



Trabajo Práctico 2

Aprendizaje Profundo
Alejo Ordoñez

27 de noviembre de 2024

Índice

1. Perceptrón simple	3
1.1. Capacidad del perceptrón	4
2. Perceptrón multicapa	6
2.1. Backpropagation	6
2.2. Función XOR	7
2.3. Suma de lineal y trigonométricas	8
3. Máquina restringida de Boltzmann	10
3.1. Preentrenamiento	10
3.2. Reconstrucción de MNIST	11
4. Red convolucional	14
4.1. Clasificación MNIST	15
5. Autoencoder	17
5.1. MNIST	17

Resumen

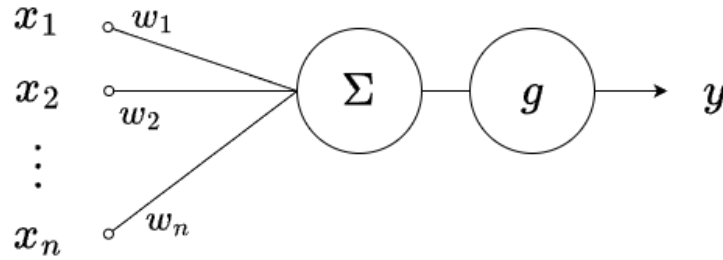
En este trabajo se abordan el diseño, implementación y análisis de varias arquitecturas de redes neuronales. En particular, se cubren un perceptrón simple, un perceptrón multicapa, una máquina restringida de Boltzmann, una red convolucional y un auto-encoder. Se cubre el algoritmo de aprendizaje gradient descent, usando backpropagation para el cálculo de los gradientes, y se usa el algoritmo de preentrenamiento introducido en el trabajo de Hinton y Salakhutdinov (2006) para la de la máquina de Boltzmann.

1. Perceptrón simple

El perceptrón simple es la unidad básica de una red neuronal. Es una neurona artificial que usa una función de activación escalón. La función mapea la entrada \mathbf{x} (un vector de números reales) a una salida binaria $f(\mathbf{x})$:

$$f(\mathbf{x}) = h(\mathbf{w} \cdot \mathbf{x} + b),$$

donde h es la función escalón, \mathbf{w} es el vector de pesos, \mathbf{x} es el vector de entradas y b es el sesgo. El sesgo genera una traslación del límite de decisión, que es el hiperplano definido por $\mathbf{w} \cdot \mathbf{x} + b = 0$.

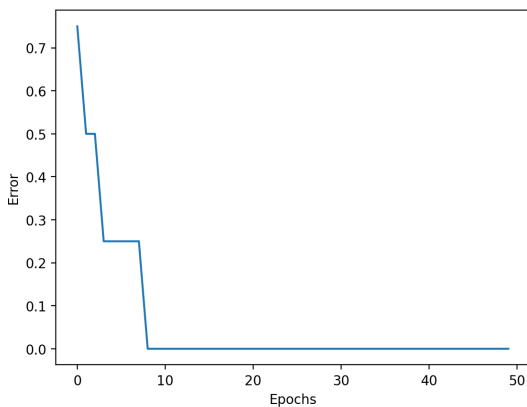


El entrenamiento de un perceptrón simple, consiste en la actualizar los pesos y el sesgo de acuerdo a

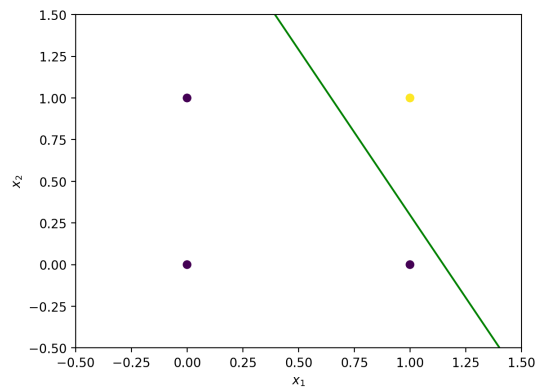
$$w_i = \alpha(y - \hat{y})x_i, \quad b = \alpha - (y - \hat{y}),$$

donde α es la tasa de aprendizaje, el par x, y es un ejemplo de entrenamiento y \hat{y} es la salida del perceptrón. En primer lugar se implementó un perceptrón simple que aprendió la función lógica AND de dos entradas.

La evolución del error durante el entrenamiento y la recta discriminadora aprendida por el perceptrón, $x_2 = -(w_1x_1 = b)/w_2$, se muestran a continuación:

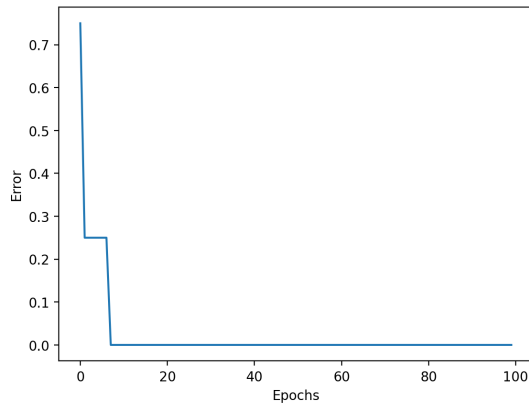


Evolución del error

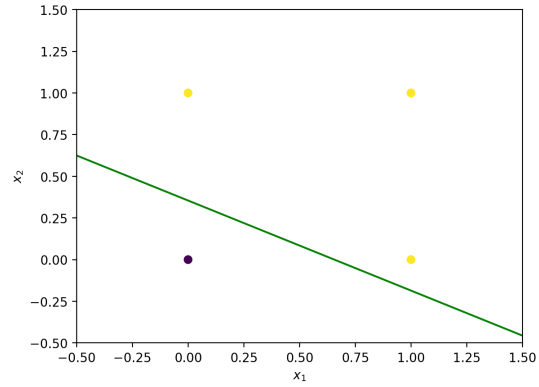


Recta discriminadora

Seguidamente, se entrenaron perceptrones para que aprendieran las funciones lógicas OR de dos entradas, y AND y OR de cuatro entradas. Los resultados se muestran a continuación:

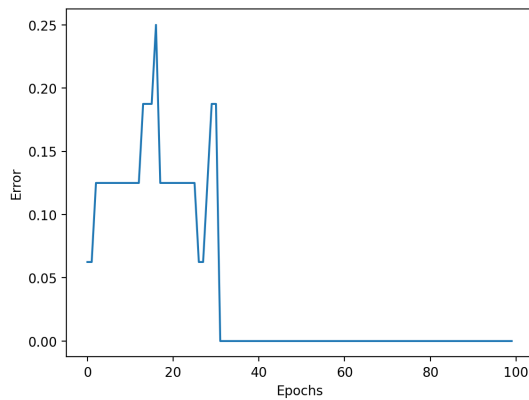


Evolución del error

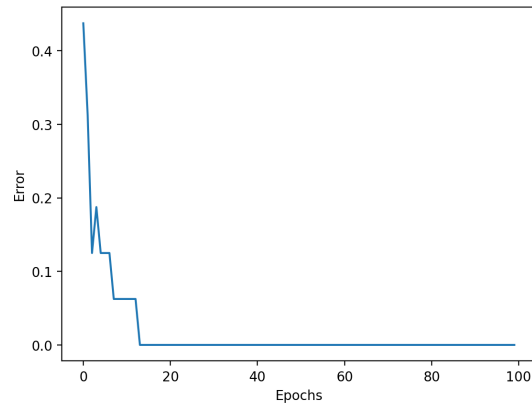


Recta discriminadora

OR de dos entradas



AND de cuatro entradas

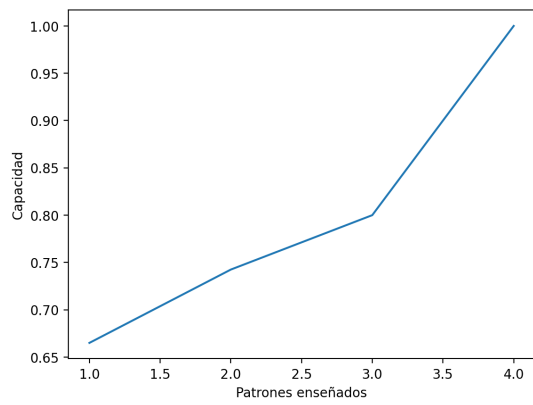


OR de cuatro entradas

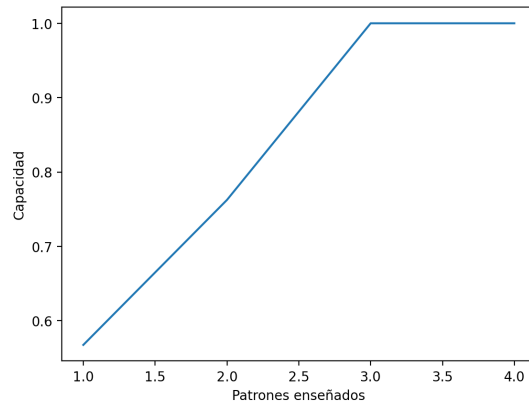
Evolución del error

1.1. Capacidad del perceptrón

El perceptrón simple es capaz de aprender funciones linealmente separables. Una función es linealmente separable si existe un hiperplano que separa los puntos de una clase de los de otra. Tanto las funciones AND y OR de dos entradas como las de cuatro entradas son linealmente separables. A continuación se evalúa la capacidad del perceptrón para estas cuatro funciones, en función de los patrones enseñados. Para eso se calcula el promedio de la tasa de aciertos todos los patrones, de un perceptrón que sólo ve una fracción de los mismos, en cien experimentos. Los resultados se muestran a continuación.

