I Regression

Task (probabilistic view):

Given feature x, model state w as probability distribution

$P(w|x)$

by: 1. Choosing an appropriate form for prior $P(w)$
2. Parametrize function $w = f(x; \Theta)$

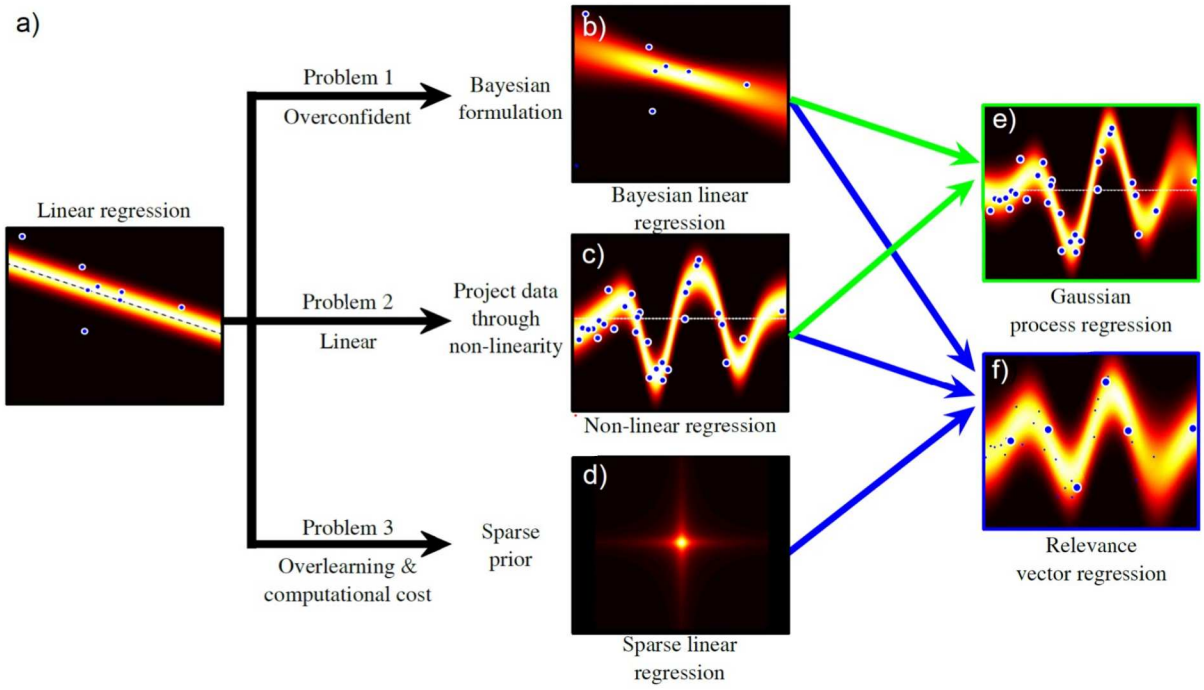
Learning: Learn parameters Θ from training data x, w

Inference: Evaluate $P(w|x)$

Notation: Add bias to input vector

$$\mathbf{x}_i \leftarrow [1 \quad \mathbf{x}_i^T]^T \quad \boldsymbol{\phi} \leftarrow [\phi_0 \quad \boldsymbol{\phi}^T]^T$$

Regression Models:



Linear Regression

Goal: Model $Pr(w_i|\mathbf{x}_i, \boldsymbol{\theta}) = \text{Norm}_{w_i}\left[\boldsymbol{\phi}^T\mathbf{x}_i, \sigma^2\right]$

Log representation:

$$\log \text{Norm}_{\mathbf{w}}[\mathbf{X}^T\boldsymbol{\phi}, \sigma^2\mathbf{I}] = \log[(2\pi)^{-I/2}] + \log[(\sigma^2)^{-I/2}] + \log\exp\left[-\frac{(\mathbf{w}-\mathbf{X}^T\boldsymbol{\phi})^T(\mathbf{w}-\mathbf{X}^T\boldsymbol{\phi})}{2\sigma^2}\right]$$
$$= -\frac{I\log[(2\pi)]}{2} - \frac{I\log[\sigma^2]}{2} - \frac{(\mathbf{w}-\mathbf{X}^T\boldsymbol{\phi})^T(\mathbf{w}-\mathbf{X}^T\boldsymbol{\phi})}{2\sigma^2}$$

