Tips for Deep Learning

Recipe of Deep Learning



Step 1: define a set of function

Step 2: goodness of function

Step 3: pick the best function

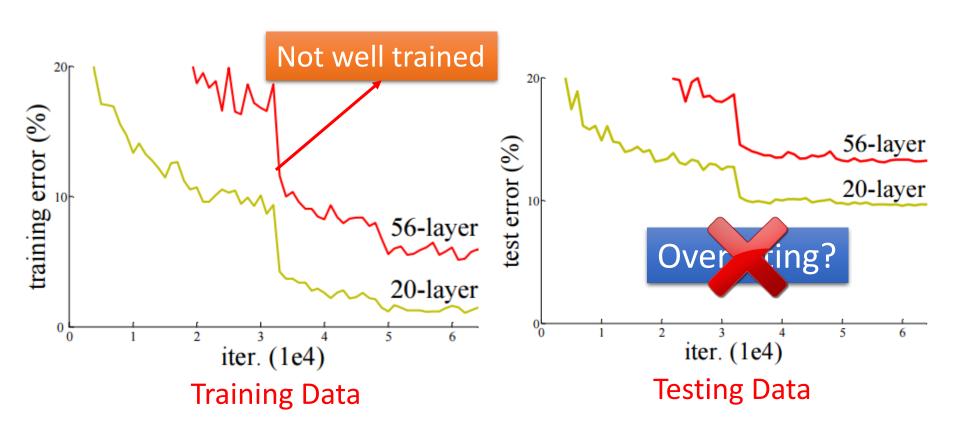
NO

NO

Overfitting!

Neural Network

Do not always blame Overfitting



Deep Residual Learning for Image Recognition http://arxiv.org/abs/1512.03385

Recipe of Deep Learning



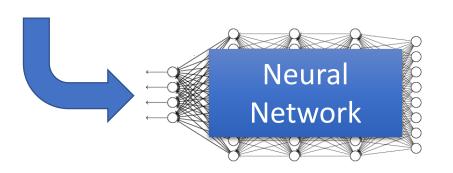
Different approaches for different problems.

e.g. dropout for good results on testing data

Good Results on Testing Data?



Good Results on Training Data?





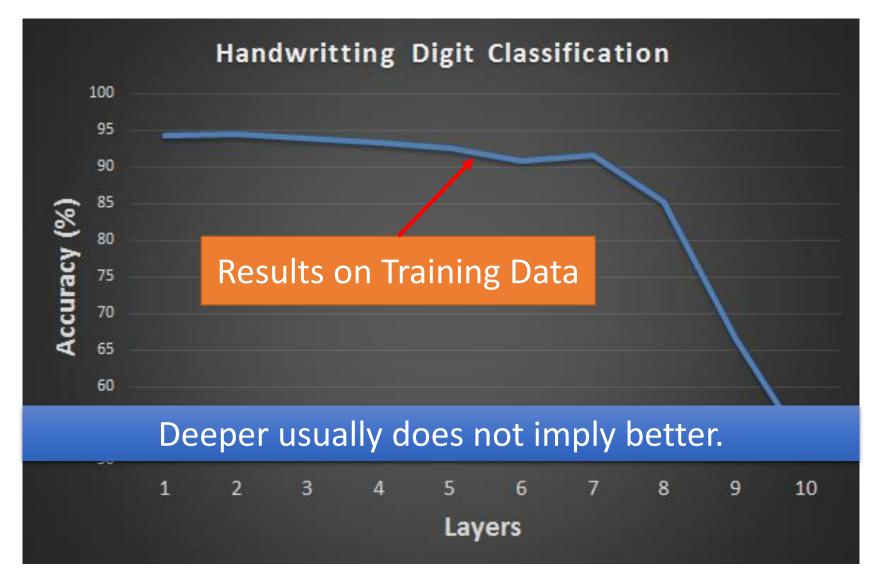
Recipe of Deep Learning YES **Early Stopping** Good Results on **Testing Data?** Regularization YES Dropout

New activation function

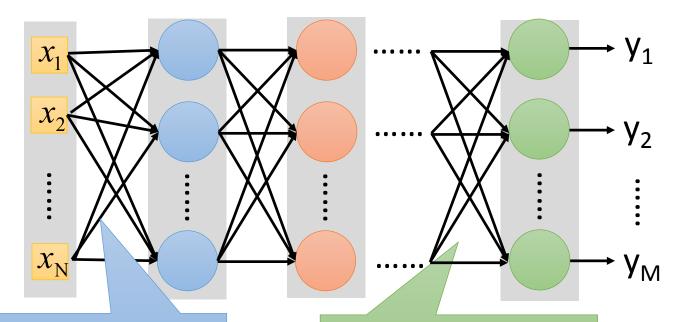
Adaptive Learning Rate

Good Results on Training Data?

Hard to get the power of Deep ...



Vanishing Gradient Problem



Smaller gradients

Learn very slow

Almost random

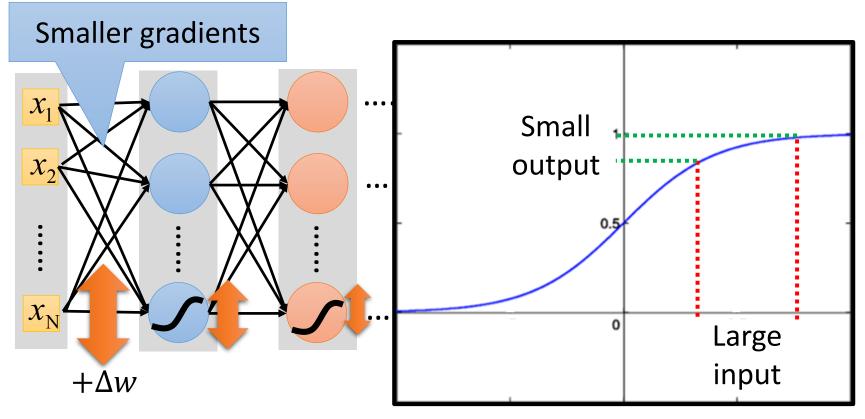
Larger gradients

Learn very fast

Already converge

based on random!?

