

Lecture 4. Deep learning architectures

Bayesian Statistical Learning

24.03.2025-28.03.2025 Instructors: Alina Bazarova, Jose Robledo

Ingredients of the Deep Learning framework. Reminder

Data: training set (train the model), validation set (compare models), test set (final evaluation of the model)

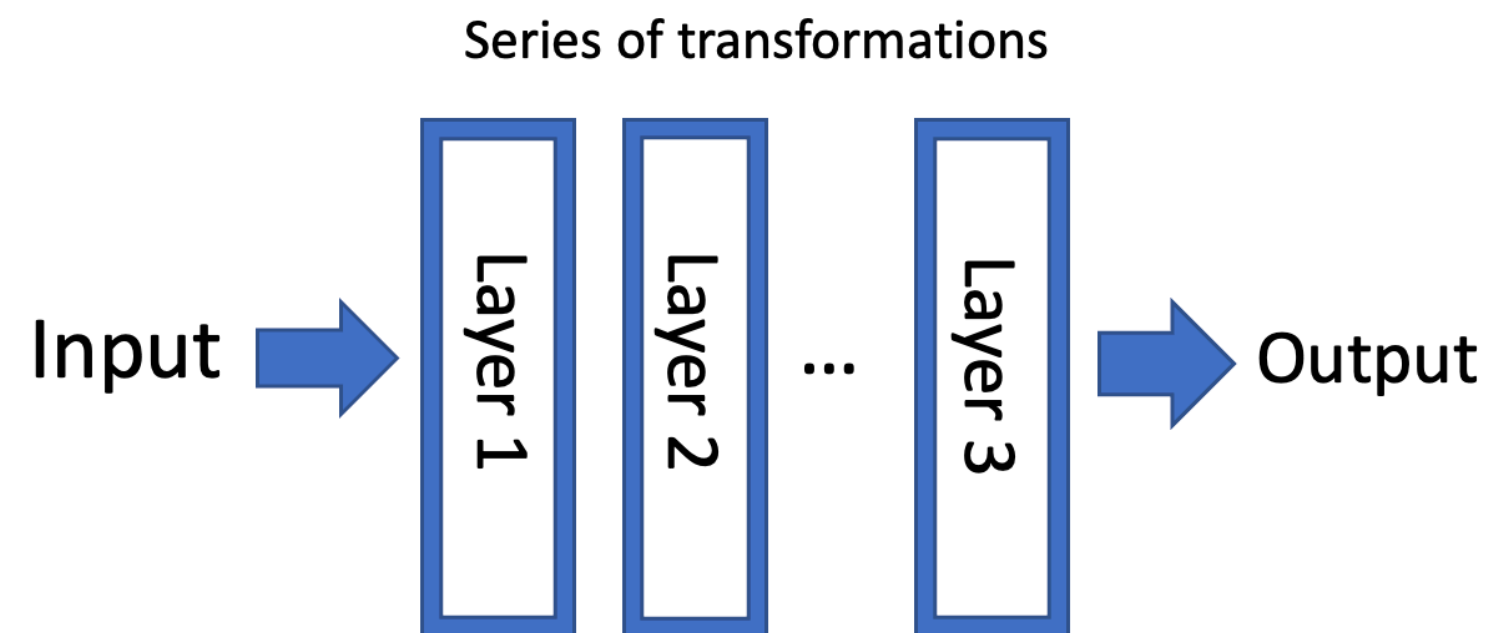
Data can be labelled (supervised learning), unlabelled (unsupervised learning), partially labelled (semi-supervised learning) etc.

Ingredients of the Deep Learning framework. Reminder

Data: training set (train the model), validation set (compare models), test set (final evaluation of the model)

Data can be labelled (supervised learning), unlabelled (unsupervised learning), partially labelled (semi-supervised learning) etc.

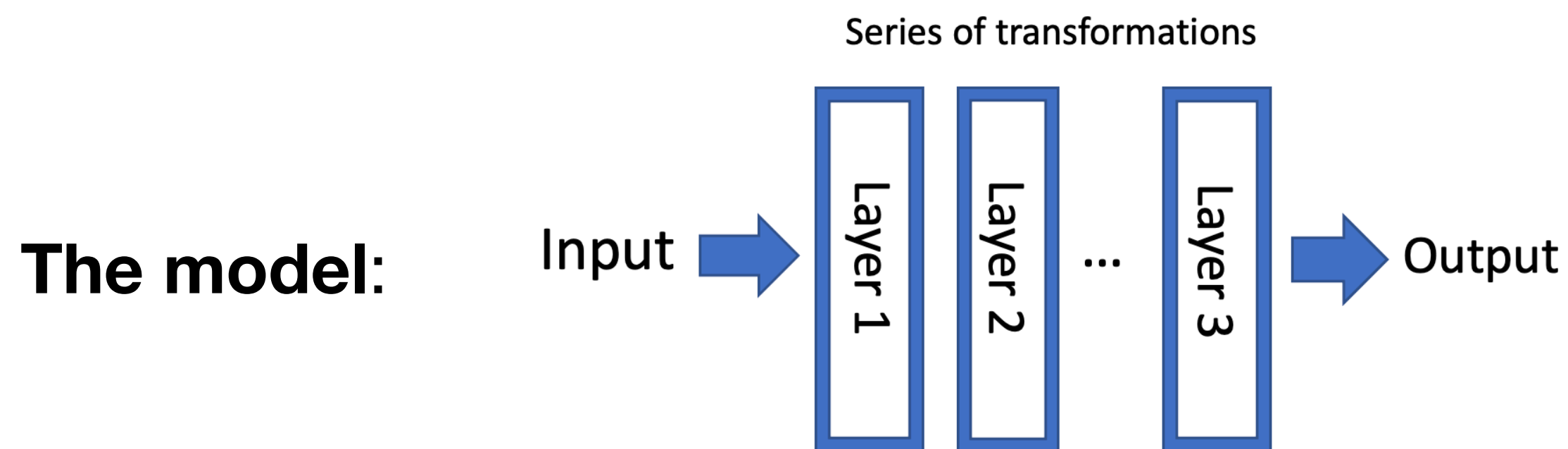
The model:



