The Ottawa Hospital RESEARCH INSTITUTE | L'Hôpital d'Ottawa INSTITUT DE RECHERCHE

uOttawa | Institut de recherche sur le cerveau / Brain and Mind Research Institute

Workshop on Applied Deep Learning in Intracranial Neurophysiology

Part 4 – Introduction to (Variational) Auto-Encoders

June 20, 2019

Presented by Guillaume Doucet, PhD

Sachs Lab

From the previous sessions

We learned that deep learning algorithms can learn to **predict/decode/classify** from either engineered or extracted **signal features**.

But, in some cases:

- Data are noisy and contain large artefacts
- Data dimensionality is too high to decode directly or to visualize (e.g. NeuroPixel or multi-channel LFPs)
- Both labels and features aren't known or need to be explicitly found
- Data don't need to be classified

From the previous sessions

For example:

In [ ]:

```
# Create our data
freqs = [15.0, 40.0]  # Peak frequencies, in Hz
amps = [2.0, 1.0]     # Sinusoidal component amplitudes

N = 160
n_trials = 2000
batch_size = 5
n_epochs = 5
n_channels = 1
n_samples = len(t)
X = []
Y = []
for tr_idx in range(n_trials):
    t_offset = t + np.random.rand(1) / 15  # shift timestamps by up to 1/15th of a second
    y = amps[1] * np.sin(2*np.pi*freqs[1]*t_offset)
    x = amps[0] * np.sin(2*np.pi*freqs[0]*t_offset) + y
    x += 0.5 * np.random.randn(*x.shape)  # Uncomment to add noise
    Y.append(y.reshape(n_samples, n_channels).astype(np.float32))
    X.append(x.reshape(n_samples, n_channels).astype(np.float32))
dataset = tf.data.Dataset.from_tensor_slices((X, Y))
dataset = dataset.shuffle(n_trials).batch(batch_size).repeat()
# Recreate the above model (to reset its weights)
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv1D(1, N, padding='same', activation='linear', input_shape=(n_samples, n_channels)))
model.compile(loss='mse', optimizer='rmsprop')
# Train
history = model.fit(x=dataset, epochs=n_epochs, steps_per_epoch=n_trials // batch_size)
```

From the previous sessions

```
y = amps[1] * np.sin(2*np.pi*freqs[1]*t_offset)
x = amps[0] * np.sin(2*np.pi*freqs[0]*t_offset) + y
x += 0.5 * np.random.randn(*x.shape)  # Uncomment to add noise
```

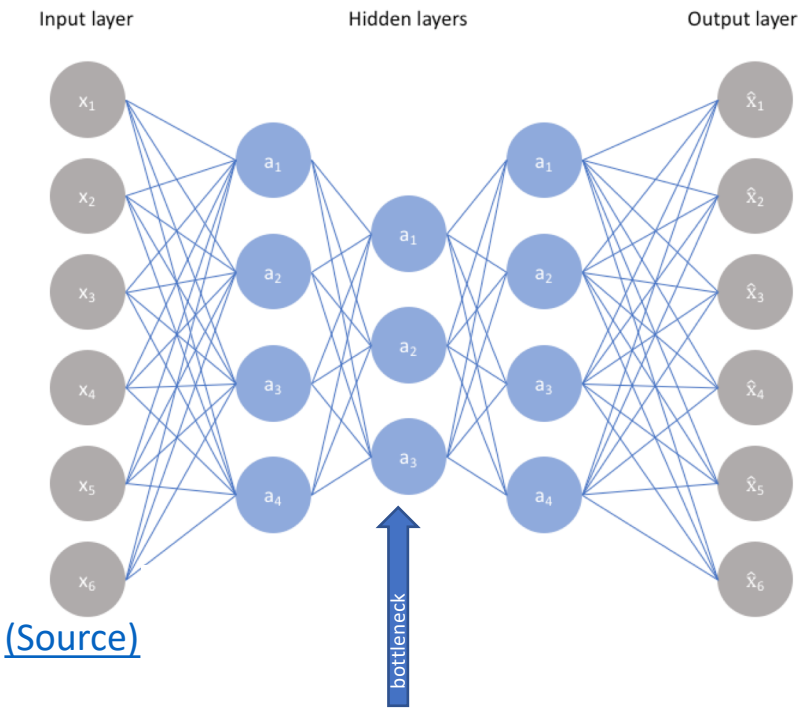
We are training a model to generate a reconstructed version ($\hat{Y}$) of the original signal (Y), from a "noisy" signal ($Y + X + noise$), where both X and $noise$ are unwanted.

$$Y + X + noise \rightarrow MODEL \rightarrow \hat{Y} \approx Y$$

In this case, there are **no labels** and **no classification** accuracy. The network is simply **encoding** its input.

This network is known as an **Auto-Encoder (AE).**

What are Auto-Encoders?



(Source)

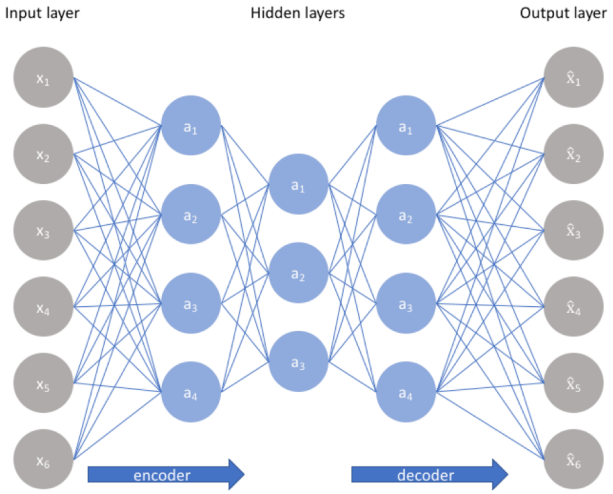
AEs are:

- "self-supervised" neural networks that learn to reconstruct the input data

- subjected to a representational **bottleneck**, forcing them to learn **compressed** and **distributed** representations of input data.
  - **Compressed:** N bottleneck << N input
  - **Distributed:** where each input is represented by multiple features and similar input will be closer in the feature space than dissimilar ones.

- flexible, as they can contain any type of layers (e.g. convolutional, dense, recurrent,…)

What are Auto-Encoders?



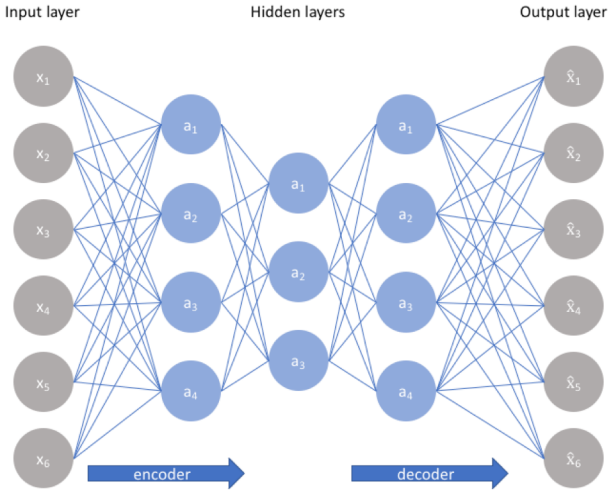
(Source)

AEs are:

- comprised of an **encoder** and a **decoder**.
  - **Encoder** extracts the latent features from the data.
  - **Decoder** tries to reconstruct the input data from the extracted features.

The **objective/loss function** of the network is to **minimize reconstruction error**, while **learning the most informative features** of the input data (i.e. discarding non-informative features, such as noise).

What are Auto-Encoders?

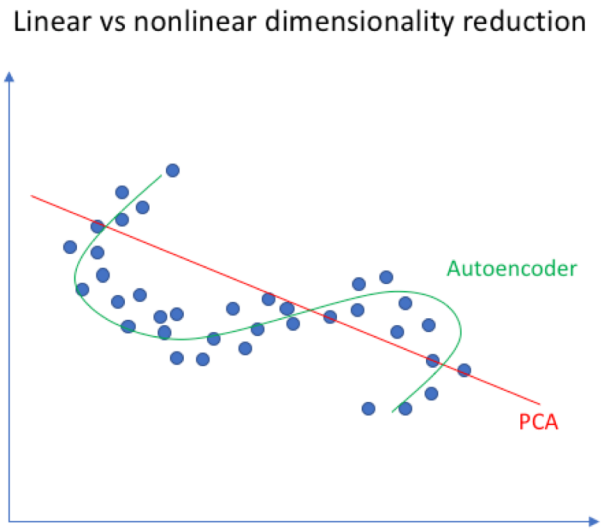


[(Source)]

AEs are used to:

- Remove unwanted parts of the signal (e.g. noise or artefacts)
- Reduce dimensionality and compress data
- Learn meaningful features and representations from the data
- Generate novel signals and data

But what about PCA?



(Source)

PCA is:

- Linear
- Inefficient when dealing with large number of features
- Generating orthogonal components/features

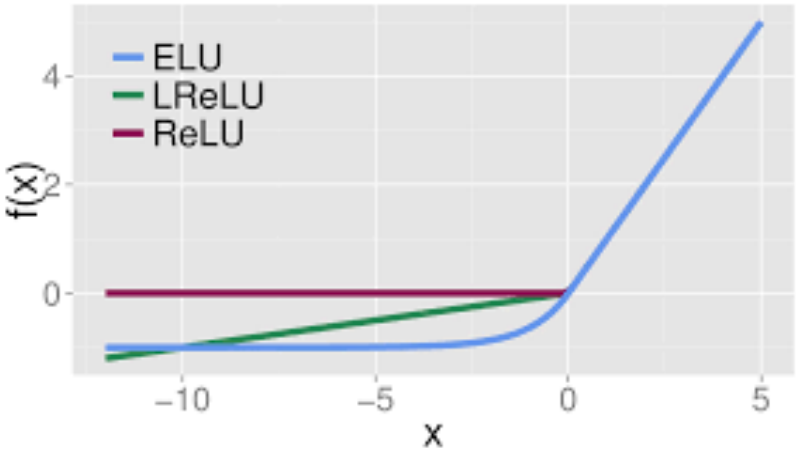
AEs are:

- Non-linear
- Good at handling large sets of features
- Not necessarily generating orthogonal features, but can be constrained to do so

Basic implementation

Network architecture:

- Encoder network:
  - Input: N x 48 x 1; N = batch size
  - 1D convolution: 25 filters, filter size = 5, stride = 1 activation ELU
  - Max pooling: size= 2, stride= 1
  - 1D convolution: 25 filters, filter size = 3, stride = 1 activation ELU
  - Max pooling: size= 2, stride= 1
  - 1D convolution: 25 filters, filter size = 3, stride = 1 activation ELU
  - Max pooling: size= 2, stride= 1
  - Dense layer: 50 units, activation ELU
  - Bottleneck



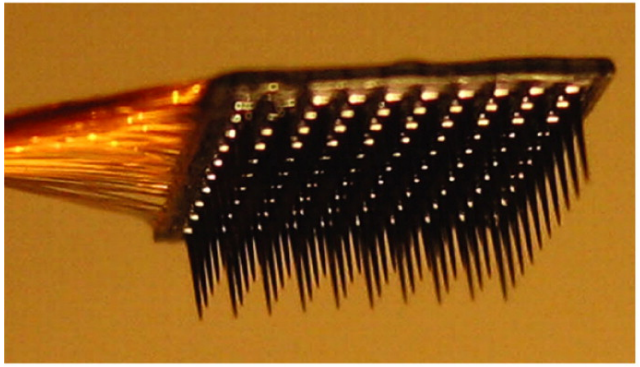
Basic implementation

Network architecture:

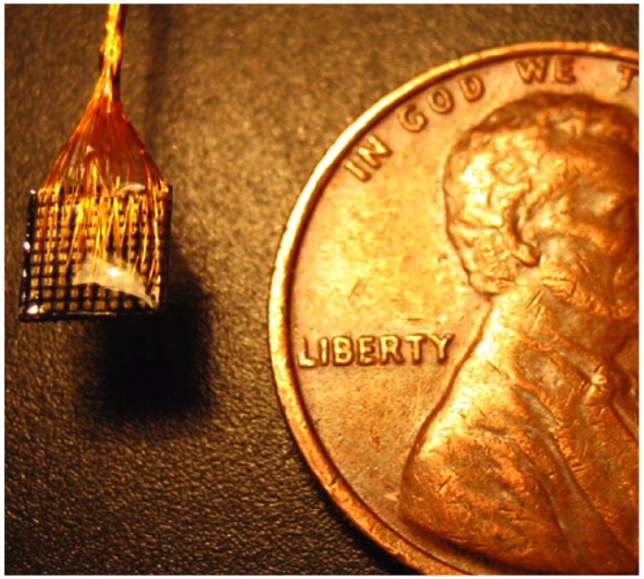
- Decoder network:
  - Encoder network in reverse replacing max pooling by up sampling
  - Last layer is 1D convolution: 1 filter, size = 1, stride = 1

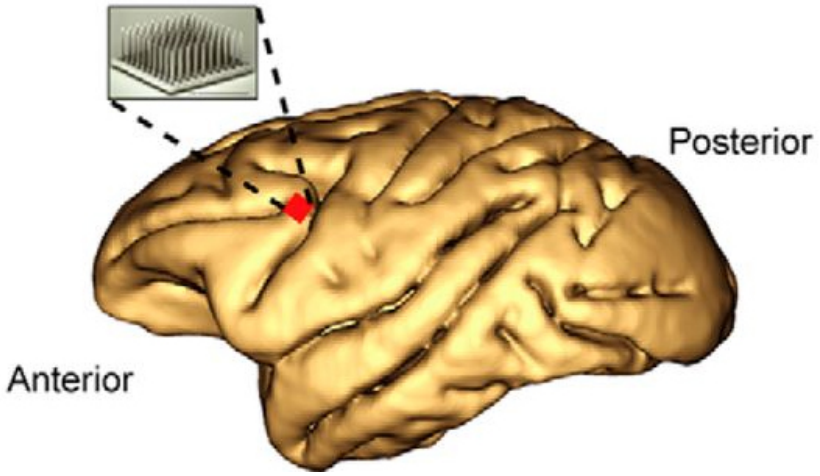
- Loss function:
  - For binarized data: binary cross-entropy
  - For discrete data: categorical cross-entropy
  - For continuous data: **Mean Squared Error**

Dataset

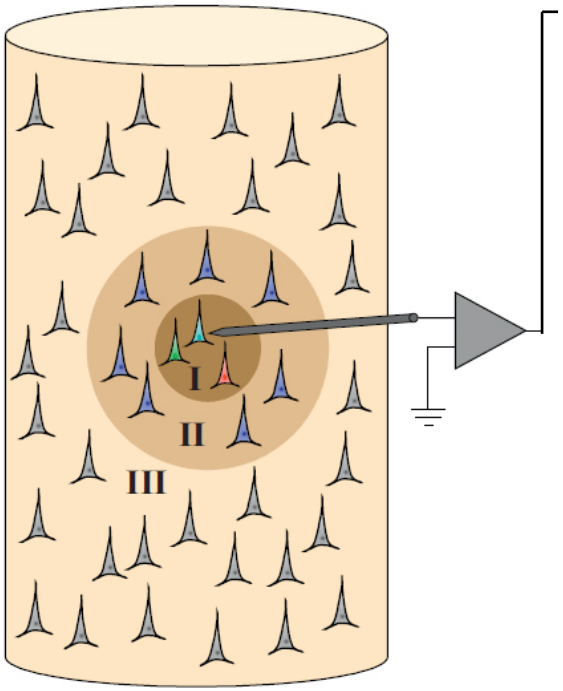


Recording from a 96 channel Utah Array in the lateral prefrontal cortex of a macaque monkey.

