Autoencoders

DIG-UM Training course

"Advanced Deep learning"

28.11. – 1.12.22, Meinerzhagen

Autoencoders

Michael Krämer
(RWTH Aachen University)

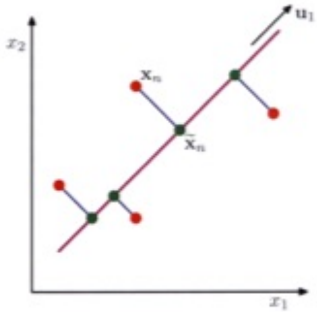
Content:

- Unsupervised learning
- Autoencoders
- Variational autoencoders
- Anomaly detection

Principle component analysis

→ data reduction

→ decorrelation of features



← minimize sum of squares of projection errors

from book by Bishop

Data: $\{\vec{x}^{(1)}, \ldots, \vec{x}^{(m)}\}$ with $\vec{x}^{(i)} \in \mathbb{R}^n$

Want to compress data $\vec{x}^{(i)} \to \vec{c}^{(i)}$ with $\vec{c}^{(i)} \in \mathbb{R}^l$ and $l < n$
but minimal loss of information

Try to find encoding and decoding functions:

$f(\vec{x}) = \vec{c}$ and $\vec{x} \approx g(\vec{c}) = g(f(\vec{x}))$

Simple choice: $g(\vec{c}) = D\vec{c}$; $D \in \mathbb{R}^{n \times l}$

Constraint: $D_{:,i}$ (columns of D) are orthogonal to each other; $D_{:,i}$ have unit norm.

Proceed in 2 steps: 1) find optimal c^*

2) find optimal D^*

1) Find $\vec{c}^* = \arg\min_{\vec{c}} \|\vec{x} - g(\vec{c})\|_2$

or equivalently $\vec{c}^* = \arg\min_{\vec{c}} \|\vec{x} - g(\vec{c})\|_2^2$

$$= \arg\min_{\vec{c}} (\vec{x} - g(\vec{c}))^T (\vec{x} - g(\vec{c}))$$

$\rightarrow (\vec{x} - g(\vec{c}))^T (\vec{x} - g(\vec{c})) = \underbrace{\vec{x}^T\vec{x}}_{\text{not relevant for finding } \vec{c}^*} - \vec{x}^T g(\vec{c}) - \underbrace{g(\vec{c})^T\vec{x}}_{= (g(\vec{c})^T\vec{x})^T = \vec{x}^T g(\vec{c})} + g(\vec{c})^T g(\vec{c})$

$\rightarrow$ consider $\vec{c}^* = \arg\min_{\vec{c}} (-2\vec{x}^T g(\vec{c}) + g(\vec{c})^T g(\vec{c}))$

Recall that $g(\vec{c}) = D\vec{c}$:

$$\vec{c}^* = \arg\min_{\vec{c}} (-2\vec{x}^T D\vec{c} + \vec{c}^T \underbrace{D^T D}_{= \mathbb{1}_l} \vec{c})$$

$$= \arg\min_{\vec{c}} (-2\vec{x}^T D\vec{c} + \vec{c}^T\vec{c})$$

Minimize a function $f(\vec{c})$:

$$\vec{\nabla}_{\vec{c}} (-2\vec{x}^T D\vec{c} + \vec{c}^T\vec{c}) = 0$$

$\Gamma\ \frac{\partial}{\partial c_i} \left(-2 \sum_{j,k}' x_j D_{jk} c_k + \sum_j' c_j^2\right) = -2 \sum_{j,k}' x_j D_{jk} \frac{\partial c_k}{\partial c_i} + 2 \sum_j' c_j \frac{\partial c_j}{\partial c_i}$

$$= -2 \sum_j' x_j D_{ji} + 2 c_i$$

$$\Rightarrow \vec{\nabla}_{\vec{c}} (-2\vec{x}^T D\vec{c} + \vec{c}^T\vec{c}) = -2D^T\vec{x} + 2\vec{c} \,\rfloor$$

$\Rightarrow \vec{c} = D^T\vec{x}$

$\rightarrow$ complete reconstruction: $r(\vec{x}) = g(f(\vec{x})) = D\vec{c} = DD^T\vec{x}$

2) Find optimal matrix D:

data point i, $i = 1, \ldots, m$

$$D^* = \arg\min_D \underbrace{\sqrt{\sum_{i,j}' \left(x_j^{(i)} - r(\vec{x}^{(i)})_j\right)^2}}_{\text{Frobenius norm}} \quad \text{subject to } D^T D = \mathbb{1}_l$$

j-component of vector $\in \mathbb{R}^n$

Consider simplest case: $l = 1$

$\rightarrow D \in \mathbb{R}^{n\times l} \rightarrow \vec{d} \in \mathbb{R}^n$

In the simplest case $l=1$ we consider

$$\vec{d}^* = \arg\min_{\vec{d}} \sum_i \| \vec{x}^{(i)} - \underbrace{\vec{d}\,\vec{d}^T \vec{x}^{(i)}} \|_2^2 \quad \text{subject to } \|\vec{d}\|_2 = 1$$

$$= \vec{d}^T \vec{x}^{(i)} \vec{d} = (\vec{d}^T \vec{x}^{(i)})^T \vec{d}$$

$$= \vec{x}^{(i)T} \vec{d}\,\vec{d}$$

Introduce some helpful notation: design matrix $X \in \mathbb{R}^{m \times n}$:

$$X_{i,:} = \vec{x}^{(i)} \quad \text{or explicitely} \quad X = \begin{bmatrix} x_1^{(1)} & \cdots & x_n^{(1)} \\ x_1^{(2)} & \cdots & x_n^{(2)} \\ \vdots & & \\ x_1^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}$$

Then $\vec{d}^* = \arg\min_{\vec{d}} \| X - X\vec{d}\,\vec{d}^T \|_F^2$ subject to $d^T d = 1$