Learning (direct solution): Maximize log-likelihood

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\text{argmax}}\left[Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\theta})\right] = \underset{\boldsymbol{\theta}}{\text{argmax}}\left[\log Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\theta})\right]$$

with: $$\hat{\boldsymbol{\phi}}, \hat{\sigma}^2 = \underset{\boldsymbol{\phi}, \sigma^2}{\text{argmax}}\left[-\frac{I\log[2\pi]}{2} - \frac{I\log[\sigma^2]}{2} - \frac{(\mathbf{w}-\mathbf{X}^T\boldsymbol{\phi})^T(\mathbf{w}-\mathbf{X}^T\boldsymbol{\phi})}{2\sigma^2}\right]$$

Solution:

$$\hat{\boldsymbol{\phi}} = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{w}$$
$$\hat{\sigma}^2 = \frac{(\mathbf{w}-\mathbf{X}^T\boldsymbol{\phi})^T(\mathbf{w}-\mathbf{X}^T\boldsymbol{\phi})}{I}$$

Derivation:

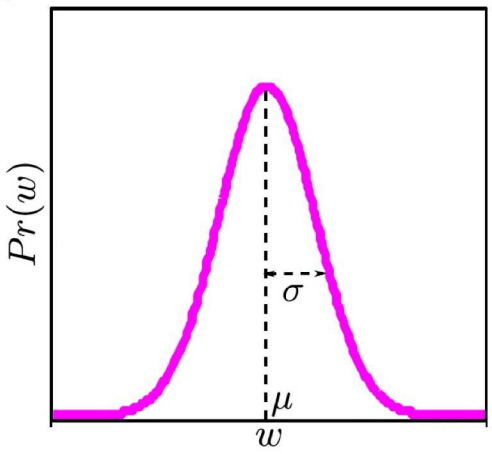
1. Take derivative of log probability and set to zero

$$-\frac{1}{2\sigma^2}\frac{\partial}{\partial\phi}(w - X^T\phi)^T(w - X^T\phi) = \frac{1}{\sigma^2}X(w - X^T\phi) = 0$$

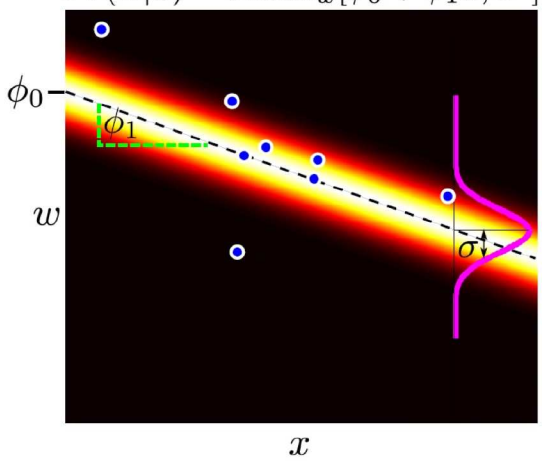
2. Re-arrange

$$Xw = XX^T\phi \qquad \frac{(w - X^T\phi)^T(w - X^T\phi)}{\sigma^2} = I \qquad \hat{\phi} = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{w}$$
$$\phi = (XX^T)^{-1}Xw \iff \sigma^2 = \frac{(w - X^T\phi)^T(w - X^T\phi)}{I} \iff \hat{\sigma}^2 = \frac{(\mathbf{w}-\mathbf{X}^T\boldsymbol{\phi})^T(\mathbf{w}-\mathbf{X}^T\boldsymbol{\phi})}{I}$$

a)



