Lecture 18

Dropout

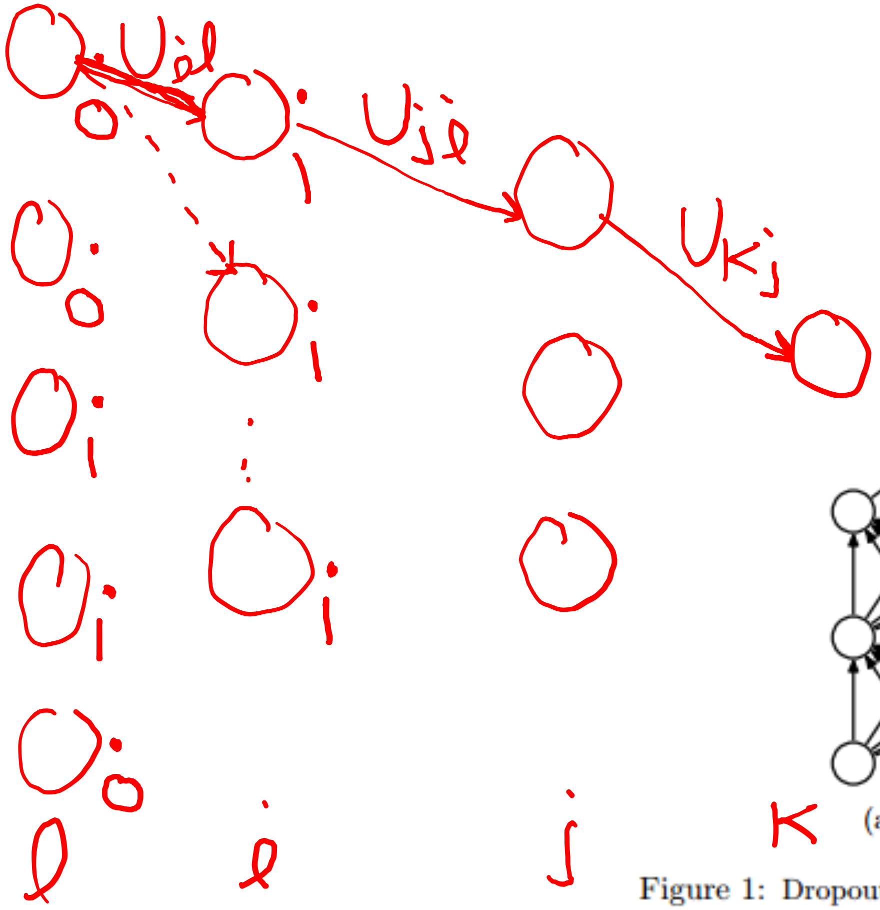
- Dropout is one of the techniques for preventing overfitting in deep neural network which contains a large number of parameters.

Original Paper

- Title:
  - Dropout: A Simple Way to Prevent Neural Networks from Overfitting.
- Authors:
  - Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov
- Organization:
  - Department of Computer Science, University of Toronto
- URL:
  - https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

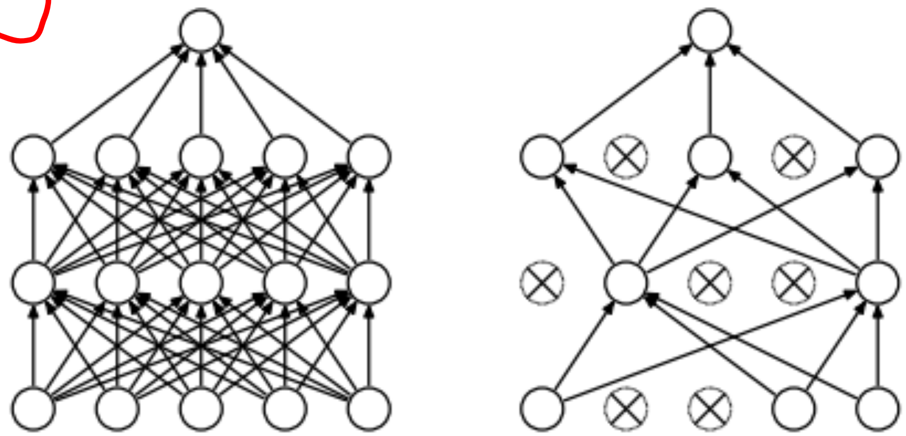
Overview

- The key idea is to randomly drop units from the neural network during training.
- During training, dropout samples from an exponential number of different "thinned" network.
- At test time, we approximate the effect of averaging the predictions of all these thinned networks.



$$U_{jl} \leftarrow U_{jl} - \eta \delta_j \cdot Z_l$$

Bernoulli $\left(p = \frac{1}{2}\right)$



(a) Standard Neural Net

(b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

$$U_{il} \leftarrow U_{il} - \eta \delta_i \cdot Z_l$$

Model

Consider a neural network with L hidden layer. Let $\mathbf{z^{(l)}}$ denote the vector inputs into layer l, $\mathbf{y}^{(l)}$ denote the vector of outputs from layer l. $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weights and biases at layer l. With dropout, the feed-forward operation becomes:

$$r_j^{(l)} \sim Bernoulli(p)$$

$r \in \{0, 1\}$

$$\tilde{\mathbf{y}}^{(l)} = \mathbf{r}^{(l)} * \mathbf{y}^{(l)}$$, here * denotes an element-wise product.

$$z_i^{(l+1)} = \mathbf{w}_i^{(l+1)} \tilde{\mathbf{y}}^{l} + b_i^{(l+1)}$$

$y, b, w, z \in \mathbb{R}$

$$y_i^{(l+1)} = f(z_i^{(l+1)}),$$ where f is the activation function.

For any layer l, $\mathbf{r}^{(l)}$ is a vector of independent Bernoulli random variables each of which has probability of p of being 1. $\tilde{\mathbf{y}}$ is the input after we drop some hidden units. The rest of the model remains the same as the regular feed-forward neural network.

Training

- Dropout neural network can be trained using stochastic gradient descent.
- The only difference here is that we only back propagate on each thinned network.
- The gradient for each parameter are averaged over the training cases in each mini-batch.

Test Time

- use a single neural net without dropout.
- If a unit is retained with probability p during training, the outgoing weights of that unit are multiplied by p at test time.

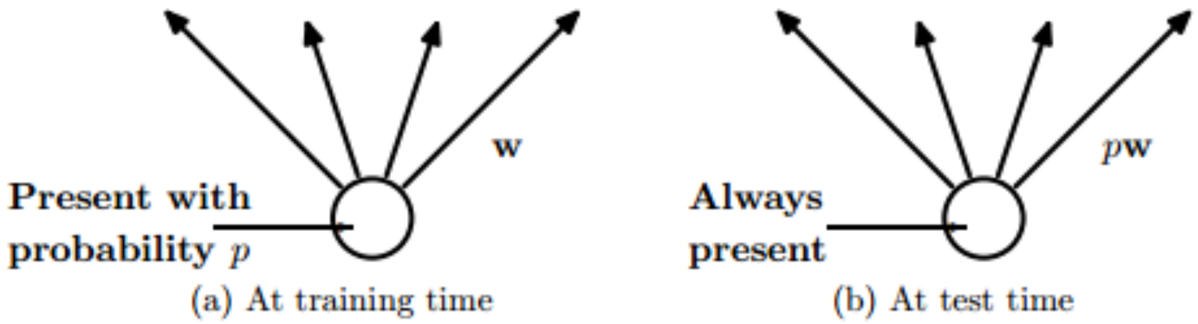
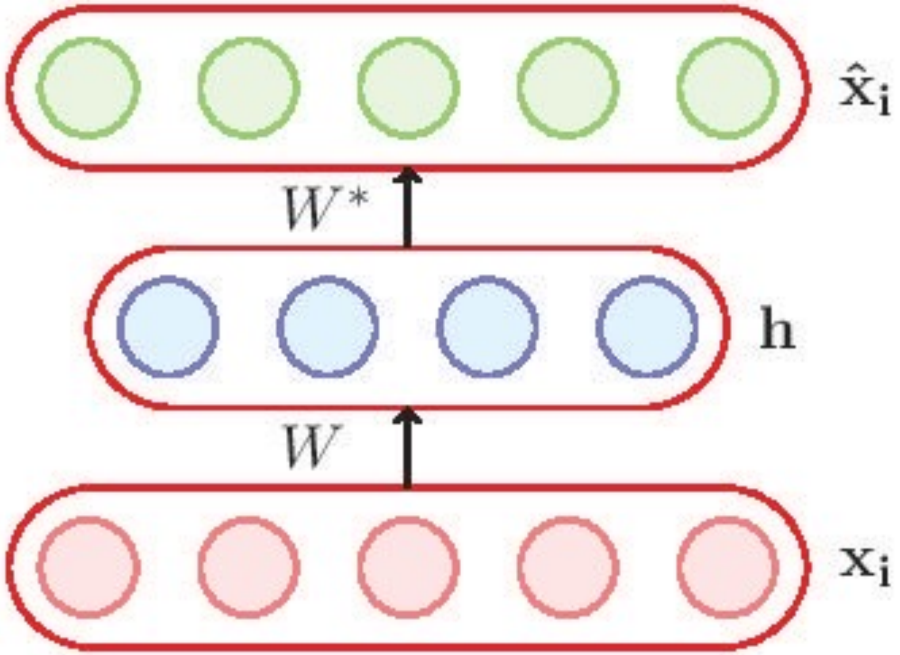


Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Autoencoders

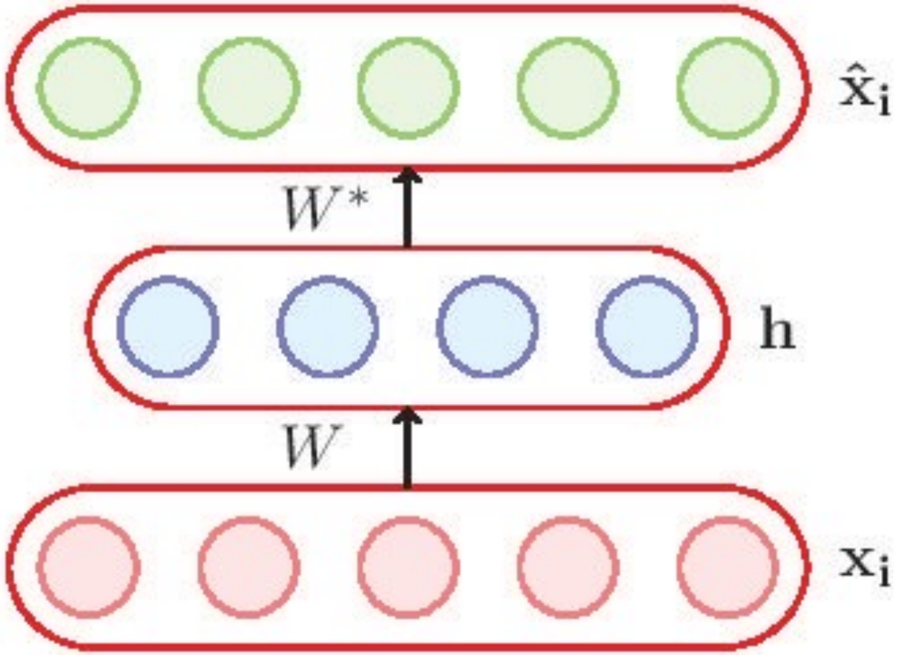
- Slides obtained from Dr. Khapra and edited with his permission

Module 7.1: Introduction to Autoencoders



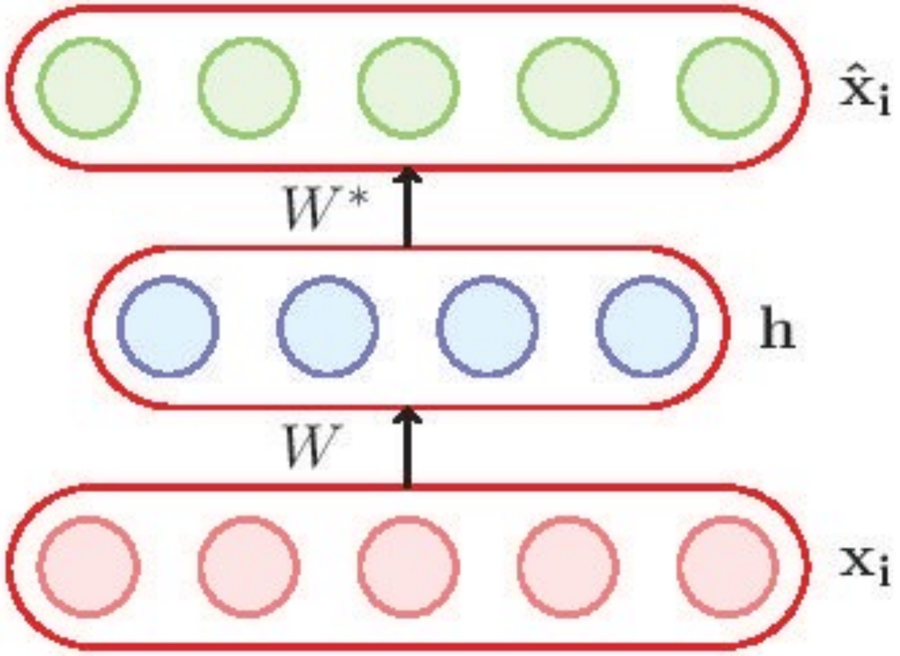
$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$

- An autoencoder is a special type of feed forward neural network which does the following
- Encodes its input $\mathbf{x}_\mathbf{i}$ into a hidden representation $\mathbf{h}$



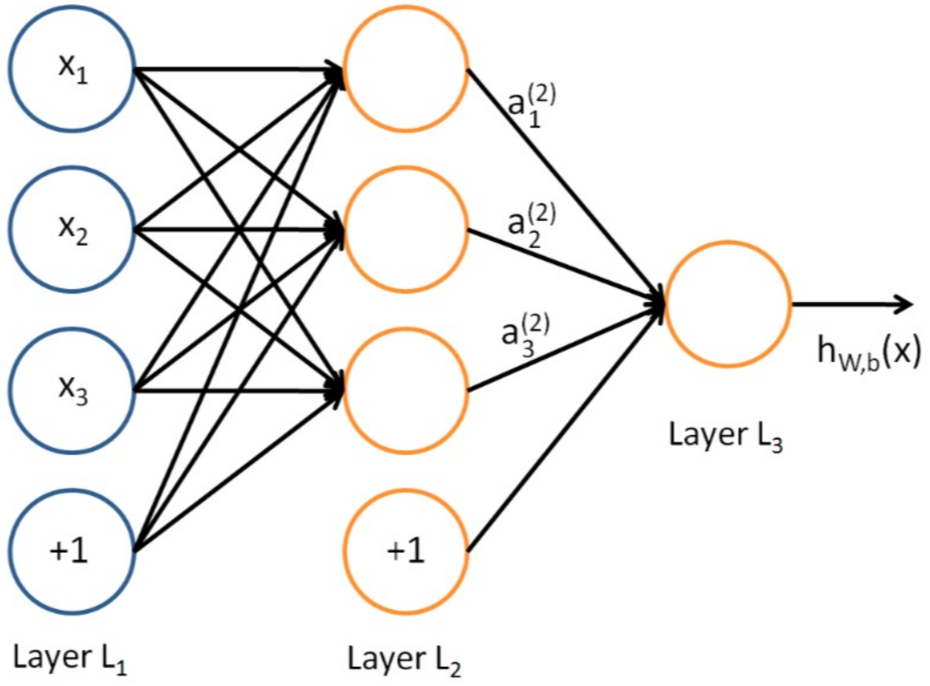
$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$

- An autoencoder is a special type of feed forward neural network which does the following
- Encodes its input $\mathbf{x}_\mathbf{i}$ into a hidden representation $\mathbf{h}$
- Decodes the input again from this hidden representation



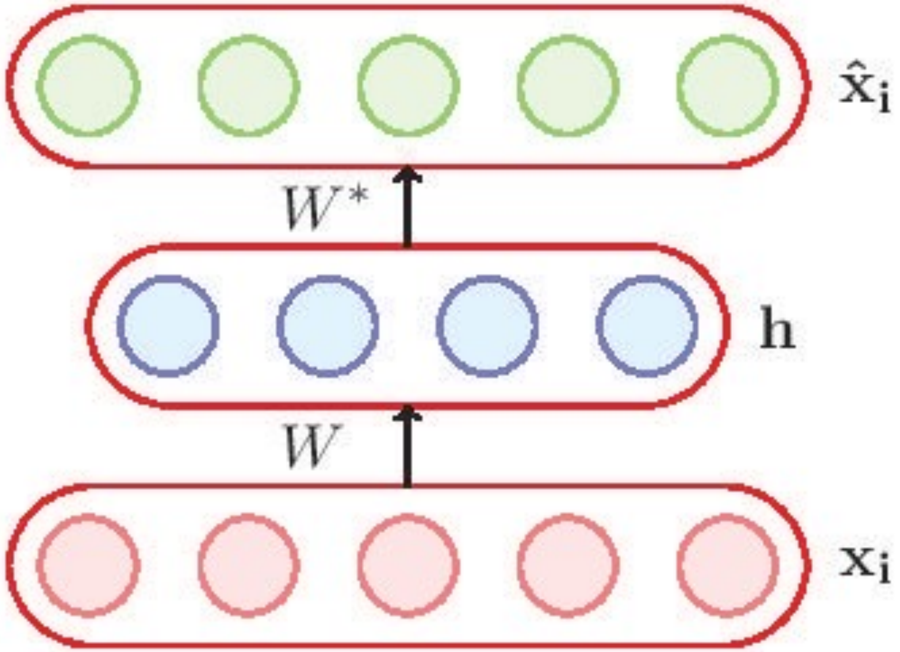
$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- An autoencoder is a special type of feed forward neural network which does the following
- <ins>Encodes</ins> its input $\mathbf{x}_\mathbf{i}$ into a hidden representation $\mathbf{h}$
- <ins>Decodes</ins> the input again from this hidden representation
- The model is trained to minimize a certain loss function which will ensure that $\hat{\mathbf{x}}_\mathbf{i}$ is close to $\mathbf{x}_\mathbf{i}$ (we will see some such loss functions soon)