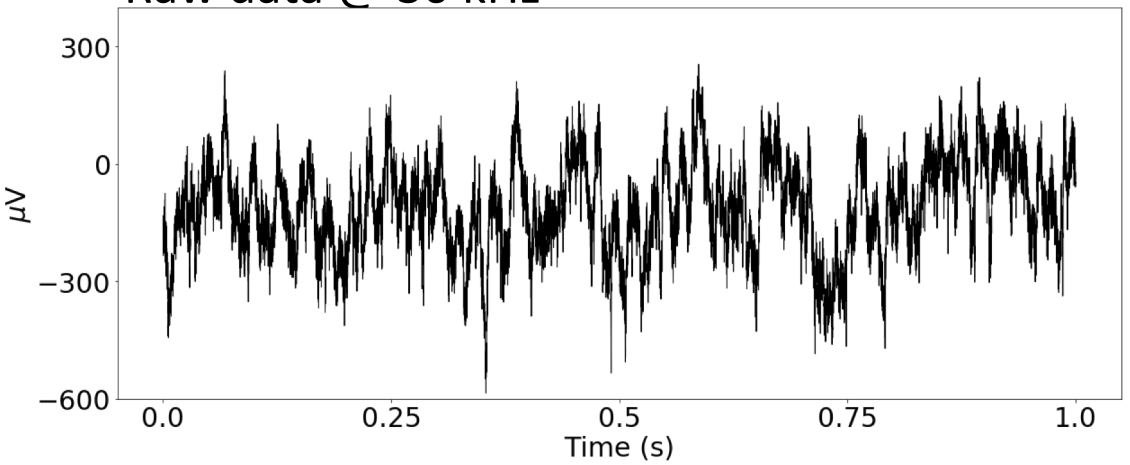


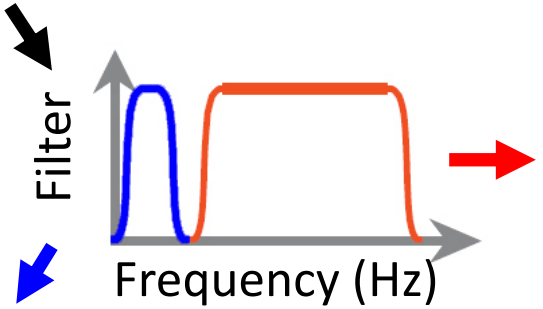


Dataset

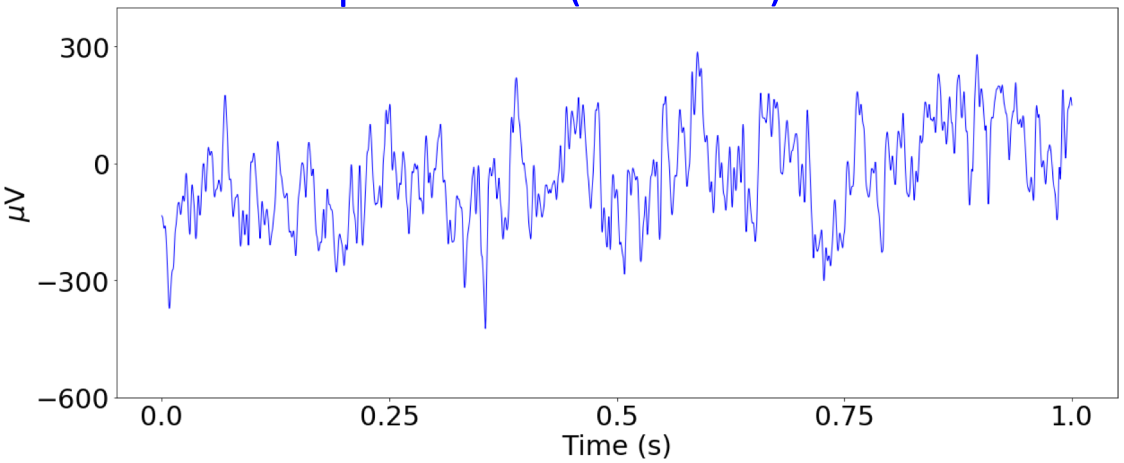


Raw data @ 30 kHz

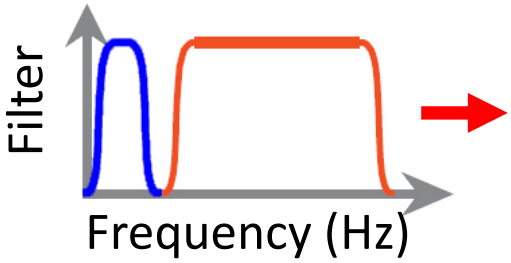




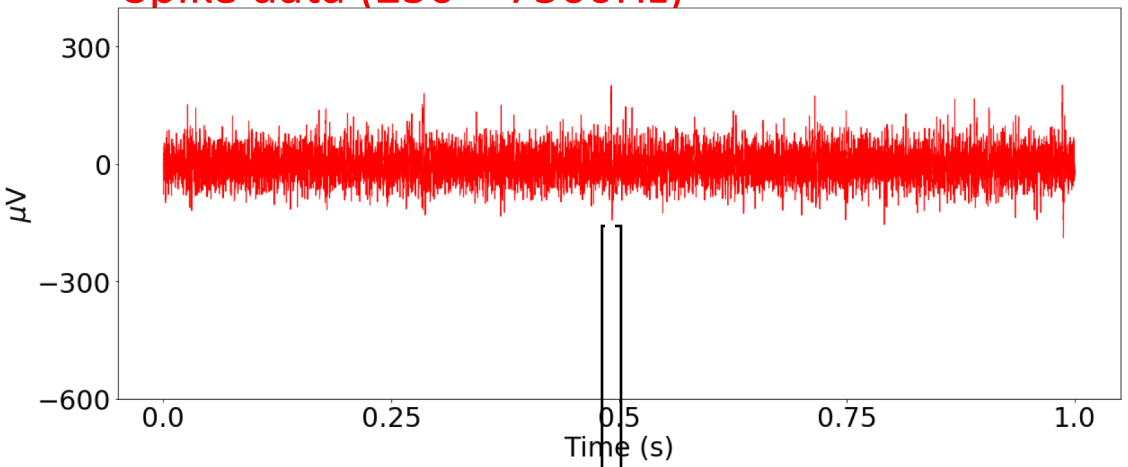
Local field potentials (≤ 250 Hz)

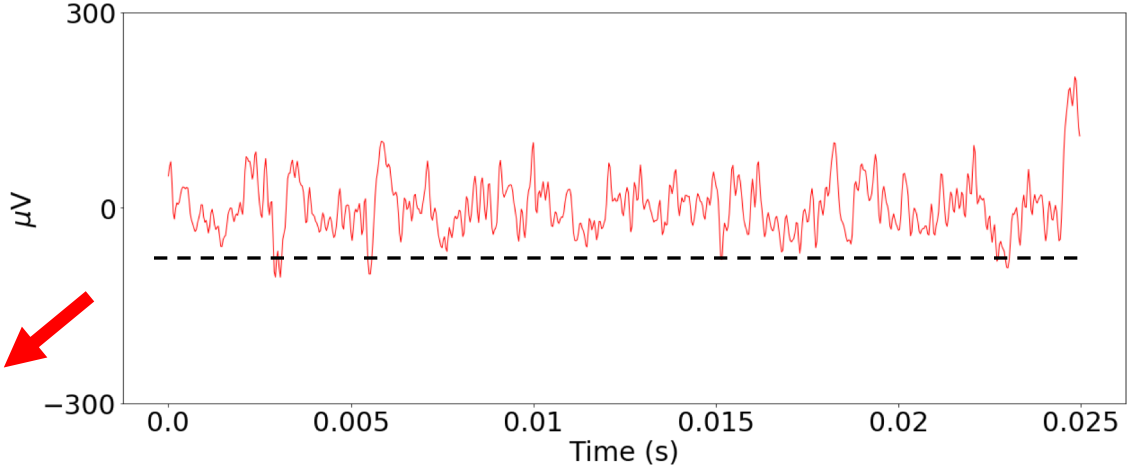


Dataset

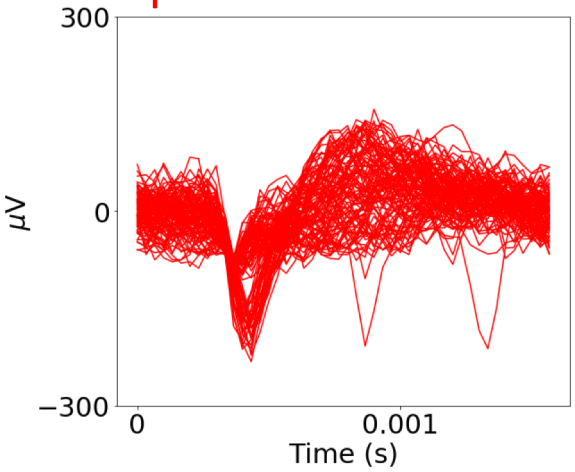


Spike data (250 – 7500Hz)

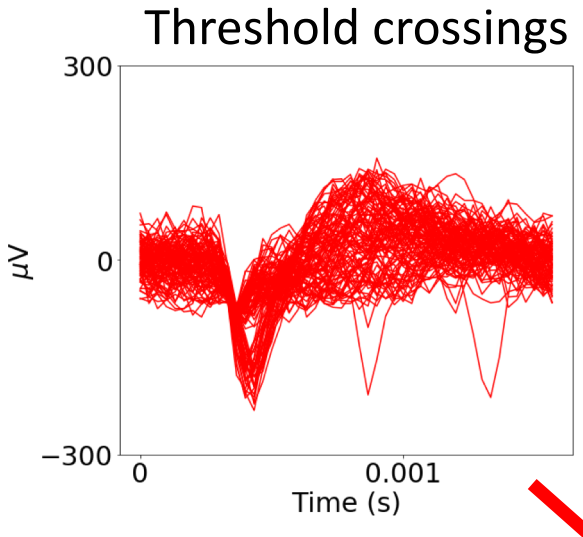


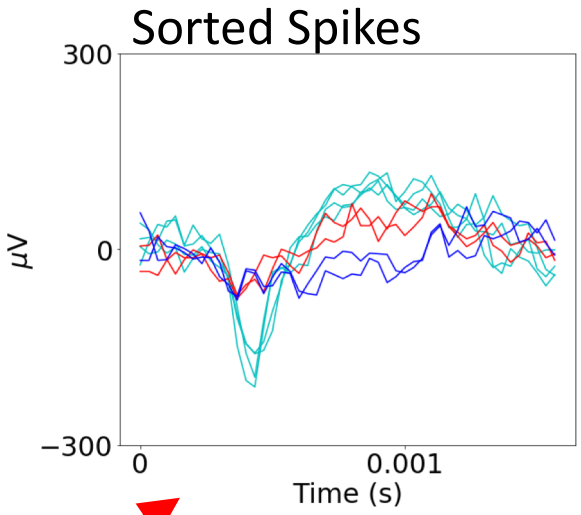


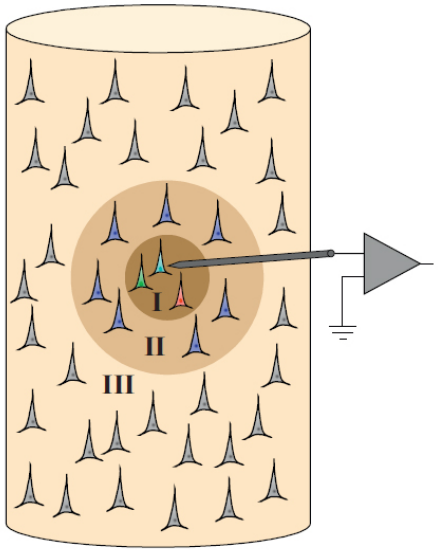
Spike waveforms

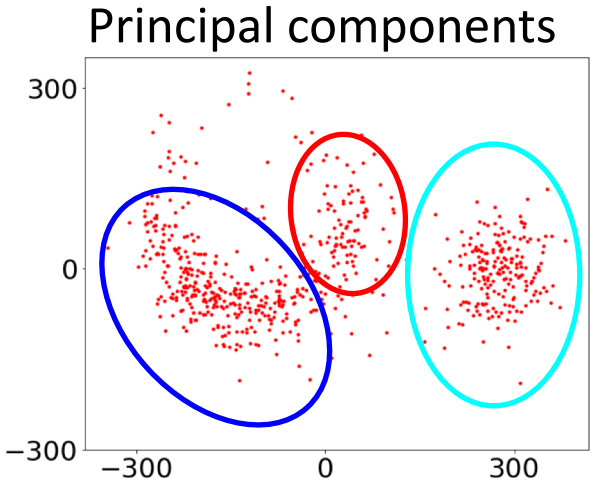


Dataset


Threshold crossings
300
0
−300
μV
0
0.001
Time (s)


Sorted Spikes
300
0
−300
μV
0
0.001
Time (s)


I
II
III


Principal components
300
0
−300
−300
0
300

Dataset

If we want to create our own spike sorting algorithm, we need to address:

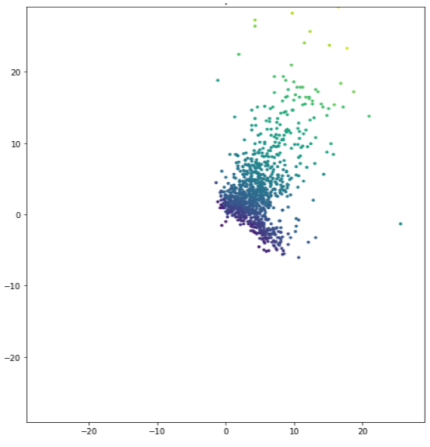
- The data contains a lot of noise and artefacts/invalid waveforms.
- There are no labels on the data (i.e. ground truth is unknown).
- Manual spike sorting results are highly variable across individuals and it is extremely time consuming.
- PCA might not be optimal in separating spike features.

So we need a way to generate artificial, yet realistic, spiking data.

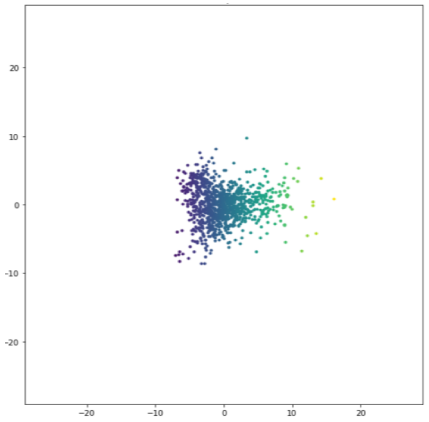
Implementation 1

Issues with implementation 1

AE Space



PCA space



- Points in the AE space are non-uniformly spread out, which creates a lot of empty space.
  - The model could **fail to generalize** to novel examples.
- Latent features seem to have some level of correlation (i.e. diagonal).
  - The model's encoding is **not optimal**.

These drawbacks might not seem so bad, but imagine a latent space with > 50 variables. Correlations and spread between features could greatly impair the network's efficiency.

Possible solution?

It could be possible to add a penalty (i.e. a regularization term) to the loss function to:

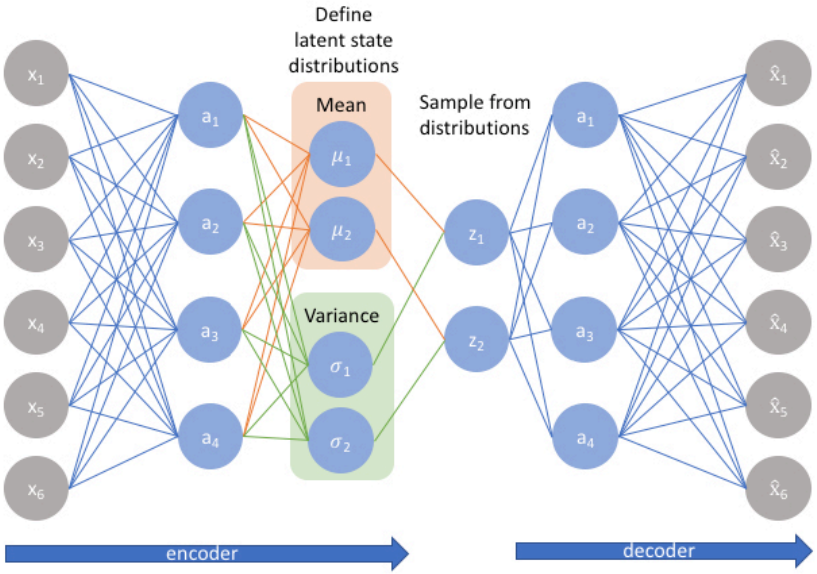
- Keep the latent variables' values minimal
- De-correlate the latent variables

For example, we could push the latent feature space towards a standard multivariate Gaussian (i.e. covariance of features = 0) to simplify data generation.

To do this, we need to map each input, **not to a single point**, but to a **distribution of points**. This means adding some **stochasticity** to the network.

Enter **Variational Auto-Encoders.**

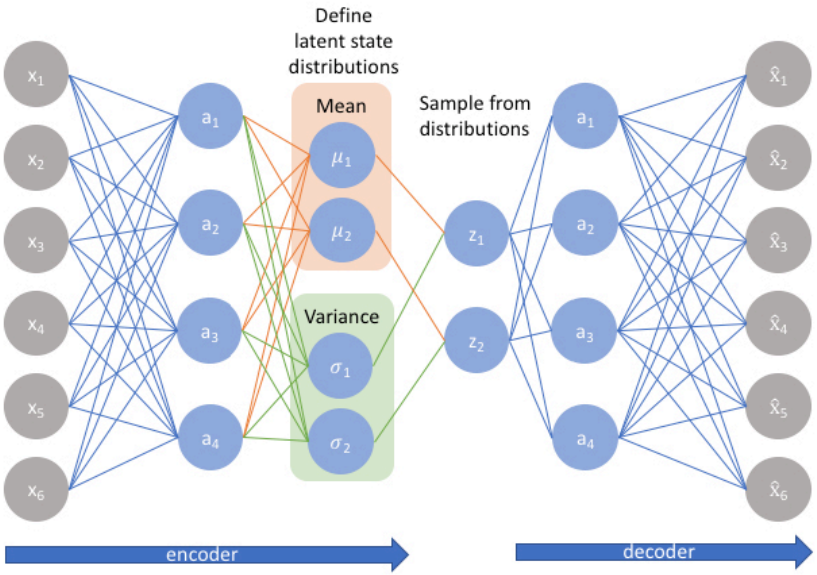
What are Variational Auto-Encoders?



VAEs:

- Are generative models.
- Have a smooth and continuous latent space.
- Learn a **probabilistic latent variable model** of the data: assumes that data generation is a statistical process (e.g. signal + white noise).

What are Variational Auto-Encoders?