Ingredients of the Deep Learning framework. Reminder

Data: training set (train the model), validation set (compare models), test set (final evaluation of the model)

Data can be labelled (supervised learning), unlabelled (unsupervised learning), partially labelled (semi-supervised learning) etc.

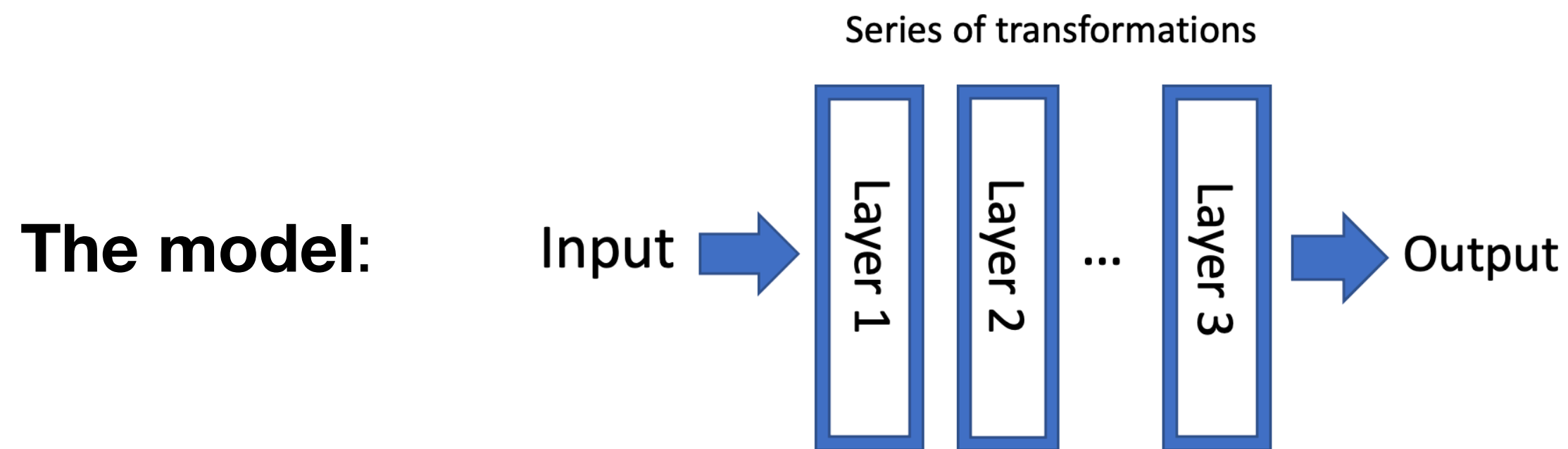


Training: backpropagation, i.e. minimising the given loss function using gradient descent method -> updating the weights of the model

Ingredients of the Deep Learning framework. Reminder

Data: training set (train the model), validation set (compare models), test set (final evaluation of the model)

Data can be labelled (supervised learning), unlabelled (unsupervised learning), partially labelled (semi-supervised learning) etc.



Training: backpropagation, i.e. minimising the given loss function using gradient descent method -> updating the weights of the model

/exactly what we did previously when minimising Kullback-Leibler divergence(maximising free energy), but the model is way more complicated/