Let's work out $\| X - X\vec{d}\,\vec{d}^T \|_F^2$

$$\| X - X\vec{d}\,\vec{d}^T \|_F^2 = \mathrm{Tr}\Big(\underbrace{(X - X\vec{d}\,\vec{d}^T)^T (X - X\vec{d}\,\vec{d}^T)}\Big)$$

$$= \underbrace{X^T X}_{\text{not relevant}} - X^T X \vec{d}\,\vec{d}^T - \vec{d}\,\vec{d}^T X^T X + \vec{d}\,\vec{d}^T X^T X \vec{d}\,\vec{d}^T$$

$$= -\mathrm{Tr}(X^T X \vec{d}\,\vec{d}^T) - \underbrace{\mathrm{Tr}(\vec{d}\,\vec{d}^T X^T X)}_{= \mathrm{Tr}(X^T X \vec{d}\,\vec{d}^T)} + \underbrace{\mathrm{Tr}(\vec{d}\,\vec{d}^T X^T X \vec{d}\,\vec{d}^T)}_{= \mathrm{Tr}(X^T X \vec{d} \underbrace{\vec{d}^T \vec{d}}_{=1} \vec{d}^T)}$$

$$= -2\,\mathrm{Tr}(X^T X \vec{d}\,\vec{d}^T) + \mathrm{Tr}(X^T X \vec{d}\,\vec{d}^T)$$

$$\rightarrow \vec{d}^* = \arg\min_{\vec{d}} \big(-\mathrm{Tr}(X^T X \vec{d}\,\vec{d}^T)\big) \quad \text{subject to } \vec{d}^T \vec{d} = 1$$

$$= \arg\max_{\vec{d}} \big(+\mathrm{Tr}(X^T X \vec{d}\,\vec{d}^T)\big) \quad \text{subject to } \vec{d}^T \vec{d} = 1$$

$$= \arg\max_{\vec{d}} \mathrm{Tr}\big(\underbrace{\vec{d}^T X^T X \vec{d}}_{\text{this is a scalar}}\big) \quad \text{— " —}$$

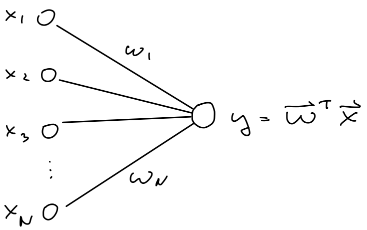
$$= \arg\max_{\vec{d}} (\vec{d}^T X^T X \vec{d}) \quad \text{subject to } \vec{d}^T \vec{d} = 1$$

$\rightarrow$ problem of maximizing quadratic form $\vec{d}^T A \vec{d}$, where $A = X^T X$ is symmetric $\rightarrow$ optimal $\vec{d}$ is given by eigenvector of $X^T X$ with largest eigenvalue.

Unsupervised learning

Machine learning for unlabeled data: find patterns, detect outliers, generate data, ...

let us start with a simple example: Oja's rule



consider data distribution $p(\vec{x})$ and i.i.d. drawn samples $\vec{x} = (x_1, \dots, x_N)^T$

Weight update: $\vec{w} \to \vec{w} + \delta \vec{w}$ with $\delta \vec{w} = \varepsilon y \vec{x}$

ε: learning rate

$\to$ output $|y|$ becomes the larger the more often input feature occurs in data distribution.

Can write this as a differential equation:

$$\tau \frac{d\vec{w}}{dt} = y\vec{x} \quad \text{since} \quad \tau \frac{\vec{w}(t+\Delta t) - \vec{w}(t)}{\Delta t} = y\vec{x}$$

$$\Rightarrow \quad \vec{w}(t+\Delta t) = \vec{w}(t) + \frac{\Delta t}{\tau} y \vec{x}$$

Is this stable?

$$\frac{d\|\vec{w}\|^2}{dt} = 2\vec{w}^T \frac{d\vec{w}}{dt} = 2\vec{w}^T \frac{1}{\tau} y\vec{x} = \frac{2}{\tau} y^2 > 0$$

$\to$ $\vec{w}$ grows without bound

$\to$ add some weight decay:

$$\vec{w} \to \vec{w} + \varepsilon y(\vec{x} - y\vec{w}) \quad \text{or} \quad \tau \frac{d\vec{w}}{dt} = y\vec{x} - \alpha y^2 \vec{w} \qquad \text{[Oja 82]}$$

Is Oja's rule stable?

$$\frac{d\|\vec{w}\|^2}{dt} = 2\vec{w}^T\frac{d\vec{w}}{dt} = \frac{2}{\tau}\vec{w}^T\left(y\vec{x} - \alpha y^2\vec{w}\right) = \frac{2}{\tau}\left(y^2 - \alpha y^2\vec{w}^T\vec{w}\right)$$

$$= \frac{2}{\tau}y^2\left(1 - \alpha\|\vec{w}\|^2\right)$$

$\rightarrow$ converges to $\|\vec{w}\|^2 = 1/\alpha$

What does Hebbian learning do?

Consider average over input data:

$$\tau\frac{d\vec{w}}{dt} = \langle y\vec{x}\rangle_{\vec{x}} = \langle \vec{x}\,\vec{x}^T\vec{w}\rangle_{\vec{x}} = \langle \vec{x}\,\vec{x}^T\rangle_{\vec{x}}\,\vec{w} = C\,\vec{w}$$

data covariance matrix
for input data with zero mean

Let us write $\vec{w}$ in terms of the eigenvectors of C:

$$\vec{w}(t) = \sum_i c_i(t)\,\vec{u}_i$$ [C is real & symmetric $\rightarrow$ $\vec{u}_i$ form basis]

$$\Rightarrow \tau\sum_i\frac{dc_i(t)}{dt}\vec{u}_i = \sum_i c_i(t)\lambda_i\vec{u}_i$$

$$\Rightarrow \tau\frac{dc_i(t)}{dt} = c_i(t)\lambda_i \Rightarrow c_i(t) = c_i(0)\exp(\lambda_i t/\tau)$$

$$\Rightarrow \vec{w}(t) = \sum_i c_i(t)\vec{u}_i = \sum_i c_i(0)\exp(\lambda_i t/\tau)\,\vec{u}_i$$

For $t \gg \tau$, the term corresponding to the largest eigenvalue (say λ_1) dominates:

$\vec{w}(t) \propto \vec{u}_1(t)$ [or $\vec{w}(t) \propto \vec{u}_1/\sqrt{\alpha}$ for Oja's rule]

$\Rightarrow$ Hebbian learning implements the principle component analysis!

Let us now try to implement feature learning on unlabeled data with non-linear neural networks.

