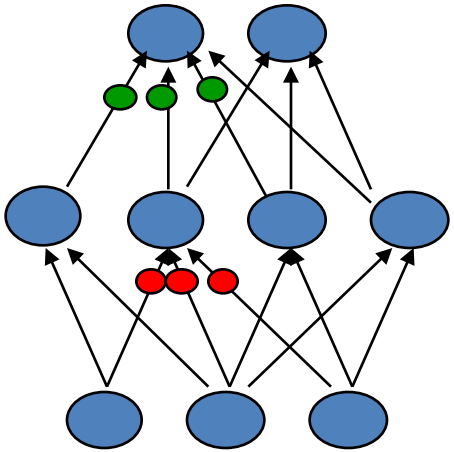
Algoritmo de Backpropagation

Basadas en las diapositivas de
Geoffrey Hinton

Aprendizaje por perturbación de pesos

Perturbar aleatoriamente un peso, y si mejora el desempeño, mantener la perturbación.

- Forma muy primitiva de aprendizaje por refuerzo.
- Muy ineficiente.
- Al final del aprendizaje, ¡Los cambios solo pueden emeorar las cosas!.

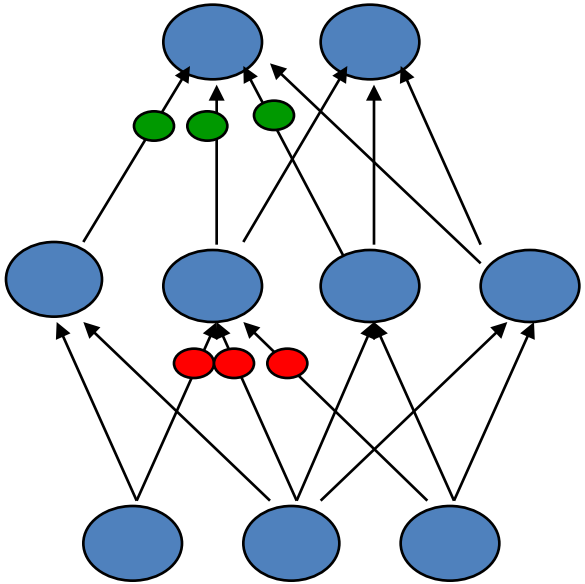


Alternativa

- Perturbar aleatoriamente todos los pesos en paralelo, y correlacionar los cambios con el desempeño, para mantener los más prometedores.

- Perturbar aleatoriamente las activaciones de las neuronas.

  – Si sabemos que queremos que una activación sea modificada, entonces tenemos una idea clara de como modificar los pesos de la neurona.



La idea detrás del algoritmo de backpropagation

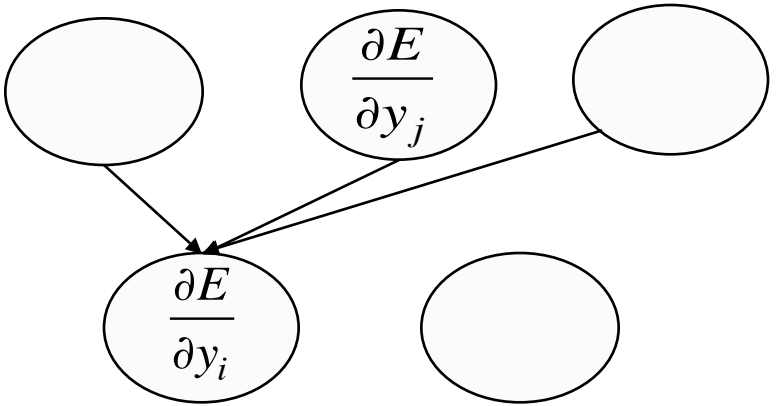
- No sabemos que queremos que hagan las neuronas ocultas, pero podemos calcular que tan rápido cambia el error, si nosotros modificamos la actividad de la neurona.
- En lugar de usar los valores de activación perturbados, usamos la variación del error respecto a las activaciones de las unodades ocultas.
- Cada activación de una neurona oculta afecta a muchas neuronas y por lo tanto tiene muchos efectos separados. Estos efectos deben de combinarse.
- El algoritmo de *b-prop* propone una forma eficiente de calcular la derivada del error para todas las neuronas ocultas en un solo paso.