Vanishing Gradient Problem

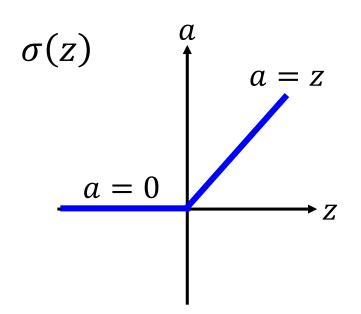


Intuitive way to compute the derivatives ...

$$\frac{\partial l}{\partial w} = ? \frac{\Delta l}{\Delta w}$$

ReLU

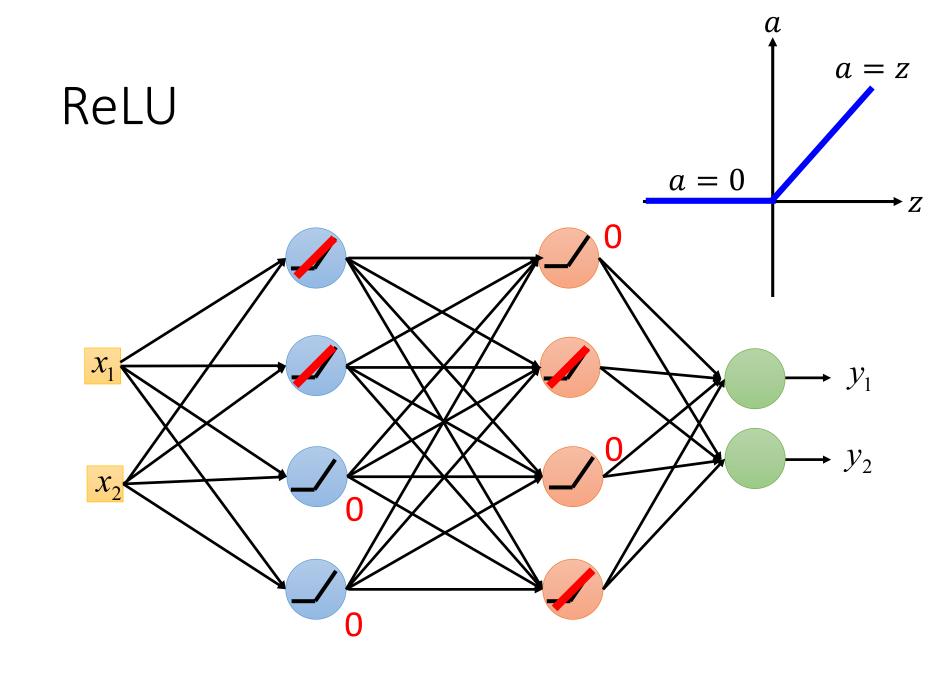
Rectified Linear Unit (ReLU)



[Xavier Glorot, AISTATS'11] [Andrew L. Maas, ICML'13] [Kaiming He, arXiv'15]

Reason:

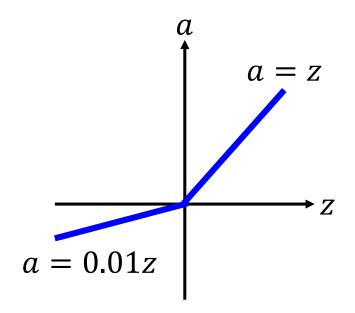
- 1. Fast to compute
- 2. Biological reason
- 3. Infinite sigmoid with different biases
- 4. Vanishing gradient problem



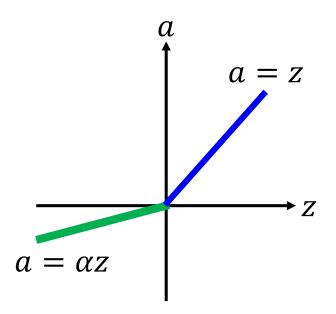
a = zReLU a = 0A Thinner linear network y_2 Do not have smaller gradients

ReLU - variant

Leaky ReLU



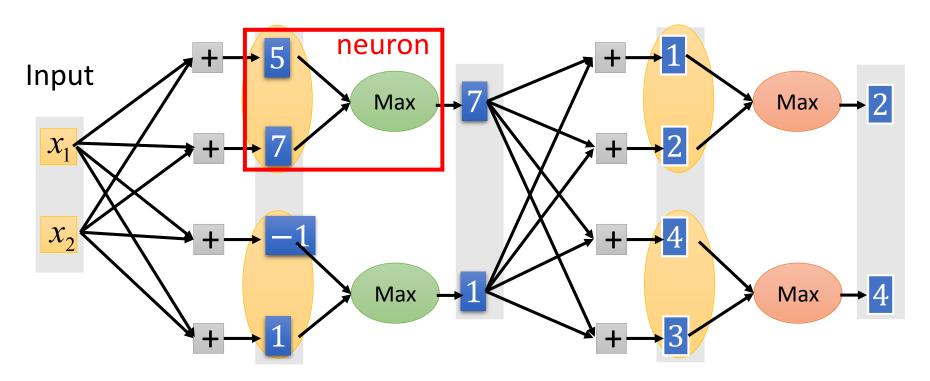
Parametric ReLU



α also learned by gradient descent

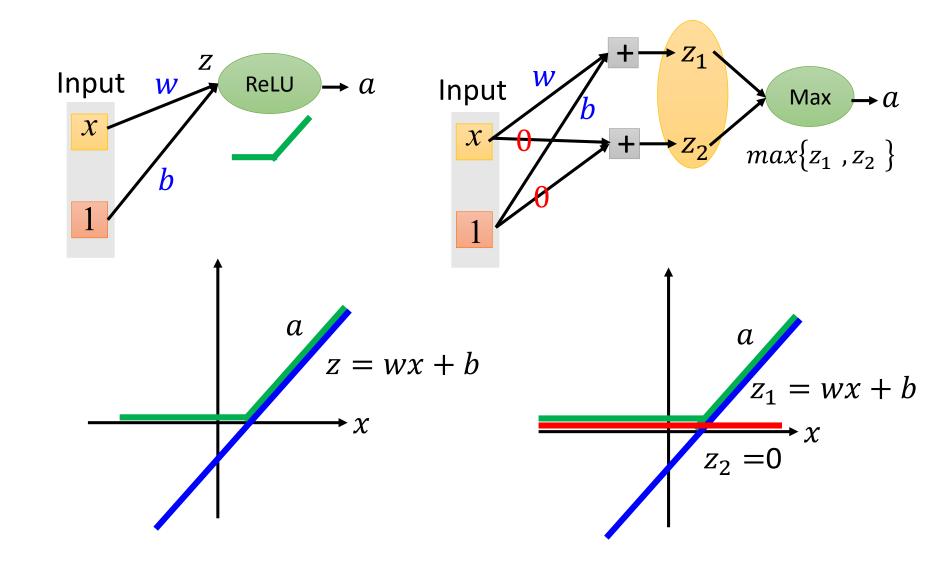
ReLU is a special cases of Maxout

• Learnable activation function [lan J. Goodfellow, ICML'13]

