BEATING MNIST DATASET

Axel Lijdens - Cristian Raña - Francisco Pirra
axel.lijdens@gmail.com - cristian.rana@hotmail.com - fpirra@lsia.fi.uba.ar
Universidad de Buenos Aires
Facultad de Ingenieria

Resumen

La clasificación de imágenes automatizada, es un proceso por el cual buscamos lograr, mediante el uso de distintas herramientas, que una computadora pueda -entender- una imagen y asi poder extraer conocimiento de la misma. En este caso, estudiaremos el uso de redes neuronales convolucionales, aplicadas al set de datos MNIST, para poder clasificar los dígitos manuscritos que figuran en cada imagen del set.

Índice

I. Introducción

En un principio decidimos utilizar un conjunto de algoritmos para enfrentar el desafio de reconocimiento de digitos manuscritos en Kaggle.
Parte de la investigación fue encontrar algoritmos con buenos resultados al procesar el set por si solos, pero que además se complementen. Esto quiere decir que cada algoritmo debe ser 'fuerte' donde los demás fracasen, y de esta forma poder corregir las falencias de cada uno.
Como algoritmo principal, habiamos decidido utilizar una red neuronal, en base a lo que investigamos sobre el exito de la aplicacion de las mismas en los últimos años en areas como reconocimiento de voz, imágenes, etc; debido a que pueden relacionar de manera más eficientemente los datos. Prueba de esto, es el sistema de reconocimiento de voz desarrollado por google [1].

II. Comenzando el procesamiento

En principio la idea era, como ya dijimos, utilizar un 'conjunto' de algoritmos para procesar los datos, y luego realizar una comparacion local de los resultados, con un sistema de 'ponderacion' entre los resultados y al cruzarlos buscar encontrar los valores en los que todos los algoritmos coincidieran, y los que no, reprocesarlos.

Comenzamos probando el desempeño de una red neuronal básica en el set MNIST. Para esto realizamos una implementación ingénua y se subieron los resultados en Kaggle, obteniendo aproximadamente un 96 % de éxito (Nos resulta importante destacar que con una implementacion muy basica de una red neuronal, se logró prácticamente el mismo resultado que con mucho más trabajo se llego al usar otros algoritmos, como KNN, lo cual refuerza la idea de usar una red como algoritmo principal). Dicha red consistió en 3 capas (1 de entradas, 1 'oculta' y la última de salidas), donde todas las neuronas de una capa están conectadas a las de la siguiente. Los valores utilizados para los hiper-parámetros fueron hallados por prueba y error, ya que no hay al día de hoy una forma conocida para optimizarlos.

Teniendo en cuenta que no se realizó una refinación exhaustiva de los hiper-parámetros, ni del algoritmo de entrenamiento, los resultados fueron bastante alentadores.

Por otro lado, realizamos diversas pruebas con KNN, variando algunos de sus parametros:

- Valor de k (cantidad de vecinos cercanos)
- Algoritmo (ball tree, kd tree)
- Métrica (minkowski, euclideana, manhattan, mahalanobis)

Además de la variación de parametros, intentamos ver como podiamos preprocesar los datos de entrada para obtener mejores resultados, por lo que generamos una imagen que sea el promedio de las demas para cada clase y en base a esta imagen 'promedio' usarla para clasificar (Lo que nos dio un 80 % de exito en kaggle). No es un dato menor, ya que en vez de comparar contra todo el set, solo comparo con 10 imagenes 'promedios', y 80 % no nos pareció un mal resultado para esta prueba. Por lo que concluimos que los mejores resultados para KNN provenían de utilizar k=3 y norma cuadrada, con casi un 97 %.

Por último, aplicamos el algoritmo de SVM, aplicando primero una reduccion de dimensiones con PCA, y logramos un resultado cercano al 98.5 %

Luego de estas pruebas, podemos ver los resultados obtenidos en kaggle hasta el momento:

Algoritmo	Detalles y parámetros	Acierto
Red Neuronal	784-100-10 (neuronas por capa), gamma=2.85, 17 imágenes por iteración	0,988 %
PCA + SVM	components=0.8; whiten=True; kernel='rbf; C=2	0,984 %
KNN	K=4, reduciendo set a 14x14 (prom. pix), euclideana	0,975 %
KNN	K=4; P=6; Distancia Minkowski	0,974 %
KNN	K=1; Euclideana (quitando cols de 0s)	0,971 %
KNN	K=3; Euclideana, Ball Tree	0,968 %
KNN	K=9; Euclideana	0,967 %
KNN	K=9; Manhattan	0,959 %

Cuadro 1: *Resultados*

Como podemos observar, sin mucho esfuerzo la red neuronal le estaba ganando a los demas algoritmos, en base a esto fue que empezamos a pensar que, quizá el éxito podria encontrarse en el uso de las redes únicamente, entonces decidimos empezar a buscar 'afinar' los parámetros de la red, para ver como podiamos mejorar los resultados.