Esquema básico del b-prop

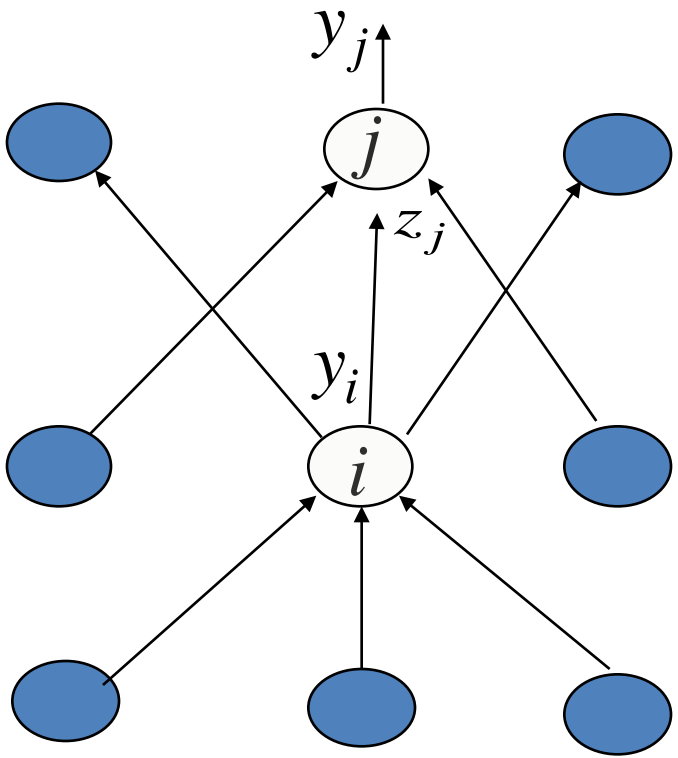
1. Calcula el error de cada neurona de salida.
2. Calcula la derivada del error respecto a cada activación de las neuronas de salida.
3. Calcula la derivada del error respecto a la activación de las neuronas de la capa anterior a la de salida.
4. Continua calculando las derivadas por capas, hasta la última capa oculta
5. Usa estas derivadas para calcular la derivada del error respecto a los pesos.

$$E = \frac{1}{2} \sum_{j \in output} (t_j - y_j)^2$$

$$\frac{\partial E}{\partial y_j} = -(t_j - y_j)$$



Calculo de dE/dy



$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j}\frac{\partial E}{\partial y_j} = y_j(1-y_j)\frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dz_j}{dy_i}\frac{\partial E}{\partial z_j} = \sum_j w_{ij}\frac{\partial E}{\partial z_j}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial E}{\partial z_j} = y_i\frac{\partial E}{\partial z_j}$$

Del calculo de derivadas al aprendizaje

- El b-prop es un algoritmo eficiente para calcular la derivada del error respecto a un peso dE/dw para todos los pesos, en un único caso de entrenamiento.

- Para tener un procedimiento de aprendizaje completo, todavía falta tomar una buena cantidad de decisiones sobre como usar dichas derivadas.

  - Optimización: ¿Como usar las derivadas en casos individuales para encontrar un buen conjunto de pesos (o al menos aceptables)?

  - Generalización: ¿Como asegurarnos que los pesos encontrados van a tener un comportamiento aceptable en casos que no se vieron en el aprendizaje?

Optimización: Los problemas principales

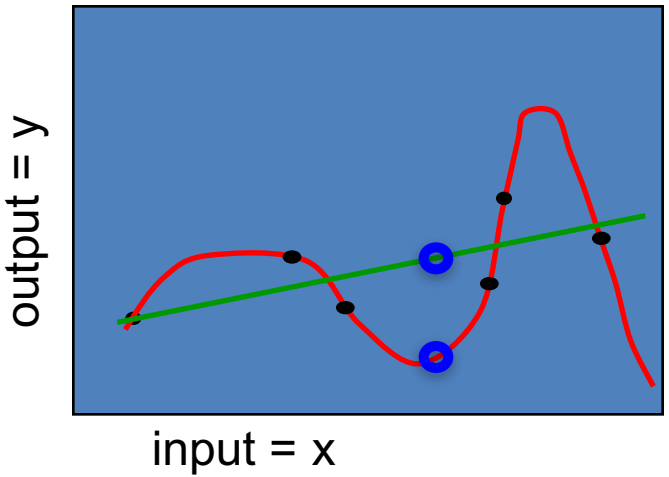
- Que tan seguido actualizar los pesos:
  - Online: después de cada caso de entrenamiento.
  - Full batch: después de todos los ejemplos de entrenamiento.
  - Mini-batch: después de un conjunto de casos de entrenamiento.

