Reconocedor de Dígitos con Redes Neuronales

NICOLÁS ZORZANO

FACULTAD DE INGENIERÍA DE LA UBA - REDES NEURONALES

9 de febrero de 2018

Resumen

El presente trabajo consiste en el desarrollo de un sistema reconocedor de audio capaz de clasificar dígitos en ingles del cero al nueve, haciendo uso de redes neuronales. Se utilizó para tal fin **TFLearn**, una API de TensorFlow que permite acelerar los tiempos de cómputo y por lo tanto mejorar la elección de hiperparámetros.

1. Reconocimiento de habla

La generación de las señales de habla es un mecanismo complejo que se efectúa en la laringe, donde se encuentran los pliegues vocales que regulan el paso del aire mediante el cierre y abertura de la glotis. La señal de habla resulta en una señal no estacionaria, que no solo depende de las palabras pronunciadas, sino también del tracto vocal de la persona parlante, de la duración, del tono, y demás factores que hacen difícil la tarea de encontrar un denominador común entre distintas pronunciaciones de una misma palabra. Por otra parte, las señales de habla pueden ser pensadas como una combinación de unidades básicas conocidas como **fonemas**, los cuales son la articulación mínima de un sonido vocálico[1] y consonántico[2].

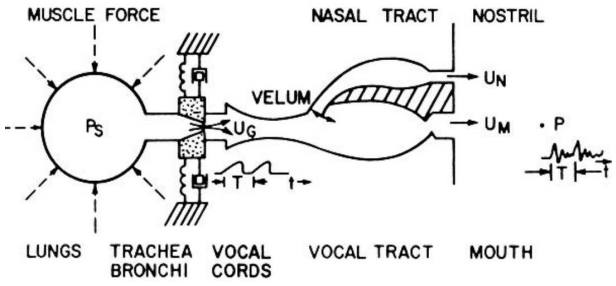

Figura 1.1: Representación del tracto vocal.

De todo esto, resulta que la señal de habla es una señal no estacionaria que puede ser modelada mediante filtros no lineales. Sin embargo, utilizando ventanas de aproximadamente 10 ms o 20 ms, es posible aproximar variaciones mínimas y así utilizar filtros lineales de la forma:

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 - \sum_{k=1}^{N} a_k z^{-k}}$$

Debido a que las señales de habla incluyen información irrelevante para su reconocimiento (duración de los fonemas, tono, sexo, etc.), es necesario utilizar un método de representación que permita extraer aquellas características introducidas por el tracto vocal que son dependientes de cada persona en particular. Para ésto es conveniente utilizar los **Mel Frequency Cepstral Coefficients (MFCC)**, donde no solo se utiliza el cepstrum sino que también se introduce la escala no lineal Mel que aproxima el comportamiento del sistema auditivo. La obtención de estos coeficientes se realiza de la siguiente manera:

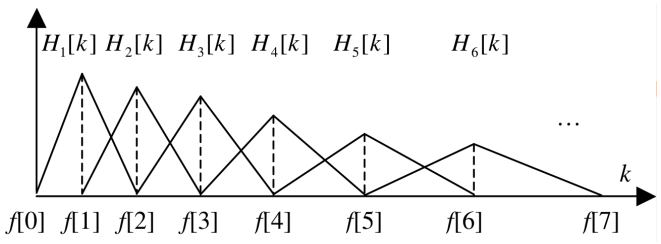

Figura 1.2: Banco de filtros en la escala Mel.

- Se ventanea la señal en distintos frames.
- A cada frame se le aplica la DFT:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi nk}{N}} \qquad 0 \leq k < N$$

- Se define el banco de filtros en la escala Mel. Estos filtros calculan el espectro promedio alrededor de cada frecuencia central, utilizando anchos de banda cada vez mayores (ver figura 1.2). Una posible definición para dichos filtros es:

$$H_m[k] = \begin{cases} 0 & k < f[m-1] \\ \frac{2(k-f[m-1])}{(f[m+1]-f[m-1])(f[m]-f[m-1])} & f[m-1] \leq k < f[m] \\ \frac{2(f[m+1]-k)}{(f[m+1]-f[m-1])(f[m]-f[m-1])} & f[m] \leq k < f[m+1] \\ 0 & k > f[m+1] \end{cases}$$

