03 Dynamic Representation

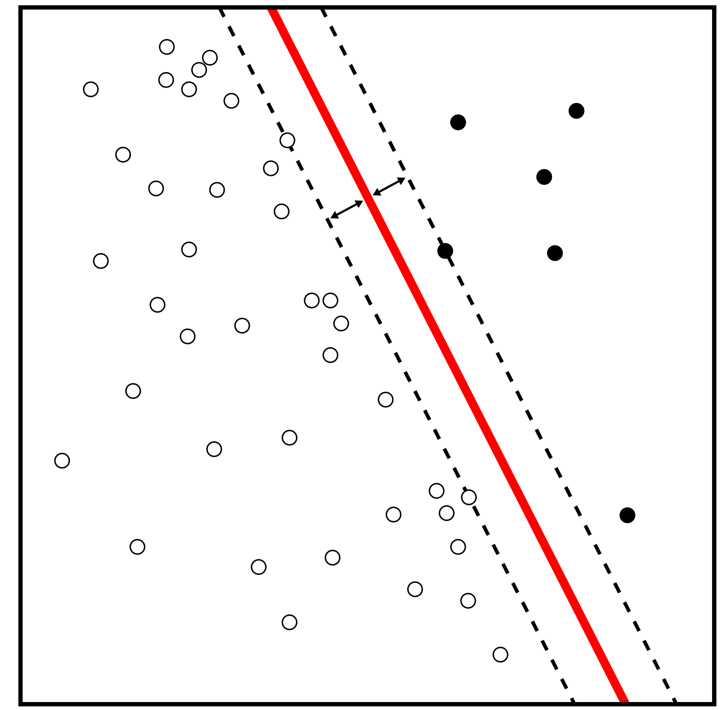
Advanced Machine Learning

Malte Schilling, Neuroinformatics Group, Bielefeld University

Goals for Today

Support Vector Machines

Support Vector Machine



from Wikipedia:Support-vector machine

Distance (geometric margin) of data points to border:

$$y^{(i)}\left(\left(\frac{\mathbf{w}}{\|\mathbf{w}\|}\right)^T x^{(i)} + \frac{b}{\|\mathbf{w}\|}\right) = \gamma^{(i)}$$

For details and derivation see (Ng 2018)

Finding the Largest Margin

$$\gamma = min_{i=1,\ldots,m}\ \gamma^{(i)} = min_{i=1,\ldots,m}\ y^{(i)}\left(\left(\frac{\mathbf{w}}{\|\mathbf{w}\|}\right)^T x^{(i)} + \frac{b}{\|\mathbf{w}\|}\right)$$

$$\Rightarrow max_{\gamma,\mathbf{w},b}\ y^{(i)}\left(\mathbf{w}^T x^{(i)} + b\right) \geq \gamma, i = 1,\ldots,m, \text{ with } \|\mathbf{w}\| = 1$$

Unfortunately, the constraint on $\mathbf{w}$ makes this none convex and not directly solvable.

$$\Rightarrow max_{\hat{\gamma},\mathbf{w},b}\ \frac{\hat{\gamma}}{\|\mathbf{w}\|}, \text{introducing a functional margin as the relation } \gamma = \frac{\hat{\gamma}}{\|\mathbf{w}\|}$$

$$\text{s.t. } y^{(i)}\left(\mathbf{w}^T x^{(i)} + b\right) \geq \hat{\gamma}, i = 1,\ldots,m$$

Now, we can freely choose $\hat{\gamma}$ which requires us to find appropriate weights. We set $\hat{\gamma} = 1$ and can now instead solve:

$$min_{\mathbf{w},b}\ \frac{1}{2}\|\mathbf{w}\|^2, \text{ s.t. } y^{(i)}\left(\mathbf{w}^T x^{(i)} + b\right) \geq 1, i = 1,\ldots,m$$

In General: Lagrangian Multiplier

We can reformulate a (primal) optimization problem:

$$
\begin{aligned}
min_{\mathbf{w}}\ & f(\mathbf{w}) \\
\text{s.t. } & g_i(\mathbf{w}) \leq 0, i = 1, \ldots, k \\
& h_j(\mathbf{w}) = 0, j = 1, \ldots, l
\end{aligned}
$$

And instead solve the *generalized Lagrangian* (when the Karush-Kuhn-Tucker conditions are met):

$$\mathcal{L}(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathbf{w}) + \sum_{j=1}^{L} \beta_j h_j(\mathbf{w})$$

α_i and β_j are the Lagrangian multipliers.

Apply Lagrangian Multiplier

$$min_{\mathbf{w},b}\ \frac{1}{2}\|\mathbf{w}\|^2,\ \text{s.t. } y^{(i)}\left(\mathbf{w}^T x^{(i)}+b\right)\geq 1, i=1,\ldots,m$$

We can formulate $f(\mathbf{w})=1/2\|\mathbf{w}\|^2$ and the constraints as:

$$g_i(\mathbf{w})=-y^{(i)}\left(\mathbf{w}^T x^{(i)}+b\right)+1\leq 0, i=1,\ldots,m$$

and now solve the Lagrangian

$$\mathcal{L}(\mathbf{w},b,\boldsymbol{\alpha})=\frac{1}{2}\|\mathbf{w}\|^2-\sum_{i=1}^{m}\alpha_i\left(y^{(i)}\left(\mathbf{w}^T x^{(i)}+b\right)-1\right)$$

by setting the derivatives to zero ($\nabla_b\mathcal{L}=0$ because of the KKT conditions):

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w},b,\boldsymbol{\alpha})=\mathbf{w}-\sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)}=0\Rightarrow\mathbf{w}=\sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)}$$

Finding optimal weights

This gives us the optimal weights when having found the Lagrangian multipliers on our inputs:

$$\mathbf{w}^* = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$$

Importantly, when now use this for a novel input $\mathbf{x}'$ we would apply ${\mathbf{w}^*}^T \mathbf{x}' + b$ and decide classification based on the sign.

