Benjamin Gutierrez Garcia bencouver@gmail.com

CIFAR-100 Image Classification

 [1]

Introduction

Este ejercicio consiste en clasificar las 100 imágenes del dataset CIFAR-100 en sus 100 categorías y proporcionar como métrica del rendimiento el parámetro *multiclass classification accuracy, o también conocido como Categorical Cross Entropy.* Esta es una medida de el error en la probabilidad de clasificación en las que las clases son mutuamente exclusivas, es decir, es decir, cada imagen pertenece a una sola clase, y tiene una sola etiqueta, puede ser un perro o un camión pero no ambos.

[1] From https://www.cs.toronto.edu/~kriz/cifar.html

El CIFAR-100 está dividido en 100 clases, las cuales a su vez están agrupadas en 20 superclases. Cada imagen tiene dos etiquetas, una de la clase y otra de la superclase. Dado que por cada clase existen solo 500 imágenes (de baja calidad) para entrenamiento y 100 para verificación, la clasificación de este conjunto de datos es relativamente difícil.

Solución Implementada

En este documento describo el código propuesto como una posible solución. Este código está escrito en python 3 y utiliza principalmente la biblioteca TFLearn[2]. Esta última está escrita "encima" de Tensorflow y proporciona una nivel de abstracción mayor, es decir, sus funciones facilitan muchas operaciones y permiten escribir menos código y con mayor facilidad. La abstracción está basada en capas (o layers) de TFLearn, ya sea de convolución o de máxima agrupación (max pooling) , las cuales permiten construir redes de forma sencilla, llevar a cabo operaciones sobre ellas, y se aísla al usuario de manejar manualmente parámetros como bias o pesos. Tambien es posible visualizar la ejecucion del entrenamiento usando Tensorboard.

Requisitos

Este código requiere los siguientes paquetes: python3, pip3, numpy, urlib.request, Tensorflow, Tensorboard y TFLearn.

Ingesta de Datos

Se incluyen funciones que toman el dataset del sitio de web de referencia en la Universidad de Toronto y se descomprime localmente en el directorio de trabajo. Dado que el formato en el que viene es binario (python pickled), una función lo decodifica a caracteres normales (un pickle). Los datos de verificación y entrenamiento son leídos en forma separada a la memoria. Los segundos constan de 50,000 imagenes con 32x32x3 pixeles (3 colores), que se ingestan como una matriz de 50,000x3072, con una imagen por renglón. Este formato no es adecuado para la clasificacion asi que cada imagen se convierte en un 4-tensor . También el arreglo de etiquetas se debe convertir a un formato adecuado, lo cuale s tipico de datos categóricos (número de registros, ancho, largo, profundidad). En los comentarios del código se ofrecen más detalles.

PreProcesamiento de Datos

Este comienza con un escalamiento de los valores de las características de los datos (Feature values) usando funciones de TFLearn. Esto se hace para evitar que existan valores muy

[2] http://tflearn.org/tutorials/quickstart.html

grandes que generen cierto sesgo de los gradientes que son retro-propagados en la red.Lo adecuado es que estén más o menos a la misma escala.

Después se realiza un proceso llamado expansión de los datos (Data augmentation)[3]. Consiste en generar mas imagenes de entrenamiento mediante reflexiones y rotaciones al azar de las que ya tenemos. Estos e hace cuando el número de imágenes de entrenamiento es relativamente pequeño.

Modelo de Aprendizaje

La arquitectura que se implementa en este código es una red de convolución (ConvNet). Tiene tres redes de convolución con un filtro de 3x3, y muy importante utiliza como función de activación ReLU (*Rectifier Linear Unit)*. Se usan también capas de *max-pooling* (agrupamiento) de 2x2 y dos capas tipo "*fully connected*", con función de activación softmax. El algoritmo de optimización para los pesos es Adam[4], que es como un descenso de gradiente donde el ritmo de aprendizaje es adaptativo. Es muy recomendado en la documentación de TFLearn. Finalmente, se utiliza la técnica de dropout (apagar nodos de las redes al azar para mitigar *overfitting)*. Para aplicar el filtro en forma uniforme, incluyendo los bordes de la imagen, se utiliza el parámetro stride. Más detalles se pueden proporcionar si es requerido.

Parámetros de Ejecución

El código se ejecuta con el comando:

python3 cifar-100.py

Y proporciona un output del tipo:

[3] Ver por ejemplo: Benjamin Graham, Fractional Max-Pooling, Dept of Statistics, University of Warwick,, May 13, 2015, y Benjamin Graham, Spatially-sparse convolutional neural networks, Dept of Statistics, University of Warwick,, UK, September 23, 2014.
[4] https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

El ritmo de aprendizaje para ADAM (learning rate) es de 0.001. El código se ejecutó por 50 épocas, es decir, el Dataset entero paso de adelante hacia atrás 50 veces.

Este código se ejecutó en Ubuntu 16.04.4 LTS, python3.5.2. En una máquina de 8 cores (Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz) con 32 GB de RAM. 50 épocas se ejecutaron en aproximadamente dos horas.

Resultados

Para analizar los resultados utilizamos Tensorboard[5]. Este nos permite visualizar mediante un web browser logs, funciones de pérdida, métricas (Loss, metric, gradients, weights, activations, sparsity, etc).

Se requiere *tensorboard_verbose=3* en la definición de la clase del modelo de Deep neural network en el código para generar los logs requeridos por tensorboard:

```
with tf.device('cpu:0'):
    #Possible Arguments see http://tflearn.org/models/dnn/.
    #tensorboard_verbose=3 is needed for Tensorboard. Use Firefox, graphs dont show in Chrome
    model = tflearn.DNN(network, tensorboard_verbose=3)
    model.fit(X, Y, n_epoch=50, run_id='CIFAR100-BEN',shuffle=True,
              validation_set=(X_test, Y_test), show_metric=True, batch_size=100)
```

[5] https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard

Al ejecutar, Tensorboard activa un webserver local que genera la visualización para el browser:

Podemos también observar la evoluciones de los valores de la métrica durante todas las epocas:

El valor de la métrica obtenido fue de 0.4927, aproximadamente 49%.
multiclass classification accuracy = Categorical Cross Entropy.

Me gustaría notar que con Tensorboard es posible visualizar histogramas de error en todas las redes de la arquitectura:

Y se puede generar una gráfica del flujo a través de toda la arquitectura: