Visualisation and Dimension Reduction

Christophe Ambroise

christophe.ambroise@univ-evry.fr

UEVE, UMR CNRS 8071

February 18, 2025

Evaluation

A Natural Language Processing (NLP) project to be completed in pairs:

- Code (in a Python notebook or R Markdown)
- Presentation (scheduled for April 4, 2025)

Factor Analyser

Factor Analysis

- Using discrete latent variables provides limited summary (clustering)
- An alternative is to use a vector of real-valued latent variables, $\boldsymbol{z} \in \mathbb{R}^L$.
- “Factor analysis (FA) is a statistical method used to describe variability among observed, correlated variables in terms of a potentially lower number of unobserved variables called factors.” Wikipedia quote.
- PCA and FA are related, but not identical.

The model of factor analysis

We consider the observation $\boldsymbol{x} \in \mathbb{R}^D$

$$\boldsymbol{x} = \boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu} + \epsilon$$

where

- the noise $\epsilon \sim \mathcal{N}_D(\mathbf{0}, \boldsymbol{\Psi})$
- the hidden (latent) vector $\boldsymbol{z} \sim \mathcal{N}_L(\mathbf{0}, \boldsymbol{I}_L)$

$$p(\boldsymbol{x}|\boldsymbol{z}, \theta) = \mathcal{N}(\boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu}, \boldsymbol{\Psi})$$

the mean is a linear function of the (hidden) inputs

- $\boldsymbol{W}$ is a $D \times L$ matrix, known as the factor loading matrix,
- $\boldsymbol{\Psi}$ is a $D \times D$ covariance matrix that we take to be diagonal

The special case in which $\boldsymbol{\Psi} = \sigma^2 I$ is called probabilistic principal components analysis or PPCA.

Reminder: Joint and conditional Gaussian distribution (see Murphy chapter 4)

Let us recall that if

- $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_{zz})$ and $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_{xx})$ and $cov(\boldsymbol{z}, \boldsymbol{x}) = \boldsymbol{\Sigma}_{zx}$

then

$$p(\boldsymbol{z}, \boldsymbol{x}) = N(\begin{bmatrix} \boldsymbol{z} \\ \boldsymbol{x} \end{bmatrix} | \begin{bmatrix} \boldsymbol{\mu}_z \\ \boldsymbol{\mu}_x \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{zz} & \boldsymbol{\Sigma}_{zx} \\ \boldsymbol{\Sigma}_{xz} & \boldsymbol{\Sigma}_{zz} \end{bmatrix})$$

and

$$p(\boldsymbol{z}, \boldsymbol{x}) = p(\boldsymbol{z}|\boldsymbol{x})p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_{z|x}, \boldsymbol{\Sigma}_{z|x})\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_{xx})$$

where

- $\boldsymbol{\mu}_{z|x} = \boldsymbol{\mu}_z + \boldsymbol{\Sigma}_{zx}\boldsymbol{\Sigma}_{xx}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_x)$
- $\boldsymbol{\Sigma}_{z|x} = \boldsymbol{\Sigma}_{zz} - \boldsymbol{\Sigma}_{zx}\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\Sigma}_{xz}$

Marginal and posterior distribution

Marginal distribution

$$\boldsymbol{x} \sim \mathcal{N}_D(\boldsymbol{\mu}, \boldsymbol{\Sigma_{xx}} = \boldsymbol{WW^T} + \boldsymbol{\Psi})$$

Posterior distribution

$$\boldsymbol{z}|\boldsymbol{x} \sim \mathcal{N}_L(\boldsymbol{\mu}_{z|x}, \boldsymbol{\Sigma}_{z|x})$$

where

- $\boldsymbol{\Sigma}_{z|x} = (\boldsymbol{I_L} + \boldsymbol{W^T\Psi^{-1}W})^{-1} = S$
- $\boldsymbol{\mu}_{z|x} = \boldsymbol{\Sigma_{z|x}\Sigma_{xx}^{-1}}(\boldsymbol{x} - \boldsymbol{\mu}) = \boldsymbol{SW^T\Psi^{-1}}(\boldsymbol{x} - \boldsymbol{\mu})$

Exercice

Demonstrate the above formulas

Estimation

The mean μ

can be estimated by maximum likelihood

$$\boldsymbol{\mu}_{mle} = \bar{\boldsymbol{x}}$$

W and Ψ

are estimated using an EM algorithm

EM algorithm

Data

- Observed data : $x_{1:n}$
- Missing (or hidden) data : : $z_{1:n}$

Principle

- Starting from θ^0
- At step q
  - → E(xpectation) step: $Q(\theta, \theta^q) = E_{Z_{1:n}|\boldsymbol{x}_{1:n}}[\log P(\boldsymbol{x}_{1:n}, \boldsymbol{z}_{1:n}, \theta)]$
  - → M(aximisation) step: $\theta^{q+1} = argmax_\theta Q(\theta, \theta^q)$

EM for factor analysis

Let us assume that $\boldsymbol{\mu} = 0$ (centering of the $\boldsymbol{x}_i$), the complete log-likelihood is

$$\begin{aligned}\log p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\mu}, \boldsymbol{W}, \boldsymbol{\Psi}) &= \sum_i \log \mathcal{N}_L(z_i; \mathbf{0}, \boldsymbol{I}) + \log \mathcal{N}_D(x_i; \boldsymbol{W}\boldsymbol{z_i}, \boldsymbol{\Psi}) \\ &= -\frac{n}{2}\log|\boldsymbol{I}_L| - \frac{n}{2}Tr(\hat{\boldsymbol{\Sigma}}_{\boldsymbol{zz}}) \\ &\quad -\frac{n}{2}\log|\boldsymbol{\Psi}| - \frac{n}{2}Tr(\hat{\boldsymbol{\Sigma}}_{\boldsymbol{xx}}\boldsymbol{\Psi}^{-1}) + Cst\end{aligned}$$

where

$$\hat{\boldsymbol{\Sigma}}_{\boldsymbol{xx}} = \frac{1}{n}\sum_i (\boldsymbol{x}_i - \boldsymbol{W}\boldsymbol{z_i})(\boldsymbol{x}_i - \boldsymbol{W}\boldsymbol{z_i})^T$$

Exercice

Demonstrate the above formula

E step

The expectation of the complete log-likelihood requires

1. $\mathbb{E}_{z|x}[\boldsymbol{z}_i] = \boldsymbol{S}\boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{\Psi}^{-\mathbf{1}}(\boldsymbol{x}_i - \boldsymbol{\mu})$ where $\boldsymbol{S} = (\boldsymbol{I}_{\boldsymbol{L}} + \boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{\Psi}^{-\mathbf{1}}\boldsymbol{W})^{-\mathbf{1}}$
2. $\mathbb{E}_{z|x}[\boldsymbol{z_i}\boldsymbol{z_i^T}] = E_{z|x}[\boldsymbol{z}_i]E_{z|x}[\boldsymbol{z}_i^T] + \boldsymbol{S}$

M step

Reminders

$$\frac{\partial(b^T a)}{\partial a} = b$$

$$\frac{\partial(a^T A a)}{\partial a} = (A + A^T)a$$

$$\frac{\partial}{\partial A} tr(BA) = B^T$$

$$\frac{\partial}{\partial A} \log |A| = (A^{-1})^T$$

$$tr(ABC) = tr(CAB) = tra(BCA)$$

Thus if x is a vector

$$x^T A x = tr(x^T A x) = tr(A x x^T)$$

M step for $\boldsymbol{\Psi}$

$$\mathbb{E}_{z|x}\left[\frac{\partial L(\boldsymbol{W},\boldsymbol{\Psi})}{\partial \boldsymbol{\Psi}^{-1}}\right] = \mathbb{E}_{z|x}\left[\frac{n}{2}(\boldsymbol{\Psi} - \hat{\boldsymbol{\Sigma}}_{\boldsymbol{xx}})\right] = \frac{n}{2}(\boldsymbol{\Psi} - \mathbb{E}_{z|x}\left[\hat{\boldsymbol{\Sigma}}_{\boldsymbol{xx}}\right]) = 0$$

where

$$\begin{aligned}
\mathbb{E}_{z|x}\left[\hat{\boldsymbol{\Sigma}}_{\boldsymbol{xx}}\right] &= \frac{1}{n}\Big(\sum_i \boldsymbol{x_i}\boldsymbol{x_i^T} + W(\sum_i \mathbb{E}_{z|x}\left[\boldsymbol{z_i}\boldsymbol{z_i^T}\right])W^T - 2W\sum_i \boldsymbol{E}_{\boldsymbol{z|x}}\left[\boldsymbol{z_i}\right]\boldsymbol{x_i^T}\Big) \\
&= \frac{1}{n}\Big(\sum_i \boldsymbol{x_i}\boldsymbol{x_i^T} + W(\sum_i \mathbb{E}_{z|x}\left[\boldsymbol{z_i}\boldsymbol{x_i^T}\right]) - 2W\sum_i \boldsymbol{E}_{\boldsymbol{z|x}}\left[\boldsymbol{z_i}\right]\boldsymbol{x_i^T}\Big) \\
&= \frac{1}{n}\Big(\sum_i \boldsymbol{x_i}\boldsymbol{x_i^T} - W\sum_i \boldsymbol{E}_{\boldsymbol{z|x}}\left[\boldsymbol{z_i}\right]\boldsymbol{x_i^T}\Big)
\end{aligned}$$

M step for $\boldsymbol{W}$

$$\mathbb{E}_{z|x}\left[\frac{\partial L(\boldsymbol{W},\boldsymbol{\Psi})}{\partial \boldsymbol{W}}\right] = \mathbb{E}_{z|x}\left[-\boldsymbol{\Psi}^{-1}\sum_i x_i z_i^T + \boldsymbol{\Psi}^{-1}W\sum_i z_i z_i^T\right] = 0$$

M Step summary

Loading matrix

$$\boldsymbol{W}^{q+1} = \left(\sum_i (\boldsymbol{x}_i - \bar{\boldsymbol{x}}) \mathbb{E}_{z|x}[\boldsymbol{z}_i]^T \right) \left(\sum_i \mathbb{E}_{z|x}[\boldsymbol{z}_i \boldsymbol{z}_i^T] \right)^{-1}$$

Noise covariance matrix

$$\boldsymbol{\Psi}^{q+1} = \frac{1}{N} diag\left\{ \sum_i \boldsymbol{x_i} \boldsymbol{x_i^T} - \boldsymbol{W}^{q+1} \mathbb{E}_{z|x}[\boldsymbol{z}_i] \boldsymbol{x}_i^T \right\}$$

Log-likelihood

The log-likelihood can be computed using the EM decomposition

$$\log P(X; \boldsymbol{\Theta}) = E_{Z_{1:n}|\boldsymbol{x}_{1:n}}[\log P(\boldsymbol{x}_{1:n}, \boldsymbol{z}_{1:n}; \theta)] - E_{Z_{1:n}|\boldsymbol{x}_{1:n}}[\log P(\boldsymbol{z}_{1:n}|\boldsymbol{x}_{1:n}; \theta)]$$

Implementation of the algorithm

Initialisation via a PCA

```r
initialisation.FA<-function(X,L=1){
  # Return W and Psi
  d<-ncol(X)
  Sigmaxx<-var(X)
  W<-eigen(Sigmaxx)$vectors[,1:L]
  if (L==1) W<-cbind(W)
  Psi<-rep(1,d)
  return(list(W=W,Psi=Psi))
}
```

E step

```r
FA.E.step<-function(X,W,Psi){
  # X is assumed to be centered
  # M contain the contionnal expectation of the latent factor
  # S contains the covariance of the latent factor
  L<-ncol(W)
  S <- solve(diag(L) + t(W)%*%diag(1/Psi)%*%W)
  M<- X%*%diag(1/Psi)%*%W%*%S
  return(list(S=S,M=M))
}
```

M Step

```r
FA.M.step<-function(X,S,M,W,Psi){
  n<-nrow(X)
  Psi<-1/n*diag(t(X)%*%X -W%*%t(M)%*%X)
  W<- (t(X)%*%M)%*%solve(n*S+t(M)%*%M)
  return(list(Psi=Psi,W=W))
}
```

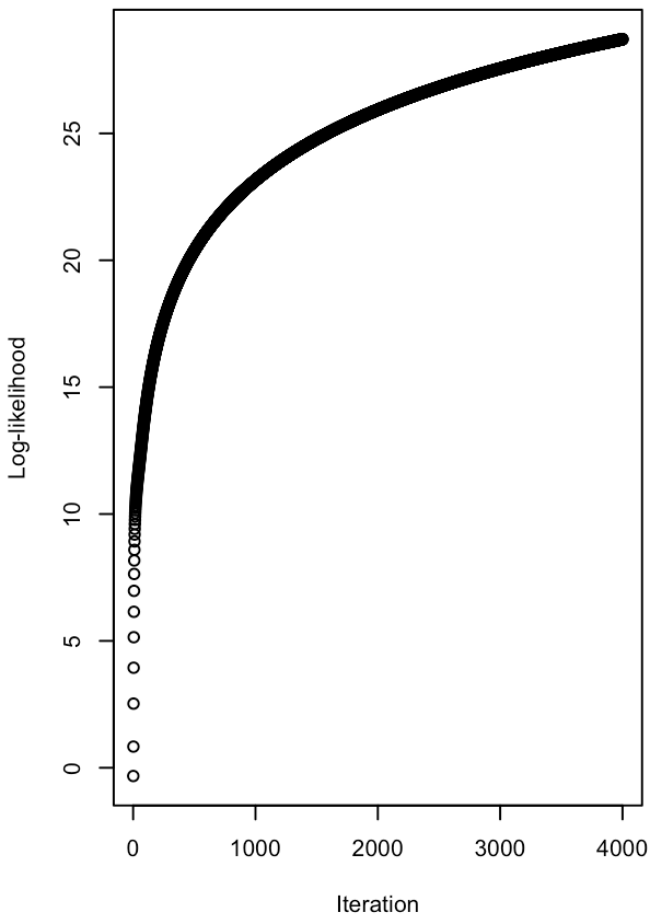
Computation of the criterion

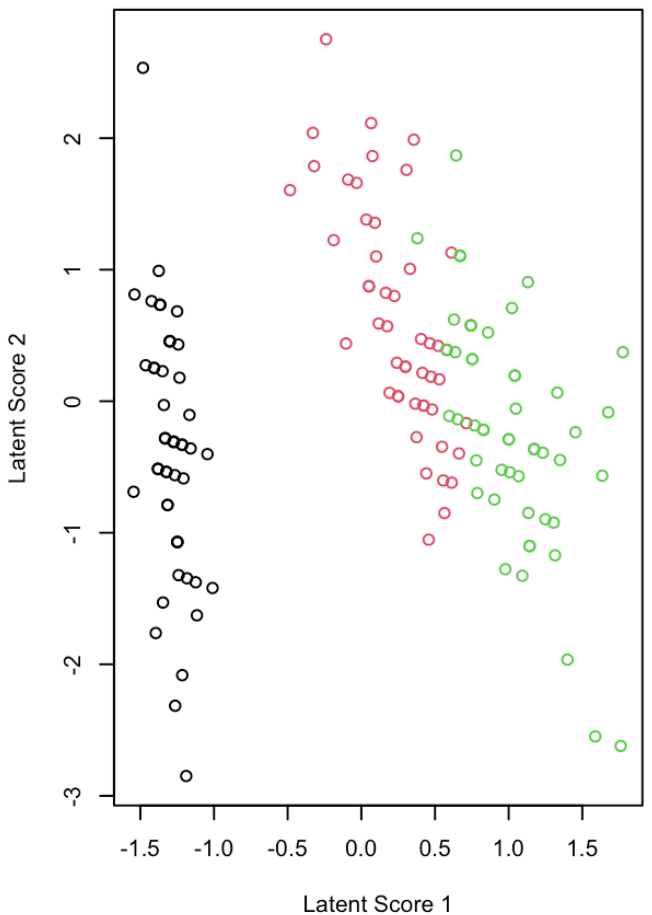
```r
log.likelihood.FA<-function(X,S,M,Psi,W){
  n<-nrow(X)
  Sigmax<-(t(X)%*%X-W%*%t(M)%*%X)
  return(-(sum(diag(S+t(M)%*%M/n))
          +log(det(diag(Psi)))+
            log(det(S))+
            1/n*sum(diag(Sigmax%*%diag(1/Psi)))))
}
```

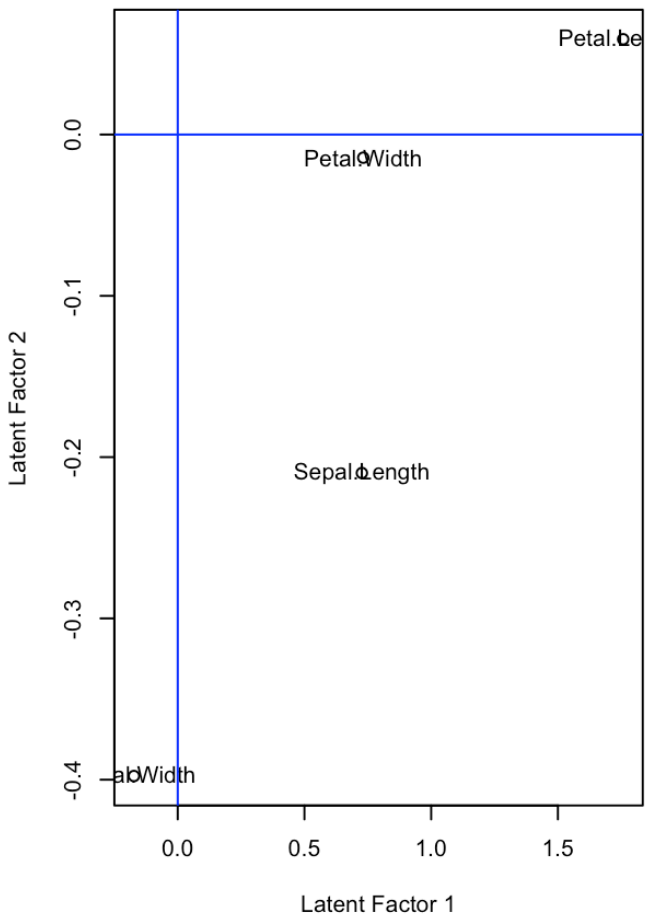
Putting it all together

```r
FA.EM<-function(X,L=1,max.iter=50){
  X<-scale(X,scale=FALSE);mu<-attr(X,"scaled:center")
  log.likelihood<-NULL; init<-initialisation.FA(X,L)
  W<-init$W;  Psi<-init$Psi; criterion<- Inf; iteration<-1;
  log.likelihood[iteration]<--Inf
  while ((criterion>1e-6)&&(iteration<=max.iter)){
    E.step<-FA.E.step(X,W,Psi); E.step$S->S; E.step$M->M
    M.step<-FA.M.step(X,S,M,W,Psi); M.step$Psi->Psi; M.step$W->W
    iteration<-iteration+1
    log.likelihood[iteration]<-log.likelihood.FA(X,S,M,Psi,W)
    criterion<-abs((log.likelihood[iteration] - log.likelihood[iteration-1])/max(log.likelihood[iteration], log.likelihood[iter
  }
  return(list( W=data.frame(W), Psi=Psi,
               M=data.frame(M), S=S,mu=mu,
               log.likelihood= log.likelihood[-1]))
}
```

Example with the Iris







Unidentifiability

- If we consider $\boldsymbol{R}$ an orthogonal rotation matrix such that

$$\boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}$$

It appears that $\tilde{\boldsymbol{W}} = \boldsymbol{W}\boldsymbol{R}$ produces the same log-likelihood.

- $\boldsymbol{W}$ cannot be uniquely identified.

Possible rotations

1. Forcing $\boldsymbol{W}$ to be orthogonal with colmuns ordered by deacresing variance
2. Forcing $\boldsymbol{W}$ to be lower triangular (problem of founder variables)
3. Choosing an informative rotation matrix. For example the varimax rotation.
4. ...

Varimax

Varimax rotation maximizes the sum of the variance of the squared correlations between variables and factors

$$R_{\text{VARIMAX}} = \arg\max_{R} \left(\frac{1}{p} \sum_{j=1}^{k} \sum_{i=1}^{p} (WR)_{ij}^{4} - \sum_{j=1}^{k} \left(\frac{1}{p} \sum_{i=1}^{p} (WR)_{ij}^{2} \right)^{2} \right)$$

This results in high factor loadings for a small number of variables and low factor loadings for the rest.

Varimax

```r
W.FA<-FA.result$W
W.Varimax<-varimax(as.matrix(FA.result$W))$loadings
print(W.Varimax)
```

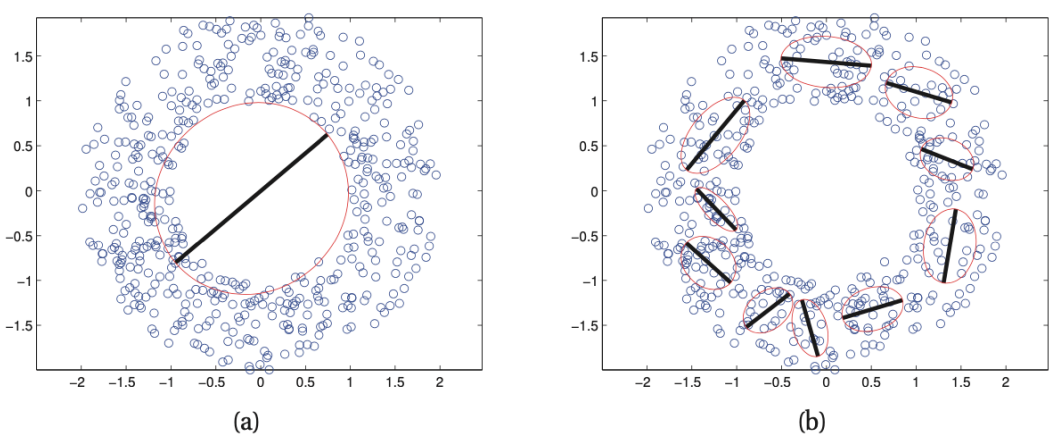
```
Loadings:
             Latent Factor 1 Latent Factor 2
Sepal.Length  0.756
Sepal.Width                  -0.429
Petal.Length  1.683           0.509
Petal.Width   0.711           0.174

               Latent Factor 1 Latent Factor 2
SS loadings              3.916           0.473
Proportion Var           0.979           0.118
Cumulative Var           0.979           1.097
```

Mixture of factor analysers

- Factor analyses is a way to estimate a variance matrix with few parameters
- This property can be used in the context of Gaussian mixture model assuming the following parameterization for component densities:

$$p(\boldsymbol{x}_i|\boldsymbol{z}_i, q_i = k) = \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_k + \boldsymbol{W}_k\boldsymbol{z}_i, \boldsymbol{\Psi})$$

  where $\boldsymbol{k}$ is the component number and $\boldsymbol{W}_k$ is a loading matrix defining the relation between the observation $\boldsymbol{x}_i$ and the latent vector $\boldsymbol{z}_i$
- This approach is simular to the Banfield-Raftery idea of decomposing the component variance matrix $\boldsymbol{k}$ in volume, form et direction.



Mixture of 1d PPCAs with 1 and 10 components (from Murphy Chapter 12)

Relation to principal component analysis

Assumption

If

- $\boldsymbol{\Psi} = \sigma^2 \boldsymbol{I}$
- $\boldsymbol{W}$ is orthogonal

and

- $\sigma^2 \to 0$

Then

Tipping, M. and C. Bishop (1999, Probabilistic principal component analysis. J. of Royal Stat. Soc. Series B 21(3), 611–622) showed that FA is equivalent to PCA

Criterion

$$J(\boldsymbol{W}, \boldsymbol{Z}) = \|\boldsymbol{X} - \boldsymbol{Z}\boldsymbol{W}^T\|_F^2$$

where $\boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{I}$

A Constrained EM Algorithm for PCA (from Ahn, J.-H. and J.-H. Oh, 2003)

```
upper<-function(A){A[lower.tri(A,diag=FALSE)]<-0;return(A)}
lower<-function(A){A[upper.tri(A,diag=FALSE)]<-0;return(A)}
PCA.EM<-function(X,q=2){
  p<-ncol(X); n<-nrow(X)
  W<-diag(p)[,1:q]; M<-X%*%W # Initialisation
  Jold<-0; J<-1; iteration<-0; Error<-NULL
  while ((abs(J - Jold)>1e-3)){
    Jold<-sum((X-M%*%t(W))^2)
    S <- solve(upper(t(W)%*%W)); M<- X%*%W%*%S # E-step
    W<- (t(X)%*%M)%*%solve(lower(n*S+t(M)%*%M))# M-step
    W<-apply(W,2,function(x){x/sqrt(sum(x^2))})#orthogonalisation
    J<-sum((X-M%*%t(W))^2); Error[iteration<-iteration+1]<-J
  }
  return(list(W=data.frame(W),M=data.frame(M),Error=Error))}
```

Independent Component Analysis

Independent Component analysis (Wikipedia)

Independent component analysis attempts to decompose a multivariate signal into independent non-Gaussian signals.

Cocktail party problem

Underlying speech signals are separated from a sample data consisting of people talking simultaneously in a room.

That the ICA separation of mixed signals gives very good results is based on two assumptions

Two assumptions:

- The source signals are independent of each other.
- The values in each source signal have non-Gaussian distributions.

ICA Historical context (French Wikipedia)

Blind source separation

The first formulation was carried out in 1984 by Jeanny Hérault and Bernard Ans, two researchers in neuroscience and signal processing, to model in the form of a neuromimetic network self-adaptive encoding and decoding of movement in humans.

France and Finland

- The French signal processing community adopted a statistical formalism
- While Finnish researchers aimed to extend principal component analysis by means of a connectionist formalism (1985)

Formalisation

- A first formalism of the blind source separation problem, as well as an algorithm making it possible to obtain a solution, was proposed by C. Jutten and J. Hérault in 1991.
- Mathematical formalization in the simplest case (linear mixing snapshot) was carried out in 1994 by P. Comon

ICA Model

Let $\boldsymbol{x}_t \in \mathbb{R}^D$ be the observed signal at the sensors at "time" $\boldsymbol{t}$, and $\boldsymbol{z}_t \in \mathbb{R}^L$ be the vector of source signals:

$$\boldsymbol{x}_t = \boldsymbol{W}\boldsymbol{z}_t + \boldsymbol{\epsilon}_t$$

- $\boldsymbol{W}$ is an $D \times L$ matrix,
- $\boldsymbol{\epsilon}_t \sim \mathcal{N}(0, \boldsymbol{\Psi})$.

The model is identical to factor analysis (or PCA if there is no noise, except we don't in general require orthogonality of W).

However, we will use a **different prior** for $p(\boldsymbol{z}_t)$.

In FA, we assume each source is independent, and has a Gaussian distribution.

Relax this Gaussian assumption on latent variables (sources)

$$p(\boldsymbol{z}_t) = \prod_j p_j(z_{tj}).$$

Additionnal assumptions

- Without loss of generality the variance of the source distributions is contrained to unity
- W is assumed square and hence invertible.

Maximum likelihood estimation of ICA

If the data is centered and whitened, we have

$$\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^t] = I = \boldsymbol{W}\mathbb{E}[\boldsymbol{z}\boldsymbol{z}^T]\boldsymbol{W}^T = \boldsymbol{W}\boldsymbol{W}^T$$

also have

Hence we see that $\boldsymbol{W}$ must be orthogonal. This reduces the number of parameters we have to estimate from D^2 to $D(D-1)/2$.

Recognition weights/Generative weights

Let $\boldsymbol{V} = \boldsymbol{W}^{-1}$ these are often called the recognition weights, as opposed to $\boldsymbol{W}$, which are the generative weights

Log-likelihood

Since $\boldsymbol{x} = \boldsymbol{W}\boldsymbol{z}$, we have

$$p_x(\boldsymbol{W}\boldsymbol{z}_t) = p_z(\boldsymbol{z}_t)|det(\boldsymbol{W}^{-1})| = p_z(\boldsymbol{V}\boldsymbol{x}_t)|det(\boldsymbol{V})|$$

Hence assuming T iid samples:

$$L(\boldsymbol{V}) = \frac{1}{T}\log p(\mathcal{D}|\boldsymbol{V}) = \underbrace{\log|det(\boldsymbol{V})|}_{0} + \frac{1}{T}\sum_{jt}\log p_j(v_j^T \boldsymbol{x}_t)$$

since $\boldsymbol{V}$ is orthogonal.

Estimation via Gradient Ascent

Let us define $h_j = \boldsymbol{v}_j^T \boldsymbol{x}$,

$$g_j(h_j) = \frac{\partial \log p_j(h_j)}{\partial h_j},$$

$$\frac{\partial L(\boldsymbol{V})}{\partial V_{ij}} = W_{ji} + x_i g_j(h_j).$$

where the datapoint $\boldsymbol{x}^T = (x_i)_{i=1 \dots D}$.

Repeat for each datapoint $\boldsymbol{x}$:

1. Put $\boldsymbol{x}$ through a linear mapping:

$$\boldsymbol{h} = \boldsymbol{V}\boldsymbol{x}$$

.

2. Put $\boldsymbol{h}$ through a nonlinear map:

$$g_j = g_j(h_j)$$

, where a popular choice is $g() = -tanh()$.