Using the above equality, we can reformulate this as

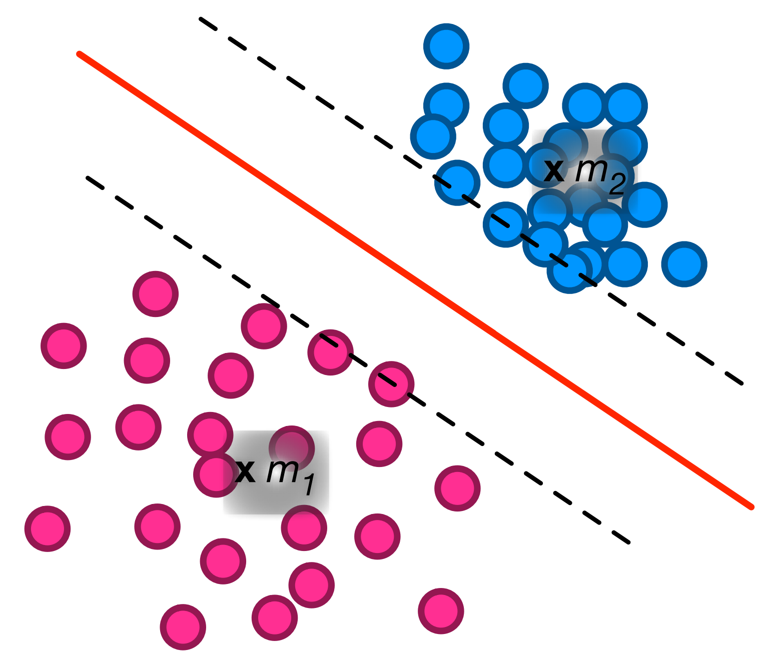
$$\begin{aligned} {\mathbf{w}^*}^T \mathbf{x}' + b &= \left(\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} \right)^T \mathbf{x}' + b \\ &= \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, \mathbf{x}' \rangle + b \end{aligned}$$

Largest margin Separation

- this only involves some data points (support vectors)
- the constrained optimization can be solved through a Lagrange multiplier
- this leads to the hyperplane decision function

$$\alpha_i \geq 0,$$

$$f(\mathbf{x}) = sgn(\sum_{i=1}^{m} \alpha^{(i)} y^{(i)} \langle \mathbf{x}, \mathbf{x}^{(i)} \rangle + b\ )$$

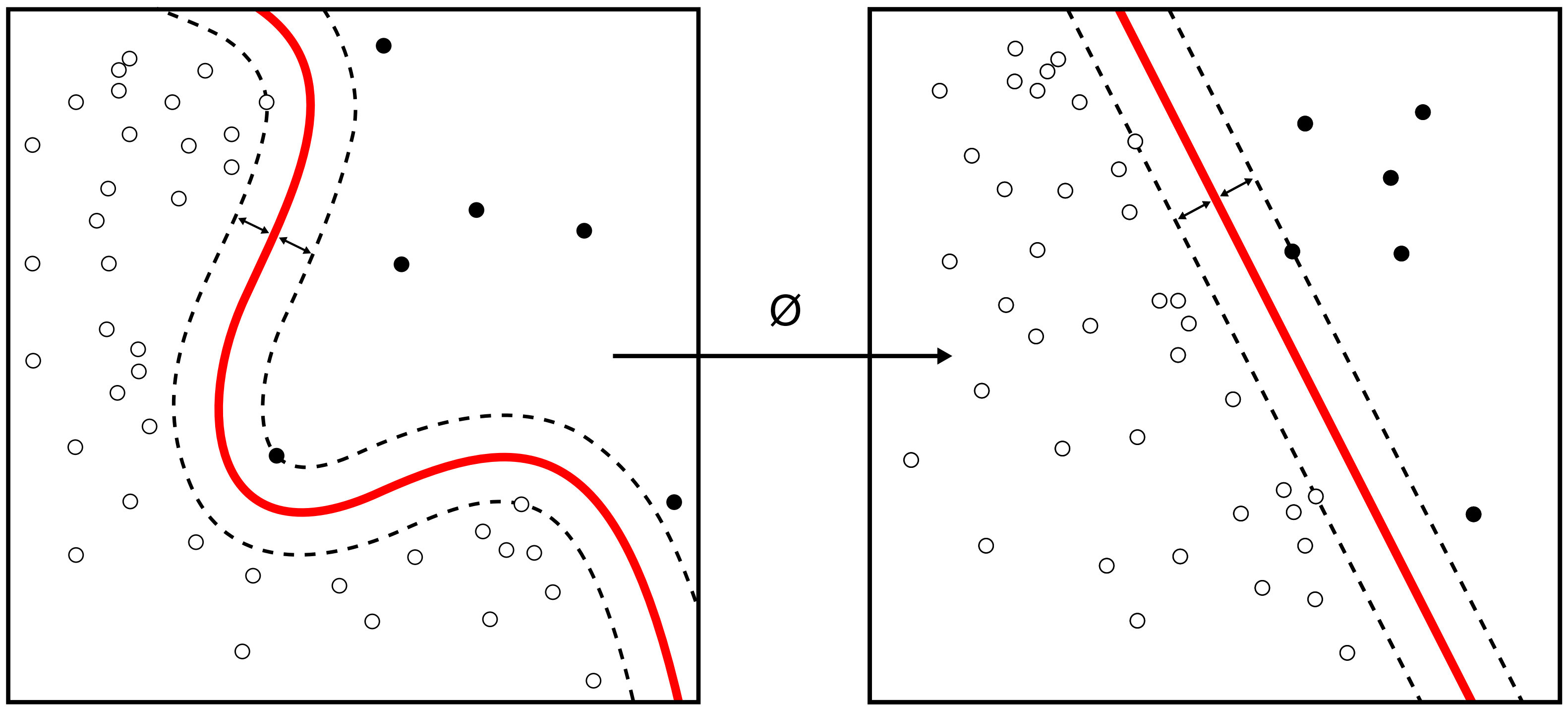


$$\max_{\mathbf{w},b} \min\{\|\mathbf{x} - \mathbf{x}^{(i)}\|\}$$

$$with \langle \mathbf{w}, \mathbf{x} \rangle + b = 0 \text{ defining the hyperplane}$$

Application of Kernel

Support Vector Machine



SVMs go back to (Vapnik 1998) , and a good tutorial can be found in (Burges 1998).