Start with data i.i.d. drawn from $p(\vec{x})$, and set up neural network with a hidden layer to map data to itself:

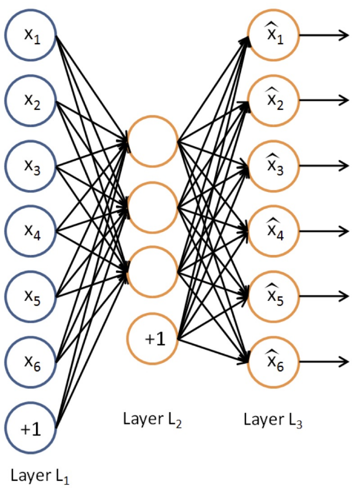
$$\underbrace{\vec{x} \xrightarrow{f} h}_{\text{encoder}} \underbrace{\xrightarrow{g} \hat{\vec{x}}}_{\text{decoder}} \quad \rightarrow \text{autoencoder}$$

Learning the identity mapping is not very useful. Instead construct

undercomplete autoencoders:

- h has lower dimension than $\vec{x}$
- f or g has low capacity (e.g. linear g)
- discard information in h

or autoencoders with regularization.



undercomplete AE
($\dim h < \dim \vec{x}$)

want $\hat{\vec{x}} \approx \vec{x}$

$\rightsquigarrow$ AE needs to learn compressed representation of data

$\rightsquigarrow$ cf. PCA

$\rightsquigarrow$ extract salient features of data!

Regularised autoencoders, e.g. sparse autoencoders or denoising autoencoders:

Sparse autoencoders:

$f(x) = h$ and $g(h) = \hat{x}$; minimize loss function:

$$\underset{f,g}{\arg\min} \sum_{x \in \text{data}} \sum_{m=1}^{M} \mathcal{L}(x_m, g(f(x))_m) + \Omega(h)$$

sparsity penalty on hidden layer

Can derive $\Omega(\vec{h})$ through modeling joint distribution

$$p_{\text{model}}(\vec{x}, \vec{h}) = p(\vec{x}|\vec{h})\, p(\vec{h})$$

prior over latent variables

Maximize log-likelihood:

$$\ln p_{\text{model}}(\vec{x}, \vec{h}) = \ln p_{\text{model}}(\vec{x}|\vec{h}) + \ln p_{\text{model}}(\vec{h})$$

A Laplace prior on latent variables:

$$p_{\text{model}}(h_i) = \frac{\lambda}{2} e^{-\lambda |h_i|}$$

leads to penalty: $\Omega(\vec{h}) = \lambda \sum_i |h_i|$

Alternative way to enforce sparsity:

Keep average activation in each hidden layer node small: $\hat{h}_j = \frac{1}{M} \sum_{m=1}^{M} h_j(x^{(m)}) = h_0$ (e.g. $= 0.05$) (Andrew Ng)

⇝ add penalty term:

$$\Omega(h) = \sum_{j=1}^{S} \left(h_0 \ln(h_0 / \hat{h}_j) + (1-h_0) \ln\left(\frac{1-h_0}{1-\hat{h}_j}\right)\right)$$

sum over hidden units

$$= \sum_{j=1}^{S} KL(h_0 \| \hat{h}_j)$$

Denoising autoencoders: add noise to input and minimize $L(\bar{x}, g(f(\tilde{\bar{x}})))$

copy of $\bar{x}$ corrupted by noise

⇝ autoencoder must learn to undo corruption of data.

Contractive autoencoder: regularize $h = f(x)$ by penalizing derivatives of f:

$$\Omega(h) = \lambda \left\| \frac{\partial f}{\partial x} \right\|_F^2$$

← Frobenius norm

⇝ forces autoencoder to learn function that does not change much when $\bar{x}$ changes slightly.

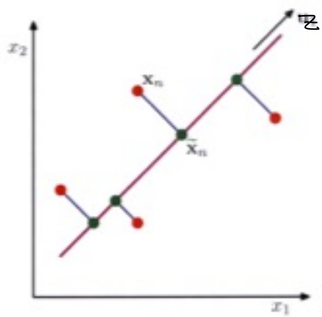
Autoencoders belong to the general class of

latent variable models

LVMs map between observation space $\vec{x} \in \mathbb{R}^D$ and latent space $\vec{z} \in \mathbb{R}^Q$: $f_\theta: \vec{x} \to \vec{z}$; $g_\phi: \vec{z} \to \hat{\vec{x}}$

- one latent variable gets associated with each data point in training set: $\vec{x}^{(m)} \to \vec{z}^{(m)}$
- latent vectors are smaller than observations, $Q < D \Rightarrow$ compression
- models can be linear or non-linear, deterministic or stochastic

Example of a linear, deterministic model: principal component analysis:



Here: find linear mapping $f: \vec{x} \to z$ with $\vec{x} \in \mathbb{R}^2$ and $z \in \mathbb{R}$ and $g: z \to \hat{\vec{x}}$ such as to minimize $\sum_{i=1}^{M} \|\vec{x}^{(i)} - \hat{\vec{x}}^{(i)}\|_2^2$

$= \sum_{i=1}^{M} \|\vec{x}^{(i)} - g(f(\vec{x}^{(i)}))\|_2^2$

Some classification:

	deterministic	non-deterministic
linear	PCA	probabilistic PCA
non-linear	Autoencoder	Variational AE
–"–, no encoder		Gen. adv. networks

Autoencoder:

non-linear fcts, e.g. neural networks

$$\vec{x} \xrightarrow{\;f\;} \vec{z} \xrightarrow{\;g\;} \hat{\vec{x}}$$

$\|\vec{x} - \hat{\vec{x}}\|_2^2$ to minimize

Both the encoder f and the decoder g are deterministic.

In this lecture we want to combine the idea of an autoencoder with the concept of generative modeling

- want to determine models of probability distributions $p(\vec{x})$ over data points $\vec{x}$;
- need to capture structural regularities in the data, e.g. correlations between pixels in images;
- generative latent variable models capture structure of data in distribution of latent variables;
- today we will discuss variational autoencoders, which only approximate $p(\vec{x})$, but allow to draw samples from $p(\vec{x})$.

Bayesian view of generative latent variable models:

$$p(\vec{x}) = \int \underbrace{p(\vec{z})\, p(\vec{x}|\vec{z})}_{\text{generative process}} d\vec{z} = \int p(\vec{x}, \vec{z})\, dz = \mathbb{E}_{\vec{z} \sim p(\vec{z})}\, p(\vec{x}|\vec{z})$$

- $p(\vec{z})$: prior over latent variable $z \in \mathbb{R}^Q$
- $p(\vec{x}|\vec{z})$: likelihood (decoder)
- $p(\vec{x})$: marginal likelihood, or evidence

Goal: maximize $p(\vec{x}) = p_\theta(\vec{x})$ by learning $p_\theta(\vec{z})$ and $p_\theta(\vec{x}|\vec{z})$.