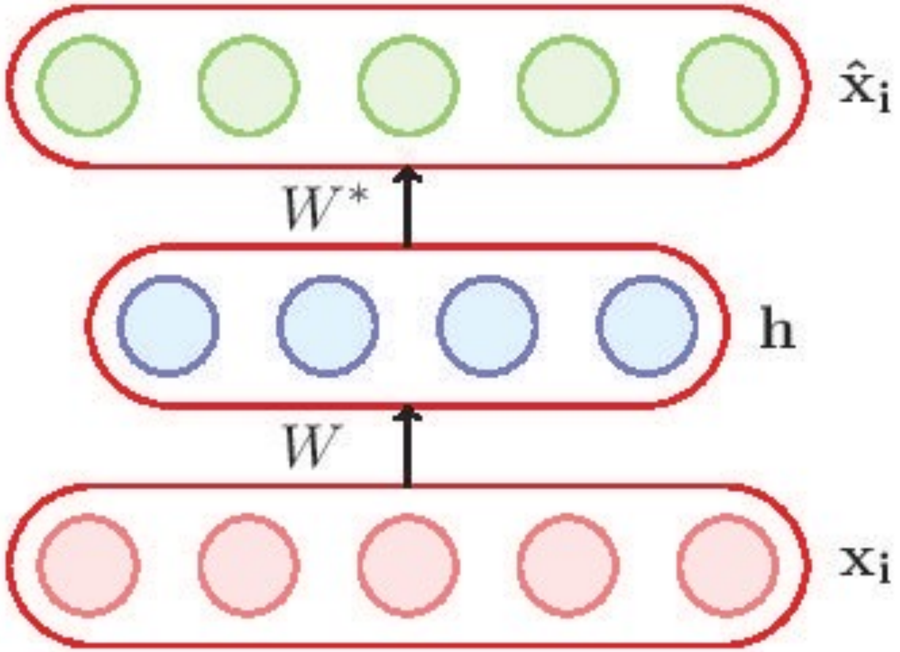
- $W_{11}^{(1)}$ is the weight connecting x_1 to first node
- $W_{12}^{(1)}$ is the weight connecting x_2 to first node
- $W_{21}^{(1)}$ is the weight connecting x_1 to second node

$$
\begin{aligned}
a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \\
a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \\
a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \\
h_{W,b}(x) &= a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})
\end{aligned}
$$



$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$

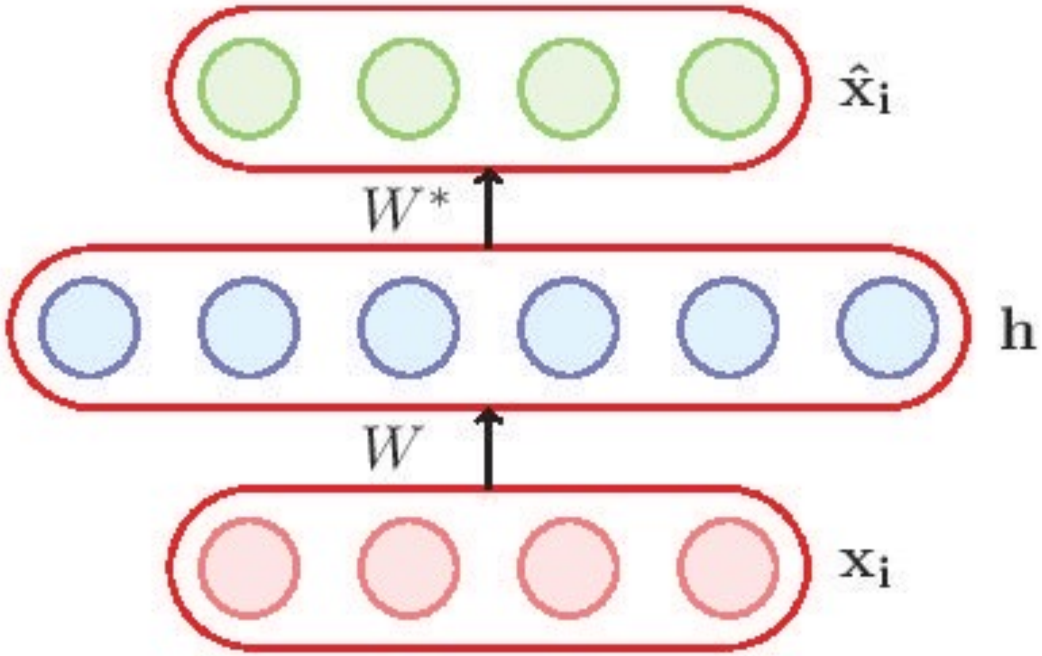
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$



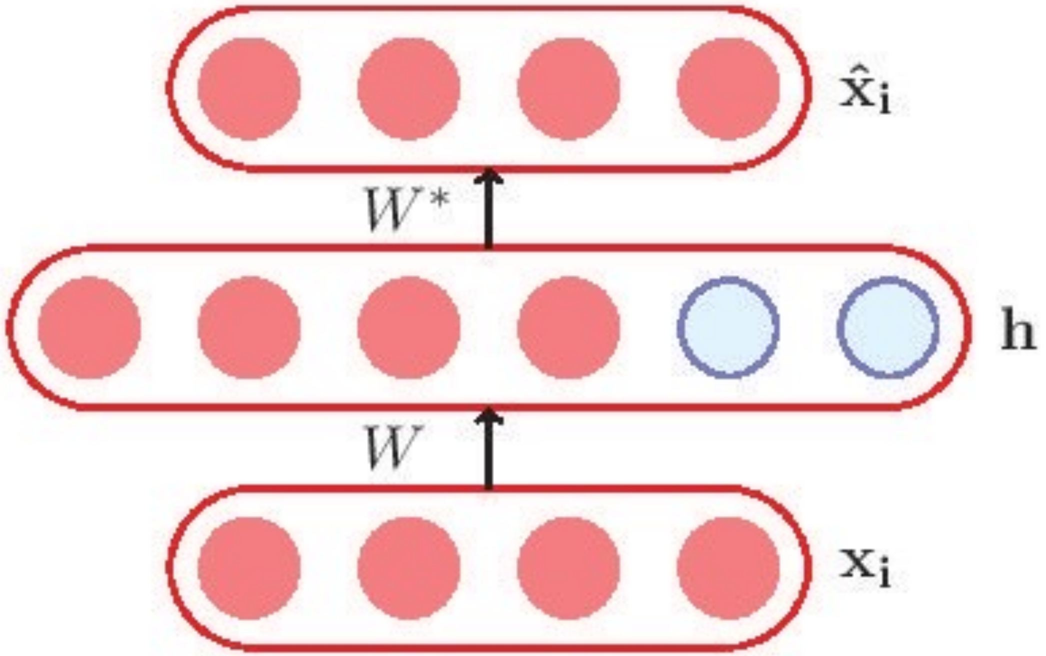
$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

An autoencoder where $\dim(\mathbf{h}) < \dim(\mathbf{x}_\mathbf{i})$ is called an <u>under complete</u> autoencoder

- Let us consider the case where $\dim(\mathbf{h}) < \dim(\mathbf{x}_\mathbf{i})$
- If we are still able to reconstruct $\hat{\mathbf{x}}_\mathbf{i}$ perfectly from $\mathbf{h}$, then what does it say about $\mathbf{h}$?
- $\mathbf{h}$ is a loss-free encoding of $\mathbf{x}_\mathbf{i}$. It captures all the important characteristics of $\mathbf{x}_\mathbf{i}$
- Do you see an analogy with PCA?



$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$



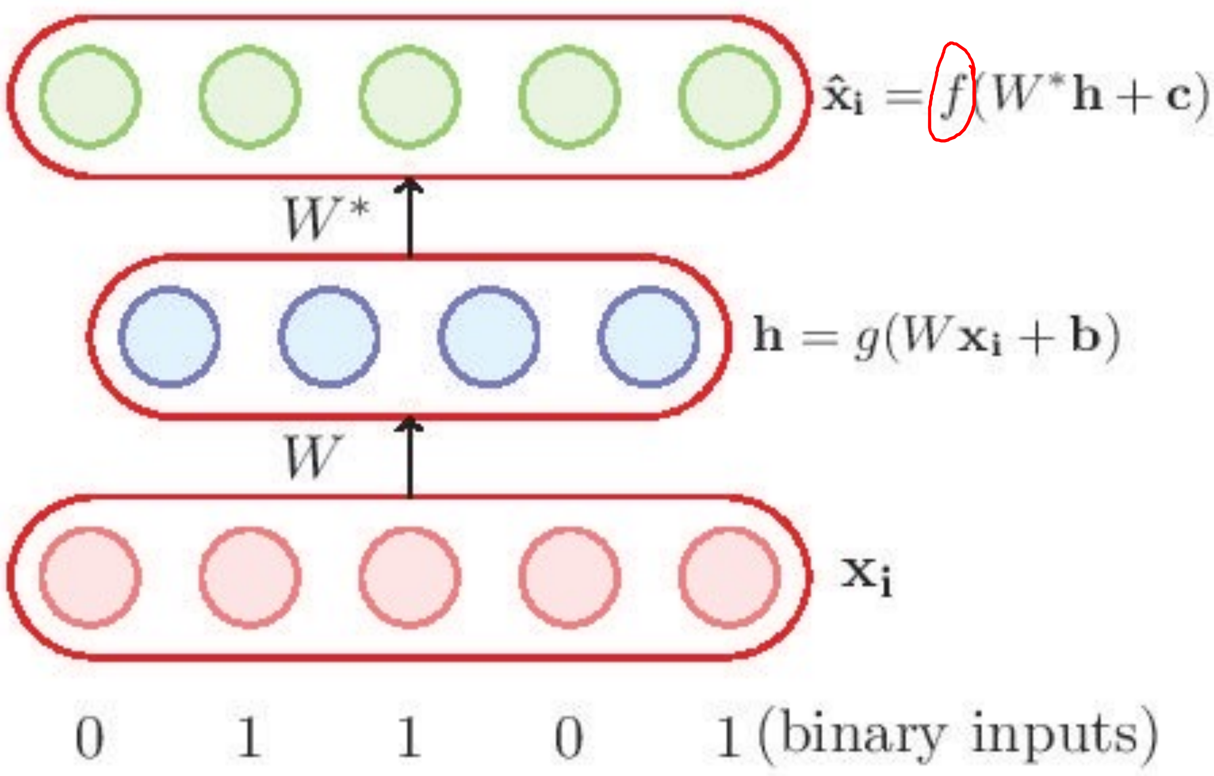
$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