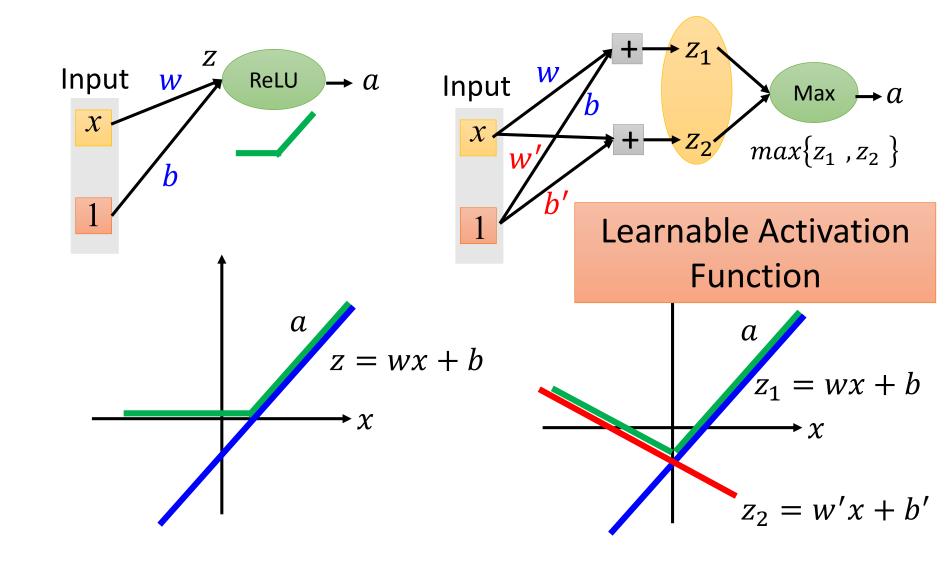


You can have more than 2 elements in a group.

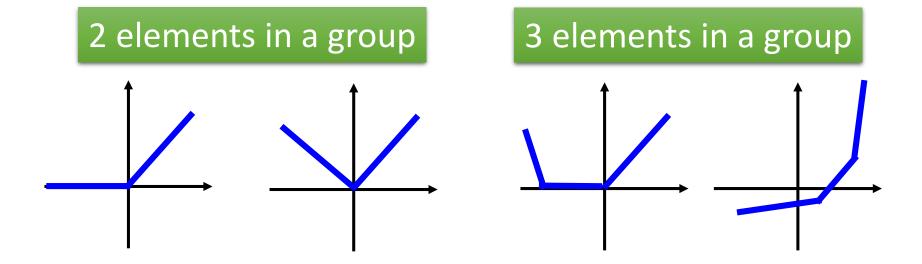
ReLU is a special cases of Maxout



More than ReLU

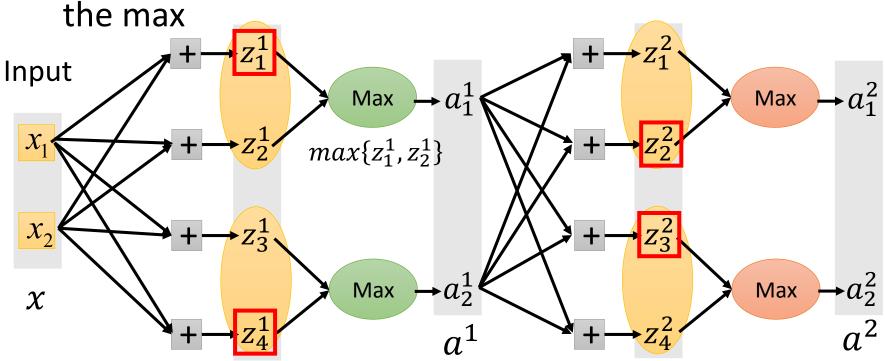


- Learnable activation function [lan J. Goodfellow, ICML'13]
 - Activation function in maxout network can be any piecewise linear convex function
 - How many pieces depending on how many elements in a group



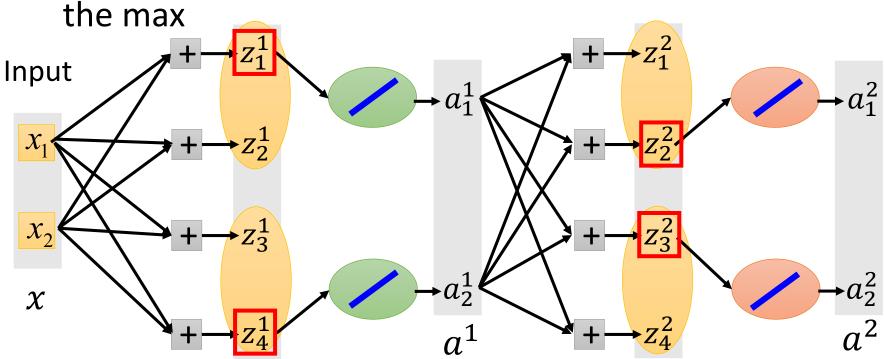
Maxout - Training

Given a training data x, we know which z would be



Maxout - Training

Given a training data x, we know which z would be



Train this thin and linear network

Different thin and linear network for different examples

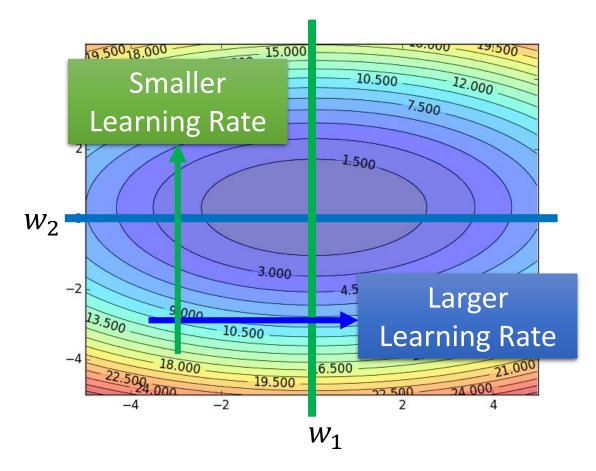
Recipe of Deep Learning YES **Early Stopping** Good Results on **Testing Data?** Regularization YES Dropout

New activation function

Adaptive Learning Rate

Good Results on Training Data?