- Se calcula el logaritmo de la energía de salida de cada uno de los filtros:

$$S[m] = \ln\left(\sum_{k=0}^{N-1} |X[k]|^2 H_m[k]\right) \qquad 0 \leq m < M$$

- Finalmente se aplica la Transformada Coseno Discreta para encontrar los coeficientes:

$$c[n] = \sum_{m=0}^{M-1} S[m] \cos\left(\frac{\pi n}{M}\left(m + \frac{1}{2}\right)\right)$$

[1] Al pronunciar los fonemas vocálicos el aire no encuentra ningún tipo de obstáculo en su salida hacia el exterior. Estos fonemas son: /a/, /e/, /i/, /o/ y /u/.

[2] En la articulación de los fonemas consonánticos, se ponen distintos obstáculos al aire para salir por la boca, ya sea con los labios,la lengua, los dientes, etc.

En general se suelen utilizar 13 coeficientes para el reconocimiento y entre 20 y 40 filtros para el banco de filtros. En la figura 1.3 se tiene una representación de los datos obtenidos de cada una de las señales de audio.

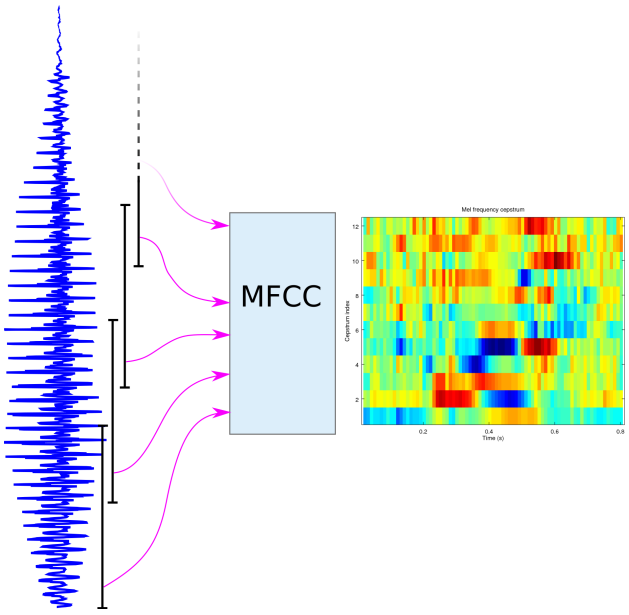

Figura 1.3: Procesamiento de las señales de habla para la obtención de los MFCC.

2. Redes Neuronales

Debido a la aparición en los últimos años de procesadores, placas de video, etc. con velocidades y precisión de cómputo muy superiores a los disponibles en el pasado, en el último tiempo las redes neuronales se han convertido en una herramienta cada vez más difundida y poderosa. Por otra parte, han surgido empresas y organizaciones que continuamente se encargan de crear y mantener bases de datos de información (imágenes, videos, textos, etc.) que hacen posible la realización de entrenamientos para una amplia gama de casos de aplicación.

Para este trabajo, se utilizaron dos modelos diferentes de redes, los cuales serán presentados a continuación.

2.1. Perceptrón multicapa

Un perceptrón es un modelo matemático que intenta aproximar el comportamiento de una neurona. Presenta una serie de entradas $\{x_1, x_2, \ldots, x_n\}$ a las cuales se les asocia un determinado peso w_i. A la salida del perceptrón se le resta un bias o threshold b y finalmente se aplica una determinada función de activación $\sigma(\cdot)$ de la siguiente manera:

$$s = \sigma\left(\sum_i w_i x_i - b\right)$$

El proceso de aprendizaje de un perceptrón entonces consiste en encontrar aquellos pesos w_i y el bias b que logran que para cada combinación de entrada $\{x_1, x_2, \ldots, x_n\}$ se tenga la salida deseada.

Creando capas sucesivas de perceptrones, como se puede ver en la figura 2.1, se obtiene un **perceptrón multicapa**, donde a medida que se avanza en la cantidad de capas se obtienen niveles de abstracción superiores y es posible representar funciones cada vez más complejas.