Kernel Trick

The kernel trick for kernel methods as SVMs is a substitution:

- All computations can be formulated in a scalar product space.
- We introduce a kernel function – this express the scalar product in the higher dimensional feature space in terms of the lower-dimensional input space.
- The kernel function evaluates the function and scalar product of the feature space only from the lower-dimensional input space.

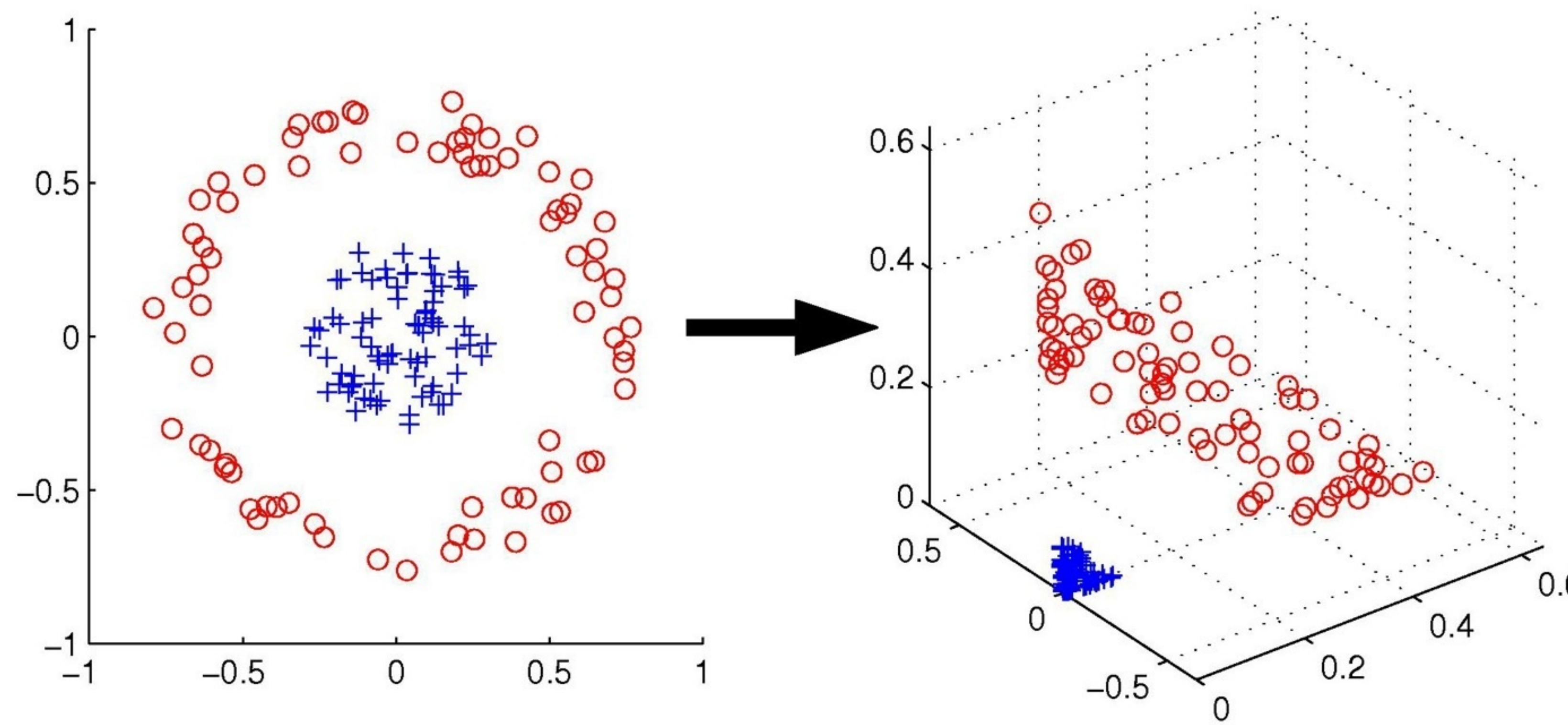
$$\text{e.g., } k(x, x') := \langle \mathbf{x}, \mathbf{x}' \rangle$$

Recap – Example for Application of Kernel

Kernel functions provide mappings that allow for separability:

$$\phi(\mathbf{x}) \rightarrow x_1^2, x_2^2, \sqrt{2}x_1 x_2$$

Importantly, the scalar product is not computed explicitly in the feature space. It can be applied in the input space – Kernel Trick.