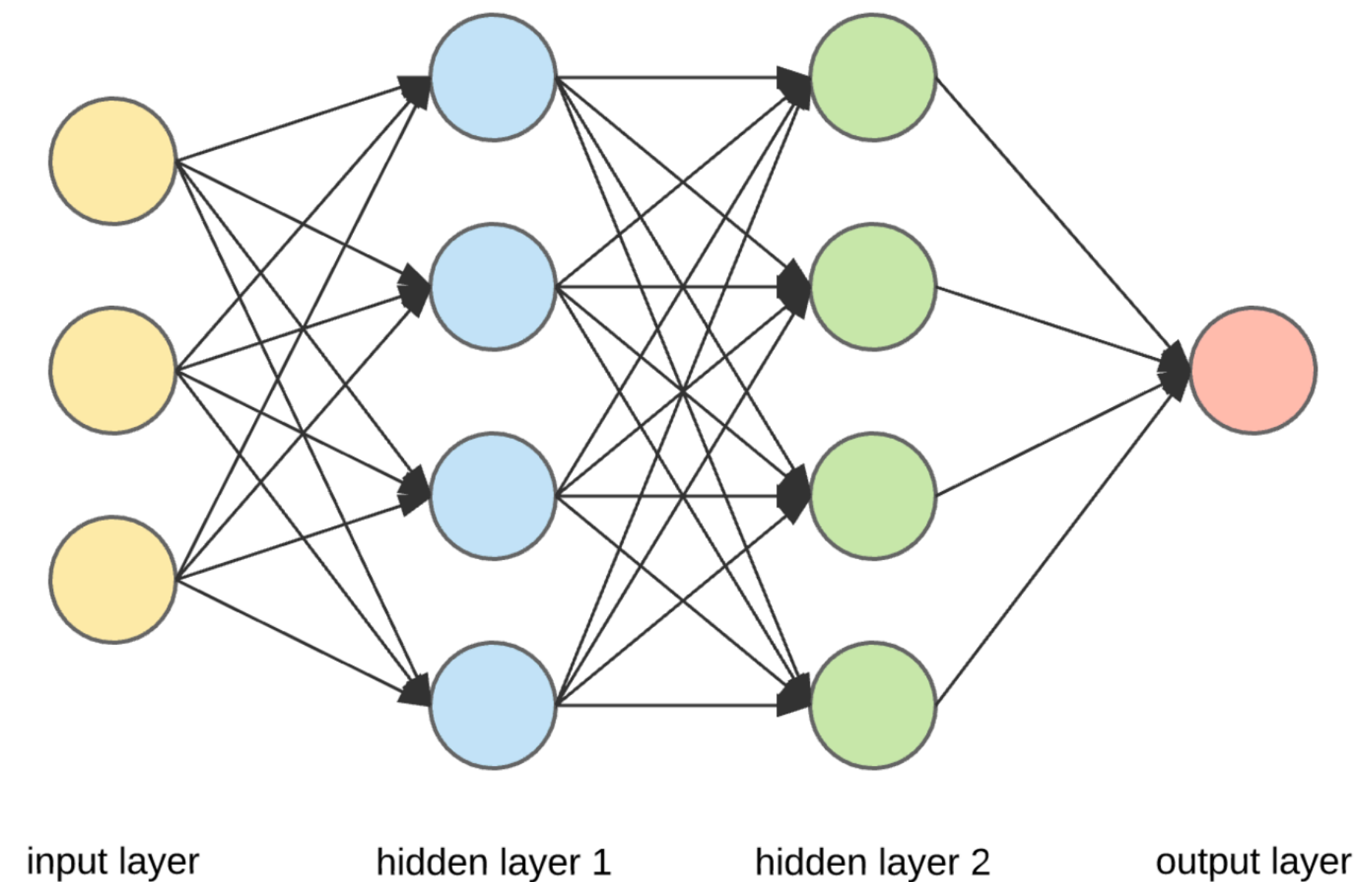
Neural Networks with dense layers

Each layer is essentially a linear transformation $z = Wx + b$

x input layer, z next layer

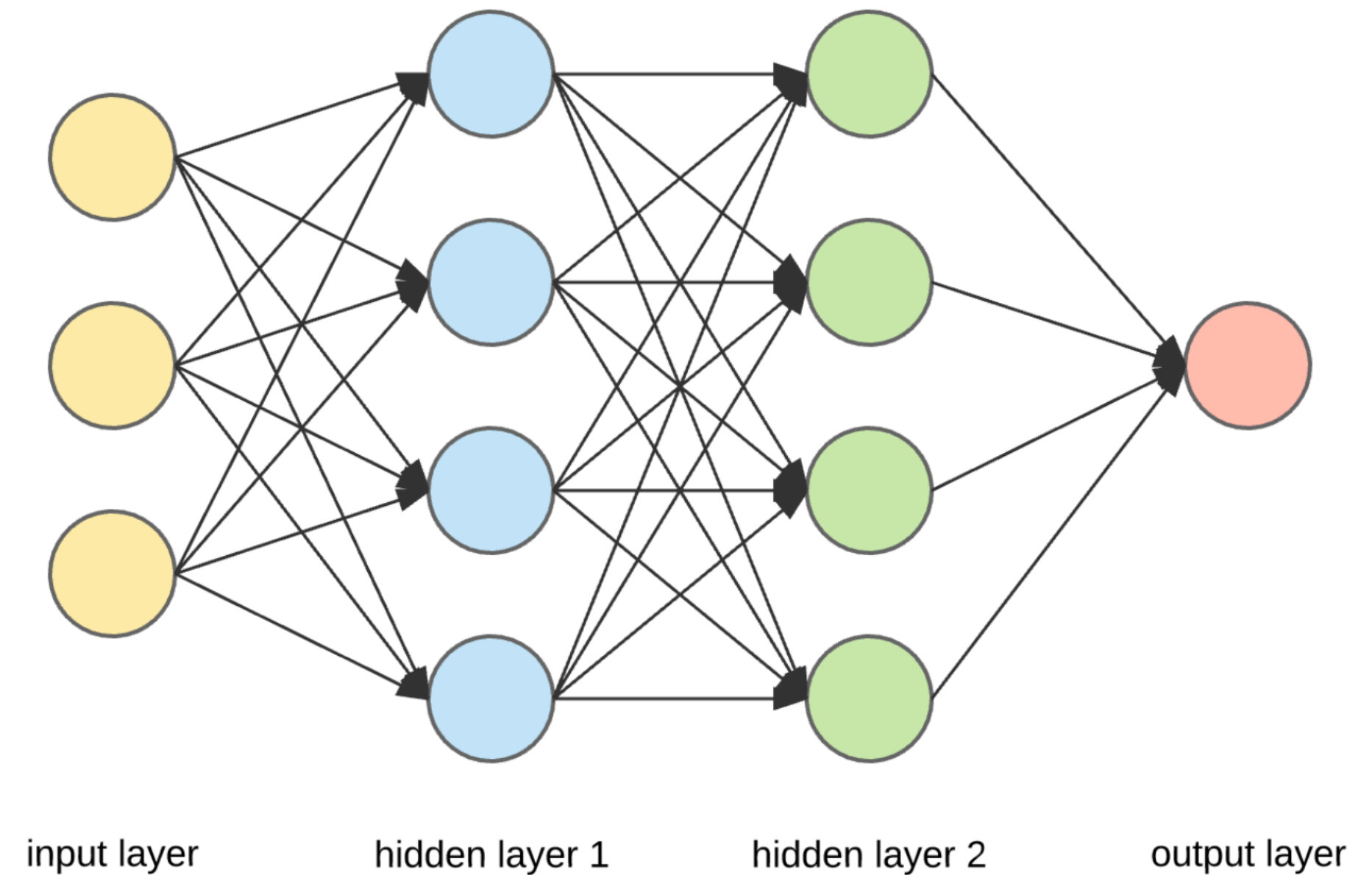
W weight matrix, b - bias

Classical network: all weights are single values



Bayesian Neural Networks with dense layers

Weight matrices W and biases b are in fact **distributions**, which are being learned by means of Variational Bayes and then one can **sample the outputs** from them