For stat nerds:

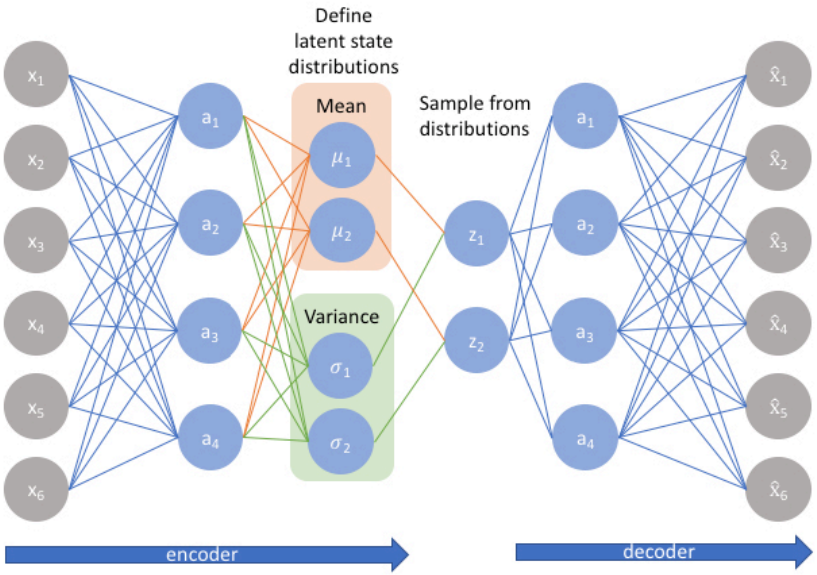
Assuming a latent variable ***z,*** sampled from a prior distribution ***p(z)***, is generating our input ***x***.

We want learn the characteristics of ***z*** from the observable data, such as:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

But this is impractical to compute over all values.

What are Variational Auto-Encoders?



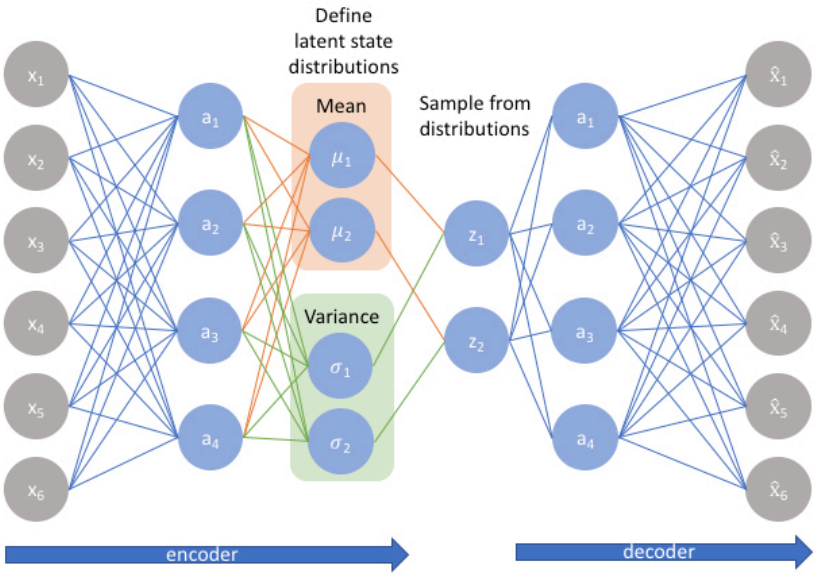
For stat nerds:

So we approximate the encoder **posterior** distribution ***p(z|x)*** by a trainable distribution ***q(z|x)***, whose **prior *p(z)*** is a data independent Gaussian.

The encoder learns the parameters mapping ***x*** to ***z*** (i.e. the posterior ***q(z|x)***).

The decoder learns the parameters maximizing the **likelihood** of generating a real ***x*** given a random sample ***z*** (i.e. ***p(x|z)***)

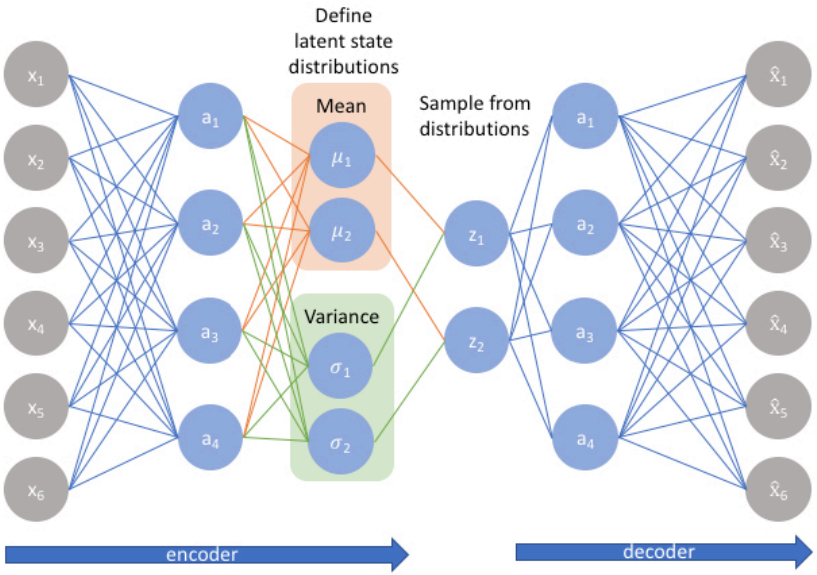
What are Variational Auto-Encoders?



In other words, VAEs learn the **mean** and **variance** of **latent features distributions** generating the input data:

- The model maps each input X to latent distributions $Z \sim N(\mu, \sigma)$, whose parameters (mean and variance) maximize the likelihood that the reconstructed output $\hat{X}$ looks like a real input X.

What are Variational Auto-Encoders?



We then need to train the model to:

- Minimize the error when reconstructing ***x*** from a random sample ***z*** (i.e. train the parameters of ***p(x|z)***)

- Minimize the divergence between the learned parameters of ***z***, our posterior ***q(z|x)***, and the assumed prior $\boldsymbol{p}(\boldsymbol{Z})$.

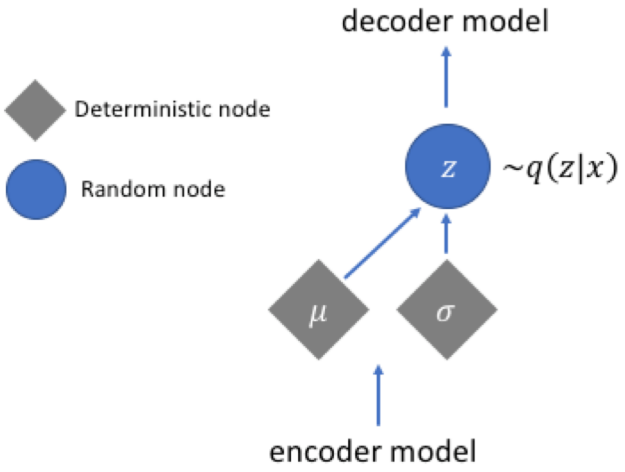
What are Variational Auto-Encoders?

Practically:

- We map the input $\boldsymbol{x}$ to the parameters of the latent feature distributions $\mathbf{z} \sim N(\boldsymbol{\mu}, \boldsymbol{\sigma})$.

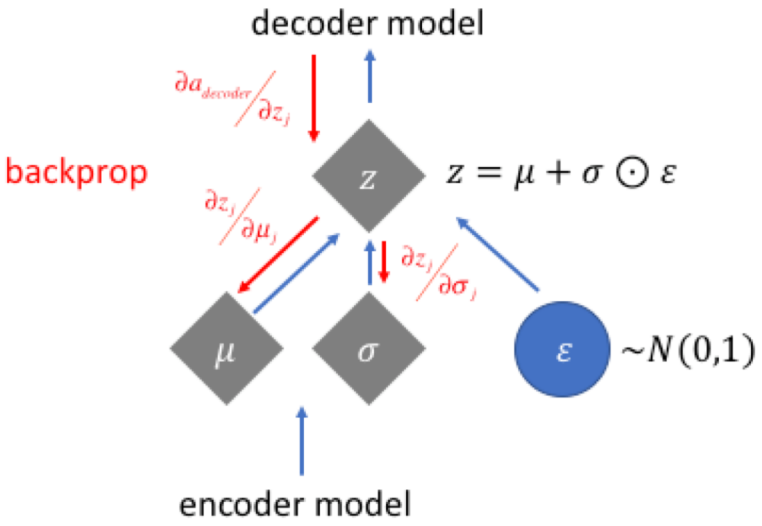
- We randomly sample from $\boldsymbol{z}$ and decode the values to generate $\hat{\boldsymbol{x}}$

But, random sampling isn't differentiable. How can we apply back propagation?



What are Variational Auto-Encoders?

With a nifty **reparameterization** trick:



What are Variational Auto-Encoders?