III. Cambio de planes

Al investigar más sobre las redes, nos topamos con las redes convolucionales[8]! Las mismas vienen dando muy buenos resultados en el área de la visión artificial, por lo que decidimos probarlas, de modo que con la idea anterior del 'pack' de algoritmos casi descartada, nos pusimos a trabajar de lleno, con las redes convolucionales.

Tomamos la decisión de dividir el set de datos en 3 partes:

- Set de entrenamiento (80 %)
- Set de prueba (10 %)
- Set de validación (10 %)

El set de entrenamiento lo usamos para entrenar la red (ajustar los pesos).
El set de prueba lo usamos para ajustar los hiper-parámetros (cantidad de capas en la red, cantidad de neuronas por capa, el valor de la tasa de aprendizaje, etc).

Por ultimo, para evitar el overfitting, realizamos pruebas sobre el tercer set, el cual nos indica el progreso en la clasificación de las imágenes realizado por la red en cada ciclo de entrenamiento. Fue importante ir viendo el desempeño de la red a medida que pasaban las epocas, para ver en que momento el aprendizaje se empieza a planchar, y alli cortar el entrenamiento [4] para no entrar en overfitting

IV. Estructura de la red

Las redes convolucionales son redes inspiradas en la corteza visual de los animales, en las que cada capa no está totalmente conectada con la otra, sino que se conectan pequeñas secciones.

De esta forma la red puede aprender patrones, ya que analiza una entrada (pixel) en su contexto (pixeles aledaños). Las capas siguientes (no necesariamente deben ser convolucionales) pueden relacionar estos patrones para identificar la imágen. A forma de ilustración, se puede tomar el siguiente ejemplo: La primera capa reconoce curvas, la siguiente reconoce ojos, boca, etc a partir de las curvas, y la siguiente decide en base a los reconocimientos de la capa anterior si es o no una cara (¿tiene 2 ojos?, ¿están a los costados de la nariz?, etc). La idea es que cada capa pueda canalizar distintos conceptos [9].

Para hacer que la red 'aprenda', vamos a usar una función que tome la salida de la red y la convierta en un escalar que tienda a cero cuanto mejor sea la respuesta de la red.
El objetivo es minimizar la funcion (encontrar los pesos y bias que la hacen tender a cero). La idea del uso de esta función es construir un campo escalar para poder utilizar un método numérico y de esta forma encontrar el mínimo, que se llama 'Gradient descent'[5]
Éste método consiste en tomar la dirección del gradiente en el punto que estamos analizando, cambiarle el signo (para estar apuntando en la direccion de máximo decrecimiento), multiplicarlo por un valor (gamma) y sumarle la posición actual. De esta forma, lo que estamos logrando es ir en contra del gradiente 'de a pasos'.
El problema que surge de ésto es calcular el gradiente, porque tenemos demasiadas variables (pesos y bias), entonces para solucionar esto, vamos a utilizar el metodo de backpropagation[6]. Básicamente son unas ecuaciones que nos dicen como, a partir de los valores de salida obtenidos, podemos 'ir para atrás', calculando un delta (pequeña variación), y de esta forma aproximar el gradiente. Este proceso se repite hasta obtener resultados 'decentes' con la red.
Por último, utilizamos el método 'stochastic gradient descent'[7], pero en lugar de hacerlo con una imagen a la vez, lo vamos a promediar entre un K número de imagenes, y así conseguir el gradiente.

Para evitar el overfitting decidimos incrementar el set de entrenamiento. Esto puede resultar dificil, costoso sino imposible. La forma en que lo conseguimos fue agregando ruido y/o distorsionando los datos de manera controlada, para poder incrementar el set de entrenamiento[2].

Otra forma de evitar overfitting en las redes neuronales es mediante una técnica llamada dropout[3], que consiste en entrenar solamente la mitad de las neuronas en cada ciclo, seleccionandolas aleatoriamente.

Por último, armamos un algoritmo que va guardando el estado de la red, a medida que los resultados mejoran en las pruebas locales, ademas de ir generando un log, para poder clasificar sin tener que entrenar cada vez, y tambien para poder hacer entrenamientos, sin necisidad de ser continuamente entrenada.