Find model parameters θ by minimizing negative log-likelihood:

$$\theta^* = \arg\min_\theta \mathbb{E}_{\vec{x} \sim p_{data}} \left(-\ln p_\theta(\vec{x})\right)$$

$$= \arg\min_\theta \mathbb{E}_{\vec{x} \sim p_{data}} \left[-\ln \left(\mathbb{E}_{\vec{z} \sim p_\theta(\vec{z})} p_\theta(\vec{x}|\vec{z})\right)\right]$$

$$\propto \arg\min_\theta \sum_{i=1}^{N} -\ln \left(\mathbb{E}_{z \sim p_\theta(\vec{z})} p_\theta(\vec{x}^{(i)}|\vec{z})\right)$$

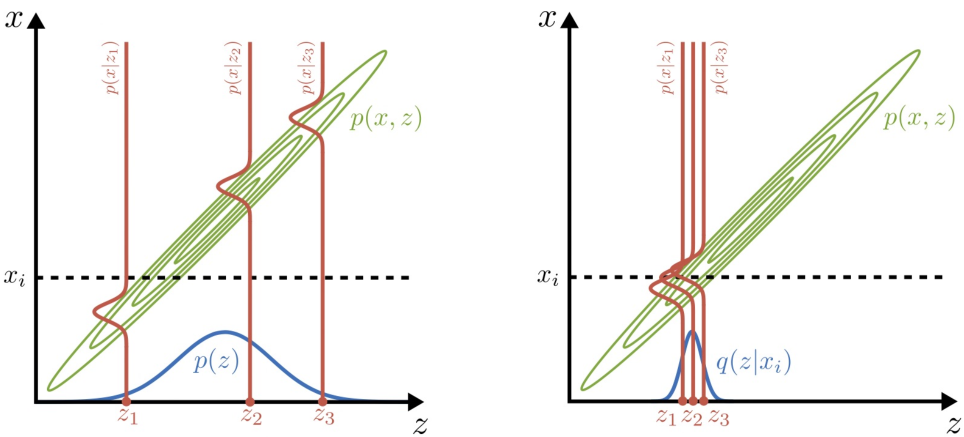
Unfortunately, calculating $p_\theta(\vec{x}^{(i)})$ is in general intractable

$\rightsquigarrow$ introduce a so-called recognition model $q_\theta(\vec{z}|\vec{x})$ to approximate true posterior $p_\theta(\vec{z}|\vec{x})$

In AE-terminology: $p_\theta(\vec{x}|\vec{z}) \rightarrow$ decoder

$p_\theta(\vec{z}|\vec{x}) \rightarrow$ encoder

Example why computing $p_\theta(\vec{x}) = \mathbb{E}_{z \sim p_\theta(z)} p_\theta(\vec{x}|\vec{z})$ is hard.



from DL course Tübingen (Prof. Geiger)

How can we use the recognition model $q(\vec{z}|\vec{x})$ to maximize likelihood?

$$\ln p(\vec{x}) = \mathbb{E}_{z\sim q(\vec{z}|\vec{x})} \ln\left(p(\vec{x})\frac{p(\vec{z}|\vec{x})}{p(\vec{z}|\vec{x})}\right)$$

$$= \mathbb{E}_{z\sim q(\vec{z}|\vec{x})}\left(\ln\left(\frac{p(\vec{z},\vec{x})}{q(\vec{z}|\vec{x})}\right) + \ln\left(\frac{q(\vec{z}|\vec{x})}{p(\vec{z}|\vec{x})}\right)\right)$$

$$= \mathbb{E}_{z\sim q(\vec{z}|\vec{x})} \ln\left(\frac{p(\vec{z},\vec{x})}{q(\vec{z},\vec{x})}\right) + \underbrace{KL\left(q(\vec{z}|\vec{x})\,\|\,p(\vec{z},\vec{x})\right)}_{\geq 0}$$

$$\geq \mathbb{E}_{z\sim q(\vec{z}|\vec{x})} \ln\left(\frac{p(\vec{z},\vec{x})}{q(\vec{z},\vec{x})}\right)$$

$\rightsquigarrow$ evidence lower bound (ELBO)

Note: $q(\vec{z}|\vec{x})$ is variational approximation to true posterior $p(\vec{z}|\vec{x})$

KL-term measures approximation error

Rather than maximizing log-likelihood, minimize negative log-likelihood:

$$-\ln p(\vec{x}) \leq \mathbb{E}_{\vec{z}\sim q(\vec{z}|\vec{x})} \ln\left(\frac{q(\vec{z}|\vec{x})}{p(\vec{z},\vec{x})}\right)$$

$$= \mathbb{E}_{\vec{z}\sim q(\vec{z}|\vec{x})} \ln\left(\frac{q(\vec{z}|\vec{x})}{p(z)\,p(\vec{x}|\vec{z})}\right)$$

$$= \mathbb{E}_{\vec{z}\sim q(\vec{z}|\vec{x})}\left(\ln\left(\frac{q(\vec{z}|\vec{x})}{p(\vec{z})}\right) - \ln p(\vec{x}|\vec{z})\right)$$

$$= KL\left(q(\vec{z}|\vec{x})\,\|\,p(\vec{z})\right) - \underbrace{\mathbb{E}_{\vec{z}\sim q(\vec{z}|\vec{x})}\left(-\ln p(\vec{x}|\vec{z})\right)}_{\sim \text{ reconstruction error}}$$

$\rightsquigarrow$ autoencoder structure: $\vec{x} \xrightarrow{q} \vec{z} \xrightarrow{p} \vec{x}$

Variational autoencoder:

Minimize bound to negative log likelihood:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\ \mathbb{E}_{\vec{x}\sim p_{data}}\left(-\ln p_\theta(\vec{x})\right)$$

$$= \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} \left(-\ln p_\theta(\vec{x}^{(i)})\right)$$

$$\approx \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} \left[KL\left(q_\theta(\vec{z}|\vec{x}^{(i)}) \| p(\vec{z})\right) + \mathbb{E}_{\vec{z}\sim q_\theta(\vec{z}|\vec{x}^{(i)})}\left(-\ln p_\theta(\vec{x}^{(i)}|\vec{z})\right)\right]$$

In VAE, $q_\theta(\vec{z}|\vec{x})$ is a multivariate Gaussian, parametrized by neural network:

$$q_\theta(\vec{z}|\vec{x}) = \frac{1}{(2\pi)^{Q/2}} \frac{1}{|\Sigma_\theta(\vec{x})|^{1/2}} \exp\left(-\frac{1}{2}(\vec{z}-\vec{\mu}_\theta(\vec{x}))^T \Sigma_\theta^{-1}(\vec{x})(\vec{z}-\vec{\mu}_\theta(\vec{x}))\right)$$

→ mean $\vec{\mu}$ and covariance Σ are functions of the data and determined by a neural network.

