



# **REDES NEURONALES 2024**

**Clase 21 parte 2**

**. Lunes 4 de noviembre 2024**

**FAMAF, UNIVERSIDAD NACIONAL DE CÓRDOBA**

**INSTITUTO DE FÍSICA ENRIQUE GAVIOLA (UNC-CONICET)**

## LOS PROBLEMAS DEL BACK-PROPAGATION

- Rugosidad de la función: El grado de rugosidad de una función está dado por el número de mínimos locales y puntos de ensilladura. En el caso de las redes neuronales con multiplicidad de capas o *profundas*, se sabe que la cantidad de mínimos y puntos de ensilladura aumentarán exponencialmente mientras crezca el número de acoplamientos sinápticos. Estos, a su vez, dependerán de la arquitectura de la red, de la cantidad de neuronas de entrada, neuronas de salida, cantidad de capas ocultas y el número de neuronas dispuestas en ellas. Entonces cuanto mayor sea la cantidad de neuronas, aumenta el número de acoplamientos y, por lo tanto, aumentarán los mínimos locales en la función (el error en estos puntos sería de un valor distinto de cero), de manera que la función error es muy rugosa (producto de la alternancia de signos y la aleatoriedad). Por consiguiente la función podría o bien quedarse atrapada en mínimos locales o alcanzar puntos de inflexión para los cuales requeriría mucho tiempo salir.

- Valores iniciales de  $w_{ij}$ : Cuando los valores asignados a la matriz de acoplamientos sinápticos se eligen inicialmente al azar, la evolución del error (es decir, el descenso que se realiza sobre la superficie en función del gradiente) dependerá de estos valores. En este caso, alcanzar un mínimo local se verá estrechamente afectado por la elección de los valores de los pesos sinápticos iniciales, de forma que la probabilidad de hallar los mínimos locales dependerá de la probabilidad de caer cerca de ellos al seleccionar los valores de los acoplamientos.

- Dependencia de las derivadas: Como es posible observar en la expresión de la función error, para corregir los acoplamientos es preciso conocer las derivadas de las funciones de activación. Esto significa que cuanto mayor sea la complejidad del proceso de derivación de las funciones, mayor será el tiempo necesario para la evolución del sistema. Cuando se analizan los tiempos de análisis, es preciso considerar igualmente lo que ocurriría si la función se encontrara en puntos de ensilladura o si las derivadas tomaran valores próximos a cero dado que la evolución del error dependerá del gradiente. En estos casos, la evolución y aprendizaje del sistema también requeriría mucho tiempo haciendo que el sistema avance cada vez más lentamente a medida que agregan capas y que se aplique la regla de la cadena al cálculo del gradiente. De modo que la anidación de funciones anteriores para cada capa siguiente implican complejidad matemática y computacional dificultando el aprendizaje de la red y la búsqueda de una solución para el problema analizado.

- Dependencia de la razón de aprendizaje: La regla de aprendizaje depende del factor  $\eta$  (razón de aprendizaje) siendo muy sensible a las variaciones en su valor y pudiendo divergir con facilidad (alejándose del valor en el cual  $E = 0$ ) sumado al hecho que conocer con exactitud su valor óptimo (de modo tal que permita alcanzar el mínimo error en el menor número de pasos) resulta una tarea difícil.

- Overfitting<sup>1</sup>: Aunque está más asociado a los problemas propios del machine-learning, el overfitting es otro inconveniente que suele presentarse y que se observa al graficar los valores de la función costo en el tiempo (aumentando la cantidad de épocas). Derivado del objetivo del aprendizaje supervisado (que la respuesta de la red coincida con la respuesta deseada), el overfitting implica una red que predice el conjunto de entrenamiento con una alta precisión de manera que es posible correr el riesgo de incluir el ruido de los datos memorizando peculiaridades y evitando encontrar un conjunto de acoplamiento que permita predecir los resultados. Es decir la red predice muy bien los elementos del conjunto de entrenamiento pero no responde de forma deseada cuando se le presentan los conjuntos de testeos haciendo que la red no generalice correctamente. El exceso de precisión a la hora ajustar el conjunto de entrenamiento se traduce en emplear modelos que incluyen más parámetros o términos de los necesarios o bien modelos más complejos que los necesarios<sup>2</sup>, lo que afecta el cálculo del gradiente e incrementa los costos computacionales. Finalmente es importante no perder de vista que el objetivo del aprendizaje supervisado de una red neuronal artificial es poder usarla para resolver problemas de complejidad variable. En otras palabras, no resulta útil contar con una red que ajusta con alta precisión los ejemplos sino es capaz de predecir la respuesta del conjunto de testeos y, por lo tanto no es confiable para ser usada con datos desconocidos.

Fundamentalmente, puede observarse que la complejidad de estas redes está vinculado con la difícil tarea de ajustar una función de  $\mathbb{R}^N$  a  $\mathbb{R}^M$ , donde N son las neuronas de entrada y M las neuronas de salida. De manera que es preciso establecer una arquitectura y una matriz de acoplamientos sinápticos que permita ajustar el espacio en que se encuentran los elementos de entrada con el espacio en el que se hallan los estados de las neuronas de salida. Por último y considerando la forma de la regla de aprendizaje, es preciso destacar que la complejidad de su cálculo (y, por lo tanto, los tiempos de procesamiento asociados) irán de la mano con la complejidad de la red neuronal y su arquitectura.

# UN PLAN DE MEJORAS PARA EL BACK-PROPAGATION

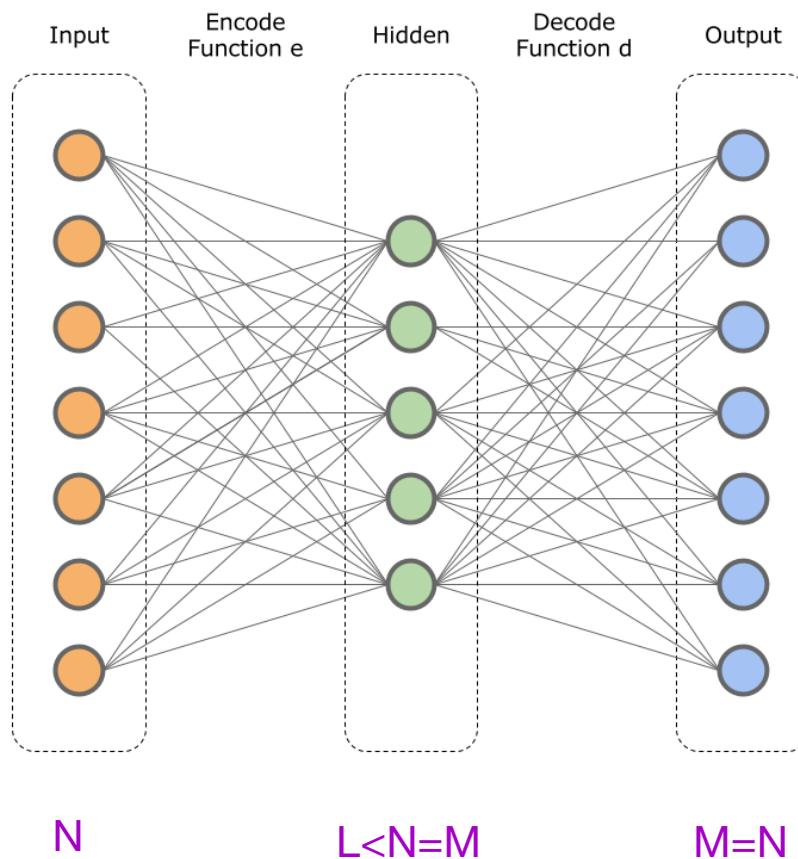
## De las redes feed forward panditas a las profundas

- ¿Cómo solucionar la elección inicial de los valores de la matriz  $w$ ?

