Autoencoders and relation to PCA, Regularization in autoencoders, Denoising autoencoders, Sparse autoencoders, Contractive autoencoders

Mitesh M. Khapra

Department of Computer Science and Engineering
Indian Institute of Technology Madras
Module 7.1: Introduction to Autoencoders
\[ \hat{x}_i = W^* x_i \]
\[ \hat{x}_i = f(W^* h + c) \]
An autoencoder is a special type of feed forward neural network which does the following:

- Encodes its input $x_i$ into a hidden representation $h$.
- Decodes the input again from this hidden representation $\hat{x}_i$.

The model is trained to minimize a certain loss function which will ensure that $\hat{x}_i$ is close to $x_i$ (we will see some such loss functions soon).
An autoencoder is a special type of feed forward neural network which does the following:

- **Encodes** its input $x_i$ into a hidden representation $h$

$$\hat{x}_i = g(W^* h + b)$$

$$\hat{x}_i = f(W^* h + c)$$
An autoencoder is a special type of feed forward neural network which does the following:

- **Encodes** its input \( x_i \) into a hidden representation \( h \)

The model is trained to minimize a certain loss function which will ensure that \( \hat{x}_i \) is close to \( x_i \) (we will see some such loss functions soon)

Mathematically, this can be represented as:

\[
h = g(Wx_i + b)
\]

\[
\hat{x}_i = f(W^*h + c)
\]
An autoencoder is a special type of feed forward neural network which does the following:

- Encodes its input $x_i$ into a hidden representation $h$
- Decodes the input again from this hidden representation

The model is trained to minimize a certain loss function which will ensure that $\hat{x}_i$ is close to $x_i$ (we will see some such loss functions soon).
An autoencoder is a special type of feed forward neural network which does the following:

- **Encodes** its input $x_i$ into a hidden representation $h$
- **Decodes** the input again from this hidden representation

The model is trained to minimize a certain loss function which will ensure that $\hat{x}_i$ is close to $x_i$ (we will see some such loss functions soon).

Mathematically:

$$h = g(Wx_i + b)$$
$$\hat{x}_i = f(W^*h + c)$$
An autoencoder is a special type of feed forward neural network which does the following:

- **Encodes** its input $x_i$ into a hidden representation $h$
- **Decodes** the input again from this hidden representation
- The model is trained to minimize a certain loss function which will ensure that $\hat{x}_i$ is close to $x_i$ (we will see some such loss functions soon)

$$h = g(Wx_i + b)$$
$$\hat{x}_i = f(W^*h + c)$$
An autoencoder where $\text{dim}(h) < \text{dim}(x)$ is called an undercomplete autoencoder.

$$h = g(Wx_i + b)$$

$$\hat{x}_i = f(W^*h + c)$$
Let us consider the case where $\dim(h) < \dim(x_i)$

\[
\begin{align*}
    h &= g(Wx_i + b) \\
    \hat{x}_i &= f(W^*h + c)
\end{align*}
\]
Let us consider the case where \( \dim(h) < \dim(x_i) \).

If we are still able to reconstruct \( \hat{x}_i \) perfectly from \( h \), then what does it say about \( h \)?

\[
\begin{align*}
  h &= g(Wx_i + b) \\
  \hat{x}_i &= f(W^*h + c)
\end{align*}
\]
Let us consider the case where $\text{dim}(h) < \text{dim}(x_i)$

If we are still able to reconstruct $\hat{x}_i$ perfectly from $h$, then what does it say about $h$?

$h$ is a loss-free encoding of $x_i$. It captures all the important characteristics of $x_i$

\[ h = g(Wx_i + b) \]
\[ \hat{x}_i = f(W^*h + c) \]
Let us consider the case where \( \dim(h) < \dim(x_i) \).

If we are still able to reconstruct \( \hat{x}_i \) perfectly from \( h \), then what does it say about \( h \)?

- \( h \) is a loss-free encoding of \( x_i \). It captures all the important characteristics of \( x_i \).
- Do you see an analogy with PCA?
Let us consider the case where \( \text{dim}(h) < \text{dim}(x_i) \).

If we are still able to reconstruct \( \hat{x}_i \) perfectly from \( h \), then what does it say about \( h \)?

\( h \) is a loss-free encoding of \( x_i \). It captures all the important characteristics of \( x_i \).

Do you see an analogy with PCA?

An autoencoder where \( \text{dim}(h) < \text{dim}(x_i) \) is called an **under complete** autoencoder.

\[
\begin{align*}
    h &= g(Wx_i + b) \\
    \hat{x}_i &= f(W^*h + c)
\end{align*}
\]
An autoencoder where $\dim(h) \geq \dim(x)$ is called an overcomplete autoencoder.

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

\[ h = g(Wx_i + b) \]
\[ \hat{x}_i = f(W^*h + c) \]
Let us consider the case when \( \dim(h) \geq \dim(x_i) \)

\[
\begin{align*}
    h &= g(W x_i + b) \\
    \hat{x}_i &= f(W^* h + c)
\end{align*}
\]
A autoencoder where \( \text{dim}(h) \geq \text{dim}(x_i) \) is called an overcomplete autoencoder.

Let us consider the case when \( \text{dim}(h) \geq \text{dim}(x_i) \)

In such a case the autoencoder could learn a trivial encoding by simply copying \( x_i \) into \( h \) and then copying \( h \) into \( \hat{x}_i \)

\[
\hat{x}_i = f(W^*h + c)
\]

\[
h = g(Wx_i + b)
\]
\[
\hat{x}_i = f(W^* h + c)
\]

Let us consider the case when \( \dim(h) \geq \dim(x_i) \).

In such a case the autoencoder could learn a trivial encoding by simply copying \( x_i \) into \( h \) and then copying \( h \) into \( \hat{x}_i \).
\[ \hat{x}_i = f(W^*h + c) \]

Let us consider the case when \( \text{dim}(h) \geq \text{dim}(x_i) \)

In such a case the autoencoder could learn a trivial encoding by simply copying \( x_i \) into \( h \) and then copying \( h \) into \( \hat{x}_i \)

In practice, such an identity encoding is useless as it does not really tell us anything about the important characteristics of the data.
\[
\hat{x}_i = f(W^*h + c)
\]

Let us consider the case when \( \text{dim}(h) \geq \text{dim}(x_i) \).

In such a case the autoencoder could learn a trivial encoding by simply copying \( x_i \) into \( h \) and then copying \( h \) into \( \hat{x}_i \).

Such an identity encoding is useless in practice as it does not really tell us anything about the important characteristics of the data.
Let us consider the case when $\dim(h) \geq \dim(x_i)$.

In such a case the autoencoder could learn a trivial encoding by simply copying $x_i$ into $h$ and then copying $h$ into $\hat{x}_i$.

Such an identity encoding is useless in practice as it does not really tell us anything about the important characteristics of the data.

$$h = g(Wx_i + b)$$
$$\hat{x}_i = f(W^*h + c)$$
Let us consider the case when $\text{dim}(h) \geq \text{dim}(x_i)$.

In such a case the autoencoder could learn a trivial encoding by simply copying $x_i$ into $h$ and then copying $h$ into $\hat{x}_i$.

Such an identity encoding is useless in practice as it does not really tell us anything about the important characteristics of the data.
An autoencoder where $\text{dim}(h) \geq \text{dim}(x_i)$ is called an overcomplete autoencoder.

Let us consider the case when $\text{dim}(h) \geq \text{dim}(x_i)$.

In such a case the autoencoder could learn a trivial encoding by simply copying $x_i$ into $h$ and then copying $h$ into $\hat{x}_i$.

Such an identity encoding is useless in practice as it does not really tell us anything about the important characteristics of the data.

\[
h = g(Wx_i + b)\]
\[
\hat{x}_i = f(W^*h + c)\]
The diagram illustrates an autoencoder, where $\text{dim}(h) \geq \text{dim}(x_i)$ is called an overcomplete autoencoder.

Let us consider the case when $\text{dim}(h) \geq \text{dim}(x_i)$. In such a case the autoencoder could learn a trivial encoding by simply copying $x_i$ into $h$ and then copying $h$ into $\hat{x}_i$.

Such an identity encoding is useless in practice as it does not really tell us anything about the important characteristics of the data.

Given:

- $h = g(Wx_i + b)$
- $\hat{x}_i = f(W^*h + c)$
Let us consider the case when $\dim(h) \geq \dim(x_i)$.

In such a case the autoencoder could learn a trivial encoding by simply copying $x_i$ into $h$ and then copying $h$ into $\hat{x}_i$.

$$h = g(Wx_i + b)$$
$$\hat{x}_i = f(W^*h + c)$$
Let us consider the case when \( \dim(h) \geq \dim(x_i) \)

In such a case the autoencoder could learn a trivial encoding by simply copying \( x_i \) into \( h \) and then copying \( h \) into \( \hat{x}_i \).

Such an identity encoding is useless in practice as it does not really tell us anything about the important characteristics of the data.
An autoencoder where $\dim(h) \geq \dim(x_i)$ is called an overcomplete autoencoder.

Let us consider the case when $\dim(h) \geq \dim(x_i)$.

In such a case the autoencoder could learn a trivial encoding by simply copying $x_i$ into $h$ and then copying $h$ into $\hat{x}_i$.

Such an identity encoding is useless in practice as it does not really tell us anything about the important characteristics of the data.

$h = g(Wx_i + b)$

$\hat{x}_i = f(W^*h + c)$
An autoencoder where $\dim(h) \geq \dim(x_i)$ is called an overcomplete autoencoder.

Let us consider the case when $\dim(h) \geq \dim(x_i)$.

In such a case the autoencoder could learn a trivial encoding by simply copying $x_i$ into $h$ and then copying $h$ into $\hat{x}_i$.

Such an identity encoding is useless in practice as it does not really tell us anything about the important characteristics of the data.

\[ h = g(W x_i + b) \]
\[ \hat{x}_i = f(W^* h + c) \]
The Road Ahead
The Road Ahead

- Choice of $f(x_i)$ and $g(x_i)$
The Road Ahead

- Choice of $f(x_i)$ and $g(x_i)$
- Choice of loss function
The Road Ahead

- Choice of $f(x_i)$ and $g(x_i)$
- Choice of loss function
\[ \hat{x}_i = f(W^* h + c) \]

\[ h = g(W x_i + b) \]

\[ x_i \]

0 \hspace{0.2cm} 1 \hspace{0.2cm} 1 \hspace{0.2cm} 0 \hspace{0.2cm} 1 \text{ (binary inputs)}
Suppose all our inputs are binary (each $x_{ij} \in \{0,1\}$)

- $\hat{x}_i = f(W^* h + c)$
- $h = g(Wx_i + b)$
- $x_i$

0 1 1 0 1 (binary inputs)
\[ x_i = f(W^* h + c) \]
\[ h = g(Wx_i + b) \]
\[ W^* \]

Suppose all our inputs are binary (each \( x_{ij} \in \{0, 1\} \)).

Which of the following functions would be most apt for the decoder?

- \( \hat{x}_i = \tanh(W^* h + c) \)
- \( \hat{x}_i = W^* h + c \)
- \( \hat{x}_i = \logistic(W^* h + c) \)

Logistic as it naturally restricts all outputs to be between 0 and 1.
Suppose all our inputs are binary (each $x_{ij} \in \{0, 1\}$)

Which of the following functions would be most apt for the decoder?

$$\hat{x}_i = \tanh(W^* h + c)$$
\[ \hat{x}_i = f(W^*h + c) \]

Suppose all our inputs are binary (each \( x_{ij} \in \{0, 1\} \))

Which of the following functions would be most apt for the decoder?