Durante el entrenamiento, se debe definir un criterio para la actualización de los pesos y el bias que permita obtener los resultados deseados. Para esto, se define una función de **costo** con la que se mide el desempeño[3], realizando una comparación entre la salida de la red obtenida y la deseada. El objetivo entonces es modificar de manera iterativa los pesos para poder minimizar la función de costo. El criterio utilizado generalmente para la minimización de la función de costo, es mediante **gradiente descendente**, donde los parámetros son modificados en la dirección en que el costo alcanza un mínimo[4]:

$$\begin{cases} w'_k = w_k - \eta \frac{\partial C}{\partial w_k} \\ b'_k = b_k - \eta \frac{\partial C}{\partial b_k} \end{cases} \quad \rightarrow \eta: \text{ Learning rate}$$

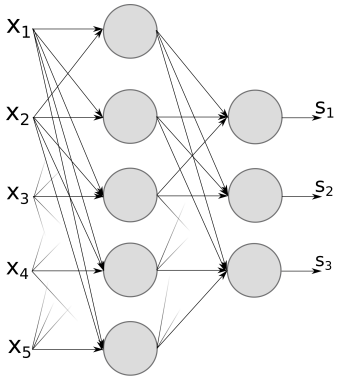

Figura 2.1: Diagrama del perceptrón multicapa.

Dado que la función de costo se encuentra definida a partir de los resultados obtenidos a la salida final de la red, se debe definir un método que permita relacionar la salida total con las salidas parciales de las capas intermedias. Esto se realiza utilizando el algoritmo de **backpropagation**, donde las actualizaciones de los pesos y el bias se propagan desde el final hasta la entrada de la red. En base a esto, se define un **epoch** de entrenamiento como el proceso de cálculo desde que se ingresa una entrada y se obtiene una salida, y se propagan desde el final hasta el inicio las actualizaciones correspondientes de los parámetros.

Para este trabajo, las muestras de entrada son matrices que en un eje tienen los coeficientes MFCC y en otro el tiempo. Para poder utilizar esta red, entonces fue necesario vectorizar las entradas, concatenando los coeficientes MFCC para crear un único vector. Por otra parte, dado que el tamaño de entrada de la red es fijo, fue necesario también rellenar las muestras de audio para que todas sean de la misma longitud. Finalmente, la salida de la red está

[3] Un caso particular puede ser el error cuadrático medio $C(w,b) = \frac{1}{2n}\sum_x ||s - y_d||$.

[4] Este método tiene problemas con los mínimos locales, por lo que se suelen utilizar variantes como el gradiente descendente estocástico.

compuesta por 10 neuronas clasificadoras, donde aquella que se active con mayor intensidad indicará el dígito ingresado.

2.2. Red convolucional

El perceptrón multicapa tiene la propiedad de que todas las salidas de una capa se encuentran conectadas con todas las entradas de la capa siguiente. Debido a esto, no se tiene información espacial sobre las entradas, es decir que dos valores de entrada que se encuentran alejados en el vector de entrada, son tratados de igual manera, lo cual no siempre es deseable cuando por ejemplo se quieren procesar imágenes y cada entrada es un pixel de la imagen.

Las redes convolucionales tienen una estructura especial que permite la introducción de dichas propiedades espaciales, lo cual las hace ideales para el procesamiento de imágenes. Están basadas en tres ideas básicas:

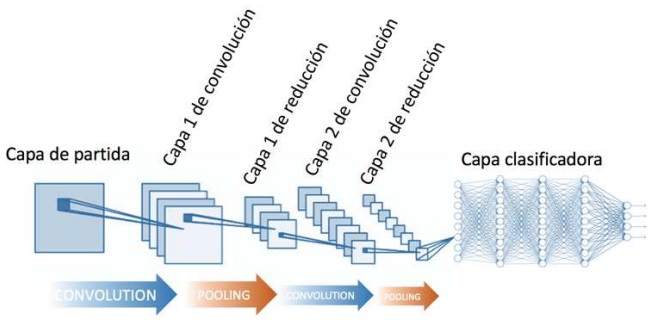

Figura 2.2: Ejemplo de red convolucional.

- **Local receptive fields:** En lugar de conectar todas las entradas con todas las neuronas, cada neurona de la capa de entrada se encuentra conectada con un grupo de valores del vector de entrada, lo cual determina las regiones conocidas como local receptive fields. Se puede pensar como que cada neurona aprende a analizar una determinada región del vector de entrada, y que con cada neurona se realiza un barrido sobre todo el vector.

