n-1)W^n + b^n))$

No siempre se usa la sigmoid, cual función de activación usar depende del problema. Para la función de costo usamos binary cross-entropy o cross entropy. Se tienen que actualizar todos los pesos. La derivada con respecto a W se tiene la capa j y la neurona i.

$$\frac{\partial L}{\partial W_i^j}$$

- Se escoge la cantidad de capas y neuronas dependiendo del tiempo y poder computacional.
- Cada capa aprende de lo mas general a lo mas especifico. Ejemplo: supongamos que se quiere reconocer caras. Por lo que la capa de entrada solo reconocería las lineas de la cara, la siguiente capa intermedia aprendería a reconocer los ojos, y las ultimas capas reconocerían mejor las caras con sus detalles específicos.

5) Fine Tuning: Es una técnica de entrenamiento de redes neuronales la cual consiste en reutilizar arquitectura de redes CNN predefinidas y preentrenadas.

Esta tecnica consiste en realiza un ”ajuste fino” de algunas capas de la red para obtener la salida deseada.

¿Por que usar el Fine Tuning?

- Permite ahorrar tiempo.
- Permite no depender de recursos de computación. Dado que la mayoría del modelo ya esta entrenado, lo que faltaría son los ”ajustes finos” solamente.

Por ejemplo, un modelo entrenada para reconocer coches puede servir de punto de partida para un nuevo modelo de reconocimiento de vehículos pesados.

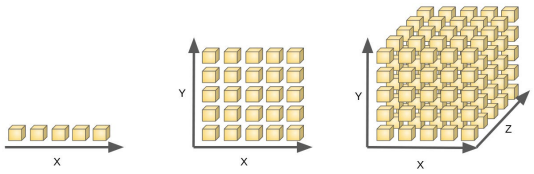

Fig. 6. Maldición de la dimensionalidad

6) Maldición de la dimensionalidad: Como se nota en la figura anterior entre mas dimensiones mas opciones se tienen que recorrer, y por solo agregar una dimensión crece exponencialmente. Entre mas dimensiones mas difícil es hallar el punto óptimo, las redes neuronales resuelven este problema aun que tienen un limite.

- A mayor cantidad de dimensiones, aumenta la complejidad.
- Aumenta la computabilidad.
- Más difícil encontrar patrones.

7) Comportamiento Jerárquico: Al igual que los humanos las redes neuronales aprenden de los mas simple a lo mas complejo, características mas generales y luego las especificas.

Ganancia exponencial en algunas funciones:

- Polinomios.
- Composición de funciones que permite reusar funciones simples para otras de orden superior.
- Representación compacta: Pocos pesos se pueden modelar funciones complejas.

En la figura 7 se representa como se comporta el 3 en cada una de las capas de la red neuronal. Se va reduciendo la dimensionalidad hasta solo ser de 4.

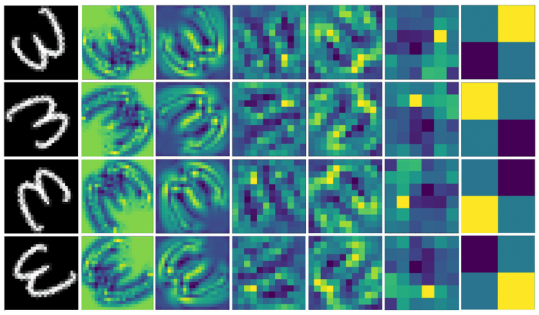

Fig. 7. MNIST Feature Map Representation

Existen bases de datos dedicadas a almacenar vectores solamente. Se hace el modelo se entrena y se guardan los vectores en bases de datos dedicadas. También existen chips dedicadas a vectores como los de Nvidia.

- Permiten representar el mundo de forma eficiente.
- Composición basada en jerarquía de representaciones aprendidas no definidas.

C. Notebook Redes Neuronales Pytorch

Ventajas de PyTorch

- Están todos los módulos listos para utilizar
- Se pueden hacer cosas muy custom.
- Las derivadas se van calculando como van pasando y al final optimiza.
- Tiene múltiples optimizadores.

En este notebook el profesor realiza un ejemplo en PyTorch de una red neuronal. Lo interesante es que se define como se realiza el forward (la pasada hacia adelante).

Batch_size: de todo el data set cada cuantos samples se pasan a la red juntos para entrenar.

Para utilizar PyTorch se usan tensores. PyTorch con transforms permite transformar las imágenes (rotarlas, invertirlas) hasta permite crear transformaciones propias.

El DataLoader retorna un enumerate que permite iterar sobre los datos.

A los modelos se les puede decir si están en etapa de evaluación o etapa de entrenamiento, esto porque la etapa de evaluación no hace cálculos de gradientes. Al entrenar el modelo después de hacer el calculo del lost al optimizador se le indica que los gradientes estén en 0, se le indica que va a hacer back propagation y luego se le indica que actualice los pesos de cada uno.

PyTorch permite guardas los pesos del modelo. Se recomienda estar haciendo guardados cada ciertos epochs, dado que google colab solo permite ciertos minutos corriendo. Por lo que se tendría que abrir un nuevo colab y cargar los datos ya entrenados.

```python
class FullyConnectedNN(nn.Module):
    def __init__(self, input_size):
        super(FullyConnectedNN, self).__init__()
        #Defino las capas que voy a utilizar
        #en mi modulo.
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(input_size[0]*
            input_size[1], 128)
        # Input size:28*28, Output Size:128
        self.relu = nn.ReLU()
        # Activation Function!!!
        self.fc2 = nn.Linear(self.fc1.
            out_features, 64)
        #Input Size:128, Output Size:64
        self.fc3 = nn.Linear(self.fc2.
            out_features, 10)
        #Input size:64, Output Size:10
        #(Clases de MNIST)

    def forward(self, x):
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        return x
```

Listing 1: Código del modelo de la red neuronal

```python
# Save model state
torch.save({
    'epoch': epoch,
    'model_state_dict': model.state_dict(),
    'optimizer_state_dict': optimizer.state_dict(),
    'loss': loss,
}, 'model_checkpoint.pth')

# Load saved model parameters
checkpoint = torch.load('model_checkpoint.pth')
model.load_state_dict
    (checkpoint['model_state_dict'])
optimizer.load_state_dict
    (checkpoint['optimizer_state_dict'])
epoch = checkpoint['epoch']
loss = checkpoint['loss']
```

Listing 2: Código de save y load del modelo

REFERENCES

[1] " ' Special to The New York Tmes, "Dr. Frank Rosenblatt Dies at 43; , Taught Neurobiology at Cornell," New York Times Company, New York, N.Y, p. 36, Jul. 13, 1971.

[2] A. Tech, ¿Cómo funciona softmax? Youtube, 2018.

[3] B. T. Moises, "¿Qué es Fine -Tuning y cómo funciona en las redes neuronales?," 2023. https://blog.pangeanic.com/es/que-es-fine-tuning (accessed 2024).