- \( \hat{x}_i = \tanh(W^*h + c) \)
- \( \hat{x}_i = W^*h + c \)

Logistic as it naturally restricts all outputs to be between 0 and 1.
\[
\hat{x}_i = f(W^*h + c)
\]

Suppose all our inputs are binary (each \(x_{ij} \in \{0, 1\}\))

Which of the following functions would be most apt for the decoder?

\[
\hat{x}_i = \tanh(W^*h + c)
\]
\[
\hat{x}_i = W^*h + c
\]
\[
\hat{x}_i = \text{logistic}(W^*h + c)
\]

Logistic as it naturally restricts all outputs to be between 0 and 1
\[ \hat{x}_i = f(W^*h + c) \]

\[ h = g(Wx_i + b) \]

\[ \hat{x}_i = \text{tanh}(W^*h + c) \]

\[ \hat{x}_i = W^*h + c \]

\[ \hat{x}_i = \text{logistic}(W^*h + c) \]

- Suppose all our inputs are binary (each \( x_{ij} \in \{0, 1\} \))
- Which of the following functions would be most apt for the decoder?
- Logistic as it naturally restricts all outputs to be between 0 and 1
\[ \hat{x}_i = f(W\ast h + c) \]

\[ h = g(Wx_i + b) \]

\[ \hat{x}_i = \text{tanh}(W\ast h + c) \]
\[ \hat{x}_i = W\ast h + c \]
\[ \hat{x}_i = \text{logistic}(W\ast h + c) \]

- Suppose all our inputs are binary (each \( x_{ij} \in \{0, 1\} \))
- Which of the following functions would be most apt for the decoder?
- Logistic as it naturally restricts all outputs to be between 0 and 1

\( g \) is typically chosen as the sigmoid function
\[ \hat{x}_i = f(W^* h + c) \]

\[ h = g(Wx_i + b) \]

\[ W \]

\[ W^* \]

\[ x_i \]

0.25  0.5  1.25  3.5  4.5

(real valued inputs)

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?

\[ \hat{x}_i = \tanh(W^* h + c) \]

\[ \hat{x}_i = W^* h + c \]

\[ \hat{x}_i = \text{logistic}(W^* h + c) \]

What will logistic and tanh do? They will restrict the reconstructed \( \hat{x}_i \) to lie between \([0,1]\) or \([-1,1]\) whereas we want \( \hat{x}_i \in \mathbb{R}^n \).
\( \hat{x}_i = f(W^* h + c) \)

Suppose all our inputs are real (each \( x_{ij} \in \mathbb{R} \))

- \( h = g(Wx_i + b) \)
- \( \hat{x}_i = W^* h + c \)
- \( \hat{x}_i = \text{logistic}(W^* h + c) \)

What will logistic and tanh do? They will restrict the reconstructed \( \hat{x}_i \) to lie between \([0,1]\) or \([-1,1]\) whereas we want \( \hat{x}_i \in \mathbb{R}^n \) (real valued inputs)

\[ \begin{array}{cccccc}
0.25 & 0.5 & 1.25 & 3.5 & 4.5 \\
\end{array} \]
\( \hat{x}_i = f(W^* h + c) \)

Suppose all our inputs are real (each \( x_{ij} \in \mathbb{R} \))

Which of the following functions would be most apt for the decoder?

- \( \hat{x}_i = \text{tanh}(W^* h + c) \)
- \( \hat{x}_i = W^* h + c \)
- \( \hat{x}_i = \text{logistic}(W^* h + c) \)

What will logistic and tanh do?

They will restrict the reconstructed \( \hat{x}_i \) to lie between \([0,1]\) or \([-1,1]\) whereas we want \( \hat{x}_i \in \mathbb{R}^n \).

(real valued inputs)
\[ \hat{x}_i = f(W^*h + c) \]

\[ h = g(Wx_i + b) \]

Suppose all our inputs are real (each \( x_{ij} \in \mathbb{R} \))

Which of the following functions would be most apt for the decoder?

\[ \hat{x}_i = \tanh(W^*h + c) \]

What will \( \text{logistic} \) and \( \text{tanh} \) do?

They will restrict the reconstructed \( \hat{x}_i \) to lie between \([0,1]\) or \([-1,1]\) whereas we want \( \hat{x}_i \in \mathbb{R}^n \)
\[ \hat{x}_i = f(W^* h + c) \]

Suppose all our inputs are real (each \( x_{ij} \in \mathbb{R} \))

* Which of the following functions would be most apt for the decoder?

\[ \hat{x}_i = \tanh(W^* h + c) \]
\[ \hat{x}_i = W^* h + c \]

What will \( \text{logistic} \) and \( \text{tanh} \) do?

They will restrict the reconstructed \( \hat{x}_i \) to lie between \([0,1]\) or \([-1,1]\) whereas we want \( \hat{x}_i \in \mathbb{R}^n \).
\[ \hat{x}_i = f(W^*h + c) \]
\[ h = g(Wx_i + b) \]
\[ W^* \]
\[ W \]
\[ x_i \]
\[ 0.25 \quad 0.5 \quad 1.25 \quad 3.5 \quad 4.5 \]

(real valued inputs)

- Suppose all our inputs are real (each \( x_{ij} \in \mathbb{R} \))
- Which of the following functions would be most apt for the decoder?
  
  \[ \hat{x}_i = \tanh(W^*h + c) \]
  \[ \hat{x}_i = W^*h + c \]
  \[ \hat{x}_i = \text{logistic}(W^*h + c) \]
\[ \hat{x}_i = f(W^* h + c) \]

\[ h = g(W x_i + b) \]

\[ x_i \]

0.25  0.5  1.25  3.5  4.5  
(real valued inputs)

- Suppose all our inputs are real (each \( x_{ij} \in \mathbb{R} \))
- Which of the following functions would be most apt for the decoder?
  \[ \hat{x}_i = \text{tanh}(W^* h + c) \]
  \[ \hat{x}_i = W^* h + c \]
  \[ \hat{x}_i = \text{logistic}(W^* h + c) \]
- What will logistic and tanh do?

They will restrict the reconstructed \( \hat{x}_i \) to lie between \([0,1]\) or \([-1,1]\) whereas we want \( \hat{x}_i \in \mathbb{R}^n \).
\[ \hat{x}_i = f(W^* h + c) \]

\[ h = g(Wx_i + b) \]

\[ W^* \]

\[ x_i \]

\[ 0.25 \quad 0.5 \quad 1.25 \quad 3.5 \quad 4.5 \]

(real valued inputs)

- Suppose all our inputs are real (each \( x_{ij} \in \mathbb{R} \))
- Which of the following functions would be most apt for the decoder?
  - \( \hat{x}_i = \tanh(W^* h + c) \)
  - \( \hat{x}_i = W^* h + c \)
  - \( \hat{x}_i = \text{logistic}(W^* h + c) \)

- What will \( \text{logistic} \) and \( \tanh \) do?
  - They will restrict the reconstructed \( \hat{x}_i \) to lie between \([0,1]\) or \([-1,1]\) whereas we want \( \hat{x}_i \in \mathbb{R}^n \)
\[ x_i = f(W^*h + c) \]

\[ h = g(Wx_i + b) \]

0.25 0.5 1.25 3.5 4.5
(real valued inputs)

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?
  
  \[
  \hat{x}_i = \tanh(W^*h + c) \\
  \hat{x}_i = W^*h + c \\
  \hat{x}_i = \text{logistic}(W^*h + c)
  \]

- What will logistic and tanh do?
- They will restrict the reconstructed \( \hat{x}_i \) to lie between \([0,1]\) or \([-1,1]\) whereas we want \( \hat{x}_i \in \mathbb{R}^n \)
The Road Ahead

- Choice of $f(x_i)$ and $g(x_i)$
- Choice of loss function
Consider the case when the inputs are real valued. The objective of the autoencoder is to reconstruct \( \hat{x}_i \) to be as close to \( x_i \) as possible. This can be formalized using the following objective function:

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

\( \text{i.e.,} \)

\[
\min_{W, W^*, c, b} \frac{1}{m} \sum_i (\hat{x}_i - x_i)^T (\hat{x}_i - x_i)
\]

We can then train the autoencoder just like a regular feedforward network using backpropagation. All we need is a formula for

\[
\frac{\partial L(\theta)}{\partial W^*} \quad \text{and} \quad \frac{\partial L(\theta)}{\partial W}
\]

which we will see now.
Consider the case when the inputs are real valued

\[
\hat{x}_i = f(W^* h + c)
\]

\[
h = g(W x_i + b)
\]

The objective of the autoencoder is to reconstruct \(\hat{x}_i\) to be as close to \(x_i\) as possible.
Consider the case when the inputs are real valued

- The objective of the autoencoder is to reconstruct $\hat{x}_i$ to be as close to $x_i$ as possible

Differentiating the reconstruction error with respect to the weights and biases gives us the gradients we need for training the autoencoder using backpropagation.

$$h = g(Wx_i + b)$$

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

$$\min_{W, W^*, c, b} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2$$
\[
\hat{x}_i = \hat{x}'_i = f(W^* h + c)
\]

- Consider the case when the inputs are real valued.
- The objective of the autoencoder is to reconstruct \( \hat{x}_i \) to be as close to \( x_i \) as possible.
- This can be formalized using the following objective function:

\[
\min_{W, W^*, c, b} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2
\]
Consider the case when the inputs are real valued.

The objective of the autoencoder is to reconstruct $\hat{x}_i$ to be as close to $x_i$ as possible.

This can be formalized using the following objective function:

$$
\min_{W, W^*, c, b} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2 \\
\text{i.e.,} \\
\min_{W, W^*, c, b} \frac{1}{m} \sum_{i=1}^{m} (\hat{x}_i - x_i)^T (\hat{x}_i - x_i)
$$
Consider the case when the inputs are real valued

The objective of the autoencoder is to reconstruct $\hat{x}_i$ to be as close to $x_i$ as possible

This can be formalized using the following objective function:

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

i.e.,

$$
\min_{W, W^*, c, b} \frac{1}{m} \sum_{i=1}^{m} (\hat{x}_i - x_i)^T (\hat{x}_i - x_i)
$$

We can then train the autoencoder just like a regular feedforward network using backpropagation
Consider the case when the inputs are real valued

The objective of the autoencoder is to reconstruct \( \hat{x}_i \) to be as close to \( x_i \) as possible.

This can be formalized using the following objective function:

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

\[i.e., \min_{W, W^*, c, b} \frac{1}{m} \sum_{i=1}^{m} (\hat{x}_i - x_i)^T (\hat{x}_i - x_i)\]

We can then train the autoencoder just like a regular feedforward network using backpropagation.