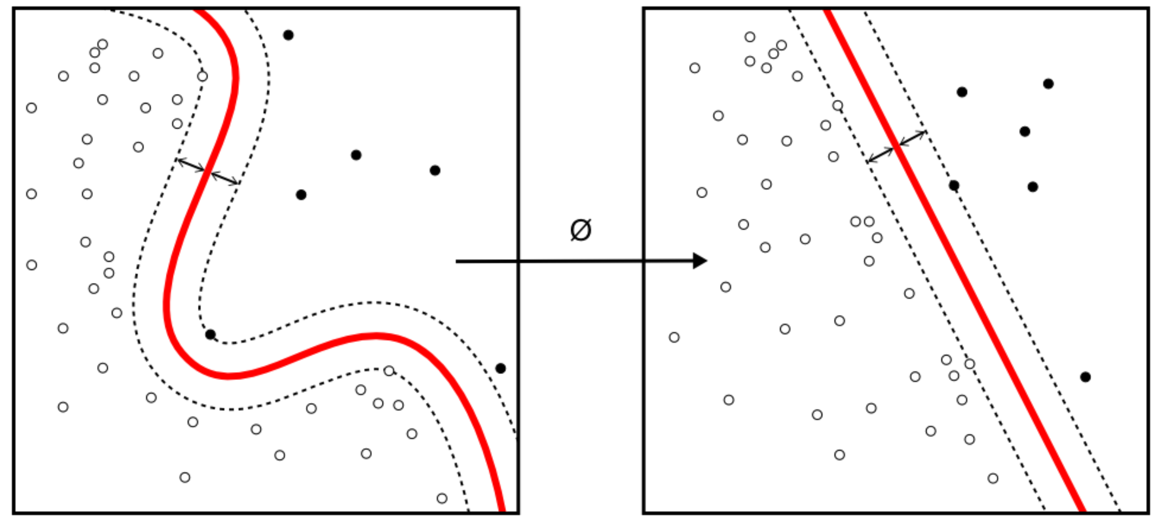


Kernel trick

$$\phi(\mathbf{x}) \to x_1^2, x_2^2, \sqrt{2}x_1x_2$$

$$\begin{aligned}
\langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle &= \langle (x_1^2, x_2^2, \sqrt{2}x_1x_2), ({x'}_1^2, {x'}_2^2, \sqrt{2}{x'}_1{x'}_2) \rangle \\
&= \underbrace{x_1^2{x'}_1^2}_{a^2} + \underbrace{x_2^2{x'}_2^2}_{b^2} + \underbrace{2x_1x_2{x'}_1{x'}_2}_{2ab} \\
&= (\underbrace{x_1{x'}_1}_{a} + \underbrace{x_2{x'}_2}_{b})^2 \\
&= \langle \mathbf{x}, \mathbf{x}' \rangle^2 = k(\mathbf{x}, \mathbf{x}')
\end{aligned}$$

Summary: Support Vector Machine



- Support vector machines implement the large margin principle.
- They apply non-linear mappings.
- Importantly, the scalar product is not computed explicitly in the feature space. using the Kernel Trick. This is much more efficient.
- The kernel function (weighted by multipliers) is applied wrt. the support vectors.

SVMs go back to (Vapnik 1998) , and a good tutorial can be found in (Burges 1998).

Kernel functions

Polynomial kernel (of degree d)

$$k(x, x') := \langle x, x' \rangle^d$$

- Includes all polynomial terms up to degree d.

Radial Basis Function kernels

$$k(x, x') := exp(-\frac{\|x - x'\|^2}{2\sigma^2})$$

- Allows to map into an infinite dimensional feature space (Gaussian kernel can be constructed from an infinite sum over polynomial kernels).
- Scales with number of features.

SVM – Advantages

- Very robust, guaranteed to be a global minimum
- Work well on small (and high dimensional) data spaces.
- Does allow for non-linearly separable data (using Kernel trick).
- Can be softened through a simple parameter allowing for violation of the maximum margin.
- Is efficient for high-dimensional datasets as the complexity is characterized by the number of support vectors.
- Support Vectors can help to understand the problem better.
- Only a small number of hyperparameters.

SVM – Disadvantages

- Not suitable for big datasets as the training time with SVMs becomes much more computationally intensive.
- They are less effective on noisier datasets with overlapping classes.
- Are often outperformed by Deep Neural Networks.

Commercial Break

CITEC Conference today and tomorrow

For information see Conference Webpage.

Reservoir Computing

From Dynamical Features ...

Temporal Filters can be seen as dynamical systems that compute at each time step a state that is some function of previous states and the current input:

$$s_t = F(s_{t-1}, s_{t-2}, \ldots s_{t-k}, x_t, x_{t-1}, \ldots, x_{t-l})$$

In the simplest case, the function is linear in its arguments, leading to the well-known recursive filters

$$s_t = \sum_{i=1}^{K} a_i s_{t-1} + \sum_{j=0}^{L} b_j x_{t-j}$$

that allow, e.g., to selectively damp/enhance specifiable frequency bands of the input time sequence. For example a smoothing filter:

$$s_t = (1-\gamma)s_{t-1} + \gamma x_t$$

… to Dynamical Systems

Yet, combining linear filters always leads back to a linear filter.

Richer processing can only occur when non-linearities are included.

For example, we can consider non-linear filters arising from recurrent neural networks as in reservoir computing.

Learning from Random Features

Simple learning approach in a feedforward neural network:

- using randomly initialized early layers and keep them fixed (comparable to expansion in SVMs) – use large input layers that provide diversity
- During learning: only adapt output weights – linear transformation of the (random) features.
- Such an expansion of the input space can facilitate learning and allow for better separability.

Random Features in a Recurrent Neural Net

Echo state networks apply the same idea in a recurrent neural network:

- Initialize the recurrent neural network randomly and keep it fixed.
- The same holds true for the projection of the input onto the recurrent layers.
- Only train the connections towards the output layer which makes learning very simple.
- The recurrent part is called a reservoir – it should cover a diversity of dynamics that can be recruited.

Following (Hinton 2013) in his Advanced Machine Learning Course.