Standard set-up: choose $q(\vec{z}|\vec{x}) = \mathcal{N}(\vec{z}|\vec{\mu},\Sigma)$ and prior $p(\vec{z}) = \mathcal{N}(0,\mathbb{1})$ as Gaussian, and $\Sigma = \operatorname{diag}(\sigma^2)$

Then: $KL(q(\vec{z})\|p(\vec{z})) = \int q(z)\left(\ln q(z) - \ln p(z)\right) dz$

$$= \frac{1}{2}\sum_{j=1}^{Q}\left(\vec{\mu}_j^2 + \sigma_j^2 - 1 - \ln \sigma_j^2\right)$$

Can we train the VAE using backpropagation?

Need to calculate gradient of an expectation value.

Simple example:

$$\vec{\nabla}_\theta \mathbb{E}_{\vec{z}\sim p(\vec{z})}\left(f_\theta(\vec{z})\right) = \vec{\nabla}_\theta \int p(\vec{z}) f_\theta(\vec{z})\, d\vec{z} = \int p(\vec{z})\vec{\nabla}_\theta f_\theta(\vec{z})\, d\vec{z}$$

$$= \mathbb{E}_{\vec{z}\sim p(\vec{z})}\left[\vec{\nabla}_\theta f_\theta(\vec{z})\right]$$

But what happens if $p(\vec z) = p_{\vec\theta}(\vec z)$?

$$\vec\nabla_\theta \mathbb{E}_{\vec z\sim p(\vec z)}\left[f_\theta(\vec z)\right] = \vec\nabla_\theta\left[\int p_\theta(\vec z)\, f_\theta(\vec z)\, d\vec z\right]$$

$$= \mathbb{E}_{\vec z\sim p_\theta(\vec z)}\left[\vec\nabla_\theta f_\theta(\vec z)\right] + \underbrace{\int f_\theta(\vec z)\,\vec\nabla_\theta p_\theta(\vec z)\, dz}_{=\,?}$$

Reparametrisation trick:

Introduce random variable $\vec\varepsilon \sim p(\vec\varepsilon)$ and write

$z = g_\theta(\vec\varepsilon, \vec x)$ (e.g. $\vec\varepsilon \sim \mathcal{N}(\vec 0, \mathbb{1})$ and $\vec z = \vec\mu(\vec x) + \vec\varepsilon \odot \vec\sigma(\vec x)$)

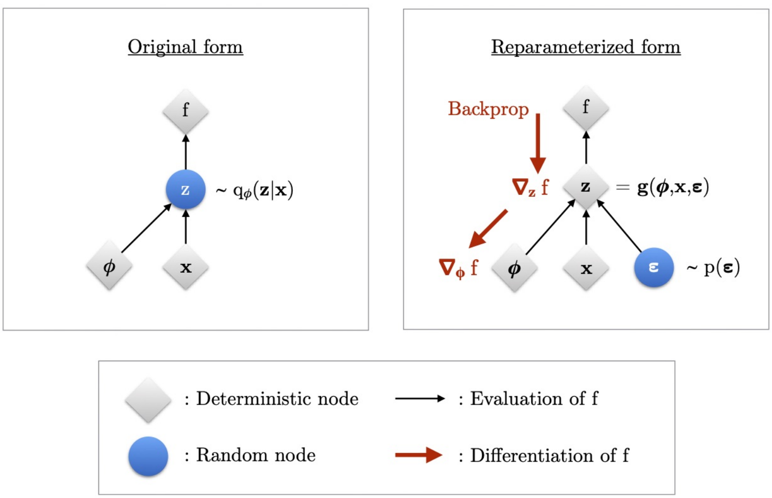
element-wise product

Then $\mathbb{E}_{\vec z\sim p_\theta(\vec z)}\left[f(\vec z)\right] = \mathbb{E}_{\vec z\sim p(\vec\varepsilon)}\left[f(g_\theta(\vec\varepsilon, \vec x))\right]$

$$\Rightarrow \nabla_\theta \mathbb{E}_{z\sim p_\theta(z)}\left[f(\vec z)\right] = \nabla_\theta \mathbb{E}_{\vec z\sim p(\vec\varepsilon)}\left[\,-\,\cdot\,-\,\right]$$