All we need is a formula for \( \frac{\partial L(\theta)}{\partial W^*} \) and \( \frac{\partial L(\theta)}{\partial W} \) which we will see now.
\[ L(\theta) = (\hat{x}_i - x_i)^T(\hat{x}_i - x_i) \]

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

We have already seen how to calculate the expressions in the boxes when we learnt backpropagation.
\[ \mathcal{L}(\theta) = (\hat{x}_i - x_i)^T(\hat{x}_i - x_i) \]

Note that the loss function is shown for only one training example.
\[ \mathcal{L}(\theta) = (\hat{x}_i - x_i)^T(\hat{x}_i - x_i) \]

\[ h_0 = x_i \]
\[ h_1 \]
\[ a_1 \]
\[ h_2 = \hat{x}_i \]
\[ a_2 \]
\[ W \]
\[ W^* \]

\[ \frac{\partial \mathcal{L}(\theta)}{\partial W^*} = \frac{\partial \mathcal{L}(\theta)}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial W^*} \]

- Note that the loss function is shown for only one training example.
\[ L(\theta) = (\hat{x}_i - x_i)^T(\hat{x}_i - x_i) \]

\[ h_2 = \hat{x}_i \]
\[ a_2 \]
\[ W^* \]
\[ h_1 \]
\[ a_1 \]
\[ W \]
\[ h_0 = x_i \]

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

\[ \frac{\partial L(\theta)}{\partial W^*} = \frac{\partial L(\theta)}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial W^*} \]

\[ \frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1} \frac{\partial a_1}{\partial W} \]
\[ \mathcal{L}(\theta) = (\hat{x}_i - x_i)^T(\hat{x}_i - x_i) \]

\[ h_2 = \hat{x}_i \]
\[ a_2 \]
\[ h_1 \]
\[ a_1 \]
\[ W^* \]
\[ W \]
\[ h_0 = x_i \]

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

\[ \frac{\partial \mathcal{L}(\theta)}{\partial W^*} = \frac{\partial \mathcal{L}(\theta)}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial W^*} \]

\[ \frac{\partial \mathcal{L}(\theta)}{\partial W} = \frac{\partial \mathcal{L}(\theta)}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1} \frac{\partial a_1}{\partial W} \]

We have already seen how to calculate the expression in the boxes when we learnt backpropagation.
\[ \mathcal{L}(\theta) = (\hat{x}_i - x_i)^T(\hat{x}_i - x_i) \]

\[ h_0 = x_i \]
\[ h_1 \]
\[ a_1 \]
\[ W \]
\[ h_2 = \hat{x}_i \]
\[ a_2 \]

\[ \frac{\partial \mathcal{L}(\theta)}{\partial W^*} = \frac{\partial \mathcal{L}(\theta)}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial W^*} \]

\[ \frac{\partial \mathcal{L}(\theta)}{\partial W} = \frac{\partial \mathcal{L}(\theta)}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1} \frac{\partial a_1}{\partial W} \]

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

\[ \frac{\partial \mathcal{L}(\theta)}{\partial h_2} = \frac{\partial \mathcal{L}(\theta)}{\partial \hat{x}_i} \]

Note that the loss function is shown for only one training example.
\[ \mathcal{L}(\theta) = (\hat{x}_i - x_i)^T(\hat{x}_i - x_i) \]

\[ h_0 = x_i \]
\[ h_2 = \hat{x}_i \]
\[ a_2 \]
\[ W^* \]
\[ W \]

\[ h_1 \]
\[ a_1 \]

\[ \nabla_{\hat{x}_i} \{ (\hat{x}_i - x_i)^T(\hat{x}_i - x_i) \} \]

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

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

\[ \frac{\partial \mathcal{L}(\theta)}{\partial W^*} = \frac{\partial \mathcal{L}(\theta)}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial W^*} \]

\[ \frac{\partial \mathcal{L}(\theta)}{\partial W} = \frac{\partial \mathcal{L}(\theta)}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1} \frac{\partial a_1}{\partial W} \]
\[ \mathcal{L}(\theta) = (\hat{x}_i - x_i)^T(\hat{x}_i - x_i) \]

\[ h_2 = \hat{x}_i \]
\[ a_2 \]
\[ \ldots \]
\[ h_0 = x_i \]
\[ a_1 \]
\[ \ldots \]

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

- \[ \frac{\partial \mathcal{L}(\theta)}{\partial W^*} = \frac{\partial \mathcal{L}(\theta)}{\partial h_2} \begin{bmatrix} \partial h_2 \\ \partial a_2 \\ \partial W^* \end{bmatrix} \]

- \[ \frac{\partial \mathcal{L}(\theta)}{\partial W} = \frac{\partial \mathcal{L}(\theta)}{\partial h_2} \begin{bmatrix} \partial h_2 \\ \partial a_2 \\ \partial h_1 \\ \partial a_1 \\ \partial W \end{bmatrix} \]

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

\[ \frac{\partial \mathcal{L}(\theta)}{\partial h_2} = \frac{\partial \mathcal{L}(\theta)}{\partial \hat{x}_i} \]
\[ = \nabla_{\hat{x}_i} \{ (\hat{x}_i - x_i)^T(\hat{x}_i - x_i) \} \]
\[ = 2(\hat{x}_i - x_i) \]
\[ f(x_i) = W^* h + c \]

\[ h = g(Wx_i + b) \]

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:

\[
\min_{\theta} \left\{ -n \sum_{j=1}^{n} \left( x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log(1 - \hat{x}_{ij}) \right) \right\}
\]

Again we need is a formula for \( \frac{\partial L(\theta)}{\partial W^*} \) and \( \frac{\partial L(\theta)}{\partial W} \) to use backpropagation.
Consider the case when the inputs are binary.

\[
\hat{x}_i = f(W^*h + c)
\]

\[
h = g(Wx_i + b)
\]

0 1 1 0 1 (binary inputs)
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.
\[ \hat{x}_i = f(W\mathbf{h} + \mathbf{c}) \]

\[ h = g(W\mathbf{x}_i + \mathbf{b}) \]

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:

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

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

(binary inputs)
\[ \hat{x}_i = f(W^* h + c) \]

\[ h = g(W x_i + b) \]

\[ W^* \]

\[ W \]

\[ x_i \]

0 1 1 0 1 (binary inputs)

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

- 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:

\[
\min \left\{ - \sum_{j=1}^{n} (x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log(1 - \hat{x}_{ij})) \right\}
\]
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 $\frac{\partial L(\theta)}{\partial W^*}$ and $\frac{\partial L(\theta)}{\partial W}$ to use backpropagation

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

- If $x_{ij} = 1$?

\[h = g(Wx_i + b)\]

\[\hat{x}_i = f(W^*h + c)\]
\[ \hat{x}_i = f(W^* h + c) \]

\[ h = g(W x_i + b) \]

\[ x_i \]

0 1 1 0 1 (binary inputs)

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:

\[
\min \{- \sum_{j=1}^{n} (x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log(1 - \hat{x}_{ij}))\}
\]
\[ \hat{x}_i = f(W^* h + c) \]

\[ h = g(W x_i + b) \]

\[ 0 \quad 1 \quad 1 \quad 0 \quad 1 \text{ (binary inputs)} \]

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

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

- Again we need is a formula for \( \frac{\partial L(\theta)}{\partial W^*} \) and \( \frac{\partial L(\theta)}{\partial W} \) to use backpropagation
\[ \hat{x}_i = f(W^*h + c) \]

\[ h = g(Wx_i + b) \]

\[
W^* \\
W
\]

\[ x_i \]

\[
0 \quad 1 \quad 1 \quad 0 \quad 1 \text{ (binary inputs)}
\]

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

\[
\text{min} \left\{ - \sum_{j=1}^{n} \left( x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log(1 - \hat{x}_{ij}) \right) \right\}
\]

- Again we need is a formula for \( \frac{\partial L(\theta)}{\partial W^*} \) and \( \frac{\partial L(\theta)}{\partial W} \) to use backpropagation
\[ L(\theta) = - \sum_{j=1}^{n} (x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log (1 - \hat{x}_{ij})) \]

\[ h_2 = \hat{x}_i \]

\[ h_1 \]

\[ a_2 \]

\[ W^* \]

\[ h_0 = x_i \]
\( \mathcal{L}(\theta) = - \sum_{j=1}^{n} (x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log (1 - \hat{x}_{ij})) \)

\( h_0 = x_i \)

\( h_2 = \hat{x}_i \)

\( h_1 \)

\( a_1 \)

\( h_2 \)

\( a_2 \)

\( W^* \)

\( W \)

\( \frac{\partial \mathcal{L}(\theta)}{\partial W^*} = \frac{\partial \mathcal{L}(\theta)}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial W^*} \)
\[ L(\theta) = -\sum_{j=1}^{n} (x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log (1 - \hat{x}_{ij})) \]

\[ h_2 = \hat{x}_i \]

\[ a_2 \]

\[ W^* \]

\[ h_1 \]

\[ a_1 \]

\[ W \]

\[ h_0 = x_i \]

\[ \frac{\partial L(\theta)}{\partial W^*} = \frac{\partial L(\theta)}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial W^*} \]

\[ \frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1} \frac{\partial a_1}{\partial W} \]
\[ L(\theta) = -\sum_{j=1}^{n} (x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log(1 - \hat{x}_{ij})) \]

We have already seen how to calculate the expressions in the square boxes when we learnt BP
\[ L(\theta) = -\sum_{j=1}^{n} (x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log(1 - \hat{x}_{ij})) \]

We have already seen how to calculate the expressions in the square boxes when we learnt BP.

- The first two terms on RHS can be computed as:

\[
\frac{\partial L(\theta)}{\partial h_{2j}} = -x_{ij} \frac{\partial \hat{x}_{ij}}{\partial h_{2j}} + \frac{1 - x_{ij}}{1 - \hat{x}_{ij}} \frac{\partial \hat{x}_{ij}}{\partial h_{2j}} = \sigma(a_{2j})(1 - \sigma(a_{2j}))
\]

\[
\frac{\partial L(\theta)}{\partial h_{2j}} = \frac{\partial L(\theta)}{\partial a_{2j}} \frac{\partial a_{2j}}{\partial h_{2j}}
\]

\[
\frac{\partial L(\theta)}{\partial a_{2j}} = \sigma'(a_{2j})(1 - \sigma(a_{2j}))
\]

\[
\frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial a_{2j}} \frac{\partial a_{2j}}{\partial h_{2j}} \frac{\partial h_{2j}}{\partial W}
\]

\[
\frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_{2j}} \frac{\partial h_{2j}}{\partial a_{2j}} \frac{\partial a_{2j}}{\partial W}
\]
\[ L(\theta) = -\sum_{j=1}^{n} (x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log (1 - \hat{x}_{ij})) \]

\[
\begin{align*}
\frac{\partial L(\theta)}{\partial h_2} &= \left( \begin{array}{c}
\frac{\partial L(\theta)}{\partial h_{21}} \\
\frac{\partial L(\theta)}{\partial h_{22}} \\
\vdots \\
\frac{\partial L(\theta)}{\partial h_{2n}}
\end{array} \right) \\
\frac{\partial h_{2j}}{\partial a_{2j}} &= \sigma(a_{2j})(1 - \sigma(a_{2j}))
\end{align*}
\]

- \[ \frac{\partial L(\theta)}{\partial W^*} = \frac{\partial L(\theta)}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial W^*} \]

- \[ \frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1} \frac{\partial a_1}{\partial W} \]

- We have already seen how to calculate the expressions in the square boxes when we learnt BP.

- The first two terms on RHS can be computed as:
  \[ \frac{\partial L(\theta)}{\partial h_{2j}} = -x_{ij} \hat{x}_{ij} + 1 - x_{ij} \]
  \[ \frac{\partial h_{2j}}{\partial a_{2j}} = \sigma(a_{2j})(1 - \sigma(a_{2j})) \]
Module 7.2: Link between PCA and Autoencoders
We will now see that the encoder part of an autoencoder is equivalent to PCA if we

\[
P^T X^T X P = D
\]
We will now see that the encoder part of an autoencoder is equivalent to PCA if we

- use a linear encoder
We will now see that the encoder part of an autoencoder is equivalent to PCA if we

- use a linear encoder
- use a linear decoder

\[ P^T X^T X P = D \]
We will now see that the encoder part of an autoencoder is equivalent to PCA if we
- use a linear encoder
- use a linear decoder
- use squared error loss function

\[
\hat{x}_i \quad \equiv \quad h \quad \equiv \quad u_1 \quad u_2
\]

\[
P^T X^T X P = D
\]
We will now see that the encoder part of an autoencoder is equivalent to PCA if we

- use a linear encoder
- use a linear decoder
- use squared error loss function
- normalize the inputs to

\[
\hat{x}_{ij} = \frac{1}{\sqrt{m}} \left( x_{ij} - \frac{1}{m} \sum_{k=1}^{m} x_{kj} \right)
\]
First let us consider the implication of normalizing the inputs to

\[ \hat{x}_{ij} = \frac{1}{\sqrt{m}} \left( x_{ij} - \frac{1}{m} \sum_{k=1}^{m} x_{kj} \right) \]
First let us consider the implication of normalizing the inputs to

$$\hat{x}_{ij} = \frac{1}{\sqrt{m}} \left( x_{ij} - \frac{1}{m} \sum_{k=1}^{m} x_{kj} \right)$$

The operation in the bracket ensures that the data now has 0 mean along each dimension $j$ (we are subtracting the mean)
First let us consider the implication of normalizing the inputs to 

$$\hat{x}_{ij} = \frac{1}{\sqrt{m}} \left( x_{ij} - \frac{1}{m} \sum_{k=1}^{m} x_{kj} \right)$$

The operation in the bracket ensures that the data now has 0 mean along each dimension $j$ (we are subtracting the mean)

Let $X'$ be this zero mean data matrix then what the above normalization gives us is $X = \frac{1}{\sqrt{m}} X'$
First let us consider the implication of normalizing the inputs to

\[ \hat{x}_{ij} = \frac{1}{\sqrt{m}} \left( x_{ij} - \frac{1}{m} \sum_{k=1}^{m} x_{kj} \right) \]

The operation in the bracket ensures that the data now has 0 mean along each dimension \( j \) (we are subtracting the mean).