- Cuanto actualizar los pesos:
  - Usar una tasa de aprendizaje fija.
  - Adaptar la tasa de aprendizaje en forma global.
  - Adaptar la teasa de aprendizaje por peso o neurona.
  - Usar un método sin tasa de aprendizaje.

Sobreajuste: El precio a pagar por un modelo poderoso

- Los datos de entrenamiento contienen información valiosa sobre las regularidades en la relacion que existe entre las entradas y las salidas de un proceso desconocido. Sin embargo tambien contiene dos tipos importantes de ruido:
  - Los valores de salida no pueden alcanzarse a partir de los valores de entrada (falta de información).
  - Regularidades que existen en los datos pero que no dependen del proceso, si no que provienen de la forma en que se escogieron los ejemplos. Esto se conoce como error de muestreo.
- Cuando ajustamos un modelo, no podemos avisarle al metodo de aprendizaje cuales son las regularidades debidas al proceso, y cuales son error de muestreo.
- Si el modelo es muy flexible, esto termina siendo un desastre.

Ejemplo de sobreaprendizaje



- ¿En cual modelo confiar?
  - El modelo complicado ajusta mejor los datos.
  - No es económico.

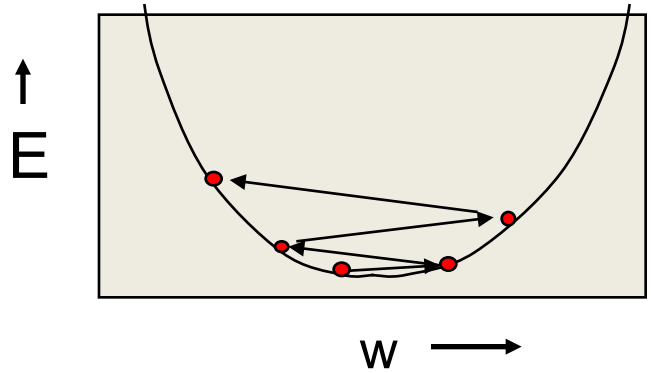
- Un modelo es convincente si ajusta un monton de datos sorprendentemente bien.
  - No es sorprendente que un modelo complicado ajuste pocos datos.

Medios para reducir el sobreaprendizaje

- Weight-decay
- Weight-sharing
- Early stopping
- Model averaging
- Bayesian fitting of neural nets
- Dropout
- Generative pre-training

¿Como el aprendizaje puede ir mal?

- Si la tasa de aprendizaje es grande, los valores de los pesos pueden cambiar de manera muy importante, y esto puede hacer que el valor de los pesos aumente.
- Hay que moverse rápido en direcciones con gradientes pequeños pero consistentes.
- Hay que moverse lento en direcciones con gradientes grandes pero inconsistentes.



Gradiente estocástico

- Si el conjunto de datos es muy redundante, el gradiente de una parte del conjunto es casi el mismo que el del conjunto completo.
- En lugar de calcular el gradiente con todos los datos, solo calculamos el gradiente con una parte de estos y ajustamos los pesos.
- La versión extrema de esta idea es calcular los pesos despuesde un solo dato. A esto se le conoce como aprendizaje on line.
- Mini-batch es en general mejor que online.
- Menor numero de calculos para actualizar los pesos.
- Al calcular el gradiente para un conjunto de datos es posible usar operaciones matriciales que pueden ser implementadas muy eficientemente en GPUs.
- Tienen que balancearse por clases los mini-batches.

Dos tipos de métodos de aprendizaje

- Si se utiliza el método de aprendizaje en modo batch (todos los datos) existen muchos métodos sofisticados para realizar la optimización.

- Los métodos deben de adaptarse a su uso para problemas de redes neuronales.

- Para problemas con conjuntos de entrenamiento muy grandes y redundantes, casi siempre el mejor método es el de minibatch.