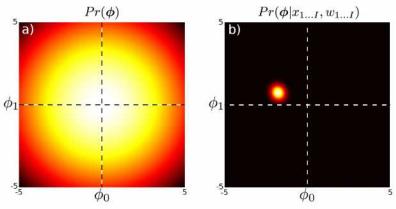
b) $Pr(w|x) = \text{Norm}_w[\phi_0 + \phi_1 x, \sigma^2]$



$$Pr(w_i|\mathbf{x}_i, \boldsymbol{\theta}) = \text{Norm}_{w_i}\left[\phi_0 + \boldsymbol{\phi}^T\mathbf{x}_i, \sigma^2\right]$$

Bayesian Regression

Goal: Model $$Pr(\phi|\mathbf{X},\mathbf{w}) = \frac{Pr(\mathbf{w}|\mathbf{X},\phi)Pr(\phi)}{Pr(\mathbf{w}|\mathbf{X})}$$



$$Pr(\phi|\mathbf{X},\mathbf{w}) = \text{Norm}_{\phi}\left[\frac{1}{\sigma^2}\mathbf{A}^{-1}\mathbf{X}\mathbf{w}, \mathbf{A}^{-1}\right] \quad \mathbf{A} = \frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T + \frac{1}{\sigma_p^2}\mathbf{I}$$

Derivation:

1. Disregard evidence term
2. Using product rule simplify the term

Product rule: $$\text{Norm}_{\mathbf{x}}[\mathbf{a},\mathbf{A}]\text{Norm}_{\mathbf{x}}[\mathbf{b},\mathbf{B}] = \kappa \cdot \text{Norm}_{\mathbf{x}}\left[\left(\mathbf{A}^{-1}+\mathbf{B}^{-1}\right)^{-1}\left(\mathbf{A}^{-1}\mathbf{a}+\mathbf{B}^{-1}\mathbf{b}\right), \left(\mathbf{A}^{-1}+\mathbf{B}^{-1}\right)^{-1}\right]$$

$$\text{Norm}_{\phi}\left[\left(\frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T\right)^{-1}\frac{1}{\sigma^2}\mathbf{X}\mathbf{w}, \left(\frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T\right)^{-1}\right]\text{Norm}_{\phi}\left[\mathbf{0}, \sigma_p^2\mathbf{I}\right] =$$

$$\text{Norm}_{\phi}\left[\left(\frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T + \frac{1}{\sigma_p^2}\mathbf{I}\right)^{-1}\left(\frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T\left(\frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T\right)^{-1}\frac{1}{\sigma^2}\mathbf{X}\mathbf{w}\right), \left(\frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T + \frac{1}{\sigma_p^2}\mathbf{I}\right)^{-1}\right] =$$

$$\text{Norm}_{\phi}\left[\frac{1}{\sigma^2}\left(\frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T + \frac{1}{\sigma_p^2}\mathbf{I}\right)^{-1}\mathbf{X}\mathbf{w}, \left(\frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T + \frac{1}{\sigma_p^2}\mathbf{I}\right)^{-1}\right]$$

Likelihood: $$Pr(\mathbf{w}|\mathbf{X},\phi) = \text{Norm}_{\mathbf{w}}[\mathbf{X}^T\phi, \sigma^2\mathbf{I}]$$

$$\text{Norm}_{\mathbf{w}}[\mathbf{X}^T\phi, \sigma^2\mathbf{I}] = \text{Norm}_{\phi}\left[\left(\frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T\right)^{-1}\frac{1}{\sigma^2}\mathbf{X}\mathbf{w}, \left(\frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T\right)^{-1}\right]$$