Let \( X' \) be this zero mean data matrix then what the above normalization gives us is \( X = \frac{1}{\sqrt{m}} X' \).

Now \( (X)^T X = \frac{1}{m} (X')^T X' \) is the covariance matrix (recall that covariance matrix plays an important role in PCA).
First we will show that if we use linear decoder and a squared error loss function then the optimal solution to the following objective function is obtained when we use a linear encoder.

\[ \min \sum_{i=1}^{m} \sum_{j=1}^{n} (x_{ij} - \hat{x}_{ij})^2 \]

\[ P^T X^T X P = D \]
First we will show that if we use linear decoder and a squared error loss function then

\[ P^T X^T X P = D \]
First we will show that if we use linear decoder and a squared error loss function then

The optimal solution to the following objective function

\[
P^T X^T X P = D
\]
First we will show that if we use linear decoder and a squared error loss function then

The optimal solution to the following objective function

\[
\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (x_{ij} - \hat{x}_{ij})^2
\]
First we will show that if we use linear decoder and a squared error loss function then

The optimal solution to the following objective function

$$\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (x_{ij} - \hat{x}_{ij})^2$$

is obtained when we use a linear encoder.
\[
\min_\theta \sum_{i=1}^{m} \sum_{j=1}^{n} (x_{ij} - \hat{x}_{ij})^2
\]
\[
\min_\theta \sum_{i=1}^{m} \sum_{j=1}^{n} (x_{ij} - \hat{x}_{ij})^2
\]  

This is equivalent to

\[
\min_W W^* H \left( \|X - HW^*\|_F^2 \right) \|A\|_F^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2
\]  

(we are ignoring the biases)

From SVD we know that optimal solution to the above problem is given by

\[
HW^* = U_{,\leq k} \Sigma_{k,k} V^T_{,\leq k}
\]
\[
\min_{\theta} \sum_{i=1}^{m} \sum_{j=1}^{n} (x_{ij} - \hat{x}_{ij})^2
\]  

This is equivalent to

\[
\min_{W^*H} \left( \| X - HW^* \|_F \right)^2
\]  

From SVD we know that the optimal solution to the above problem is given by

\[
HW^* = U_{\leq k} \Sigma_{k,k} V^T_{\leq k}
\]  

By matching variables one possible solution is

\[
H = U_{\leq k} \Sigma_{k,k} W^* = V^T_{\leq k}
\]
\[
\min_{\theta} \sum_{i=1}^{m} \sum_{j=1}^{n} (x_{ij} - \hat{x}_{ij})^2
\]  

This is equivalent to

\[
\min_{W^*H} (\|X - HW^*\|_F)^2
\]

\[
\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2}
\]
\[
\min_{\theta} \sum_{i=1}^{m} \sum_{j=1}^{n} (x_{ij} - \hat{x}_{ij})^2
\] (1)

- This is equivalent to

\[
\min_{WH} (\|X - HW^*\|_F)^2 \quad \|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2}
\]

(just writing the expression (1) in matrix form and using the definition of \(\|A\|_F\) (we are ignoring the biases))
\[
\min_{\theta} \sum_{i=1}^{m} \sum_{j=1}^{n} (x_{ij} - \hat{x}_{ij})^2
\] (1)

- This is equivalent to

\[
\min_{W^H} (\| X - HW^* \|_F)^2 \quad \| A \|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2}
\]

(just writing the expression (1) in matrix form and using the definition of \( \| A \|_F \) (we are ignoring the biases)

- From SVD we know that optimal solution to the above problem is given by

\[
HW^* = U_{:, \leq k} \Sigma_{k,k} V_{:, \leq k}^T
\]
\[
\min_{\theta} \sum_{i=1}^{m} \sum_{j=1}^{n} (x_{ij} - \hat{x}_{ij})^2
\]  

This is equivalent to
\[
\min_{W^*H} (\|X - HW^*\|_F)^2 \quad \|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2}
\]

(just writing the expression (1) in matrix form and using the definition of \(\|A\|_F\) (we are ignoring the biases)

From SVD we know that optimal solution to the above problem is given by
\[
HW^* = U_{.,\leq k} \Sigma_{k,k} V_{.,\leq k}^T
\]

By matching variables one possible solution is
\[
H = U_{.,\leq k} \Sigma_{k,k}
\]
\[
W^* = V_{.,\leq k}^T
\]
We will now show that $H$ is a linear encoding and find an expression for the encoder weights $W$.
We will now show that $H$ is a linear encoding and find an expression for the encoder weights $W$

\[ H = U \cdot \leq_k \Sigma_{k,k} \]
We will now show that $H$ is a linear encoding and find an expression for the encoder weights $W$

$$H = U_{., \leq k} \Sigma_{k,k}$$

$$= (XX^T)(XX^T)^{-1}U_{., \leq K} \Sigma_{k,k}$$

(pre-multiplying $(XX^T)(XX^T)^{-1} = I$)
We will now show that $H$ is a linear encoding and find an expression for the encoder weights $W$

$$H = U_{., \leq k} \Sigma_{k,k}$$

$$= (XX^T)(XX^T)^{-1} U_{., \leq K} \Sigma_{k,k}$$

**(pre-multiplying $(XX^T)(XX^T)^{-1} = I$)**

$$= (X V \Sigma^T U^T)(U \Sigma V^T V \Sigma^T U^T)^{-1} U_{., \leq k} \Sigma_{k,k}$$

**(using $X = U \Sigma V^T$)**
We will now show that $H$ is a linear encoding and find an expression for the encoder weights $W$

\[
H = U_{:, \leq k} \Sigma_{k,k} \\
= (XX^T)(XX^T)^{-1}U_{:, \leq K} \Sigma_{k,k} \\
= (XV\Sigma^T U^T)(U\Sigma V^T V\Sigma^T U^T)^{-1}U_{:, \leq k} \Sigma_{k,k} \\
= XV\Sigma^T U^T (U\Sigma \Sigma^T U^T)^{-1}U_{:, \leq k} \Sigma_{k,k}
\]  

\hspace{1cm} \text{(pre-multiplying $(XX^T)(XX^T)^{-1} = I$)} \hspace{1cm} \text{(using $X = U\Sigma V^T$)} \hspace{1cm} \text{($V^T V = I$)}
We will now show that $H$ is a linear encoding and find an expression for the encoder weights $W$

$$H = U_{.,\leq k} \Sigma_{k,k}$$

$$= (XX^T)(XX^T)^{-1} U_{.,\leq k} \Sigma_{k,k}$$

(pre-multiplying $(XX^T)(XX^T)^{-1} = I$)

$$= (XV \Sigma^T U^T)(U \Sigma V^T V \Sigma^T U^T)^{-1} U_{.,\leq k} \Sigma_{k,k}$$

(using $X = U \Sigma V^T$)

$$= XV \Sigma^T U^T (U \Sigma \Sigma^T U^T)^{-1} U_{.,\leq k} \Sigma_{k,k}$$

$(V^T V = I)$

$$= XV \Sigma^T U^T U (\Sigma \Sigma^T)^{-1} U^T U_{.,\leq k} \Sigma_{k,k}$$

$((ABC)^{-1} = C^{-1} B^{-1} A^{-1})$
We will now show that $H$ is a linear encoding and find an expression for the encoder weights $W$

$$H = U_{., \leq k} \Sigma_{k,k}$$

$$= (XX^T)(XX^T)^{-1}U_{., \leq K} \Sigma_{k,k}$$

$$= (XV \Sigma^T U^T)(U \Sigma V^T V \Sigma^T U^T)^{-1}U_{., \leq k} \Sigma_{k,k}$$

$$= XV \Sigma^T U^T (U \Sigma \Sigma^T U^T)^{-1}U_{., \leq k} \Sigma_{k,k}$$

$$= XV \Sigma^T U^T (\Sigma \Sigma^T)^{-1}U^T U_{., \leq k} \Sigma_{k,k}$$

$$= XV \Sigma^T (\Sigma \Sigma^T)^{-1}U^T U_{., \leq k} \Sigma_{k,k}$$

(pre-multiplying $(XX^T)(XX^T)^{-1} = I$)

(using $X = U \Sigma V^T$)

$(V^T V = I)$

$((ABC)^{-1} = C^{-1} B^{-1} A^{-1})$

$(U^T U = I)$
We will now show that $H$ is a linear encoding and find an expression for the encoder weights $W$

$$H = U_{.., \leq k} \Sigma_{k,k}$$

$$= (XX^T)(XX^T)^{-1}U_{.., \leq K} \Sigma_{k,k} \quad (pre-multiplying (XX^T)(XX^T)^{-1} = I)$$

$$= (XV \Sigma^T U^T)(U \Sigma V^T V \Sigma^T U^T)^{-1}U_{.., \leq k} \Sigma_{k,k} \quad (using X = U \Sigma V^T)$$

$$= XV \Sigma^T U^T (U \Sigma \Sigma^T U^T)^{-1}U_{.., \leq k} \Sigma_{k,k} \quad (V^T V = I)$$

$$= XV \Sigma^T U^T U (\Sigma \Sigma^T)^{-1} U^T U_{.., \leq k} \Sigma_{k,k} \quad ((ABC)^{-1} = C^{-1}B^{-1}A^{-1})$$

$$= XV \Sigma^T (\Sigma \Sigma^T)^{-1} U^T U_{.., \leq k} \Sigma_{k,k} \quad (U^T U = I)$$

$$= XV \Sigma^T \Sigma^{-1} \Sigma^{-1} U^T U_{.., \leq k} \Sigma_{k,k} \quad ((AB)^{-1} = B^{-1}A^{-1})$$

$$= XV \Sigma^T \Sigma^{-1} U^T U_{.., \leq k} \Sigma_{k,k} \quad$$
We will now show that $H$ is a linear encoding and find an expression for the encoder weights $W$

\[ H = U . , \leq_k \Sigma_{k,k} \]
\[ = (XX^T)(XX^T)^{-1}U . , \leq_k \Sigma_{k,k} \]
\[ = (XV\Sigma^T U^T)(U\Sigma V^T V \Sigma^T U^T)^{-1}U . , \leq_k \Sigma_{k,k} \]
\[ = XV\Sigma^T U^T (U\Sigma \Sigma^T U^T)^{-1}U . , \leq_k \Sigma_{k,k} \]
\[ = XV\Sigma^T U^T (\Sigma \Sigma^T)^{-1}U^T U . , \leq_k \Sigma_{k,k} \]
\[ = XV\Sigma^T (\Sigma \Sigma^T)^{-1}U^T U . , \leq_k \Sigma_{k,k} \]
\[ = XV\Sigma^{-1} I . , \leq_k \Sigma_{k,k} \]

(pre-multiplying $(XX^T)(XX^T)^{-1} = I$)

(using $X = U\Sigma V^T$)

($V^TV = I$)

$((ABC)^{-1} = C^{-1}B^{-1}A^{-1})$

($U^TU = I$)

$((AB)^{-1} = B^{-1}A^{-1})$

($U^TU . , \leq_k = I . , \leq_k$)
We will now show that $H$ is a linear encoding and find an expression for the encoder weights $W$