An autoencoder where $\dim(\mathbf{h}) \geq \dim(\mathbf{x}_\mathbf{i})$ is called an over complete autoencoder

- Let us consider the case when $\dim(\mathbf{h}) \geq \dim(\mathbf{x}_\mathbf{i})$
- In such a case the autoencoder could learn a trivial encoding by simply copying $\mathbf{x}_\mathbf{i}$ into $\mathbf{h}$ and then copying $\mathbf{h}$ into $\hat{\mathbf{x}}_\mathbf{i}$
- Such an identity encoding is useless in practice as it does not really tell us anything about the important characteristics of the data

The Road Ahead

- Choice of $f(\mathbf{x_i})$ and $g(\mathbf{x_i})$
- Choice of loss function



- Suppose all our inputs are binary (each $x_{ij} \in \{0, 1\}$)
- Which of the following functions would be most apt for the decoder?

$$\hat{\mathbf{x}}_\mathbf{i} = \tanh(W^*\mathbf{h} + \mathbf{c}) \quad \in (-1, 1)$$

$$\hat{\mathbf{x}}_\mathbf{i} = W^*\mathbf{h} + \mathbf{c} \quad \in (-\infty, \infty)$$

$$\hat{\mathbf{x}}_\mathbf{i} = \cancel{logistic}(W^*\mathbf{h} + \mathbf{c})$$

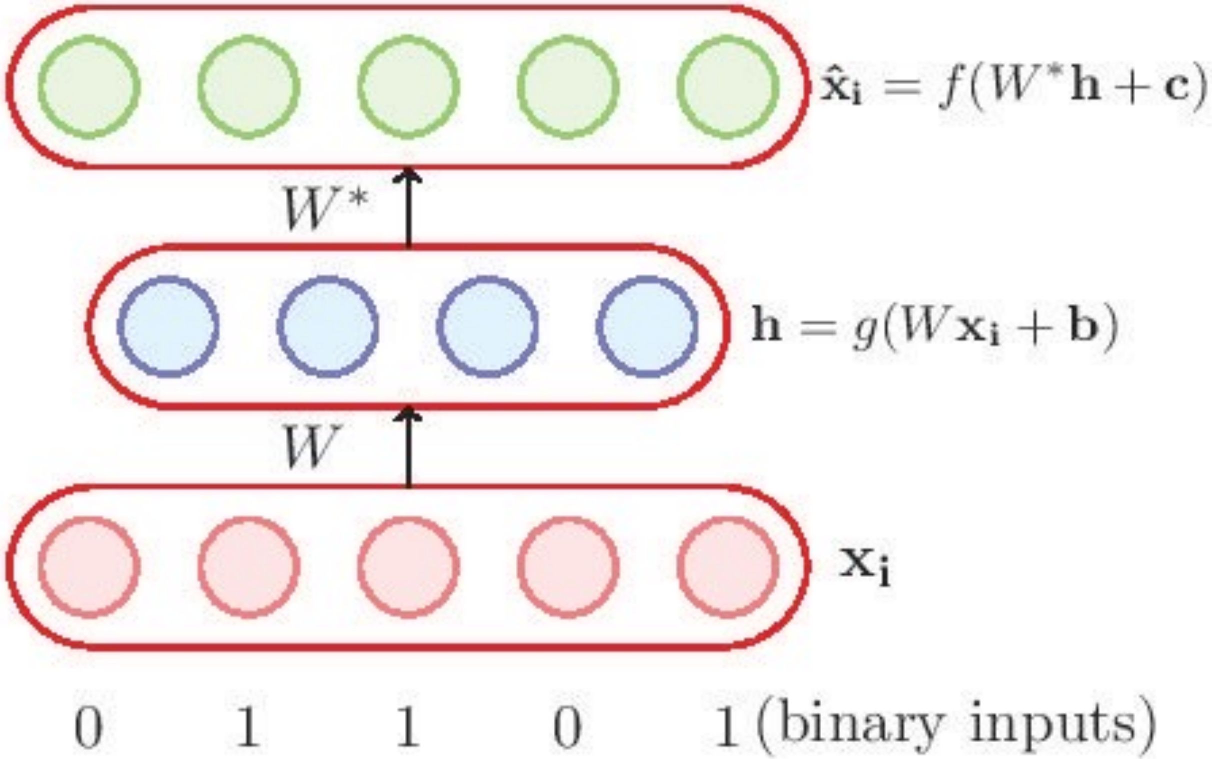
σ

sigmoid. $\in (0, 1)$

$$E = \| \hat{x}_i - x_i \|_2^2$$

$\tanh =$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

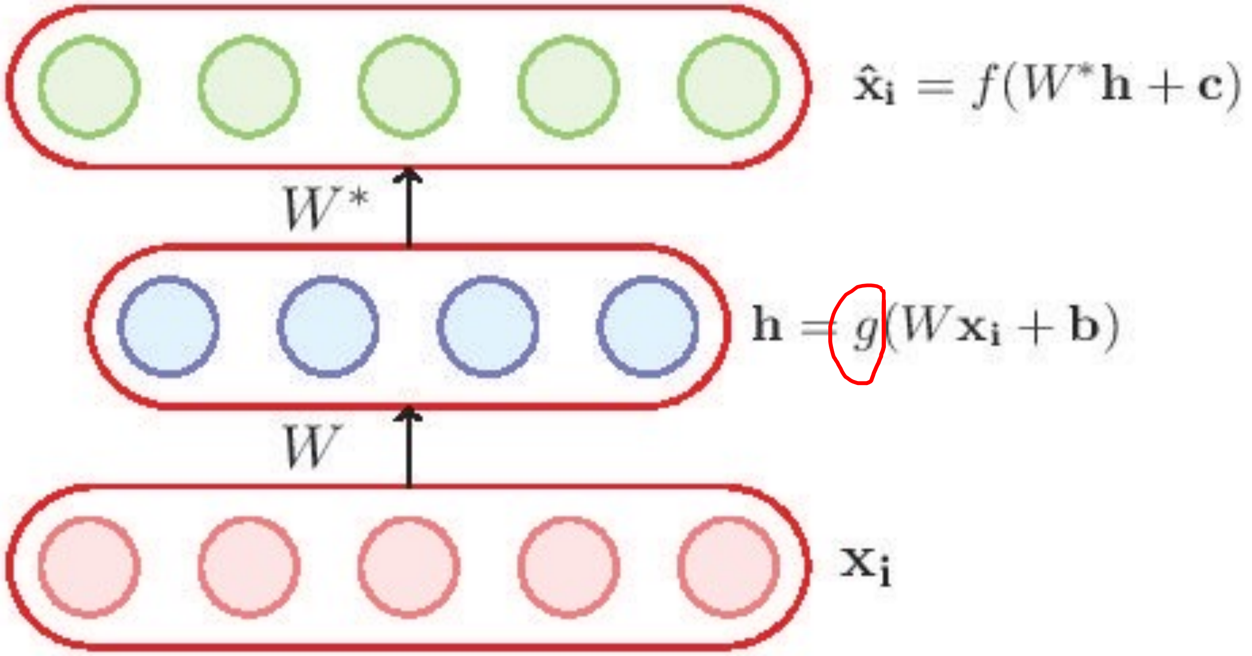


g is typically chosen as the sigmoid function

- Suppose all our inputs are binary (each $x_{ij} \in \{0, 1\}$)
- Which of the following functions would be most apt for the decoder?

$$\hat{\mathbf{x}}_\mathbf{i} = \tanh(W^*\mathbf{h} + \mathbf{c})$$
$$\hat{\mathbf{x}}_\mathbf{i} = W^*\mathbf{h} + \mathbf{c}$$
$$\hat{\mathbf{x}}_\mathbf{i} = logistic(W^*\mathbf{h} + \mathbf{c})$$