Setting the Connections inside the Reservoir

Crucial for Echo State Networks is the setup of the random recurrent connections:

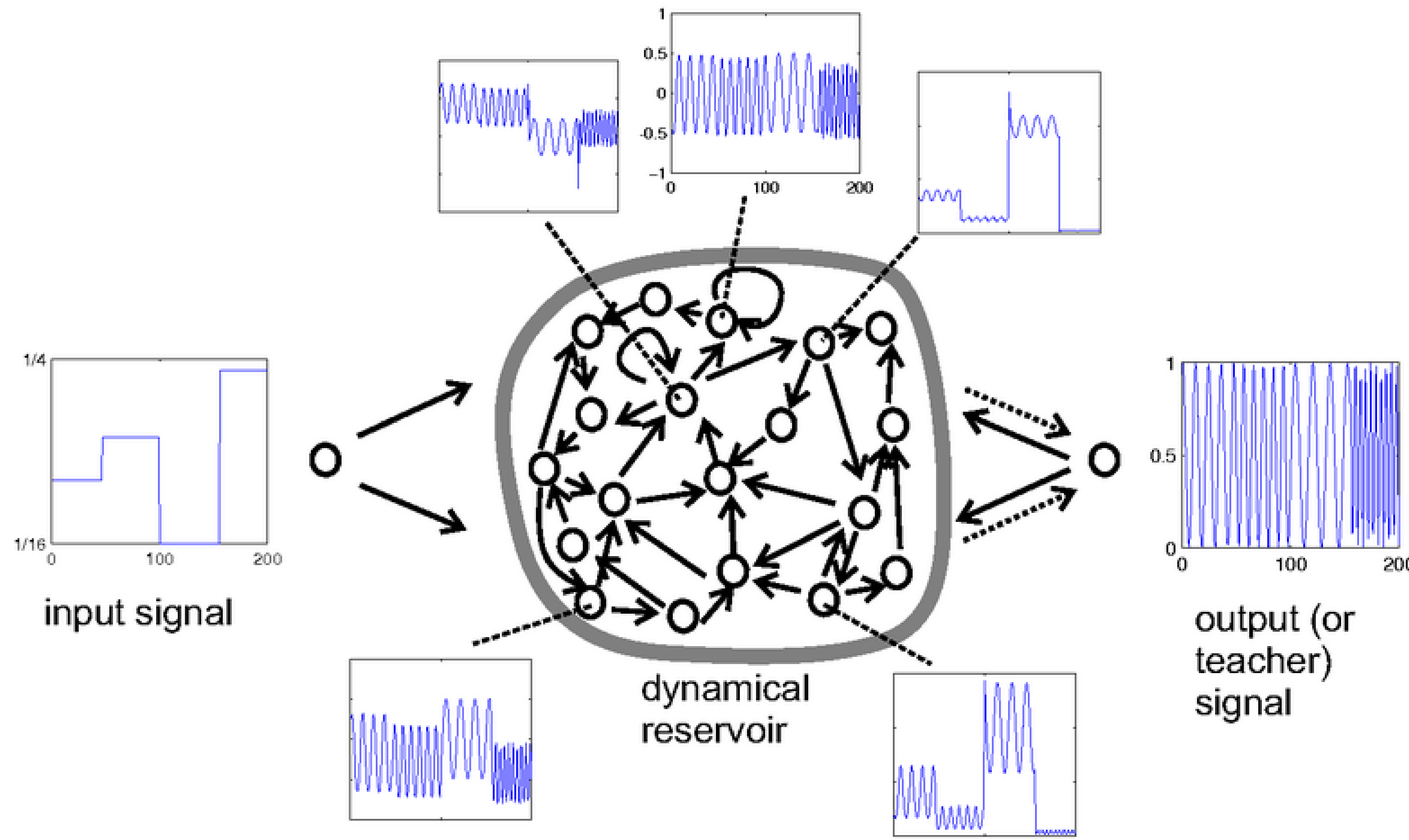
- They have to kept bound and fulfill the echo state property to prevent dying or exploding activations.
- Still, activities may decay too fast or too slowly. Therefore, the reservoir has to be tuned in a way that the dynamics of the features match the time scales of the application task.

Echo Property

Without external excitation all activities of the reservoir will decay slowly to zero. A criterion for this is that the spectral radius (the largest eigenvalue of $A^T A$) is less than 1 (or set to 1).

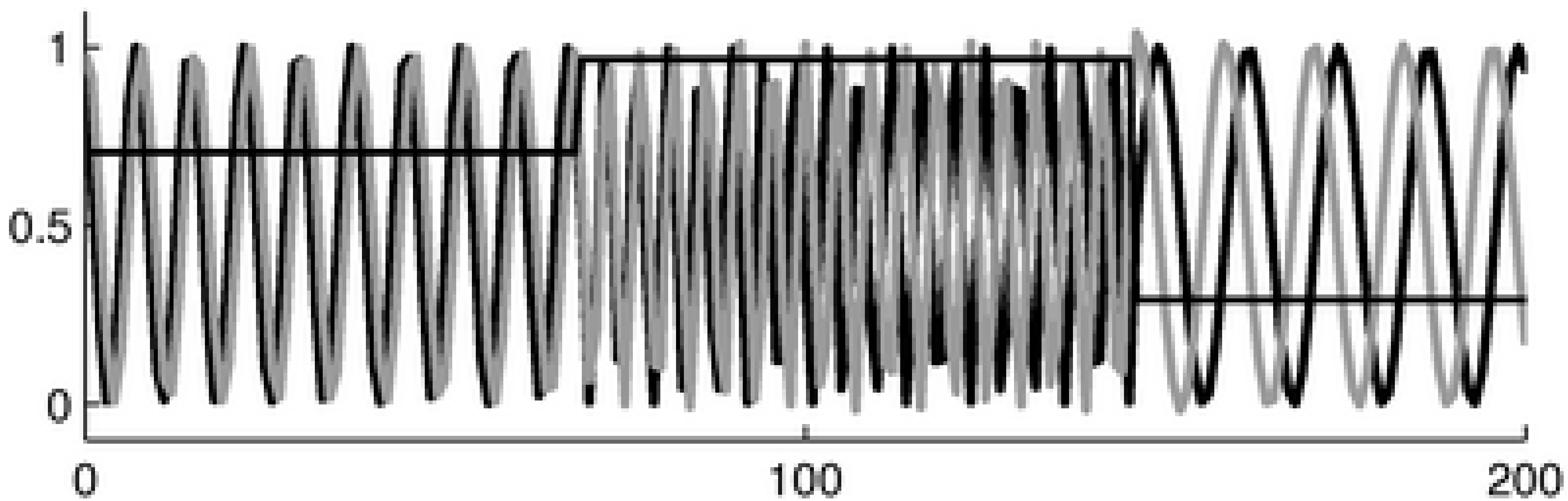
Echo State Network

- Input projects onto reservoir, here a real value.
- Target output: is a sine wave with the frequency given by the input.