$H = U_{., \leq k} \Sigma_{k,k}$

$= (XX^T)(XX^T)^{-1}U_{., \leq k} \Sigma_{k,k}$

$= (XV \Sigma^T U^T)(U \Sigma V^T V \Sigma^T U^T)^{-1}U_{., \leq k} \Sigma_{k,k}$

$= XV \Sigma^T U^T (U \Sigma \Sigma^T U^T)^{-1}U_{., \leq k} \Sigma_{k,k}$

$= XV \Sigma^T U^T U (\Sigma \Sigma^T)^{-1} U^T U_{., \leq k} \Sigma_{k,k}$

$= XV \Sigma^T (\Sigma \Sigma^T)^{-1} U^T U_{., \leq k} \Sigma_{k,k}$

$= XV \Sigma^T \Sigma^{-1} \Sigma^{-1} U^T U_{., \leq k} \Sigma_{k,k}$

$= XV \Sigma^{-1} I_{., \leq k} \Sigma_{k,k}$

$= XV I_{., \leq k}$

(pre-multiplying $(XX^T)(XX^T)^{-1} = I$)

(using $X = U \Sigma V^T$)  

$(V^T V = I)$

$((ABC)^{-1} = C^{-1} B^{-1} A^{-1})$

$((AB)^{-1} = B^{-1} A^{-1})$

$(U^T U = I)$

$(U^T U_{., \leq k} = I_{., \leq k})$

$(\Sigma^{-1} I_{., \leq k} = \Sigma_{k,k}^{-1})$
We will now show that $H$ is a linear encoding and find an expression for the encoder weights $W$

$$H = U_{.,\leq k} \Sigma_{k,k}$$

$$= (XX^T)(XX^T)^{-1}U_{.,\leq K} \Sigma_{k,k}$$

$$= (XV\Sigma^T U^T)(U\Sigma V^T V \Sigma^T U^T)^{-1}U_{.,\leq k} \Sigma_{k,k}$$

$$= XV \Sigma^T U^T (U \Sigma \Sigma^T U^T)^{-1}U_{.,\leq k} \Sigma_{k,k}$$

$$= XV \Sigma^T U^T U (\Sigma \Sigma^T)^{-1}U^T U_{.,\leq k} \Sigma_{k,k}$$

$$= XV \Sigma^T (\Sigma \Sigma^T)^{-1}U^T U_{.,\leq k} \Sigma_{k,k}$$

$$= XV \Sigma^T T^{-1} \Sigma^{-1} U^T U_{.,\leq k} \Sigma_{k,k}$$

$$= XV \Sigma^{-1} I_{.,\leq k} \Sigma_{k,k}$$

$$= XV I_{.,\leq k}$$

$$H = XV_{.,\leq k}$$

(pre-multiplying $(XX^T)(XX^T)^{-1} = I$)

(using $X = U \Sigma V^T$)

($V^T V = I$)

$((ABC)^{-1} = C^{-1} B^{-1} A^{-1})$

($U^T U = I$)

$((AB)^{-1} = B^{-1} A^{-1})$

($U^T U_{.,\leq k} = I_{.,\leq k}$)

($\Sigma^{-1} I_{.,\leq k} = \Sigma_{k,k}^{-1}$)
We will now show that $H$ is a linear encoding and find an expression for the encoder weights $W$

$$H = U.\leq_k \Sigma_{k,k}$$

$$= (XX^T)(XX^T)^{-1}U.\leq_k \Sigma_{k,k}$$

(pre-multiplying $(XX^T)(XX^T)^{-1} = I$)

$$= (XV \Sigma^T U^T)(U \Sigma V^T V \Sigma^T U^T)^{-1}U.\leq_k \Sigma_{k,k}$$

(using $X = U \Sigma V^T$)

$$= XV \Sigma^T U^T (U \Sigma \Sigma^T U^T)^{-1}U.\leq_k \Sigma_{k,k}$$

($V^T V = I$)

$$= XV \Sigma^T U^T U (\Sigma \Sigma^T)^{-1}U^T U.\leq_k \Sigma_{k,k}$$

($((ABC)^{-1} = C^{-1} B^{-1} A^{-1})$)

$$= XV \Sigma^T (\Sigma \Sigma^T)^{-1} U^T U.\leq_k \Sigma_{k,k}$$

($U^T U = I$)

$$= XV \Sigma^T \Sigma^{-1} U^T U.\leq_k \Sigma_{k,k}$$

($((AB)^{-1} = B^{-1} A^{-1})$)

$$= XV \Sigma^{-1} U.\leq_k \Sigma_{k,k}$$

($U^T U.\leq_k = I.\leq_k$)

$$= XV \Sigma^{-1} U.\leq_k$$

($\Sigma^{-1} U.\leq_k = \Sigma^{-1}_{k,k}$)

$$H = XV.\leq_k$$

Thus $H$ is a linear transformation of $X$ and $W = V.\leq_k$
• We have encoder $W = V_{:, \leq k}$
- We have encoder $W = V_{:,\leq k}$
- From SVD, we know that $V$ is the matrix of eigen vectors of $X^TX$
- We have encoder $W = V_{., \leq k}$
- From SVD, we know that $V$ is the matrix of eigen vectors of $X^T X$
- From PCA, we know that $P$ is the matrix of the eigen vectors of the covariance matrix
- We have encoder $W = V_{\cdot,\leq k}$
- From SVD, we know that $V$ is the matrix of eigen vectors of $X^T X$
- From PCA, we know that $P$ is the matrix of the eigen vectors of the covariance matrix
- We saw earlier that, if entries of $X$ are normalized by
We have encoder $W = V_{:, \leq k}$

From SVD, we know that $V$ is the matrix of eigen vectors of $X^T X$

From PCA, we know that $P$ is the matrix of the eigen vectors of the covariance matrix

We saw earlier that, if entries of $X$ are normalized by

$$\hat{x}_{ij} = \frac{1}{\sqrt{m}} \left( x_{ij} - \frac{1}{m} \sum_{k=1}^{m} x_{kj} \right)$$
We have encoder \( W = V_{:, \leq k} \).

From SVD, we know that \( V \) is the matrix of eigen vectors of \( X^T X \).

From PCA, we know that \( P \) is the matrix of the eigen vectors of the covariance matrix.

We saw earlier that, if entries of \( X \) are normalized by

\[
\hat{x}_{ij} = \frac{1}{\sqrt{m}} \left( x_{ij} - \frac{1}{m} \sum_{k=1}^{m} x_{kj} \right)
\]

then \( X^T X \) is indeed the covariance matrix.
- We have encoder $W = V_{., \leq k}$
- From SVD, we know that $V$ is the matrix of eigen vectors of $X^T X$
- From PCA, we know that $P$ is the matrix of the eigen vectors of the covariance matrix
- We saw earlier that, if entries of $X$ are normalized by
  \[
  \hat{x}_{i j} = \frac{1}{\sqrt{m}} \left( x_{i j} - \frac{1}{m} \sum_{k=1}^{m} x_{k j} \right)
  \]
  then $X^T X$ is indeed the covariance matrix
- Thus, the encoder matrix for linear autoencoder($W$) and the projection matrix($P$) for PCA could indeed be the same. Hence proved
Remember
The encoder of a linear autoencoder is equivalent to PCA if we
Remember

The encoder of a linear autoencoder is equivalent to PCA if we
- use a linear encoder
Remember

The encoder of a linear autoencoder is equivalent to PCA if we

- use a linear encoder
- use a linear decoder
Remember

The encoder of a linear autoencoder is equivalent to PCA if we

- use a linear encoder
- use a linear decoder
- use a squared error loss function

\[
\hat{x}_{ij} = \frac{1}{\sqrt{m}} \left( x_{ij} - \frac{1}{m} \sum_{k=1}^{m} x_{kj} \right)
\]
Remember

The encoder of a linear autoencoder is equivalent to PCA if we
- use a linear encoder
- use a linear decoder
- use a squared error loss function
- and normalize the inputs to

\[ \hat{x}_{ij} = \frac{1}{\sqrt{m}} \left( x_{ij} - \frac{1}{m} \sum_{k=1}^{m} x_{kj} \right) \]
Remember

The encoder of a linear autoencoder is equivalent to PCA if we

- use a linear encoder
- use a linear decoder
- use a squared error loss function
- and normalize the inputs to

\[
\hat{x}_{ij} = \frac{1}{\sqrt{m}} \left( x_{ij} - \frac{1}{m} \sum_{k=1}^{m} x_{kj} \right)
\]
Module 7.3: Regularization in autoencoders (Motivation)
While poor generalization could happen even in undercomplete autoencoders it is an even more serious problem for overcomplete autoencoders. Here, (as stated earlier) the model can simply learn to copy $x_i$ to $h$ and then $h$ to $\hat{x}_i$.

To avoid poor generalization, we need to introduce regularization.
While poor generalization could happen even in undercomplete autoencoders it is an even more serious problem for overcomplete autoencoders.
While poor generalization could happen even in undercomplete autoencoders it is an even more serious problem for overcomplete autoencoders. Here, (as stated earlier) the model can simply learn to copy $x_i$ to $h$ and then $h$ to $\hat{x}_i$.
While poor generalization could happen even in undercomplete autoencoders it is an even more serious problem for overcomplete auto encoders.

Here, (as stated earlier) the model can simply learn to copy $x_i$ to $h$ and then $h$ to $\hat{x}_i$.

To avoid poor generalization, we need to introduce regularization.
The simplest solution is to add a $L_2$-regularization term to the objective function

$$
\min_{\theta, w, w^*, b, c} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2 + \lambda \|\theta\|^2
$$
The simplest solution is to add a $L_2$-regularization term to the objective function

$$\min_{\theta,w,w^*,b,c} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2 + \lambda \| \theta \|^2$$

This is very easy to implement and just adds a term $\lambda W$ to the gradient $\frac{\partial \mathcal{L}(\theta)}{\partial W}$ (and similarly for other parameters)
Another trick is to tie the weights of the encoder and decoder.
Another trick is to tie the weights of the encoder and decoder i.e., $W^* = W^T$
- Another trick is to tie the weights of the encoder and decoder i.e., \( W^* = W^T \)
- This effectively reduces the capacity of Autoencoder and acts as a regularizer
Module 7.4: Denoising Autoencoders
A denoising encoder simply corrupts the input data using a probabilistic process \( P(\tilde{x}_{ij} | x_{ij}) \) before feeding it to the network.
A denoising encoder simply corrupts the input data using a probabilistic process \( P(\tilde{x}_{ij} | x_{ij}) \) before feeding it to the network.

A simple \( P(\tilde{x}_{ij} | x_{ij}) \) used in practice is the following:

\[
P(\tilde{x}_{ij} = 0 | x_{ij}) = q \quad \text{and} \quad P(\tilde{x}_{ij} = x_{ij} | x_{ij}) = 1 - q
\]
A denoising encoder simply corrupts the input data using a probabilistic process \( P(\tilde{x}_{ij} | x_{ij}) \) before feeding it to the network.

A simple \( P(\tilde{x}_{ij} | x_{ij}) \) used in practice is the following:

\[
P(\tilde{x}_{ij} = 0 | x_{ij}) = q
\]
A denoising encoder simply corrupts the input data using a probabilistic process \( P(\tilde{x}_{ij} | x_{ij}) \) before feeding it to the network.

A simple \( P(\tilde{x}_{ij} | x_{ij}) \) used in practice is the following:

\[
\begin{align*}
P(\tilde{x}_{ij} = 0 | x_{ij}) &= q \\
P(\tilde{x}_{ij} = x_{ij} | x_{ij}) &= 1 - q
\end{align*}
\]
A denoising encoder simply corrupts the input data using a probabilistic process \( P(\tilde{x}_{ij}|x_{ij}) \) before feeding it to the network.