- Logistic as it naturally restricts all outputs to be between 0 and 1



0.25 0.5 1.25 3.5 4.5

(real valued inputs) $\in (-\infty, \infty)$

Again, g is typically chosen as the sigmoid function

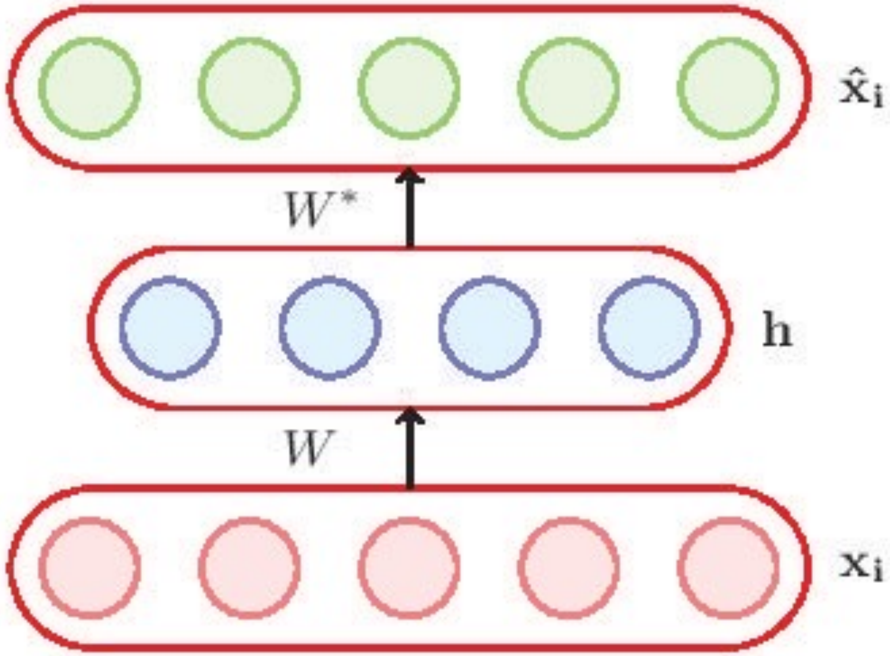
- Suppose all our inputs are real (each $x_{ij} \in \mathbb{R}$)
- Which of the following functions would be most apt for the decoder?

$$\hat{\mathbf{x}}_\mathbf{i} = \tanh(W^*\mathbf{h} + \mathbf{c})$$
$$\hat{\mathbf{x}}_\mathbf{i} = W^*\mathbf{h} + \mathbf{c}$$
$$\hat{\mathbf{x}}_\mathbf{i} = \underset{\sigma}{\cancel{\text{logistic}}}(W^*\mathbf{h} + \mathbf{c})$$

- What will logistic and tanh do?
- They will restrict the reconstructed $\hat{\mathbf{x}}_\mathbf{i}$ to lie between [0,1] or [-1,1] whereas we want $\hat{\mathbf{x}}_\mathbf{i} \in \mathbb{R}^n$

The Road Ahead

- Choice of $f(\mathbf{x_i})$ and $g(\mathbf{x_i})$
- Choice of loss function



$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

$X_i \in R^d$

an element of $X_i = \{x_{ij}\}$

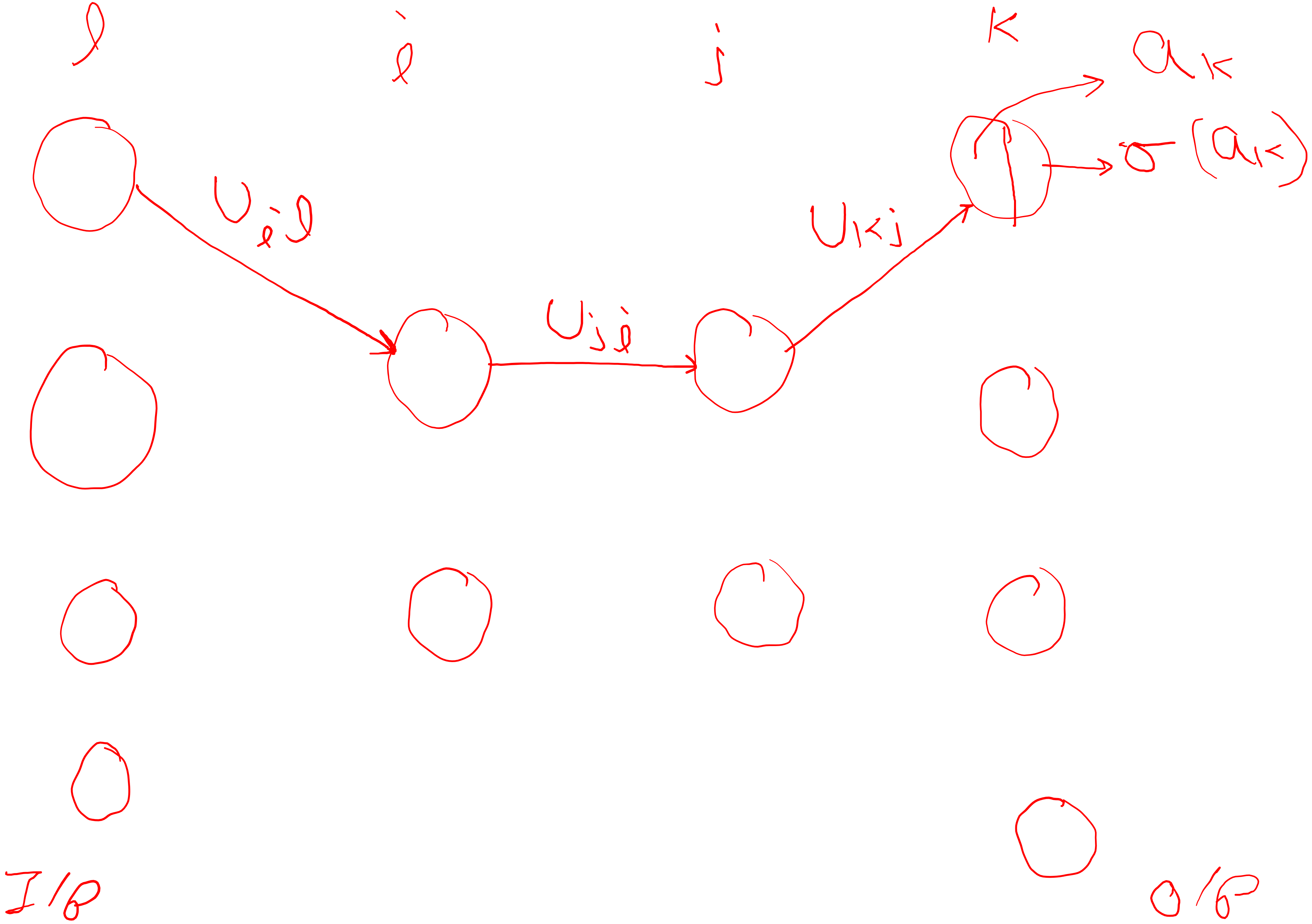
- Consider the case when the inputs are real valued
- The objective of the autoencoder is to reconstruct $\hat{\mathbf{x}}_\mathbf{i}$ to be as close to $\mathbf{x}_\mathbf{i}$ as possible
- This can be formalized using the following objective function:

$n = d$

$$\min_{W,W^*,\mathbf{c},\mathbf{b}} \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}(\hat{x}_{ij} - x_{ij})^2$$

$$i.e., \min_{W,W^*,\mathbf{c},\mathbf{b}} \frac{1}{m}\sum_{i=1}^{m}(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})^T(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})$$

- We can then train the autoencoder just like a regular feedforward network using backpropagation
- All we need is a formula for $\frac{\partial \mathscr{L}(\theta)}{\partial W^*}$ and $\frac{\partial \mathscr{L}(\theta)}{\partial W}$ which we will see now


l
i
j
k
U_{il}
U_{ji}
U_{kj}
a_k
$\sigma(a_k)$
I/P
O/P

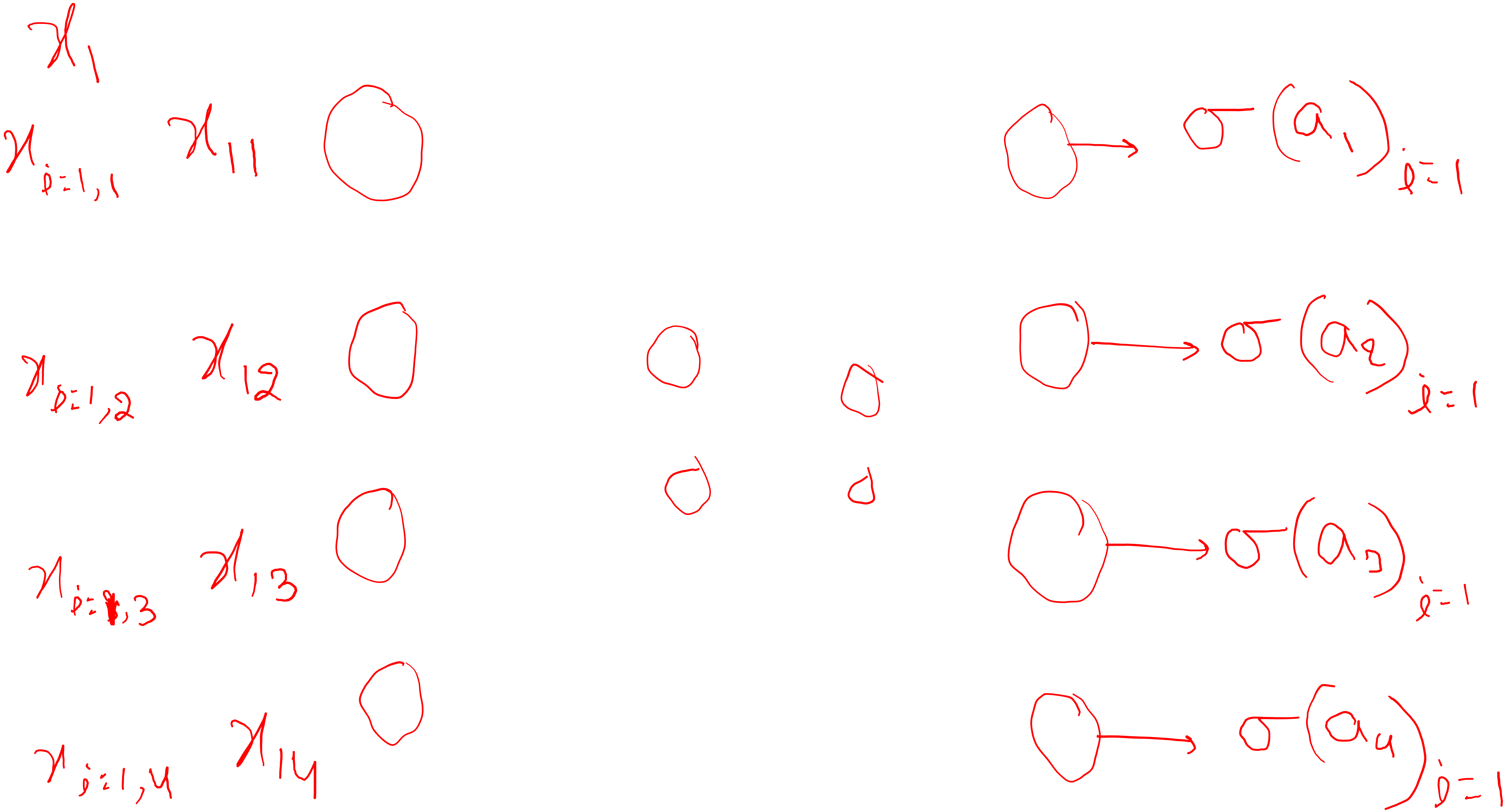
$$U_{il} \leftarrow U_{il} - \eta \delta_i \cdot Z_l$$