In code:

```
def encode(self, x):
    mean, logvar = tf.split(self.inference_net(x), num_or_size_splits=2, axis=1)
    return mean, logvar

def reparameterize(self, mean, logvar):
    eps = tf.random.normal(shape=mean.shape)
    return (eps * tf.exp(logvar * .5)) + mean

def decode(self, z, apply_sigmoid=False):
    logits = self.generative_net(z)
    if apply_sigmoid:
        probs = tf.sigmoid(logits)
        return probs
    return logits

def compute_loss(self, x, noisy_x):
  # Encoder part of the network
  if noisy_x is None:
    mean, log_var = self.encode(x)
  else:
    mean, log_var = self.encode(noisy_x)

  if self.is_VAE:
    z = self.reparameterize(mean, log_var)
  else:
    z = mean

  # Decoder
  x_decoded = self.decode(z)
```

What are Variational Auto-Encoders?

In code:

```
# MSE
recons_loss = tf.reduce_sum(tf.math.squared_difference(x, x_decoded), axis=1)
recons_loss = tf.reduce_mean(recons_loss)


# KLD
latent_loss = -0.5 * tf.reduce_sum(1 + log_var - tf.square(mean) - tf.exp(log_var), axis=1)
latent_loss = tf.reduce_mean(latent_loss)

return tf.reduce_mean(recons_loss +
                      self.beta * (tf.math.abs(latent_loss - self.capacity))),
       recons_loss, latent_loss
```

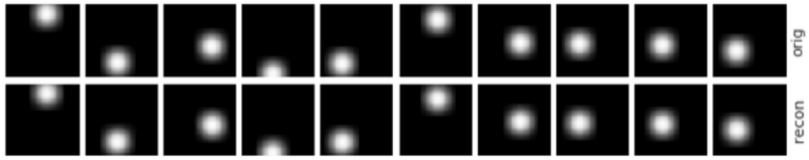
Our loss function combines:

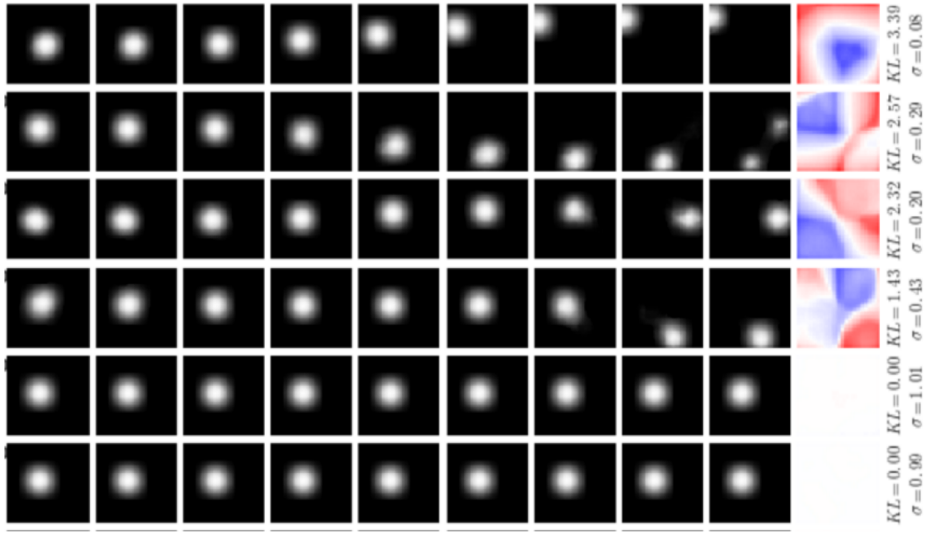
- The reconstruction loss: sum of squared differences
- The regularization term, penalizing deviations from the multivariate Gaussian prior: the Kullback-Liebler divergence (KLD)
- For a normal VAE the “self.beta” term is equal to 1. Ignore capacity for now.

Implementation 2

Can we improve implementation 2

β = 1





If we recall from the early slides that the network learns a distributed representation.

- **Distributed** where each input is represented by multiple features and similar input will be closer in the feature space than dissimilar ones.

Ideally, we'd like **each unit** in the bottleneck to represent a **single feature**, in other word we want a **disentangled** representation.

Can we improve implementation 2

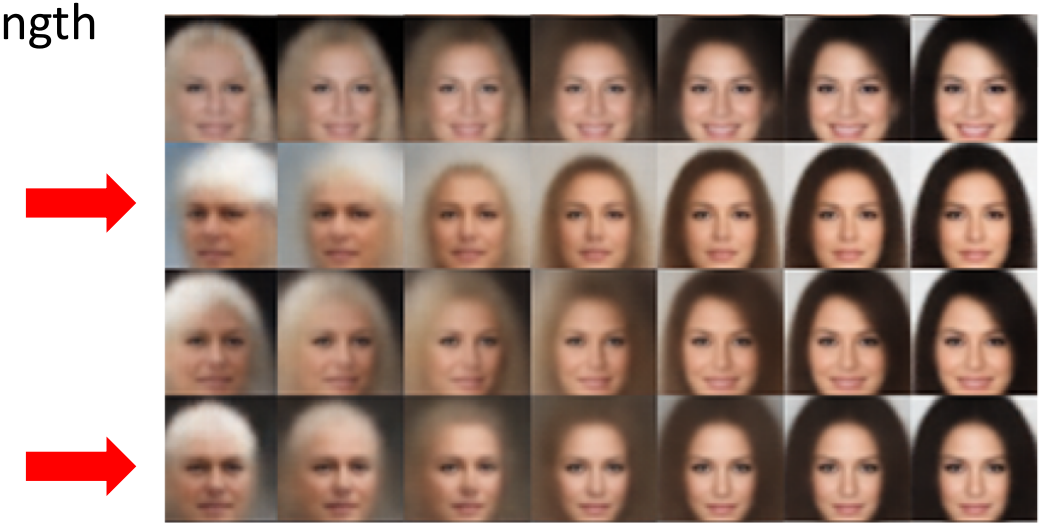
Disentangled ≠ orthogonal.

For example, when analyzing faces, some axes of variation aren't necessarily orthogonal or independent:

- Gender and (facial) hair length

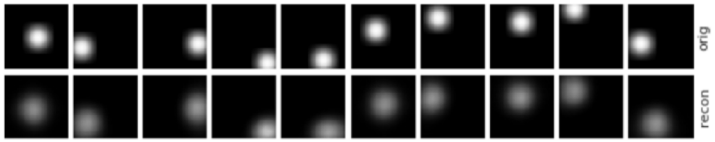
But we want these to be independent when generating new faces by shifting the latent representations along a single axis, such as:

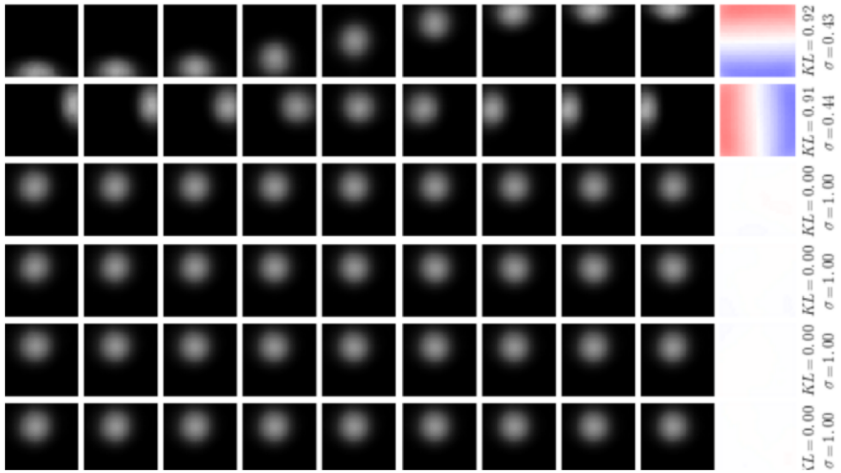
- A male face doesn't shift towards a more feminine face when increasing hair length



Disentangled representations

β = 150 ←





From (Burgess et al., 2018):
"[…] β-VAE finds latent components […] (that) tend to correspond to features in the data that are intuitively qualitatively different, and therefore may align with the generative factors in the data."