A simple \( P(\tilde{x}_{ij}|x_{ij}) \) used in practice is the following:

\[
\begin{align*}
P(\tilde{x}_{ij} = 0| x_{ij}) &= q \\
P(\tilde{x}_{ij} = x_{ij}| x_{ij}) &= 1 - q
\end{align*}
\]

In other words, with probability \( q \) the input is flipped to 0 and with probability \((1 - q)\) it is retained as it is.
For example, it will have to learn to reconstruct a corrupted \( x_{ij} \) correctly by relying on its interactions with other elements of \( x_i \).

How does this help?

This helps because the objective is still to reconstruct the original (un-corrupted) \( x_i \):

\[
\arg \min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2
\]

It no longer makes sense for the model to copy the corrupted \( \tilde{x}_i \) into \( h(\tilde{x}_i) \) and then into \( \hat{x}_i \) (the objective function will not be minimized by doing so).

Instead the model will now have to capture the characteristics of the data correctly.
For example, it will have to learn to reconstruct a corrupted $x_{ij}$ correctly by relying on its interactions with other elements of $x_i$.

How does this help?

This helps because the objective is still to reconstruct the original (uncorrupted) $x_i$:

$$\arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2$$

It no longer makes sense for the model to copy the corrupted $\tilde{x}_i$ into $h(\tilde{x}_i)$ and then into $\hat{x}_i$ (the objective function will not be minimized by doing so).

Instead the model will now have to capture the characteristics of the data correctly.
How does this help?
This helps because the objective is still to reconstruct the original (uncorrupted) $x_i$

$$\arg \min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2$$

It no longer makes sense for the model to copy the corrupted $\tilde{x}_i$ into $h(\tilde{x}_i)$ and then into $\hat{x}_i$ (the objective function will not be minimized by doing so)
For example, it will have to learn to reconstruct a corrupted \( x_{ij} \) correctly by relying on its interactions with other elements of \( x_i \). How does this help?

This helps because the objective is still to reconstruct the original (uncorrupted) \( x_i \)

\[
\arg \min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2
\]

It no longer makes sense for the model to copy the corrupted \( \tilde{x}_i \) into \( h(\tilde{x}_i) \) and then into \( \hat{x}_i \) (the objective function will not be minimized by doing so)

Instead the model will now have to capture the characteristics of the data correctly.
For example, it will have to learn to reconstruct a corrupted $x_{ij}$ correctly by relying on its interactions with other elements of $x_i$.

- How does this help?
- This helps because the objective is still to reconstruct the original (uncorrupted) $x_i$

$$\arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2$$

- It no longer makes sense for the model to copy the corrupted $\tilde{x}_i$ into $h(\tilde{x}_i)$ and then into $\hat{x}_i$ (the objective function will not be minimized by doing so)
- Instead the model will now have to capture the characteristics of the data correctly.
We will now see a practical application in which AEs are used and then compare Denoising Autoencoders with regular autoencoders
Task: Hand-written digit recognition

Figure: MNIST Data

\[
|x_i| = 784 = 28 \times 28
\]

Figure: Basic approach (we use raw data as input features)
Task: Hand-written digit recognition

Figure: MNIST Data

Figure: AE approach (first learn important characteristics of data)
Task: Hand-written digit recognition

Figure: MNIST Data

\[ |x_i| = 784 = 28 \times 28 \]

Figure: AE approach (and then train a classifier on top of this hidden representation)
We will now see a way of visualizing AEs and use this visualization to compare different AEs
We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration $x_i$.
- We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration \( x_i \).
- For example,
  
  \[
  h_1 = \sigma(W_1^T x_i) \quad [ignoring \ bias \ b]
  \]

  Where \( W_1 \) is the trained vector of weights connecting the input to the first hidden neuron.
We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration \( x_i \).

For example,

\[
\mathbf{h}_1 = \sigma(\mathbf{W}_1^T \mathbf{x}_i) \ [\text{ignoring bias } b]
\]

Where \( \mathbf{W}_1 \) is the trained vector of weights connecting the input to the first hidden neuron.

What values of \( x_i \) will cause \( \mathbf{h}_1 \) to be maximum (or maximally activated)?
We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration $x_i$.

For example,

$$h_1 = \sigma(W_1^T x_i) \ [ignoring \ bias \ b]$$

Where $W_1$ is the trained vector of weights connecting the input to the first hidden neuron.

What values of $x_i$ will cause $h_1$ to be maximum (or maximally activated)?

Suppose we assume that our inputs are normalized so that $\|x_i\| = 1$. 

$h_1$, where $h_1$ is the output of the first hidden layer, is a function of the input $x_i$ and the weights $W_1$. The activation function $\sigma$ is typically a sigmoid function, which maps any real-valued number to a value between 0 and 1. This allows the neuron to be maximally activated for certain input configurations.
We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration \( \mathbf{x}_i \)

For example,

\[
\mathbf{h}_1 = \sigma(W_1^T \mathbf{x}_i) \quad [\text{ignoring bias } b]
\]

Where \( W_1 \) is the trained vector of weights connecting the input to the first hidden neuron.

What values of \( \mathbf{x}_i \) will cause \( \mathbf{h}_1 \) to be maximum (or maximally activated)?

Suppose we assume that our inputs are normalized so that \( \| \mathbf{x}_i \|^2 = \mathbf{x}_i^T \mathbf{x}_i = 1 \)
We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration $\mathbf{x}_i$.

For example,

$$\mathbf{h}_1 = \sigma(\mathbf{W}_1^T \mathbf{x}_i) \ [ignoring \ bias \ b]$$

Where $\mathbf{W}_1$ is the trained vector of weights connecting the input to the first hidden neuron.

What values of $\mathbf{x}_i$ will cause $\mathbf{h}_1$ to be maximum (or maximally activated)?

Suppose we assume that our inputs are normalized so that $||\mathbf{x}_i|| = 1$.

$$\begin{align*}
\max_{\mathbf{x}_i} \ & \{ \mathbf{W}_1^T \mathbf{x}_i \} \\
\text{s.t.} \ & ||\mathbf{x}_i||^2 = \mathbf{x}_i^T \mathbf{x}_i = 1
\end{align*}$$

Solution: $\mathbf{x}_i = \frac{\mathbf{W}_1}{\sqrt{\mathbf{W}_1^T \mathbf{W}_1}}$
Thus the inputs

\[ x_i = \frac{W_1}{\sqrt{W_1^T W_1}}, \frac{W_2}{\sqrt{W_2^T W_2}}, \ldots, \frac{W_n}{\sqrt{W_n^T W_n}} \]

will respectively cause hidden neurons 1 to n to maximally fire.

Let us plot these images (\( \hat{x}_i \)'s) which maximally activate the first \( k \) neurons of the hidden representations learned by a vanilla autoencoder and different denoising autoencoders. These \( \hat{x}_i \)'s are computed by the above formula using the weights (\( W_1, W_2, \ldots, W_k \)) learned by the respective autoencoders.

\[
\max_{x_i} \{ W_1^T x_i \} \\
\text{s.t.} \quad ||x_i||^2 = x_i^T x_i = 1 \\
\text{Solution:} \quad x_i = \frac{W_1}{\sqrt{W_1^T W_1}}
\]
Thus the inputs

\[
x_i = \frac{W_1}{\sqrt{W_1^T W_1}}, \frac{W_2}{\sqrt{W_2^T W_2}}, \ldots \frac{W_n}{\sqrt{W_n^T W_n}}
\]

will respectively cause hidden neurons 1 to n to maximally fire.

Let us plot these images (\(x_i\)'s) which maximally activate the first \(k\) neurons of the hidden representations learned by a vanilla autoencoder and different denoising autoencoders.
Thus the inputs

\[ x_i = \frac{W_1}{\sqrt{W_1^T W_1}}, \frac{W_2}{\sqrt{W_2^T W_2}}, \ldots, \frac{W_n}{\sqrt{W_n^T W_n}} \]

will respectively cause hidden neurons 1 to \( n \) to maximally fire

- Let us plot these images (\( x_i \)'s) which maximally activate the first \( k \) neurons of the hidden representations learned by a vanilla autoencoder and different denoising autoencoders.

- These \( x_i \)'s are computed by the above formula using the weights \( (W_1, W_2 \ldots W_k) \) learned by the respective autoencoders.
*The vanilla AE does not learn many meaningful patterns*
The vanilla AE does not learn many meaningful patterns.

The hidden neurons of the denoising AEs seem to act like pen-stroke detectors (for example, in the highlighted neuron the black region is a stroke that you would expect in a ‘0’ or a ‘2’ or a ‘3’ or a ‘8’ or a ‘9’).
The vanilla AE does not learn many meaningful patterns.

The hidden neurons of the denoising AEs seem to act like pen-stroke detectors (for example, in the highlighted neuron the black region is a stroke that you would expect in a '0' or a '2' or a '3' or a '8' or a '9').

As the noise increases the filters become more wide because the neuron has to rely on more adjacent pixels to feel confident about a stroke.
We saw one form of $P(\tilde{x}_{ij}|x_{ij})$ which flips a fraction $q$ of the inputs to zero.

Another way of corrupting the inputs is to add a Gaussian noise to the input:

$$\tilde{x}_{ij} = x_{ij} + N(0,1)$$

We will now use such a denoising AE on a different dataset and see their performance.
- We saw one form of $P(\tilde{x}_{ij}|x_{ij})$ which flips a fraction $q$ of the inputs to zero.
- Another way of corrupting the inputs is to add a Gaussian noise to the input:

  $$\tilde{x}_{ij} = x_{ij} + \mathcal{N}(0, 1)$$
We saw one form of $P(\tilde{x}_{ij}|x_{ij})$ which flips a fraction $q$ of the inputs to zero.

Another way of corrupting the inputs is to add a Gaussian noise to the input:

$$\tilde{x}_{ij} = x_{ij} + \mathcal{N}(0, 1)$$

We will now use such a denoising AE on a different dataset and see their performance.
The hidden neurons essentially behave like edge detectors.
The hidden neurons essentially behave like edge detectors
PCA does not give such edge detectors
Module 7.5: Sparse Autoencoders
A hidden neuron with sigmoid activation will have values between 0 and 1. We say that the neuron is activated when its output is close to 1 and not activated when its output is close to 0. A sparse autoencoder tries to ensure the neuron is inactive most of the times.
A hidden neuron with sigmoid activation will have values between 0 and 1.
A hidden neuron with sigmoid activation will have values between 0 and 1. We say that the neuron is activated when its output is close to 1 and not activated when its output is close to 0.
A hidden neuron with sigmoid activation will have values between 0 and 1.

We say that the neuron is activated when its output is close to 1 and not activated when its output is close to 0.