$$\delta_i = \sum_j \delta_j U_{ji} \sigma'(a_i)$$

$$\delta_j = \sum_k \delta_k U_{kj} \sigma'(a_j)$$

$$\delta_k = \frac{\partial E}{\partial a_k} = \frac{\partial}{\partial a_k} \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{n} \left(\sigma(a_k)_i - x_{ik} \right)^2$$

$$\delta_k = \frac{1}{m} \sum_{i=1}^{m} 2 \left[\sigma(a_k)_i - x_{ik} \right] \sigma'(a_k)_i$$



x_1
$x_{i=1,1}$
x_{11}
$\sigma(a_1)_{i=1}$
$x_{i=1,2}$
x_{12}
$\sigma(a_2)_{i=1}$
$x_{i=1,3}$
x_{13}
$\sigma(a_3)_{i=1}$
$x_{i=1,4}$
x_{14}
$\sigma(a_4)_{i=1}$

$$\left(\sigma(a_1)_{i=1} - y_{i=1,1}\right)^2$$

$$+\left(\sigma(a_2)_{i=1} - y_{i=1,2}\right)^2$$

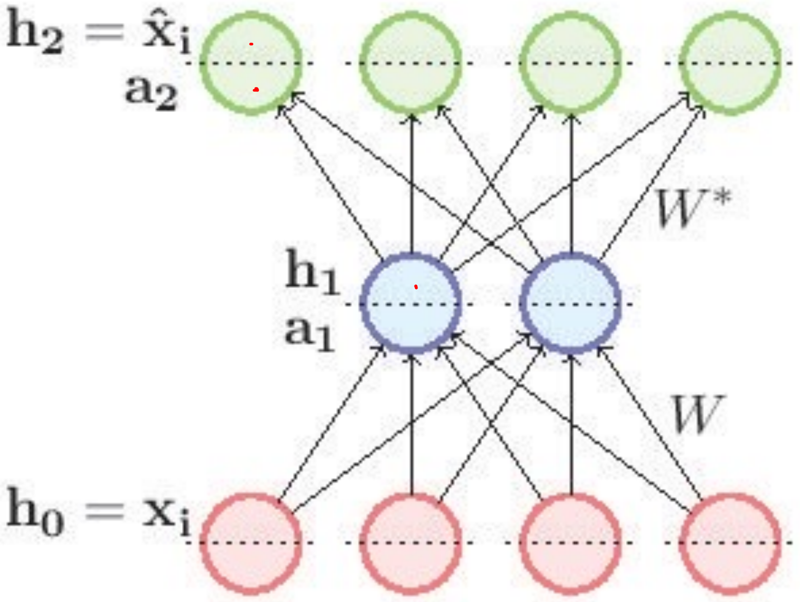
$$\vdots$$

$$+\left(\sigma(a_d)_{i=1} - y_{i=1,d}\right)^2$$

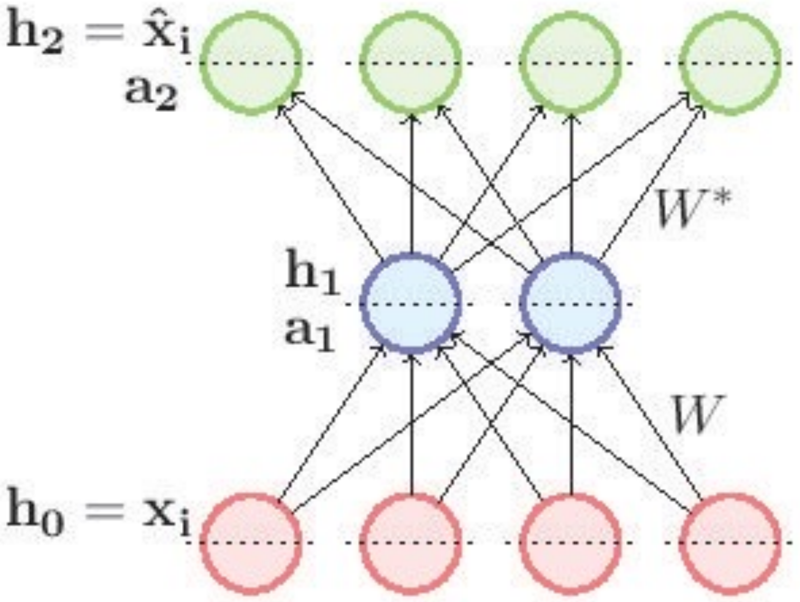
d is the dim, or represent n

$n = d$

$$\mathscr{L}(\theta) = (\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})^T(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})$$

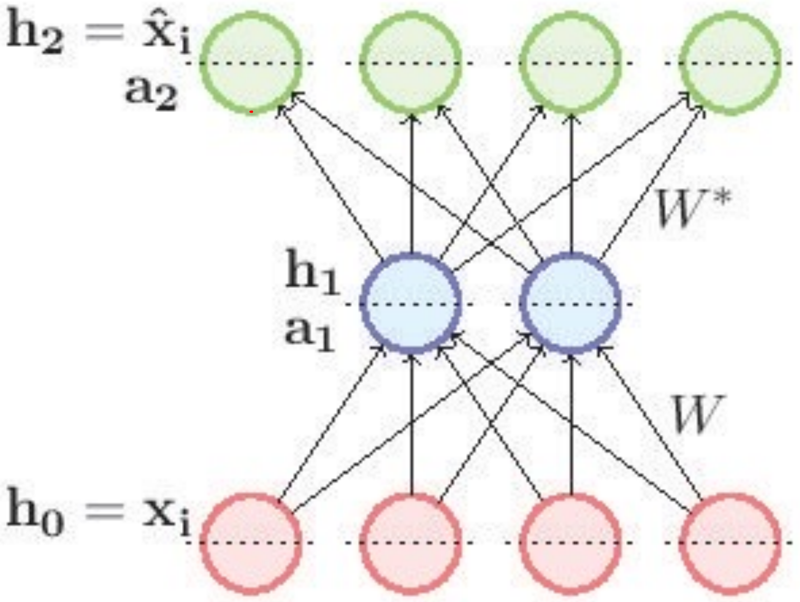


$$\mathscr{L}(\theta) = (\hat{\mathbf{x}}_{\mathbf{i}} - \mathbf{x}_{\mathbf{i}})^T(\hat{\mathbf{x}}_{\mathbf{i}} - \mathbf{x}_{\mathbf{i}})$$



- Note that the loss function is shown for only one training example.

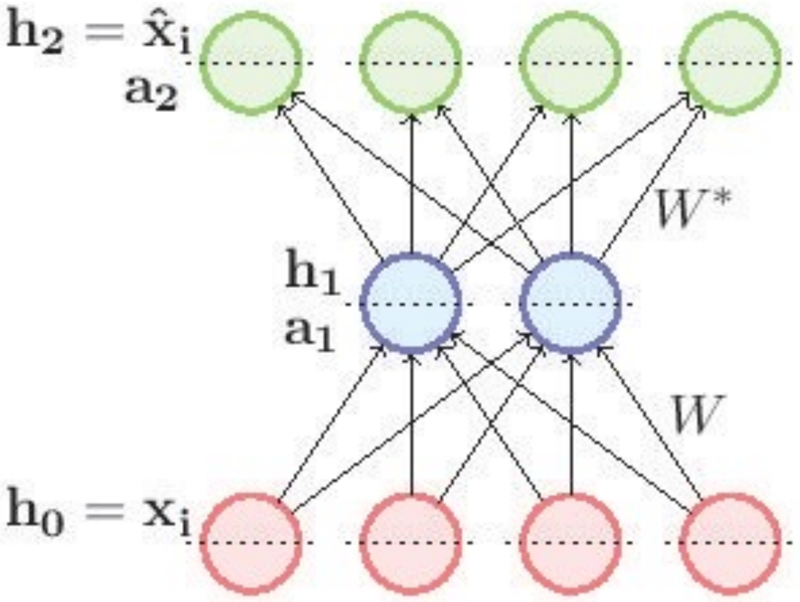
$$\mathscr{L}(\theta) = (\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})^T(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})$$



- Note that the loss function is shown for only one training example.