**Autoencoder<sup>3</sup>**: Es un algoritmo de aprendizaje que aplica back propagation y permite el preprocesamiento de los datos con el objetivo que la salida sea igual a la entrada ( $y = x$ ), es decir la red intenta aprender la función identidad donde  $O_i^\mu \approx \xi_k^\mu$  para que la salida sea similar a la entrada. La peculiaridad de este algoritmo consiste en fijar un número de neuronas en la capa intermedia que sea inferior a la cantidad de neuronas en las capas de entrada y salida (que contienen la misma cantidad de neuronas dado que la salida debe ser la entrada). Este algoritmo no parece particularmente difícil o innovador, sin embargo cuando se imponen restricciones en la red (como limitar el número de neuronas en la capa oculta), la red es obligada a encontrar patrones o correlaciones entre los elementos, de forma que es posible aprender una representación comprimida de la entrada permitiendo reducir la dimensionalidad inicial

de los datos. Entonces el autoencoder se aplica para entrenar las capas a partir de la función identidad al reducir el número de neuronas permitiendo la codificación de la información. En este sentido, se encontró que cuando se aplica un preprocesamiento de los acoplamientos y se parte de redes que han sido autoaprendidas con el autoencoder, el método de back propagation resulta mucho más eficiente que cuando se tabulan estos valores al azar.

$\rightarrow \xi$        $\rightarrow V$        $\rightarrow O$



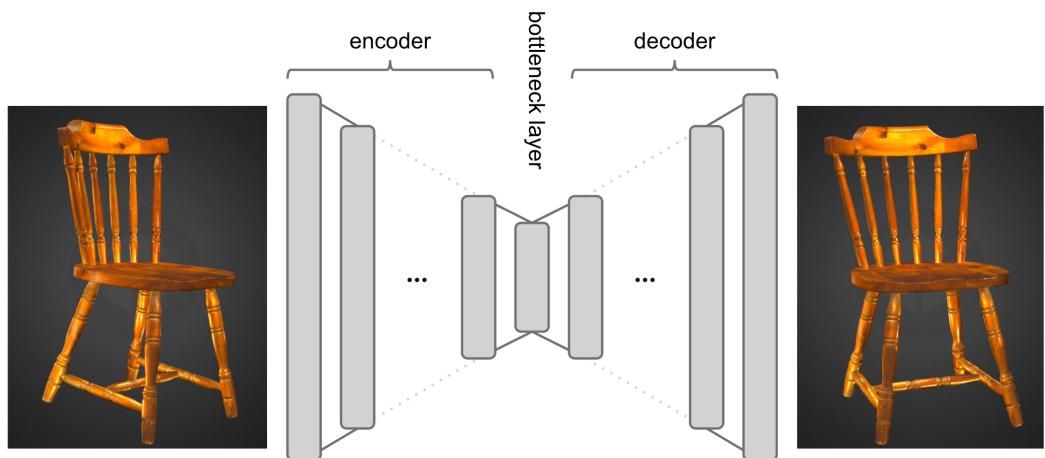
El autoencoder tiene el mismo número de neuronas en la capa de salida que en la capa de entrada para representar la identidad. En particular, debemos buscar la forma de que las capas de salida tengan el mismo tipo de neuronas que en la capa de entrada. Esto va a requerir cierto grado de dedicación. cuando la entrada tenga valores discretos, ya que para aplicar back-propagation debemos tener a la salida neuronas continuas.

Un auto encoder propiamente dicho debe aprender la función [identidad](#)

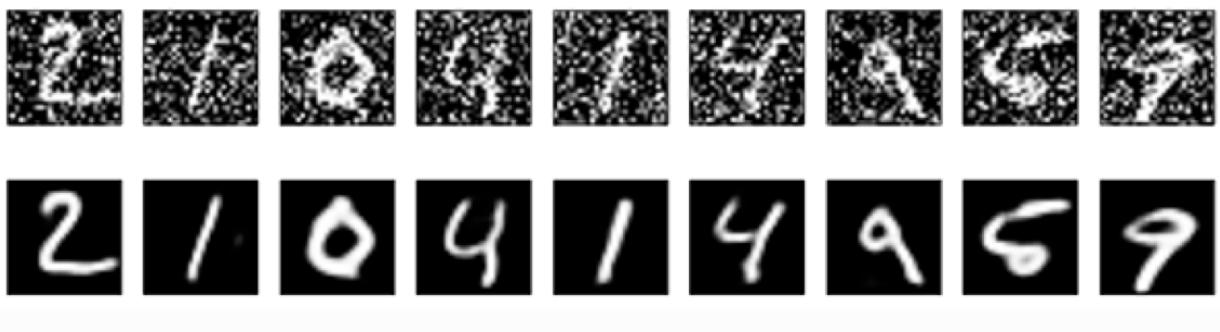
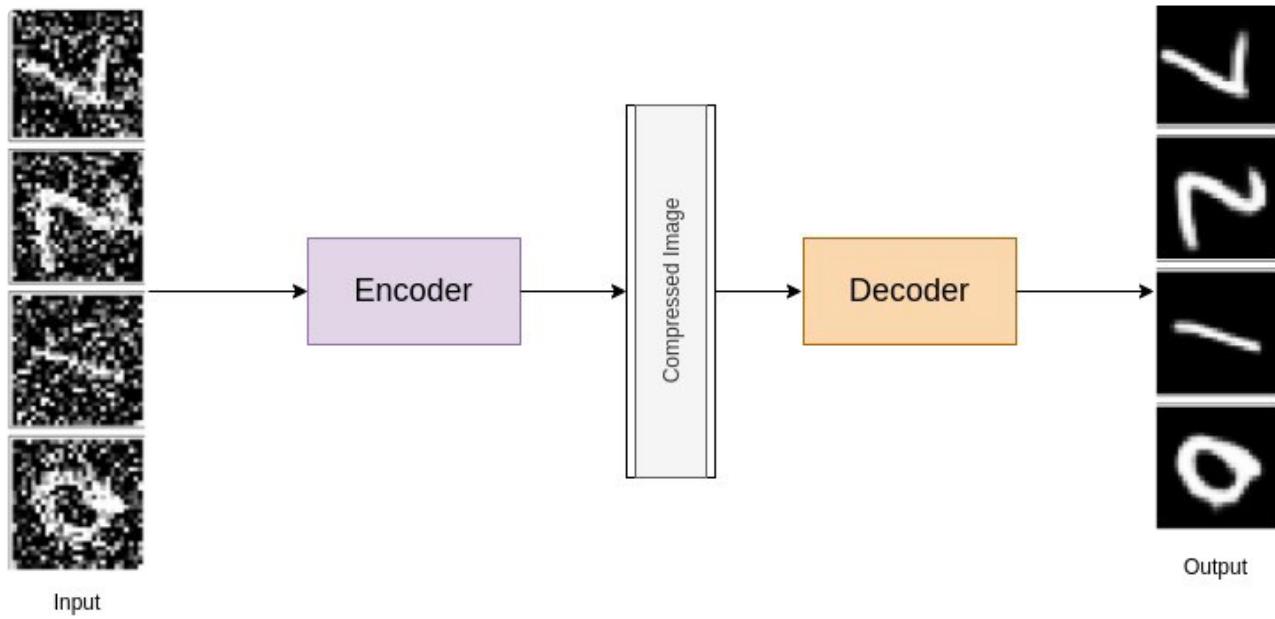
$$\phi_i^h = \xi_i^h \quad \forall i \wedge \forall h$$

y en este caso no precisamos de conjunto de entrenamiento, pues la entrada es a la vez la etiqueta correcta. Decimos que es caso muy particular de **APRENDIZAJE NO SUPERVISADO**.