(Jaeger 2007)

Echo State Network Example Results



A test run of the frequency generator from the previous slide.

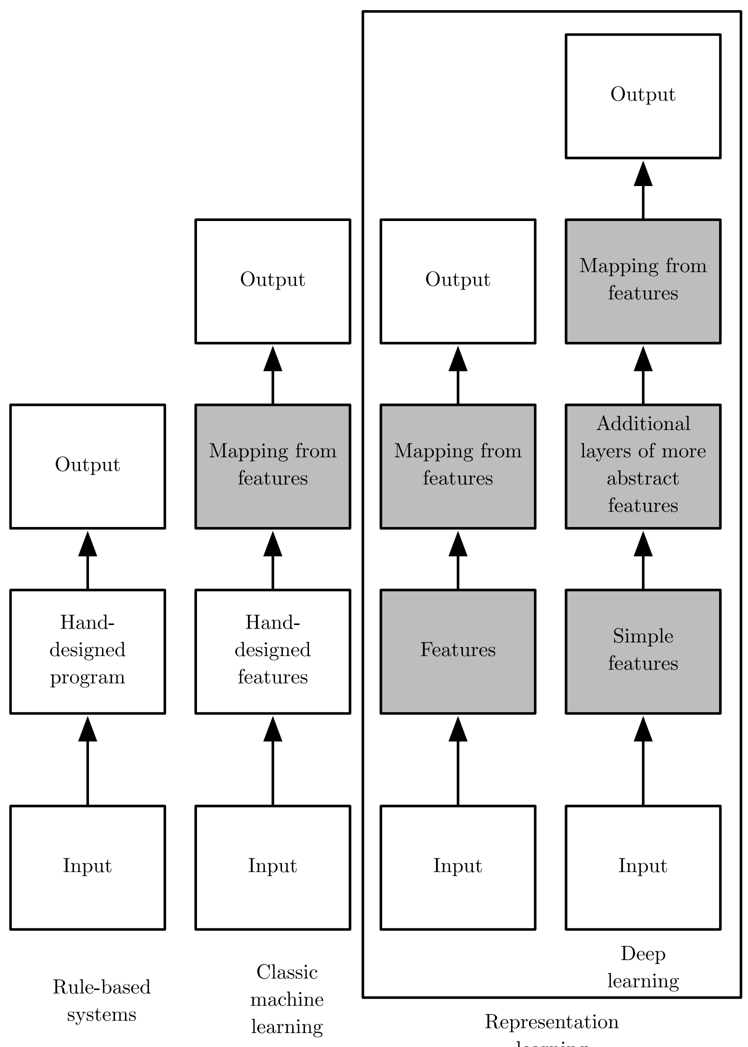
In the back, the input step function is shown.

The black sinewaves is the target output (unknown to the network).