Review



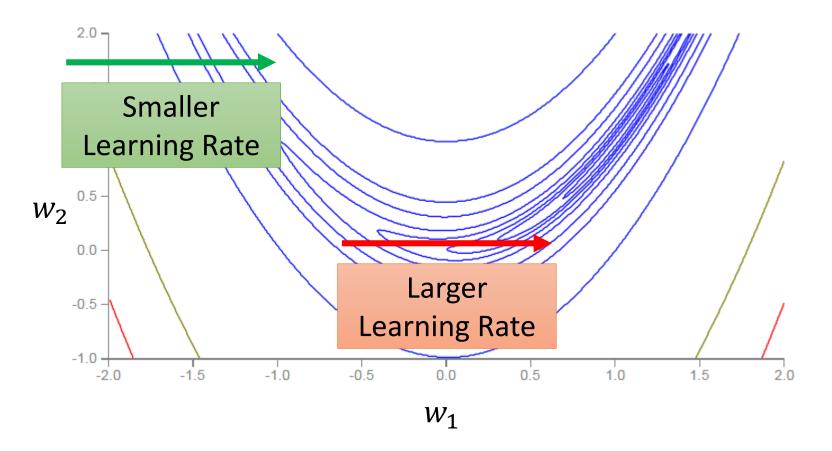
Adagrad

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^{t} (g^i)^2}} g^t$$

Use first derivative to estimate second derivative

RMSProp

Error Surface can be very complex when training NN.



RMSProp

$$w^{1} \leftarrow w^{0} - \frac{\eta}{\sigma^{0}} g^{0} \qquad \sigma^{0} = g^{0}$$

$$w^{2} \leftarrow w^{1} - \frac{\eta}{\sigma^{1}} g^{1} \qquad \sigma^{1} = \sqrt{\alpha(\sigma^{0})^{2} + (1 - \alpha)(g^{1})^{2}}$$

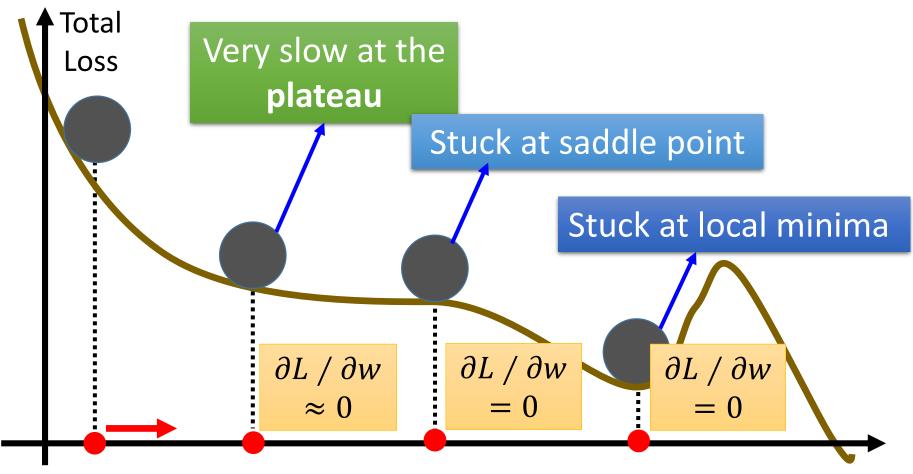
$$w^{3} \leftarrow w^{2} - \frac{\eta}{\sigma^{2}} g^{2} \qquad \sigma^{2} = \sqrt{\alpha(\sigma^{1})^{2} + (1 - \alpha)(g^{2})^{2}}$$

$$\vdots$$

 $w^{t+1} \leftarrow w^t - \frac{\eta}{\sigma^t} g^t$ $\sigma^t = \sqrt{\alpha(\sigma^{t-1})^2 + (1-\alpha)(g^t)^2}$