## Aplicación a esterovisión



## Filtrar ruido



# Identificar marcas

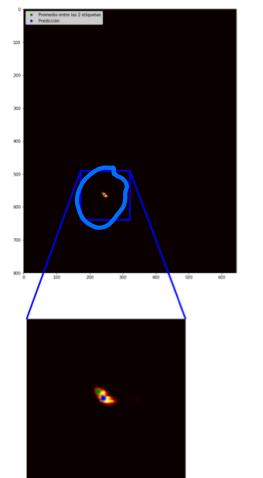
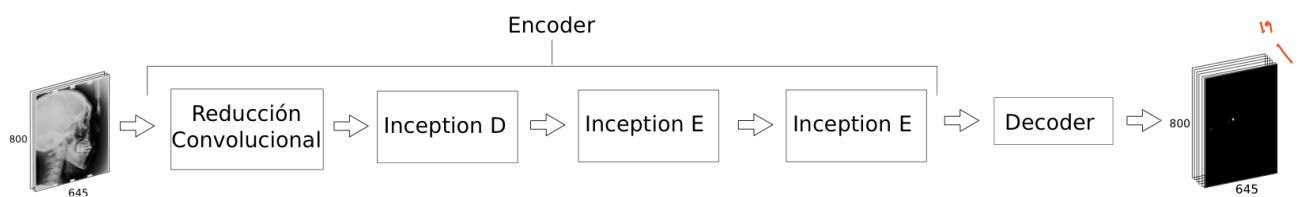


FIGURA 5.5: detecciones y etiquetas del landmark 10 sobre su respectivo mapa de probabilidad de una imagen del conjunto de test 1.

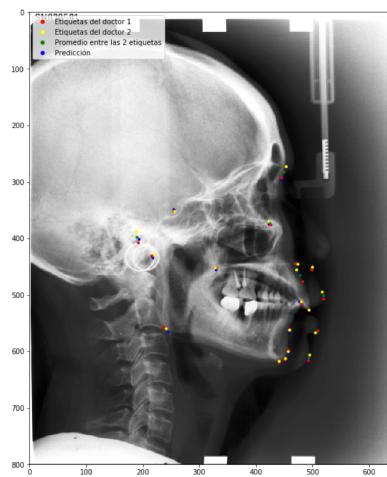
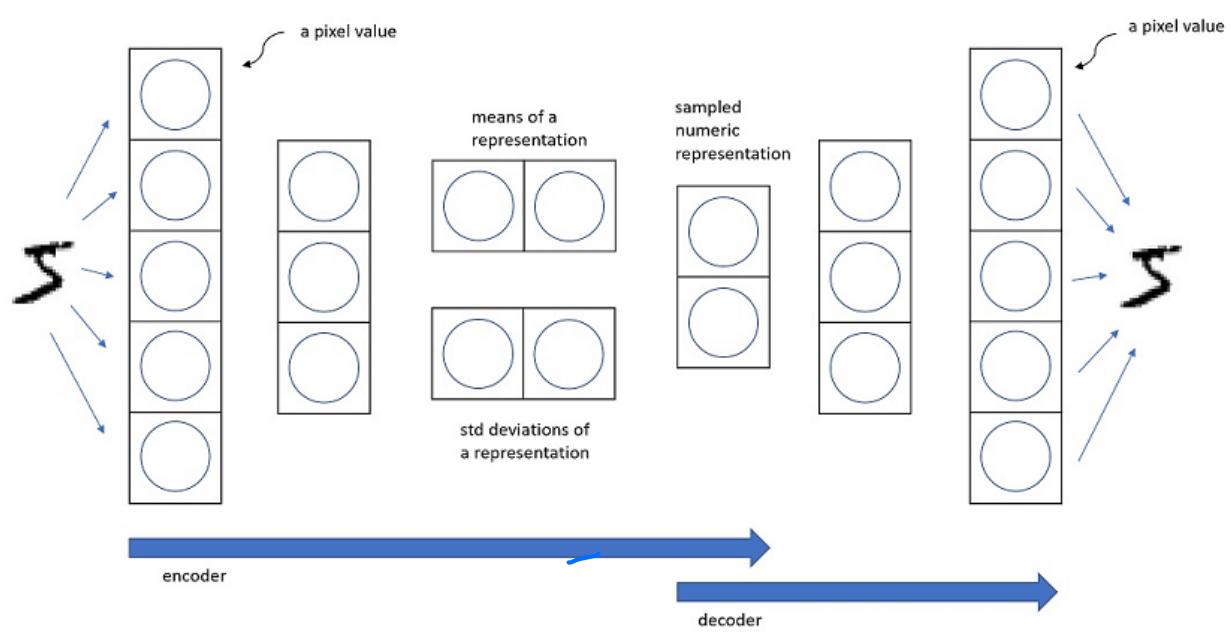
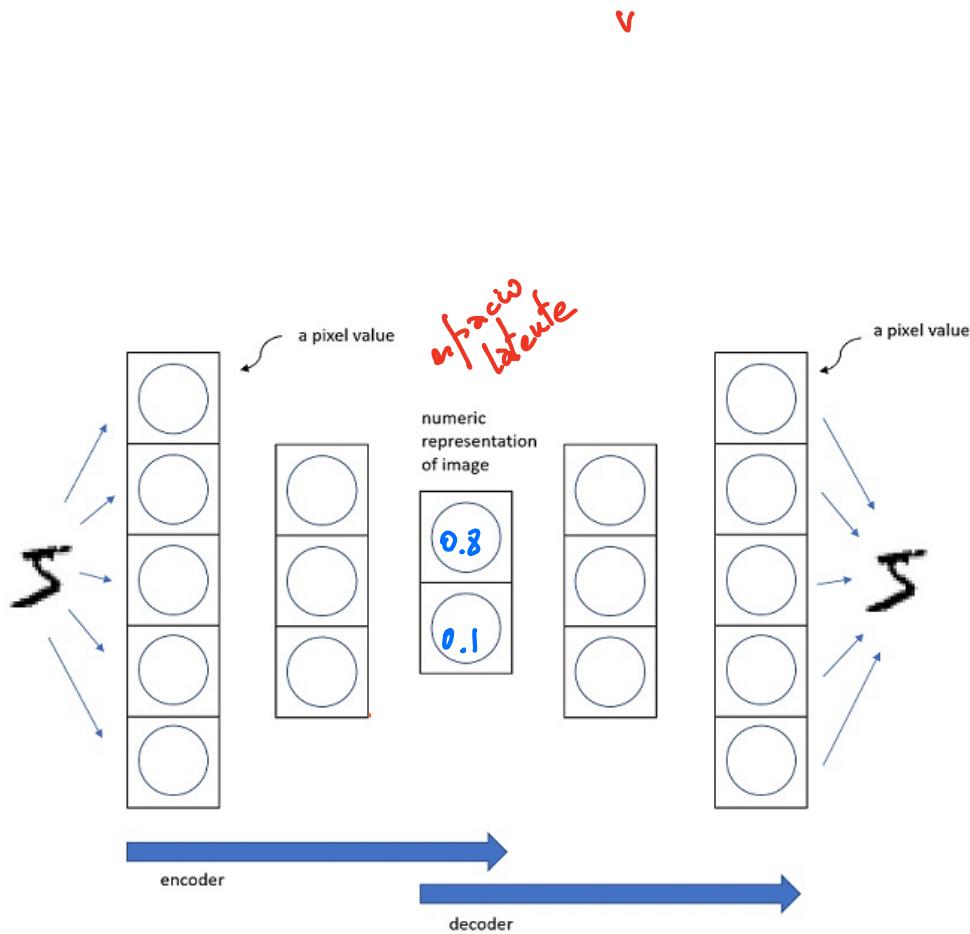
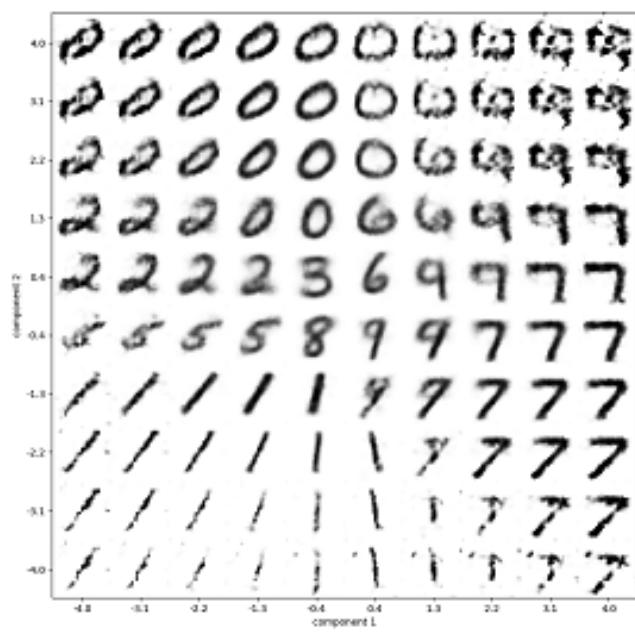
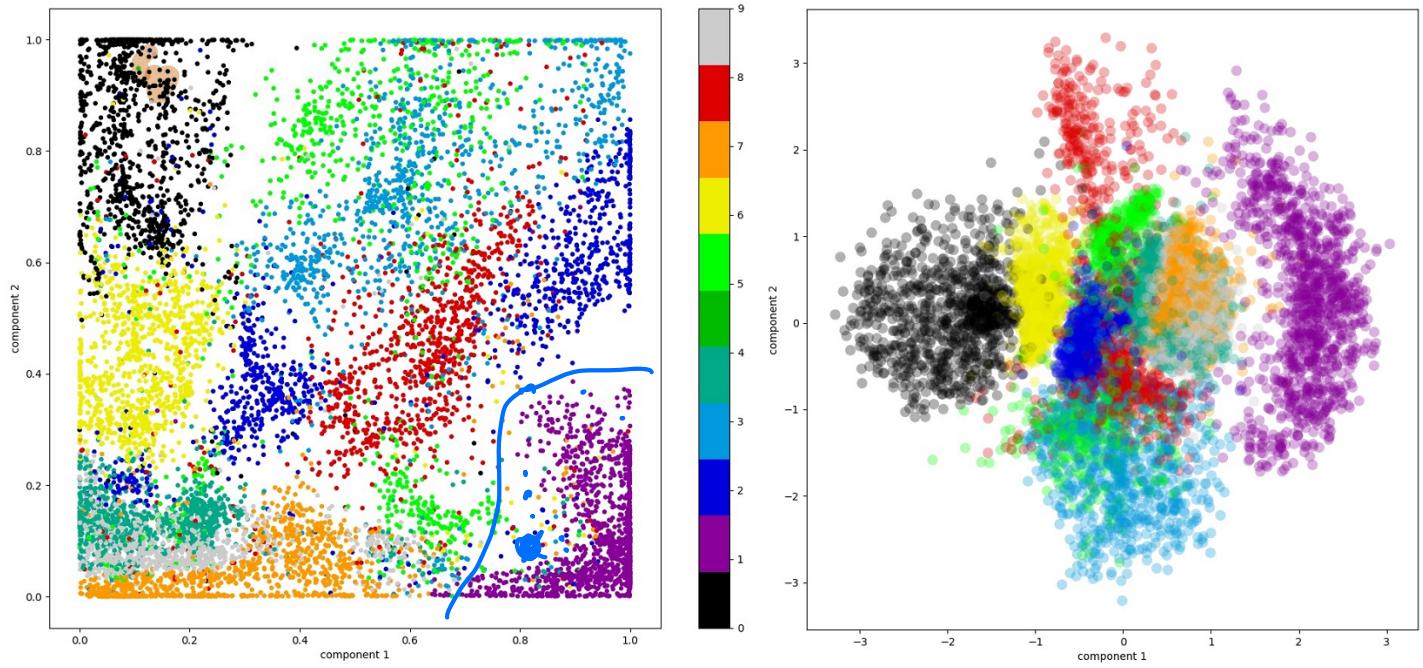