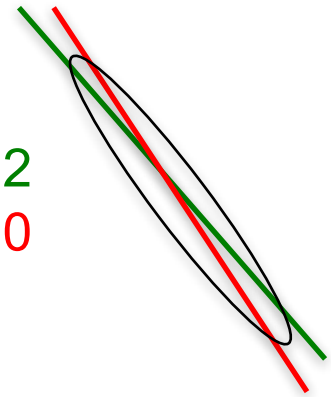
- Minibatches grandes son más eficientes computacionalmente.

Inicialización de pesos

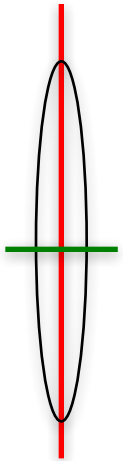
- Inicializar con valores diferentes y aleatorios todas las neuronas
- Si una neurona tiene muchas entradas, pequeños cambios en cada entrada pueden hacer grandes cambios en la neurona.
- Hay que escalar los pesos, generalmente a la raíz cuadrada del número de entradas.

Normalización de entradas

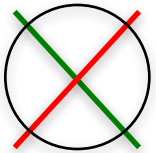
101, 101 → 2
101, 99 → 0



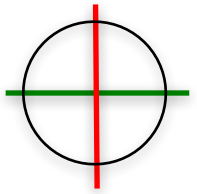
0.1, 10 → 2
0.1, -10 → 0



1, 1 → 2
1, -1 → 0



1, 1 → 2
1, -1 → 0



Todavía mejor: decorrelacionar entradas

- Para las redes neuronales si se decorrelacionan las entradas, los algoritmos (todos) se desempeñan mucho mejor.

- Existen muchas técnicas pero el Análisis en Componentes Principales (PCA) es el más usual.

- Se puede reducir la dimensionalidad de la entrada, ignorando los componentes con una varianza pequeña.

La tasa de aprendizaje

1. Suponer una tasa de aprendizaje inicial.

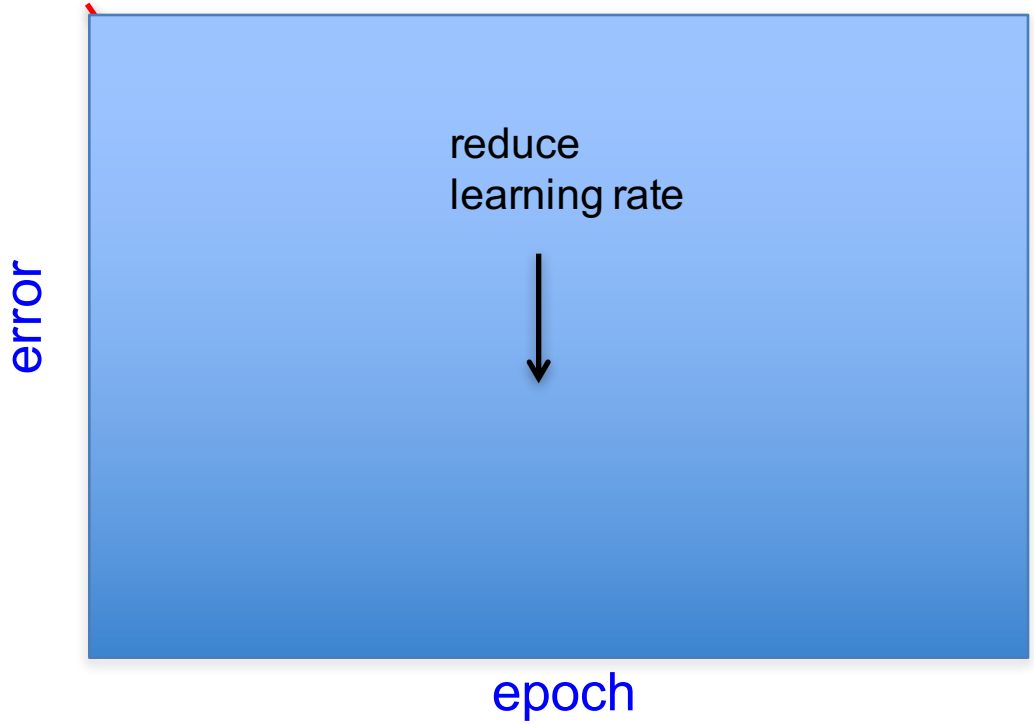
2. Si el costo aumenta u oscila, reducir la tasa de aprendizaje.