Root Mean Square of the gradients with previous gradients being decayed

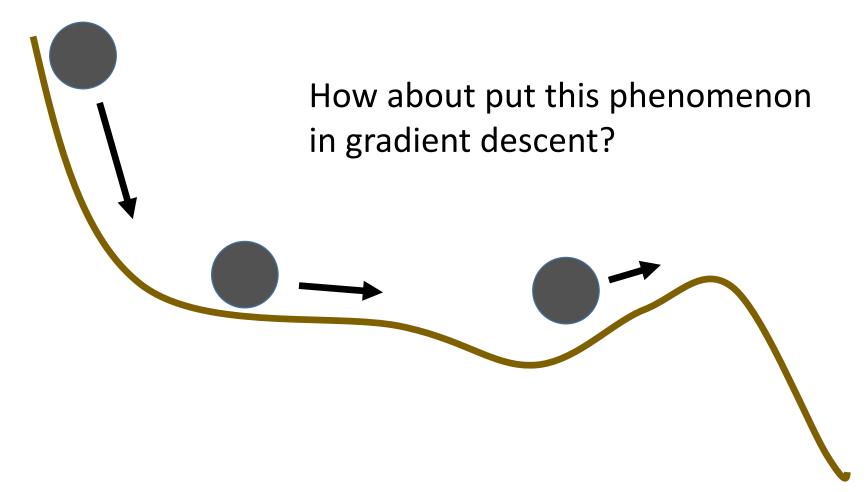
Hard to find optimal network parameters



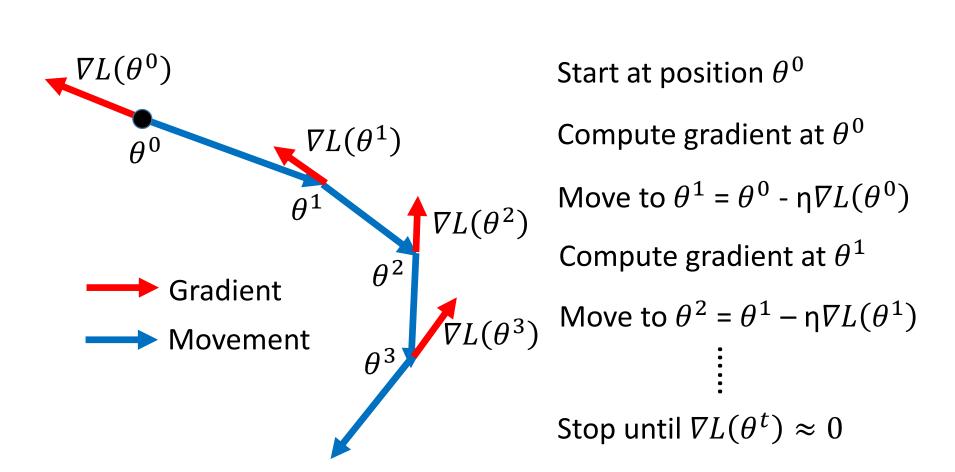
The value of a network parameter w

In physical world

Momentum

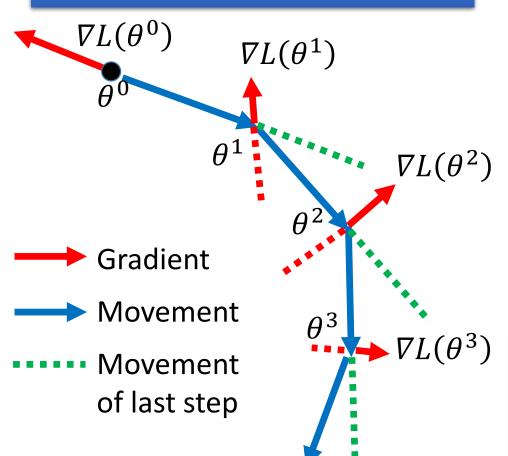


Review: Vanilla Gradient Descent



Momentum

Movement: movement of last step minus gradient at present



Start at point θ^0

Movement v⁰=0

Compute gradient at θ^0

Movement $v^1 = \lambda v^0 - \eta \nabla L(\theta^0)$

Move to $\theta^1 = \theta^0 + v^1$

Compute gradient at θ^1

Movement $v^2 = \lambda v^1 - \eta \nabla L(\theta^1)$

Move to $\theta^2 = \theta^1 + v^2$

Movement not just based on gradient, but previous movement.

Momentum

Movement: movement of last step minus gradient at present

vⁱ is actually the weighted sum of all the previous gradient:

$$\nabla L(\theta^0), \nabla L(\theta^1), \dots \nabla L(\theta^{i-1})$$

$$v^0 = 0$$

$$v^1 = - \eta \nabla L(\theta^0)$$

$$v^2 = -\lambda \, \eta \nabla L(\theta^0) - \eta \nabla L(\theta^1)$$

Start at point θ^0

Movement $v^0=0$

Compute gradient at θ^0

Movement $v^1 = \lambda v^0 - \eta \nabla L(\theta^0)$

Move to $\theta^1 = \theta^0 + v^1$

Compute gradient at θ^1

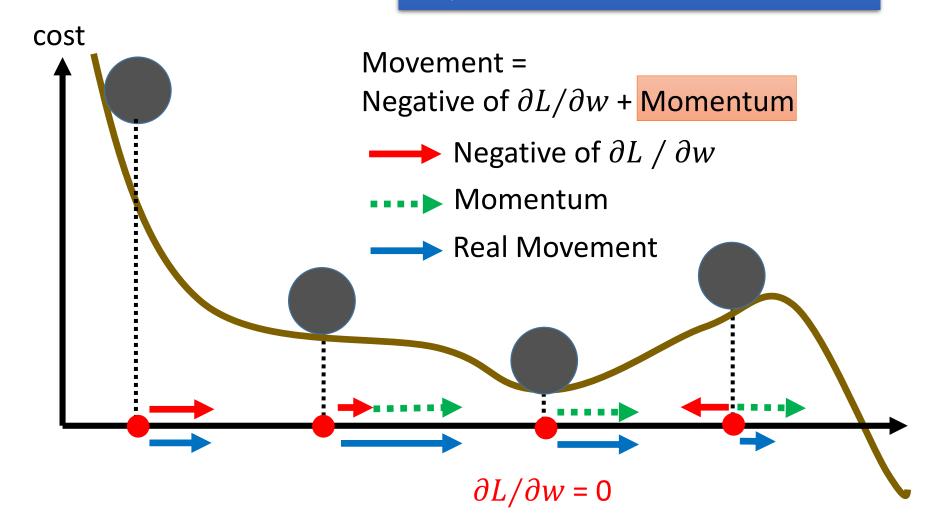
Movement $v^2 = \lambda v^1 - \eta \nabla L(\theta^1)$

Move to $\theta^2 = \theta^1 + v^2$

Movement not just based on gradient, but previous movement

Momentum

Still not guarantee reaching global minima, but give some hope



Adam

RMSProp + Momentum

Algorithm 1: Adam, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

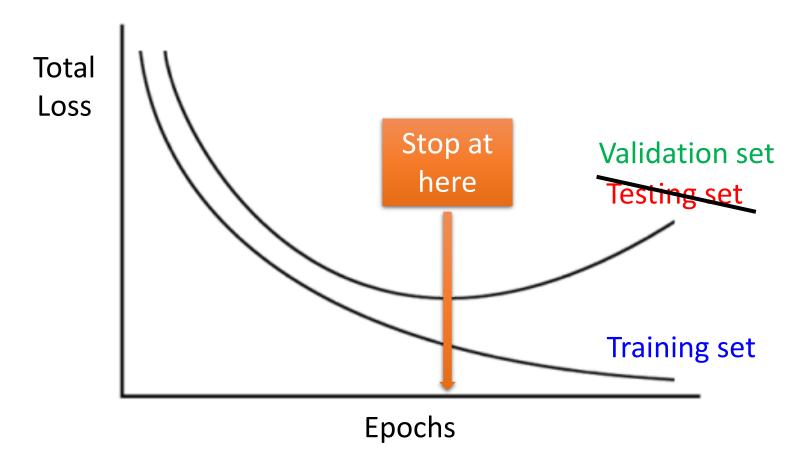
```
Require: \alpha: Stepsize
Require: \beta_1, \beta_2 \in [0, 1): Exponential decay rates for the moment estimates
Require: f(\theta): Stochastic objective function with parameters \theta
Require: \theta_0: Initial parameter vector
   m_0 \leftarrow 0 (Initialize 1<sup>st</sup> moment vector) \rightarrow for momentum
   v_0 \leftarrow 0 (Initialize 2<sup>nd</sup> moment vector)

→ for RMSprop

   t \leftarrow 0 (Initialize timestep)
   while \theta_t not converged do
      t \leftarrow t + 1
      g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1}) (Get gradients w.r.t. stochastic objective at timestep t)
      m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t (Update biased first moment estimate)
      v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 (Update biased second raw moment estimate)
      \widehat{m}_t \leftarrow m_t/(1-\beta_1^t) (Compute bias-corrected first moment estimate)
      \hat{v}_t \leftarrow v_t/(1-\beta_2^t) (Compute bias-corrected second raw moment estimate)
      \theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon) (Update parameters)
   end while
   return \theta_t (Resulting parameters)
```

Recipe of Deep Learning YES **Early Stopping** Good Results on **Testing Data?** Regularization YES Dropout Good Results on New activation function **Training Data?** Adaptive Learning Rate

Early Stopping



Keras: http://keras.io/getting-started/faq/#how-can-i-interrupt-training-when-the-validation-loss-isnt-decreasing-anymore