Keeping in mind that our prior distribution of latent features is normal Gaussian, the first way to improve disentanglement is:

- to **increase** the loss **pressure** on the **KL divergence** term by increasing the factor **β.**

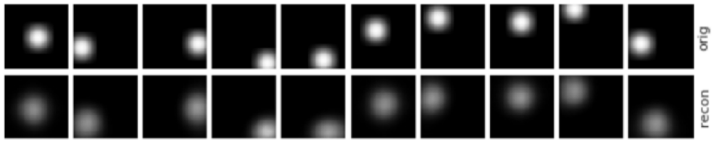
Disentangled representations

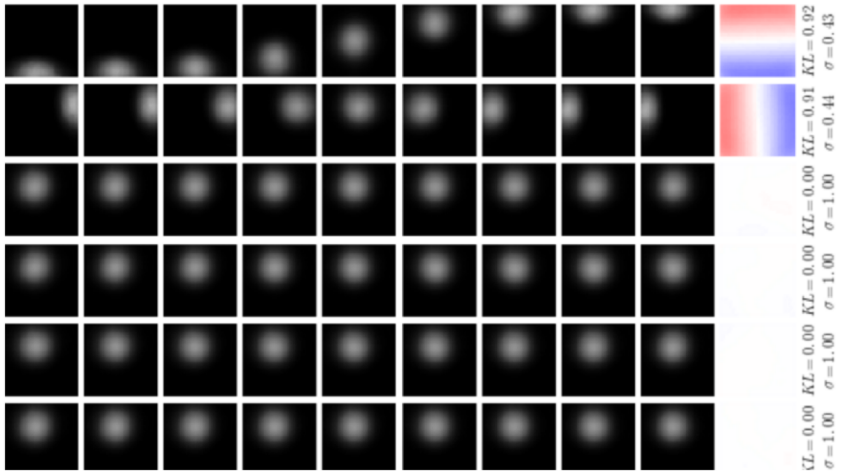
Again from (Burgess et al., 2018):

> “Intuitively, when optimizing a pixel-wise decoder log likelihood, information about position will result in the most gains compared to information about any of the other factors of variation in the data, since the likelihood will vanish if reconstructed position is off by just a few pixels.”
>
> “Continuing this intuitive picture, we can imagine that if the capacity of the information bottleneck were gradually increased, the model would continue to utilize those extra bits for an increasingly precise encoding of position, until some point of diminishing returns is reached for position information, where a larger improvement can be obtained by encoding and reconstructing another factor of variation in the dataset, such as sprite scale.”

Disentangled representations

β = 150





The second way is then to gradually increase the informational bottleneck of the network through training.

To do this we add a new parameter called **capacity** to the loss function:

Loss = reconstruction loss + β (KLD – Capacity)

And we increase the capacity value over training.

Implementation 3

In summary

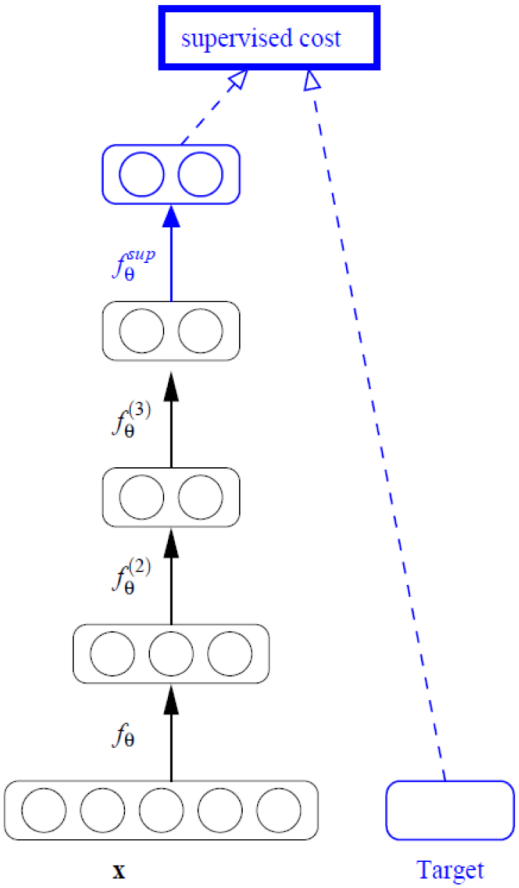
In basic AEs, there is **no pressure** to learn **useful**, **meaningful**, **interpretable** or **disentangled** features of the data.

But there are a few tricks we can use:

- Corrupt the input data with noise (not only adding white noise but: sparsity, L1 regularization,…)
- Use a narrow bottleneck
- Increase the weight on the KL loss
- Slowly decrease the importance of the term across training epochs, forcing better reconstruction

Research using (V)AEs

“Stacked” Auto-Encoders

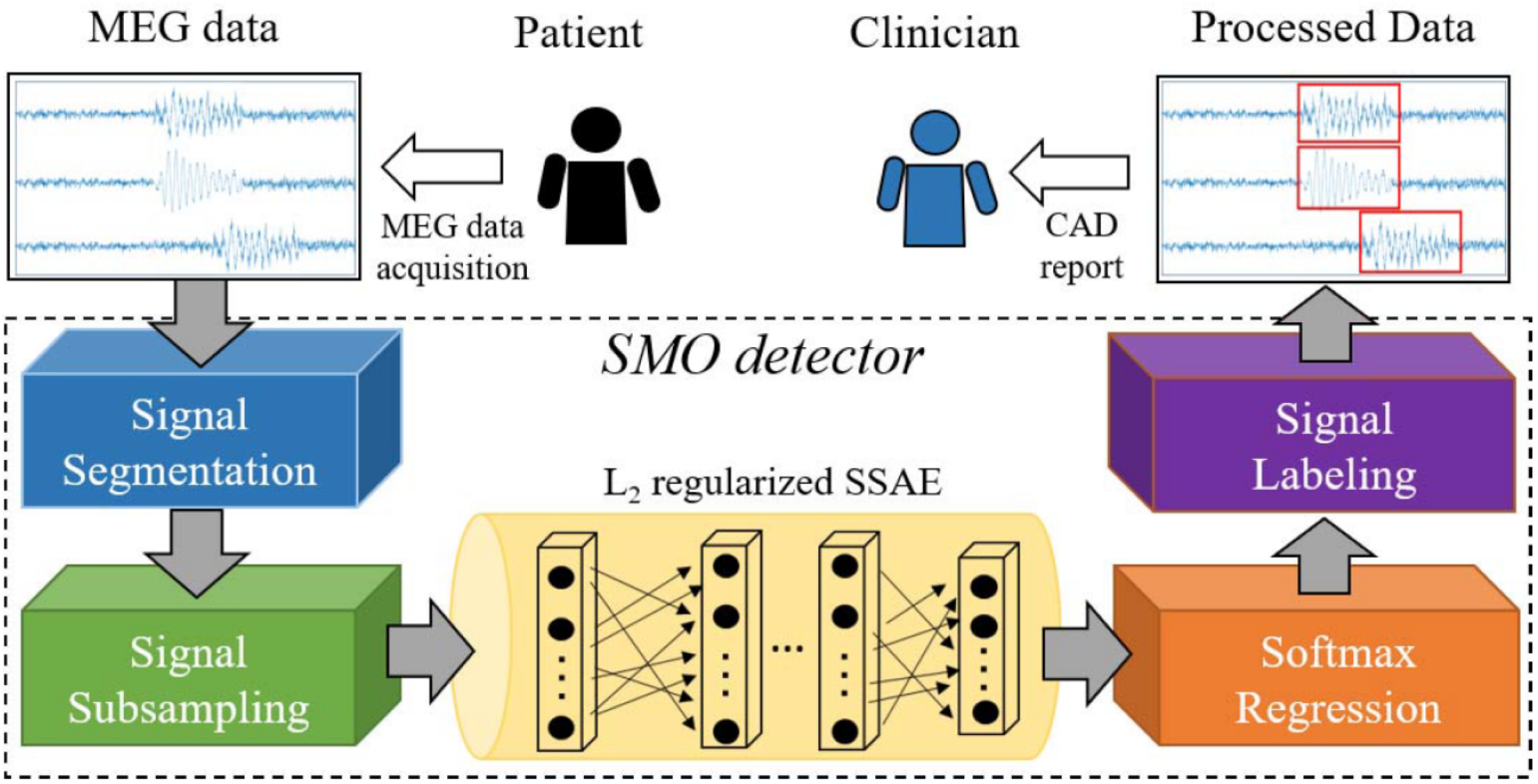


Frequently encountered in the literature, “stacking” comes from the “early” days of deep learning, when training a deep architecture was problematic.

Researchers then trained each layer separately and “stacked” them to form a deeper network.