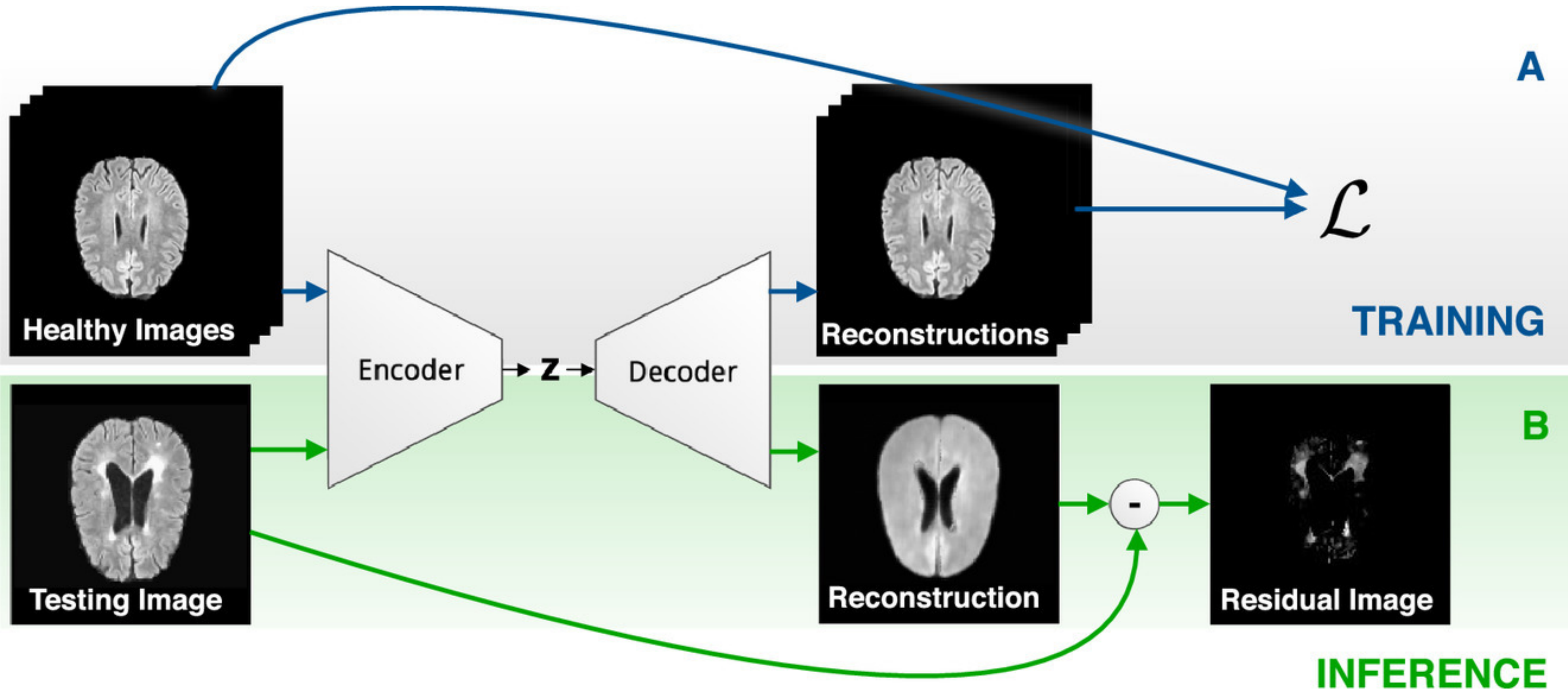
$$= \mathbb{E}_{\vec z\sim p(\vec\varepsilon)}\left[\underbrace{\nabla_\theta f(g_\theta(\vec\varepsilon, \vec x))}\right]$$

evaluate with MC methods: $\frac{1}{L}\sum_{l=1}^{L} \nabla_\theta f(g_\theta(\vec\varepsilon^{(l)}, \vec x))$