Jupyter notebook `bayesian_neural_networks_wine`

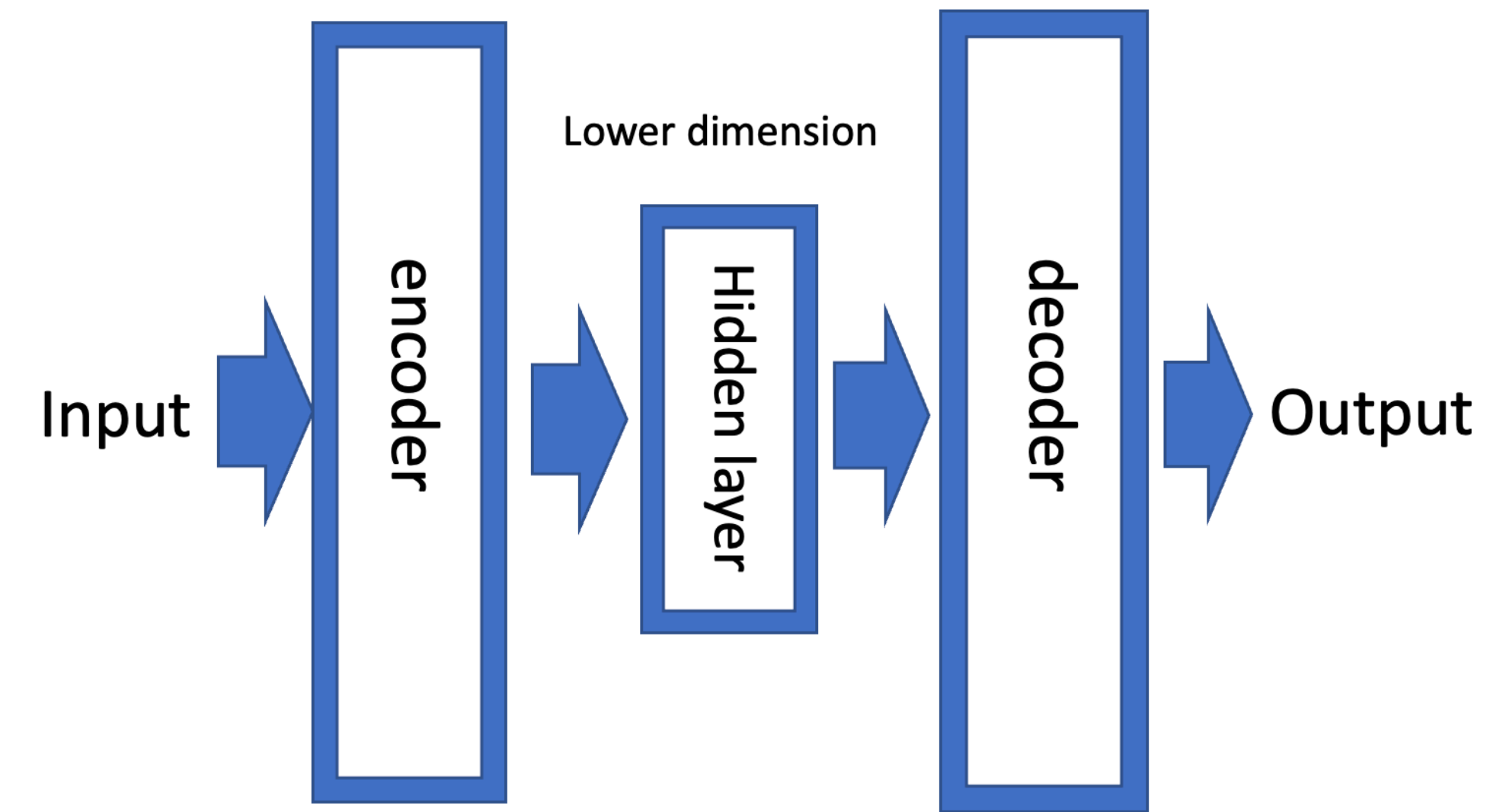
Variational Autoencoder

General autoencoder: **unsupervised** (no labels),

input features are projected onto a lower dimensional

hidden layer (bottleneck) via **encoder**, and then transformed back to the original dimension using **decoder**.

The **aim** is to **reconstruct the original input**.