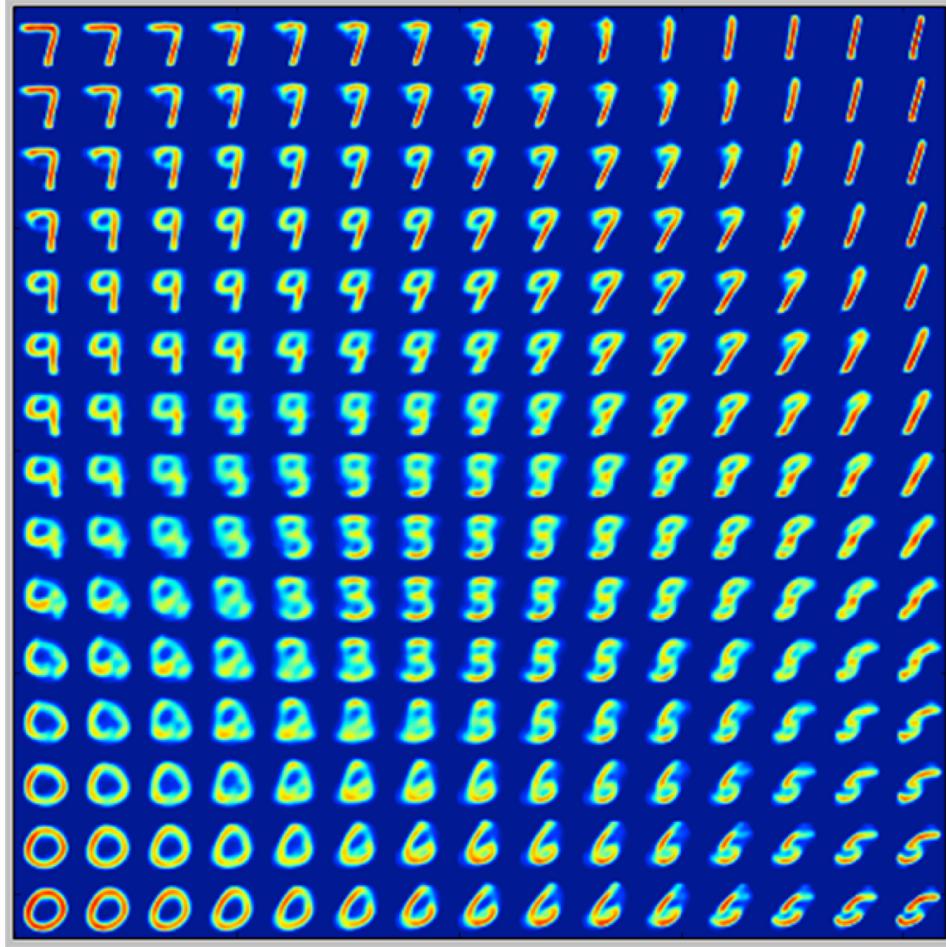
FIGURA 5.4: Detecciones y etiquetas sobre una imagen del conjunto de test 1. Recordar que los puntos considerados como verdaderos son el promedio entre las etiquetas de los dos doctores, marcados en verde.