3. Adjust the weights in accordance with

$$\nabla \boldsymbol{V} \propto [\boldsymbol{V}^T]^{-1} + \boldsymbol{x}\boldsymbol{g}^T.$$

- matrix inversion results in a slow algorithm

Fast ICA (for one latent factor $\boldsymbol{v}$)

Let consider $G(z) = -\log p(z)$, $g(z) = G'(z)$.

- $L(\boldsymbol{v}) = \mathbb{E}[G(\boldsymbol{v}^T\boldsymbol{x})] + \lambda(1 - \boldsymbol{v}^T\boldsymbol{v})$, the theoritical objective function (**to be minimized**)
- $\nabla L(\boldsymbol{v}) = \mathbb{E}[\boldsymbol{x}g(\boldsymbol{v}^T\boldsymbol{x})] - 2\lambda\boldsymbol{v}$, the gradient
- $H(\boldsymbol{v}) = \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T g'(\boldsymbol{v}^T\boldsymbol{x})] - 2\lambda I$, the hessian matrix

Let us make the approximation

$$\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T g'(\boldsymbol{v}^T\boldsymbol{x})] = \underbrace{\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]}_{I}\mathbb{E}[g'(\boldsymbol{v}^T\boldsymbol{x})] = \mathbb{E}[g'(\boldsymbol{v}^T\boldsymbol{x})]$$

This makes the Hessian very easy to invert, giving rise to the following Newton update:

$$\boldsymbol{v}^* \triangleq v - H(\boldsymbol{v})^{-1}\nabla L(\boldsymbol{v}) = \boldsymbol{v} - \frac{\mathbb{E}[\boldsymbol{x}g(\boldsymbol{v}^T\boldsymbol{x})] - 2\lambda\boldsymbol{v}}{\mathbb{E}[g'(\boldsymbol{v}^T\boldsymbol{x})] - 2\lambda}$$

Which can be expressed as

$$\boldsymbol{v} := E[\boldsymbol{x}g(\boldsymbol{v}^T\boldsymbol{x})] - E[g'(\boldsymbol{v}^T\boldsymbol{x})]\boldsymbol{v}$$

(In practice, the expectations can be replaced by Monte Carlo estimates from the training set, which gives an efficient online learning algorithm.)

After performing this update, one should project back onto the constraint surface using

$$\boldsymbol{v} := \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|}$$

Fast ICA for $L > 1$

Centering and Withining is assumed

For $L > 1$ the process is iterated for all $\boldsymbol{v}_j$ with orthogonalisation

Fast ICA for $L > 1$

- Intitilisation of $\boldsymbol{V}$
- **for** j in 1 to L:
  - → **while** $\boldsymbol{v}_j$ changes
    - Newton update : $\boldsymbol{v}_j := E[\boldsymbol{x} g(\boldsymbol{v}_j^T \boldsymbol{x})] - E[g'(\boldsymbol{v}_j^T \boldsymbol{x})]\boldsymbol{v}_j$
    - Gram-Schmidt orthogonalisation : $\boldsymbol{v}_j := \boldsymbol{v}_j - \sum_{k=1}^{j-1} (\boldsymbol{v}_j^T \boldsymbol{v}_k)\boldsymbol{v}_k$
    - Normalisation: $\boldsymbol{v}_j := \frac{\boldsymbol{v}_j}{\|\boldsymbol{v}_j\|}$
- Output :$\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{V}$

Notice that the expectation can be approximated with monte-carlo (chosing for example 1 datapoint...)

Non-Gaussianity

For non-Gaussianity, FastICA relies on a nonquadratic nonlinear function $f(u)$, its first derivative $g(u)$, and its second derivative $g'(u)$.

Classical cost

- $G(u) = \log \cosh(u)$ which gives $g(u) = \tanh(u)$
- $G(u) = -\exp(-u^2/2)$ which gives $g(u) = u \exp(-u^2/2)$
- $G(u) = u^4/4)$ which gives $g(u) = u^3$

Neural networks and unsupervised learning

Modeling of a neuron

The first modeling of the neuron was suggested in the 1940s by Mac Culloch and Pitts. It was a unit which according to several signals transmitted a binary response.

In general, a formal neuron has

- dendrites which receive the input signal and
- an axon which transmits the output signal

The input signal

$\boldsymbol{x}$ is a vector belonging most often to $\mathbb{R}^d$ or $\{0,1\}^d$.

Dendrites

are characterized by a weight vector $\boldsymbol{w}$

The output

is a function of $\boldsymbol{x}$ and $\boldsymbol{w}$, which is the composition of an input function, $h(\boldsymbol{x}, \boldsymbol{w})$ and an output (or activation) function, $f(h)$

The neuron as a funtion

Most of the time the input function is a simple dot product:

$$h(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{x}^T \boldsymbol{w}$$

The activation functions are diverse but belong to large families (radial bases, sigmoid functions ...).

A typical activation function is for example:

$$f(\boldsymbol{x}, \boldsymbol{w}) = \eta \cdot \frac{\exp\{(\boldsymbol{x}, \boldsymbol{w}\} - 1}{\exp\ (\boldsymbol{x}, \boldsymbol{w})\ + 1}.$$

The functions which have this appearance are said to be sigmoid

Neural networks

- Neurons can be connected to each other and then form a network.
- Learning consists of adjusting the free network parameters according to the desired goal, that is, to calculate the values of the weight vectors as a function of the inputs.

Two layers networks

- with one input layer transmitting the input vector to the second layer neurons
- with one output layer compressing the information of the first layer with linear activation function
- allows to rewrite k-means and PCA with online learning (gradient descent)

Hebbian Learning (Hebb 1949)

Principle

An increase of synaptic strength between an input and an output neuron may be related to the firing rates of the input and output [Hebb, 1949].

Practical implementation

As a result, synaptic strengths will increase fastest between pairs of neurons whose responses are correlated, and the resulting increase in synaptic strength will lead to a further increase in the correlation.

$$\Delta \mathbf{w} \;=\; \eta\, y(\mathbf{x})\mathbf{x},$$

or in scalar form with implicit n-dependence,

$$w_i(n+1) \;=\; w_i(n) + \eta\, y(\mathbf{x}) x_i$$

Increasing the correlation in this manner may lead to a useful pattern of synaptic strengths over a population of neurons.

Stochastic Gradient Descent (from Wikipedia)

Statistical estimation and machine learning consider the problem of minimizing an objective function that has the form of a sum:

$$Q(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} Q_i(\boldsymbol{w}),$$

where the parameter $\boldsymbol{w}$ that minimizes $Q(\boldsymbol{w})$ is to be estimated.

Each summand function Q_i is typically associated with the $i-th$ observation in the data set (used for training).

Sum-minimization problems arise:

- in least squares and in maximum-likelihood estimation,
- empirical risk minimization.

When used to minimize the above function, a standard (or “batch”) gradient descent method would perform the following iterations:

$$\boldsymbol{w} := \boldsymbol{w} - \eta \nabla Q(\boldsymbol{w}) = \boldsymbol{w} - \frac{\eta}{n} \sum_{i=1}^{n} \nabla Q_i(\boldsymbol{w}),$$

where η is a step size (sometimes called the learning rate in machine learning).

Iterative method

Fluctuations in the total objective function as gradient steps with respect to mini-batches are taken.

In stochastic (or “on-line”) gradient descent, the true gradient is approximated by a gradient at a single example:

$$\boldsymbol{w} := \boldsymbol{w} - \eta \nabla Q_i(\boldsymbol{w}).$$

- As the algorithm sweeps through the training set, it performs the above update for each training example.
- Several passes can be made over the training set until the algorithm converges.
- If this is done, the data can be shuffled for each pass to prevent cycles.
- Typical implementations may use **an adaptive learning rate** so that the algorithm converges.

Stochastic Gradient in pseudocode

1. Choose an initial vector of parameters $\boldsymbol{w}$ and learning rate η
2. Repeat until an approximate minimum is obtained:
    a. Randomly shuffle examples in the training set.
    b. For $i = 1, 2, \cdots, n$ do: $\boldsymbol{w} := \boldsymbol{w} - \eta \nabla Q_i(\boldsymbol{w})$.

Adaptative learning rate

A distinction exists between constant gain algorithms,

$$\eta_n \geq 0, \quad lim_{n\to\infty}\eta_n = \eta > 0$$

and decreasing gain algorithms,

$$\sum_{n=0}^{\infty}\eta_n = \infty, \quad \sum_{n=0}^{\infty}\eta_n^2 < \infty.$$

The first are dedicated to the estimation of parameters changing slowly over time and the second to the estimation of stable parameters.

K-means and Winner take all

is a computational principle applied in computational models of neural networks by which neurons in a layer compete with each other for activation.

The simplest form of competitive learning modifies only the weight vector of “the best” neuron at every stage of learning.

In fact, with each presentation of a input (a vector of the training set), two steps are performed:

1. choose the best neuron, i.e. the one that shows the most important output
2. modify the weight vector of this neuron.

When the activation function is increasing (which is not true for the functions with radial basis), the winning neuron is the one that produces the greatest value of function entry.

If we consider a dot product as an input function, the weight vector of the winner, i^*, checks:

$$\forall i,\ (\boldsymbol{w}_{i^*} \cdot \boldsymbol{x}) \geq (\boldsymbol{w}_i \cdot \boldsymbol{x}).$$

And if *the weight vectors are normalized*, the winner is the neuron that has the weight vector, closest to the input $\boldsymbol{x}$, in the sense of the Euclidean distance.

The coordinates of the winner's weight vector are updated using a rule of the type following :

$$\boldsymbol{w}_{i*}(t+1) = \boldsymbol{w}_{i^*}(t) + \eta(t) \cdot (\boldsymbol{x} - \boldsymbol{w}_{i^*}(t)),\ \eta(t) \leq 1$$

where $\eta(t)$ is the training step at iteration t.

Stochastic Gradient for Kmeans (online Kmeans)

The criterion to be optimized can be written as

$$Q(\boldsymbol{w}_1, \cdots, \boldsymbol{w}_K) = \frac{1}{2n} \sum\nolimits_i \sum_k I_{(k=\arg min_\ell \|\boldsymbol{x}_i - \boldsymbol{w}_\ell\|^2)} \left\|\boldsymbol{x}_i - \boldsymbol{w}_k\right\|^2$$

1. Choose an initial vector of parameters $\boldsymbol{w}$ and learning rate η
2. Repeat until an approximate minimum is obtained:
    a. Randomly shuffle examples in the training set.
    b. For $i = 1, 2, \cdots, n$ do:
        1. For $k = 1, 2, \cdots, K$
        - $\nabla Q_i(\boldsymbol{w}_k) = -\frac{1}{n} I_{(k=\arg min_\ell \|\boldsymbol{x}_i - \boldsymbol{w}_\ell\|^2)} (\boldsymbol{x}_i - \boldsymbol{w}_k)$
        - $\boldsymbol{w}_k := \boldsymbol{w}_k - \eta \nabla Q_i(\boldsymbol{w}_k)$.

Kmeans implementation

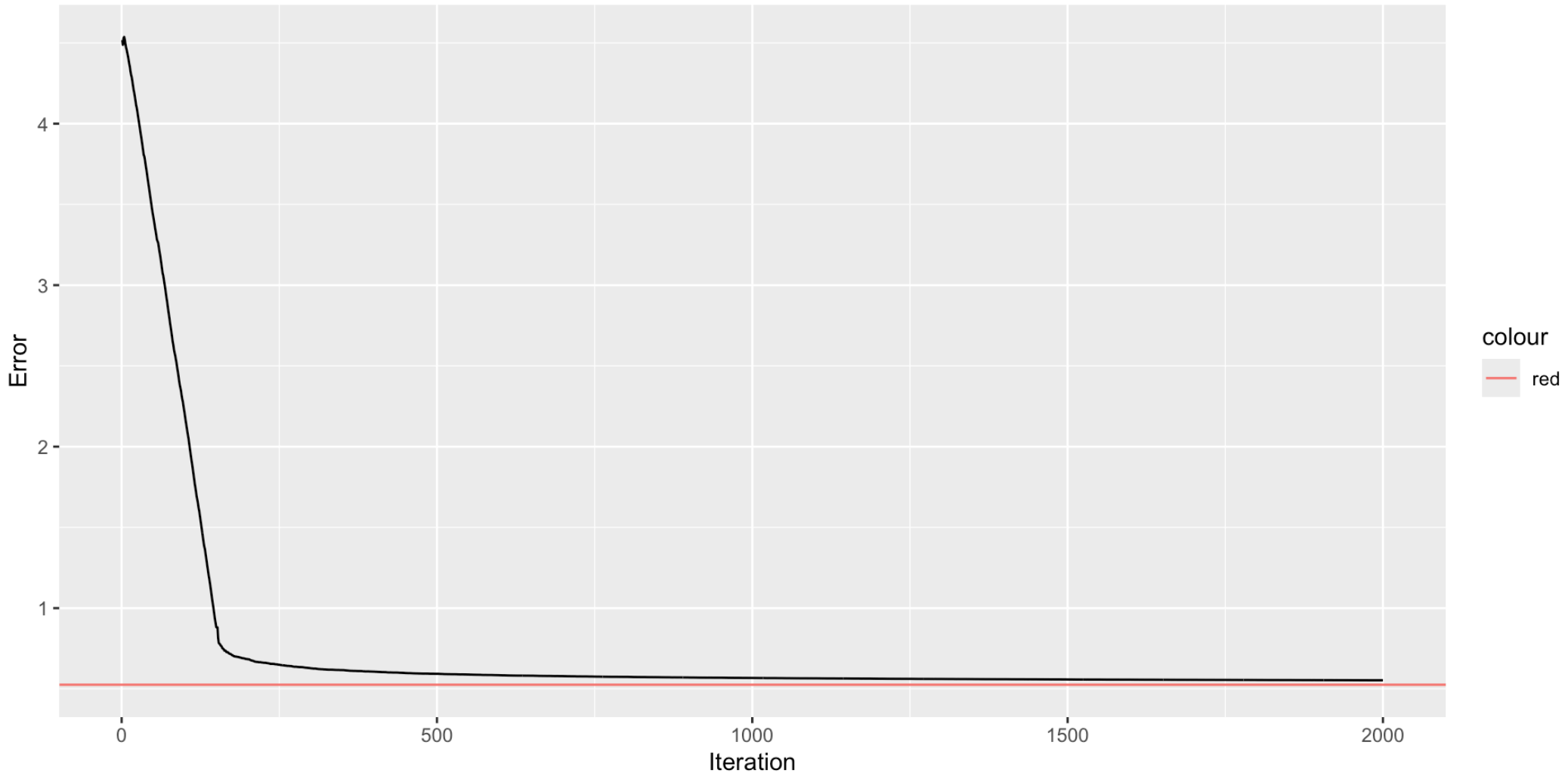
```r
kmeans.winner.take.all<-function(X,K=2,max.iteration=2000){
  p<-ncol(X);  n<-nrow(X);shuffling<-sample(1:n,n)
  X<-X[shuffling,];  W<-X[sample(1:n,K),]
  Q<-rep(0,max.iteration); cluster<-rep(0,n)
  distances<-rep(sum(diag(var(X)))*(n-1)/n,n)
  for (i in 1:max.iteration){
    x<-cbind(X[(i-1)%%n + 1,])
    distances.x.to.W<-sum(x^2)-2*as.matrix(x)%*%t(W)+ colSums(t(W^2))
    winner.index<-which.min(distances.x.to.W)
    W[winner.index,]<-W[winner.index,] + 1/i*(x-W[winner.index,])
    cluster[(i-1)%%n + 1]<-winner.index
    distances[(i-1)%%n + 1]<-distances.x.to.W[winner.index]
    Q[i]<-mean(distances) }
  return(list(W=W,Q=Q,cluster=cluster[order(shuffling)]))
}
```

One line kmeans example with Fisher iris

```
data(iris)
X<-iris[,1:4]
set.seed(1)
kmeans.winner.take.all(X,3)->res
table(res$cluster,iris$Species)
```

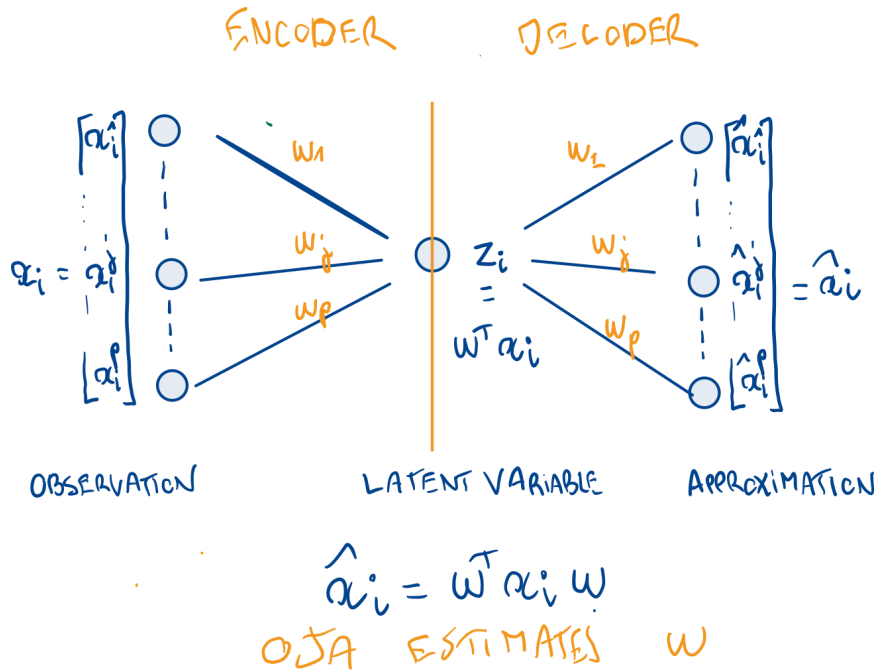
```
  setosa versicolor virginica
1     50          0         0
2      0         45        11
3      0          5        39
```

One line kmeans example with Fisher iris



PCA and Oja's rule

Consider a linear neuron with output $z = \boldsymbol{w}^T \boldsymbol{x}$ that returns a linear combination of its inputs $\boldsymbol{x}$ using presynaptic weights $\boldsymbol{w}$.



PCA according OJA

Oja's rule defines the change in presynaptic weights $\boldsymbol{w}$ given the output response y of a neuron to its inputs $\boldsymbol{x}$ to be

$$\boldsymbol{w} := \boldsymbol{w} - \eta z(\boldsymbol{x} - z\boldsymbol{w})$$

Stochastic Gradient PCA and Oja's rule

The criterion to be optimized can be written as

$$Q(\boldsymbol{w}) = \frac{1}{2n}\sum_i \|\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i\|^2 = \frac{1}{2n}\sum_i \|\boldsymbol{x}_i - y_i\boldsymbol{w}\|^2 = \frac{1}{2n}\sum_i \|\boldsymbol{x}_i - \boldsymbol{w}^T\boldsymbol{x}_i\boldsymbol{w}\|^2$$

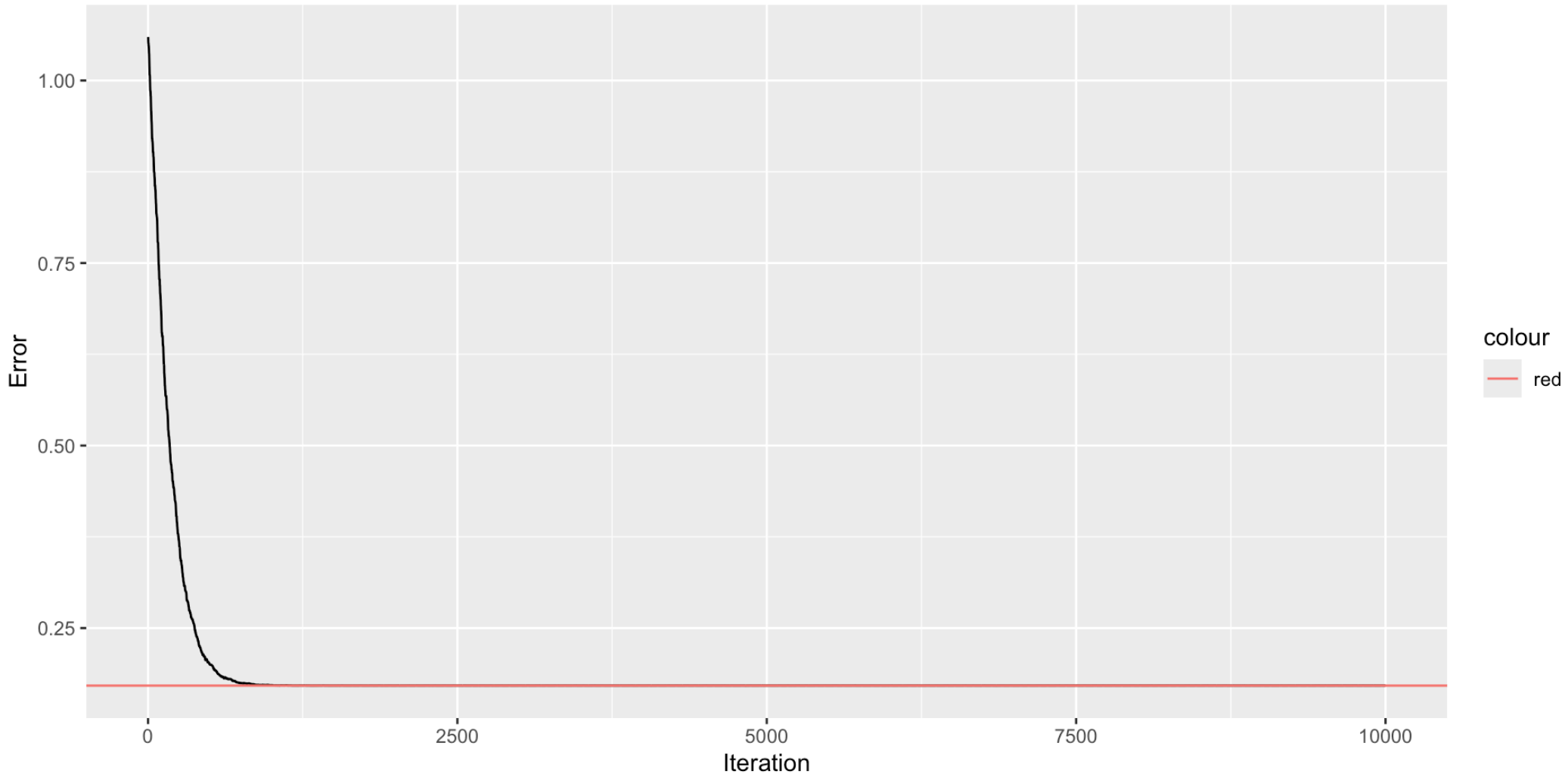
where $\|\boldsymbol{w}\|^2 = 1$.

1. Choose an initial vector of parameters $\boldsymbol{w}$ and learning rate η
2. Repeat until an approximate minimum is obtained:
    a. Randomly shuffle examples in the training set.
    b. For $i = 1, 2, \cdots, n$ do:
        - $\nabla Q_i(\boldsymbol{w}) = y_i(\boldsymbol{x}_i - y_i\boldsymbol{w})$
        - $\boldsymbol{w} := \boldsymbol{w} - \eta\nabla Q_i(\boldsymbol{w})$.

Oja's rule implementation

```r
Oja.rule<-function(X,max.iteration=10000,eta=0.001){
  p<-ncol(X);  n<-nrow(X)
  w<-rbind(rep(1,p)); w<-w/(sqrt(sum(w^2)))
  Q<-rep(0,max.iteration)
  for (i in 1:max.iteration){
    Q[i]<- 1/(2*n) *sum((X - X%*%t(w)%*%w)^2)
    x<-X[(i-1)%%n + 1,]
    y<-sum(w*x)
    w<-w + eta*y*(x-w*y)}
  return(list(w=w,Q=Q))
}
```

Oja's rule example with Fisher iris



Variational auto-encoder

Variational autoencoder ideas

The original papers

- Rezende, D. J., Mohamed, S., & Wierstra, D. (2014, June). Stochastic backpropagation and approximate inference in deep generative models. In International conference on machine learning (pp. 1278-1286). PMLR.
- Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114.

Main reference

- Diederik P. Kingma and Max Welling (2019), “An Introduction to Variational Autoencoders”, Foundations and Trends R in Machine Learning:

What is does

- generate realistic samples of data,
- allow for accurate imputations of missing data,
- high-dimensional data visualisation
- Clustering

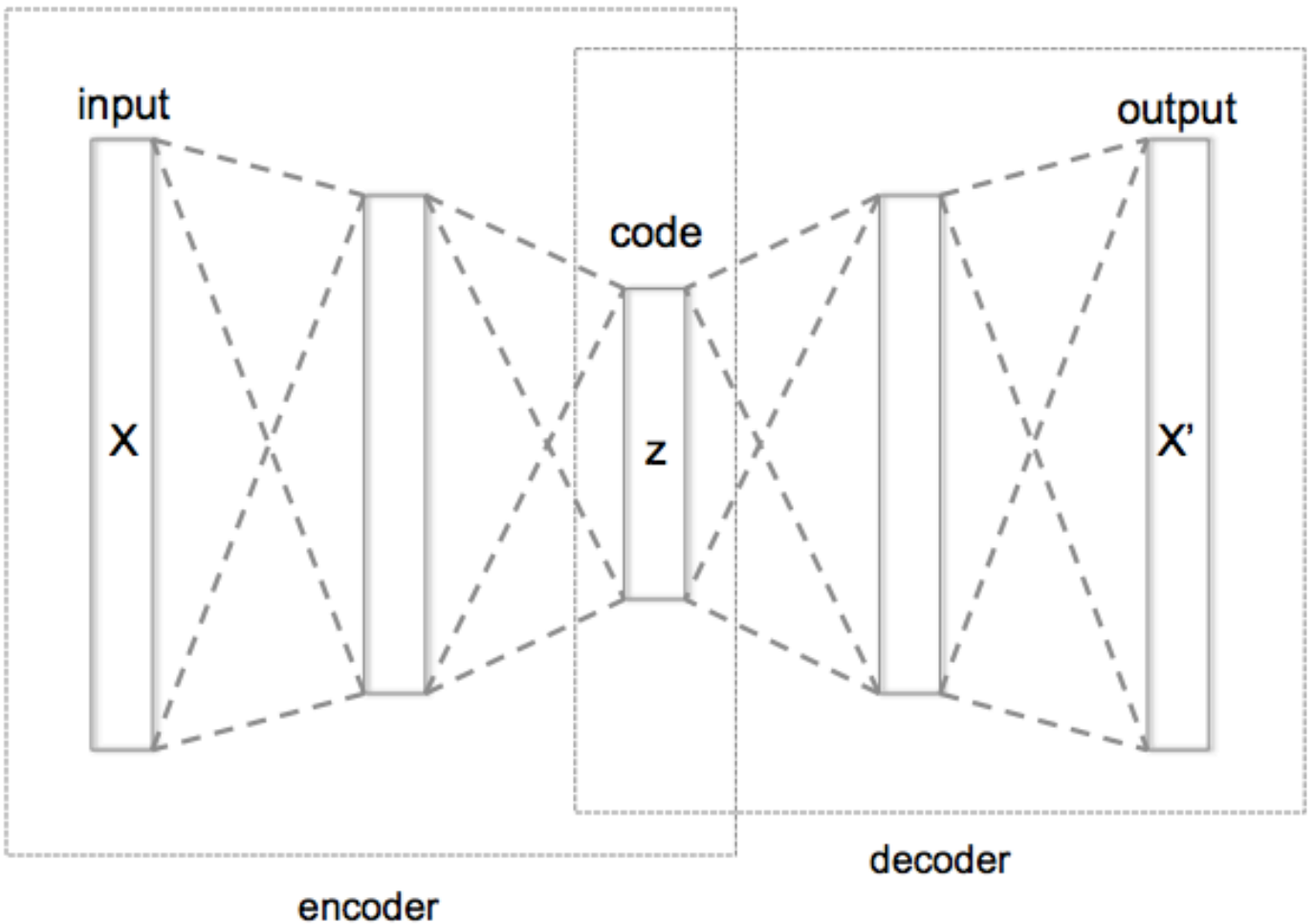
How it works

Latent variables models which marry ideas from

- approximate Bayesian inference
  - → ELBO (Evidence Lower BOund)
  - → Reparametrization
- deep neural networks
  - → Stochastic Gradient Descent
  - → Retropropagation of the Gradient

to represent an approximate posterior distribution through variational lower bound optimization

Auto-encoder Structure



What is a VAE ?