- **Pesos y bias compartidos:** Todas las neuronas de una misma capa tienen los mismos pesos y bias. Esto significa que cada neurona se activa con el mismo feature, pero que se encuentra ubicado en distintos lugares del vector de entrada. Esto por ejemplo permite encontrar en imágenes patrones que pueden estar ubicados en distintos lugares, rotados, etc. Los pesos y bias compartidos se los suele llamar **kernels**, mientras que las capas extractoras de features se las conoce como **feature maps**. Una ventaja de este tipo de redes es que presentan menos parámetros que el perceptrón multicapa y son más resistentes al overfitting.

- **Pooling layers:** Son capas de agrupamiento de información. Estas capas toman la salida de un feature map y genera un feature map simplificado con su información comprimida, y así para todos los feature maps. Algunos ejemplos son:

  - **Max-pooling:** Cada neurona se encuentra conectada a una determinada región de la entrada, y la salida de cada neurona es el máximo de cada una de las regiones. Esta capa permite saber si un determinado feature está presente o no, sin interesar demasiado su ubicación exacta.
  - **L2 pooling:** En lugar de usar como salida el máximo de cada región, la salida es la raíz cuadrada de la suma de los cuadrados de las salidas de la región.

A la salida de las capas convolucionales se suele agregar perceptrones multicapa para ajustar los features encontrados a la salida deseada.

Debido a la capacidad que tienen este tipo de redes para encontrar features en imágenes, y dado que los datos de entrada son matrices, las redes convolucionales resultan una gran opción para la resolución de este problema. De esta forma, lo que se hizo fue utilizar las matrices de MFCC como entrada a la red convolucional, y luego se utilizó un perceptrón multicapa con 10 capas de salida para clasificar los dígitos de la misma manera que para la red anterior.

3. Resultados

A continuación se presentan los resultados obtenidos para los distintos tipos de redes. Una observación muy importante, fue que los incrementos más importantes en la precisión del sistema clasificador no se dieron ajustando los parámetros de las redes, sino que se dieron luego de acondicionar mejor las señales de audio. Las pruebas realizadas a lo largo del trabajo fueron:

- Inicialmente se realizaron pruebas con las señales de audio sin procesar, utilizando para el entrenamiento los audios de manera directa. Los resultados obtenidos fueron similares a elegir de manera aleatoria la clasificación, es decir una precisión cercana al 10 %.

- Luego se calcularon las matrices de MFCC para realizar la clasificación. Los resultados obtenidos fueron más cercanos a los obtenidos finalmente.

- El siguiente paso consistió en eliminar los silencios de las muestras de audio. Para esto se utilizó la herramienta `sox` de Unix la cual permite especificar la duración mínima a eliminar y la potencia mínima para la cual se considera silencio. Al realizar esto se alcanzaron precisiones superiores al 80 % para ambos tipos de redes.
  El problema que introducen los silencios es el siguiente: Los silencios son un feature presente en todos los audios, siendo para algunas palabras un rasgo más distintivo que en otras. Por lo tanto, se corre el riesgo que la red neuronal tome la duración de los silencios como feature predominante para la clasificación, lo cual hace que la clasificación se vea afectada por el procedimiento de muestreo o de las diferentes pronunciaciones.

- Se utilizó un script de Octave para normalizar todos los audios, de manera tal de que las intensidades no influyeran

en el aprendizaje. Realizando esto se alcanzaron precisiones cercanas al 90 %, siendo las de las redes convolucionales superiores a las del perceptrón multicapa. Esto permite evitar que la cercanía al micrófono o tono de voz de las personas influyan en el reconocimiento.

Un problema importante en el desarrollo de este trabajo fue la obtención de muestras de entrenamiento. En principio se reunieron 500 muestras de manera particular grabando a familiares y amigos. Finalmente, la base de datos fue completada con 2250 muestras obtenidas de la base de datos TIDIGITS[2] proporcionada por los profesores de Procesamiento del Habla[5]. Sin embargo, ésta cantidad de datos sigue siendo pequeña para lo que suelen requerir este tipo de problemas (por ejemplo las bases de datos para reconocimiento de imágenes suelen ser de valores cercanos a las cientos de miles de muestras)[6].

