of the input. Backpropagation for each neuron works the same way as it would if it were the only neuron calculating.

The notion of **deep** learning comes from the fact that the alternate reprensentation computed by one layer A can be fed as input to another layer B. This allows networks to infer multiple levels of representation, same as one first processes simple geometrical forms before recognizing more complex compositions. Backpropagation is straight-forwardly computed on layer B. It is computed on layer A by looking at $\frac{\partial d}{\partial input_B}$ instead of $\frac{\partial d}{\partial e}$ which is simply calculated using the rule of chain derivation:

transition sentence

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial x_2}\cdots\frac{\partial x_k}{\partial x} \quad . ? \tag{7}$$

B. Autoencoders

An autoencoder ([5]) is a neuronal network with a particular architecture which is defined below.

a) A default task for a different goal: Neural networks' ability to extract internal meaningful representations from the original raw data is very enticing. Indeed, providing a meaningful representation of the initial data is one of the biggest hurdles of classical machine learning. However, neural networks need to be trained on a task, and require labeled data to that end.

not clear

Since we are no longer interested in the specific task needed to train the network, it is possible to default to a task that is not interesting in itself but might require the network to learn salient features of the data in order to be completed. A very natural one is to match the output to the input, which would not require man made labeling. Hopefully, the learned representation will capture features specific to the input so that it may be reconstructed.

ref

b) Structure:

link w/ text?

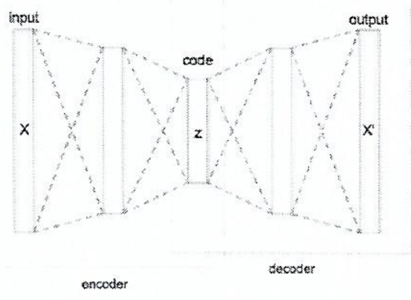
$$x_{latent} = encoder(x_{input}) \tag{8}$$

$$x_{output} = decoder(x_{latent}) \simeq x_{input} \tag{9}$$

Typically, the **input** is *encoded* into a **latent representation** which is then *decoded* into an **output** that should match the input as closely as possible, hence the autoencoder denomination. The main danger of such an approach is having the networks that does not transform the input at all and yields a latent representation that is no different to the raw input data. A multitude of solutions exist to avoid such a scenario:

- Compress the input into a latent representation of lower dimension
- Put the input through a corrupting input layer: the autoencoder cannot just do nothing and give back the same thing because it never had the actual input to begin with
- Use various regularisation methods.

Figure 4. Structure of an autoencoder



c) Why use autoencoders?: Autoencoders can be used for denoising, using corrupted data as input and the original data as objective.

not defined

Autoencoders may also be very useful to learn a representation. In fact, the latent represention contains enough information to reconstruct the data.

In our context, autoencoders could be used to find a representation of a speaker, by giving to the autoencoder a supervector from a speaker as input, and an another supervector from the same speaker as objective. This idea is based on considerating the within-speaker variability as a noise, and using a denoising autoencoder. Repeating this with several pairs of supervectors, we may learn a representation of the speaker. This new representation, as well as i-vectors, is more condensed than supervectors.

Since the problem is symmetrical, we can perform the learning in both directions. One way to perform this training is to use an autoencoder that reconstruct each supervector from the other using tied weights.

C. Tied weight autoencoder

We talk about tied weight autoencoder when the same weights appears in two different places in the autoencoder. For example, in the architecture proposed in [11] (Fig. 5), the weights in light blue in the upper part of the autoencoder are the same as the weights in light blue in the lower part. So, if the function that transform the upper first hidden layer to the upper part of the representation is

$$h_1(x) = f(W \times x + b_1) \tag{10}$$

then the function that transform the lower part of the representation to the next layer is

$$h_2(x) = f(W^T \times x + b_2) \tag{11}$$

In this architecture, the pink weights are also tied.

This architecture learns a joint representation of the two modalities which gives encouraging results multimodal query expansion [11]. In our context, the two modalities will be two supervectors from the same speaker.