Coupling of 2 parametric models

VAE is a latent variable vector $\boldsymbol{z}$ and an observation $\boldsymbol{x} \in \mathbb{R}^D$

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z}$$

- encoder g (recognition model): $p(\boldsymbol{z}|\boldsymbol{x})$, which is approximated by $q_\Phi(\boldsymbol{z}|\boldsymbol{x})$
- decoder $h \approx g^{-1}$ (generative model): $p_\Theta(\boldsymbol{x}|\boldsymbol{z})$
- encoder and decoder could be neural networks

Optimization of ELBO via Stochastic Gradient Ascent

- The VAE ELBO approximates the likelihood of a **latent variable** model
- The Gradient computation uses the re-parametrization trick
- Each step of the gradient ascent augment the ELBO as an **EM iteration**

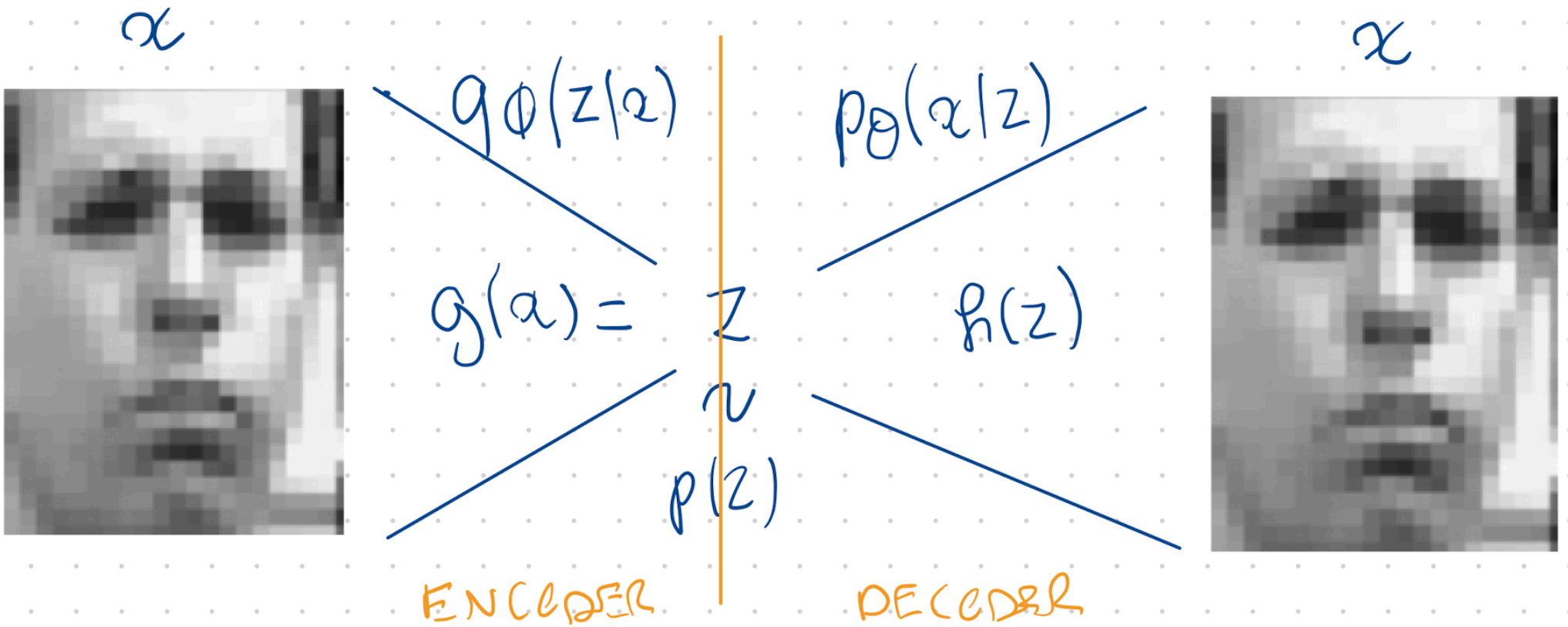
Example of use

Data

Left batch of original training set - right : random generation of images

Example of use

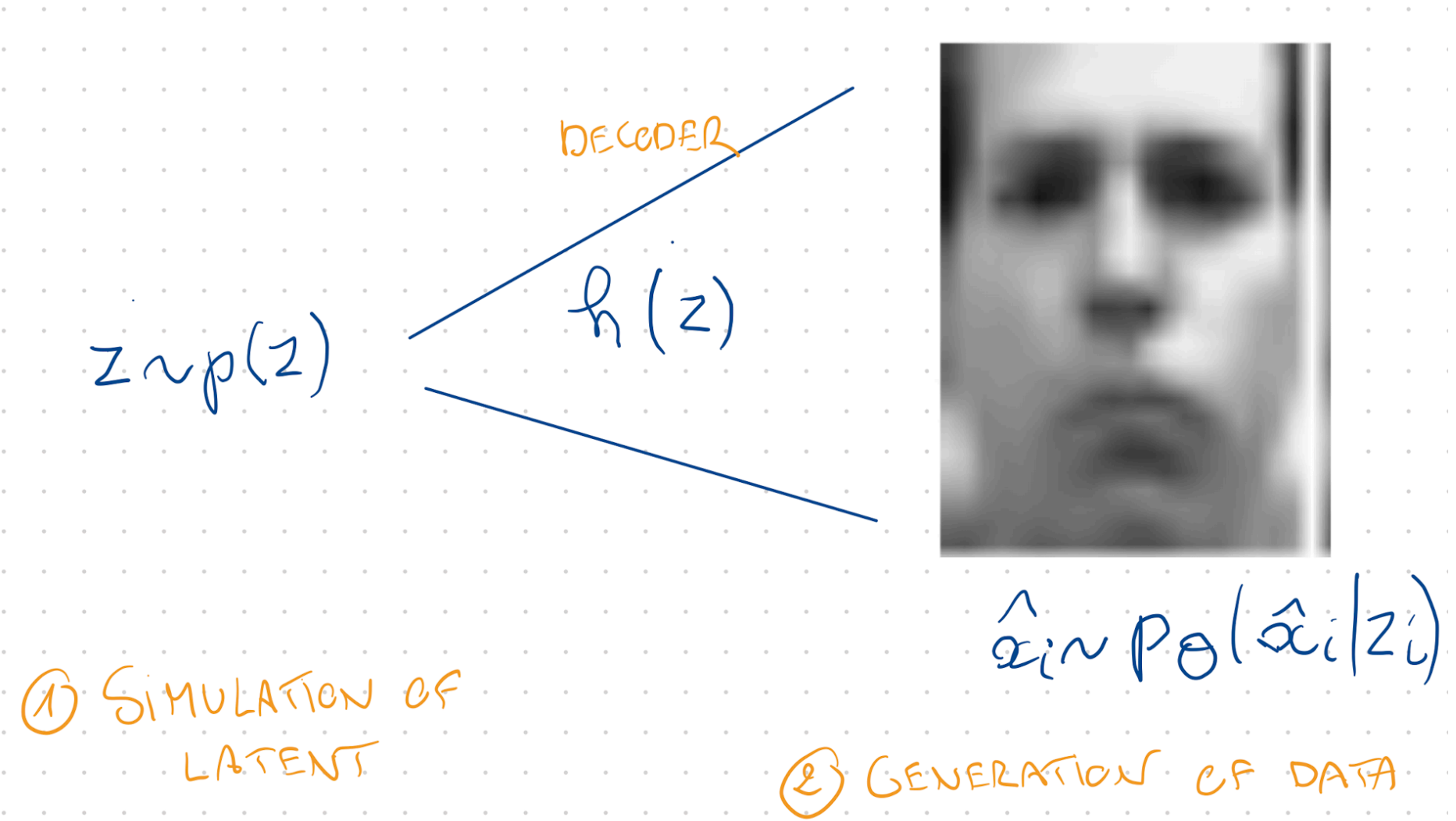
Learning from data



Frey image learning

Example of use

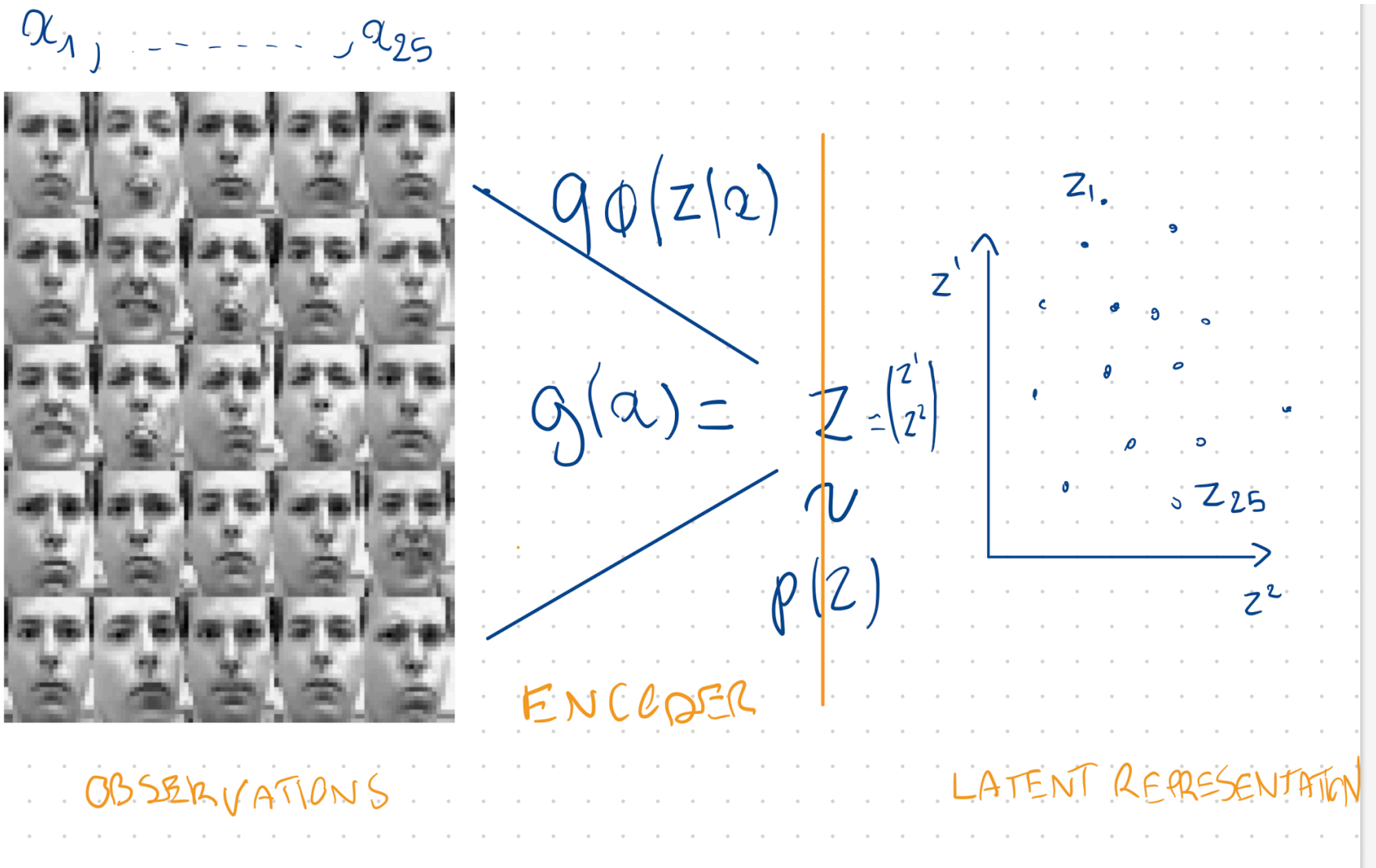
New Data generation from latent simulation



Frey image generation

Example of use

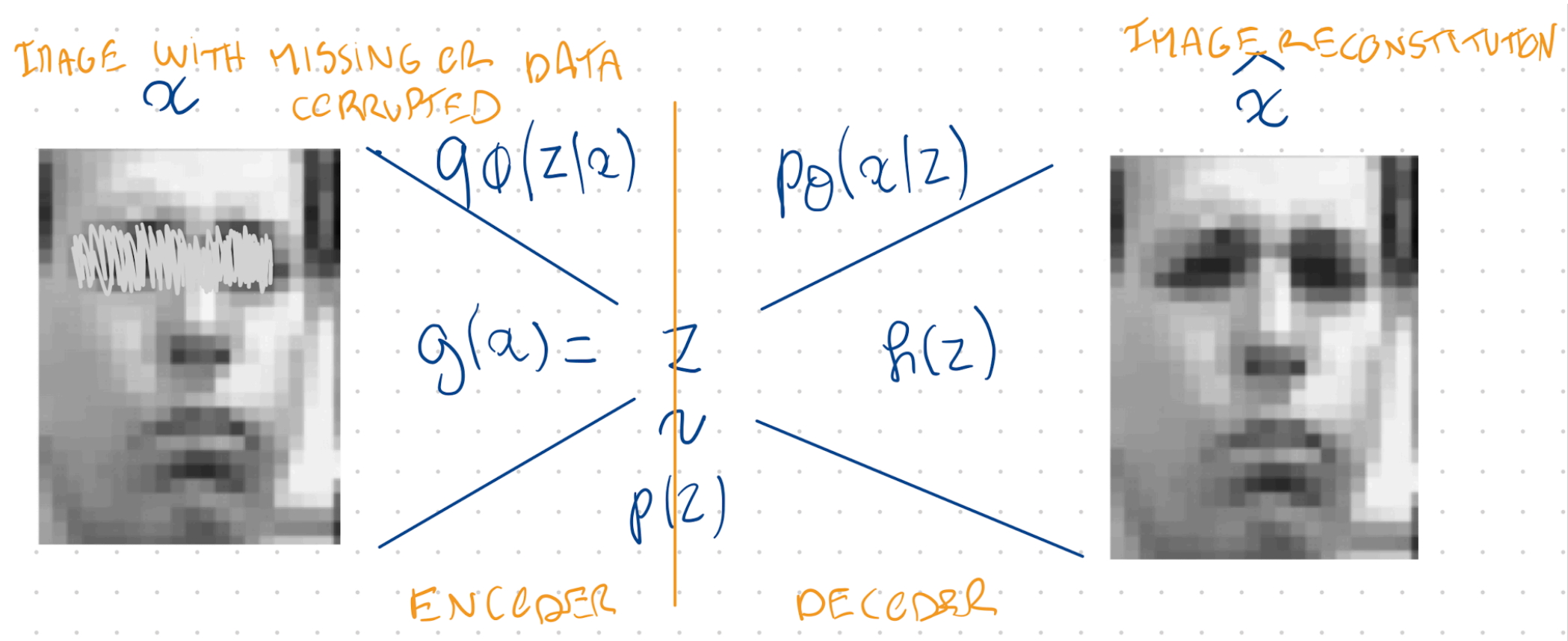
Data representation in latent space



Frey image representation

Example of use

Missing data imputation after learning



Frey image pixel imputation

Ingredient: Parameterization of conditional distributions with Neural Networks

Modeling joint distribution

- $\boldsymbol{x}$: Observed random variables
- $p^*(\boldsymbol{x})$: underlying unknown distribution
- $p_\theta(\boldsymbol{x})$: model distribution
- Goal: $p_\theta(\boldsymbol{x}) \approx p^*(\boldsymbol{x})$

We wish flexible $p_\theta(\boldsymbol{x})$

Modeling Conditional distribution

Classification and regression

$$p_\theta(y|\boldsymbol{x}) \approx p^*(y|\boldsymbol{x})$$

Parameterization of conditional distributions with Neural Networks

Classification

$$\theta = NeuralNet(\boldsymbol{x})$$

$$p_\theta(y|\boldsymbol{x}) = Categorical(y, \theta)$$

Ingredient: Stochastic Gradient

What differences between Oja's rule and VAE

What is a VAE ?

Coupling of 2 **parametric models**

- decoder (generative model): $\boldsymbol{x} = h_\Theta(\boldsymbol{z})$ where $p_\Theta(\boldsymbol{x}|\boldsymbol{z})$
- encoder (recognition model): $\boldsymbol{z} = g_\Phi(\boldsymbol{x})$ where $p(\boldsymbol{z}|\boldsymbol{x})$ is approximated by $q_\Phi(\boldsymbol{z}|\boldsymbol{x})$

In Oja's rule

- There are No probabilistic models
  - → $z = g(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$ and
  - → $\hat{\boldsymbol{x}} = h(z) = z\boldsymbol{w}$,
- Encoder and decoder share the same parameters $w = \Theta = \Phi$

Factor analysis generalizes Oja and is closer to a VAE

Factor analysis considers the observation $\boldsymbol{x} \in \mathbb{R}^D$

$$\boldsymbol{x} = \boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu} + \epsilon$$

where

- the noise $\epsilon \sim \mathcal{N}_D(\mathbf{0}, \boldsymbol{\Psi})$
- the hidden (latent) vector $\boldsymbol{z} \sim \mathcal{N}_L(\mathbf{0}, \boldsymbol{I}_L)$

$$p(\boldsymbol{x}|\boldsymbol{z}, \theta) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu}, \boldsymbol{\Psi})$$

the mean is a linear function of the (hidden) inputs

- $\boldsymbol{W}$ is a $D \times L$ matrix, known as the factor loading matrix,
- $\boldsymbol{\Psi}$ is a $D \times D$ covariance matrix that we take to be diagonal

The special case in which $\boldsymbol{\Psi} = \sigma^2 I$ is called probabilistic principal components analysis or PPCA.

Ingredient : Evidence Lower BOund Minimization

Missing data

In a missing data framework the log-likelihood of the parameters is advantageously expressed as

$$log P_\Theta(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{z}|\boldsymbol{x}} \left[log \frac{P(\boldsymbol{x}, \boldsymbol{z})}{P(\boldsymbol{z}|\boldsymbol{x})} \right]$$

Approximation

When the distribution of $\boldsymbol{z}|\boldsymbol{x}$ is intractable, an **approximation** $q_\Phi(\boldsymbol{z}|\boldsymbol{x})$ is used

$q_\Phi(\boldsymbol{z}|\boldsymbol{x})$ is the inverse function for $p_\Theta(\boldsymbol{x}|\boldsymbol{z})$ in a **Bayes sense**

ELBO

$$
\begin{aligned}
log P_\Theta(\boldsymbol{x}) &= \mathbb{E}_{q_\Phi(\boldsymbol{z}|\boldsymbol{x})}\left[log \frac{P_\Theta(\boldsymbol{x},\boldsymbol{z})}{P_\Theta(\boldsymbol{z}|\boldsymbol{x})}\right] \\
&= \mathbb{E}_{q_\Phi(\boldsymbol{z}|\boldsymbol{x})}\left[log \frac{P_\Theta(\boldsymbol{x},\boldsymbol{z}) q_\Phi(\boldsymbol{z}|\boldsymbol{x})}{P_\Theta(\boldsymbol{z}|\boldsymbol{x}) q_\Phi(\boldsymbol{z}|\boldsymbol{x})}\right] \\
&= \underbrace{\mathbb{E}_{q_\Phi(\boldsymbol{z}|\boldsymbol{x})}\left[log \frac{P_\Theta(\boldsymbol{x},\boldsymbol{z})}{q_\Phi(\boldsymbol{z}|\boldsymbol{x})}\right]}_{ELBO} + \underbrace{\mathbb{E}_{q_\Phi(\boldsymbol{z}|\boldsymbol{x})}\left[log \frac{q_\Phi(\boldsymbol{z}|\boldsymbol{x})}{p_\Theta(\boldsymbol{z}|\boldsymbol{x})}\right]}_{D_{KL}(q_\Phi(\boldsymbol{z}|\boldsymbol{x})|p_\Theta(\boldsymbol{z}|\boldsymbol{x}))}
\end{aligned}
$$

where $D_{KL}(q_\Phi(\boldsymbol{z}|\boldsymbol{x})|p_\Theta(\boldsymbol{z}|\boldsymbol{x})) = -\mathbb{E}_q[\log \frac{p}{q}] \geq -\log \mathbb{E}_q[\frac{p}{q}] \geq 0$ from Jensen

Two for one

- VAE finds parameters which approximately maximize the marginal likelihood $P_\Theta(\boldsymbol{x})$ (good generative function)
- VAE finds the approximation of the recognition model which minimizes the KL divergence

Alternative formulation of ELBO

ELBO can be rewritten as

$$ELBO = E_{q_\Phi(\boldsymbol{z}|\boldsymbol{x})}\left[logP_\Theta(\boldsymbol{x}|\boldsymbol{z})\right] - D_{KL}(q_\Phi(\boldsymbol{z}|\boldsymbol{x})|p_\Theta(\boldsymbol{z}))$$

For a suited choice of $p(\boldsymbol{z})$ and $q(\boldsymbol{z}|\boldsymbol{x})$, $D_{KL}(q_\Phi(\boldsymbol{z}|\boldsymbol{x})|p_\Theta(\boldsymbol{z}))$ can be calculated in closed form.

Exercices

1. Show that the ELBO can be rewritten as above
2. Compute the KL divergence between two multivariate Gaussians

The ELBO is maximized by Stochastic Gradient

Let $\boldsymbol{X}$ be a i.i.d sample of random vector $\boldsymbol{x}$

$$ELBO(\boldsymbol{X}) = \sum_{\boldsymbol{x} \in \boldsymbol{X}} \mathcal{L}(\Theta, \Phi; \boldsymbol{x})$$

Gradient

The Gradient can be separated into 2 parts

1. $\nabla_{\Theta} \mathcal{L}(\Theta, \Phi; \boldsymbol{x})$
2. $\nabla_{\Phi} \mathcal{L}(\Theta, \Phi; \boldsymbol{x})$

Decoder Gradient: $\nabla_\Theta \mathcal{L}(\Theta, \Phi; \boldsymbol{x})$

Given some usually verified conditions and a Monte Carlo Approximation

$$
\begin{aligned}
\nabla_\Theta \mathcal{L}(\Theta, \Phi; \boldsymbol{x}) &= \nabla_\Theta \mathbb{E}_{q_\Phi(\boldsymbol{z}|\boldsymbol{x})} \left[log \frac{P_\Theta(\boldsymbol{x}, \boldsymbol{z})}{q_\Phi(\boldsymbol{z}|\boldsymbol{x})} \right] \\
&= \nabla_\Theta \mathbb{E}_{q_\Phi(\boldsymbol{z}|\boldsymbol{x})} \left[\log P_\Theta(\boldsymbol{x}, \boldsymbol{z}) \right] \\
&= \mathbb{E}_{q_\Phi(\boldsymbol{z}|\boldsymbol{x})} \left[\nabla_\Theta \log P_\Theta(\boldsymbol{x}, \boldsymbol{z}) \right] \\
&\approx \nabla_\Theta \log P_\Theta(\boldsymbol{x}, \boldsymbol{z})
\end{aligned}
$$

Encoder Gradient: $\nabla_\Phi \mathcal{L}(\Theta, \Phi; \boldsymbol{x})$

Encoder Gradient is more difficult to compute since in general

$$\begin{aligned}\nabla_\Phi \mathcal{L}(\Theta, \Phi; \boldsymbol{x}) &= \nabla_\Phi \mathbb{E}_{q_\Phi(\boldsymbol{z}|\boldsymbol{x})}\left[log \frac{P_\Theta(\boldsymbol{x}, \boldsymbol{z})}{q_\Phi(\boldsymbol{z}|\boldsymbol{x})}\right] \\ &= \nabla_\Phi \mathbb{E}_{q_\Phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log P_\Theta(\boldsymbol{x}, \boldsymbol{z}) - \log q_\Phi(\boldsymbol{z}|\boldsymbol{x})\right] \\ &\neq \mathbb{E}_{q_\Phi(\boldsymbol{z}|\boldsymbol{x})}\left[\nabla_\Phi \log P_\Theta(\boldsymbol{x}, \boldsymbol{z}) - \nabla_\Phi \log q_\Phi(\boldsymbol{z}|\boldsymbol{x})\right]\end{aligned}$$

A reparametrization (variable change) trick allows a workaround

Encoder function

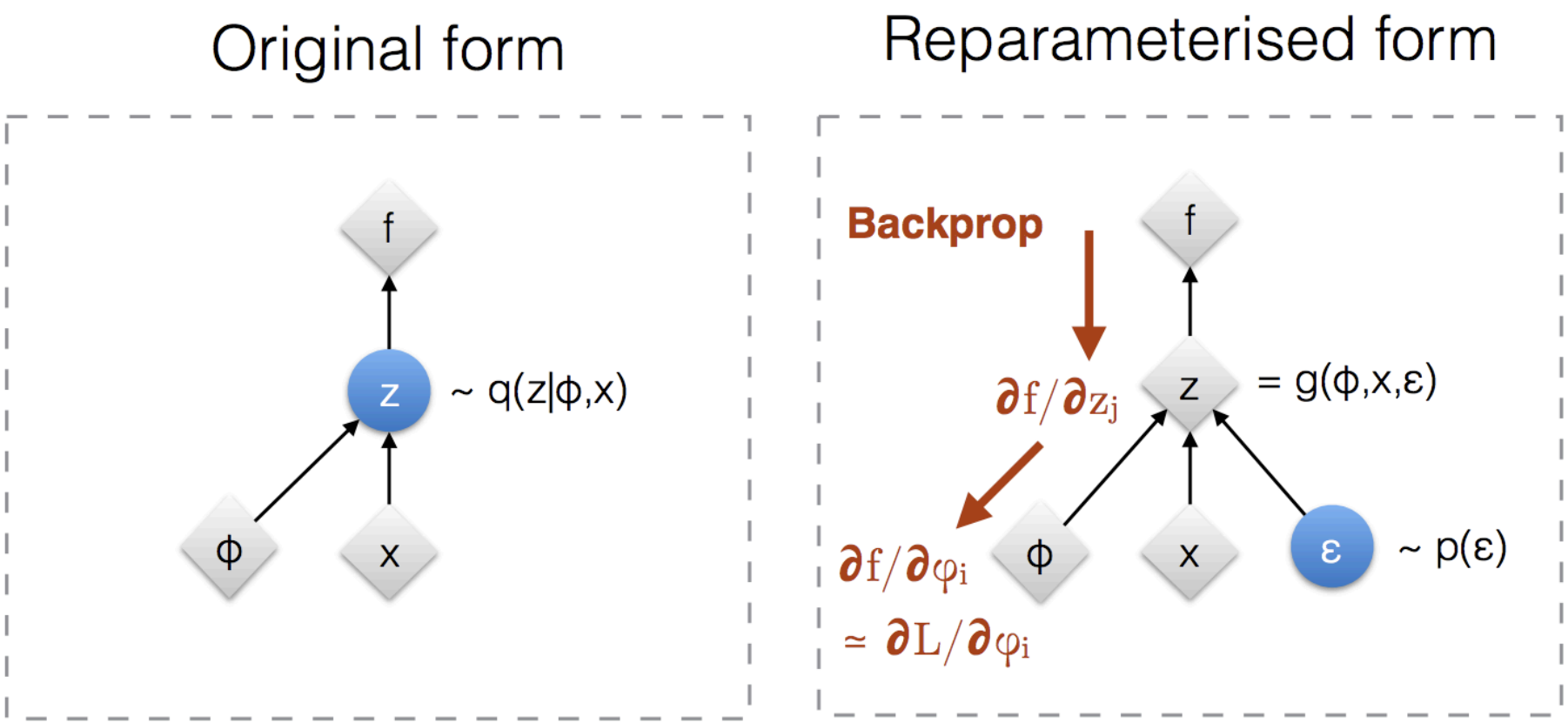
Let us rewrite the decoder function with a random vector $\boldsymbol{\epsilon}$ whose distribution is not parametrized by $\boldsymbol{\Phi}$:

$$\boldsymbol{z} = g(\boldsymbol{x}, \boldsymbol{\epsilon}; \boldsymbol{\Phi})$$

$$\begin{aligned}
\nabla_{\boldsymbol{\Phi}} \mathcal{L}(\boldsymbol{\Theta}, \boldsymbol{\Phi}; \boldsymbol{x}) &= \nabla_{\boldsymbol{\Phi}} \mathbb{E}_{p(\boldsymbol{\epsilon})} \left[log \frac{P_{\boldsymbol{\Theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\Phi}}(\boldsymbol{z}|\boldsymbol{x})} \right] \\
&= \nabla_{\boldsymbol{\Phi}} \mathbb{E}_{p(\boldsymbol{\epsilon})} \left[\log P_{\boldsymbol{\Theta}}(\boldsymbol{x}, \boldsymbol{z}) - \log q_{\boldsymbol{\Phi}}(\boldsymbol{z}|\boldsymbol{x}) \right] \\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})} \left[\nabla_{\boldsymbol{\Phi}} \log P_{\boldsymbol{\Theta}}(\boldsymbol{x}, \boldsymbol{z}) - \nabla_{\boldsymbol{\Phi}} \log q_{\boldsymbol{\Phi}}(\boldsymbol{z}|\boldsymbol{x}) \right] \\
&\approx \nabla_{\boldsymbol{\Phi}} \log P_{\boldsymbol{\Theta}}(\boldsymbol{x}, \boldsymbol{z}) - \nabla_{\boldsymbol{\Phi}} \log q_{\boldsymbol{\Phi}}(\boldsymbol{z}|\boldsymbol{x})
\end{aligned}$$

We just have to compute $\log q_{\boldsymbol{\Phi}}(\boldsymbol{z}|\boldsymbol{x})$ after the change of variable

Reparametrization trick



: Deterministic node

: Random node

[Kingma, 2013]
[Bengio, 2013]
[Kingma and Welling 2014]
[Rezende et al 2014]

Computing $\log q_\Phi(\boldsymbol{z}|\boldsymbol{x})$ with a change of variable

$$\log q_\Phi(\boldsymbol{z}|\boldsymbol{x}) = \log p(\boldsymbol{\epsilon}) - \log d_\Phi(\boldsymbol{x}, \boldsymbol{\epsilon})$$

where the second term is the log of the absolute value of the determinant of the Jacobian matrix:

$$\log d_\Phi(\boldsymbol{x}, \boldsymbol{\epsilon}) = \log \left| \det \begin{pmatrix} \frac{\partial z_1}{\partial \epsilon_1} & \cdots & \frac{\partial z_1}{\partial \epsilon_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_k}{\partial \epsilon_1} & \cdots & \frac{\partial z_k}{\partial \epsilon_k} \end{pmatrix} \right|$$

variable change $g()$ is chosen for the logdet being computationally affordable/simple