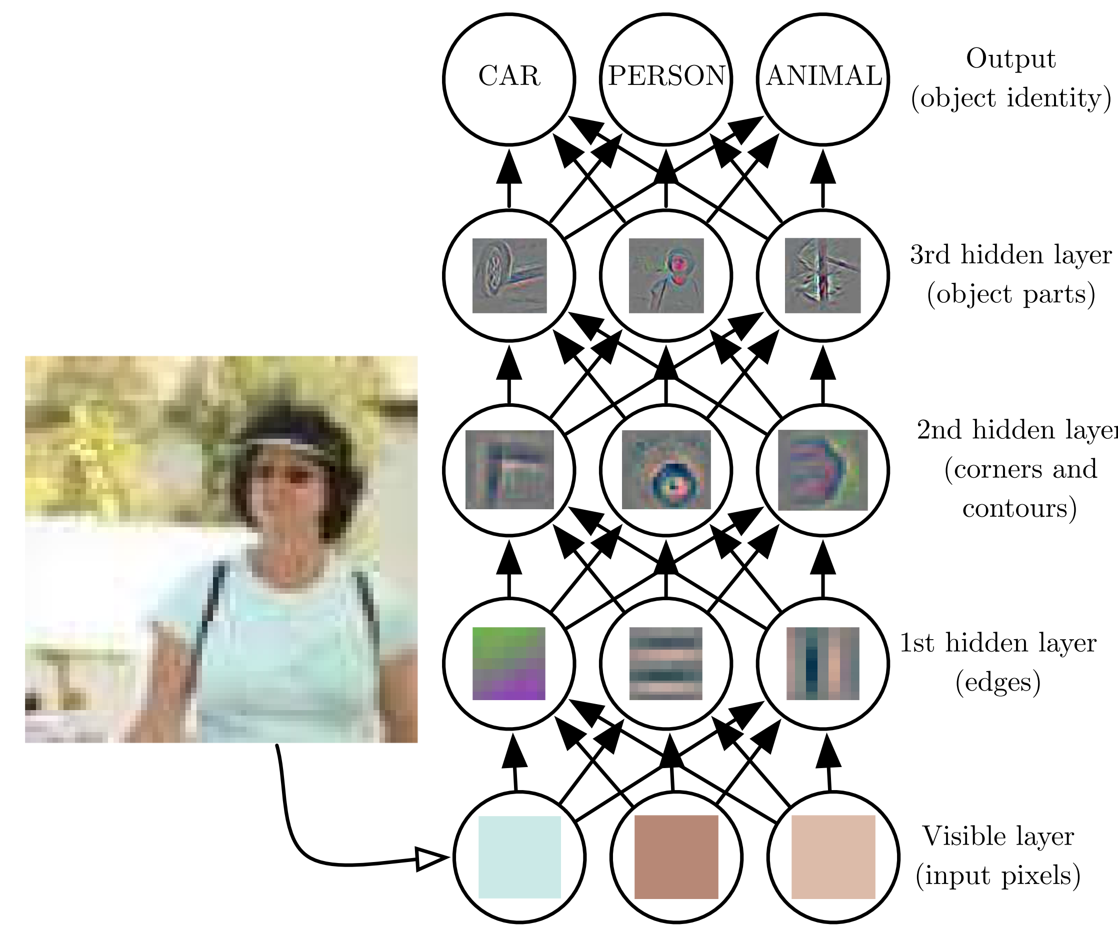
Recap – Representation Learning

Representation Learning

Current ML Pipeline



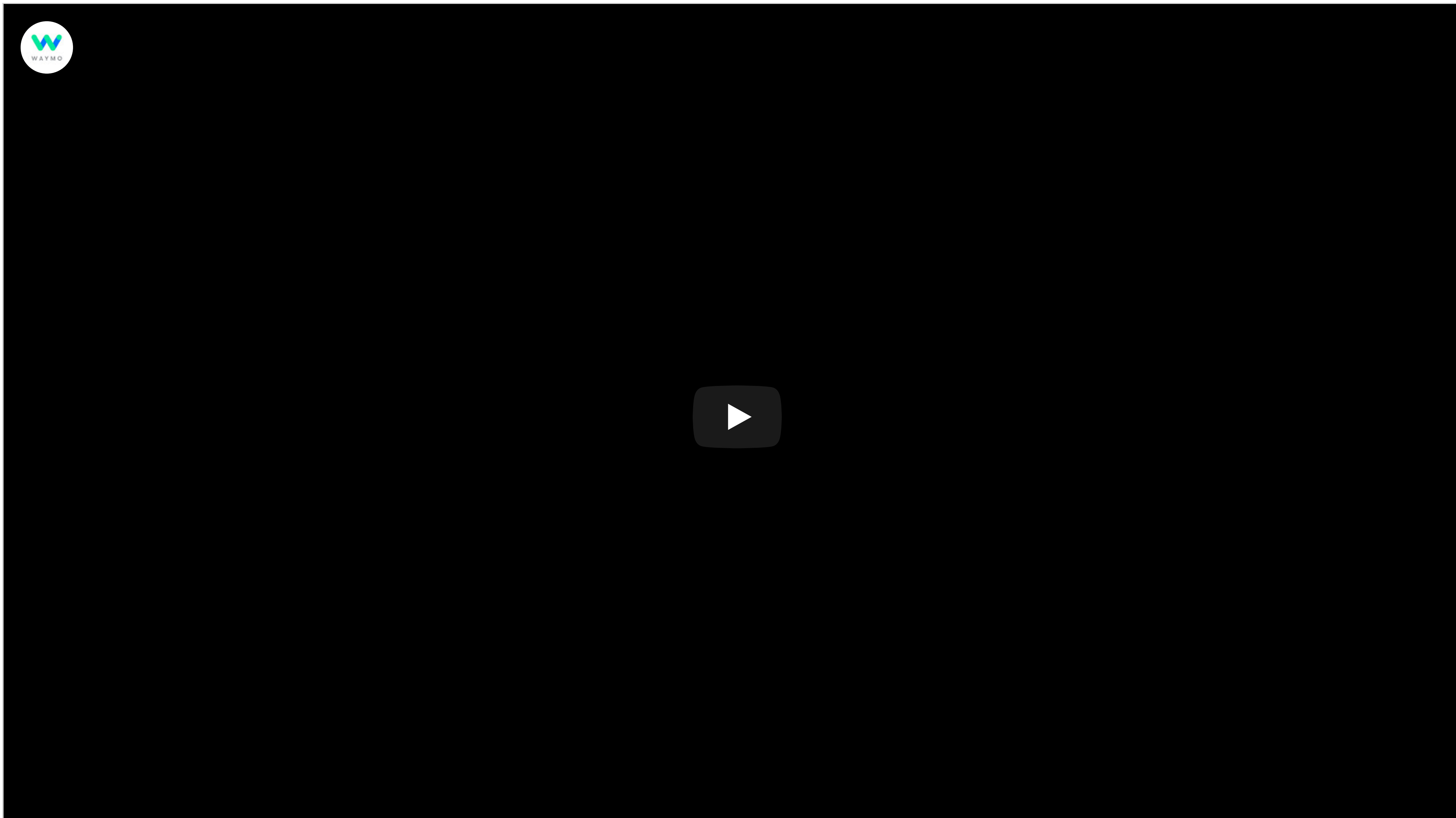
End-to-End Learning in Deep NN



(Goodfellow, Bengio, and Courville 2016)

Example: Waymo

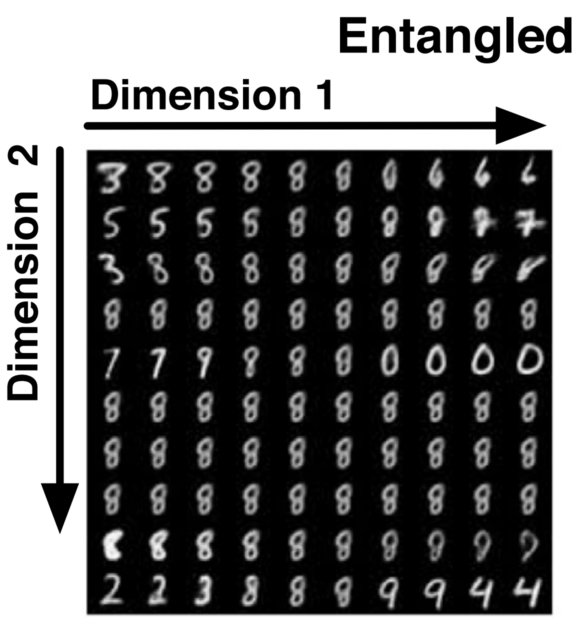
Scene Representation in Autonomous Driving