Prior: $$Pr(\phi) = \text{Norm}_{\phi}[\mathbf{0}, \sigma_p^2\mathbf{I}]$$

Inference: Integrate over parameter space

$$\begin{aligned} Pr(w^*|\mathbf{x}^*,\mathbf{X},\mathbf{w}) &= \int Pr(w^*|\mathbf{x}^*,\phi)Pr(\phi|\mathbf{X},\mathbf{w})d\phi \\ &= \int \text{Norm}_{w^*}[\phi^T\mathbf{x}^*, \sigma^2]\text{Norm}_{\phi}\left[\frac{1}{\sigma^2}\mathbf{A}^{-1}\mathbf{X}\mathbf{w}, \mathbf{A}^{-1}\right]d\phi \\ &= \text{Norm}_{w^*}\left[\frac{1}{\sigma^2}\mathbf{x}^{*T}\mathbf{A}^{-1}\mathbf{X}\mathbf{w}, \mathbf{x}^{*T}\mathbf{A}^{-1}\mathbf{x}^* + \sigma^2\right]. \end{aligned}$$

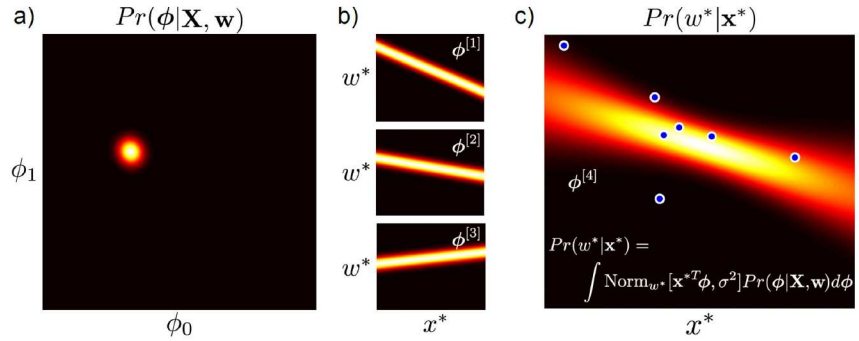
Problem: A might be too big to invert with a high-dimensional feature space D

Reformulation:

If D<I: $$\mathbf{A}^{-1} = \left(\frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T + \frac{1}{\sigma_p^2}\mathbf{I}_D\right)^{-1}$$

If I>D: $$= \sigma_p^2\mathbf{I}_D - \sigma_p^2\mathbf{X}\left(\mathbf{X}^T\mathbf{X} + \frac{\sigma^2}{\sigma_p^2}\mathbf{I}_I\right)^{-1}\mathbf{X}^T$$

Matrix inversion lemma: $$(\mathbf{A}^{-1} + \mathbf{B}^T\mathbf{C}^{-1}\mathbf{B})^{-1}\mathbf{B}^T\mathbf{C}^{-1} = \mathbf{A}\mathbf{B}^T(\mathbf{B}\mathbf{A}\mathbf{B}^T + \mathbf{C})^{-1}$$



Fitting Variance: Optimize the variance by maximizing the likelihood

$$\begin{aligned} Pr(\mathbf{w}|\mathbf{X},\sigma^2) &= \int Pr(\mathbf{w}|\mathbf{X},\boldsymbol{\phi},\sigma^2)Pr(\boldsymbol{\phi})\, d\boldsymbol{\phi} \\ &= \int \text{Norm}_{\mathbf{w}}[\mathbf{X}^T\boldsymbol{\phi},\sigma^2\mathbf{I}]\text{Norm}_{\boldsymbol{\phi}}[\mathbf{0},\sigma_p^2\mathbf{I}]\, d\boldsymbol{\phi} \\ &= \text{Norm}_{\mathbf{w}}[\mathbf{0},\sigma_p^2\mathbf{X}^T\mathbf{X}+\sigma^2\mathbf{I}] \end{aligned}$$

→ can be estimated by grid-search or non-linear optimization

Non-linear Regression

Goal: Keep principles of linear regression and embed the input into a feature space using a non-linear embedding function