Factorized Gaussian Posterior

Model

$$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) = EncoderNN_{\Phi}(\boldsymbol{x})$$

$$q_{\Phi}(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, diag(\boldsymbol{\sigma}^2))$$

$$q_{\Phi}(\boldsymbol{z}|\boldsymbol{x}) = \prod_i q_{\Phi}(z_i|\boldsymbol{x}) = \prod_i \mathcal{N}(z_i|EncoderNN_{\Phi}(\boldsymbol{x})) = \prod_i \mathcal{N}(z_i|\mu_i, \sigma_i^2)$$

Reparametrization

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

$$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$$

Factorized Gaussian Posterior

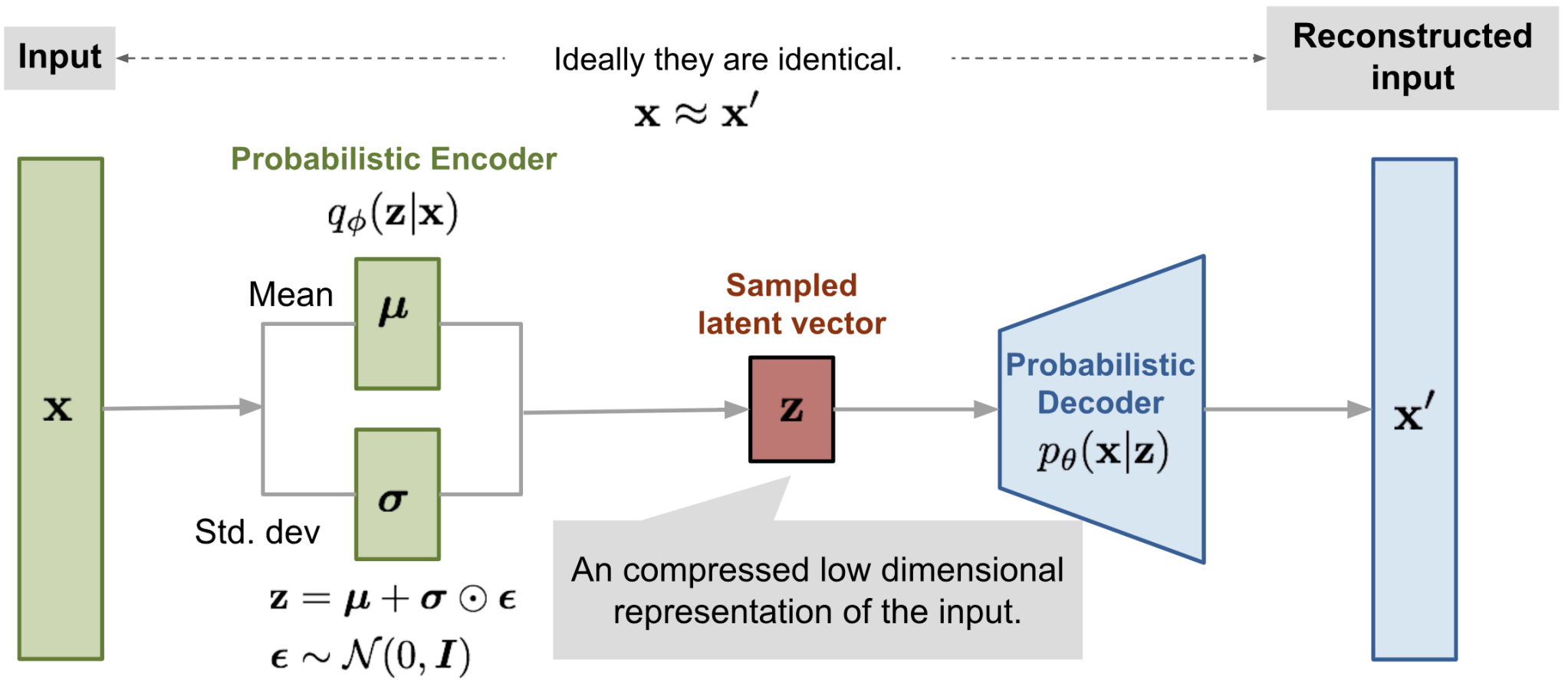
The Jacobian of the transformation is

$$\log d_\Phi(\boldsymbol{x}, \boldsymbol{\epsilon}) = \log \left| \det \begin{pmatrix} \frac{\partial z_1}{\partial \epsilon_1} & \cdots & \frac{\partial z_1}{\partial \epsilon_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_k}{\partial \epsilon_1} & \cdots & \frac{\partial z_k}{\partial \epsilon_k} \end{pmatrix} \right| = \log \prod_i \sigma_i$$

The log posterior density is

$$\begin{aligned} \log q_\Phi(\boldsymbol{z}|\boldsymbol{x}) &= \log p(\boldsymbol{\epsilon}) - \log |det(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}})| \\ &= \sum_i \log \mathcal{N}(\epsilon_i; 0, 1) - \log \sigma_i \end{aligned}$$

Reparametrization trick



Full Gaussian posterior

Model

$$q_\Phi(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$$

Reparametrization

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

$$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{L}\boldsymbol{\epsilon}$$

Where $\boldsymbol{L}$ is a lower triangular matrix obtained from a Cholesky decompostion of $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^T$

Full Gaussian posterior

The Jacobian has a simple form

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}} = \boldsymbol{L}$$

As the determinant of a triangular matrix is the product of its diagonal terms,

$$\begin{aligned}\log q_\Phi(\boldsymbol{z}|\boldsymbol{x}) &= \log p(\boldsymbol{\epsilon}) - \log |det(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}})| \\ &= \sum_i \log \mathcal{N}(\epsilon_i; 0, 1) - \log L_{ii}\end{aligned}$$

Algorithm input and output

Input:

- $\boldsymbol{X}$: Dataset
- $h(\boldsymbol{z})$ decoding function, with dist. $p_\Theta(\boldsymbol{x}, \boldsymbol{z})$
- $g(\boldsymbol{x})$ encoding function with dist. $q_\Psi(\boldsymbol{z}|\boldsymbol{x})$

Output:

- Θ
- Φ

Algorithm

Initialisation of Θ and Φ

While SGD not converged do

- Draw a random minibatch $\boldsymbol{X}^M \in \boldsymbol{X}$
- $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ (Random noise for every datapoint in $\boldsymbol{X}^M$)
- Compute
  - $\tilde{\mathcal{L}}_{\Theta,\Phi}(\boldsymbol{X}^M, \boldsymbol{\epsilon}) = \frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{X}^M} \left(\log p_\Theta(\boldsymbol{x}, \boldsymbol{z}) - \underbrace{\log q_\Phi(\boldsymbol{z}|\boldsymbol{x})}_{\log p(\epsilon) - \log d_\Phi(\boldsymbol{x}, \boldsymbol{\epsilon})} \right)$ and
  - its gradients
    - $\nabla_\Theta \tilde{\mathcal{L}}_{\Theta,\Phi}(\boldsymbol{X}^M, \boldsymbol{\epsilon}) = \frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{X}^M} \frac{\partial \log p_\Theta(\boldsymbol{x}, \boldsymbol{z})}{\partial \Theta}$
    - $\nabla_\Phi \tilde{\mathcal{L}}_{\Theta,\Psi}(\boldsymbol{X}^M, \boldsymbol{\epsilon}) = \frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{X}^M} \frac{\log \partial d_\Phi(\boldsymbol{x}, \boldsymbol{\epsilon})}{\partial \Phi}$
- $\begin{pmatrix} \Theta \\ \Phi \end{pmatrix} := \begin{pmatrix} \Theta \\ \Phi \end{pmatrix} + \eta \begin{pmatrix} \nabla_\Theta \tilde{\mathcal{L}}_{\Theta,\Phi}(\boldsymbol{X}^M, \boldsymbol{\epsilon}) \\ \nabla_\Phi \tilde{\mathcal{L}}_{\Theta,\Phi}(\boldsymbol{X}^M, \boldsymbol{\epsilon}) \end{pmatrix}$

Original example from Kingma: Gaussian model with MLP parametrization

Multivariate Gaussian decoder with a diagonal covariance structure

Decoder $p_\Theta(\boldsymbol{x}|\boldsymbol{z})$ or decoder (just swap $\boldsymbol{x}$ and $\boldsymbol{z}$) are assumed to have multivariate Gaussian dist. with a diagonal covariance structure:

Decoder

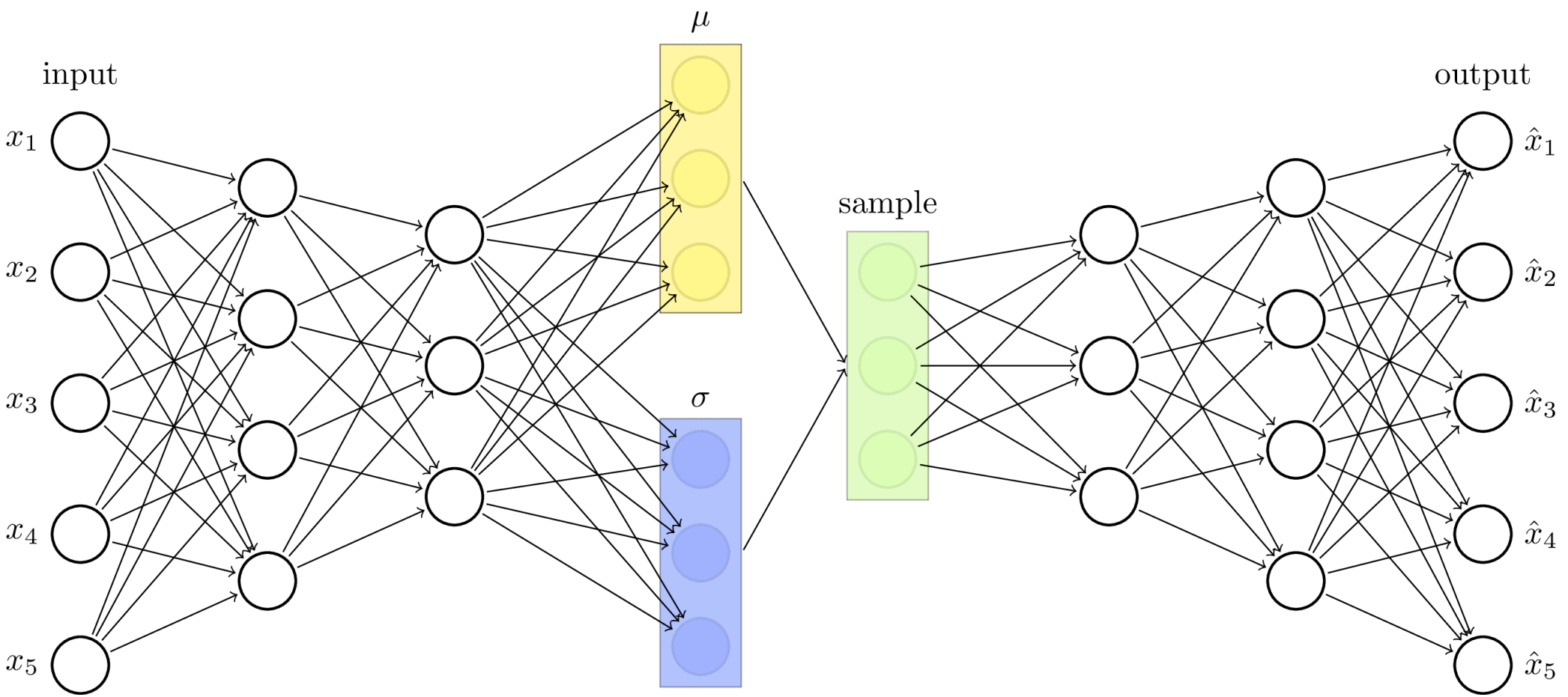
- $logp(x|z) = \log \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \sigma^2 I)$ where $\boldsymbol{\mu} = W_4 \boldsymbol{h} + b_4$
- $\log \sigma^2 = W_5 \boldsymbol{h} + b_5$
- $\boldsymbol{h} = \tanh(W_3 \boldsymbol{z} + b_3)$

where $\{W_3, W_4, W_5, b_3, b_4, b_5\}$ are the weights and biases of the MLP and part of Θ when used as decoder.

Encoder

- Let us consider the prior $p_\Theta(\boldsymbol{z}) = \mathcal{N}(0, I)$
- swap $\boldsymbol{x}$ and $\boldsymbol{z}$ in the decoder above to get $q_\Phi(\boldsymbol{z}|\boldsymbol{x})$

Gaussian VAE illustrated



Mixture of Experts

Historical Context

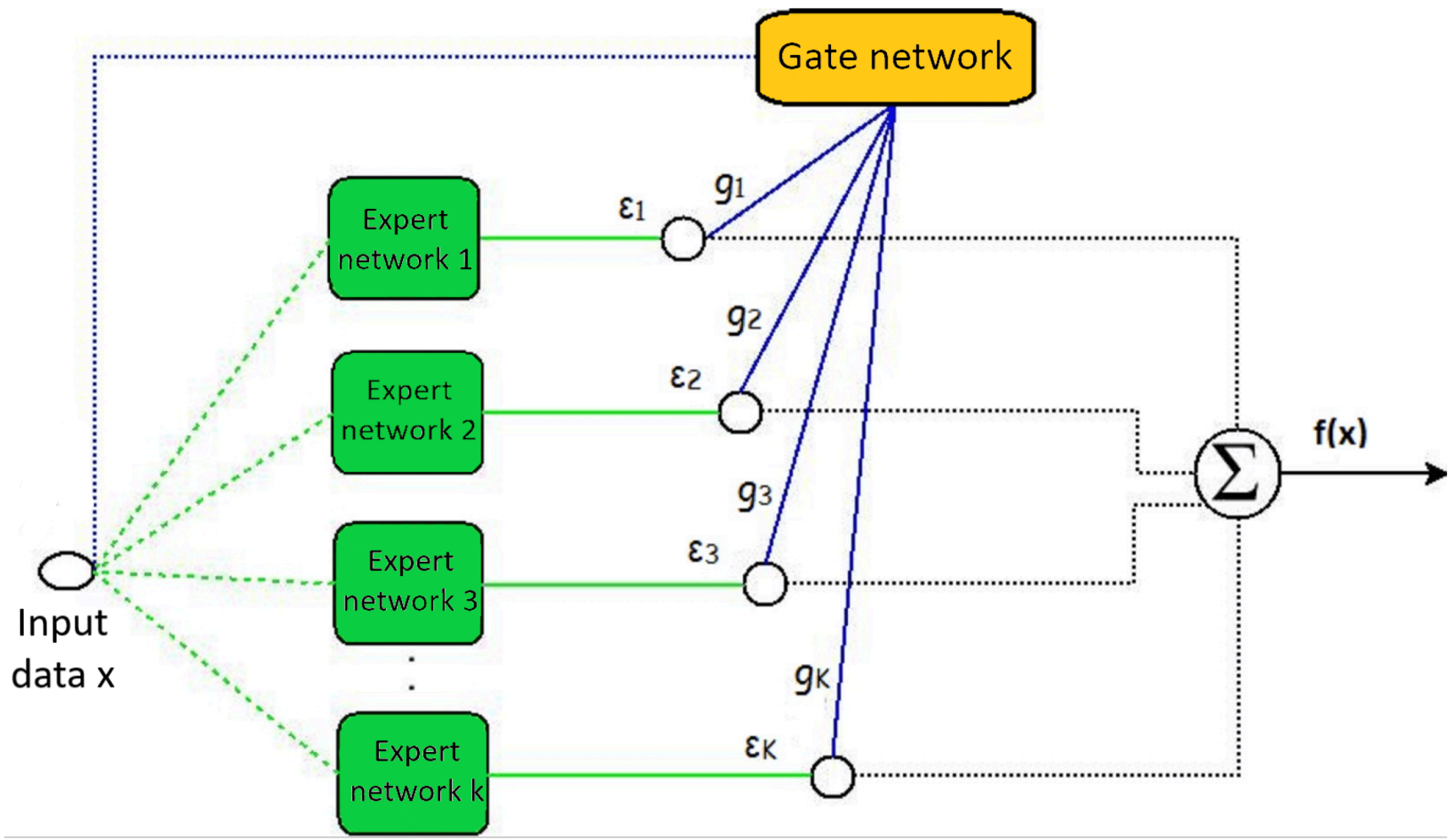
- Introduced in the early 1990s by Jacobs et al. (1991).

Ideas and Intuitions

- **Inspired by cognitive science**, mimicking expert decision-making by assigning specialized sub-models to different tasks.
- **Enhances generalization** by combining multiple specialized models rather than relying on a single monolithic structure.
- **Divide-and-conquer approach** enables decomposition of complex tasks into manageable subtasks, improving prediction accuracy (Masoudnia & Ebrahimpour, 2014; Yuksel et al., 2012).
- **Widely applied in machine learning**, including speech recognition, image processing, LLM (Deepseek, Mixtral, ChatGPT…)

MoE Components

- **Experts**: Independent models (e.g., neural networks, regressions) specialized in subsets of data.
- **Gating Network**: Evaluates input data and assigns relevance weights to experts.
- Combines expert outputs into a final prediction (Gormley & Eisner, 2019).



Model Formulation

$$f(y_i; \boldsymbol{x}_i, \boldsymbol{\Theta}) = \sum_{k=1}^{K} g_k(\boldsymbol{x}_i; \boldsymbol{\alpha}_k) f_k(y_i; \boldsymbol{x}_i, \boldsymbol{\beta}_k)$$

- $g_k(\boldsymbol{x}_i)$: Output of the gating network

Probabilistic formulation

- $f(y_i; \boldsymbol{x}_i, \boldsymbol{\Theta}) = p(y_i | \boldsymbol{x}_i, \boldsymbol{\Theta})$
- the gating network comes from a latent variable z:

$$g_k(\boldsymbol{x}_i) = p(z_i = k \mid \boldsymbol{x}_i)$$

where

$$z_i = k \mid \boldsymbol{x}_i \sim M\left(1; g(\boldsymbol{x}_i) = (g_1(\boldsymbol{x}_i), \ldots, g_K(\boldsymbol{x}_i))\right)$$

- g is a sigmoid function for $K = 2$ or softmax for $K > 2$.
- The expert $f_k(y_i; \boldsymbol{x}_i, \boldsymbol{\beta}_k)$ models a pdf corresponding the nature of y_i

MoE Architectures

- Different types of expert models (Gormley & Eisner, 2019).
- Variants include mixture models (unsupervised), prediction models (supervised).

Advantages & Challenges

- MoEs enable large-scale models while reducing computational costs.
- Challenges: high parameter count, complex gating network training.

Example of regression mixture

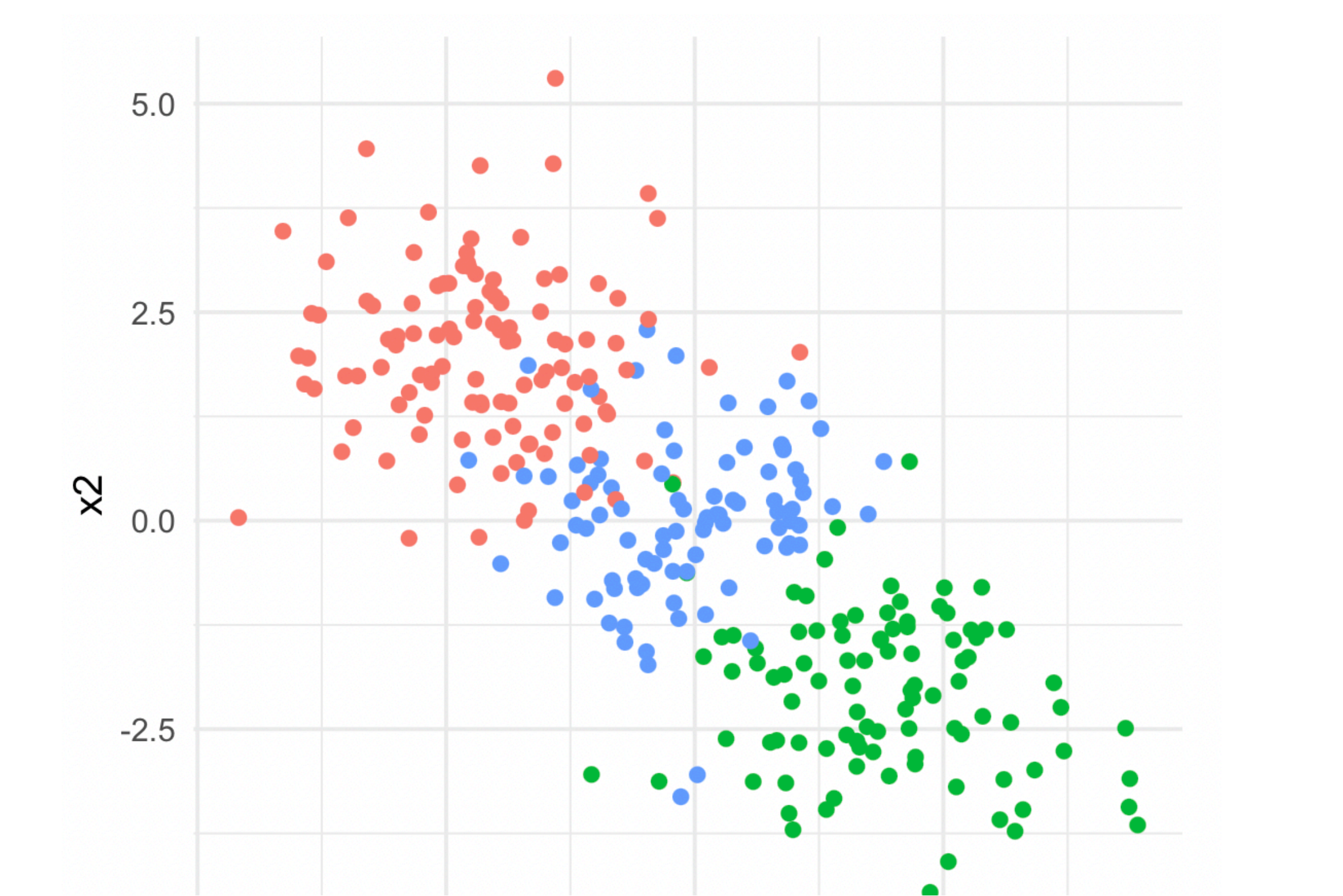
Training Process

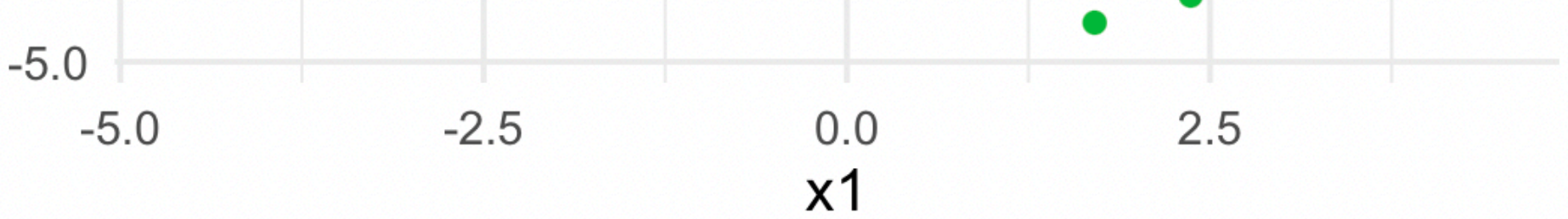
- **Labeled Data:**
  → The MoE is trained on a dataset where each input is associated with a known output.
- **Expert Specialization:**
  → Each expert learns to master a specific subset of the data, allowing for finer modeling of variations present in the data.
- **Gating Network Training:**
  → The gating network learns to associate inputs with the appropriate experts based on data characteristics.

Inference Phase

- **Input Routing:**
  → For new data, the gating network evaluates its characteristics and selects the most suitable expert to make the prediction.
- **Output Combination:**
  → In some cases, outputs from multiple experts may be combined to obtain a more robust final prediction.

Example of regression mixture


5.0
2.5
0.0
-2.5
x2



##

Example of regression mixture

Singular Value Decomposition

Singular Value Decomposition

Eigendecomposition of symmetric matrices

$\forall \boldsymbol{A} \in \mathbb{R}^{n \times n}$, there exist an orthonormal matrix $\boldsymbol{Q} \in R^{n \times n}$ and a diagonal matrix $\boldsymbol{\Lambda} = diag(\lambda_1, \cdots, \lambda_n)$

$$A = \boldsymbol{Q} \boldsymbol{\Lambda} \boldsymbol{Q}^T$$

Singular Value Decomposition

Extend the decomposition to **rectangular matrices**

$$\boldsymbol{X} = \boldsymbol{U} \boldsymbol{S} \boldsymbol{V}^{\boldsymbol{T}}$$

Applications in machine learning

- **Dimensionality Reduction**: SVD can be used for dimensionality reduction by reducing the rank of a matrix
- **Latent Semantic Analysis**: By decomposing a term-document matrix using SVD, LSA can capture the latent semantic structure of the data
- **Principal Component Analysis (PCA)**: PCA is a SVD
- **Recommender Systems**: By factorizing the matrix using SVD, we can identify latent factors or features that capture underlying patterns and preferences.
- **Image Compression**: SVD is used in image compression techniques such as JPEG.
- **Matrix Completion**: SVD-based techniques are used in matrix completion problems, where missing or incomplete data needs to be imputed.
- ...

Existence of the SVD for general matrices

For any matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, there exist two orthogonal matrices $\boldsymbol{U} \in \mathbb{R}^{n \times n}$, $\boldsymbol{V} \in \mathbb{R}^{d \times d}$ and a nonnegative, "diagonal" matrix $\boldsymbol{S} \in \mathbb{R}^{n \times d}$ such that

$$\boldsymbol{X}_{n \times d} = \boldsymbol{U}_{n \times n} \boldsymbol{S}_{n \times d} \boldsymbol{V}^T_{d \times d}$$

where $\boldsymbol{U}^T \boldsymbol{U} = \boldsymbol{I}$ and $\boldsymbol{V}^T \boldsymbol{V} = \boldsymbol{I}$.

In a vector form

$$\boldsymbol{X}_{n \times d} = \sum_{j=1}^{r} S_{jj} \boldsymbol{u}_j \boldsymbol{v}_j^T$$

where $r = rank(\boldsymbol{X})$.