La implementación de las redes, junto con las bases de datos y algoritmos de acondicionamiento, se encuentran definidas en el repositorio de Github [3].

3.1. Perceptrón multicapa

La red utilizada en este caso se encuentra definida de la siguiente manera:

- **Capa de entrada:** Cada MFCC es una matriz de tamaño `[n_mfcc,n_frames]`[7]. Para poder utilizarlas con el perceptrón multicapa, son transformadas en vectores (tensores) de tamaño `[1,n_mfcc*n_frames]`. TensorFlow permite paralelizar el entrenamiento, por lo que se ingresan a la red `batch_size` tensores por vez[8].

- **Capas ocultas:** Luego de realizar varias pruebas, se llegó a la conclusión que el mejor resultado se obtiene utilizando dos capas ocultas, de 100 y 90 neuronas respectivamente, utilizando como función de activación `tanh` y regularización `L2`. El tamaño de cada una de estas capas juega un rol crucial para la solución de los problemas de overfitting y underfitting. Si la red es demasiado pequeña, no tiene capacidad suficiente para clasificar correctamente y presenta problemas de underfitting. Por otro lado, si la red es demasiado grande, tiene la capacidad suficiente para "aprender de memoria" las muestras de entrenamiento, pero al intentar generalizar a las muestras de testeo no logra hacerlo, y por lo tanto se encuentra dentro del problema de overfitting.

- **Capa de salida:** La salida está compuesta por una capa de 10 neuronas, una por clase, donde se considera que aquella neurona que presente el máximo valor de activación es la que indica a qué clase pertenece el audio de entrada. En este caso se utilizó la función de activación `softmax`, ya que en este caso se busca dar a la salida un sentido probabilístico[9].

- **Regresión:** Se utilizó el optimizador `Adam`, el cual es muy similar al gradiente descendente estocástico pero introduce la información de los momentos de primer y segundo orden para adaptar el learning rate. Por otra parte, el costo se encuentra medido mediante `categorical_crossentropy`. Esta función permite estimar qué tan cerca se encuentra una función de distribución de la distribución real.

Los resultados obtenidos en dos predicciones diferentes se presentan en el cuadro 3.1.

	0	1	2	3	4	5	6	7	8	9	Acc.
0	26		1		1						92.8
1		24			2					2	85.7
2			27				1				96.4
3		1	2	24					1		85.7
4					26	2					92.8
5					2	26					92.8
6							28				100
7		1			1		1	25			89.2
8		1	1	1					25		89.2
9		1		1		1				25	89.2
Tot											**91.4**

(a) Se utilizó un learning rate de 0.0005 y se entrenó por 800 epochs.

	0	1	2	3	4	5	6	7	8	9	Acc.
0	26		1				1				92.8
1		22			1					5	78.5
2	1		24	1			1		1		85.7
3			1	26			1				92.8
4	2	1			25						89.2
5					2	25				1	89.2
6							27		1		96.4
7		1			1		2	23		1	82.1
8		2	1	2			1		22		78.5
9		2				1				25	89.2
Tot											**87.5**

(b) Se utilizó un learning rate de 0.00005 y se entrenó por 1600 epochs.

Cuadro 3.1: Resultados obtenidos con el perceptrón multicapa.

Se puede ver que los resultados obtenidos no son los deseados. Éste resultado era esperable, ya que la red utilizada no es las más convenientes para la resolución de este tipo de problemas (los features son matrices vectorizadas, por lo que se pierde la información temporal). Por otra parte, se puede ver que al incrementar la cantidad de epochs de entrenamiento, el reconocimiento se reduce. Esto puede provenir de un problema de overfitting, pero no pudo ser solucionado ya que al reducir el tamaño de la red, y por ende su capacidad, la precisión disminuyó.

[5] Se obtuvieron otras 1500 muestras de otras bases de datos, pero debido a su baja calidad no fueron utilizadas.

[6] Debido a esto existe un factor aleatorio que influye en el valor de la precisión obtenida, que surge por la elección aleatoria de las muestras de entrenamiento y de testeo.

[7] Se utilizaron 13 coeficientes MFCC por cada frame.

[8] Al utilizar grupos pequeños de muestras de entrenamiento, se entra en una situación de compromiso entre la capacidad de paralelización y la representatividad del batch o subgrupo de muestras.

