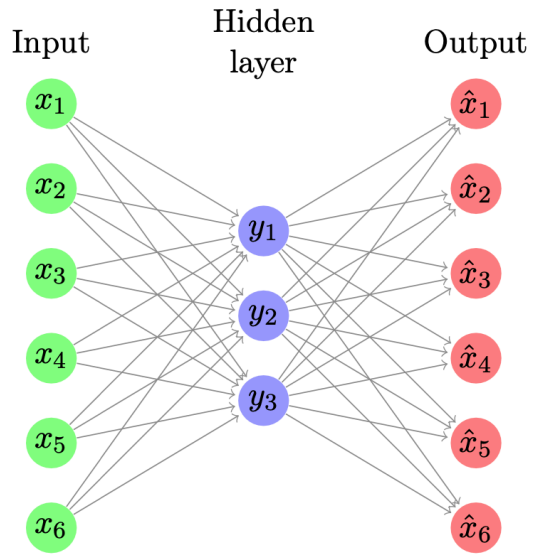
3.3 Autoencoders

An autoencoder is a particular network architecture allowing to do dimension reduction. The idea is to learn to *encode* some vector $x \in \mathbb{R}^D$ in the feature space by another latent vector $y \in \mathbb{R}^d$, and then decode y into a vector $\hat{x}$ with the hope that $\hat{x} \approx x$.
By taking $d << D$, it would in principle allow to do dimension reduction.
The architecture of a 1-hidden layer neural network is given in this figure



In this case, the autoencoder can be written as

$$\hat{x} := \hat{f}(x) := W_2\sigma(W_1x + b_1) + b_2$$

Remember: x is input data, W_1 is a weight matrix that maps input from $\mathbb{R}^d$ to $\mathbb{R}^D$ (the number of columns in W_1 must match the dimension of the input vector) where D (number of columns) is the original dimension and d is the reduced dimension. Analogue, W_2 maps the compressed representation ($\mathbb{R}^d$) back to the original space $\mathbb{R}^d$
with $x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{d\times D}, W_2 \in \mathbb{R}^{D\times d}, b_1 \in \mathbb{R}^d, b_2 \in \mathbb{R}^D$

Example

Suppose $D = 4, d = 2$
$W_1 \in \mathbb{R}^{2\times 4}$: Maps from $\mathbb{R}^4$ to $\mathbb{R}^2$ (Matrix with 2 rows and 4 columns), $W_2 \in \mathbb{R}^{4\times 2}$ maps back to $\mathbb{R}^4$
Now multiplying for example

$$W_1 = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

So multiplying

$$z = W_1 x = \begin{bmatrix} w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + w_{14}x_4 \\ w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + w_{24}x_4 \end{bmatrix}$$

results in a 2 dimensional vector! (reduced dimension). So $W \in \mathbb{R}^{d \times D} x \in \mathbb{R}^D$ results in a d-dimensional vector. **Remember**: The number of columns in any matrices A must equal to the number of rows in B, else the matrix multiplication won't work. That's why the dimension of W are defined like that.

The latent vector is $y = \sigma(W_1 x + b_1)$ (compressed representation). In a classical autoencoder, since the purpose is to reconstruct x from itself, the classical loss function is chosen to be, for a dataset $\{x_i\}_{1 \leq i \leq N}$:

$$\mathcal{L}_{AE}(\theta) := \frac{1}{N} \sum_{i=1}^{N} ||\hat{x}_i - x_i||^2$$

Explanation

You may be wondering why θ is passed since no parameters are used in the equation but remember that $\hat{x}_i = W_2 \sigma(W_1 x_i + b_1) + b_2$. The reconstruction error $||\hat{x}_i - x_i||^2$ depends on $\hat{x}_i$ which in turn depends on θ.