A sparse autoencoder tries to ensure the neuron is inactive most of the times.
The average value of the activation of a neuron $l$ is given by

$$\hat{\rho}_l = \frac{1}{m} \sum_{i=1}^{m} h(x_i)_l$$

- If the neuron $l$ is sparse (i.e. mostly inactive) then $\hat{\rho}_l \to 0$

A sparse autoencoder uses a sparsity parameter $\rho$ (typically very close to 0, say, 0.005) and tries to enforce the constraint $\hat{\rho}_l = \rho$. One way of ensuring this is to add the following term to the objective function

$$\Omega(\theta) = \sum_{l=1}^{k} \rho \log \hat{\rho}_l + (1 - \rho) \log 1 - \hat{\rho}_l$$

When will this term reach its minimum value and what is the minimum value? Let us plot it and check.
The average value of the activation of a neuron $l$ is given by

$$\hat{\rho}_l = \frac{1}{m} \sum_{i=1}^{m} h(x_i)_l$$

- If the neuron $l$ is sparse (i.e. mostly inactive) then $\hat{\rho}_l \to 0$
- A sparse autoencoder uses a sparsity parameter $\rho$ (typically very close to 0, say, 0.005) and tries to enforce the constraint $\hat{\rho}_l = \rho$

When will this term reach its minimum value and what is the minimum value? Let us plot it and check.
The average value of the activation of a neuron $l$ is given by

$$\hat{\rho}_l = \frac{1}{m} \sum_{i=1}^{m} h(x_i)_l$$

- If the neuron $l$ is sparse (i.e. mostly inactive) then $\hat{\rho}_l \rightarrow 0$
- A sparse autoencoder uses a sparsity parameter $\rho$ (typically very close to 0, say, 0.005) and tries to enforce the constraint $\hat{\rho}_l = \rho$
- One way of ensuring this is to add the following term to the objective function

$$\Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\hat{\rho}_l} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_l}$$
The average value of the activation of a neuron $l$ is given by

$$\hat{\rho}_l = \frac{1}{m} \sum_{i=1}^{m} h(x_i)_l$$

- If the neuron $l$ is sparse (i.e. mostly inactive) then $\hat{\rho}_l \to 0$
- A sparse autoencoder uses a sparsity parameter $\rho$ (typically very close to 0, say, 0.005) and tries to enforce the constraint $\hat{\rho}_l = \rho$
- One way of ensuring this is to add the following term to the objective function

$$\Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\hat{\rho}_l} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_l}$$

- When will this term reach its minimum value and what is the minimum value? Let us plot it and check.
The function will reach its minimum value(s) when $\hat{\rho}_l = \rho$.
The function will reach its minimum value(s) when \( \hat{\rho}_l = \rho \).
Now,

\[ \hat{\mathcal{L}}(\theta) = \mathcal{L}(\theta) + \Omega(\theta) \]
Now,

\[ \hat{L}(\theta) = L(\theta) + \Omega(\theta) \]

\( \mathcal{L}(\theta) \) is the squared error loss or cross entropy loss and \( \Omega(\theta) \) is the sparsity constraint.
Now,
\[
\hat{\mathcal{L}}(\theta) = \mathcal{L}(\theta) + \Omega(\theta)
\]

\(\mathcal{L}(\theta)\) is the squared error loss or cross entropy loss and \(\Omega(\theta)\) is the sparsity constraint.

We already know how to calculate \(\frac{\partial \mathcal{L}(\theta)}{\partial W}\)
Now,

$$\hat{\mathcal{L}}(\theta) = \mathcal{L}(\theta) + \Omega(\theta)$$

$$\mathcal{L}(\theta)$$ is the squared error loss or cross entropy loss and \(\Omega(\theta)\) is the sparsity constraint.

We already know how to calculate \(\frac{\partial \mathcal{L}(\theta)}{\partial W}\).

Let us see how to calculate \(\frac{\partial \Omega(\theta)}{\partial W}\).
$\Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\hat{\rho}_l} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_l}$

- Now,
  \[ \mathcal{L}(\theta) = \mathcal{L}(\theta) + \Omega(\theta) \]

- $\mathcal{L}(\theta)$ is the squared error loss or cross entropy loss and $\Omega(\theta)$ is the sparsity constraint.

- We already know how to calculate $\frac{\partial \mathcal{L}(\theta)}{\partial W}$

- Let us see how to calculate $\frac{\partial \Omega(\theta)}{\partial W}$.
\[ \Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\hat{\rho}_l} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_l} \]

Can be re-written as

\[ \Omega(\theta) = \sum_{l=1}^{k} \rho \log \rho - \rho \log \hat{\rho}_l + (1 - \rho) \log (1 - \rho) - (1 - \rho) \log (1 - \hat{\rho}_l) \]

- Now,
  \[ \hat{\mathcal{L}}(\theta) = \mathcal{L}(\theta) + \Omega(\theta) \]

- \( \mathcal{L}(\theta) \) is the squared error loss or cross entropy loss and \( \Omega(\theta) \) is the sparsity constraint.

- We already know how to calculate \( \frac{\partial \mathcal{L}(\theta)}{\partial W} \)

- Let us see how to calculate \( \frac{\partial \Omega(\theta)}{\partial W} \).
\[ \Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\hat{\rho}_l} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_l} \]

Can be re-written as

\[ \Omega(\theta) = \sum_{l=1}^{k} \rho \log \rho - \rho \log \hat{\rho}_l + (1 - \rho) \log (1 - \rho) - (1 - \rho) \log (1 - \hat{\rho}_l) \]

By Chain rule:

\[ \frac{\partial \Omega(\theta)}{\partial W} = \frac{\partial \Omega(\theta)}{\partial \hat{\rho}} \cdot \frac{\partial \hat{\rho}}{\partial W} \]

- Now,
  \[ \hat{\mathcal{L}}(\theta) = \mathcal{L}(\theta) + \Omega(\theta) \]

- \( \mathcal{L}(\theta) \) is the squared error loss or cross entropy loss and \( \Omega(\theta) \) is the sparsity constraint.

- We already know how to calculate \( \frac{\partial \mathcal{L}(\theta)}{\partial W} \).

- Let us see how to calculate \( \frac{\partial \Omega(\theta)}{\partial W} \).
\[ \Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\hat{\rho}_l} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_l} \]

Can be re-written as

\[ \Omega(\theta) = \sum_{l=1}^{k} \rho \log \rho - \rho \log \hat{\rho}_l + (1 - \rho) \log (1 - \rho) - (1 - \rho) \log (1 - \hat{\rho}_l) \]

By Chain rule:

\[ \frac{\partial \Omega(\theta)}{\partial W} = \frac{\partial \Omega(\theta)}{\partial \hat{\rho}} \cdot \frac{\partial \hat{\rho}}{\partial W} \]

\[ \frac{\partial \Omega(\theta)}{\partial \hat{\rho}} = \left[ \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_1}, \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_2}, \ldots, \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_k} \right]^T \]

- Now,
  \[ \hat{\mathcal{L}}(\theta) = \mathcal{L}(\theta) + \Omega(\theta) \]

- \( \mathcal{L}(\theta) \) is the squared error loss or cross entropy loss and \( \Omega(\theta) \) is the sparsity constraint.

- We already know how to calculate \( \frac{\partial \mathcal{L}(\theta)}{\partial W} \).

- Let us see how to calculate \( \frac{\partial \Omega(\theta)}{\partial W} \).
\[ \Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\hat{\rho}_l} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_l} \]

Can be re-written as

\[ \Omega(\theta) = \sum_{l=1}^{k} \rho \log \rho - \rho \log \hat{\rho}_l + (1 - \rho) \log (1 - \rho) - (1 - \rho) \log (1 - \hat{\rho}_l) \]

By Chain rule:

\[ \frac{\partial \Omega(\theta)}{\partial W} = \frac{\partial \Omega(\theta)}{\partial \hat{\rho}} \cdot \frac{\partial \hat{\rho}}{\partial W} \]

\[ \frac{\partial \Omega(\theta)}{\partial \hat{\rho}} = \left[ \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_1}, \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_2}, \ldots, \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_k} \right]^T \]

For each neuron \( l \in 1 \ldots k \) in hidden layer, we have

- Now,
  \[ \hat{\mathcal{L}}(\theta) = \mathcal{L}(\theta) + \Omega(\theta) \]

- \( \mathcal{L}(\theta) \) is the squared error loss or cross entropy loss and \( \Omega(\theta) \) is the sparsity constraint.

- We already know how to calculate \( \frac{\partial \mathcal{L}(\theta)}{\partial W} \).

- Let us see how to calculate \( \frac{\partial \Omega(\theta)}{\partial W} \).
\[ \Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\hat{\rho}_l} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_l} \]

Can be re-written as

\[ \Omega(\theta) = \sum_{l=1}^{k} \rho \log \rho - \rho \log \hat{\rho}_l + (1 - \rho) \log (1 - \rho) - (1 - \rho) \log (1 - \hat{\rho}_l) \]

By Chain rule:

\[ \frac{\partial \Omega(\theta)}{\partial W} = \frac{\partial \Omega(\theta)}{\partial \hat{\rho}} \cdot \frac{\partial \hat{\rho}}{\partial W} \]

\[ \frac{\partial \Omega(\theta)}{\partial \hat{\rho}} = \left[ \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_1}, \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_2}, \ldots, \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_k} \right]^T \]

For each neuron \( l \in 1 \ldots k \) in hidden layer, we have

\[ \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_l} = -\frac{\rho}{\hat{\rho}_l} + \frac{(1 - \rho)}{1 - \hat{\rho}_l} \]

- Now,
  \[ \hat{\mathcal{L}}(\theta) = \mathcal{L}(\theta) + \Omega(\theta) \]

- \( \mathcal{L}(\theta) \) is the squared error loss or cross entropy loss and \( \Omega(\theta) \) is the sparsity constraint.

- We already know how to calculate \( \frac{\partial \mathcal{L}(\theta)}{\partial W} \).

- Let us see how to calculate \( \frac{\partial \Omega(\theta)}{\partial W} \).
\[ \Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\hat{\rho}_l} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_l} \]

Can be re-written as
\[ \Omega(\theta) = \sum_{l=1}^{k} \rho \log \rho - \rho \log \hat{\rho}_i + (1 - \rho) \log (1 - \rho) - (1 - \rho) \log (1 - \hat{\rho}_i) \]

By Chain rule:
\[ \frac{\partial \Omega(\theta)}{\partial W} = \frac{\partial \Omega(\theta)}{\partial \hat{\rho}} \cdot \frac{\partial \hat{\rho}}{\partial W} \]
\[ \frac{\partial \Omega(\theta)}{\partial \hat{\rho}} = \left[ \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_1}, \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_2}, \ldots, \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_k} \right]^T \]

For each neuron \( l \in 1 \ldots k \) in hidden layer, we have
\[ \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_l} = -\frac{\rho}{\hat{\rho}_l} + \frac{(1 - \rho)}{1 - \hat{\rho}_l} \]

and \( \frac{\partial \hat{\rho}_l}{\partial W} = x_i (g'(W^T x_i + b))^T \) (see next slide)

- Now,
  \[ \hat{L}(\theta) = L(\theta) + \Omega(\theta) \]

- \( L(\theta) \) is the squared error loss or cross entropy loss and \( \Omega(\theta) \) is the sparsity constraint.

- We already know how to calculate \( \frac{\partial L(\theta)}{\partial W} \)

- Let us see how to calculate \( \frac{\partial \Omega(\theta)}{\partial W} \).
\[ \Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\hat{\rho}_l} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_l} \]

Can be re-written as

\[ \Omega(\theta) = \sum_{l=1}^{k} \rho \log \rho - \rho \log \hat{\rho}_l + (1 - \rho) \log (1 - \rho) - (1 - \rho) \log (1 - \hat{\rho}_l) \]

By Chain rule:

\[ \frac{\partial \Omega(\theta)}{\partial W} = \frac{\partial \Omega(\theta)}{\partial \hat{\rho}} \cdot \frac{\partial \hat{\rho}}{\partial W} \]

\[ \frac{\partial \Omega(\theta)}{\partial \hat{\rho}} = \left[ \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_1}, \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_2}, \ldots, \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_k} \right]^T \]

For each neuron \( l \in 1 \ldots k \) in hidden layer, we have

\[ \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_l} = -\rho \frac{1}{\hat{\rho}_l} + \frac{(1 - \rho)}{1 - \hat{\rho}_l} \]

and

\[ \frac{\partial \hat{\rho}_l}{\partial W} = x_i (g'(W^T x_i + b))^T \]

Now,

\[ \hat{L}(\theta) = L(\theta) + \Omega(\theta) \]

\( L(\theta) \) is the squared error loss or cross entropy loss and \( \Omega(\theta) \) is the sparsity constraint.

We already know how to calculate \( \frac{\partial L(\theta)}{\partial W} \).

Let us see how to calculate \( \frac{\partial \Omega(\theta)}{\partial W} \).