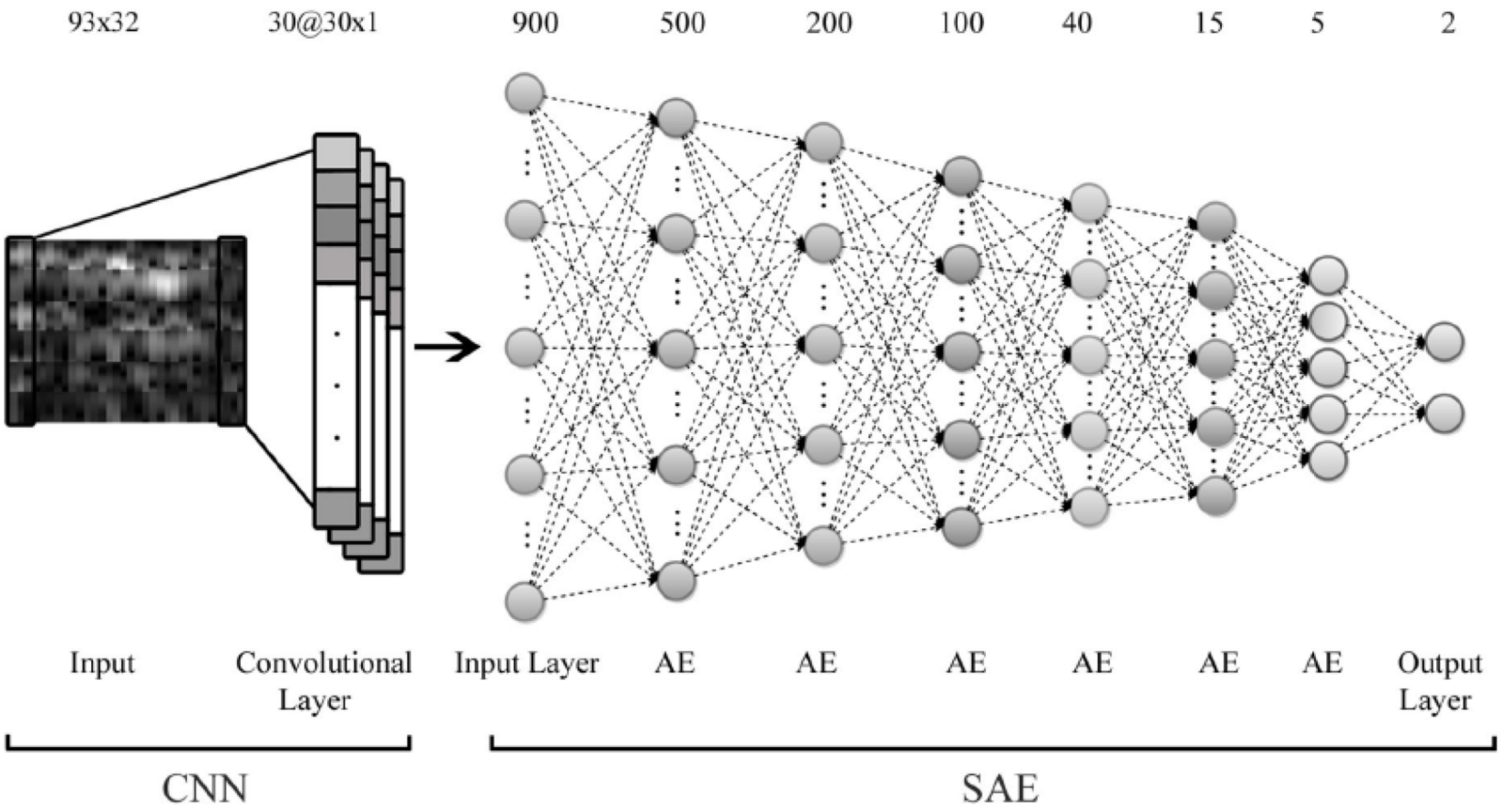
Most of the time, the decoder part of the AE network is directly fed into a supervised classification layer/network to increase decoding performance.
This is often referred to: pre-training with stacked AEs.

MEG High-frequency oscillations detection in epilepsy



A Stacked Sparse Autoencoder-Based Detector for Automatic Identification of Neuromagnetic High Frequency Oscillations in Epilepsy.
https://www.ncbi.nlm.nih.gov/pubmed/29994761

Motor imagery classification in EEG



A novel deep learning approach for classification of EEG motor imagery signals.

https://www.ncbi.nlm.nih.gov/pubmed/27900952

Other paper examples.

Deep neural network with weight sparsity control and pre-training extracts hierarchical features and enhances classification performance: Evidence from whole-brain resting-state functional connectivity patterns of schizophrenia.
https://www.ncbi.nlm.nih.gov/pubmed/25987366

Latent feature representation with stacked auto-encoder for AD/MCI diagnosis.
https://www.ncbi.nlm.nih.gov/pubmed/24363140

Stacked Autoencoders for the P300 Component Detection.
https://www.ncbi.nlm.nih.gov/pubmed/28611579

Semi-supervised Stacked Label Consistent Autoencoder for Reconstruction and Analysis of Biomedical Signals.
https://www.ncbi.nlm.nih.gov/pubmed/27893378

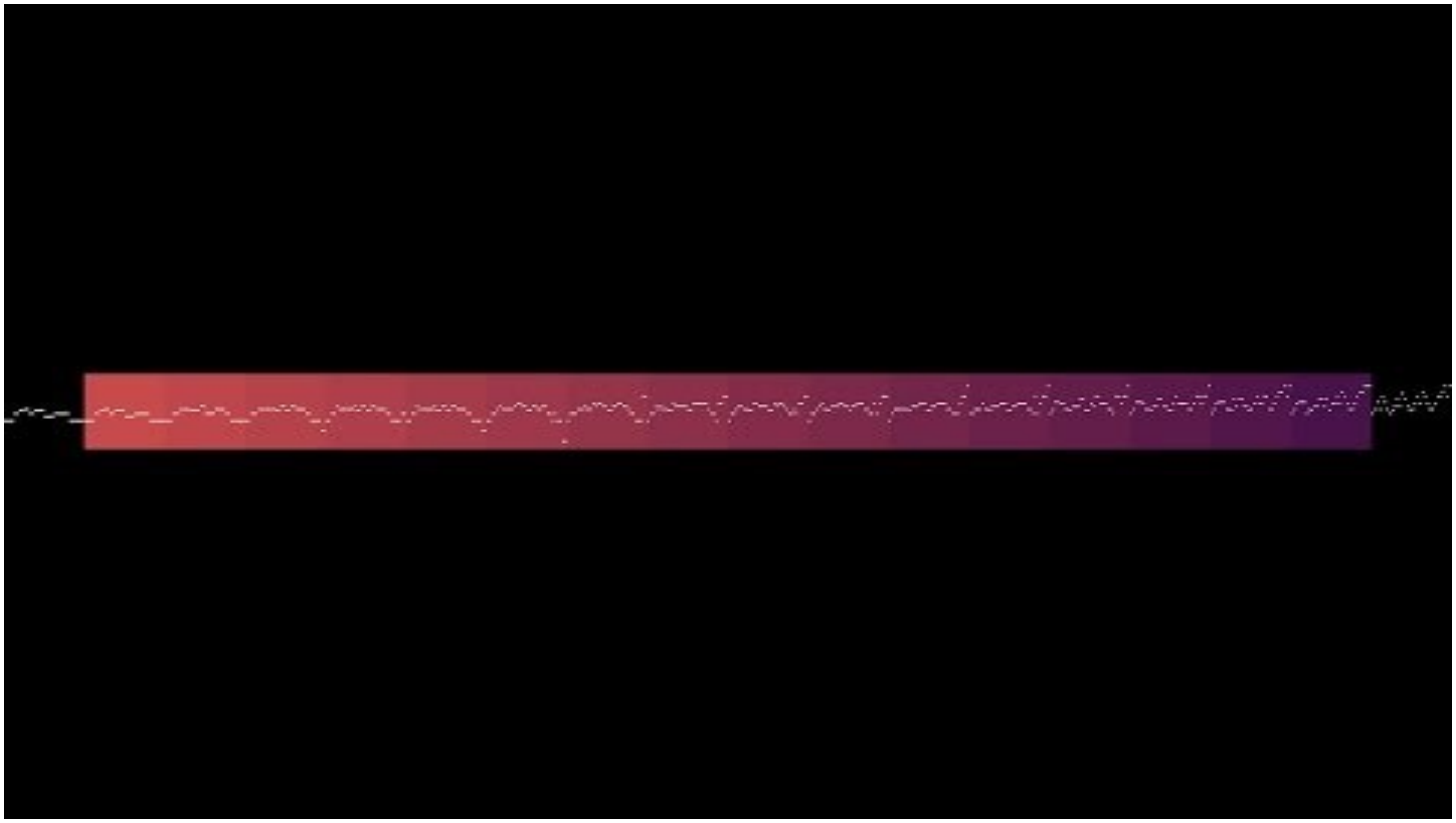
Other paper examples.

A stacked contractive denoising auto-encoder for ECG signal denoising.
https://www.ncbi.nlm.nih.gov/pubmed/27869101

Automatic Sleep Stage Scoring Using Time-Frequency Analysis and Stacked Sparse Autoencoders.
https://www.ncbi.nlm.nih.gov/pubmed/26464268

Feature extraction with stacked autoencoders for epileptic seizure detection.
https://www.ncbi.nlm.nih.gov/pubmed/25570914

“Research” using (V)AEs



https://magenta.tensorflow.org/music-vae

Thanks!

Questions?