Remember

$$||v||^2 = v_1^2 + v_2^2 + \ldots v_D^2$$

So here, it's

$$(\hat{x}_{i1} - x_{i1})^2 + (\hat{x}_{i2} - x_{i2}) + \ldots$$

where i refers to the i-th data point in the dataset. The subscript 1 or 2 etc refer to the first component of the vector x_i

Example

Suppose you have a dataset of 3 data points

$$x_1 = [1.0 \quad 2.0 \quad 3.0 \quad 4.0], \quad x_2 = [5.0 \quad 6.0 \quad 7.0 \quad 8.0], \quad x_3 = [9.0 \quad 10.0 \quad 11.0 \quad 12.0]$$

So for example $x_{11} = 1.0, x_{23} = 5.0, x_{32} = 10.0$

Back to the script (formula)

where θ denotes the parameters of the network (such as W_1, b_1, W_2, b_2 So $\theta = \{W_1, b_1, W_2, b_2\}$).

Minimizing $\mathcal{L}_{AE}$ will force $\hat{x}$ to be as similar as possible to x, while being represented by a vector y of lower dimension. (the last part here means that the autoencoder doesnt just reconstruct x directly. Instead, it first compresses x into latent space y)
Apart from dimension reduction, autoencoders can be used for other purposes, for instance image denoising. **In this setup** (when using autoencoders for image denoising), we perturb ($\sim$ disturb) each data point x_i by some random noise ϵ_i (usually Gaussian) and learn to reconstruct x_i from the corrupted image $x_i + \epsilon_i$ by minimizing

$$\mathcal{L}_{AE} := \frac{1}{N} \sum_{i=1}^{N} ||\hat{f}(x_i + \epsilon_i) - x_i||^2$$

($\hat{f}(x_i + \epsilon_i)$ just returns a reconstructed *denoised* image). By minimizing L_{AE} the autoencoder learns to remove noise and reconstruct the original image!
While usually, autoencoders verifies the bottleneck property (latent representation) $d << D$, it may be interesting to allow for $d \geq D$ (latent space can be larger than input space) while forcing some sparsity on y.

What does sparsity mean?

Sparsity (=spärlich) means most of the elements in the latent vector y are **zero** or close to zero
This forces the autoencoder to **focus** on a small number of **important** features in the data, rather than using all available dimensions.

Back to the script

This can be achieved by taking the loss $\mathcal{L} = \mathcal{L}_{AE} + \mathcal{L}_{sp}$ ($\mathcal{L}_{sp}$ is the sparsity penalty) penalizes large values $y_i = \sigma(W_1 x_i + b_i)$. For instance,

$$\mathcal{L}_{sp}(\theta) = \frac{1}{N} \sum_{i=1}^{N} ||y_i||_1$$

leads to the L^1-sparse autoencoder.

Explanation

$||y_i||_1$ Is the sum of absolute values of the elements in y_i. y_i is the **latent representation** of the i-th data point. $||y_i||_1$ is the L^1-norm of y_i, which is the sum of absolute values of its elements.
For example (values are just imaginary): $||y_i||_1 = |0.1| + |-0.3| + |0.0| + |0.7| = 1.1$
Minimizing $\mathcal{L}_{sp}$ encourages y_i to have many zeros.

L_{sp} is the average L^1-norm over all data points in the dataset (bc we average $||y_i||_1$ over all data points!)

3.3.1 Linear autoencoder

In this section, we treat the case of linear autoencoder, i.e. the case where σ is the identity function ($\sigma(z) = z$). This case may seem uninteresting at first glance, but we will see that it nicely links to a previously seen algorithm.
In this particular case, the network reduces to

$$\hat{f}(x) = W_2\sigma(W_1x + b_1) + b_2$$

Since $\sigma(z) = z$, just multiply everything out:

$$\hat{f}(x) = W_2W_1x + W_2b_1 + b_2$$

while the loss is

$$\mathcal{L}_{AE}(b_1, b_2, W_1, W_2) = \frac{1}{N}\sum_{i=1}^{N}||W_2W_1x_i + W_2b_1 + b_2 - x_i||^2$$

(Notice that above we had θ but here since σ is the identity function we can just substitute $\hat{x}_i$ with our network)
and we aim at solving

$$\min_{b_1, b_2, W_1, W_2} \mathcal{L}_{AE}(b_1, b_2, W_1, W_2)$$

We want to make the optimization easier, first we eliminate b_2 because this reduces the number of variables we need to optimize.
First, by taking the gradient w.r.t b_2 and using the optimality condition $\nabla_{b_2}\mathcal{L}_{AE} = 0$, we get that $b_2 = \overline{x} - W_2W_1\overline{x} - W_2b_1$.