Recipe of Deep Learning YES **Early Stopping** Good Results on **Testing Data?** Regularization YES Dropout Good Results on New activation function **Training Data?** Adaptive Learning Rate

Regularization

- New loss function to be minimized
 - Find a set of weight not only minimizing original cost but also close to zero

$$L'(\theta) = L(\theta) + \lambda \frac{1}{2} \|\theta\|_{2} \longrightarrow \text{Regularization term}$$

$$\theta = \{w_{1}, w_{2}, \ldots\}$$

Original loss (e.g. minimize square error, cross entropy ...) L2 regularization:

$$\|\theta\|_{2} = (w_{1})^{2} + (w_{2})^{2} + \dots$$

(usually not consider biases)

Regularization

L2 regularization:

$$\|\theta\|_2 = (w_1)^2 + (w_2)^2 + \dots$$

New loss function to be minimized

$$L'(\theta) = L(\theta) + \lambda \frac{1}{2} \|\theta\|_2$$
 Gradient: $\frac{\partial L'}{\partial w} = \frac{\partial L}{\partial w} + \lambda w$

Update:
$$w^{t+1} \rightarrow w^t - \eta \frac{\partial L'}{\partial w} = w^t - \eta \left(\frac{\partial L}{\partial w} + \lambda w^t \right)$$

$$= \underbrace{(1 - \eta \lambda)w^t}_{\text{Closer to zero}} - \eta \frac{\partial L}{\partial w} \quad \text{Weight Decay}$$

Regularization

L1 regularization:

$$\|\theta\|_1 = |w_1| + |w_2| + \dots$$

New loss function to be minimized

$$L'(\theta) = L(\theta) + \lambda \frac{1}{2} \|\theta\|_{1} \qquad \frac{\partial L'}{\partial w} = \frac{\partial L}{\partial w} + \lambda \operatorname{sgn}(w)$$
Update:
$$w^{t+1} \to w^{t} - \eta \frac{\partial L'}{\partial w} = w^{t} - \eta \left(\frac{\partial L}{\partial w} + \lambda \operatorname{sgn}(w^{t})\right)$$

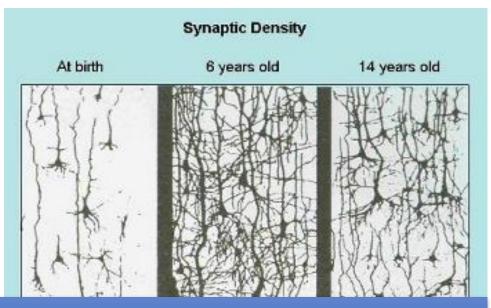
$$= w^{t} - \eta \frac{\partial L}{\partial w} - \underline{\eta \lambda \operatorname{sgn}(w^{t})} \text{ Always delete}$$

$$= (1 - \eta \lambda)w^{t} - \eta \frac{\partial L}{\partial w} \qquad \dots \qquad \text{L2}$$

Regularization - Weight Decay

Our brain prunes out the useless link between

neurons.

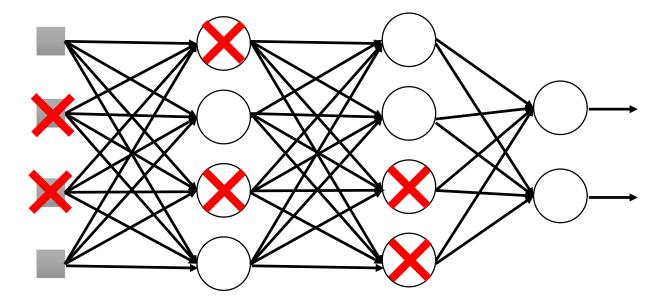


Doing the same thing to machine's brain improves the performance.



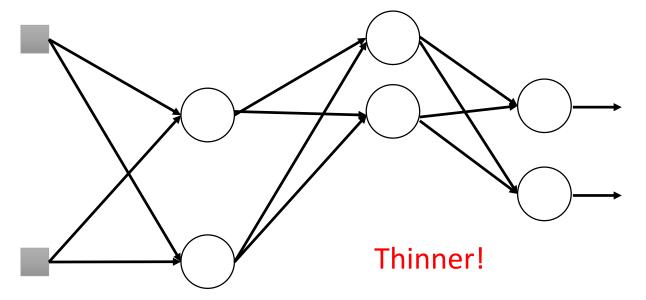
Recipe of Deep Learning YES **Early Stopping** Good Results on **Testing Data?** Regularization YES Dropout Good Results on New activation function **Training Data?** Adaptive Learning Rate

Training:



- > Each time before updating the parameters
 - Each neuron has p% to dropout

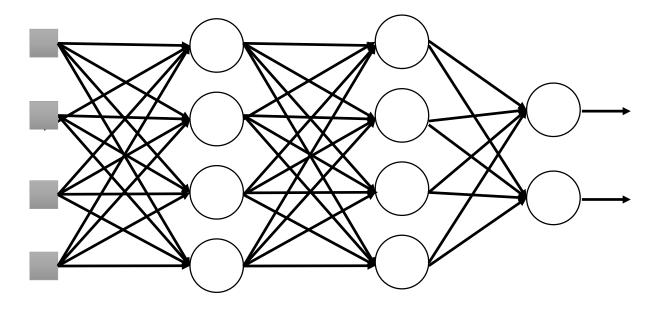
Training:



- > Each time before updating the parameters
 - Each neuron has p% to dropout
 - The structure of the network is changed.
 - Using the new network for training

For each mini-batch, we resample the dropout neurons

Testing:



No dropout

- If the dropout rate at training is p%,
 all the weights times 1-p%
- Assume that the dropout rate is 50%. If a weight w = 1 by training, set w = 0.5 for testing.

- Intuitive Reason

Training

Dropout (腳上綁重物)

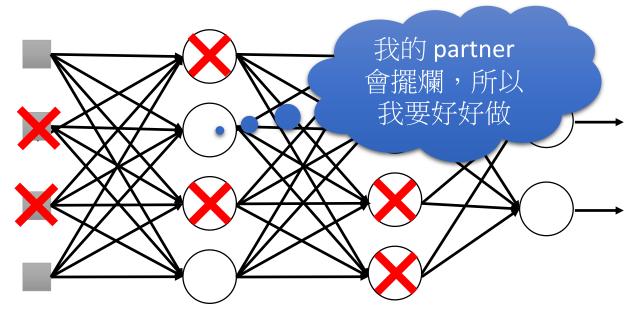


Testing

No dropout (拿下重物後就變很強)



Dropout - Intuitive Reason



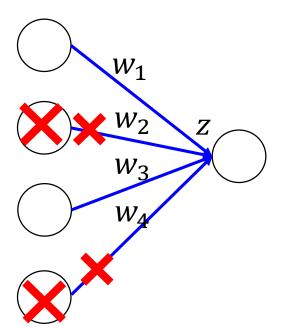
- ➤ When teams up, if everyone expect the partner will do the work, nothing will be done finally.
- ➤ However, if you know your partner will dropout, you will do better.
- When testing, no one dropout actually, so obtaining good results eventually.