3. Si el error se reduce, consistentemente pero despacio, aumentar la tasa de aprendizaje.

4. Al final de el aprendizaje con minibatch, reducir la tasa de aprendizaje.

5. Reducir la tasa de aprendizaje cuando el error deje de decrecer.

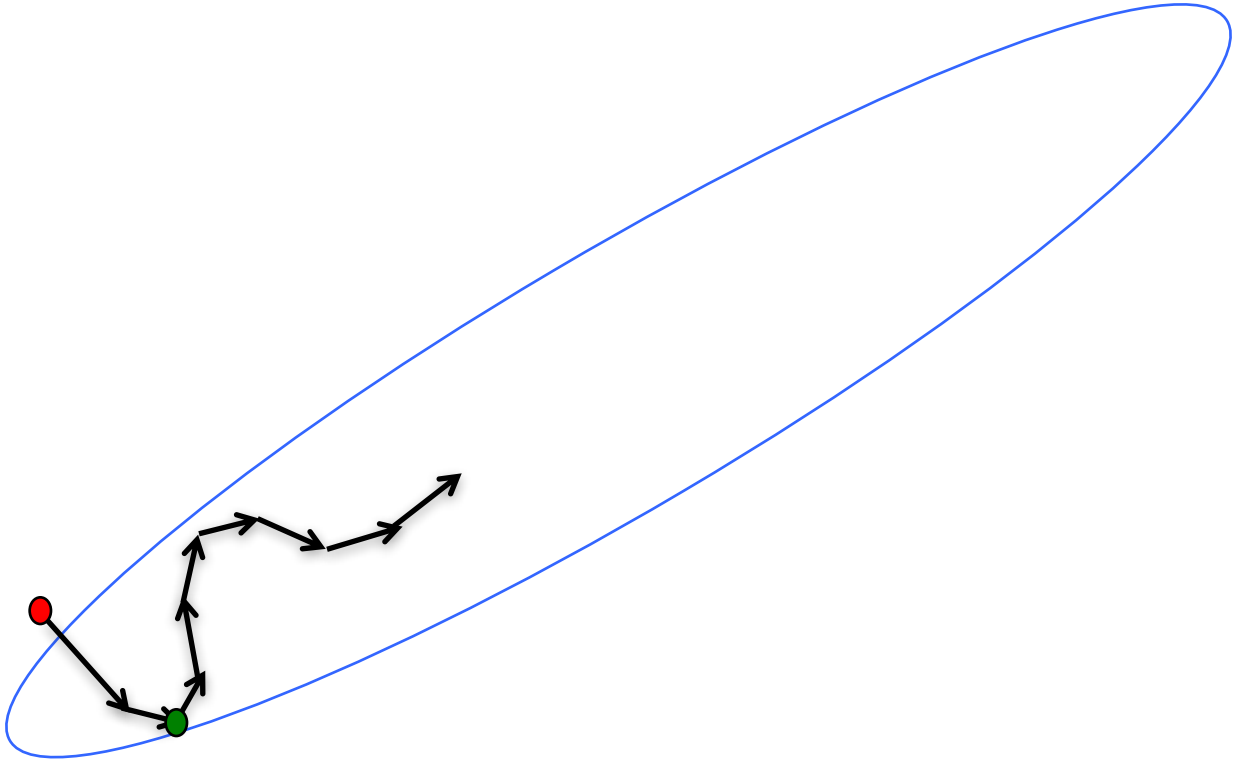
¡Cuidado con la tasa de aprendizaje!



Cuatro formas de ajustar la tasa de aprendizaje

- Usar el método de momento o inercia.
- Usar un método de ajuste adaptivo de la tasa de aprendizaje por cada parámetro (rprop).
- Dividir la tasa de aprendizaje de cada peso por el promedio de las magnitudes de los gradientes recientes (rmsprop).
- Usar un método recientemente desarrollado (típicamente por Hitton, Bengio o LeCun).