Variational Autoencoder

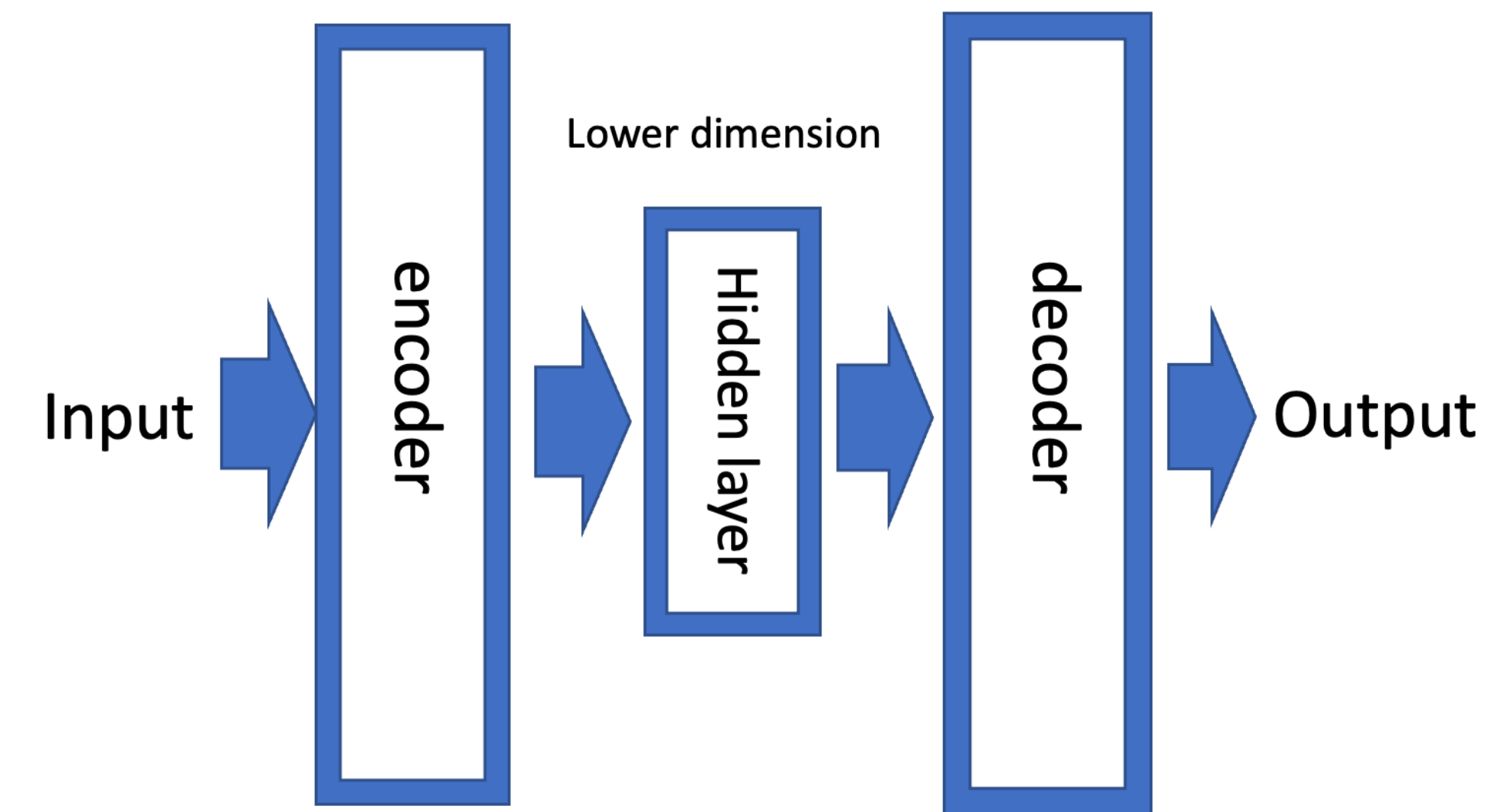
General autoencoder: **unsupervised** (no labels),

input features are projected onto a lower dimensional

hidden layer (bottleneck) via **encoder**, and then transformed

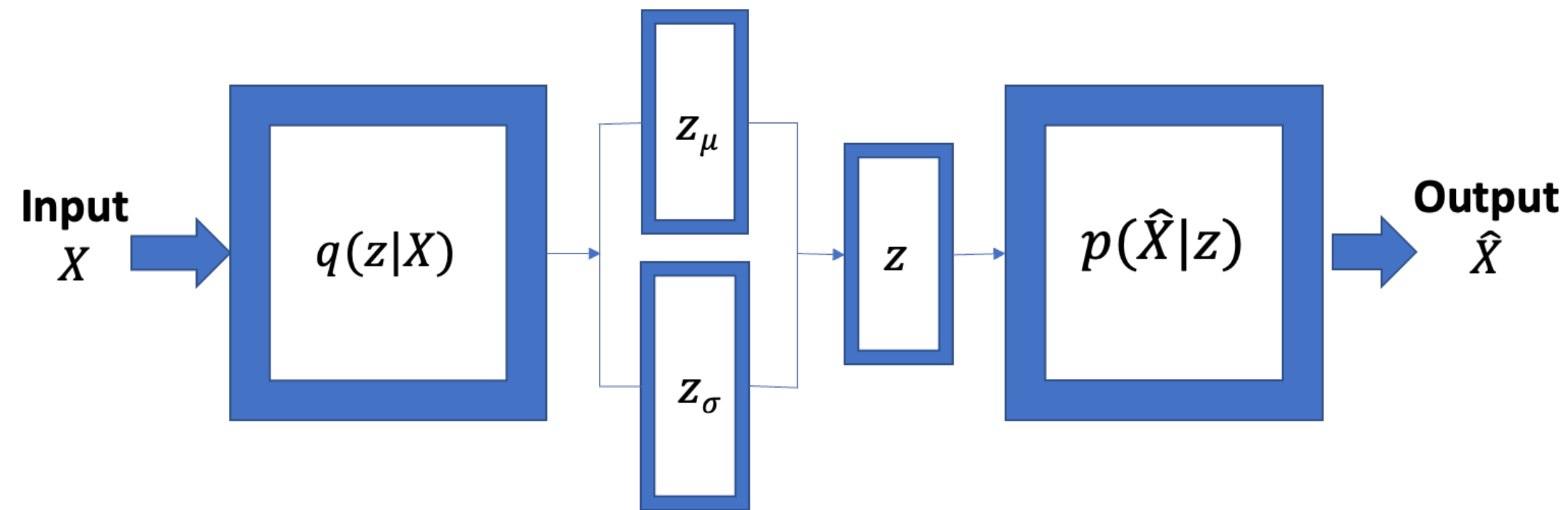
back to the original dimension using **decoder**.