Gradient calculation

$$\nabla_{b_2}\mathcal{L}_{AE} = \frac{2}{N}\sum_{i=1}^{N}(W_2W_1x_i + W_2b_1 + b_2 - x_i)$$

Setting the gradient to zero:

$$\frac{2}{N}\sum_{i=1}^{N}(W_2W_1x_i + W_2b_1 + b_2 - x_i) = 0$$

Simplifying:

$$\sum_{i=1}^{N}(W_2W_1x_i + W_2b_1 + b_2 - x_i) = 0$$

Splitting the sum:

$$W_2W_1\sum_{i=1}^{N}x_i + W_2b_1N + b_2N - \sum_{i=1}^{N}x_i = 0$$

Let $\overline{x} = \frac{1}{N}\sum_{i=1}^{N}x_i$ (mean of data points, then)

$$W_2W_1(N\overline{x}) + W_2b_1N + b_2N - N\overline{x} = 0$$

Dividing by N:

$$W_2W_1\overline{x} + W_2b_1 + b_2 - \overline{x} = 0$$

Solving for b_2:

$$b_2 = \overline{x} - W_2W - \overline{x} - W_2b_1$$

Back to the script

Thus the problem is reduced to

$$\min_{W_1,W_2}\frac{1}{N}\sum_{i=1}^{N}||W_2W_1(x_i - \overline{x}) - (x_i - \overline{x})||^2$$

Expanded version

$$\min_{W_1,W_2}\frac{1}{N}\sum_{i=1}^{N}||W_2W_1x_i + W_2b_1 + (\overline{x} - W_2W_1 - \overline{x} - W_2b_1) - x_i||^2$$

(substituted b_2 here into original loss function)
Notice that

$$W_2W_1x_i + W_2b_1 + \overline{x} - W_2W_1 - \overline{x} - W_2b_1 - x_i$$

needs to be simplified. Notice that W_2b_1 and $-W_2b_1$ cancel out:

$$W_2W_1x_i + \overline{x} - W_2W_1\overline{x} - x_i = W_2W_1x_i - W_2W_1\overline{x} + \overline{x} - x_i$$

Factoring out we get

$$W_2W_1(x_i - \overline{x}) + \overline{x} - x_i$$

We can rewrite the last two terms and thus get

$$W_2W_1(x_i - \overline{x}) - (x_i - \overline{x})$$

Back to the script

$$\min_{W_1,W_2}\frac{1}{N}\sum_{i=1}^{N}||W_2W_1(x_i - \overline{x}) - (x_i - \overline{x})||^2$$

Note that both the dependency in b_2 and b_1 have disappeared. Now let $X_0 \in \mathbb{R}^{D \times N}$ be the matrix having $x_i - \overline{x}$ as the i-th column. The previous optimization problem can be rewritten as (dropping the $1/N$ factor because scaling doesn't affect the minimization)

$$\min_{W_2} \min_{W_1} ||W_2 W_1 X_0 - X_0||^2_{\text{Fro}}$$

We now aim at finding a closed-form solution of the inner problem

$$\min_{W_1} ||W_2 W_1 X_0 - X_0||^2_{\text{Fro}}$$

Let $\phi(W_1) = ||W_2 W_1 X_0 - X_0||^2_{\text{Fro}}$. We recall that the gradient of ϕ at W_1 is defined as the matrix $\nabla \phi(W_1)$ such that for all $H \in \mathbb{R}^{d \times D}$, $D\phi(W_1).H = \langle \phi(W_1), H \rangle_{\text{Fro}} = Tr(\nabla \phi(W_1) H^T)$. Going back to the definition of the differential, we can show that

$$\nabla \phi(W_1) = 2X_0(X_0 - W_2 W_1 X_0)^T W_2$$

Hence, the optimality condition is $X_0 X_0^T W_2 = X_0 X_0^T W_1^T W_2^t W_2$. Remark that the matrix $X_0 X_0^T$ is the covariance matrix of the data. Under the assumption that this matrix is invertible, the previous optimality conditions reduces to

$$W_2^T = W_2^T W_2 W_1$$

Lemma 3.3.1

Let $A \in \mathbb{R}^{n \times m}$. Then

$$A^T = A^T A A^\dagger$$

Proof: Write the SVD $A = U \Sigma V^T$
First calculate $A^T A$

$$A^T = (U \Sigma V^T)^T = (V \Sigma^T U^T), \quad A = (U \Sigma V^T)$$
$$A^T A = (V \Sigma^T U^T)(U \Sigma V^T) = V \Sigma^T \Sigma V^T$$