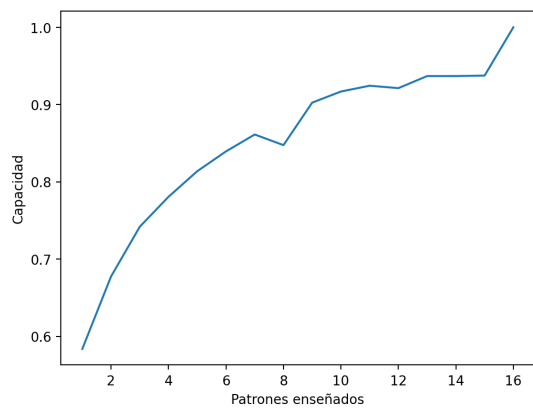


AND de dos entradas

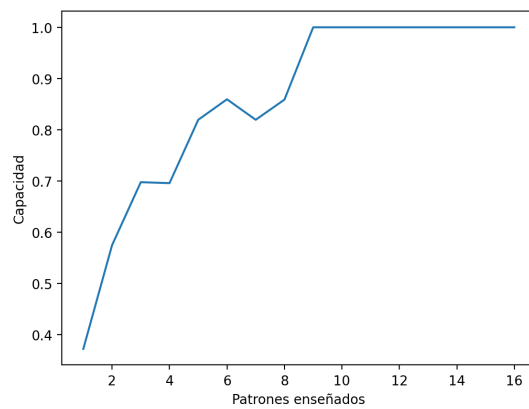


OR de dos entradas

Capacidad



AND de cuatro entradas

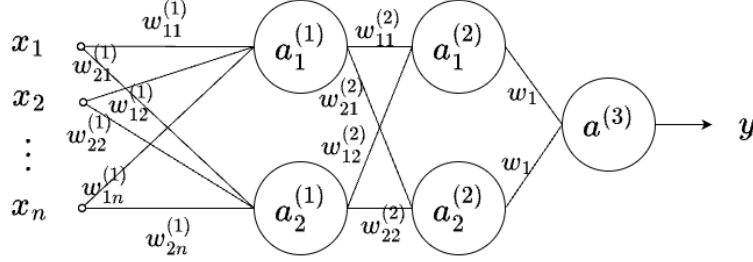


OR de cuatro entradas

Capacidad

2. Perceptr3n multicapa

El perceptr3n multicapa (MLP) es una red neuronal artificial que consiste en m3ltiples capas de perceptrones. La idea de un perceptr3n multicapa es aproximar una funci3n f^* . Por ejemplo, para un clasificador, $y = f^*(\mathbf{x})$ mapea una entrada \mathbf{x} a una categor3a y . Un MLP define un mapeo $y = f^*(\mathbf{x}, \boldsymbol{\theta})$ y aprende los valores de los par3metros $\boldsymbol{\theta}$ que resultan en la mejor aproximaci3n. Los MLPs son llamados redes porque generalmente est3n representados por la composici3n de distintas funciones. El modelo est3 asociado con un grafo ac3clico que describe c3mo las funciones est3n compuestas.



La salida de un MLP, la activaci3n de su 3ltima capa, est3 dada por la aplicaci3n de la funci3n de activaci3n σ a la suma pesada de las activaciones de la capa anterior $a_i^{(L)} = \sigma(\sum_{j=1}^m w_{ij}^{(L)} a_j^{(L-1)} + b_i^{(L)})$, donde m es la cantidad de n3uronas en la capa $L - 1$. En forma vectorial:

$$\begin{aligned} \mathbf{a}^{(L)} &= \sigma(\mathbf{W}^{(L)} \mathbf{a}^{(L-1)} + \mathbf{b}^{(L)}) \\ \mathbf{a}^{(L-1)} &= \sigma(\mathbf{W}^{(L-1)} \mathbf{a}^{(L-2)} + \mathbf{b}^{(L-1)}) \\ &\vdots \\ \mathbf{a}^{(2)} &= \sigma(\mathbf{W}^{(2)} \mathbf{a}^{(1)} + \mathbf{b}^{(2)}) \\ \mathbf{a}^{(1)} &= \sigma(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)}), \end{aligned}$$

donde σ es la funci3n de activaci3n, $\mathbf{W}^{(l)}$ es el conjunto de matrices de pesos de la capa l , $\mathbf{b}^{(l)}$ es el vector de sesgos de la capa l y $\mathbf{a}^{(l)}$ es la activaci3n de la capa l , compuesta por las n unidades $a_1^{(l)}, a_2^{(l)}, \dots, a_n^{(l)}$.

Para entrenar perceptr3n multicapa, se usa el algoritmo del gradiente descendiente. Se calcula el gradiente de la funci3n de p3rdida $L = L(y, f(\mathbf{x}))$ con respecto a los pesos y sesgos de la red, y se actualizan los par3metros en la direcci3n opuesta al gradiente, que apunta en la direcci3n de m3ximo crecimiento de 3sta funci3n. Para calcular el gradiente, se usa el algoritmo de backpropagation, que calcula los gradientes de la funci3n de p3rdida con respecto a los pesos y sesgos de la red, propagando el error hacia atr3s en la red.

2.1. Backpropagation

Es 3til para definir el algoritmo definir una cantidad intermedia, δ_j^l , que mide el error en el nodo j de la capa l .

$$d_j^l = \frac{\partial L}{\partial z_j^l}.$$

Las cuatro ecuaciones fundamentales del algoritmo son: el error en la capa de salida δ^L

$$\delta_j^L = \frac{\partial L}{\partial a_j^L} \sigma'(z_j^L),$$

que en forma matricial es

$$\delta^L = \nabla_a L \odot \sigma'(z^L), \quad (\text{BP1})$$

donde $\nabla_a L$ es el vector cuyas componentes son las derivadas parciales $\partial L / \partial a_j^L$. El error δ^l en función de δ^{l+1} :

$$\delta^l = ((w^{l+1})^\top \delta^{l+1}) \odot \sigma'(z^l). \quad (\text{BP2})$$

Combinando (BP1) y (BP2) se obtiene δ^l para cualquier capa en la red. Una ecuación para la tasa de cambio del costo L con respecto a cualquier sesgo en la red:

$$\frac{\partial L}{\partial b_j^l} = \delta_j^l. \quad (\text{BP3})$$

Por último, una ecuación para la tasa de cambio del costo L con respecto a cualquier peso en la red:

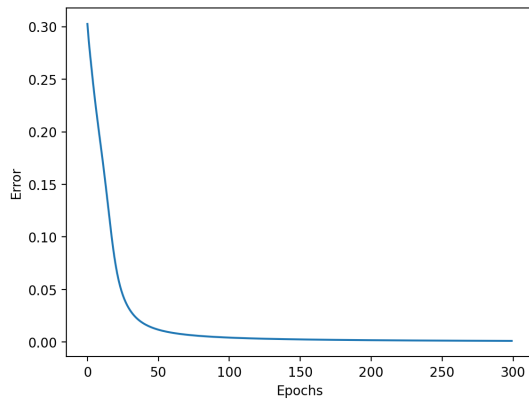
$$\frac{\partial L}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l. \quad (\text{BP4})$$