Geometrical interpretation

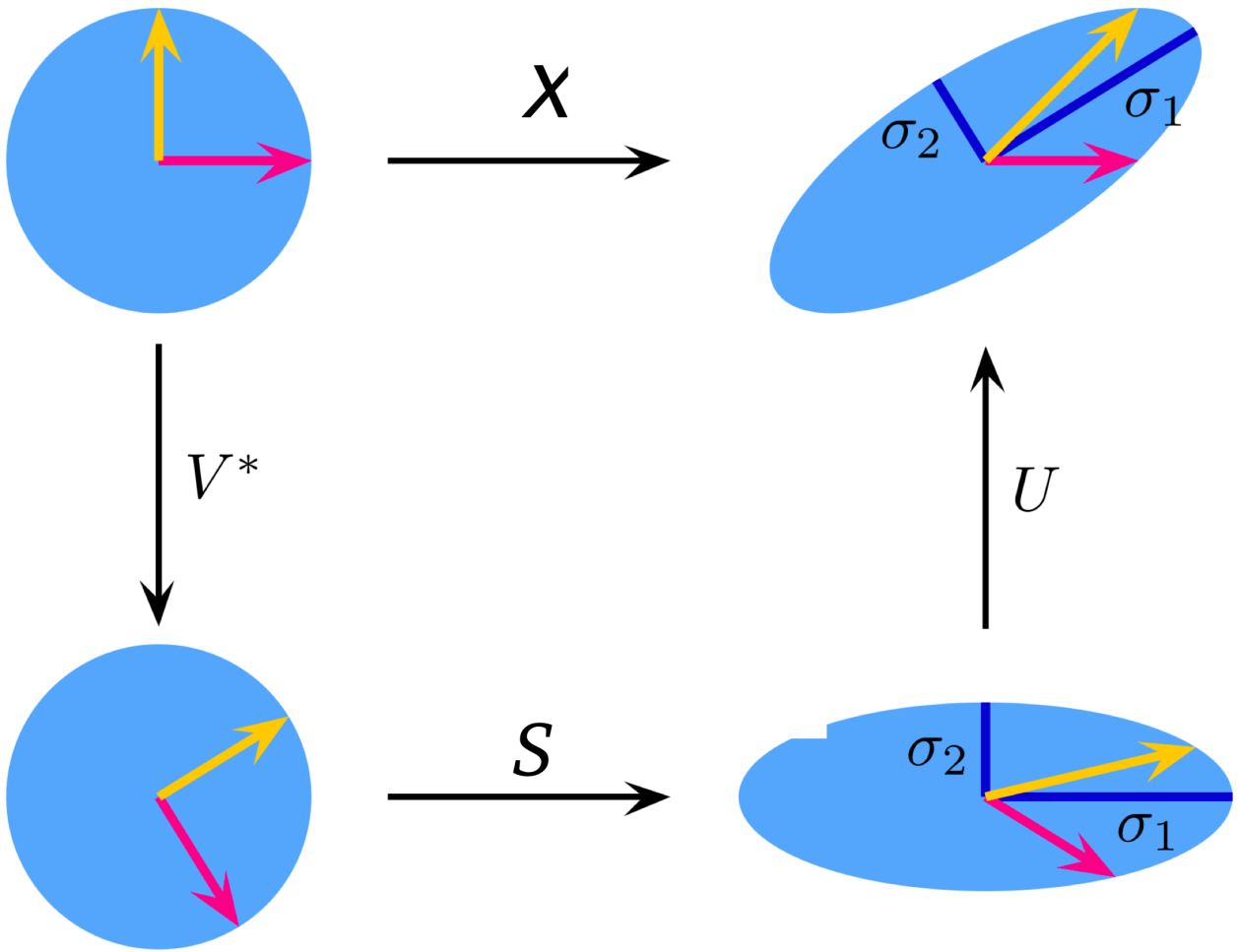
Given any matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ it defines a linear transformation:

$$f : \mathbb{R}^d \to \mathbb{R}^n, f(\boldsymbol{x}) = \boldsymbol{X}\boldsymbol{x}.$$

The linear transformation f can be decomposed into three operations:

$$\underbrace{\boldsymbol{X}}_{linear\ transformation}\ \boldsymbol{x} = \underbrace{\boldsymbol{U}}_{rotation}\ \underbrace{\boldsymbol{S}}_{scaling}\ \underbrace{\boldsymbol{V}^T}_{rotation}\ \boldsymbol{x}$$

Geometrical interpretation



$$X = U \cdot S \cdot V^*$$

Different versions of SVD

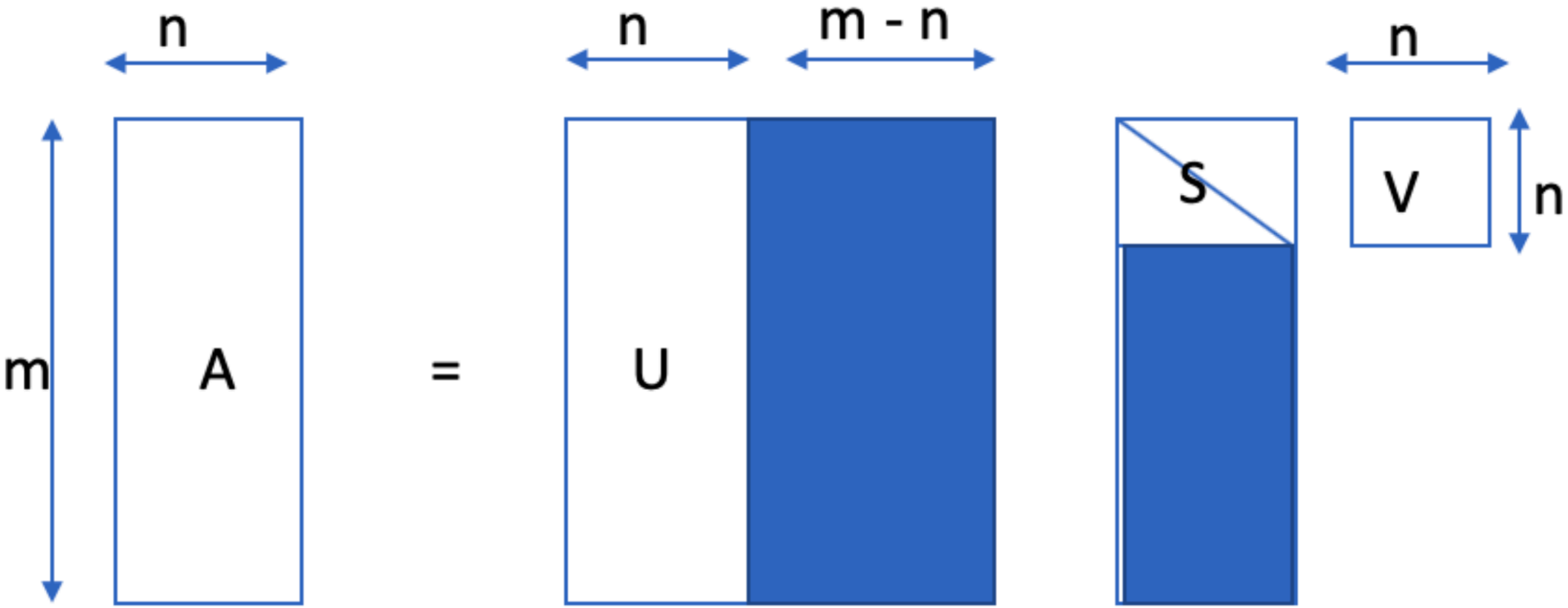
- Full SVD:

$$X_{n\times d} = \boldsymbol{U}_{n\times n}\boldsymbol{S}_{n\times d}\boldsymbol{V}^T_{d\times d}$$

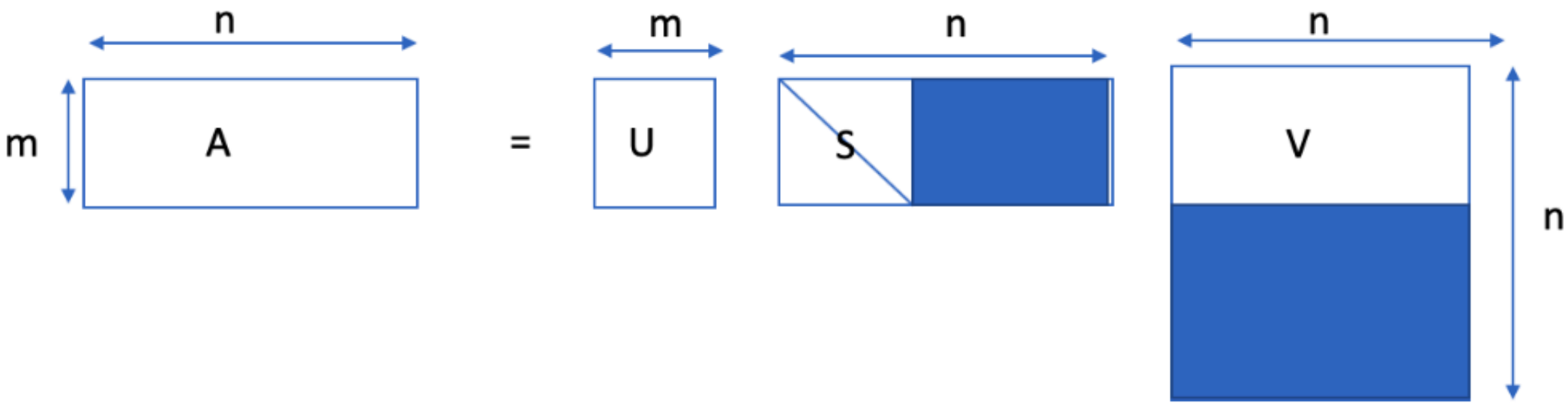
- Economy sized (thin, compact) SVD:

$$X_{n\times d} = \boldsymbol{U}_{n\times r}\boldsymbol{S}_{r\times r}\boldsymbol{V}^T_{r\times d}$$

SVD $n > d$



SVD $n < d$



Existence of the SVD

Consider $\boldsymbol{A} = \boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$ where $\boldsymbol{\Lambda} = diag(\lambda_1, \cdots, \lambda_d)$ with $\lambda_1 \geq \cdots \geq \lambda_r > 0 = \lambda_{r+1} = \cdots = \lambda_d$ (where $r = rank(\boldsymbol{X}) \leq d$).

Let $\sigma_i = \sqrt{\lambda_i}$ and correspondingly form the matrix

$$\boldsymbol{S}_{n\times d} = \begin{pmatrix} diag(\sigma_1, \cdots, \sigma_r) & 0_{r\times(d-r)} \\ 0_{(n-r)\times r} & 0_{(n-r)\times(d-r)} \end{pmatrix}$$

Define also

$$\boldsymbol{u}_i = \frac{1}{\sigma_i}\boldsymbol{X}\boldsymbol{v}_i \in \mathbb{R}^n,$$

for each $1 \leq i \leq r$.

Existence of the SVD

Exercice

It is easy to show that the $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_r$ are orthonormal vectors.

Completion if needed

Choose $\boldsymbol{u}_{r+1}, \cdots, \boldsymbol{u}_n \in \mathbb{R}^n$ (through basis completion) such that

$$\boldsymbol{U} = [\boldsymbol{u}_1 \cdots \boldsymbol{u}_n] \in \mathbb{R}^{n \times n}$$

is an orthogonal matrix.

It verifies

$$\boldsymbol{XV} = \boldsymbol{US},$$

i.e.,

Existence of the SVD

$$\boldsymbol{X}[\boldsymbol{v}_1, \cdots \boldsymbol{v}_r \boldsymbol{v}_{r+1} \cdots \boldsymbol{v}_d] = [\boldsymbol{u}_1 \cdots \boldsymbol{u}_r \boldsymbol{u}_{r+1} \boldsymbol{u}_n] \begin{pmatrix} diag(\sigma_1, \cdots, \sigma_r) & 0_{r \times (d-r)} \\ 0_{(n-r) \times r} & 0_{(n-r) \times (d-r)} \end{pmatrix}$$

Two possible cases:

- $1 \leq i \leq r$: $\boldsymbol{X}\boldsymbol{v}_i = \sigma_i \boldsymbol{u}_i$ by construction.
- $i > r$: $\boldsymbol{X}\boldsymbol{v}_i = 0$, which is due to $\boldsymbol{X}^T \boldsymbol{X}\boldsymbol{v}_i = \boldsymbol{C}\boldsymbol{v}_i = 0\boldsymbol{v}_i = 0$.

Consequently, we have obtained that

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$$

Properties

The linear application characterized by $\boldsymbol{X}$ has the following properties:

- $rank(\boldsymbol{X}) = r$ is the number of non zero singular values
- $kernel(\boldsymbol{X}) = span(\boldsymbol{v}_{r+1}, \cdots, \boldsymbol{v}_n)$
- $range(\boldsymbol{X}) = span(\boldsymbol{u}_1, \cdots, \boldsymbol{u}_r)$

Low rank approximation of a matrix X

Goal

Approximate a given matrix $\boldsymbol{X}$ with a rank-k matrix, for a target rank k.

Motivations

- Compression
- De-noising
- Matrix completion

A first toy example

```r
X<-matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12),4,3,byrow=TRUE)
X.svd<-svd(X)
cat("Original matrix:\n")
```

```
Original matrix:
```

```r
print(X)
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
```

```r
k<-2
cat("Approximation of rank 2:\n")
```

```
Approximation of rank 2:
```

```r
print(X.svd$u[,1:k]%*%diag(X.svd$d[1:k])%*%t(X.svd$v[,1:k]))
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
```

```r
cat("A basis of the column space:\n")
```

```
A basis of the column space:
```

```r
print(X.svd$u[,1:k])
```

```
           [,1]        [,2]
[1,] -0.1408767 -0.82471435
[2,] -0.3439463 -0.42626394
[3,] -0.5470159 -0.02781353
[4,] -0.7500855  0.37063688
```

```r
cat("\nA basis of the kernel:\n")
```

A basis of the kernel:

```
print(X.svd$u[,1:k])
```

```
           [,1]        [,2]
[1,] -0.1408767 -0.82471435
[2,] -0.3439463 -0.42626394
[3,] -0.5470159 -0.02781353
[4,] -0.7500855  0.37063688
```

Illustration of svd in image compression

See the demo of Tim Baumann

Example borrowed from rich-d-wilkinson.github.io

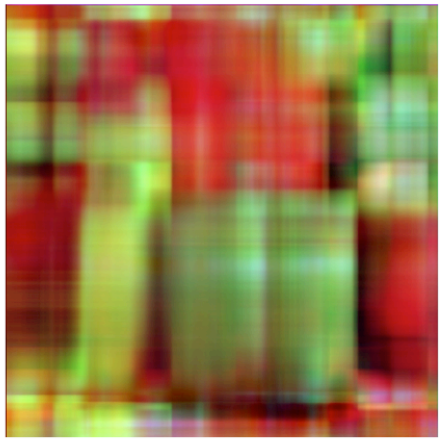
The 512 × 512 colour image is stored as three matrices R, B, G of the same dimension 512×512 giving the intensity of red, green, and blue for each pixel. Naively storing this matrix requires 5.7Mb.

```
1 library(tiff)
2 library(rasterImage)
3 peppers<-readTIFF("../Silo-Images/Peppers.tiff")
4 plot(as.raster(peppers))
```

Illustration of svd in image compression

Below the SVD of the three colour intensity matrices, and the view the image that results from using reduced rank versions with rank $k \in \{5, 30, 100, 300\}$

```
svd_image <- function(im,k){
  s <- svd(im)
  Sigma_k <- diag(s$d[1:k])
  U_k <- s$u[,1:k]
  V_k <- s$v[,1:k]
  im_k <- U_k %*% Sigma_k %*% t(V_k)
   ## the reduced rank SVD produces some intensities <0 and >1.
  # Let's truncate these
  im_k[im_k>1]=1
  im_k[im_k<0]=0
  return(im_k)
}

par(mfrow=c(2,2), mar=c(1,1,1,1))

pepprssvd<- peppers
for(k in c(4,30,100,300)){
  svds<-list()
  for(ii in 1:3) {
    pepprssvd[,,ii]<-svd_image(peppers[,,ii],k)
  }
  plot(as.raster(pepprssvd))
}
```



Low rank approximation of a matrix $\boldsymbol{X}$

Frobenius norm

The Frobenius norm of a matrix $\boldsymbol{X}$ is defined as

$$\|\boldsymbol{X}\|_F^2 = \sum_{ij} X_{ij}^2 = trace(\boldsymbol{X}^T\boldsymbol{X}) = \sum_{j=1}^{r} \sigma_j^2$$

Rank k matrix $\hat{\boldsymbol{X}}_k$

Let

$$\hat{\boldsymbol{X}}_k = \sum_{j=1}^{k} \sigma_j \boldsymbol{u}_j \boldsymbol{v}_j^T$$

Low rank approximation of a matrix

For any matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ with non null singular values $\sigma_1 \geq \sigma_2 \geq \quad \cdots \geq \sigma_r$

$$\hat{\boldsymbol{X}}_k = \arg \min_{\hat{X}: rank(\hat{X}) = k} \|\boldsymbol{X} - \hat{\boldsymbol{X}}\|_F^2$$

$$\min_{\hat{X}: rank(\hat{X}) = k} \|\boldsymbol{X} - \hat{\boldsymbol{X}}\|_F^2 = \sum_{j=k+1}^{r} \sigma_j^2$$

Proof

We have

$$\|X - X_k\|_F^2 = \left\| \sum_{i=k+1}^{n} \sigma_i u_i v_i^\top \right\|_F^2 = \sum_{i=k+1}^{n} \sigma_i^2$$

We need to show that if $Y_k = AB^\top$ where A and B have k columns then

$$\|X - X_k\|_F^2 = \sum_{i=k+1}^{n} \sigma_i^2 \leq \|X - Y_k\|_F^2.$$

Proof

By the triangle inequality with the spectral norm, if $X = X' + X''$ then $\sigma_1(X) \leq \sigma_1(X') + \sigma_1(X'')$.

Suppose X'_k and X''_k respectively denote the rank k approximation to X' and X'' by SVD.

Then, for any $i, j \geq 1$

$$\begin{aligned}
\sigma_i(X') + \sigma_j(X'') &= \sigma_1(X' - X'_{i-1}) + \sigma_1(X'' - X''_{j-1}) \\
&\geq \sigma_1(X - X'_{i-1} - X''_{j-1}) \\
&\geq \sigma_1(X - X_{i+j-2}) \qquad (\text{since } \mathbf{rank}(X'_{i-1} + X''_{j-1}) \leq \mathbf{rank}\,(X_{i+j-2})) \\
&= \sigma_{i+j-1}(X).
\end{aligned}$$

Proof

Since $\sigma_{k+1}(Y_k) = 0$, when $X' = X - Y_k$ and $X'' = Y_k$ we conclude that for $i \geq 1, j = k + 1$

$\sigma_i(X - Y_k) + \underbrace{\sigma_{k+1}(Y_k)}_{0} \geq \sigma_{k+i}(X)$. Therefore,

$$\|X - Y_k\|_F^2 = \sum_{i=1}^{n} \sigma_i(X - Y_k)^2 \geq \sum_{i=k+1}^{n} \sigma_i(X)^2 = \|X - X_k\|_F^2.$$

Low rank approximation of a matrix and projection

If $rank(\hat{X}) = k$, then we can assume columns $\hat{X}_i$ of $\hat{\boldsymbol{X}} \in E_k = span\{\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_k\}$ where $\{\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_k\}$ is a set of orthonormal vectors for the linear space of columns of X_k. First, observe that

$$\|\boldsymbol{X} - \hat{\boldsymbol{X}}\|_F^2 = \sum_i \|X_i - \hat{X}_i\|^2$$

Optimum solution is the orthogonal projection

For each term $\|X_i - v\|_2^2$,the optimum solution is the projection of X_i onto $E_k = span\{\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_k\}$:

$$\hat{X}_i = \sum_{j=1}^{k} \langle X_i, \boldsymbol{w}_j \rangle \boldsymbol{w}_j = \Pi_{E_k} X_i.$$

where $\Pi_{E_k} = \sum_{j=1}^{k} \boldsymbol{w}_j \boldsymbol{w}_j^T$

Projection on the orthogonal subspace

Consider $\Pi_{E_k^\perp}$ the projection matrix on the space orthogonal to E_k. More precisely, let us add $\boldsymbol{w}_{k+1}, \cdots, \boldsymbol{w}_n$ such that $\boldsymbol{w}_1, \cdots, \boldsymbol{w}_n$ form an orthonormal basis of $\mathbb{R}^n$. Then,

$$\Pi_{E_k^\perp} = \sum_{j=k+1}^{n} \boldsymbol{w}_j \boldsymbol{w}_j^T$$

$$\|X - \hat{X}\|_F^2 = \|X - \Pi_{E_k} X\|_F^2 = \|(I - \Pi_{E_k})X\|_F^2 = \|\Pi_{E_k^\perp} X\|_F^2$$

Relation to principal component analysis

Warning

$\boldsymbol{X}$ **is considered as centered**. This tranformation (cloud translation allows considerable simplification)

Decomposition of X

Considering the orthonal projection on E_k

$$\boldsymbol{X} = \Pi_{E_k}\boldsymbol{X} + \Pi_{E_k^\perp}\boldsymbol{X}$$

Criterion

$$\|\boldsymbol{X}\|_F^2 = \underbrace{\|\Pi_{E_k}\boldsymbol{X}\|_F^2}_{approximation} + \underbrace{\|\Pi_{E_k^\perp}\boldsymbol{X}\|_F^2}_{error}$$

In terms of intertia, PCA maximizes the projected inertia (approximation) while minimizing the ditances to the space of projection (error):

$$I_T = I_E + I_{E_k^\perp}$$

Best low rank approximation

$$\hat{\boldsymbol{X}}_k = \Pi_{E_k} \boldsymbol{X} = \sum_{j=1}^{k} \sigma_j \boldsymbol{u}_j \boldsymbol{v}_j^T = \boldsymbol{U}_{\bullet,1:k} \boldsymbol{S}_{1:k,1:k} \boldsymbol{V}_{\bullet,1:k}^T$$

where $\boldsymbol{X} = \underbrace{\boldsymbol{U}}_{n \times n} \underbrace{\boldsymbol{S}}_{n \times d} \overbrace{\boldsymbol{V}^T}^{d \times d}$

Different views of the approximation

The approximation

$$\|\boldsymbol{X} - \hat{\boldsymbol{X}}\|_F^2$$

can be considered in multiple ways:

- approximation of the row
- approximation of the columns

Notations

If $\boldsymbol{X}$ is a data table,

- each row $\boldsymbol{x}_i^T$ is a description of an individual
- each colum $\boldsymbol{X}_j$ is variable describing $\boldsymbol{n}$ individuals

Rows approximation (projection of the individuals)

Transposing the matrix the best low rank approximation becomes

$$\hat{\boldsymbol{X}^T}_k = \Pi_{F_k} \boldsymbol{X}^T = \sum_{j=1}^{k} \sigma_j \boldsymbol{v}_j \boldsymbol{u}_j^T = \boldsymbol{V}_{\bullet,1:k} \boldsymbol{S}_{1:k,1:k} \boldsymbol{U}_{\bullet,1:k}^T$$

where $F_k = span\{\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k\}$

The approximation error

$$\|\boldsymbol{X} - \hat{\boldsymbol{X}}\|_F^2 = \|\boldsymbol{X}^T - \hat{\boldsymbol{X}}^T\|_F^2 = \sum_i \|\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i\|_2^2$$

Each row $\boldsymbol{x}_i$ is approximated by

$$\hat{\boldsymbol{x}}_i = \Pi_{F_k} \boldsymbol{x}_i = V_{F_k} V_{F_k}^T \boldsymbol{x}_i$$

where $\boldsymbol{V}_{F_k}$ is the matrix composed of the vectors defining $\boldsymbol{F}_k$.

Projection of the variables

Π_{E_k} is the projection matrix on $E = span(\boldsymbol{u}_1, \cdots, \boldsymbol{u}_k)$

$$\Pi_E = \boldsymbol{U}_{E_k} \boldsymbol{U}_{E_k}^T$$

and

$$\Pi_{E_k} \boldsymbol{X} = \boldsymbol{U}_{1:n,1:k} \underbrace{\boldsymbol{U}_{1:n,1:k}^T \boldsymbol{U}_{1:n,1:n}}_{(I_k, 0_{k,n-k})} \boldsymbol{S}_{1:n,1:k} \boldsymbol{V}_{1:d,1:d}^T = \boldsymbol{U}_{1:n,1:k} \boldsymbol{S}_{1:k,1:k} \boldsymbol{V}_{1:d,1:k}^T$$

k first principal components

$$\boldsymbol{C}_{1:n,1:k} = \boldsymbol{U}_{E_k}\boldsymbol{S}_{1:k,1:k}$$

where $\boldsymbol{S}_{1:k,1:k} = diag(\sigma_1, \cdots, \sigma_k)$.

The principal component are the coordinates of the projection of the rows of $\boldsymbol{X}$ on F_k:

$$\boldsymbol{C}_{1:n,1:k} = \boldsymbol{X}\boldsymbol{V}_{1:d,1:k}$$

Percentage of information

We have $\boldsymbol{C}_{\bullet,1:k}^T \boldsymbol{C}_{\bullet,1:k} = S_{1:k,1:k}^2 = diag(\sigma_1^2, \cdots, \sigma_k^2)$, thus

$$\|\boldsymbol{C}_{\bullet,1:k}\|_F^2 = \sum_{j=1}^{k} \sigma_j^2$$

and

$$\frac{\|\boldsymbol{C}_{\bullet,1:k}\|_F^2}{\|\boldsymbol{X}\|_F^2} = \frac{\sum_{j=1}^{k} \sigma_j^2}{\sum_{j=1}^{d} \sigma_j^2} \in [0, 1]$$

Correlations

$$\widehat{cor}(\boldsymbol{X}_{\bullet,j}, \boldsymbol{C}_{\bullet,k}) = \frac{X_{\bullet,j}^T \boldsymbol{C}_{\bullet,k}}{\|X_{\bullet,j}\| \|\boldsymbol{C}_{\bullet,k}\|} = \cos\left(\widehat{\boldsymbol{X}_{\bullet,j}, \boldsymbol{C}_{\bullet,k}}\right)$$

Duality

It is easy to show that

- the columns of $\boldsymbol{V}$ are the eigenvector of $\boldsymbol{X}^T\boldsymbol{X}$
- the columns of $\boldsymbol{U}$ are the eigenvector of $\boldsymbol{X}\boldsymbol{X}^T$

Thus the principal component of $\boldsymbol{X}^T\boldsymbol{X}$ are the eigenvectors of $\boldsymbol{X}\boldsymbol{X}^T$ and vice-versa

Multi Dimensional Scaling

Dimensionality reduction and manifold learning

Goal

Given pairwise

- dissimilarities δ_{ij} or
- high-dimensional input data $\{\boldsymbol{x}_i\}_{i=1\cdots...n}$

reconstruct a lower dimensional map with embedded data $\{\boldsymbol{y}_i\}_{i=1\cdots...n}$

Principle

- Minimize an objective function quantiying the discrepancies between the δ_{ij} and the distances $d(\boldsymbol{y}_i, \boldsymbol{y}_j)$
- w.r.t $\{\boldsymbol{y}_i\}_{i=1\cdots...n}$.

A Brief history

The methodology of Multidimensional Positioning (Multidimensional Scaling, MDS) was born in the USA in the 1950s

- Works of Torgerson (1952, 1958),
- Works of Shepard (1962),

initial applications to sparse data in

- psychometrics,
- marketing,
- sensory analysis.

Proximity measures

Distance

The function d from $E \times E$ in $\mathbb{R}^+$ is a distance if it satisfies the following properties:

- $\forall i, j \in E,\ d_{ij} = d_{ji}$
- $\forall i \in E,\ d_{ii} = 0$
- $\forall i, j\ and\ k \in E,\ d_{ik} \leq d_{ij} + d_{jk}$

Dissimilarity

The function δ of $E \times E$ in $\mathbb{r}^+$ is a distance if it satisfies the following properties:

- $\forall i, j \in E,\ \delta_{ij} = \delta_{ji}$
- $\forall i \in E,\ \delta_{ii} = 0$

Classical scaling

Also known as Torgerson scaling, principal coordinate analysis (PCoA), …

Principle

- assume that the dissimilarities are distances and
- find the main coordinates that explain the distances.

From distances to scalar product

- How to calculate the distance matrix from X?

$$d_{ij}^2 = \sum_{a=1}^{p}(x_{ia}^2 + x_{ja}^2 - 2x_{ia}x_{ja})$$

$$= \sum_{a=1}^{p} x_{ia}^2 + \sum_{a=1}^{p} x_{ja}^2 - 2\sum_{a=1}^{p} x_{ia}x_{ja}$$

from where

$$D^2 = diag(XX^t)\mathbb{I}_{(1,n)} + \mathbb{I}_{(n,1)}diag(XX^t)^t - 2XX^t$$

Distance and data table (continued)}

- Let J be the centering matrix,

$$J = I - \frac{1}{n}\mathbb{I}_{(n,n)}$$

- Centering of X in columns

$$JX = X - \frac{1}{n}\mathbb{I}_{(n,n)}X$$

- Centering of X in rows

$$X^tJ = X^t - \frac{1}{n}X^t\mathbb{I}_{(n,n)}X$$

Distance and data table (continued)}

Problem: we know D^2, we want X: Double centering

$$\begin{aligned} -\frac{1}{2}JD^2J = & -\frac{1}{2}Jdiag(XX^t) * \mathbb{I}_{(1,n)}J \\ & -\frac{1}{2}J\mathbb{I}_{(n,1)}diag(XX^t)^tJ \\ & +JXX^tJ \\ = & XX^t \end{aligned}$$

Low rank approximation and principal coordinates

The idea is to minimize

$$\|XX^T - YY^T\|_F^2$$

w.r.t $rank(YY^T) = k$

We know the minimum is a low rank apprixation $B = YY^T$ of rank k

$$B = (U_{E_k} S_{1:k,1:k})(S_{1:k,1:k}^T U_{E_k}^T) = YY^T$$

- the analysis of the triple consists in replacing the matrix D^2 by the matrix of the square of dissimilarities Δ^2

Remark

If Δ^2 is a distance matrix the solution $Y = U_{E_k} S_{1:k,1:d}$ (first k principal components of $X^T X$) is optimal and PCoA is equivalent to PCA.

Algorithm

1. Compute the matrix Δ^2
2. Double centering of Δ^2:

$$B_{\Delta^2} = -\frac{1}{2} J \Delta^2 J$$

3. Spectral decomposition of B_{Δ^2}~:

$$B_{\Delta^2} = U \Lambda U^t$$

4. Let k be the representation dimension chosen for the solution.

- Λ_+ is the matrix of k largest eigenvalues, listed in descending order on the diagonal.
- U^+ is the matrix of k corresponding eigenvectors
- **The solution of the problem** is

$$Y = U^+ \Lambda_+^{\frac{1}{2}}$$

Optimized criterion

- the optimized criterion is

$$\begin{aligned} L(X) &= || - \frac{1}{2} J(D^2(X) - \Delta^2)J||^2 \\ &= ||XX^t + \frac{1}{2} J\Delta^2)J||^2 \\ &= ||XX^t - B_{\Delta^2}||^2 \end{aligned}$$

  with B_{Δ^2} of rank k
- Remarks: If Δ is a dissimilarity matrix then some principal components will be negative!

Models and functions for the MDS

MDS matches proximities δ_{ij} to distances d_{ij} between objects:

$$f : \delta_{ij} \longrightarrow d_{ij}(Y)$$

with Y a classical array of low dimensional data - the dissimilarities δ_{ij} are the data of the pb - the distances $d_{ij}(Y)$ are the unknowns

- the MDS finds a configuration Y in a space of dimension m, used to calculate distances d_{ij}

In practice

the dissimilarities are marred by errors and one does not look for f such that

$$f(\delta_{ij}) = d_{ij}(Y)$$

but rather

$$f(\delta_{ij}) \approx d_{ij}(Y)$$

Error function

Definition of an error function

$$e_{ij} = \left(f(\delta_{ij}) - d_{ij}(Y)\right)^2$$

Raw global error (raw stress)

$$\sigma_r^2(Y) = \sum_{i>j} \left(f(\delta_{ij}) - d_{ij}(Y)\right)^2$$

- The global error is not very informative~: A large value of $\sigma_r^2(Y)$ does not necessarily indicate bad result.