Now compute $A^\dagger$

$$A^\dagger = (V \Sigma^\dagger U^T)$$

$$\begin{aligned} A^T A A^\dagger &= (V \Sigma^T \Sigma V^T)(V \Sigma^\dagger U^T) \\ &= V \Sigma^T \Sigma \Sigma^\dagger U^T \end{aligned}$$

Now lets look at $\Sigma^T \Sigma \Sigma^\dagger$
We know that $\Sigma^\dagger$ is $\frac{1}{\sigma_k}$ on its diagonal and Σ is σ_k on its diagonal, thus $\Sigma \Sigma^\dagger = I$ and we get:

$$A^T A^\dagger = V \Sigma^T U^T = A^T$$

The previous lemma allows us to take $W_1 = W_2^\dagger$, which further reduces the problem to

3.3.1

$$\min_{W_2} ||W_2 W_2^\dagger X_0 - X_0||_{\text{Fro}}^2$$

Proposition 3.3.1

3.3.1 is equivalent to

3.3.2

$$\max_{W_2} Tr(W_2^\dagger X_0 X_0^T W_2)$$

Proof: Developing:

$$||W_2 W_2^\dagger X_0 - X_0||_{\text{Fro}}^2 = ||X_0||_{\text{Fro}}^2 - 2\langle X_0, W_2, W_2^\dagger X_0\rangle_{\text{Fro}} + ||W_2 W_2^\dagger X_0||_{\text{Fro}}^2$$

we can rewrite $\langle X_0, W_2, W_2^\dagger X_0\rangle_{\text{Fro}} = Tr(W_2 W_2^\dagger X_0 X_0^T) = Tr(W_2^\dagger X_0 X_0^T W_2)$
In a similar way,

$$\begin{aligned} ||W_2 W_2^\dagger X_0||_{\text{Fro}}^2 &= Tr((W_2 W_2^\dagger X_0)(W_2 W_2^\dagger X_0)^T) \\ &= Tr(W_2 W_2^\dagger X_0 X_0^T (W_2^\dagger)^T W_2^T) \\ &= Tr(W_2^\dagger X_0 X_0^T (W_2^\dagger)^T W_2^T W_2) \\ &= Tr(W_2^\dagger X_0 X_0^T W_2) \end{aligned}$$

where we **used the fact that** $(W_2^\dagger)^T W_2^T W_2 = W_2$ using the previous Lemma. Hence, 3.3.1 can be rewritten as

$$\min_{W_2} ||X_0||_{\text{Fro}}^2 - Tr(W_2^\dagger X_0 X_0^T W_2)$$

which is equivalent to 3.3.2
We now restrict to the extreme case where $d = 1$. In this case $W_2 = w_2 \in \mathbb{R}^D$.
By noticing that for all $w_2, Tr(w_2^\dagger X_0 X_0^T w_2) = ||w_2 w_2^\dagger X_0||_{\text{Fro}}^2 \geq 0$, we can assume that $w_2 \neq 0$. In this case, $w_2^\dagger = \frac{1}{||w_2||_2^2} w_2^T$ and thus the problem becomes

$$\max_{w_2 \in \mathbb{R}^D \setminus \{0\}} \frac{w_2^T X_0 X_0^T w_2}{||w_2||_2^2}$$

which is equivalent to

$$\max_{||w_2||_2 = 1} w_2^T X_0 X_0^T w_2$$

As we have seen in the exercises, the optimal w_2 is nothing else that the first eigenvalue of the covariance matrix $X_0 X_0^T$. In other words: a linear autoencoder performs PCA!

With a little more work it is possible to show that this is actually the case for all $d \in \{1, \ldots, D\}$. Hence, a general (nonlinear) autoencoder can be seen as a generaliztion of the PCA