[9] La función `softmax` suele ser utilizada como una función de distribución sobre k clases.

3.2. Red convolucional

La red fue definida como:

- **Capa de entrada:** Este tipo de redes permite utilizar como entrada matrices, por lo que se utilizaron las matrices de MFCC de manera directa.
- **Capas convolucionales:** Se utilizaron dos capas convolucionales de 10 y 6 filtros respectivamente (ambas con kernels de tamaño 5x5). Luego de cada capa convolucional se aplicó una capa de `max_pooling` con kernels de tamaño 2x2. Nuevamente se utilizó regularización `L2`, pero en este caso la función de activación fue `relu`. La ventaja de estas funciones es que su gradiente es unitario cuando se encuentran activadas, y por ende la actualización de pesos y bias no se ve afectada, beneficiando a un entrenamiento más rápido.
- **Capas ocultas:** A la salida de la red convolucional se conecta una capa de 100 perceptrones, los cuales serán los encargados de clasificar los features obtenidos en las capas convolucionales. Se utilizó `tanh` como función de activación y regularización `L2`.
- **Capa de salida:** Nuevamente se tiene una capa de 10 neuronas con función de activación `softmax` para indicar la clase más probable a la que pertenece cada audio.

En el cuadro 4.1 se tienen los resultados obtenidos con la red convolucional para dos corridas diferentes.

Como era de esperarse, los resultados obtenidos con la red convolucional fueron superiores a los del perceptrón multicapa, y se acercaron más a los resultados que se suelen tener para este tipo de redes. Por otra parte, se puede ver que al refinar el proceso de entrenamiento los resultados mejoraron, por lo que indica que la precisión puede ser aumentada simplemente incrementando el número de epochs y retocando el learning rate[10].

4. Conclusiones

Una de las observaciones más importantes que surgen de este trabajo es que tanto el tamaño como la calidad de los datos de entrenamiento son mas determinantes en los resultados obtenidos que una buena elección de hiperparámetros. Si bien en este trabajo se logró reunir mas de 2000 muestras, éstas cantidades siguen siendo menores a las disponibles por ejemplo para reconocimiento de imágenes, que es un problema que actualmente se encuentra más difundido.

Otro factor que tiene una gran influencia es el preprocesamiento de los audios. En el paper [4], por ejemplo, se logra alcanzar un reconocimiento del 99 % utilizando un perceptrón multicapa. La diferencia principal con este trabajo radica en que las matrices de MFCC fueron preprocesadas para reducir sus tamaños y codificar mejor los features. Por lo tanto, es esperable que mejorando el preprocesamiento de los audios utilizados para este trabajo, se logren tener resultados similares.

[10] El segundo entrenamiento llevó alrededor de 5 horas, por lo que se decidió no continuar incrementando la cantidad de epochs.

	0	1	2	3	4	5	6	7	8	9	Acc.
0	28										100
1		24				1				3	85.7
2			27				1				96.4
3				28							100
4					27	1					96.4
5					1	26				1	92.8
6							28				100
7		1	1	2				24			85.7
8			1						26	1	92.8
9										28	100
Tot											**95**

(a) Se utilizó un learning rate de 0.0005 y se entrenó por 120 epochs.

	0	1	2	3	4	5	6	7	8	9	Acc.
0	28										100
1		27								1	96.4
2	2		26								92.8
3				28							100
4					26	2					92.8
5					1	27					96.4
6							28				100
7								28			100
8			1	1					25	1	89.2
9				1						27	96.4
Tot											**96.4**

(b) Se utilizó un learning rate de 0.00005 y se entrenó por 600 epochs.

Cuadro 4.1: Resultados obtenidos con la red convolucional.

Referencias

[1] "Spoken Language Processing", Xuedong Huang, Alex Acero, Hsiao-Wuen Hon.

[2] TIDIGITS: `https://catalog.ldc.upenn.edu/ldc93s10`. 3

[3] `https://github.com/nicozorza/TensorFlowTest`. 3

[4] *"Urdu Spoken Digits Recognition Using Classified MFCC and Backpropgation Neural Network"*, S. M. Azam, Z.A. Mansoor, M. Shahzad Mughal, S. Mohsin COMSATS Institute of Information Technology Abbottabad, Pakistan.

4