El algoritmo backpropagation queda entonces definido de la siguiente manera:

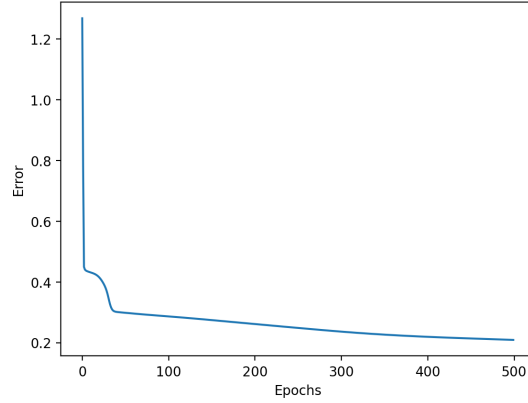
1. **Entrada x :** Inicializar las activaciones $a^{(1)}$ para la capa de entrada.
2. **Feedforward:** Para cada capa $l = 2, 3, \dots, L$ calcular $z^{(l)} = w^{(l)} a^{(l-1)} + b^{(l)}$ y $a^{(l)} = \sigma(z^{(l)})$.
3. **Error en la salida $\delta^{(L)}$:** Computar el vector $\delta^{(L)} = \nabla_a L \odot \sigma'(z^{(L)})$.
4. **Retropropagación del error:** Para cada capa $l = L - 1, L - 2, \dots, 2$ computar $\delta^{(l)} = ((w^{(l+1)})^\top \delta^{(l+1)}) \odot \sigma'(z^{(l)})$.
5. **Salida:** El gradiente de la función de pérdida con respecto a los pesos y sesgos de la red está dado por $\frac{\partial L}{\partial w_{jk}^{(l)}} = a_k^{(l-1)} \delta_j^{(l)}$ y $\frac{\partial L}{\partial b_j^{(l)}} = \delta_j^{(l)}$.

2.2. Función XOR

Se entrenaron perceptrones multicapa para que aprendieran las funciones lógicas XOR de dos y cuatro entradas. Para esto se utilizó el algoritmo de gradient descent actualizando en mini batches, junto con backpropagation para calcular los gradientes correspondientes. Se usaron dos capas ocultas, con 3 neuronas cada una, la función de activación tanh, y dos capas ocultas, de 5 neuronas cada una, con la función de activación tanh, respectivamente. Los resultados se muestran a continuación:



XOR de dos entradas

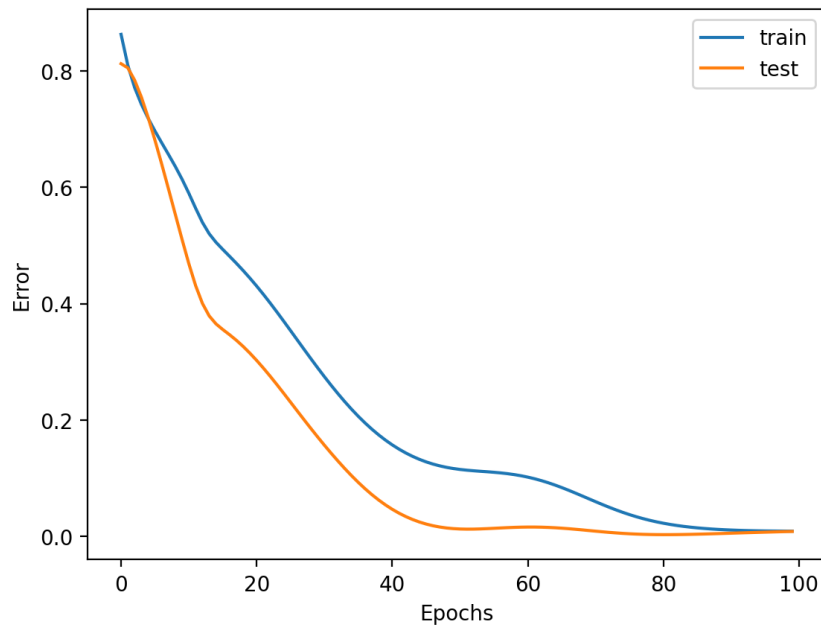


XOR de cuatro entradas

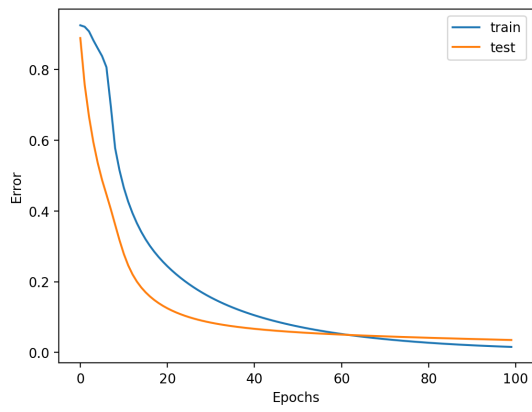
Evolución del error

2.3. Suma de lineal y trigonométricas

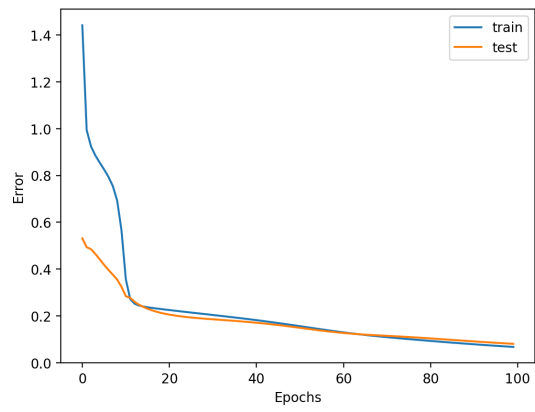
A continuación, se muestran los resultados obtenidos de la evaluación del perceptrón ante la función $f(x, y, z) = \sin x + \cos y + z$, con x e $y \in [0, 2\pi]$ y $z \in [-1, 1]$. Para esto se construyó un set de datos de entrenamiento y otro para evaluación, y se analizó la evolución del error en ambos sets a lo largo del entrenamiento. Primero se usaron dos capas ocultas con nueve neuronas cada una:



Luego se analizó el mismo problema para una red con una única capa oculta con un conjunto de entrenamiento de 40 muestras, para mini batches de 40 y 1 muestra.



Mini batch de 40 muestras



Mini batch de 1 muestra

3. Máquina restringida de Boltzmann

Las máquinas restringidas de Boltzmann (RBMs) son uno de los pilares fundamentales de los modelos estocásticos profundos. Las RBMs son modelos estocásticos profundos indireccionales que contienen una capa de variables observables y una capa de variables latentes. Las RBMs pueden ser apiladas para generar modelos más profundos.

Una máquina de Boltzmann es un grafo bipartito completo, las unidades del modelo pueden ser divididas en dos grupos, en los que las unidades no están interconectadas, pero sí lo están con todas las unidades del otro grupo.

La RBM es un modelo basado en energía, con la probabilidad conjunta definida por su función de energía (distribución de Boltzmann):

$$P(\mathbf{v} = \mathbf{v}, \mathbf{h} = \mathbf{h}) = \frac{\exp(-E(\mathbf{v}, \mathbf{h}))}{Z},$$

donde E es la función de energía, definida

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{b}^\top \mathbf{v} - \mathbf{c}^\top \mathbf{h} - \mathbf{v}^\top \mathbf{W} \mathbf{h},$$

y Z es la constante de normalización conocida como función de partición

$$Z = \sum_{\mathbf{v}} \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h})).$$

De la definición, es aparente que calcular la constante de normalización Z (sumar exhaustivamente todos los estados) es computacionalmente intratable. Para el caso de las máquinas de Boltzmann, esta afirmación está demostrada formalmente. Pero existe la posibilidad de calcularla más rápido aprovechando regularidades en la función de probabilidad, las probabilidades condicionales $P(\mathbf{h}|\mathbf{v})$ y $P(\mathbf{v}|\mathbf{h})$ son factoriales y relativamente simples de computar y de muestrear. Más aún, las distribuciones factoriales dadas la capa oculta y la capa visible son

$$P(\mathbf{h}|\mathbf{v}) = \prod_{j=1}^{n_h} \sigma((2\mathbf{h} - 1) \odot (\mathbf{c} + \mathbf{W}^\top \mathbf{v}))_j,$$
$$P(\mathbf{v}|\mathbf{h}) = \prod_{i=1}^{n_v} \sigma((2\mathbf{v} - 1) \odot (\mathbf{b} + \mathbf{W} \mathbf{h}))_i.$$

3.1. Preentrenamiento

En el trabajo de Hinton y Salakhutdinov (2006) se propuso un algoritmo de preentrenamiento para RBMs. El algoritmo surge de que gradient descent puede ser utilizado para hacer un reajuste fino de los pesos siempre y cuando los pesos se encuentren inicialmente en un estado cercano a una solución.

La idea es hacer que el modelo aprenda una representación de baja dimensionalidad de los datos, usando un **encoder** multicapa que transforme los datos de alta dimensionalidad en un código de baja dimensionalidad y en **decoder** similar que reconstruye los datos originales a partir del código.

Si partimos de pesos aleatorios en ambas redes, pueden ser entrenadas en conjunto mediante la minimización de la discrepancia entre la entrada y la reconstrucción, usando por

ejemplo gradient descent con backpropagation. El sistema en conjunto tiene el nombre de **autoencoder**. Es difícil optimizar los pesos en un autoencoders no lineales que tienen múltiples capas ocultas. Con pesos iniciales altos, generalmente malos mínimos locales son encontrados; con pesos iniciales bajos, los gradientes son demasiado pequeños para entrenar autoencoders profundos. Por otro lado, si los pesos están cerca de una buena solución, el gradiente descendiente funciona bien. El algoritmo de preentrenamiento consiste en varias partes. Primero, la probabilidad de de una imagen del set de entrenamiento puede ser aumentada ajustando los pesos y sesgos para disminuir la energía de la esa imagen y aumentar la energía de las imágenes similares “confabuladas” que la red prefiere en vez de los datos reales. Dada una imagen de entrenamiento, el estado binario h_j de cada detector de features j se pone en 1 con probabilidad $\sigma(b_j + \sum_i v_i w_{ij})$, donde σ es la función sigmoide $1/(1 + \exp(-x))$, b_j es el sesgo de j , v_i es el estado del pixel i y w_{ij} es el peso entre i y j . Una vez elegidos los estados binarios de las unidades ocultas, se produce una confabulación activando (poniendo en 1) cada v_i con probabilidad $\sigma(b_i + \sum_j h_j w_{ij})$, donde b_i es el sesgo de i . Luego se actualizan una vez más los estados de las capas ocultas para que representen features de la confabulación. El cambio en cada peso está dado por

$$\Delta w_{ij} = \epsilon(\langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{recon}}),$$

donde ϵ es la tasa de aprendizaje, $\langle v_i h_j \rangle_{\text{data}}$ es la fracción de veces que el pixel i y la el detector de features j están activados en simultáneo cuando los detectores de features están siendo estimulados por los datos de entrenamiento, y $\langle v_i h_j \rangle_{\text{recon}}$ es la correspondiente fracción de tiempo para las confabulaciones.