# Generación sintética de datos



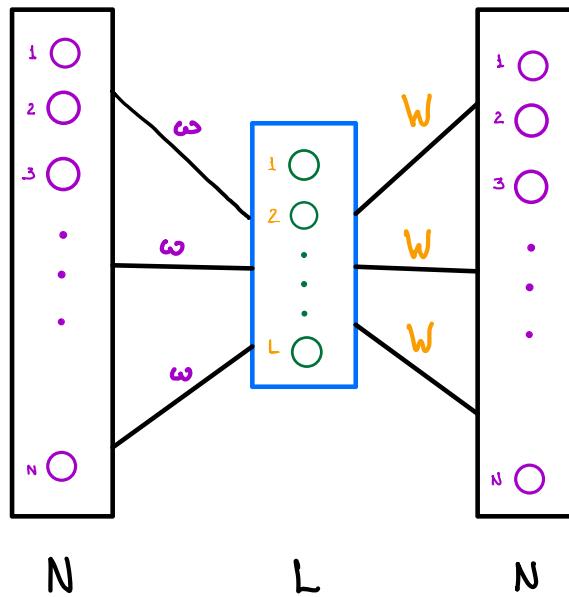


Imágenes sintéticas

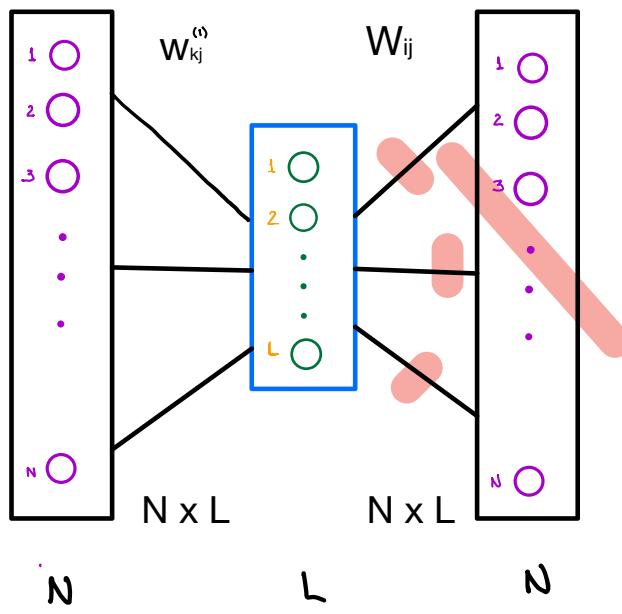


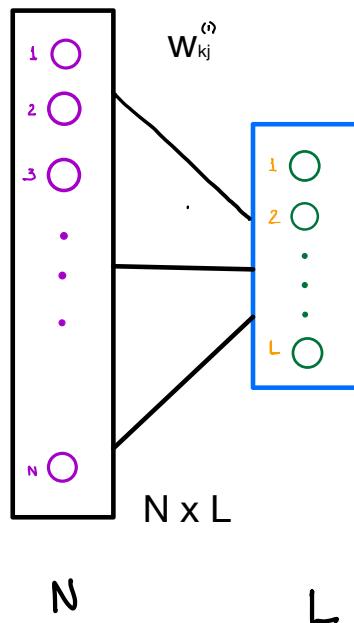
## Apilado de autoencoder

En lugar de elegir los parámetros sinápticos de nuestra red feed forward vamos a pre entrenar cada capa de parámetros desde la entrada a la salida secuencialmente.



Entreno la red autoencoder con el conjunto de entrenamiento hasta que lo aprende suficientemente bien.

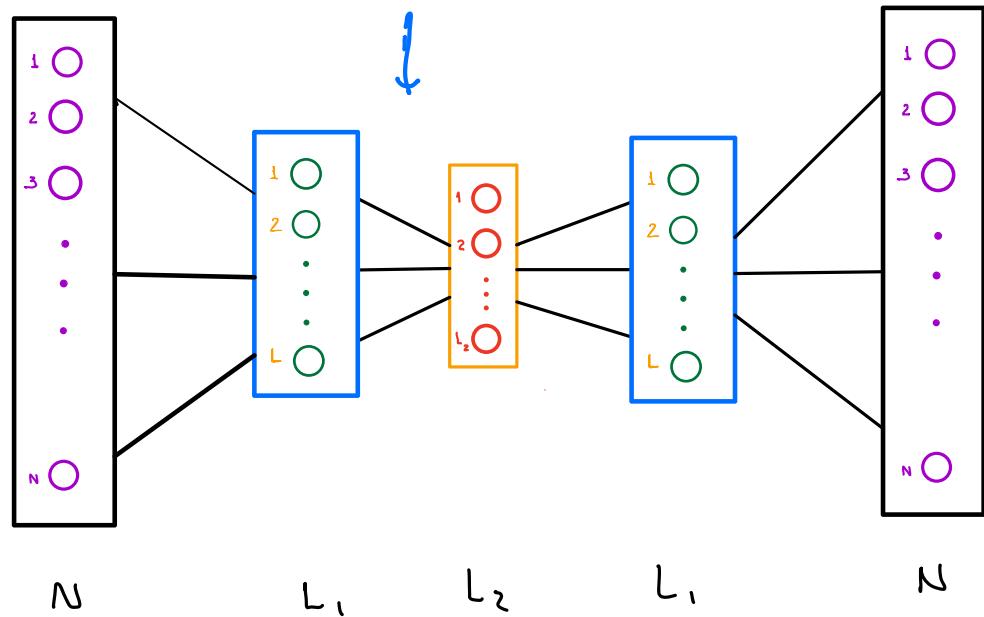
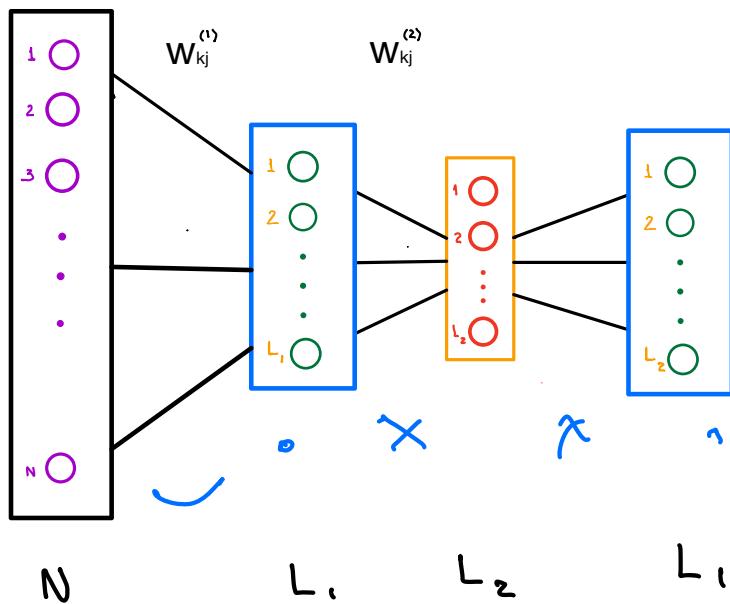




Así predeterminamos el valor inicial de las sinapsis que van de la capa de entrada a la primera capa oculta.