Normalized error function

Normalized Stress

$$\sigma_n^2(Y) = \frac{\sum_{i>j} \Big(f(\delta_{ij}) - d_{ij}(Y) \Big)^2}{\sum_{i>j} d_{ij}^2(Y)}$$

A general form

$$\sigma^2(Y) = \sum_{i>j} w_{ij} \Big(f(\delta_{ij}) - d_{ij}(Y) \Big)^2$$

Different approaches

- metric: quantitative approach
- non-metric : qualitative approach

Metric approach

Algebraic transformation of δ_{ij} (e.g. $f(\delta_{ij}) = a\delta_{ij} + b$)

With

$$f(\delta_{ij}) = \delta_{ij} + b(1 - \mathbb{I}_{(i=j)})$$

assurance of the existence of an Y configuration of dimension $n - 2$ at most such that

$$f(\delta_{ij}) = d_{ij}(Y)$$

Non-metric approach

- f is a monotone function that preserves order relations
- if $\delta_{ij} < \delta_{kl}$ then $f(\delta_{ij} < f(\delta_{kl}))$

Sammon's Projection

Metric Method: "Projection" into a space of low dimension (1,2, or 3)

- $Y = (\mathbf{y}_1, \ldots, \mathbf{y}_n)^t$ configuration sought
- Euclidian distance

$$\begin{aligned} d_{ij} &= d(\mathbf{y}_i, \mathbf{y}_j) \\ &= \|\mathbf{y}_i - \mathbf{y}_j\|_2 \end{aligned}$$

Sammon stress function

Sammon searches for the $\mathbf{y}_i$ minimizing

$$S(Y) = \frac{1}{\sum_{i<j} \delta_{ij}} \frac{\sum_{i<j} (\delta_{ij} - d_{ij}(Y))^2}{\delta_{ij}}$$

- f identity
- $w_{kl} = \frac{1}{(\sum_{i<j} \delta_{ij}) \delta_{kl}}$

Remarks:

Sammon's Stress

- is invariant to rotations, translation and scaling
- focuses on small distances

Minimisation of Sammon's Stress

- minimisation of a function from $\mathbb{R}^{n \times k}$ to $\mathbb{R}$
- Global minimum is difficult to reach
- Gradient descent

Remarks

- Many possible optimmisation methods
- Initial Random configuration

Example in R

sammon from library MASS

```
   Sammon's Non-Linear Mapping

Description
 One form of non-metric multidimensional scaling.

Usage
 sammon(d, y = cmdscale(d, k), k = 2, niter = 100, trace = TRUE,
 magic = 0.2, tol = 1e-4)
```

Sammon gradient

Let show that

$$\frac{\partial S(Y)}{\partial y_{ik}} = -2 \sum_{j<i} w_{ij} \frac{\delta_{ij} - d_{ij}}{d_{ij}} (y_{ik} - y_{jk})$$

From chain rule

$$\frac{\partial S(Y)}{\partial y_{ik}} = \underbrace{\frac{\partial S(Y)}{\partial \|y_i - y_j\|}}_{1} \times \underbrace{\frac{\partial \|y_i - y_j\|}{\partial y_{ik}}}_{2}$$

1. $\frac{\partial S(Y)}{\partial d_{ij}} == -2 \sum_{j<i} w_{ij}(\delta_{ij} - d_{ij})$
2. $\frac{\partial \|y_i - y_j\|}{\partial y_{ik}} = \frac{\partial d_{ij}}{\partial d_{ij}^2} \frac{\partial d_{ij}^2}{\partial y_{ik}} = \frac{y_{ik} - y_{jk}}{d_{ij}}$

Sammon gradient descent

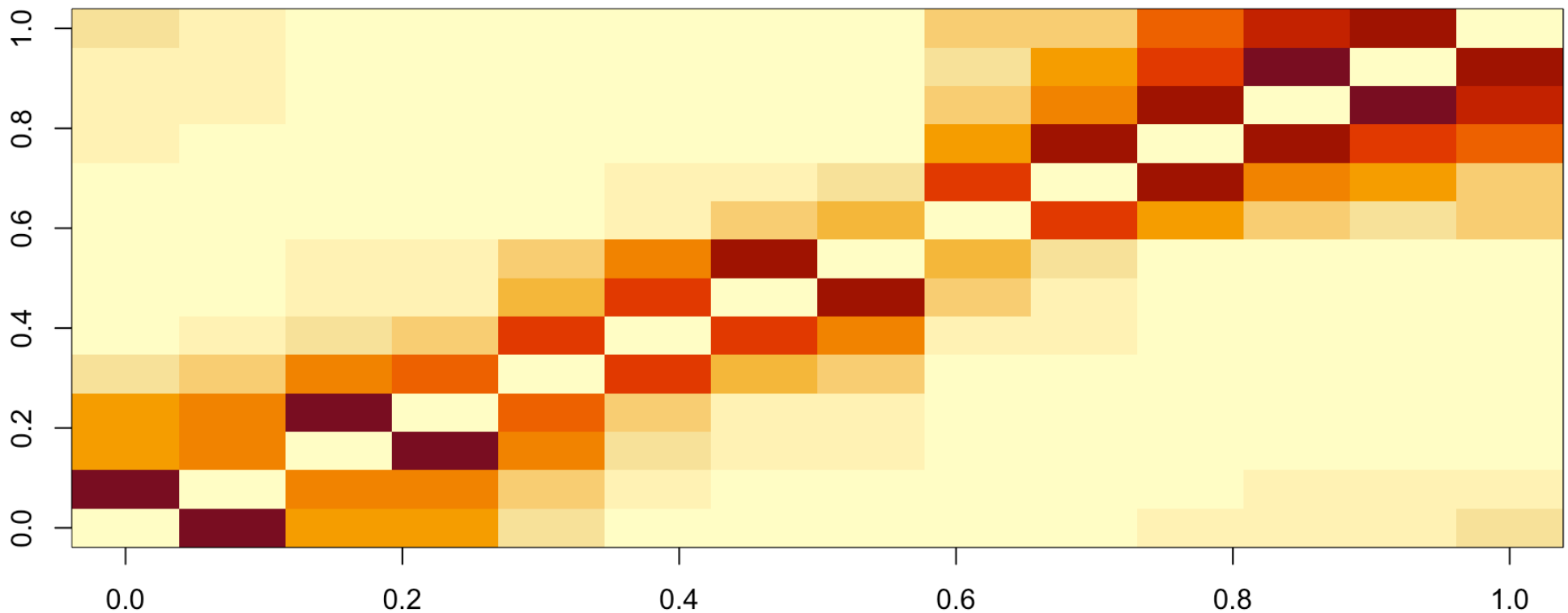
At iteration q

$$y_{ik}^{(q+1)} := y_{ik}^{(q)} - \eta \frac{\partial S(Y)}{\partial y_{ik}}$$

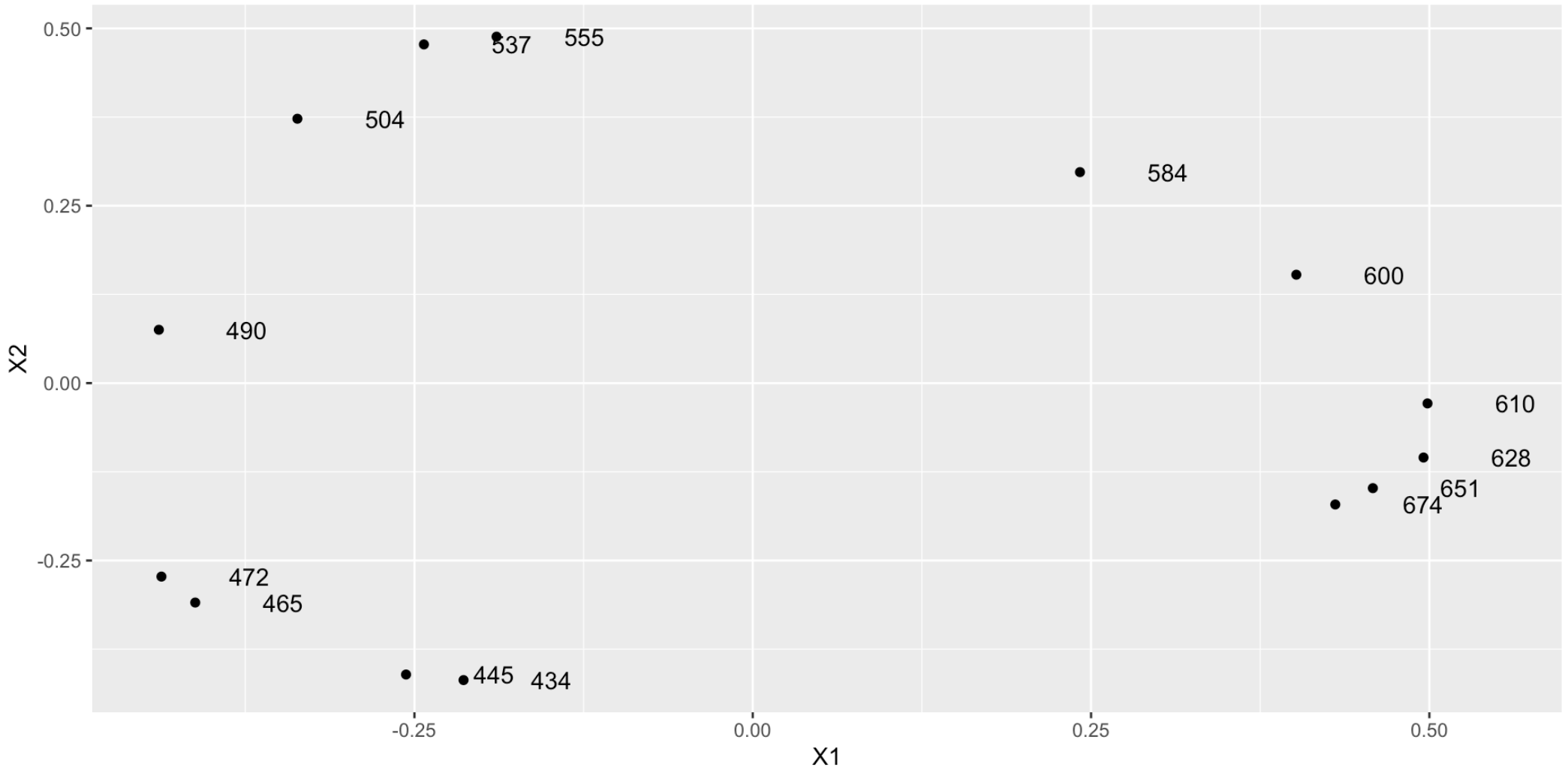
Example Ekman colors

Ekman presents similarities for 14 colors which are based on a rating by 31 subjects where each pair of colors was rated on a 5-point scale (0 = no similarity up to 4 = identical). After averaging, the similarities were divided by 4 such that they are within the unit interval. Similarities of colors with wavelengths from 434 to 674 nm.

Example Ekman colors



Example Classical scaling



t-SNE

Variation of Stochastic Neighbor Embedding

- easier to optimize,
- produces significantly better visualizations by reducing the tendency
- scalable: for very large data sets, t-SNE can use random walks on neighborhood graphs

Stochastic Neighbor Embedding

Stochastic Neighbor Embedding (SNE) converts distances into conditional probabilities that represent similarities

Conditional probabilities $p_{j|i}$

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/(2\sigma_i^2))}{\sum_{k\neq i}\exp(-\|x_i - x_k\|^2/2\sigma_i^2)}\,.$$

Interpretation

Probability that x_i would pick x_j as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at x_i:

Stochastic Neighbor Embedding

similar conditional probability between the y_i, which we denote by $q_{j|i}$.

Conditional probabilities, $q_{j|i}$

$$q_{j|i} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq i} \exp(-\|y_i - y_k\|^2)} \,.$$

with $q_{i|i} = 0$.

SNE can also be applied directly to similarity data

provided similarities can be interpreted as conditional probabilities we set

Cost function

A natural measure of the faithfulness with which $q_{j|i}$ models $p_{j|i}$ is the Kullback-Leibler divergence (equal to the cross-entropy up to an additive constant).

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}} \ ,$$

the SNE cost function focuses on retaining the local structure of the data in the map (for *reasonable values of the variance* of the Gaussian in the high-dimensional space).

Selecting the variance σ_i

SNE performs a binary search for the value of σ_i that produces a P_i with a fixed perplexity that is specified by the user

Perplexity

$$Perp(P_i) = 2^{H(P_i)},$$

where $H(P_i)$ is the Shannon entropy of P_i measured in bits

$$H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}.$$

Remarks

- the perplexity increases monotonically with the variance σ_i
- smooth measure of the effective number of neighbors
- Typical values are between 5 and 50.

Gradient descent

Gradient

$$\frac{\partial C}{\partial y_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j) \, .$$

Interpretation

The gradient may be interpreted as the resultant force created by a *set of springs* between the map point y_i and all other map points y_j.

- force exerted by the spring between y_i and y_j is proportional to its length
- also proportional to its stiffness $(p_{j|i} - q_{j|i} + p_{i|j} + q_{i|j})$

Initialisation

n points randomly sampled from an isotropic Gaussian with small variance that is centered around the origin

Gradient in practice

- To speed up the optimization and to avoid poor local minima, a relatively large momentum term is added to the gradient:

$$\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\partial C}{\partial \mathcal{Y}} + \alpha(t) \left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)} \right),$$

where $\mathcal{Y}^{(t)}$ indicates the solution at iteration t, η indicates the learning rate, and $\alpha(t)$ represents the momentum at iteration t.

- In addition, in the early stages of the optimization, Gaussian noise is added to the map points after each iteration. Gradually reducing the variance of this noise performs a type of simulated annealing

t-Distributed Stochastic Neighbor Embedding

- cost function of SNE is difficult to optimize
- t-SNE suffers from the “crowding problem”

t-SNE vs SNE

1. Uses a symmetrized version of the SNE cost function with simpler gradients
2. Student-t distribution rather than a Gaussian to compute the similarity between two points *in the low dimensional space*.

Symmetric SNE

$$C = KL(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \,,$$

where p_{ii} and q_{ii} to zero.

- $p_{ij} = p_{ji}$ and
- $q_{ij} = q_{ji}$ for all i, j.

Joint probabilies p_{ij}

$$p_{ij} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma^2)}{\sum_{k \neq l} \exp(-\|x_k - x_l\|^2/2\sigma^2)}$$

Joint probabilies q_{ij}

$$q_{ij} = \frac{\exp(-\|x_i - x_j\|^2)}{\sum_{k \neq l} \exp(-\|x_k - x_l\|^2)} \,.$$

t-SNE

Problems when a high-dimensional datapoint x_i is an outlier, the values of p_{ij} are extremely small for all j, so the location of its low-dimensional map point y_i has very little effect on the cost function.

- $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$. ensures that $\sum_j p_{ij} > \frac{1}{2n} \; \forall x_i$,
- each x_i makes a significant contribution to the cost function.

t-SNE Gradient

The main advantage of the symmetric version of SNE is the simpler form of its gradient, which is faster to compute.

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j) \, .$$

symmetric SNE produces maps that are just as good as asymmetric SNE

The Crowding Problem

> the area of the two-dimensional map that is available to accommodate moderately distant datapoints will not be nearly large enough compared with the area available to accommodate nearby datapoints.

- Modeling the small distances accurately has detrimental effect on moderate distances representation
- the crowding problem is not specific to SNE

Mismatched tails can compensate for mismatched dimensionalities

t-distribution for the q_{ij}

Student distribution with a single degree of freedom also known as a Cauchy distribution

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l}(1 + \|y_k - y_t\|^2)^{-1}}$$

Gradient

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1}. \quad (1)$$

Two optimization tricks

"Early compression"

- Force the map points to stay close together at the start of the optimization.
- Easy for clusters to move through one another and thus to explore the space of solutions
- implemented by adding an additional L2-penalty to the cost function that is proportional

"Early exaggeration"

Multiply all of the p_{ij}'s by, for example, 4, in the initial stages of the optimization.

- natural clusters in the data tend to form tight widely separated clusters in the map.
- creates a lot of relatively empty space in the map
- makes it much easier for the clusters to move around relative to one another

Uniform manifold approximation and projection (UMAP)

UMAP works directly with similarities and uses cross-entropy as a criterion.

Distance in high dimensional space

$$v_{j|i} = \exp\left[(-d(x_i, xj) - \rho_i)/\sigma_i\right]$$

- $d(x_i, x_j)$ is the distance between x_i and $x_j)$, which UMAP does not require to be Euclidean.
- ρ_i is the distance to the nearest neighbor of i.
- σ_i is the normalizing factor, which is chosen by a specific Algorithm and plays a similar role to the perplexity-based calibration of σ_i in t-SNE.

Symmetrization is carried out by fuzzy set union

$$v_{ij} = v_{j|i} + v_{i|j} - v_{j|i}v_{i|j}$$

Uniform manifold approximation and projection (UMAP)

Low dimensional similarities

$$w_{ij} = (1 + a\|y_i - y_j\|_2^{2b})^{-1}$$

where a and b are user-defined positive values

Cost function (cross-entropy)

$$C_{UMAP} = \sum_{i \neq j} v_{ij} \log \frac{v_{ij}}{w_{ij}} + (1 - v_{ij}) \log \frac{1 - v_{ij}}{1 - w_{ij}}$$

Optimization process is done by stochastic gradient descent

UMAP vs t-SNE

In terms of performance,

- UMAP is often considered to produce more stable and consistent results than t-SNE,
- especially when dealing with large datasets.

On the other hand,

- t-SNE is often considered to produce more visually appealing results,
- especially when dealing with small datasets.

Word Embedding

Word embedding

Words and Vectors

Vector or distributional models of meaning are generally based on a co-occurrence matrix

Two common co-occurrence matrix

- the word-document matrix
- the word-word matrix.

Word-document matrix

In a Word-document matrix, each row represents a word in the vocabulary and each column represents a document from some collection of documents.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Number of words occurence in 4 Shakespeare plays (from speech and langage processing)

Word-Word matrix

In the Word-context matrix the columns are labeled by words rather than documents.

- each cell records the number of times the row (target) word and the column (context) word co-occur in some context in some training corpus.

Context

- a context could be the document, in which case the cell represents the number of times the two words appear in the same document.
- smaller contexts are common: generally a window around the word

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

word-word matrix from a Wikipedia corpus (from speech and langage processing)

Cosine for measuring similarity

The cosine similarity metric (or empirical correlation) between two word vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ (lines of a word-document or word-word matrix)

$$\cos(\boldsymbol{v}, \boldsymbol{w}) = \frac{\boldsymbol{v}^T \boldsymbol{w}}{\|\boldsymbol{v}\| \|\boldsymbol{w}\|}$$

TF-IDF: Weigthing terms in the vector

Term frequency (tf)

- Raw frequency $tf_{t,d} = count(t, d)$ is very skewed and not very discriminative.
- Usually squashed by using the $log10$: a word appearing 100 times in a document doesn't make that word 100 times more relevant

$$\mathrm{tf}_{t,d} = \log_{10}\left(count(t, d) + 1\right)$$

Inverse document frequency

The second factor in tf-idf is used to give a higher weight to words that occur only in a few documents.

$$idf_t = \log_{10}\left(\frac{N}{\mathrm{df}_t}\right)$$

where N is the total number of documents, and df_t is the number of documents in which term t occurs.

TF-IDF: Weigthing terms in the vector

The tf-idf weighted value $w_{t,d}$ for word t in document d thus combines term frequency $tf_{t,d}$ with idf:

$$w_{t,d} = tf_{t,d} \times idf_t$$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

A tf-idf weighted term-document matrix. Note that the tf-idf weighting has eliminated the importance of the ubiquitous word good and vastly reduced the impact of the almost-ubiquitous word fool.

Pointwise Mutual Information (PMI)

An alternative weighting function to tf-idf, PPMI (positive pointwise mutual information), is used for **term(target)-term(context)-matrices**.

- Target word, word we focus on.
- Context words surrounding the target word

The pointwise mutual information between a target word w and a context word c (Church and Hanks 1989, Church and Hanks 1990) is defined as:

$$PMI(w, c) = \log_2 \frac{P(w, c)}{P(w)P(c)}$$

Positive PMI (called PPMI)

Negative PMI values (which imply things are co-occurring less often than we would expect by chance) tend to be unreliable unless our corpora are enormous.

Positive PMI

replaces all negative PMI values with zero (Church and Hanks 1989, Dagan et al. 1993, Niwa and Nitta 1994)

$$PPMI(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0)$$

	computer	data	result	pie	sugar
cherry	0	0	0	4.38	3.30
strawberry	0	0	0	4.10	5.51
digital	0.18	0.01	0	0	0
information	0.02	0.09	0.28	0	0

A PPMI matrix showing the association between words and context words

Applications of the tf-idf or PPMI vector models

Principle

- Computing two documents (or word) similarity after transformation.
- Given two documents d_1 and d_2, the similarity is $\cos(d_1, d_2)$.

Applications

- **Documents** : information retrieval, plagiarism detection, news recommender systems, and even for digital humanities tasks like comparing different versions of a text to see which are similar to each other.
- **Words** : finding word paraphrases, tracking changes in word meaning, or automatically discovering meanings of words in different corpora.

Latent Semantic Analysis (LSA)

Goal and assumptions

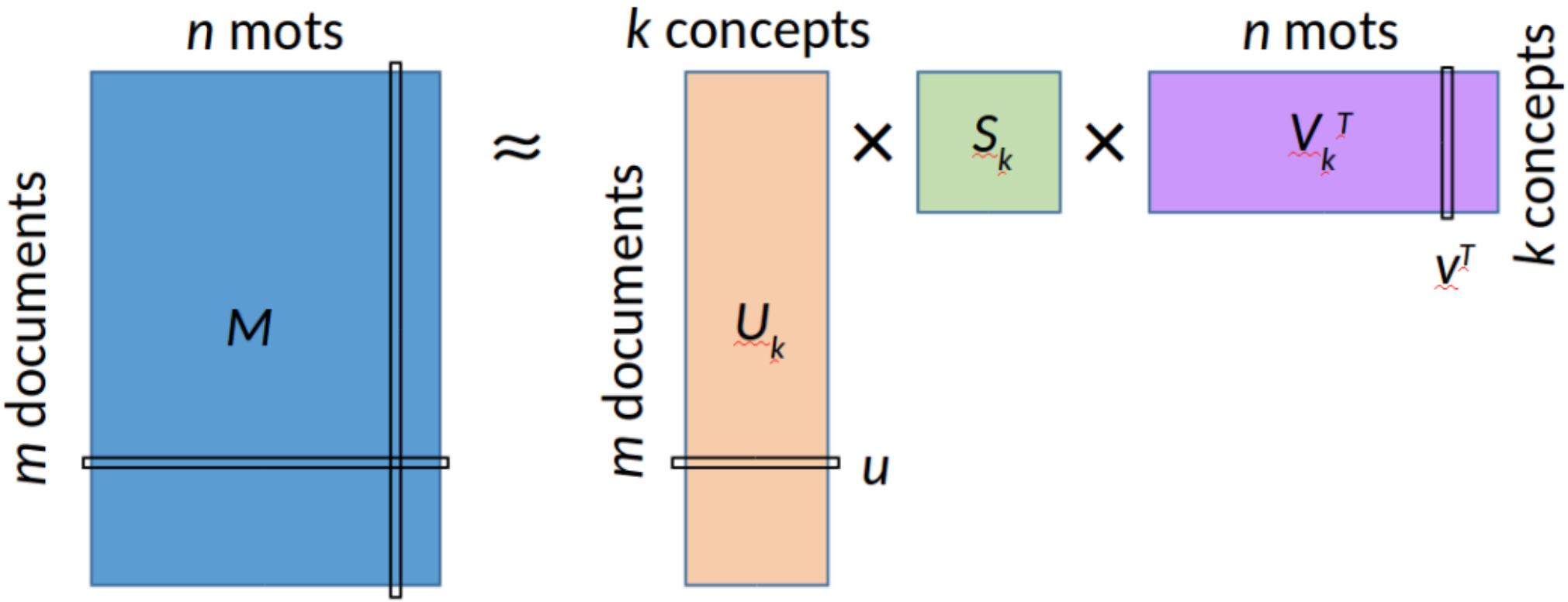
- extracting and representing the underlying meaning of words in a corpus of texts.
- words that occur in similar contexts have similar meanings.

Principle

LSA uses singular value decomposition (SVD) to identify latent, or hidden, patterns in word co-occurrence data.

Latent Semantic Analysis

LSA uses SVD on the matrix of word-document matrix to identify the latent concepts (contexts, topics, …) that underlie the relationships between words in the corpus: $W = USV^T$ with classical low rank approximation $U_k S_k V_k^T$



Latent Semantic Analysis

- The values on the main diagonal of S_k indicate the 'importance' of each of the k main 'latent concepts' (or factors).
- For each of the document, the corresponding row of U_k allows us to see which concepts are present and with what weights.
- For each of the concept, the associated column of V_k indicates which terms form the concept (and with what weights).

Calculating the similarity between a term and a document involves choosing:

- the row $\mathbf{u}$ corresponding to the document in the matrix $\mathbf{U}_k$,
- the row $\mathbf{v}$ corresponding to the term in the matrix $\mathbf{V}_k$,
- and then computing the product $\mathbf{u}\mathbf{S}_k\mathbf{v}^T$.

To determine the similarity of a term to each of the documents

$$\mathbf{U}_k\mathbf{S}_k\mathbf{v}^T$$

Latent Dirichlet Allocation (LDA)

History

In the context of population genetics, LDA was proposed by J. K. Pritchard, M. Stephens and P. Donnelly in 2000.

LDA was applied in machine learning by David Blei, Andrew Ng and Michael I. Jordan in 2003.

Principle

Latent Dirichlet Allocation (LDA) is a probabilistic generative model that assumes that

- each document is generated by a mixture of latent topics
- each topic is generated by a mixture of words from the vocabulary.

Documents are represented as random mixtures over latent topics, where each topic is characterized by a distribution over all the words.

LDA Generative process

For a corpus D consisting of M documents

1. Choose topic parameter vector for document i: $\theta_i \sim \mathbf{Dir}(\alpha)$, where $i \in \{1, \ldots, M\}$ and $\mathbf{Dir}(\alpha)$ is a Dirichlet distribution with a symmetric parameter α which typically is sparse (α).
2. Choose the word parameters vector of topic k: $\varphi_k \sim \mathbf{Dir}(\beta)$, where $k \in \{1, \ldots, K\}$ and β typically is sparse
3. For each of the word positions i, j, where $i \in \{1, \ldots, M\}$, and $j \in \{1, \ldots, N_i\}$

a. Choose a topic $z_{i,j} \sim \mathbf{Cat}(\theta_i)$.

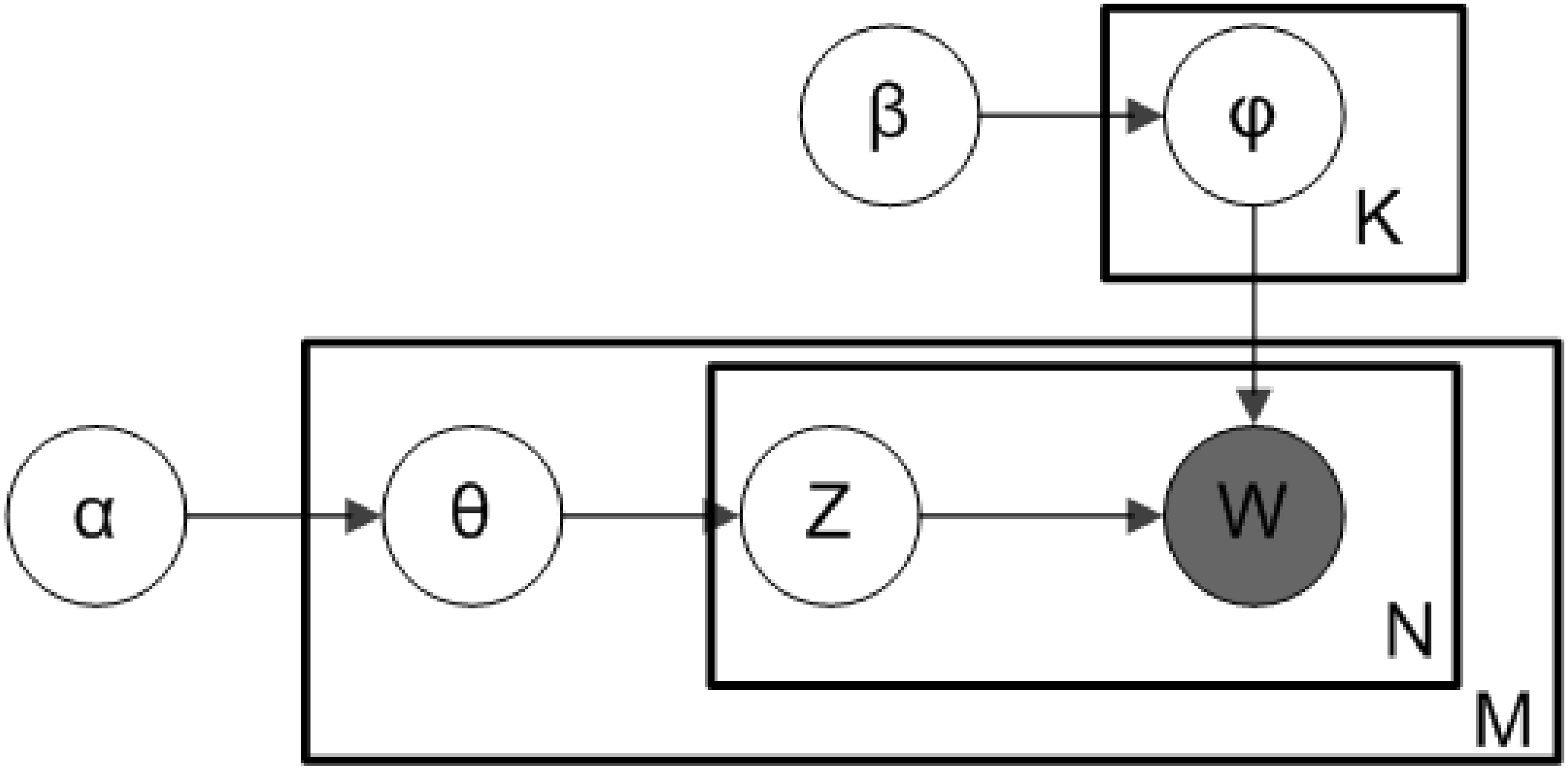
b. Choose a word $w_{i,j} \sim \mathbf{Cat}(\varphi_{z_{i,j}})$.