Dropout - Intuitive Reason

• Why the weights should multiply (1-p)% (dropout rate) when testing?

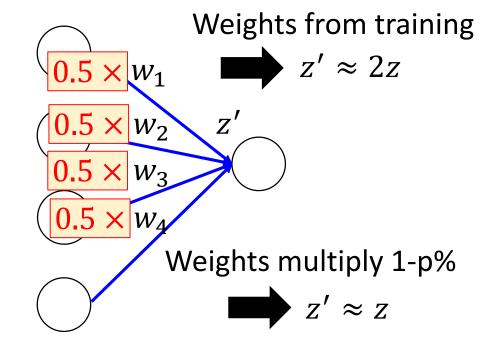
Training of Dropout

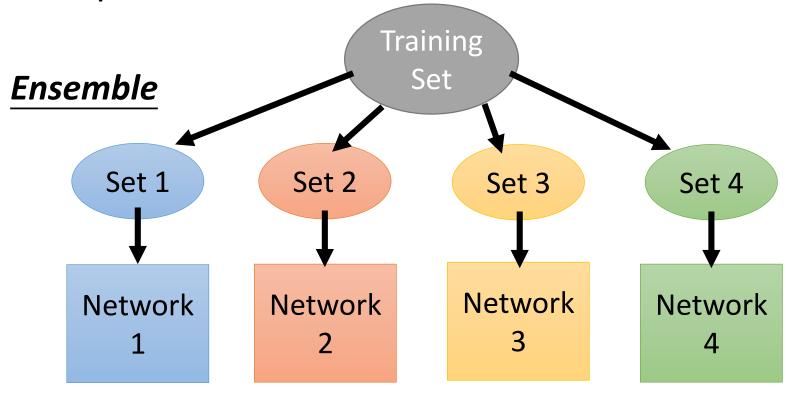
Assume dropout rate is 50%



Testing of Dropout

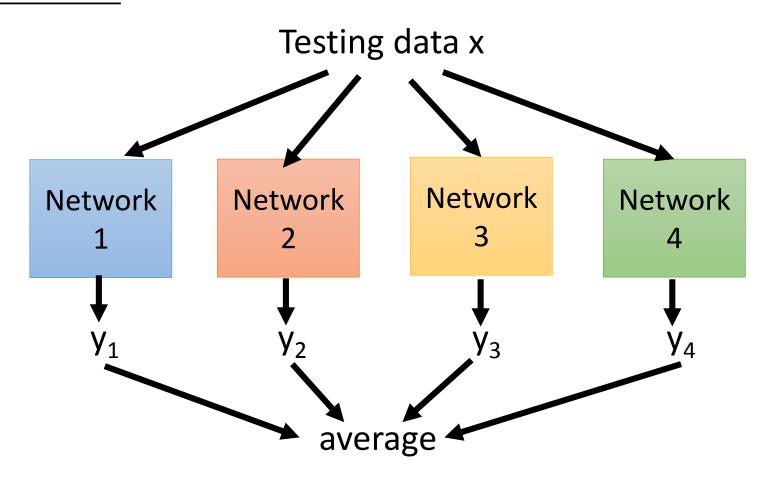
No dropout

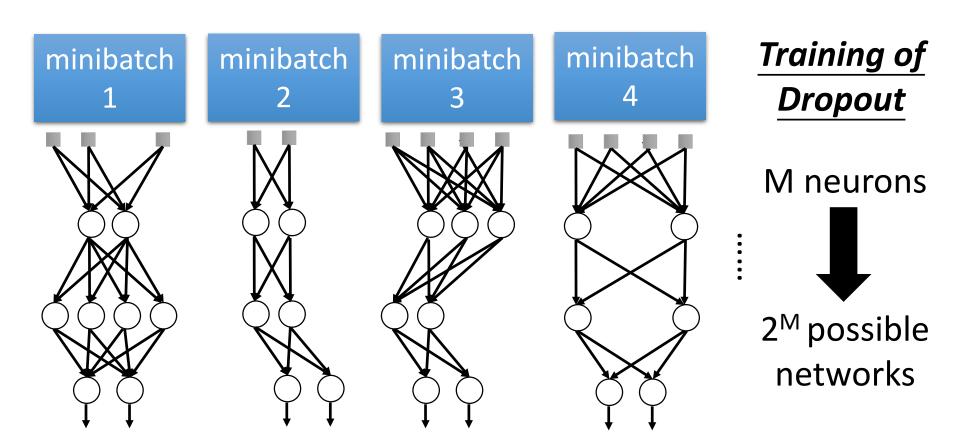




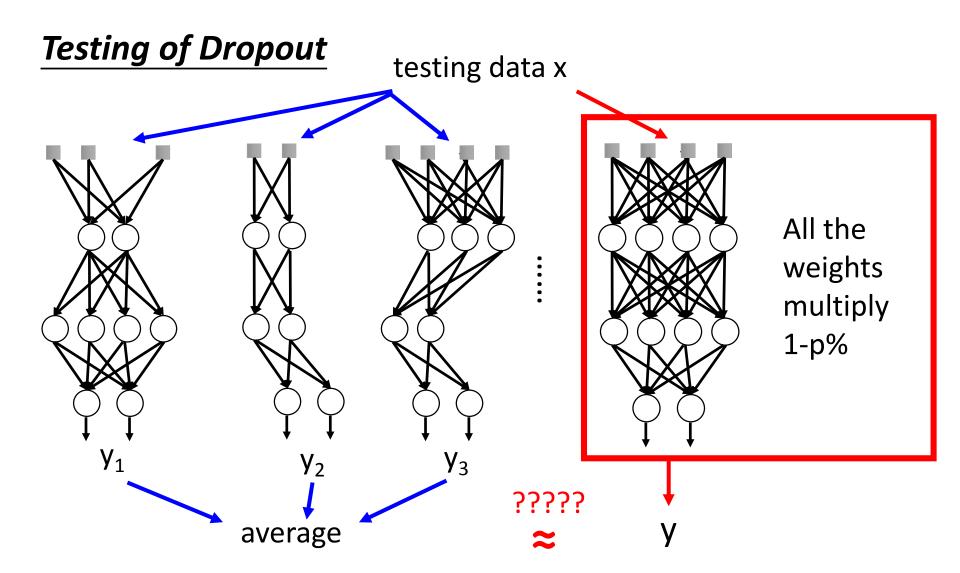
Train a bunch of networks with different structures