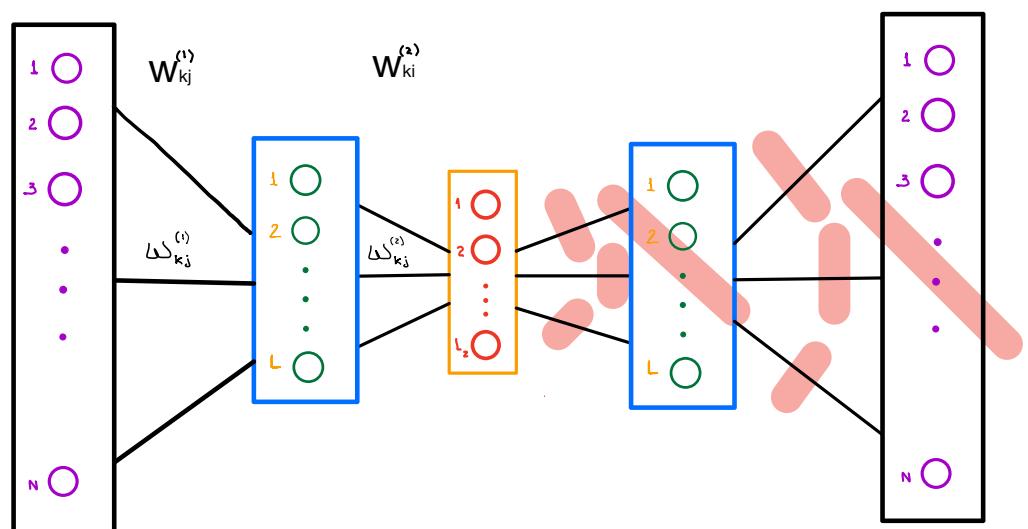
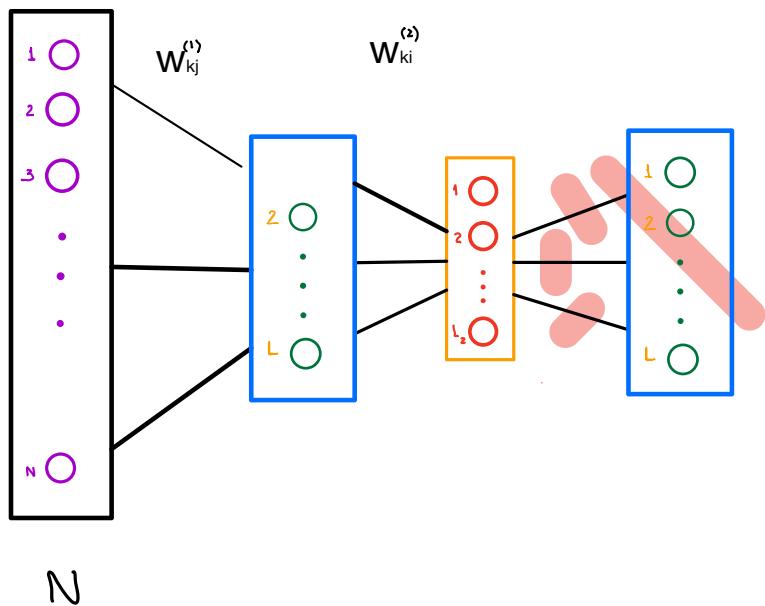
En realidad al principio eran aleatorias, solo que antes de aplicar back propagation para resolver nuestro problema de interés, entrenamos esa capa para que sea la capa de codificación de un autoencoder.

Usando el mismo procedimiento podemos elegir valores iniciales de la segunda capa de sinapsis.

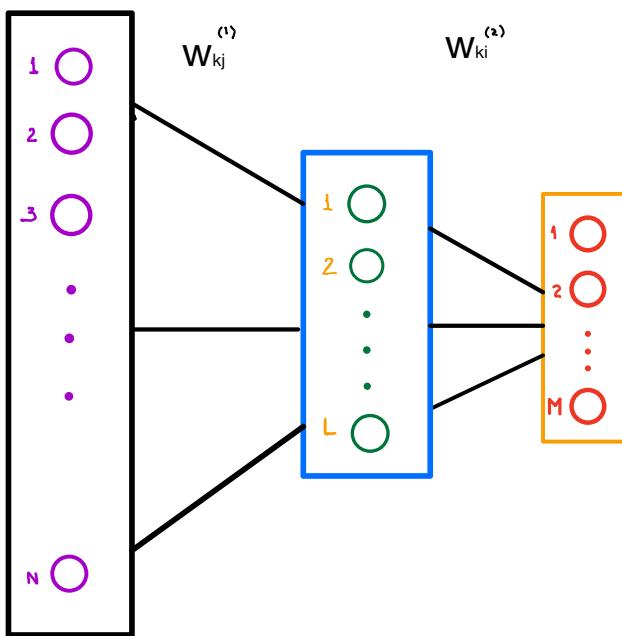


Acá solo actualizamos la segunda capa del encoder y todo el decoder.

Ahora eliminamos todo e decoder y ya tenemos valores de partida para entrenar nuestra red feed forward.



En este proceso mantenemos fijos los valores  $W_{ki}^{(1)}$ .



Ahora tenemos las dos primeras capas de nuestra red feed forward pre-entrenadas para autocodificar la información contenida en el dataset.

De esta forma podremos pre-entrenar cada capa de sinapsis y umbrales en nuestra red neuronal de muchas capas. Esto mejora muchísimo el aprendizaje de la red profunda. En otras palabras, si pre-entrenamos, para un mismo número de épocas tendremos menor error de aprendizaje  $E_{in}$  (loss de aprendizaje).

El propio **Geoffrey Hinton**, desde la Universidad de Toronto, quien en la década de los ochenta del siglo pasado inventara el algoritmo de Back-propagation, ideó el método de pre-procesar las sinapsis y ya no comenzar el algoritmo a partir de sinapsis y umbrales aleatorios. En verdad los parámetros son aleatorios en cada autoencoder, pero no en el aprendizaje del problema particular que queremos tratar. Llamó a estas redes **Deep Belief Networks** y lo publicaron 2006.

## A Fast Learning Algorithm for Deep Belief Nets

**Geoffrey E. Hinton**

*hinton@cs.toronto.edu*

**Simon Osindero**

*osindero@cs.toronto.edu*

*Department of Computer Science, University of Toronto, Toronto, Canada M5S 3G4*

**Yee-Whye Teh**

*tehyw@comp.nus.edu.sg*

*Department of Computer Science, National University of Singapore,  
Singapore 117543*

We show how to use “complementary priors” to eliminate the explaining-away effects that make inference difficult in densely connected belief nets that have many hidden layers. Using complementary priors, we derive a fast, greedy algorithm that can learn deep, directed belief networks one layer at a time, provided the top two layers form an undirected associative memory. The fast, greedy algorithm is used to initialize a slower learning procedure that fine-tunes the weights using a contrastive version of the wake-sleep algorithm. After fine-tuning, a network with three hidden layers forms a very good generative model of the joint distribution of handwritten digit images and their labels. This generative model gives better digit classification than the best discriminative learning algorithms. The low-dimensional manifolds on which the digits lie are modeled by long ravines in the free-energy landscape of the top-level associative memory, and it is easy to explore these ravines by using the directed connections to display what the associative memory has in mind.

### 1 Introduction

---

Learning is difficult in densely connected, directed belief nets that have many hidden layers because it is difficult to infer the conditional distribu-

Esta publicación es considerada el punto de partida del **aprendizaje profundo**, que iniciaría un vertiginoso cambio en la calidad de la algorítmica neuronal aplicada a aprendizaje supervisados.

igurations. The measured SHG sonance in Fig. tion), we find bove the noise signal closely incident power SHG emission null angle with eviations from ie SRRs (see Small detuning ler wavelength uces the SHG 20%. For ex- with vertical ve find a small For excitation zontal incident but significant is again po-