The **aim** is to **reconstruct the original input**.



Variational autoencoder: instead of outputting single values onto the hidden layer it outputs a **probability distribution**, thereby forcing the decoder not to take a deterministic values as input but rather to sample from the provided distributions

Variational autoencoder



$z = z_\mu + z_\sigma \varepsilon$, where $\varepsilon \sim N(0,1)$ (good old reparametrisation trick), hence z_μ and z_σ are deterministic layers

Loss = reconstruction loss + $KL(q(z|X) || p(z))$, where $p(z) \sim N(0,1)$

Very similar set-up to stochastic variational Bayes! Jupyter notebook var_mod

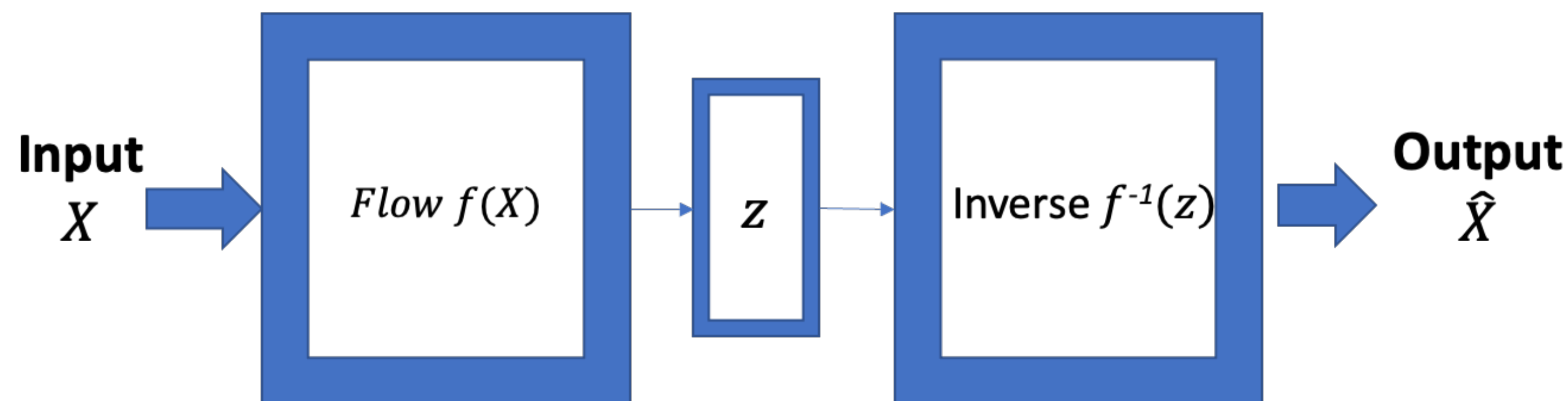
Normalising flows

The major difference compared to VAE:

- Uses **invertible** functions to map onto the latent space z

- For that z has to be the same shape as X

- Given a prior probability density $p_z(z)$ (e.g. normally distributed) and resulting distribution $p_x(x)$ and bijective f



Normalising flows

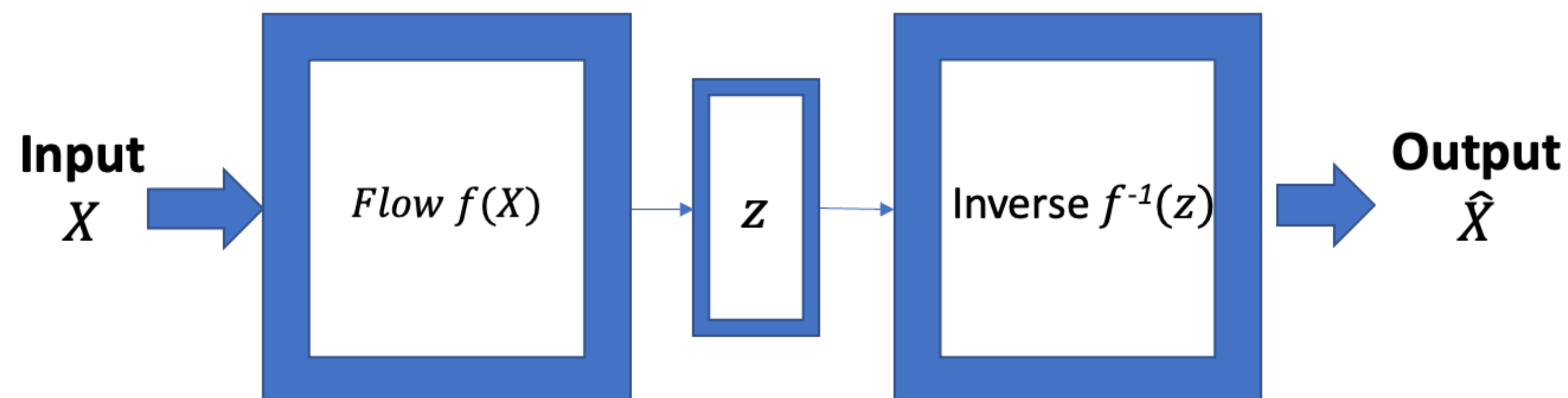
The major difference compared to VAE:

- Uses **invertible** functions to map onto the latent space z

- For that z has to be the same shape as X

- Given a prior probability density $p_z(z)$ (e.g. normally distributed) and resulting distribution $p_x(x)$ and bijective f

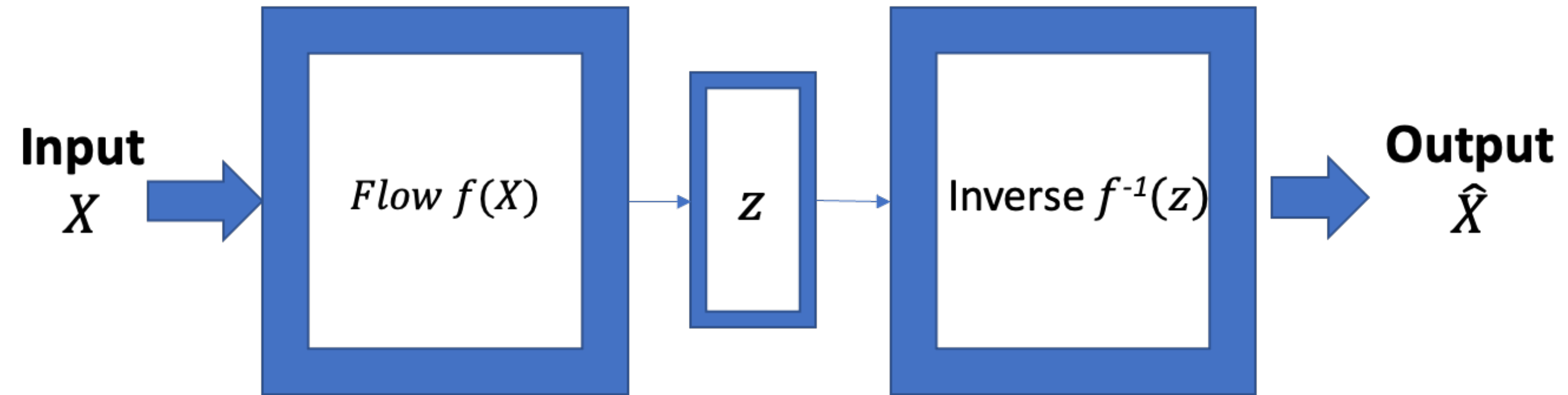
$$\int p_z(z) dz = \int p_x(x) dx = 1$$



Normalising flows

The major difference compared to VAE:

- Uses **invertible** functions to map onto the latent space z



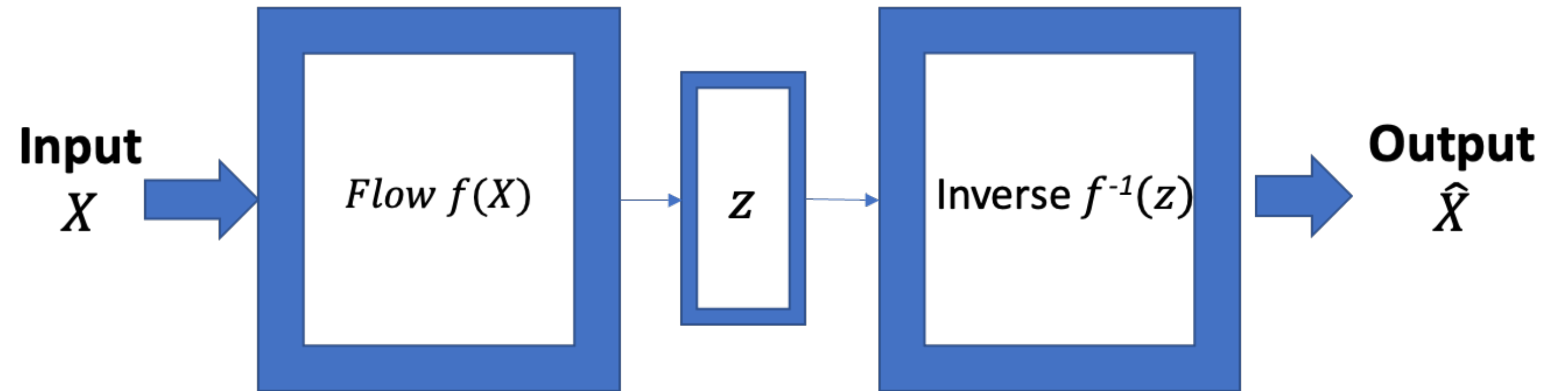
- For that z has to be the same shape as X
- Given a prior probability density $p_z(z)$ (e.g. normally distributed) and resulting distribution $p_x(x)$ and bijective f

$$\int p_z(z) dz = \int p_x(x) dx = 1, p_x(x) = p_z(z) \left| \frac{dz}{dx} \right| = p_z(f(x)) \left| \frac{df(x)}{dx} \right|, \text{ hence}$$