Ensemble

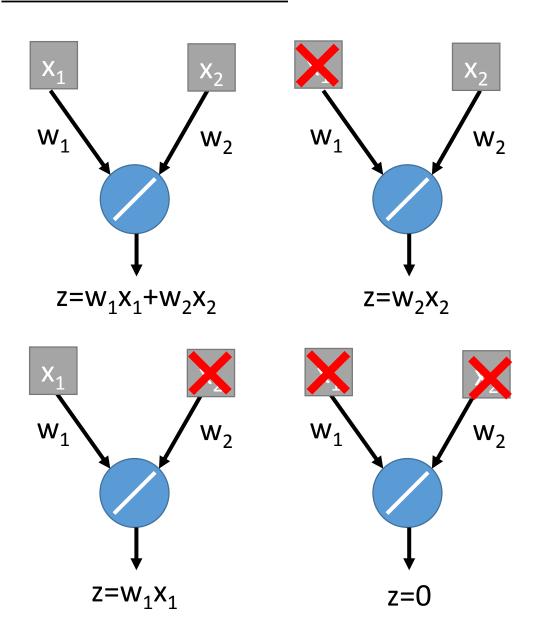


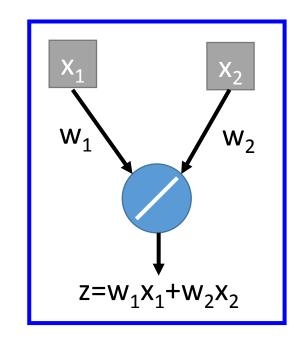


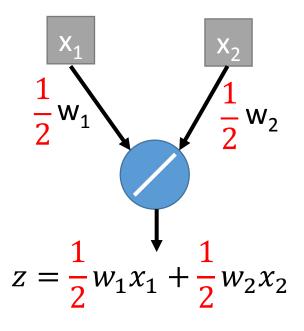
- ➤ Using one mini-batch to train one network
- Some parameters in the network are shared



Testing of Dropout







Recipe of Deep Learning



Step 1: define a set of function

Step 2: goodness of function

Step 3: pick the best function

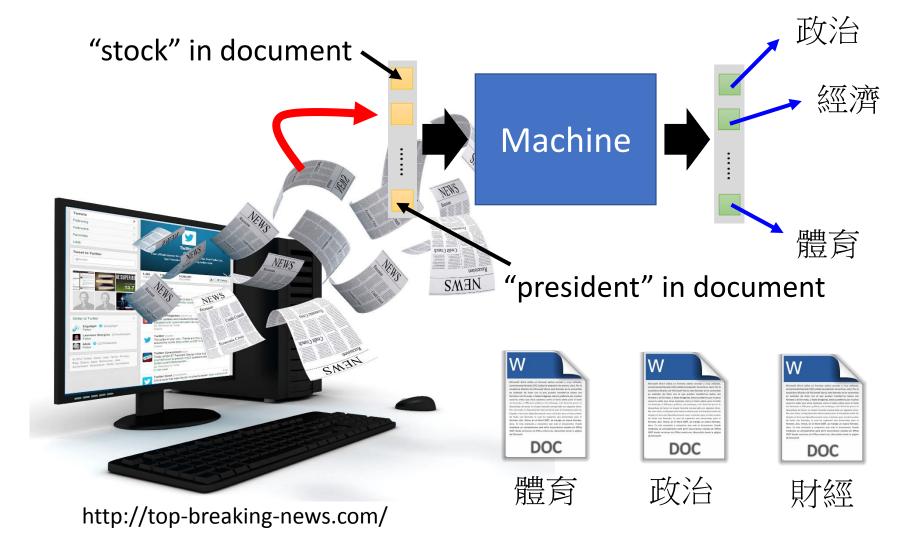
NO

Overfitting!

NO

Neural Network

Try another task



Try another task

```
In [9]: y train.shape
                                           Out[9]: (8982, 46)
In [12]: x train[0]
                                          In [10]: x test.shape
Out[12]:
array([ 0., 1., 1., 0., 1., 1., 1., 1., 10ut[10]: (2246, 1000)
                   1., 1., 0., 1., 0.,
                                     0.,
                                           In [11]: y test.shape
                   0., 1., 1., 0.,
                                     0.,
                                     0.,
               0.,
                   0.,
                        0., 0.,
                                0.,
                        0.,
               1.,
                            0., 0.,
                   0.,
                                     0.,
                                         0.,
                                              0., 0.,
                   0.,
                        0., 1., 1.,
               0.,
                                     0.,
                                         0.,
                                              0.,
                                                  0.,
                   0.,
                        0.,
                            0., 0.,
                                     0.,
               0.,
                                              0.,
                                                  0.,
               0.,
                        0.,
                   0.,
                            1., 0., 1.,
                                              0.,
                                                  0.,
               0.,
                   0.,
                        0.,
                            0., 0., 1.,
                                         0.,
                                              0.,
                                                  0.,
               1.,
                   0., 1.,
                            0., 0.,
                                     0.,
                                              0.,
                                                  0.,
                            0., 0.,
                   0.,
                        0.,
                                     0., 1.,
               0.,
                                              0.,
                                                  0.,
               0.,
                   0.,
                        0.,
                            0., 0.,
                                     0.,
                                         0.,
                                              0.,
                                                  0.,
                        0., 0., 0.,
                                         0.,
               0.,
                   0.,
                                     0.,
                                              0.,
                                                  0.,
                            0., 0.,
                                     0.,
                                         0.,
                                              0.,
               0.,
                   0.,
                       1.,
                                                  0.,
               1.,
                   0., 0., 0., 0., 0.,
                                         0.,
                                              0.,
                                                  0.,
      0., 1., 0., 0., 0., 0., 0., 0.,
                                         0.,
                                              0.,
                                                  0.,
In [13]: y train[0]
Out[13]:
           0., 0., 1., 0., 0., 0., 0., 0., 0.,
array([ 0.,
                                                  0.,
               0.,
                   0., 0.,
                           0., 0., 0., 0., 0.,
                   0., 0.,
               0.,
                           0., 0., 0., 0., 0.,
           0., 0., 0., 0., 0., 0.])
```

In [8]: x_train.shape Out[8]: (8982, 1000)