Luego de aprender una capa de detectores de features, podemos tratar sus actividades cuando están siendo estimulados por los datos de entrenamiento como datos de entrada para aprender una segunda capa de features. Luego la primer capa de detectores de features se convierte en la capa visible para la próxima RBM.

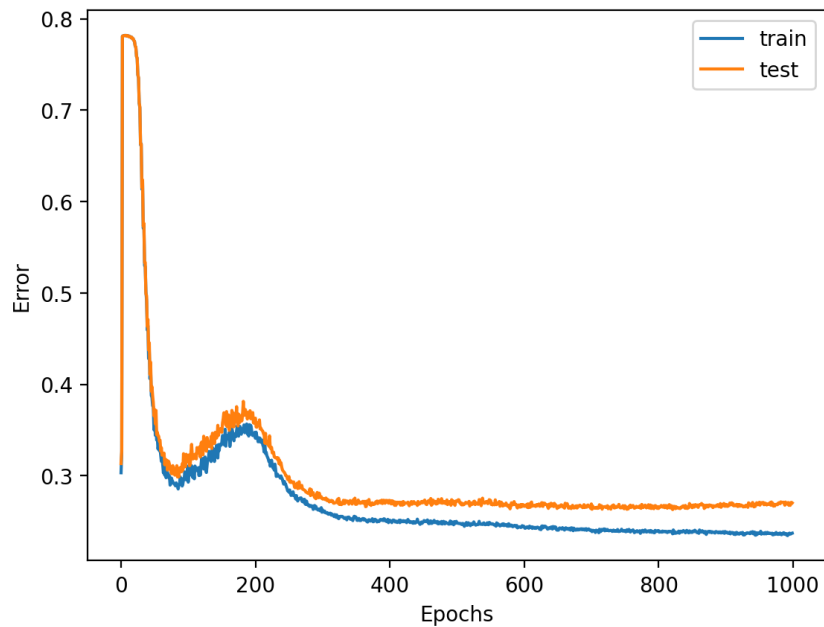
Este entrenamiento capa por capa puede repetirse arbitrariamente. Se puede mostrar que añadir una capa extra siempre mejora la cota inferior de la log-verosimilitud de que el modelo asigna a los datos de entrenamiento, siempre que el número de detectores de features por capa no disminuya y siempre que sus pesos se inicialicen correctamente.

Cada capa de features captura fuertes correlaciones entre las actividades de las unidades en la capa anterior. Luego de preentrenar múltiples capas de detectores de features, el modelo se despliega para producir un encoder y un decoder que inicialmente usan los mismos pesos. La etapa de reajuste fino global reemplaza entonces las actividades estocásticas por probabilidades reales determinísticas y usa backpropagation sobre todo el autoencoder para realizar el ajuste fino de los pesos para lograr una reconstrucción óptima. Para datos continuos, las unidades ocultas de la RBM de la primera capa siguen siendo binarias, pero las unidades visibles son reemplazadas por unidades lineales con ruido gaussiano. Si el ruido tiene varianza unitaria, la regla estocástica de actualización para las unidades ocultas permanece entonces igual y la regla de actualización para cada unidad visible i es simplemente muestrear de una gaussiana con varianza unitaria y media $b_i + \sum_j h_j w_{ij}$.

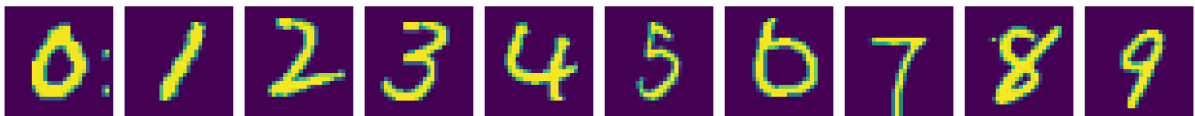
3.2. Reconstrucción de MNIST

Se entrenó una pila de RBMs para reconstruir imágenes del set de datos MNIST. Se usaron tres máquinas restringidas de Boltzmann con 600, 400 y 200 unidades ocultas por capa respectivamente, con 784 unidades visibles en la primer máquina, siendo que las

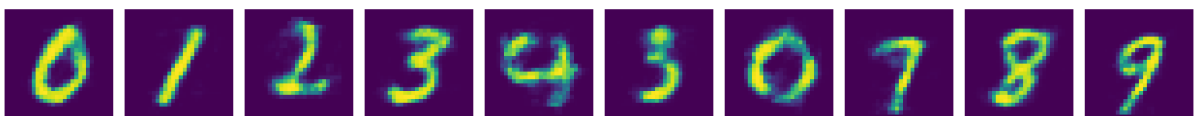
imágenes del set son de 72 por 72 píxeles. El algoritmo utilizado para el aprendizaje es el de preentrenamiento descrito en el paper de Hinton y Salakhutdinov. A continuación se muestra la evolución de los errores de entrenamiento en mil épocas, usando como métrica de error la suma de los errores absolutos en cada píxel, y la reconstrucción de imágenes del set para cada clase del mismo junto a las imágenes originales.