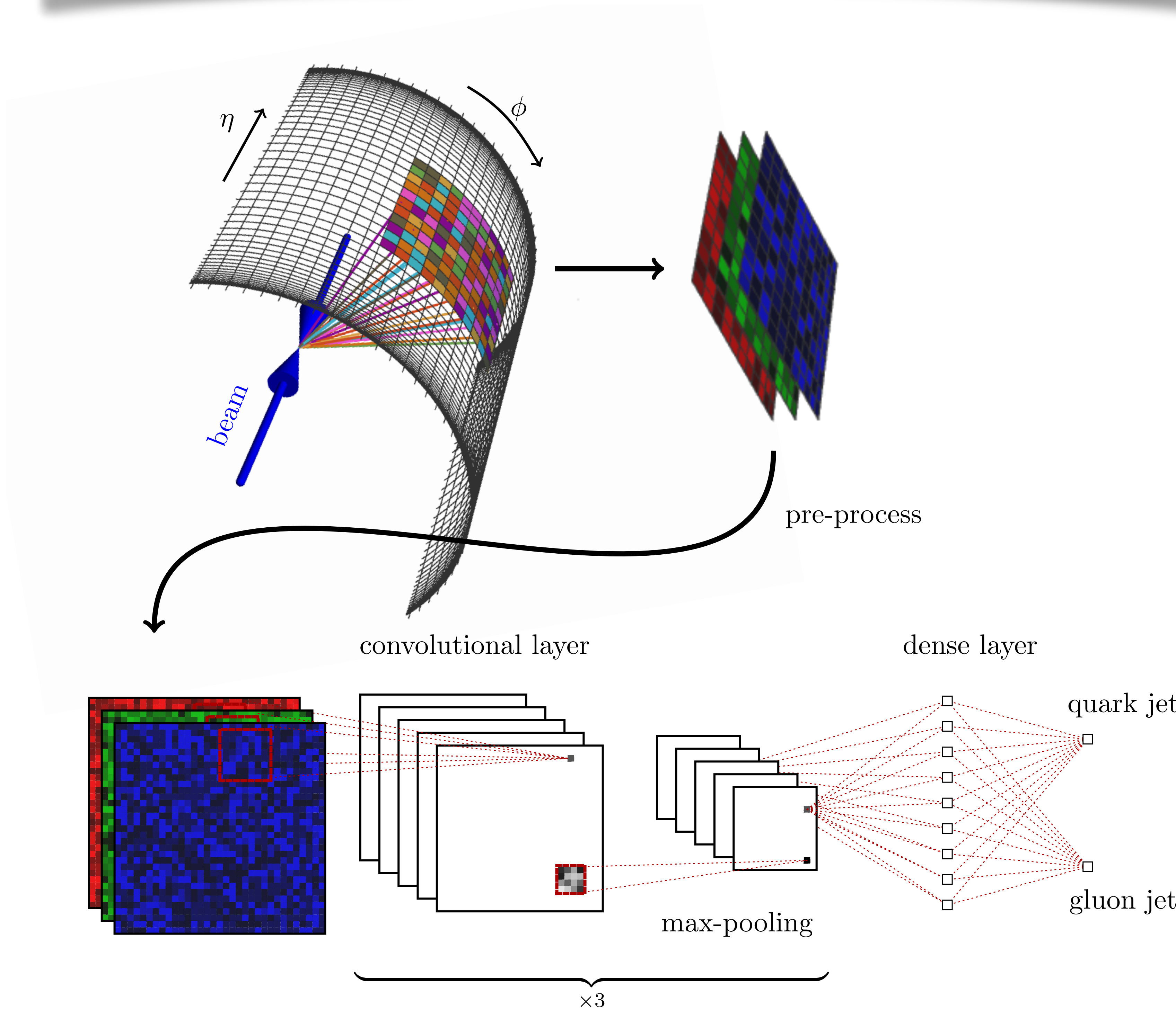
See Kingma and Welling 2014

Anomaly detection with autoencoders



Baur et al., 2020

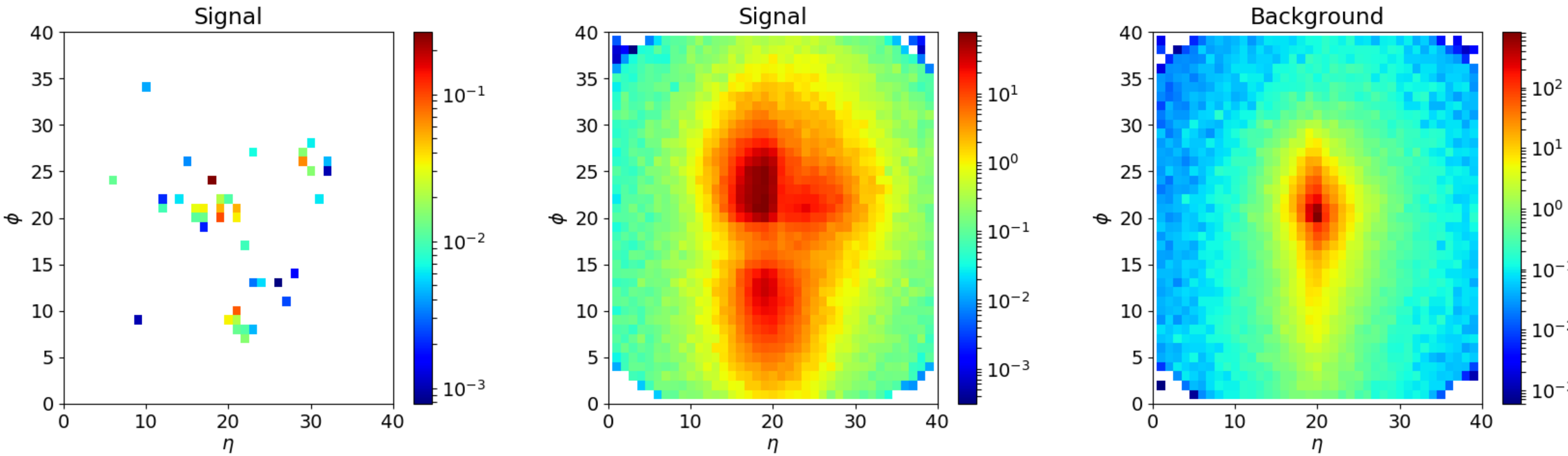
Convolutional neural networks for jet images



Komiske PT, Metodiev EM, Schwartz MD. J. High Energy Phys. 01:110 (2017)

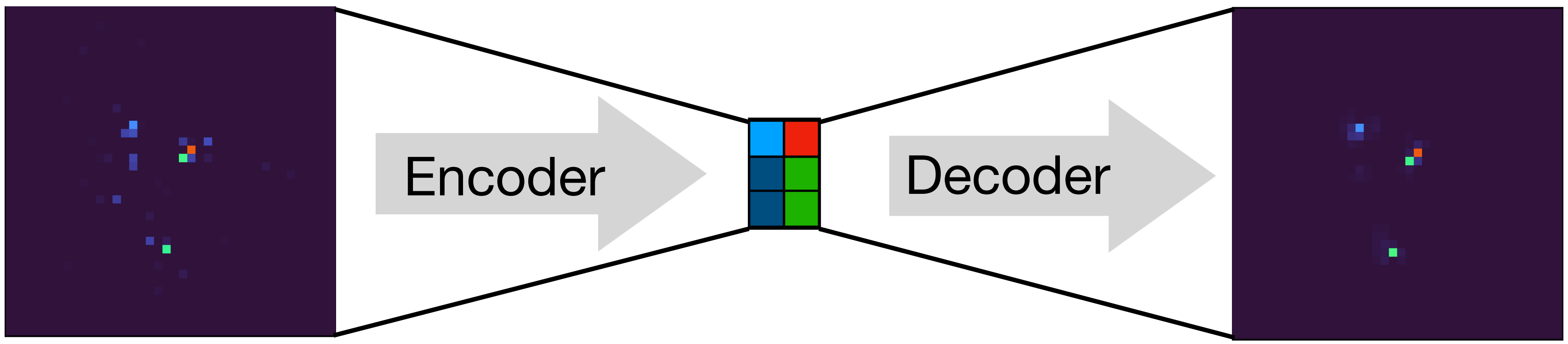
Jet images

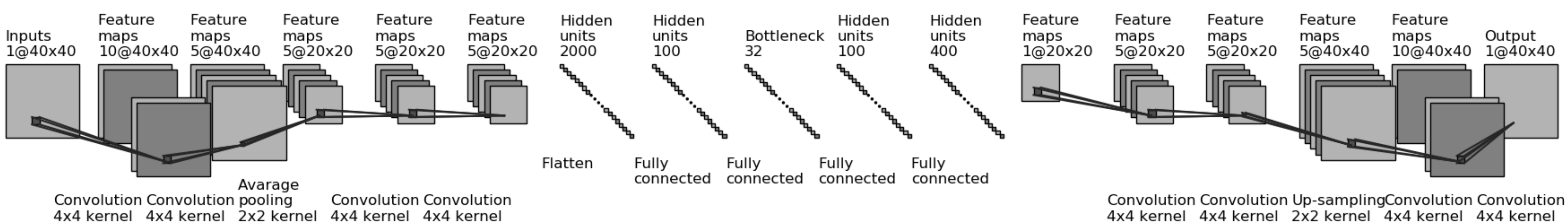
Typical single top-jet | Average top-jet | Average QCD-jet



Kasieczka et al., SciPost Phys. 7, 014 (2019)