---

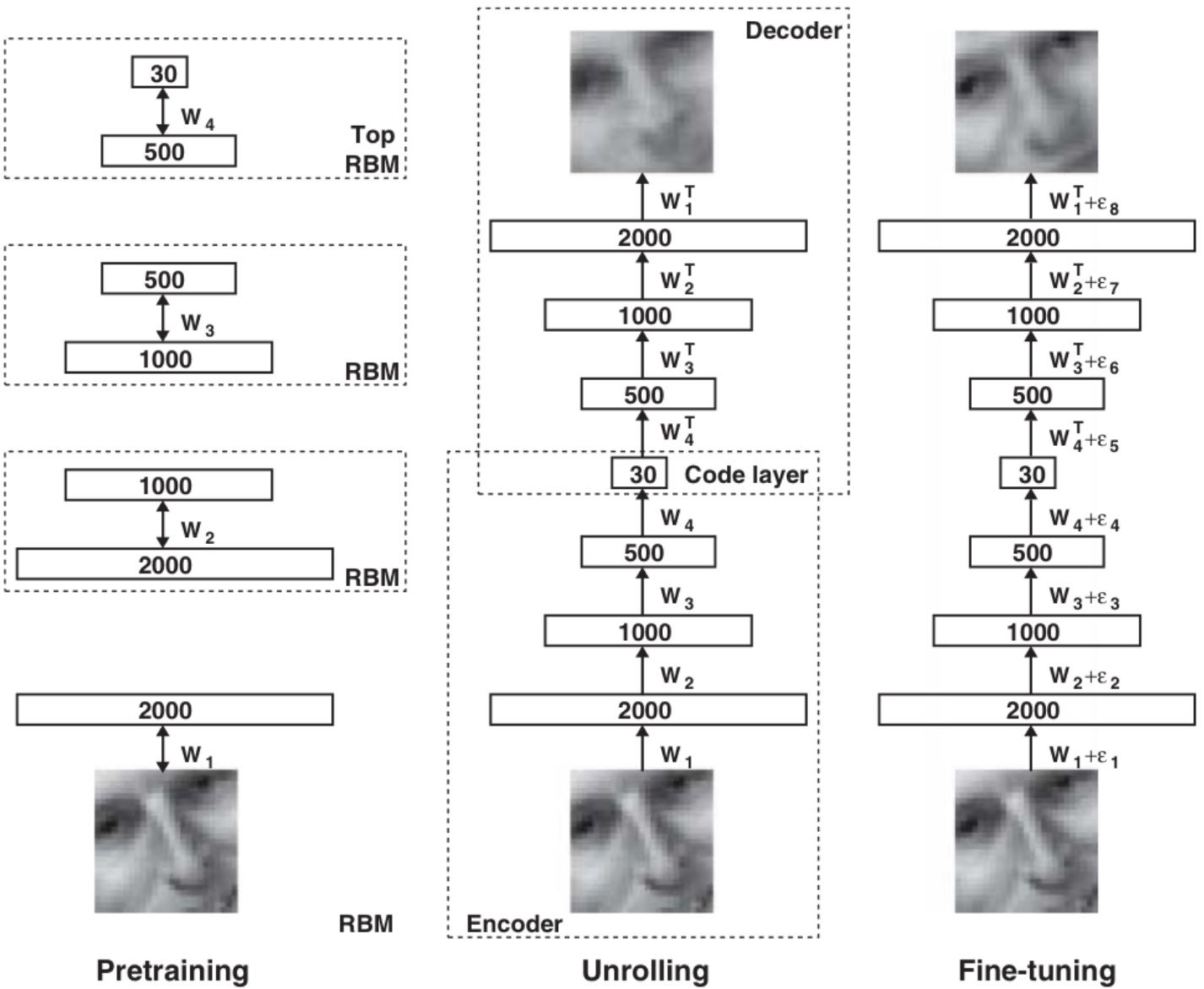
# Reducing the Dimensionality of Data with Neural Networks

G. E. Hinton\* and R. R. Salakhutdinov

High-dimensional data can be converted to low-dimensional codes by training a multilayer neural network with a small central layer to reconstruct high-dimensional input vectors. Gradient descent can be used for fine-tuning the weights in such “autoencoder” networks, but this works well only if the initial weights are close to a good solution. We describe an effective way of initializing the weights that allows deep autoencoder networks to learn low-dimensional codes that work much better than principal components analysis as a tool to reduce the dimensionality of data.

Dimensionality reduction facilitates the classification, visualization, communication, and storage of high-dimensional data. A simple and widely used method is principal components analysis (PCA), which

finds the directions of greatest variance in the data set and represents each data point by its coordinates along each of these directions. We describe a nonlinear generalization of PCA that uses an adaptive, multilayer “encoder” network



**Fig. 1.** Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the “data” for training the next RBM in the stack. After the pretraining, the RBMs are “unrolled” to create a deep autoencoder, which is then fine-tuned using backpropagation of error derivatives.

En el paper original no usaron autoencoders sino otro tipo de máquinas de aprendizaje supervisado llamadas máquinas de Boltzmann, las cuales no veremos en este curso. Son redes recurrentes, con sinapsis simétricas y con motivación física. Son máquinas que aprenden a que la red visite cada posible configuración con determinada probabilidad.

# Why Does Unsupervised Pre-training Help Deep Learning?

**Dumitru Erhan\***

**Yoshua Bengio**

**Aaron Courville**

**Pierre-Antoine Manzagol**

**Pascal Vincent**

*Département d'informatique et de recherche opérationnelle*

*Université de Montréal*

*2920, chemin de la Tour*

*Montréal, Québec, H3T 1J8, Canada*

DUMITRU.ERHAN@UMONTREAL.CA

YOSHUA.BENGIO@UMONTREAL.CA

AARON.COURVILLE@UMONTREAL.CA

PIERRE-ANTOINE.MANZAGOL@UMONTREAL.CA

PASCAL.VINCENT@UMONTREAL.CA

**Samy Bengio**

*Google Research*

*1600 Amphitheatre Parkway*

*Mountain View, CA, 94043, USA*

BENGIO@GOOGLE.COM

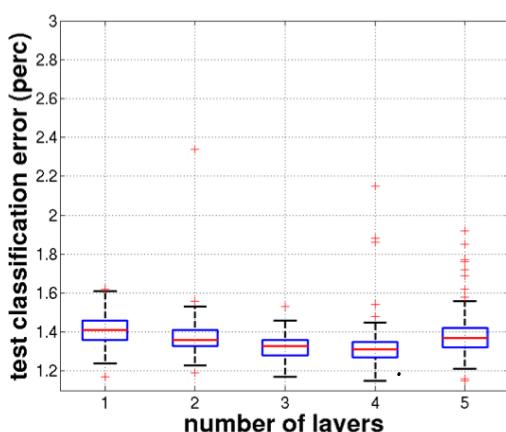
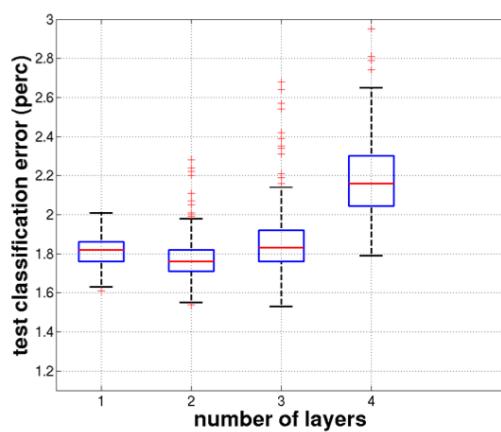


Figure 1: Effect of depth on performance for a model trained (**left**) without unsupervised pre-training and (**right**) with unsupervised pre-training, for 1 to 5 hidden layers (networks with 5 layers failed to converge to a solution, without the use of unsupervised pre-training). Experiments on MNIST. Box plots show the distribution of errors associated with 400 different initialization seeds (top and bottom quartiles in box, plus outliers beyond top and bottom quartiles). Other hyperparameters are optimized away (on the validation set). *Increasing depth seems to increase the probability of finding poor apparent local minima.*

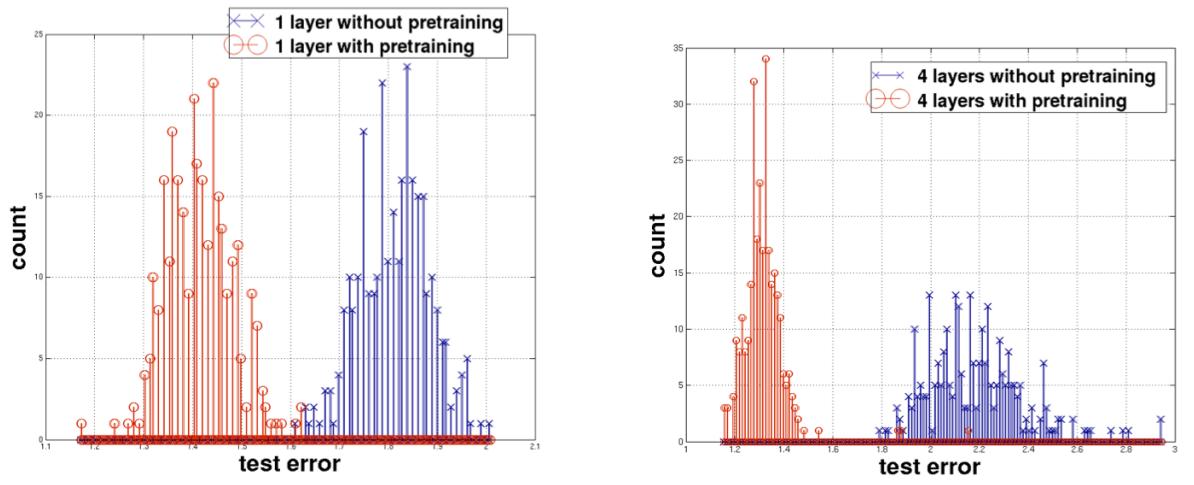


Figure 2: Histograms presenting the test errors obtained on MNIST using models trained with or without pre-training (400 different initializations each). **Left:** 1 hidden layer. **Right:** 4 hidden layers.