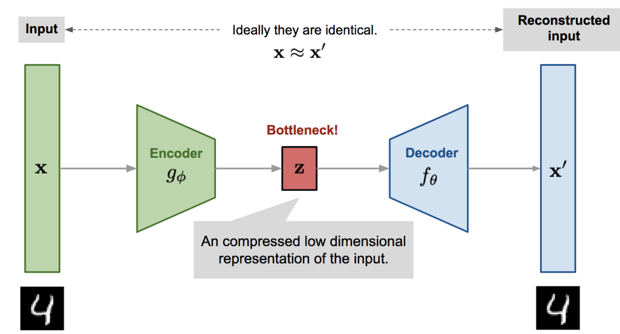
Features: Transfer Learning

Autoencoder (Weng 2018)

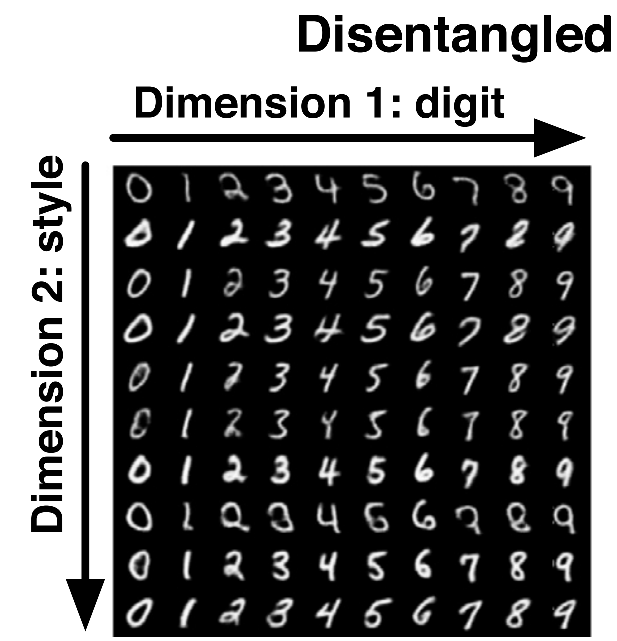
Entangled Representation



Autoencoder

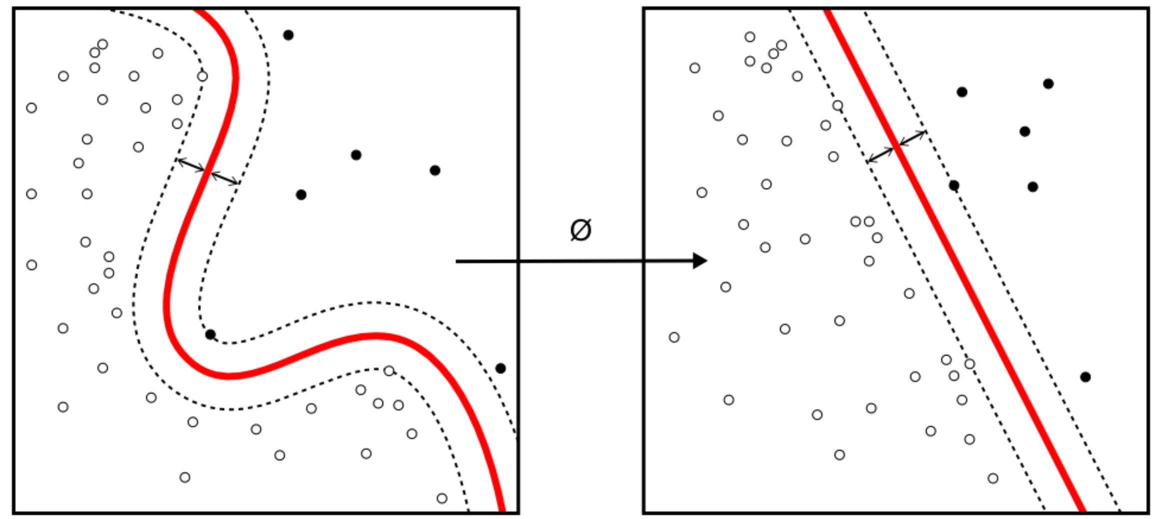


Disentangled Representation



- Encoder translates high-dimension input into latent low-dimensional code.
- Decoder recovers data from the code.

Support Vector Machine



- Support vector machines implement the large margin principle.
- They apply non-linear mappings.
- Importantly, the scalar product is not computed explicitly in the feature space. using the Kernel Trick. This is much more efficient.
- The kernel function (weighted by multipliers) is applied wrt. the support vectors.

SVMs go back to (Vapnik 1998) , and a good tutorial can be found in (Burges 1998).

References

Burges, Christopher J. C. 1998. “A Tutorial on Support Vector Machines for Pattern Recognition.” *Data Mining and Knowledge Discovery* 2: 121–67.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

Hinton, Geoffrey E. 2013. “Recurrent Neural Networks.” CSC 2535: Advanced Machine Learning Course.

Jaeger, Herbert. 2007. “Echo State Network.” *Scholarpedia* 2 (9): 2330. https://doi.org/10.4249/scholarpedia.2330.

Nayak, Sunita. 2019. “Image Classification Using Transfer Learning in Pytorch.” 2019. https://www.learnopencv.com/image-classification-using-transfer-learning-in-pytorch/.

Ng, Andrew. 2018. “Support Vector Machines.” Course CS229, Stanford University, Lecture Notes.

Vapnik, Vladimir N. 1998. *Statistical Learning Theory*. Wiley-Interscience.