- $$\frac{\partial \mathscr{L}(\theta)}{\partial W^*} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \boxed{\frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \frac{\partial \mathbf{a_2}}{\partial W^*}}$$

$$\mathscr{L}(\theta) = (\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})^T(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})$$

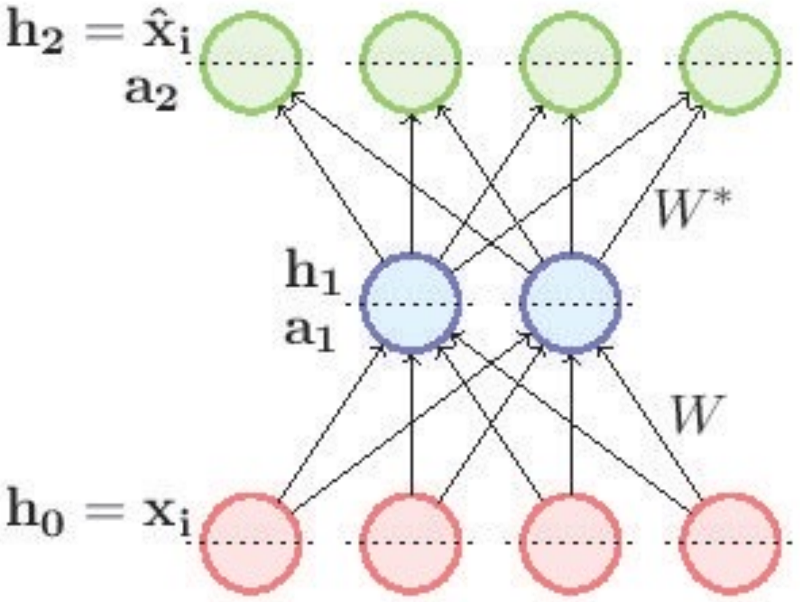


- Note that the loss function is shown for only one training example.

- $$\frac{\partial \mathscr{L}(\theta)}{\partial W^*} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \boxed{\frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \frac{\partial \mathbf{a_2}}{\partial W^*}}$$

- $$\frac{\partial \mathscr{L}(\theta)}{\partial W} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \boxed{\frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \frac{\partial \mathbf{a_2}}{\partial \mathbf{h_1}} \frac{\partial \mathbf{h_1}}{\partial \mathbf{a_1}} \frac{\partial \mathbf{a_1}}{\partial W}}$$

$$\mathscr{L}(\theta) = (\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})^T (\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})$$



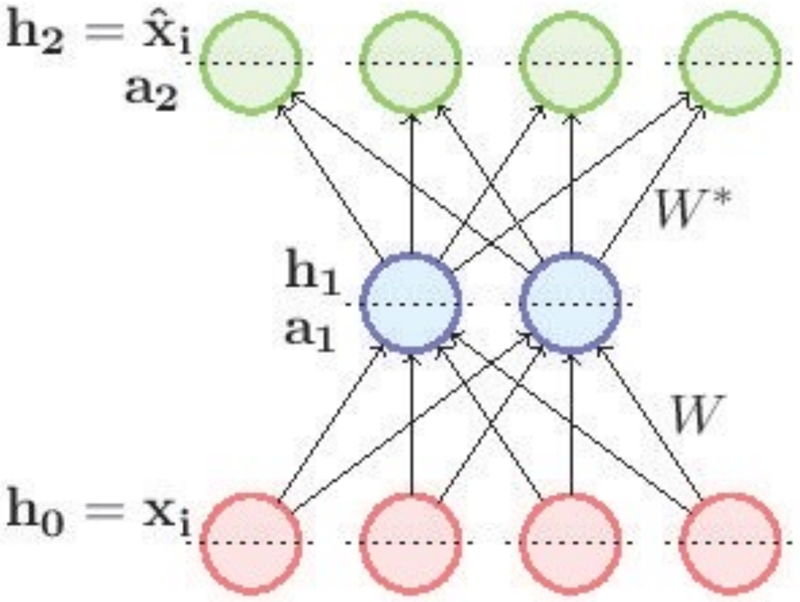
- Note that the loss function is shown for only one training example.

- $$\frac{\partial \mathscr{L}(\theta)}{\partial W^*} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \boxed{\frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \frac{\partial \mathbf{a_2}}{\partial W^*}}$$

- $$\frac{\partial \mathscr{L}(\theta)}{\partial W} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \boxed{\frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \frac{\partial \mathbf{a_2}}{\partial \mathbf{h_1}} \frac{\partial \mathbf{h_1}}{\partial \mathbf{a_1}} \frac{\partial \mathbf{a_1}}{\partial W}}$$

- We have already seen how to calculate the expression in the boxes when we learnt backpropagation

$$\mathscr{L}(\theta) = (\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})^T(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})$$



- Note that the loss function is shown for only one training example.

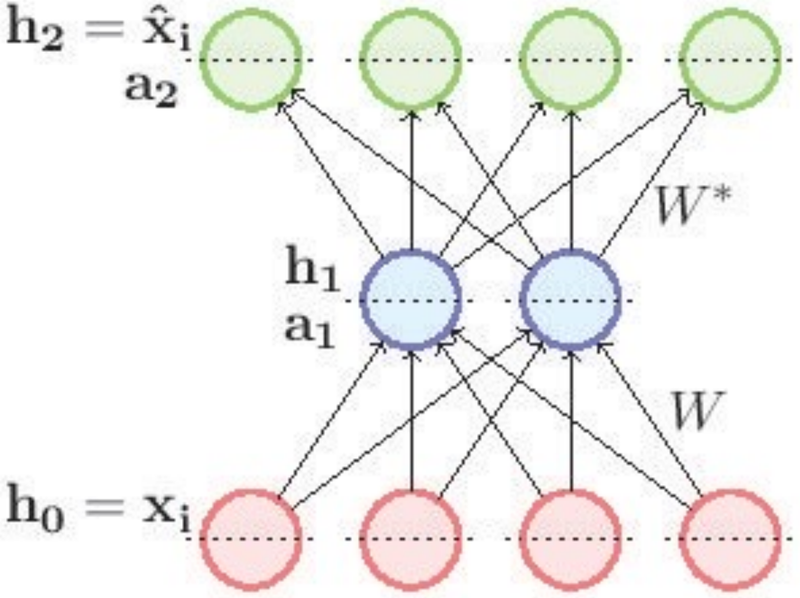
- $$\frac{\partial \mathscr{L}(\theta)}{\partial W^*} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \boxed{\frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \frac{\partial \mathbf{a_2}}{\partial W^*}}$$

- $$\frac{\partial \mathscr{L}(\theta)}{\partial W} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \boxed{\frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \frac{\partial \mathbf{a_2}}{\partial \mathbf{h_1}} \frac{\partial \mathbf{h_1}}{\partial \mathbf{a_1}} \frac{\partial \mathbf{a_1}}{\partial W}}$$

- We have already seen how to calculate the expression in the boxes when we learnt backpropagation

$$\frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} = \frac{\partial \mathscr{L}(\theta)}{\partial \hat{\mathbf{x}}_\mathbf{i}}$$

$$\mathscr{L}(\theta) = (\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})^T(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})$$



- Note that the loss function is shown for only one training example.

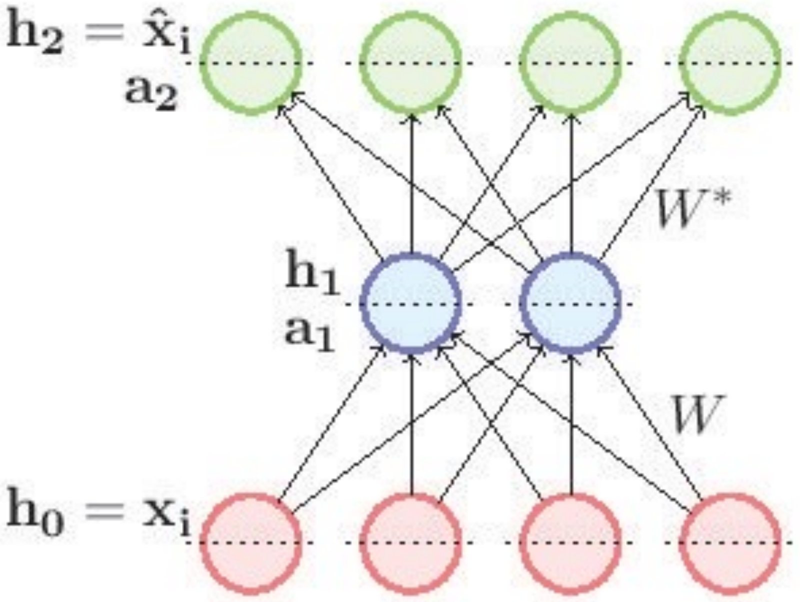
- $$\frac{\partial \mathscr{L}(\theta)}{\partial W^*} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \boxed{\frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \frac{\partial \mathbf{a_2}}{\partial W^*}}$$