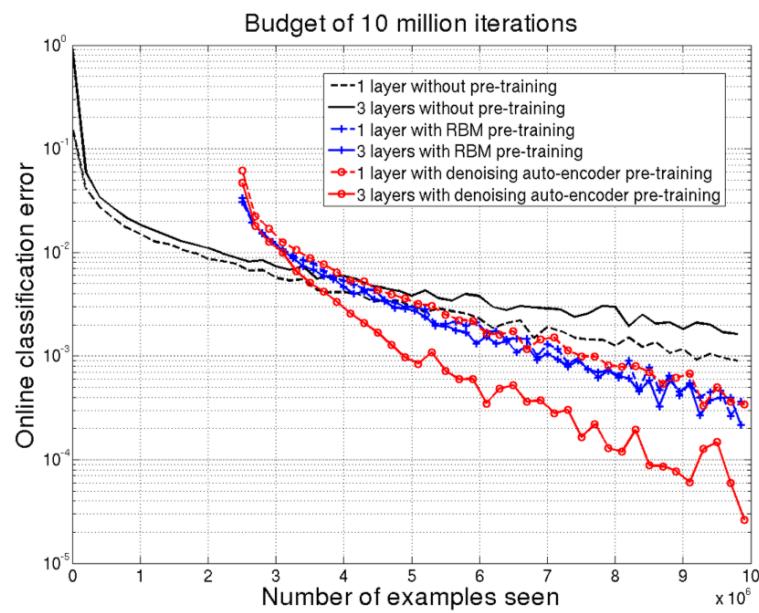


Figure 11: Comparison between 1 and 3-layer networks trained on InfiniteMNIST. Online classification error, computed as an average over a block of last 100,000 errors.

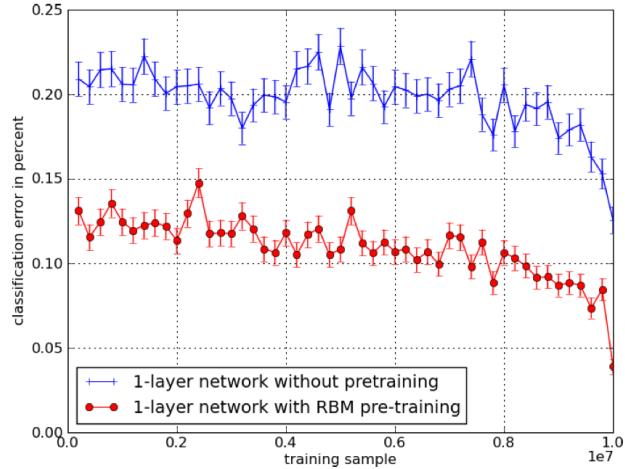


Figure 12: Error of 1-layer network with RBM pre-training and without, on the 10 million examples used for training it. The errors are calculated in the same order (from left to right, above) as the examples were presented during training. Each error bar corresponds to a block of consecutive training examples.

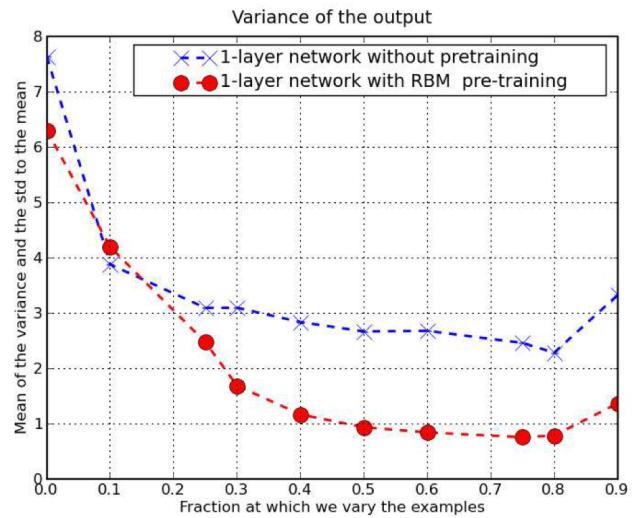


Figure 13: Variance of the output of a trained network with 1 layer. The variance is computed as a function of the point at which we vary the training samples. Note that the 0.25 mark corresponds to the start of pre-training.

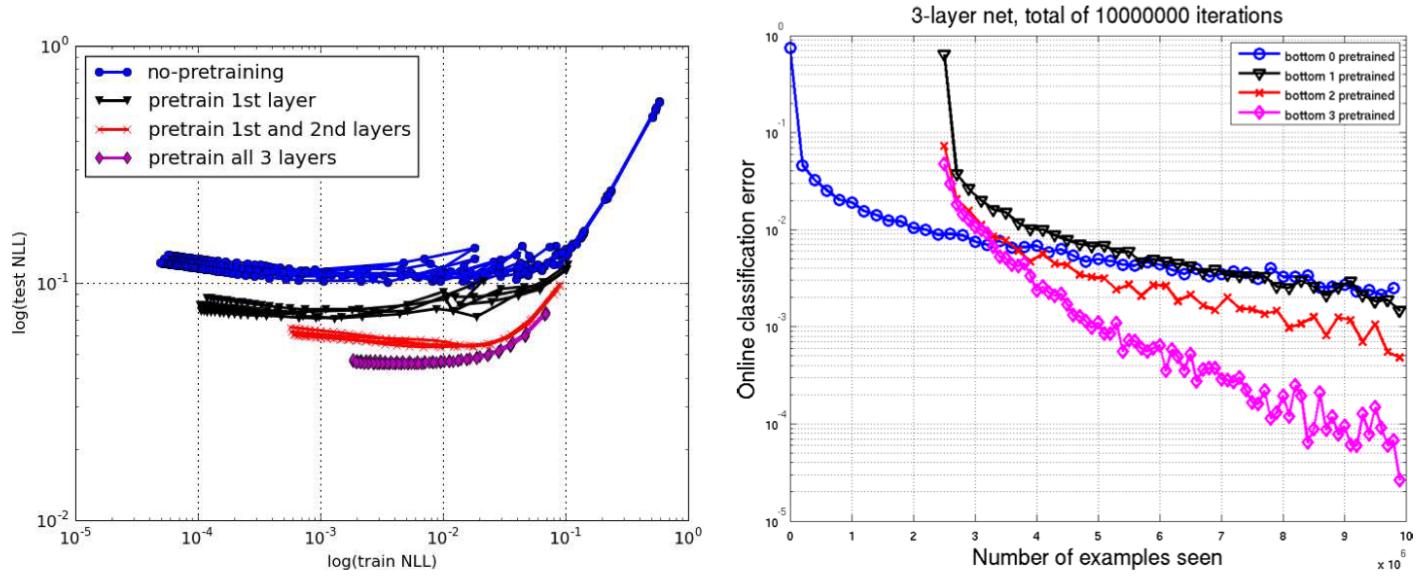


Figure 14: *On the left:* for MNIST, a plot of the  $\log(\text{train NLL})$  vs.  $\log(\text{test NLL})$  at each epoch of training. We pre-train the first layer, the first two layers and all three layers using RBMs and randomly initialize the other layers; we also compare with the network whose layers are all randomly initialized. *On the right:* InfiniteMNIST, the online classification error. We pre-train the first layer, the first two layers or all three layers using denoising auto-encoders and leave the rest of the network randomly initialized.