Latent Dirichlet Allocation (LDA) in summary

$$
\begin{aligned}
\boldsymbol{\varphi}_{k=1\ldots K} &\sim \text{Dirichlet}_V(\boldsymbol{\beta}) \\
\boldsymbol{\theta}_{d=1\ldots M} &\sim \text{Dirichlet}_K(\boldsymbol{\alpha}) \\
z_{d=1\ldots M, w=1\ldots N_d} &\sim \text{Categorical}_K(\boldsymbol{\theta}_d) \\
w_{d=1\ldots M, w=1\ldots N_d} &\sim \text{Categorical}_V(\boldsymbol{\varphi}_{z_{dw}})
\end{aligned}
$$

$$
P(\boldsymbol{W}, \boldsymbol{Z}, \boldsymbol{\theta}, \boldsymbol{\varphi}; \alpha, \beta) = \prod_{i=1}^{K} P(\varphi_i; \beta) \prod_{j=1}^{M} P(\theta_j; \alpha) \prod_{t=1}^{N} P(Z_{j,t} \mid \theta_j) P(W_{j,t} \mid \varphi_{Z_{j,t}}),
$$

Latent Dirichlet Allocation (LDA) estimation



Estimating the parameters could be achieved through Variationnal Bayes EM algorithm.

Example of LDA in Action

Corpus of Three Documents

1. “cats dogs pets love”
2. “dogs bark loud outside”
3. “politics government election law”

Step 1: Identify Topics

After running LDA, it may discover two topics (dist. over words): - Topic 1 (Pets): {cats, dogs, pets, bark, love} - Topic 2 (Politics): {politics, government, election, law}

Step 2: Assign Topics to Documents

```
- Document 1 -> $90\%$ Topic 1, $10\%$ Topic 2
- Document 2 -> $80\%$ Topic 1, $20\%$ Topic 2
-   Document 3 -> $95\%$ Topic 2, $5\%$ Topic 1
```

Pro and Cons

Advantages of LDA

- Unsupervised Learning: No labeled data needed.
- Interpretable Topics: Extracts meaningful topics from text.

Limitations of LDA

- Fixed Number of Topics: Must specify the number of topics beforehand.
- Bag-of-Words Assumption: Ignores word order and context.
- Computationally Expensive: Requires approximation methods for inference.

Applications of LDA

- Topic Modeling: Organizing large text corpora (news articles, research papers).
- Recommendation Systems: Suggesting articles based on topic similarity.
- Text Classification: Assigning labels based on topic distributions.
- Social Media Analysis: Identifying trending topics in posts.

Word2vec

Embeddings

- more powerful word representation
- short dense vectors: d ranging from 50-1000, rather than the much larger vocabulary size $|V|$ or number of documents D
- dense vectors work better in every NLP task than sparse vectors

Skip-gram with negative sampling

- The skip-gram algorithm is one of two algorithms in a software package called word2vec
- The other algorithm is CBOW (continuous bag of words)

Embedding derived from classification

Data : words and their context (neighboring words ($\pm L$)

… lemon, a [tablespoon of apricot jam, a] pinch … c1 c2 w c3 c4

Classification task

Given a tuple $(\boldsymbol{w}, \boldsymbol{c}) \in \mathbb{R}^d \times \boldsymbol{R}^d$ of a target word $\boldsymbol{w}$ paired with a candidate context word $\boldsymbol{c}$ (for example (apricot, jam), or perhaps (apricot, aardvark)) return the probability that $\boldsymbol{c}$ is a real context word (true for jam, false for aardvark):

$$P(y_{wc} = 1|\boldsymbol{w}, \boldsymbol{c}) = 1 - P(y_{wc} = 0|\boldsymbol{w}, \boldsymbol{c})$$

Similarity

we rely on the intuition that two vectors are similar if they have a high dot product (after all, cosine is just a normalized dot product). In other words:

$$Similarity(\boldsymbol{w}, \boldsymbol{c}) = \boldsymbol{c}^T \boldsymbol{w}$$

From dot product to logistic regression

Logit

$$P(y_{wc} = 1|\boldsymbol{w}, \boldsymbol{c}) = \frac{1}{1 + \exp(-\boldsymbol{c}^T \boldsymbol{w})}$$

Criterion

The criterion is a kind of likelihood including positive examples (observed association) and negative examples (non observed associations). In fact skip-gram with negative sampling (SGNS) uses more negative examples than positive examples (with the ratio between them set by a parameter $\boldsymbol{k}$).

$$L = \sum_{\boldsymbol{w},\boldsymbol{c}:y_{wc}=1} \log P(y_{wc} = 1|\boldsymbol{w}, \boldsymbol{c}) + \sum_{\boldsymbol{w}',\boldsymbol{c}':y_{w'c'}=0} \log P(y_{w'c'} = 0|\boldsymbol{w}', \boldsymbol{c}')$$

Example

- The learning algorithm for skip-gram embeddings takes as input a corpus of text
- random embedding vector initialization for each of the N vocabulary words
- iteratively shift the embedding of each word $\boldsymbol{w}$ to be more like the context words

Let's start by considering a single piece of training data:

… lemon, a [tablespoon of apricot jam, a] pinch …

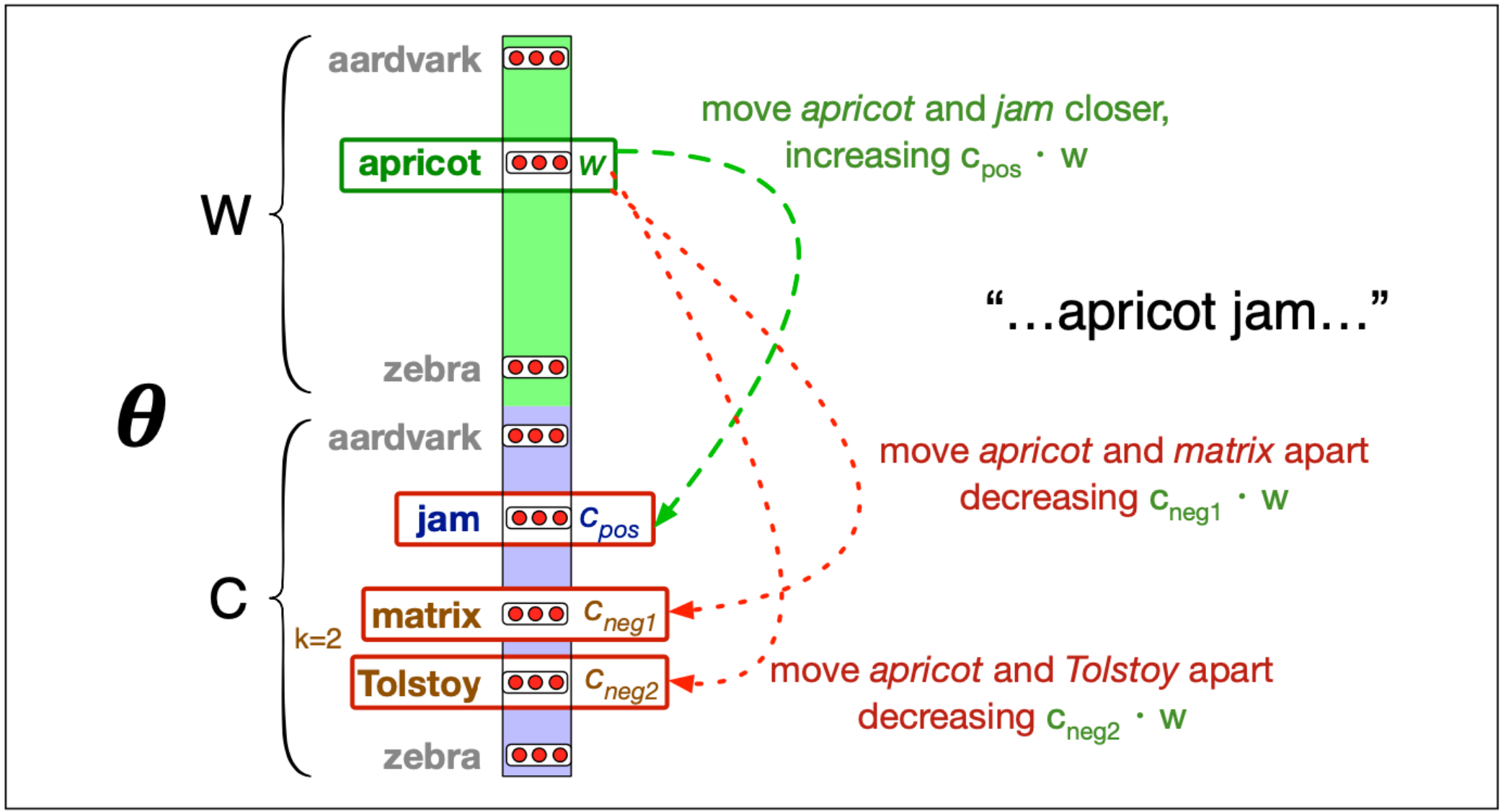
positive examples +

w	c_{pos}
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

w	c_{neg}	w	c_{neg}
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

Skip-gram model learns two separate embeddings



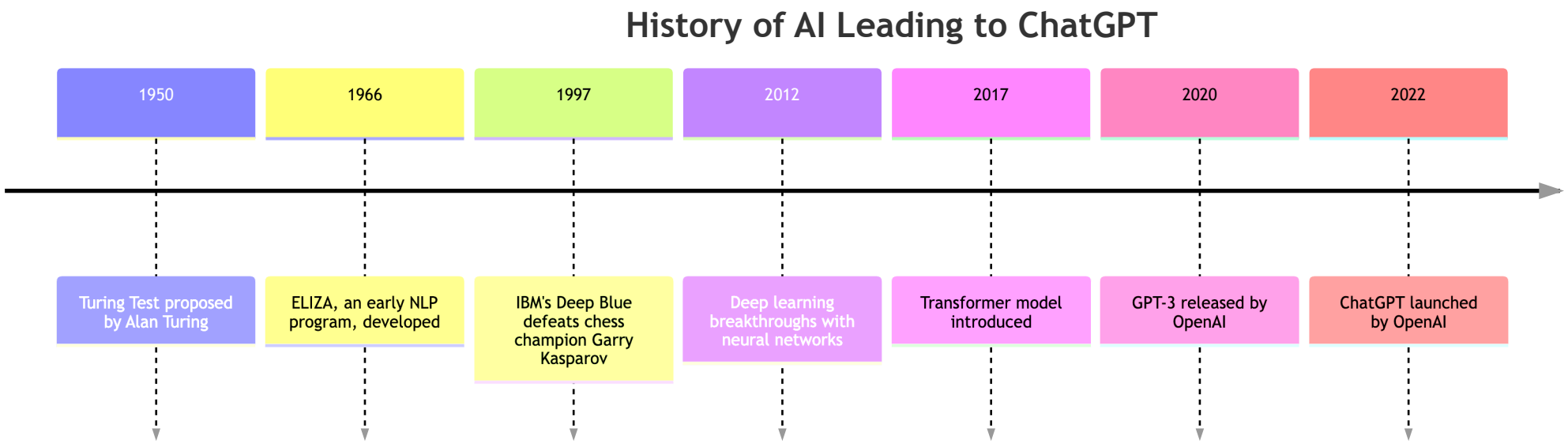
Final embeddings

- skip-gram outputs the target matrix W and the context matrix C.
- It's common to just add them together, representing word i with the vector $\boldsymbol{w}_i + \boldsymbol{c}_i$.
- Alternatively we can throw away the C matrix and just represent each word i by the vector $\boldsymbol{w}_i$.
- As with the simple count-based methods like tf-idf, the context window size L affects the performance of skip-gram embeddings, and experiments often tune the parameter L on a devset.

Transformers

The nature of transformers

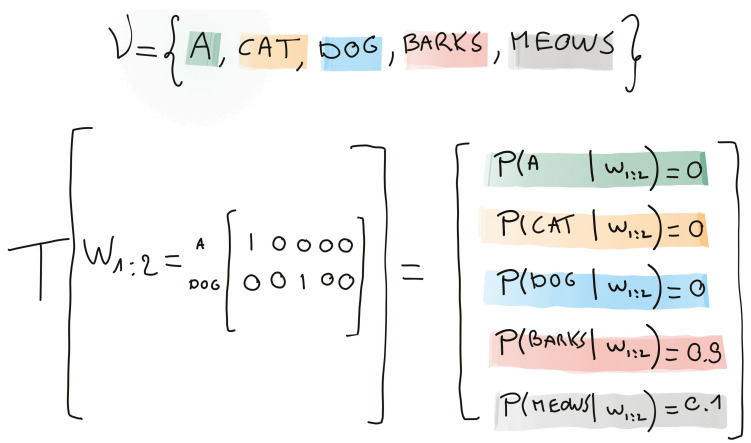
- Deep learning architecture based on **attention mechanisms**
- Weight the importance of different tokens in a sequence to **model long-range dependencies efficiently**



The nature of transformers

A classical transformer is a function

$$T(w_{1:n}) = \begin{bmatrix} p_1 \\ \vdots \\ p_{|V|} \end{bmatrix}$$



- $\mathcal{V}$ is a finite set (vocabulary) (Open AI uses 50,000 token as vocabulary)
- $w_{1:n}$ be a sequence of token, where each $w_i \in \mathcal{V}$ (In practice context varies from 1 to n, the blocksize)
- p_j is the probability of $w_{(j)}$ the jth token in $\mathcal{V}$

Remarks

- $w_{1:n}$ is usually coded a matrix of one-hot vectors with $w_{1:n} \in \mathbb{R}^{n \times |V|}$.
- For classical chatbot $p_j = P(w_{n+1} = w_{(j)} | w_{1:n})$ where $w_{(j)}$ is the jthe element of $\mathcal{V}$.

Optimized criteria

Transformers typically optimize different criteria depending on the task they are trained on.

Causal Language Modeling (CLM) / Autoregressive Loss

- **Used in:** GPT models
- **Objective:** Predict the next token given the previous tokens

The training set is a set of sequences (s_j) of n tokens $s_j = (w_1^j, w_2^j, \ldots, w_n^j)$:

$$L_{CLM} = -\sum_j \sum_{t=1}^{n} \sum_{c \in V} \mathbb{1}_{w_t^j = c} \log P(w_t^j = c | w_{1:t-1}^j)$$

where $w_{1:t-1}^j$ denotes the tokens preceding position t in the sequence j

The criterion is a kind of pseudo log-likelihood.

If nothing is learned each token has the same probability of appearing: the largest possible value of the criterion is

$$-\log \frac{1}{|V|}$$

Masked Language Modeling (MLM)

- **Used in:** BERT models
- **Objective:** Predict randomly masked tokens in a sentence.

Let $M_j \subset \{1, \ldots, n\}$ be the set of masked positions in the sequence (s_j):

$$L_{MLM} = -\sum_j \sum_{t \in M_j} \sum_{c \in V} \mathbb{I}_{w_t^j = c} \log P(w_t^j = c | w^j_{(1:n) \setminus M})$$

Conditional Language Modeling / Translation Loss

- **Used in:** T5, BART
- **Objective:** Predict an output sequence given an input sequence (e.g., translation, summarization).

Given input sequence $w^j = (w_1^j, \ldots, w_n^j)$ and output sequence $v^j = (v_1^j, \ldots, v_m^j)$:

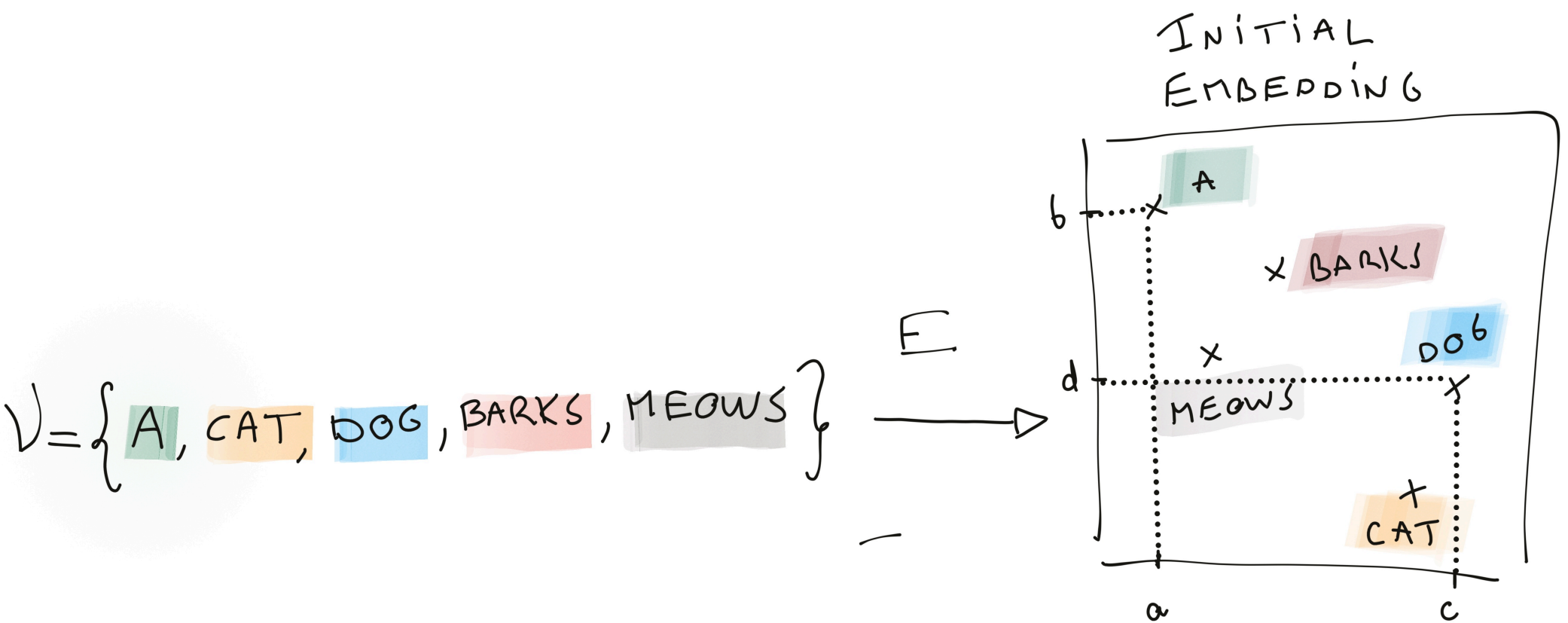
$$L_{Seq2Seq} = -\sum_j \sum_{t=1}^{m} \sum_{c \in V} \mathbb{I}_{v_t^j = c} \log P(v_m^j = c | v_{1:m-1}^j, w_{1:n}^j)$$

Other losses

Transformers exact objective functions depends on type of considered problems

Initial embedding

- The token are first embedded in a d dimensional space (see previous topic)
- the representation $x_i = Ew_i$ is independent of context.
- Let $X = x_{1:n} = w_{1:n}E^\top \in \mathbb{R}^{n \times d}$.



A minimal self-attention architecture for decoder

Attention is a communication mecanism between the tokens of a sequence.

Self attention proposes a contextual representation h_i of x_i as a linear combination the sequence:

$$h_i = V. \sum_{j=1}^{n} \alpha_{ij} x_j$$

- h_i aggregates the past between query (current postion i) and key (other past position j)
  → $\sum_j \alpha_{ij} = 1$
- In decoder the future cannot communicate with the past
  → $\forall j > i, \alpha_{ij} = 0$
- the structure of the attention matrix is thus lower triangular in classical GPT decoder,
- it is not necessary to have a triangular attention matrix (i.e. sentiment analysis)
- V is learned matrix of parameters and can be seen a projection matrix modifying the embedding of the linear combination of sequence tokens. Let consider V as a square matrix $d \times d$.

Attention and matrix notation

- $\alpha_{ij} = \text{softmax}(x_i^T Q^T K x_j) = \frac{exp(x_i^T Q^T K x_j)}{\sum_{j=1}^{n} \exp\left(x_i^T Q^T K x_j\right)}$, where
  - → Q is a matrix that modifies the embedding of the token we are looking for. Let consider V as a square matrix $d \times d$.
    - ⇾ K is a matrix that modifies the embedding of the word we are comparing against. Let consider V as a square matrix $d \times d$.

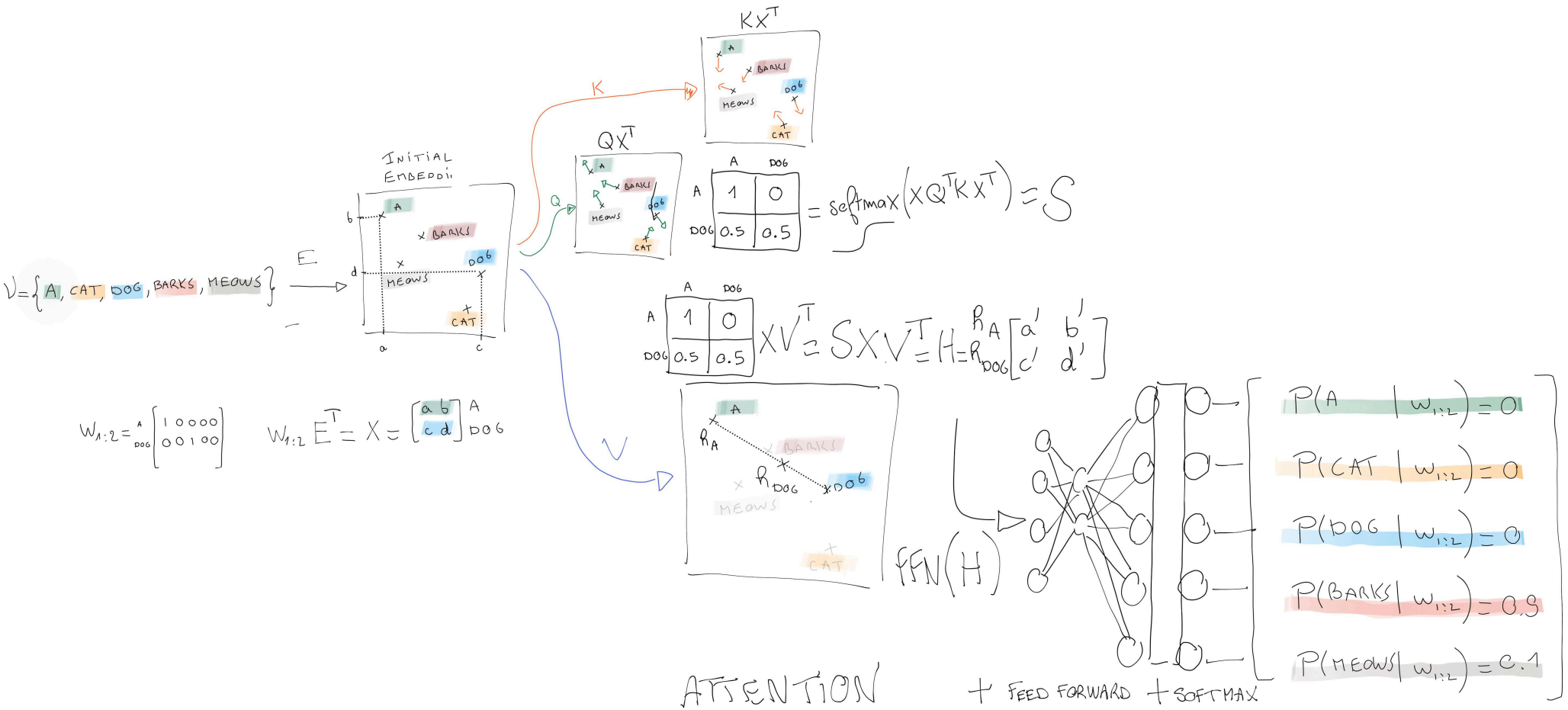
Matrix notation

$$H = \begin{bmatrix} h_1^T \\ \vdots \\ h_n^T \end{bmatrix} = \text{Attention}(\text{Q},\text{K},\text{V},\text{X}) = \text{softmax}(X(Q^T K)X^T)XV^T$$

Remark

In the original paper (“Attention is all you need”, 2017) the notation is different and notes:

Ilustration of self-Attention



Interpretation of Q, K, V

- The query is modified version of an initial embedding $q_i = Qx_i$
- The Key is modified version of an initial embedding $k_i = Kx_i$
- Value is modified version of an initial embedding $v_i = Vx_i$, which should include contextual information since it is a transformation of a linear combination of the context tokens…

Attention Concept	Sometimes Similar to
Query	Target (What to focus on)
Key	Context (What to compare against, the source of information)
Value	Contextual embedding

Important details

The original paper introduces many add-ons, which much improve the transformers performances:

- **Positional encoding** for introducing the notion of order in the sequence
- FFN: **Feed Forward Neural layer**.
  - → As the attention is a linear operation composed with a softmax, a Feed Forward Neural layer is added for making the function more flexible
- **Multi-blocks**: In the spirit of Deep learning blocks of attention and FNN are repeated…
- **Dropout layers** are used for prevenring overfitting (from “Dropout a simple way to prevent NN from overfitting” Hinton)
- Simple trick of **Skip connection** for stabilizing the gradient (from “Deep learning residuals for image recognition”)
  - → The deep learning structure tends to cause problems with gradient computation (vanishing or exploding gradient)
- In order to make tokens comparable, they are all **normalized** at the output of each blocks:
  - → The classical z-transformed is applied (with Bayesian correction)
- **Multi-head attention**: computes multiple attention functions (heads) in parallel

Positional encoding

- Attention do not consider order of tokens !
- Incorporating information about the order of tokens is achieved by adding positional encodings to the input embeddings

The original transformer implementation uses:

- even indices:

$$PE_{\text{pos},2i} = \sin\left(\frac{\text{pos}}{10000^{\frac{2i}{d}}}\right)$$

- odd indices:

$$PE_{\text{pos},2i+1} = \cos\left(\frac{\text{pos}}{10000^{\frac{2i}{d}}}\right)$$

It ensures that each position is assigned a unique encoding

Feed Forward Neural Network

- **Attention mechanism operates linearly** to capture dependencies between tokens in a sequence
- Non-linearity enhance the model's expressive power

Structure of the FFN:

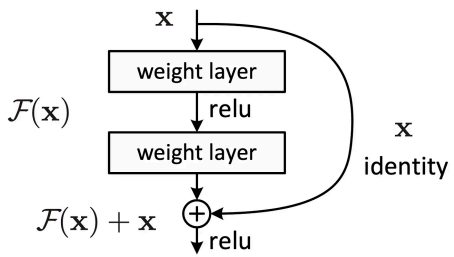
Each FFN consists of two linear transformations with a non-linear activation function in between:

1. First Linear Transformation: embedding of the input from the model's dimensionality d to a higher-dimensional space
2. Activation Function: A non-linear function is applied to introduce non-linearity
3. Second Linear Transformation: projects the output back to the original dimensionality

$$\mathrm{FFN}(x) = W_2 \cdot \mathrm{ReLU}(W_1 \cdot x + b_1) + b_2$$

where W_1 and W_2 are weight matrices, and b_1 and b_2 are bias vectors.

Skip connection (original paper “Deep Residual Learning for Image Recognition” 2015 by He et al.)



Skip connections address the vanishing (and exploding) gradient problems

- Consider a neural network layer with an input x and a desired underlying function $H(x)$.
- Incorporating a skip connection, the layer is restructured to model a residual $F(x) = H(x) - x$. Thus, the output of this layer becomes:

$$y = F(x) + x$$

Benefits:

- Enhanced Gradient Flow
- Faster convergence and higher accuracy

Multi-head attention

- Multi-Head Attention captures various aspects of the data.
- The mechanism computes multiple attention functions (heads) in parallel.
- Each head operates on linearly projected versions of the queries, keys, and values

Each head k computes the attention scores

$$\text{Attention}(Q_k, K_k, V_k) = \text{softmax}\left(\frac{XQ_k^T K_k X}{\sqrt{d_k}}\right) XV_k^T$$

where d_k is the dimension of the key vectors

Concatenation

The outputs of all heads are concatenated and projected through a final linear layer:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)P$$

The multi head matrix is $d \times d$, where - h is the number of head, - $d_k = d/h$ the size of the projection space of head k - P is a projection matrix combining the heads.