- $$\frac{\partial \mathscr{L}(\theta)}{\partial W} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \boxed{\frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \frac{\partial \mathbf{a_2}}{\partial \mathbf{h_1}} \frac{\partial \mathbf{h_1}}{\partial \mathbf{a_1}} \frac{\partial \mathbf{a_1}}{\partial W}}$$

- We have already seen how to calculate the expression in the boxes when we learnt backpropagation

$$\begin{aligned} \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} &= \frac{\partial \mathscr{L}(\theta)}{\partial \hat{\mathbf{x}}_\mathbf{i}} \\ &= \nabla_{\hat{\mathbf{x}}_\mathbf{i}}\{(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x_i})^T(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x_i})\} \end{aligned}$$

$$\mathscr{L}(\theta) = (\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})^T(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})$$



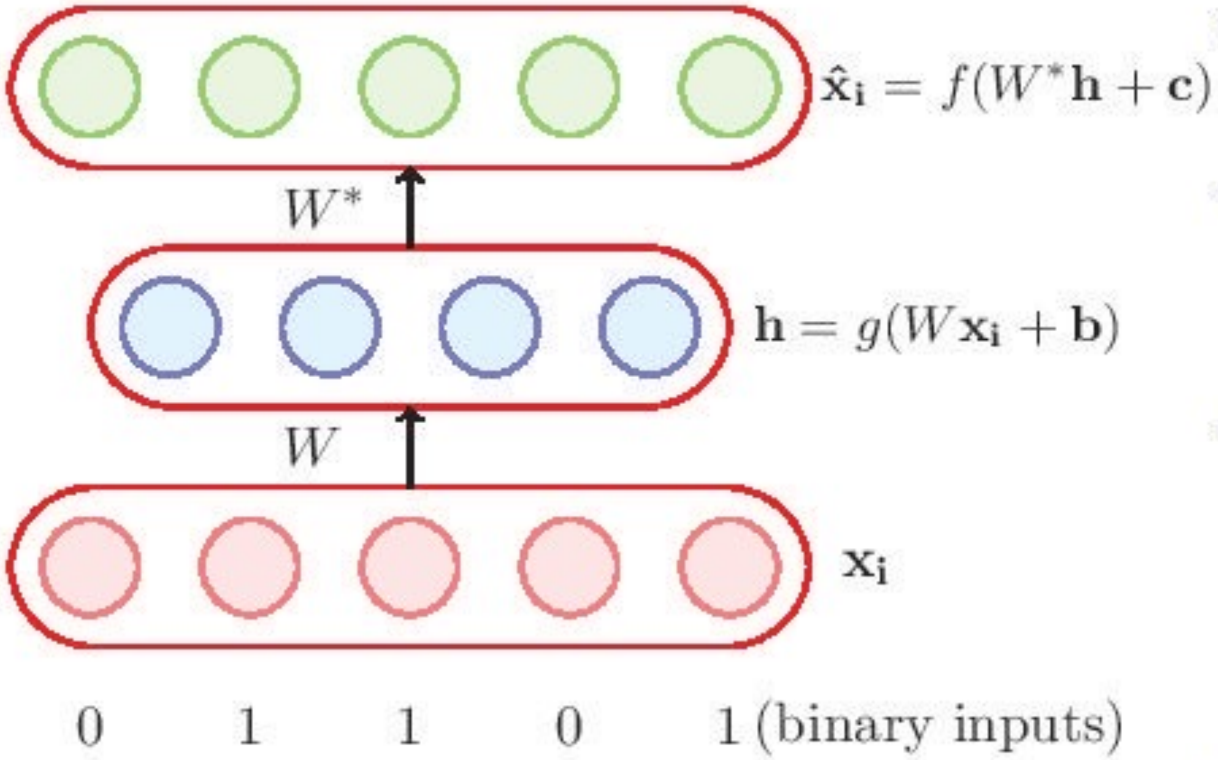
- Note that the loss function is shown for only one training example.

- $$\frac{\partial \mathscr{L}(\theta)}{\partial W^*} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \boxed{\frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \frac{\partial \mathbf{a_2}}{\partial W^*}}$$

- $$\frac{\partial \mathscr{L}(\theta)}{\partial W} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \boxed{\frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \frac{\partial \mathbf{a_2}}{\partial \mathbf{h_1}} \frac{\partial \mathbf{h_1}}{\partial \mathbf{a_1}} \frac{\partial \mathbf{a_1}}{\partial W}}$$

- We have already seen how to calculate the expression in the boxes when we learnt backpropagation

$$\begin{aligned}
\frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} &= \frac{\partial \mathscr{L}(\theta)}{\partial \hat{\mathbf{x}}_\mathbf{i}} \\
&= \nabla_{\hat{\mathbf{x}}_\mathbf{i}} \{(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x_i})^T(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x_i})\} \\
&= 2(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x_i})
\end{aligned}$$



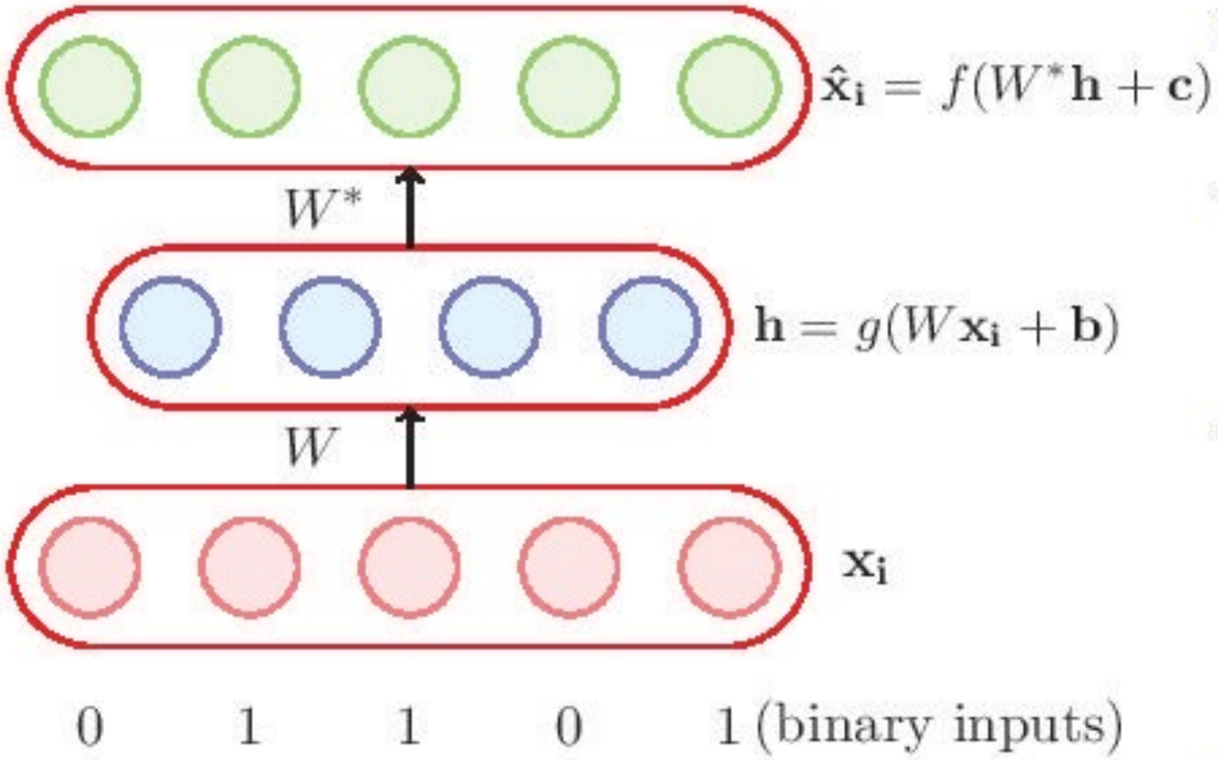
What value of $\hat{x}_{ij}$ will minimize this function?

- If $x_{ij} = 1$?
- If $x_{ij} = 0$?

- Consider the case when the inputs are binary
- We use a sigmoid decoder which will produce outputs between 0 and 1, and can be interpreted as probabilities.
- For a single n-dimensional i^{th} input we can use the following loss function

$$\min\{-\sum_{j=1}^{n}(x_{ij}\log\hat{x}_{ij} + (1 - x_{ij})\log(1 - \hat{x}_{ij}))\}$$

- Again we need is a formula for $\frac{\partial \mathscr{L}(\theta)}{\partial W^*}$ and $\frac{\partial \mathscr{L}(\theta)}{\partial W}$ to use backpropagation



What value of $\hat{x}_{ij}$ will minimize this function?

- If $x_{ij} = 1$?
- If $x_{ij} = 0$?

Indeed the above function will be minimized when $\hat{x}_{ij} = x_{ij}$!

- Consider the case when the inputs are binary
- We use a sigmoid decoder which will produce outputs between 0 and 1, and can be interpreted as probabilities.
- For a single n-dimensional i^{th} input we can use the following loss function

$$\min\{-\sum_{j=1}^{n}(x_{ij}\log\hat{x}_{ij} + (1 - x_{ij})\log(1 - \hat{x}_{ij}))\}$$

- Again we need is a formula for $\frac{\partial \mathscr{L}(\theta)}{\partial W^*}$ and $\frac{\partial \mathscr{L}(\theta)}{\partial W}$ to use backpropagation