Finally,

\[ \frac{\partial \hat{L}(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial W} + \frac{\partial \Omega(\theta)}{\partial W} \]

(and we know how to calculate both terms on R.H.S)
Derivation

\[
\frac{\partial \hat{\rho}}{\partial W} = \left[ \frac{\partial \hat{\rho}_1}{\partial W} \quad \frac{\partial \hat{\rho}_2}{\partial W} \quad \cdots \quad \frac{\partial \hat{\rho}_k}{\partial W} \right]
\]

For each element in the above equation we can calculate \(\frac{\partial \hat{\rho}_l}{\partial W}\) (which is the partial derivative of a scalar w.r.t. a matrix = matrix). For a single element of a matrix \(W_{jl}\):

\[
\frac{\partial \hat{\rho}_l}{\partial W_{jl}} = \frac{\partial}{\partial W_{jl}} \left[ \frac{1}{m} \sum_{i=1}^{m} g(W_{:,l}^T x_i + b_l) \right]
\]

\[
= \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W_{jl}} \left[ g(W_{:,l}^T x_i + b_l) \right]
\]

\[
= \frac{1}{m} \sum_{i=1}^{m} g'(W_{:,l}^T x_i + b_l) x_{ij}
\]

So in matrix notation we can write it as :

\[
\frac{\partial \hat{\rho}_l}{\partial W} = x_i (g'(W^T x_i + b))^T
\]
Module 7.6: Contractive Autoencoders
A contractive autoencoder also tries to prevent an overcomplete autoencoder from learning the identity function.

\[ \Omega(\theta) = \| J_x(h) \|_2 \]

where \( J_x(h) \) is the Jacobian of the encoder.
A contractive autoencoder also tries to prevent an overcomplete autoencoder from learning the identity function.

It does so by adding the following regularization term to the loss function

\[ \Omega(\theta) = \|J_x(h)\|^2_F \]
A contractive autoencoder also tries to prevent an overcomplete autoencoder from learning the identity function.

It does so by adding the following regularization term to the loss function

$$\Omega(\theta) = \| J_x(h) \|_F^2$$

where $J_x(h)$ is the Jacobian of the encoder.
A contractive autoencoder also tries to prevent an overcomplete autoencoder from learning the identity function. It does so by adding the following regularization term to the loss function

$$\Omega(\theta) = \| J_x(h) \|_F^2$$

where $J_x(h)$ is the Jacobian of the encoder.

Let us see what it looks like.
If the input has $n$ dimensions and the hidden layer has $k$ dimensions then

In other words, the $(l,j)$ entry of the Jacobian captures the variation in the output of the $l$th neuron with a small variation in the $j$th input.
If the input has \( n \) dimensions and the hidden layer has \( k \) dimensions then

\[
J_x(h) = 
\begin{bmatrix}
\frac{\partial h_1}{\partial x_1} & \cdots & \cdots & \frac{\partial h_1}{\partial x_n} \\
\frac{\partial h_2}{\partial x_1} & \cdots & \cdots & \frac{\partial h_2}{\partial x_n} \\
\vdots & \ddots & \ddots & \vdots \\
\frac{\partial h_k}{\partial x_1} & \cdots & \cdots & \frac{\partial h_k}{\partial x_n}
\end{bmatrix}
\]
If the input has $n$ dimensions and the hidden layer has $k$ dimensions then

In other words, the $(l,j)$ entry of the Jacobian captures the variation in the output of the $l^{th}$ neuron with a small variation in the $j^{th}$ input.

$$J_x(h) = \begin{bmatrix}
\frac{\partial h_1}{\partial x_1} & \cdots & \cdots & \frac{\partial h_1}{\partial x_n} \\
\frac{\partial h_2}{\partial x_1} & \cdots & \cdots & \frac{\partial h_2}{\partial x_n} \\
\vdots & \ddots & \ddots & \vdots \\
\frac{\partial h_k}{\partial x_1} & \cdots & \cdots & \frac{\partial h_k}{\partial x_n} 
\end{bmatrix}$$
If the input has $n$ dimensions and the hidden layer has $k$ dimensions then

In other words, the $(l, j)$ entry of the Jacobian captures the variation in the output of the $l^{th}$ neuron with a small variation in the $j^{th}$ input.

$$
J_x(h) = \begin{bmatrix}
\frac{\partial h_1}{\partial x_1} & \cdots & \cdots & \frac{\partial h_1}{\partial x_n} \\
\frac{\partial h_2}{\partial x_1} & \cdots & \cdots & \frac{\partial h_2}{\partial x_n} \\
\vdots & \ddots & \ddots & \vdots \\
\frac{\partial h_k}{\partial x_1} & \cdots & \cdots & \frac{\partial h_k}{\partial x_n}
\end{bmatrix}
$$

$$
\|J_x(h)\|_F^2 = \sum_{j=1}^{n} \sum_{l=1}^{k} \left( \frac{\partial h_l}{\partial x_j} \right)^2
$$
What is the intuition behind this?

\[ \| J_x(h) \|_F^2 = \sum_{j=1}^{n} \sum_{l=1}^{k} \left( \frac{\partial h_l}{\partial x_j} \right)^2 \]

\( \hat{x} \)

\( x \)

\( h \)
• What is the intuition behind this?
• Consider $\frac{\partial h_1}{\partial x_1}$, what does it mean if $\frac{\partial h_1}{\partial x_1} = 0$?

$$\| J_x(h) \|_F^2 = \sum_{j=1}^{n} \sum_{l=1}^{k} \left( \frac{\partial h_l}{\partial x_j} \right)^2$$

$\| J_x(h) \|_F^2$
• What is the intuition behind this?
• Consider $\frac{\partial h_1}{\partial x_1}$, what does it mean if $\frac{\partial h_1}{\partial x_1} = 0$
• It means that this neuron is not very sensitive to variations in the input $x_1$.

$$\|J_x(h)\|_F^2 = \sum_{j=1}^{n} \sum_{l=1}^{k} \left( \frac{\partial h_l}{\partial x_j} \right)^2$$

![Diagram of neural network](image-url)
What is the intuition behind this?
Consider \( \frac{\partial h_1}{\partial x_1} \), what does it mean if \( \frac{\partial h_1}{\partial x_1} = 0 \)?

It means that this neuron is not very sensitive to variations in the input \( x_1 \).

But doesn’t this contradict our other goal of minimizing \( \mathcal{L}(\theta) \) which requires \( h \) to capture variations in the input.

\[
\| J_x(h) \|_F^2 = \sum_{j=1}^{n} \sum_{l=1}^{k} \left( \frac{\partial h_l}{\partial x_j} \right)^2
\]
Indeed it does and that’s the idea

\[ \| J_x(h) \|^2_F = \sum_{j=1}^{n} \sum_{l=1}^{k} \left( \frac{\partial h_l}{\partial x_j} \right)^2 \]
Indeed it does and that’s the idea
By putting these two contradicting objectives against each other we ensure that $h$ is sensitive to only very important variations as observed in the training data.

$$\|J_x(h)\|_F^2 = \sum_{j=1}^{n} \sum_{l=1}^{k} \left( \frac{\partial h_l}{\partial x_j} \right)^2$$
Indeed it does and that’s the idea.

By putting these two contradicting objectives against each other we ensure that $h$ is sensitive to only very important variations as observed in the training data.

$\mathcal{L}(\theta)$ - capture important variations in data

$$\|J_x(h)\|_F^2 = \sum_{j=1}^{n} \sum_{l=1}^{k} \left( \frac{\partial h_l}{\partial x_j} \right)^2$$
- Indeed it does and that’s the idea
- By putting these two contradicting objectives against each other we ensure that $h$ is sensitive to only very important variations as observed in the training data.
- $\mathcal{L}(\theta)$ - capture important variations in data
- $\Omega(\theta)$ - do not capture variations in data

\[
\|J_x(h)\|_F^2 = \sum_{j=1}^{n} \sum_{l=1}^{k} \left( \frac{\partial h_l}{\partial x_j} \right)^2
\]
Indeed it does and that’s the idea
By putting these two contradicting objectives against each other we ensure that $h$ is sensitive to only very important variations as observed in the training data.

$L(\theta)$ - capture important variations in data
$\Omega(\theta)$ - do not capture variations in data
Tradeoff - capture only very important variations in the data

$$\|J_x(h)\|^2_F = \sum_{j=1}^{n} \sum_{l=1}^{k} \left( \frac{\partial h_l}{\partial x_j} \right)^2$$
Let us try to understand this with the help of an illustration.
Consider the variations in the data along directions \( u_1 \) and \( u_2 \).
Consider the variations in the data along directions $u_1$ and $u_2$

It makes sense to maximize a neuron to be sensitive to variations along $u_1$

At the same time it makes sense to inhibit a neuron from being sensitive to variations along $u_2$ (as there seems to be small noise and unimportant for reconstruction)

By doing so we can balance between the contradicting goals of good reconstruction and low sensitivity.

What does this remind you of?
Consider the variations in the data along directions $u_1$ and $u_2$

- It makes sense to maximize a neuron to be sensitive to variations along $u_1$
- At the same time it makes sense to inhibit a neuron from being sensitive to variations along $u_2$ (as there seems to be small noise and unimportant for reconstruction)
Consider the variations in the data along directions $\mathbf{u}_1$ and $\mathbf{u}_2$.

It makes sense to maximize a neuron to be sensitive to variations along $\mathbf{u}_1$.

At the same time it makes sense to inhibit a neuron from being sensitive to variations along $\mathbf{u}_2$ (as there seems to be small noise and unimportant for reconstruction).

By doing so we can balance between the contradicting goals of good reconstruction and low sensitivity.
Consider the variations in the data along directions $u_1$ and $u_2$

It makes sense to maximize a neuron to be sensitive to variations along $u_1$

At the same time it makes sense to inhibit a neuron from being sensitive to variations along $u_2$ (as there seems to be small noise and unimportant for reconstruction)

By doing so we can balance between the contradicting goals of good reconstruction and low sensitivity.

What does this remind you of?
Module 7.7 : Summary
$\hat{x} \equiv x$

$P^T X^T X P = D$
\[ h \equiv \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \]

\[ P^T X^T X P = D \]

\[ \min_\theta \| X - HW^* \|_F^2 \]

(SVD)
Regularization $\Omega(\theta) = \lambda \| \theta \|_2$

Weight decaying $\Omega(\theta) = k \sum_{l=1}^{\rho} \log \hat{\rho}_l + (1 - \rho) \log (1 - \hat{\rho}_l)$

Sparse $\Omega(\theta) = n \sum_{j=1}^{k} \sum_{l=1}^{\rho} \left( \frac{\partial h_l}{\partial x_j} \right)^2$

Contractive Mitesh M. Khapra

CS7015 (Deep Learning) : Lecture 7
Regularization

\[ \Omega(\theta) = \lambda \| \theta \|^2 \]

Weight decay:

\[ \Omega(\theta) = k \sum_{l=1}^{\rho} \rho \log \hat{\rho}_l + (1 - \rho) \log (1 - \hat{\rho}_l) \]

Sparse:

\[ \Omega(\theta) = n \sum_{j=1}^{k} \sum_{l=1}^{\rho} \left( \frac{\partial h_l}{\partial x_j} \right)^2 \]

Contractive
Regularization

\[ \Omega(\theta) = \lambda \| \theta \|^2 \]

Weight decaying

\[ \Omega(\theta) = \sum_{l=1}^{k} \rho \log \hat{\rho}_l + (1 - \rho) \log (1 - \hat{\rho}_l) \]

Sparse

\[ \Omega(\theta) = \sum_{j=1}^{n} \sum_{l=1}^{k} (\partial h_l / \partial x_j)^2 \]
Regularization

\[ \Omega(\theta) = \lambda \| \theta \|^2 \]  

- **Weight decaying**

\[ \Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\hat{\rho}_l} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_l} \]

- **Sparse**
Regularization

\[ \Omega(\theta) = \lambda \| \theta \|^2 \]  

Weight decaying

\[ \Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\hat{\rho}_l} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_l} \]  

Sparse

\[ \Omega(\theta) = \sum_{j=1}^{n} \sum_{l=1}^{k} \left( \frac{\partial h_l}{\partial x_j} \right)^2 \]  

Contractive

\[ \Omega(\theta) = \sum_{j=1}^{n} \sum_{l=1}^{k} \left( \frac{\partial h_l}{\partial x_j} \right)^2 \]