Normalising flows

The major difference compared to VAE:

- Uses **invertible** functions to map onto the latent space z
- For that z has to be the same shape as X
- Given a prior probability density $p_z(z)$ (e.g. normally distributed) and resulting distribution $p_x(x)$ and bijective f



$$\int p_z(z) dz = \int p_x(x) dx = 1, p_x(x) = p_z(z) \left| \frac{dz}{dx} \right| = p_z(f(x)) \left| \frac{df(x)}{dx} \right|, \text{ hence}$$

$$\log p_x(\mathbf{x}) = \log p_z(\mathbf{z}) + \log \det \left(\frac{df(\mathbf{x})}{d\mathbf{x}} \right)$$

Normalising flows

The major difference compared to VAE:

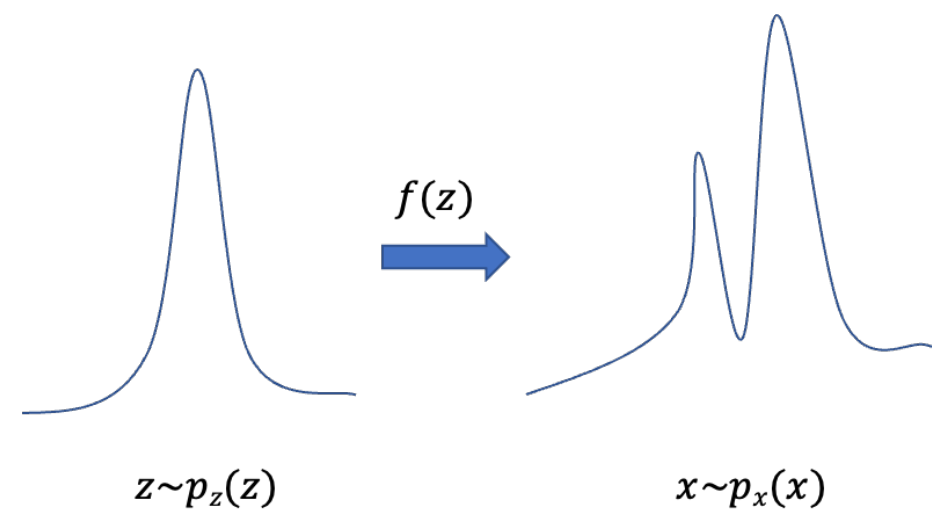
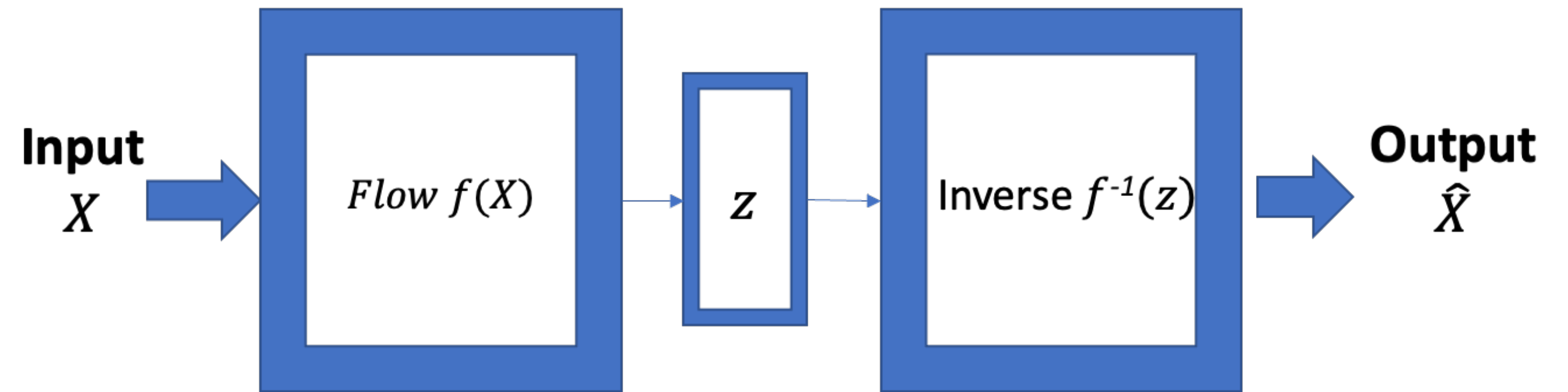
- Uses **invertible** functions to map onto the latent space z

- For that z has to be the same shape as X

- Given a prior probability density $p_z(z)$ (e.g. normally distributed) and resulting distribution $p_x(x)$ and bijective f

$$\int p_z(z) dz = \int p_x(x) dx = 1, p_x(x) = p_z(z) \left| \frac{dz}{dx} \right| = p_z(f(x)) \left| \frac{df(x)}{dx} \right|, \text{ hence}$$

$$\log p_x(\mathbf{x}) = \log p_z(\mathbf{z}) + \log \det \left(\frac{df(\mathbf{x})}{d\mathbf{x}} \right)$$



Visually:

Jupyter notebook flows