Benefits

- Diverse Representations: Each head can capture different features or relationships within the data, enabling the model to understand various aspects of the input (grammar, style, “meaning”…).
    - → Parallel Processing: Multiple heads allow the model to process different parts of the sequence simultaneously, improving efficiency.
    - → Enhanced Capacity: By attending to information from different subspaces, the model can capture complex patterns and dependencies.

Fine tuning

After a pretraining step that ressembles closely to a decoder...

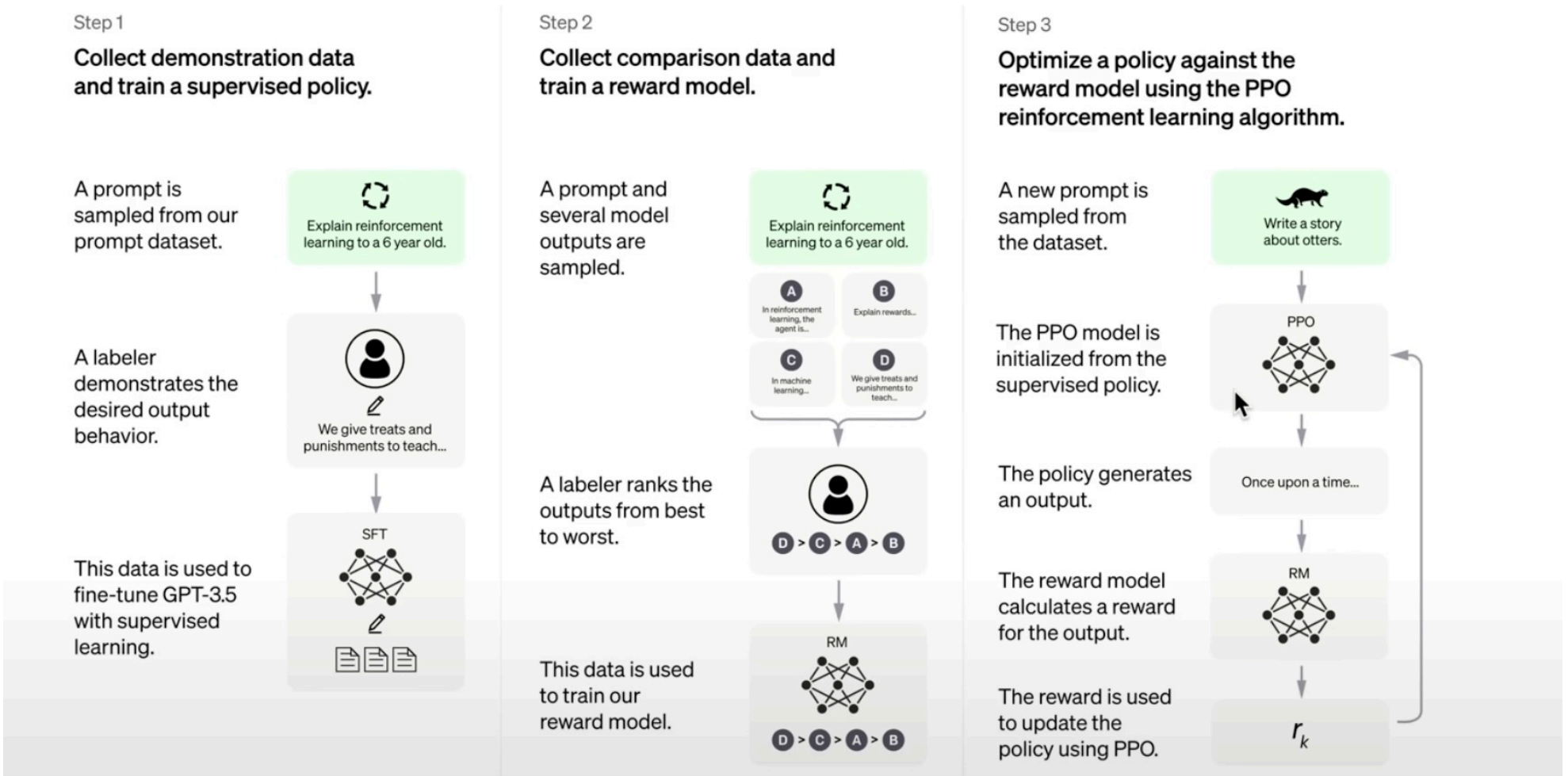


image from chapGPT (karpathy lecture)

Exercices

Schur Complement Lemma

Let A be a square matrix partitioned as follows:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

Assuming that A_{11} is invertible, the Schur complement of A_{11} in A is defined as:

$$S = A_{22} - A_{21}A_{11}^{-1}A_{12}$$

The lemma states that if A is invertible, then A is invertible if and only if S is invertible. Additionally, the inverse of A can be expressed as:

$$A^{-1} = \begin{bmatrix} A_{11}^{-1} + A_{11}^{-1}A_{12}S^{-1}A_{21}A_{11}^{-1} & A_{11}^{-1}A_{12}S^{-1} \\ -S^{-1}A_{21}A_{11}^{-1} & S^{-1} \end{bmatrix}$$

To demonstrate the Schur complement lemma, we can follow these

1. Start with the matrix equation $AX = B$, where A is invertible and partitioned as described above.
2. Write the equation using the partitioned form of matrix A:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$$

3. Apply block matrix operations to rewrite the equation:

$$\begin{aligned} A_{11}X_1 + A_{12}X_2 &= B_1 \\ A_{21}X_1 + A_{22}X_2 &= B_2 \end{aligned}$$

4. Solve the first equation for X_1:

$$X_1 = A_{11}^{-1}(B_1 - A_{12}X_2)$$

5. Substitute this value of X_1 into the second equation:

$$A_{21}A_{11}^{-1}(B_1 - A_{12}X_2) + A_{22}X_2 = B_2$$

6. Simplify the equation:

$$SX_2 = B_2 - A_{21}A_{11}^{-1}B_1$$

7. We can see that the coefficient matrix for X_2 is the Schur complement, $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$
8. Therefore, X_2 can be calculated as:

$$X_2 = S^{-1}(B_2 - A_{21}A_{11}^{-1}B_1)$$

9. Finally, we can substitute the values of X_1 and X_2 to obtain the solution vector X:

$$X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} A_{11}^{-1}(B_1 - A_{12}S^{-1}(B_2 - A_{21}A_{11}^{-1}B_1)) \\ S^{-1}(B_2 - A_{21}A_{11}^{-1}B_1) \end{bmatrix} =$$

$$\begin{bmatrix} A_{11}^{-1} + A_{11}^{-1}A_{12}S^{-1}A_{21}A_{11}^{-1} & -A_{11}^{-1}A_{12}S^{-1} \\ -S^{-1}A_{21}A_{11}^{-1} & S^{-1} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$$

Conditional Gaussian

Demonstrate the form of the distribution of $x_1|x_2$ when both vectors are gaussian.

We have the joint gaussian distribution of $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \sim \mathcal{N}(\mu, \Sigma)$$

We can proceed in three step:

1. Compute $\Sigma^{-1} = \Lambda$ the concentration matrix
2. Write the joint distribution into a sum of marginal in x_2 and conditonal in x_1
3. Derive the conditional distribution.

Concentration matrix

The joint covariance matrix is partitioned as:

$$\Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}.$$

The concentration matrix$\Lambda = \Sigma^{-1}$can be expressed as:

$$\Lambda = \begin{bmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{21} & \Lambda_{22} \end{bmatrix},$$

where the blocks of the concentration matrix are given by:

$$\Lambda_{11} = (\Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21})^{-1}, \quad \Lambda_{12} = -\Lambda_{11}\Sigma_{12}\Sigma_{22}^{-1},$$

$$\Lambda_{21} = -\Sigma_{22}^{-1}\Sigma_{21}\Lambda_{11}, \quad \Lambda_{22} = \Sigma_{22}^{-1} + \Sigma_{22}^{-1}\Sigma_{21}\Lambda_{11}\Sigma_{12}\Sigma_{22}^{-1}.$$

Here,Λ_{11} is obtained using the Schur complement of Σ_{11} in

Σ:

$$\Lambda_{11} = (\Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21})^{-1}.$$

Decompose the joint distribution

The quadratic form of the joint distribution decomposes as follow

$$Q = (x_1 - \mu_1)^T \Lambda_{11}(x_1 - \mu_1) + 2(x_1 - \mu_1)^T \Lambda_{12}(x_2 - \mu_2) + (x_2 - \mu_2)^T \Lambda_{22}(x_2 - \mu_2)$$

Replacing the Λ_{ij} with their expression in function of the Σ_{ij}, we eventually get

$$
\begin{aligned}
Q =& (x_1 - (\mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2)))^T \Sigma_{1|2}^{-1}(x_1 - (\mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2))) + \\
& (x_2 - \mu_2)^T \Sigma_{22}^{-1}(x_2 - \mu_2)
\end{aligned}
$$

where

$$
\Sigma_{1|2} = (\Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21})^{-1} = \Lambda_{11}
$$

Derive the conditional distribution

The joint distribution of $(x_1, x_2) \sim N(\mu, \Sigma)$, where

$$
\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix},
$$

can be expressed as a product of the marginal distribution of x_2 and the conditional distribution of x_1 given x_2.

The joint log-probability is:

$$
\log p(x_1, x_2) = \log p(x_2) + \log p(x_1|x_2).
$$

Expanding each term: 1. The marginal distribution of x_2:

$$
p(x_2) \sim N(\mu_2, \Sigma_{22}),
$$

with the log-density:

$$\log p(x_2) = -\frac{1}{2}(x_2 - \mu_2)^T \Sigma_{22}^{-1}(x_2 - \mu_2) - \frac{1}{2}\log|\Sigma_{22}| - \frac{d_2}{2}\log(2\pi).$$

2. The conditional distribution of x_1 given x_2:

$$p(x_1|x_2) \sim N(\mu_{1|2}, \Sigma_{1|2}),$$

where

$$\mu_{1|2} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2), \quad \Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}.$$

The log-density is:

$$\log p(x_1|x_2) = -\frac{1}{2}(x_1 - \mu_{1|2})^T \Sigma_{1|2}^{-1}(x_1 - \mu_{1|2}) - \frac{1}{2}\log|\Sigma_{1|2}| - \frac{d_1}{2}\log(2\pi).$$

Thus, the joint distribution can be written as:

$$\log p(x_1, x_2) = \log p(x_2) + \log p(x_1|x_2).$$

Stochastic Gradient and linear regression

The Mean Squared Error can be expressed as a sum of the available sample $(y_i, \boldsymbol{x}_i)_i$:

$$Q(\boldsymbol{w}) = \frac{1}{n}\sum_i \|y_i - \boldsymbol{w}^t \boldsymbol{x}_i\|^2 = \frac{1}{n}\sum_i Q_i(\boldsymbol{w})$$

The gradient

is expressed as

$$\nabla Q_i(\boldsymbol{w}) = -2\boldsymbol{x}_i(y_i - \boldsymbol{w}^t \boldsymbol{x}_i)$$

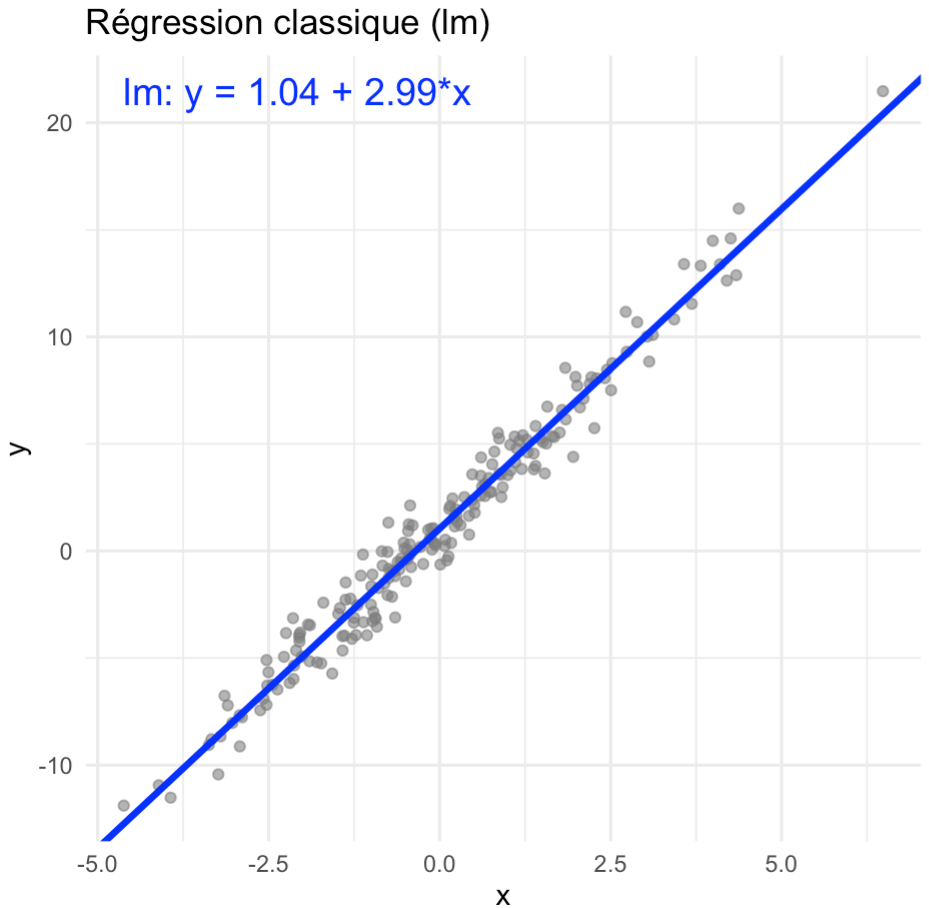
Online regression code

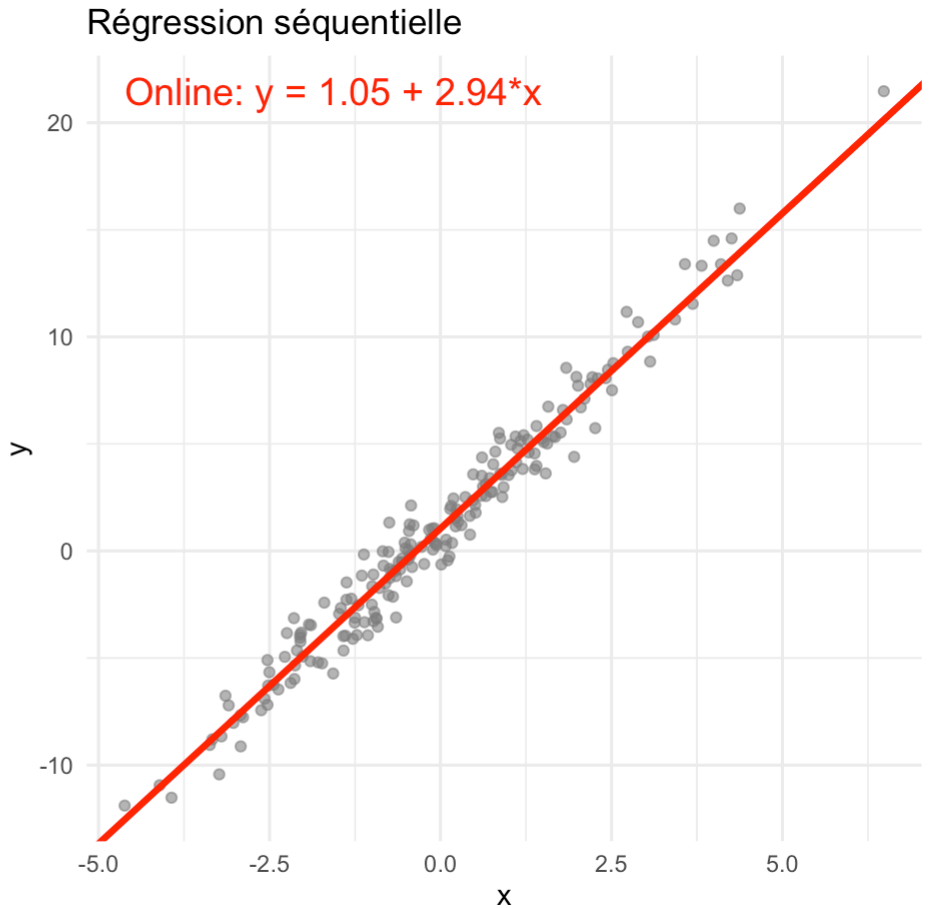
```r
library(ggplot2)
library(patchwork)  # Pour afficher les graphes côte à côte

# Fonction de régression linéaire séquentielle
regression.online <- function(X, Y, max.iteration = 300) {
  p <- ncol(X)
  n <- nrow(X)
  X <- cbind(1, X)  # Ajout de l'intercept

  shuffling <- sample(1:n, n)
  X <- X[shuffling, ]
  Y <- Y[shuffling]
  Q<-rep(0,max.iteration)
```

```r
  W <- rep(0, p + 1)  # Initialisation des coefficients
  for (i in 1:max.iteration) {
    idx <- (i - 1) %% n + 1
    x <- drop(X[idx,])  # Sélection d'un point (matrice 1x(p+1))
    y <- Y[idx]
    W <- W + (1 / i) * x * (y - drop(rbind(W) %*% cbind(x)))  # Mise à jour du vecteur de poids
    Q[i]<-mean((Y - X %*% cbind(W))^2)
  }

  return(list(W=W,Q=Q))
}
```

Comparison with batch approach

```r
# Génération des données
set.seed(123)
x <- rnorm(200, sd = 2)
y <- 1 + 3 * x + rnorm(200, sd = 1)

# Ajustement avec lm
lm_model <- lm(y ~ x)
lm_coef <- coef(lm_model)  # Coefficients de lm

# Ajustement avec la régression séquentielle
online_res <- regression.online(matrix(x, ncol = 1), y)
online_coef<-online_res$W

# Création d'un dataframe pour ggplot
df <- data.frame(x = x, y = y)

# Graphique 1 : Régression classique avec lm
p1 <- ggplot(df, aes(x = x, y = y)) +
  geom_point(color = "gray50", alpha = 0.6) +
  geom_abline(intercept = lm_coef[1], slope = lm_coef[2], color = "blue", lwd = 1.2) +
  annotate("text", x = min(x), y = max(y),
           label = sprintf("lm: y = %.2f + %.2f*x", lm_coef[1], lm_coef[2]),
           hjust = 0, color = "blue", size = 5) +
  labs(title = "Régression classique (lm)",
       x = "x", y = "y") +
```

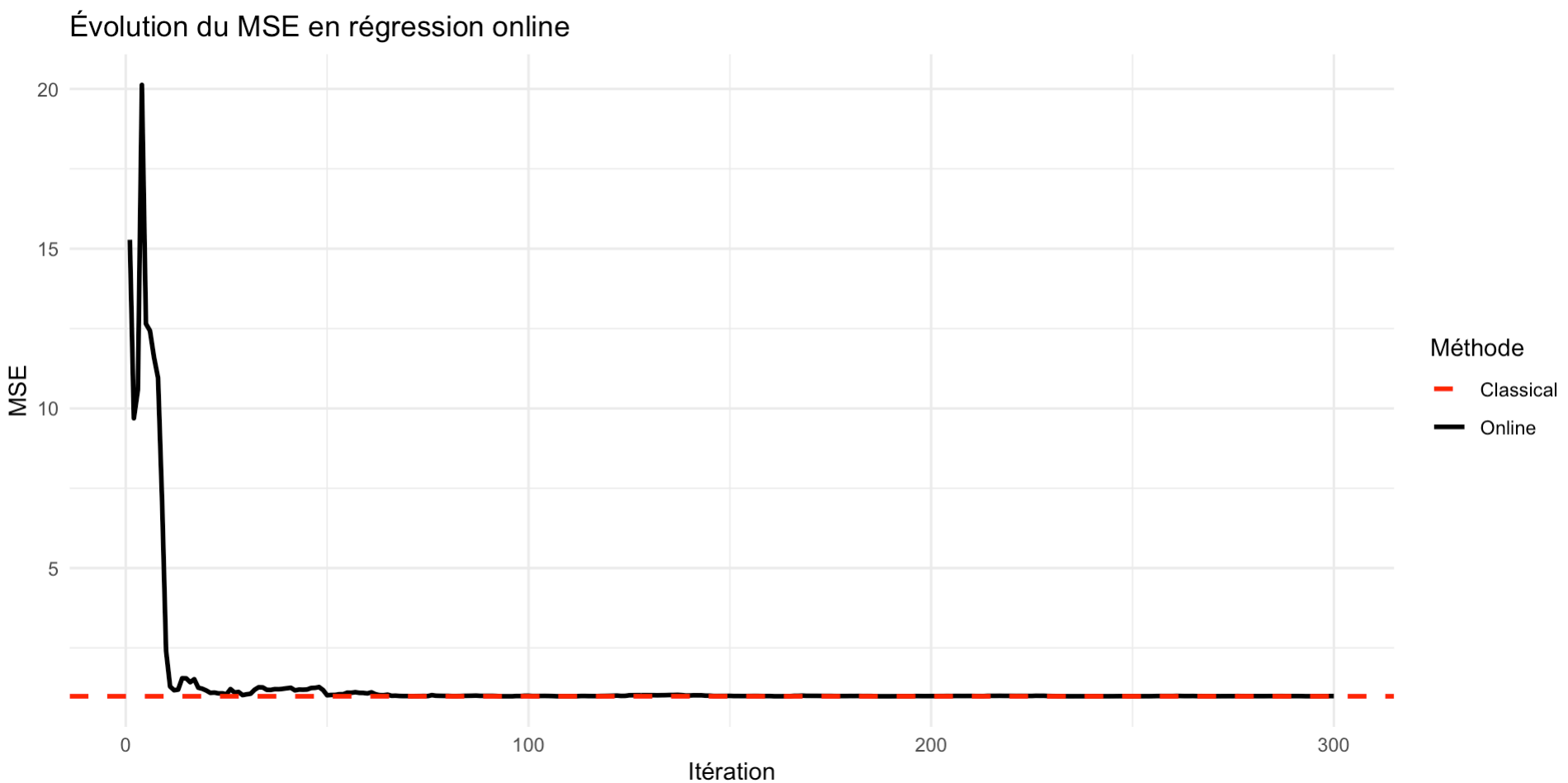




MSE minimisation

```r
library(ggplot2)

# Données pour la courbe de la régression online
df_online <- data.frame(
  MSE = online_res$Q,
  Iteration = 1:length(online_res$Q)
)

# MSE de la régression classique
mse_lm <- mean(lm_model$residuals^2)
```

```r

# Graphique avec ggplot2
ggplot(df_online, aes(x = Iteration, y = MSE)) +
  geom_line(color = "black", size = 1, aes(linetype = "Online")) +  # Courbe Online en noir
  geom_hline(aes(yintercept = mse_lm, linetype = "Classical"), color = "red", size = 1) +  # Ligne lm en rouge
  scale_linetype_manual(name = "Méthode", values = c("Online" = "solid", "Classical" = "dashed")) +  # Légende
  labs(
    title = "Évolution du MSE en régression online",
    x = "Itération",
    y = "MSE"
  ) +
  theme_minimal()
```



What is the KL (Kullback–Leibler) divergence between two multivariate Gaussian distributions?

KL divergence between two distributions P and Q of a continuous random variable is given by:

$$D_{KL}(p||q) = \int_x p(x) \log \frac{p(x)}{q(x)}$$

And probabilty density function of multivariate Normal distribution is given by:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{k/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

Now, let our two Normal distributions be $\mathcal{N}(\boldsymbol{\mu_p}, \Sigma_p)$ and $\mathcal{N}(\boldsymbol{\mu_q}, \Sigma_q)$, both k dimensional.

$$
\begin{aligned}
D_{KL}(p||q) &= \mathbb{E}_p\left[\log(p) - \log(q)\right] \\
&= \mathbb{E}_p\left[\frac{1}{2}\log\frac{|\Sigma_q|}{|\Sigma_p|} - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu_p})^T\Sigma_p^{-1}(\mathbf{x}-\boldsymbol{\mu_p}) + \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu_q})^T\Sigma_q^{-1}(\mathbf{x}-\boldsymbol{\mu_q})\right] \\
&= \frac{1}{2}\mathbb{E}_p\left[\log\frac{|\Sigma_q|}{|\Sigma_p|}\right] - \frac{1}{2}\mathbb{E}_p\left[(\mathbf{x}-\boldsymbol{\mu_p})^T\Sigma_p^{-1}(\mathbf{x}-\boldsymbol{\mu_p})\right] + \frac{1}{2}\mathbb{E}_p\left[(\mathbf{x}-\boldsymbol{\mu_q})^T\Sigma_q^{-1}\right. \\
&= \frac{1}{2}\log\frac{|\Sigma_q|}{|\Sigma_p|} - \frac{1}{2}\mathbb{E}_p\left[(\mathbf{x}-\boldsymbol{\mu_p})^T\Sigma_p^{-1}(\mathbf{x}-\boldsymbol{\mu_p})\right] + \frac{1}{2}\mathbb{E}_p\left[(\mathbf{x}-\boldsymbol{\mu_q})^T\Sigma_q^{-1}(\mathbf{x}-\right.
\end{aligned}
$$

Now, since $(\mathbf{x}-\boldsymbol{\mu_p})^T\Sigma_p^{-1}(\mathbf{x}-\boldsymbol{\mu_p})$ in the second term $\in \mathbb{R}$, we can write it as $tr\left\{(\mathbf{x}-\boldsymbol{\mu_p})^T\Sigma_p^{-1}(\mathbf{x}-\boldsymbol{\mu_p})\right\}$, where $tr\{\}$ is the trace operator. And using the trace trick (eq 16 of section 1.1 from Matrix Cookbook), we can write it as $tr\left\{(\mathbf{x}-\boldsymbol{\mu_p})(\mathbf{x}-\boldsymbol{\mu_p})^T\Sigma_p^{-1}\right\}$.

The second term now is,

$$
= \frac{1}{2}\mathbb{E}_p\left[tr\left\{(\mathbf{x}-\boldsymbol{\mu_p})(\mathbf{x}-\boldsymbol{\mu_p})^T\Sigma_p^{-1}\right\}\right]
$$

The expectation and trace can be interchanged to get,

$$= \frac{1}{2} tr\left\{\mathbb{E}_p\left[(\mathbf{x}-\boldsymbol{\mu_p})(\mathbf{x}-\boldsymbol{\mu_p})^T\Sigma_p^{-1}\right]\right\}$$
$$= \frac{1}{2} tr\left\{\mathbb{E}_p\left[(\mathbf{x}-\boldsymbol{\mu_p})(\mathbf{x}-\boldsymbol{\mu_p})^T\right]\Sigma_p^{-1}\right\}$$

We know $\mathbb{E}_p\left[(\mathbf{x}-\boldsymbol{\mu_p})(\mathbf{x}-\boldsymbol{\mu_p})^T\right] = \Sigma_p$. Simplifying it to

$$= \frac{1}{2} tr\left\{\Sigma_p\Sigma_p^{-1}\right\}$$
$$= \frac{1}{2} tr\left\{I_k\right\}$$
$$= \frac{k}{2}$$

We can simplify the third term

$$\mathbb{E}_p\left[(\mathbf{x}-\boldsymbol{\mu_q})^T\Sigma_q^{-1}(\mathbf{x}-\boldsymbol{\mu_q})\right] = \mathbb{E}_p\left[(\mathbf{x}-\boldsymbol{\mu_p}+\boldsymbol{\mu_p}-\boldsymbol{\mu_q})^T\Sigma_q^{-}\right.$$
$$= \mathbb{E}_p\left[(\boldsymbol{\mu_p}-\boldsymbol{\mu_q})^T\Sigma_q^{-1}(\boldsymbol{\mu_p}-\boldsymbol{\mu_q}) + tr\left\{\Sigma_q^{-1}\Sigma_p\right\} + 2(\boldsymbol{\mu}\right.\right.$$
$$= (\boldsymbol{\mu_p}-\boldsymbol{\mu_q})^T\Sigma_q^{-1}(\boldsymbol{\mu}$$

Combining all this we get,

Kullback–Leibler divergence between two multivariate Gaussian distributions

Compact form

$$D_{KL}(p||q) = \frac{1}{2}\left[\log\frac{|\Sigma_q|}{|\Sigma_p|} - k + (\boldsymbol{\mu_p} - \boldsymbol{\mu_q})^T\Sigma_q^{-1}(\boldsymbol{\mu_p} - \boldsymbol{\mu_q}) + tr\left\{\Sigma_q^{-1}\Sigma_p\right\}\right]$$

When q is $\mathcal{N}(0,\, I)$

$$D_{KL}(p||q) = \frac{1}{2}\left[\boldsymbol{\mu_p}^T\boldsymbol{\mu_p} + tr\left\{\Sigma_p\right\} - k - \log|\Sigma_p|\right]$$