V. Pruebas y submits

Para buscar los parametros y la estructura que mejor haga funcionar a la red, fuimos realizando pequeñas variaciones y subiendo a kaggle los resultados obtenidos en cada caso.
A continuación podemos observar algunos de los resultados para configuraciones tomada:

Detalles y parámetros	Acierto
2 convPool, 2 FullyConnected(800-600n), 1 Softmax, minib=14, noise to trainset	0,995 %
2 convPool, 3 FullyConnected(1200n), 1 Softmax, minib=14	0,99486 %
2 convPool, 3 FullyConnected(1200-1500-1800n), 1 Softmax, minib=20	0,99429 %
2 convPool, 2 FullyConnected(1200n), 1 Softmax, minib=12, learnrate=0.01	0,99386 %
2 convPool, 3 FullyConnected(1200n), 1 Softmax, minib=12, dropout=0.4	0,99357 %
2 convPool, 3 FullyConnected(2000n), 1 Softmax, minib=12, adding more rotations to set	0,99386 %
2 convPool, 3 FullyConnected(1200n), 1 Softmax, minib=14	0,99371 %
2 convPool, 1 FullyConnected(600n), 1 Softmax, minib=10	0,99357 %
2 convPool, 3 FullyConnected(2000n), 1 Softmax, minib=14, (0.3 and 0.6 skew, 10 rotate)	0,99329 %
2 convPool, 3 FullyConnected(1200n), 1 Softmax, minib=12, (0.3 and 0.6 skew, 10 rotate), learn=0.01	0,99314 %

Cuadro 2: *Resultados en kaggle de las redes convolucionales*

Por lo que finalmente, la estructura de la red, que nos llevo al exito, fue conformada por:

Capas y parametros
ConvPoolLayer: filter shape=(70, 1, 7, 7), image shape=(12, 1, 28, 28), pool size=(2, 2) ConvPoolLayer: filter shape=(40, 70, 4, 4), image shape=(12, 70, 11, 11), pool size=(2, 2) FullyConnectedLayer: in=640, out=1000, dropout=0.5 FullyConnectedLayer: in=1000, out=1200, dropout=0.5 FullyConnectedLayer: in=1200, out=1000, dropout=0.5 SoftmaxLayer: in=1000, out=10, dropout=0.5
Parametros de aprendizaje: max_epochs=60, mini_batch_size=12, learn_rate=0.03, stepts_without_improve=8

Cuadro 3: *Parametros finales de la red*

VI. Conclusiones

Como primera conclusion, podemos decir que el uso de redes convolucionales fué un gran acierto. Por otro lado, vimos que fue importante extender el set de datos, ésto nos ayudo a mejorar aún mas los resultados obtenidos

Entrenar la red puede llevar algunas horas, pero una vez entrenada, la velocidad de procesamiento es sorprendente, por lo que puede ser muy util a la hora de clasificar imagenes, y hasta videos. Dado a que es tan veloz el procesamiento, estamos bastante convencidos que puede funcionar bien en videos.

Tambien pudimos notar que el optimo de neuronas ronda los 800 a 1200... si seguimos agregando mas no mejora el funcionamiento y ademas genera un costo mayor de procesamiento, asique mas neuronas no necesariamente es mejor resultados. Asi como tambien el valor del minibatch,

notamos que el optimo ronda entre 12 y 14. En base a las imagenes que pudimos ver que fueron 'mal clasificadas' creemos que lograr el 100 % de precisión es -casi- imposible, dado que hay imágenes que incluso a una persona le costaría decir de que dígito se trata, o hasta se puede entrar en discución si es uno u otro. Dejamos algunas de estas imágenes, para la opinión del lector a continuación...

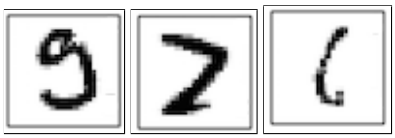

Figura 1: *Puede reconocer los digitos?*

VII. Bibliografía

Referencias

[1] http://googleresearch.blogspot.com.ar/2015/08/the-neural-networks-behind-google-voice.html

[2] http://research.microsoft.com/pubs/68920/icdar03.pdf

[3] 'Improving deep neural networks for LVCSR using rectified linear units and dropout', George E. Dahl, Tara N. Sainath† Geoffrey E. Hinton

[4] https://en.wikipedia.org/wiki/Early_stopping

[5] 'Gradient-Based Learning Applied to Document', Y. LeCun, L. Bottou, Y. Bengio and P. Haffner

[6] 'Neural Networks and Back Propagation Algorithm', Mirza Cilimkovic

[7] 'Large-Scale Machine Learning with Stochastic Gradient Descent', León Bottou

[8] https://en.wikipedia.org/wiki/Convolutional_neural_network

[9] https://en.wikipedia.org/wiki/Deep_learning#Convolutional_neural_networks