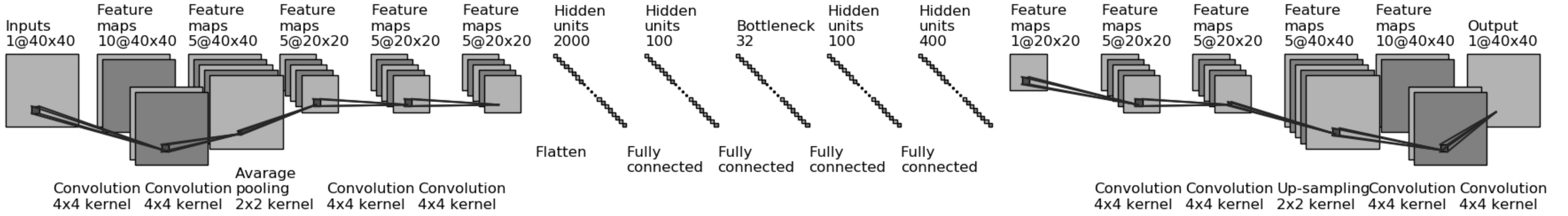
Anomaly detection with autoencoders

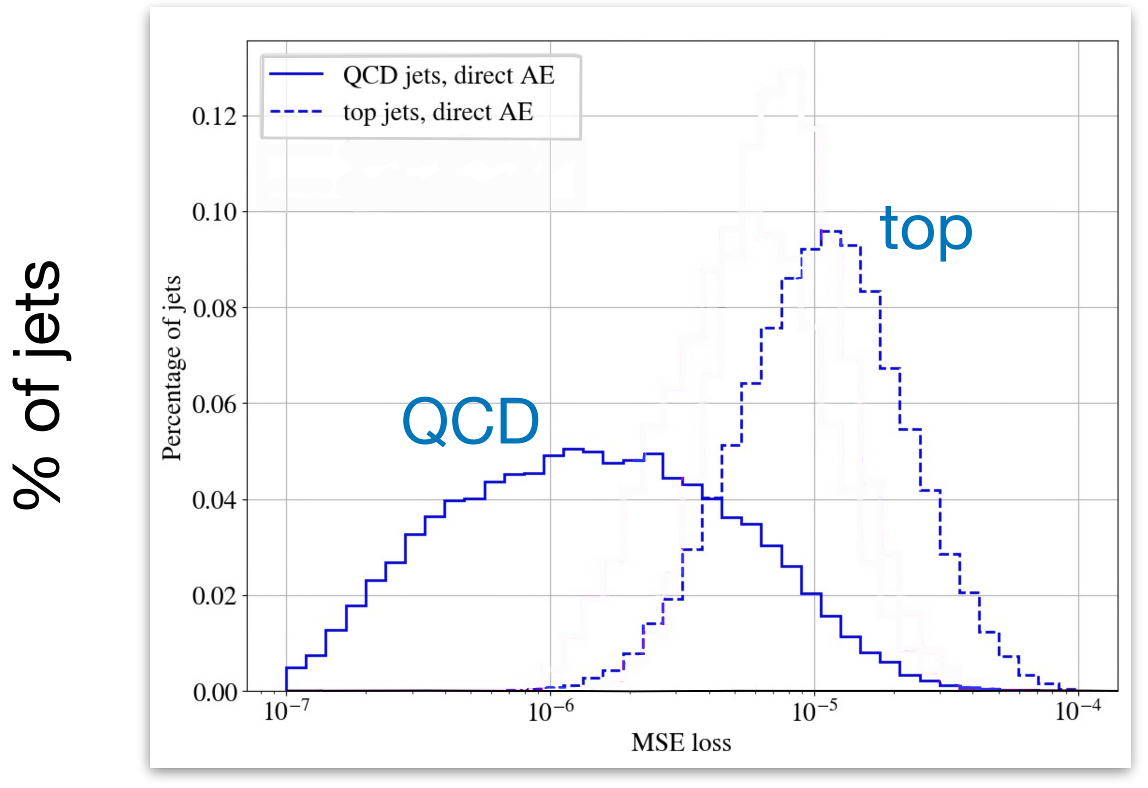




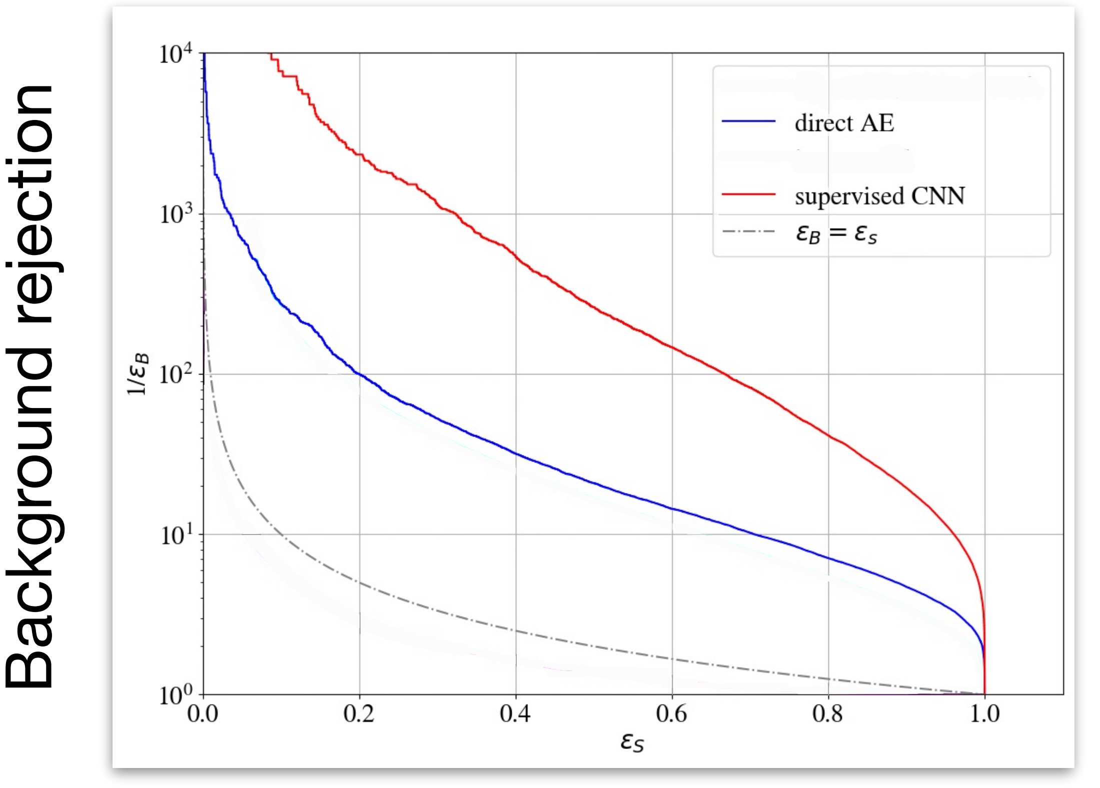
cf Heimel, Kasieczka, Plehn, Thompson, SciPost Phys. 6, 030 (2019); Farina, Nakai, Shih, PRD 101 (2020)

Anomaly detection with autoencoders





MSE loss



Top tagging efficiency