Intuición del método de momento



Equaciones del metodo de momento

$$\mathbf{v}(t) = \alpha\, \mathbf{v}(t-1) - \varepsilon \frac{\partial E}{\partial \mathbf{w}}(t)$$

El gradiente actual incrementa la velocidad. La velocidad anterior decae en una proporción α lijeramente menor que 1.

$$\Delta \mathbf{w}(t) = \mathbf{v}(t)$$

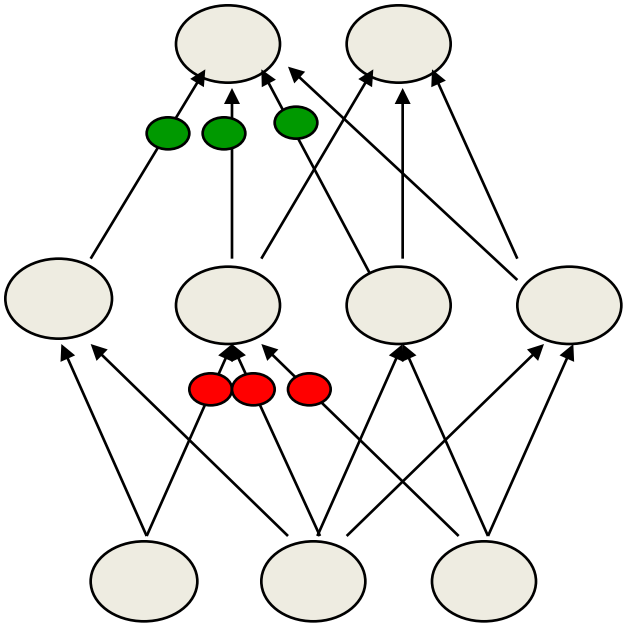
El incremento en el peso es proporcional a la velocidad actual.

$$= \alpha\, \mathbf{v}(t-1) - \varepsilon \frac{\partial E}{\partial \mathbf{w}}(t)$$

$$= \alpha\, \Delta \mathbf{w}(t-1) - \varepsilon \frac{\partial E}{\partial \mathbf{w}}(t)$$

El cambio de peso se puede explicar en funcion de la derivada del costo actual y el incremento anterior.

Intuición de los método de ajuste por peso



- Los gradientes pueden ser muy pequeños en las primeras capas ocultas.
- El número de conexiones a la entrada varía mucho entre capas.

Esquema general del método

$$\Delta w_{ij} = -\varepsilon \, g_{ij} \, \frac{\partial E}{\partial w_{ij}}$$

$$\textit{if} \quad \left(\frac{\partial E}{\partial w_{ij}}(t) \frac{\partial E}{\partial w_{ij}}(t-1) \right) > 0$$

$$\textit{then} \quad g_{ij}(t) = g_{ij}(t-1) + .05$$

$$\textit{else} \quad g_{ij}(t) = g_{ij}(t-1) * .95$$

rprop: Solo usa el signo del gradiente

- Combina la idea de usar solo el signo del gradiente con la idea de adaptar la tasa de aprendizaje en forma individual.
- Si el gradiente es positivo, incrementa la tasa de aprendizaje multiplicativamente (por ejemplo 1.2).
- En otro caso, decrementa la tasa de aprendizaje en forma multiplicativa (por ejemplo 0.5).
- Limita la tasa para ser siempre menos de 50 y mas que 0.001.

rmsprop: A mini-batch version of rprop

- El problema con R-prop aplicado a mini-batches, es que se modifica la tasa de aprendizaje por un número diferente por cada minibatch, sin tomar en cuenta cuando los minibatch son adyacentes. Lo que hace que el método sea muy inestable.
- ¿Por qué no forzar a que el valor de tasa de aprendizaje sea similar entre minibatches consecutivos?
- rmsprop: Mantiene un promedio movil sobre el cuadrado del gradiente de cada peso.

$$MeanSquare(w,t) = 0.9\ MeanSquare(w,\ t-1) + 0.1\left(\partial E / \partial w(t)\right)^2$$

- Dividir el gradiente por $\sqrt{MeanSquare(w,\ t)}$ hace que el aprendizaje funcione mucho mejor.

Recapitulando

- Para pequeñas bases de datos (por ejemplo, decenas de miles), o bases de datos grandes pero sin redundancia, usar aprendizaje en modo batch con métodos de optimización sofisticados (LBFGS).

- Para grandes bases de datos, con mucha redundancia, utilizar mini-batch con algún metodo de ajuste de tasa de aprendizaje.

- Las redes neuronales difieren mucho entre ellas, por lo que no hay receta secreta.

- Algunos problemas requieren un ajuste muy preciso de los pesos.

- Algunos problesmas incluyen algunos casos en lo que no hay mucha información.