Evolución del error



Imágenes originales



Reconstrucciones

4. Red convolucional

Las redes convolucionales o redes neuronales convolucionales (CNNs), son una especie especial de redes neuronales para procesar datos que tienen una topología de cuadrícula conocida. El nombre red convolucional indica que la red emplea una operación matemática llamada convolución, que es una especie particular de operación lineal. En una CNN, la entrada es un tensor de dimensión

cantidad de entradas \times altura de la entrada \times ancho de la entrada \times cantidad de canales,

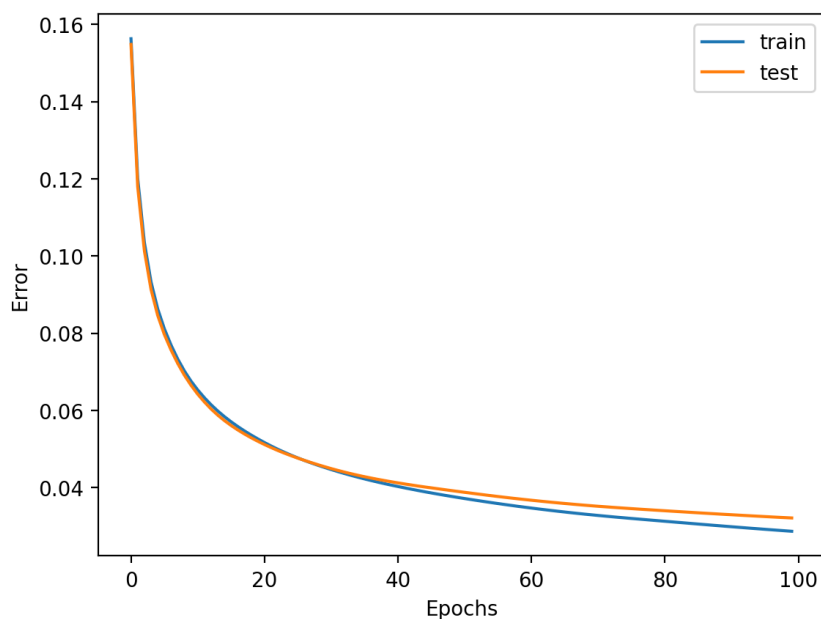
luego de pasar por una capa convolucional, la imagen se abstrae a un mapa de features, con dimensiones

cantidad de entradas \times altura del mapa \times ancho del mapa \times cantidad de canales del mapa.

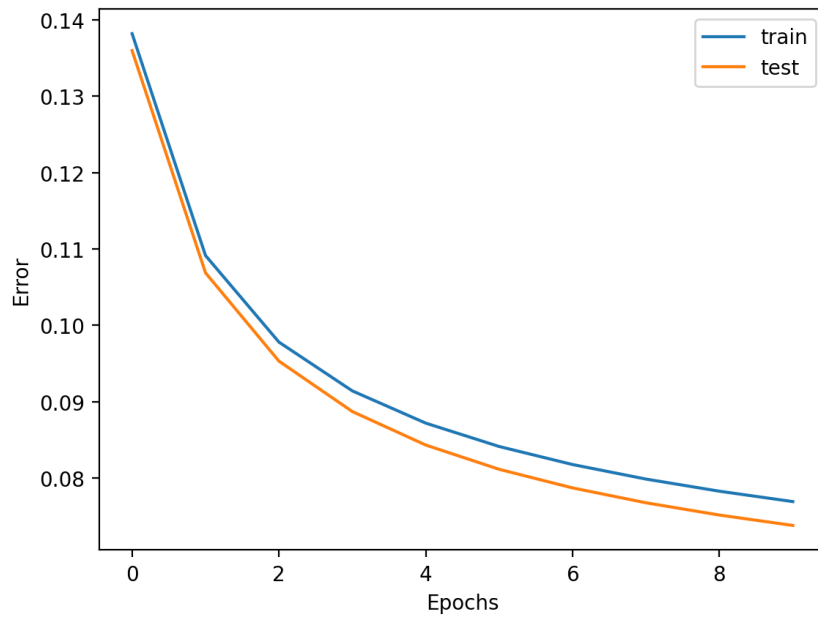
Las capas convolucionales convolucionan la entrada y pasan el resultado a la capa siguiente. La convolución reduce la cantidad de parámetros, logrando así que sea práctico entrenar redes profundas.

4.1. Clasificación MNIST

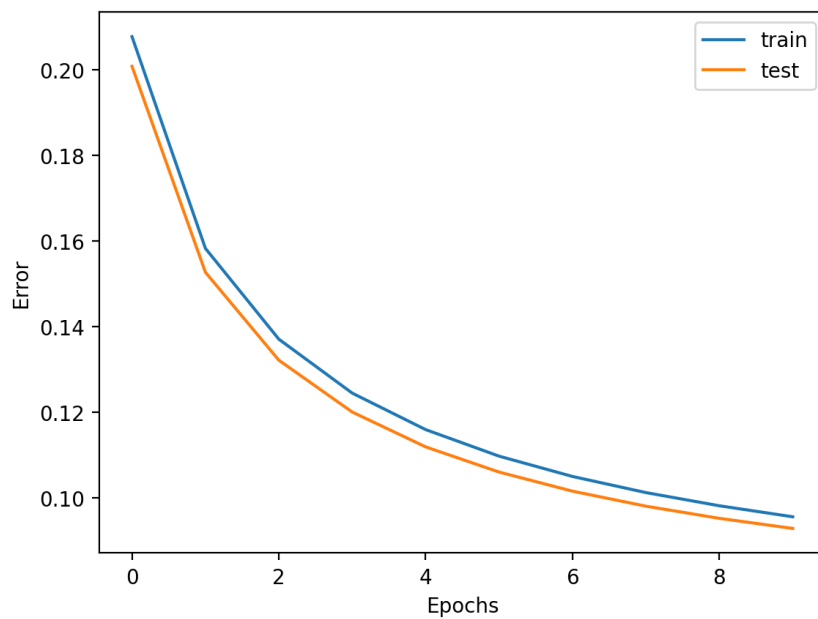
A continuación se presentan los resultados obtenidos de entrenar una red convolucional para clasificar las imágenes de la base de datos MNIST. La primera arquitectura por la que se optó, consiste en una capa convolucional con 5 filtros de 3×3 y luego tres capas densas de 3380, 100 y 10 neuronas respectivamente. A continuación se muestra la evolución de la entropía cruzada durante el entrenamiento para el set de entrenamiento y el de evaluación:



Luego se buscó encontrar la red convolucional más pequeña que obtuviera un error menor al 10 %. La arquitectura final resultó en una capa convolucional de un sólo kernel de 10×10 seguida de una dos capas densas de 261 y 10 neuronas respectivamente. Por último, se entrenaron múltiples perceptrones multicapa para la misma tarea, el perceptrón más pequeño que obtuvo un error menor al 10 % consistió en dos capas de 784 y 10 neuronas respectivamente. Los resultados de estos dos últimos experimentos se muestran a continuación:



Red convolucional pequeña



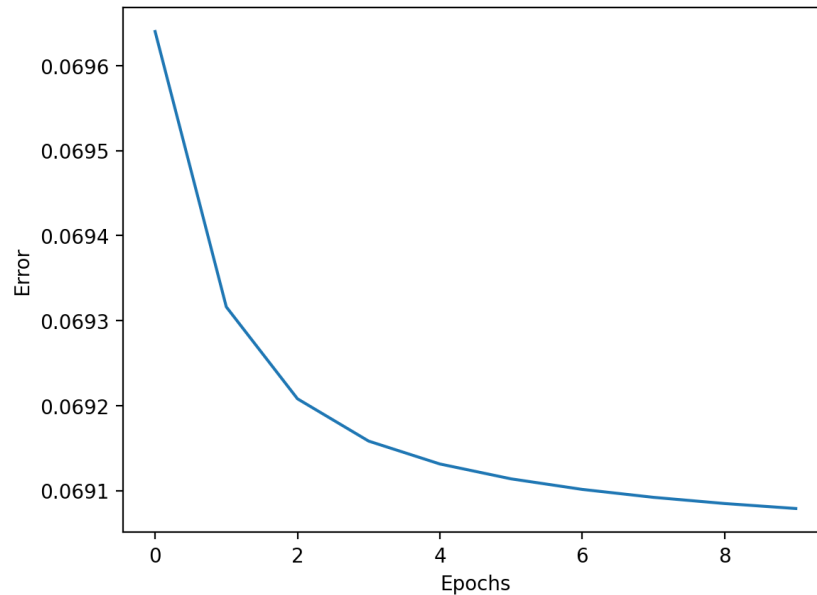
Perceptrón multicapa pequeño

5. Autoencoder

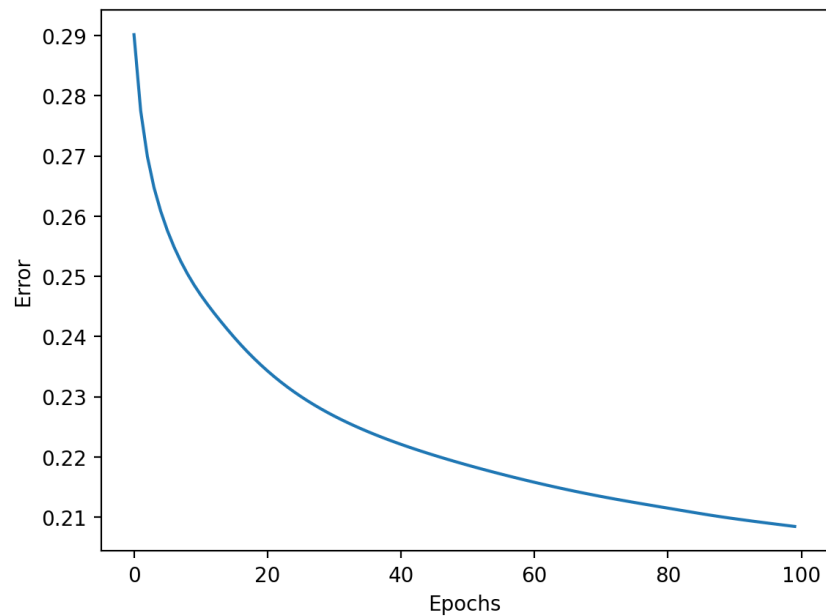
Un autoencoder es una red neuronal entrenada para intentar copiar su entrada en su salida, con una arquitectura que busca reducir la dimensionalidad del problema en las capas intermedias.

5.1. MNIST

Se entrenó un autoencoder para reconstruir imágenes de la base de datos MNIST, para luego deshacerse de las capas de decodificación y terminar con una representación de baja dimensionalidad de los datos del set. Esta representación fue utilizada para entrenar un perceptrón multicapa para clasificar las imágenes. A continuación se muestran los resultados del experimento:



Evolución del error del autoencoder



Evolución del error del MLP

Referencias

- [1] Ian Goodfellow. Yoshua Bengio. Aaron Courville, *Deep Learning*.
- [2] Michael Nielsen, *Neural Networks and Deep Learning*.
- [3] G. E. Hinton. R. R. Salakhutdinov, *Reducing the Dimensionality of Data with Neural Networks*.
- [4] Yann LeCun. Patrick Haffner. Léon Bottou. Yoshua Bengio, *Object Recognition with Gradient-Based Learning*.