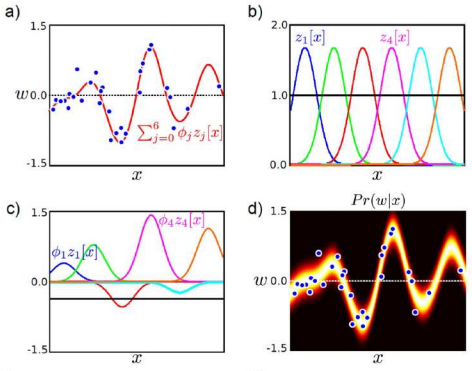
$$Pr(w_i|\mathbf{x}_i,\boldsymbol{\theta}) = \text{Norm}_{w_i}[\boldsymbol{\phi}^T\mathbf{z}_i, \sigma^2] \quad \text{with:} \quad \mathbf{z}_i = \mathbf{f}[\mathbf{x}_i]$$

Embedding function:

Radial basis function:

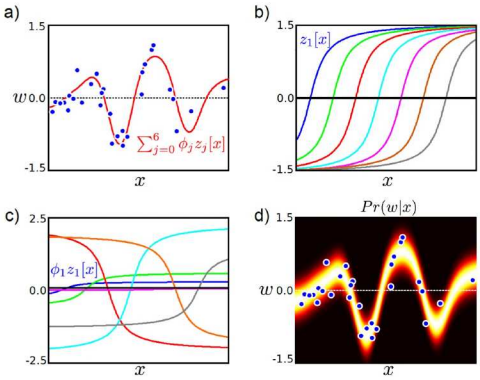
α_i: center
λ: determines width

$$\mathbf{z}_i = \begin{bmatrix} 1 \\ \exp\left[-(x_i-\alpha_1)^2/\lambda\right] \\ \exp\left[-(x_i-\alpha_2)^2/\lambda\right] \\ \exp\left[-(x_i-\alpha_3)^2/\lambda\right] \\ \exp\left[-(x_i-\alpha_4)^2/\lambda\right] \\ \exp\left[-(x_i-\alpha_5)^2/\lambda\right] \\ \exp\left[-(x_i-\alpha_6)^2/\lambda\right] \end{bmatrix}$$



Arc tan function:

$$\mathbf{z}_i = \begin{bmatrix} \arctan[\lambda x_i - \alpha_1] \\ \arctan[\lambda x_i - \alpha_2] \\ \arctan[\lambda x_i - \alpha_3] \\ \arctan[\lambda x_i - \alpha_4] \\ \arctan[\lambda x_i - \alpha_5] \\ \arctan[\lambda x_i - \alpha_6] \\ \arctan[\lambda x_i - \alpha_7] \end{bmatrix}$$



Learning: (same as linear regression)

$$\begin{aligned} \hat{\boldsymbol{\phi}} &= (\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}\mathbf{w} \\ \hat{\sigma}^2 &= \frac{(\mathbf{w}-\mathbf{Z}^T\boldsymbol{\phi})^T(\mathbf{w}-\mathbf{Z}^T\boldsymbol{\phi})}{I} \end{aligned}$$

Gaussian Process Regression

Idea: Use the reformulation but instead of embedding the input into a feature space, compute the dot product in the feature space through a kernel function. → Kernel Trick

$$Pr(w^*|\mathbf{x}^*, \mathbf{X}, \mathbf{w}) = \text{Norm}_{w^*}\left[\frac{\sigma_p^2}{\sigma^2}\mathbf{K}[\mathbf{x}^*, \mathbf{X}]\mathbf{w} - \frac{\sigma_p^2}{\sigma^2}\mathbf{K}[\mathbf{x}^*, \mathbf{X}]\left(\mathbf{K}[\mathbf{X}, \mathbf{X}] + \frac{\sigma^2}{\sigma_p^2}\mathbf{I}\right)^{-1}\mathbf{K}[\mathbf{X}, \mathbf{X}]\mathbf{w}, \sigma_p^2\mathbf{K}[\mathbf{x}^*, \mathbf{x}^*] - \sigma_p^2\mathbf{K}[\mathbf{x}^*, \mathbf{X}]\left(\mathbf{K}[\mathbf{X}, \mathbf{X}] + \frac{\sigma^2}{\sigma_p^2}\mathbf{I}\right)^{-1}\mathbf{K}[\mathbf{X}, \mathbf{x}^*] + \sigma^2\right]$$

Kernels: Measure similarity between input vectors

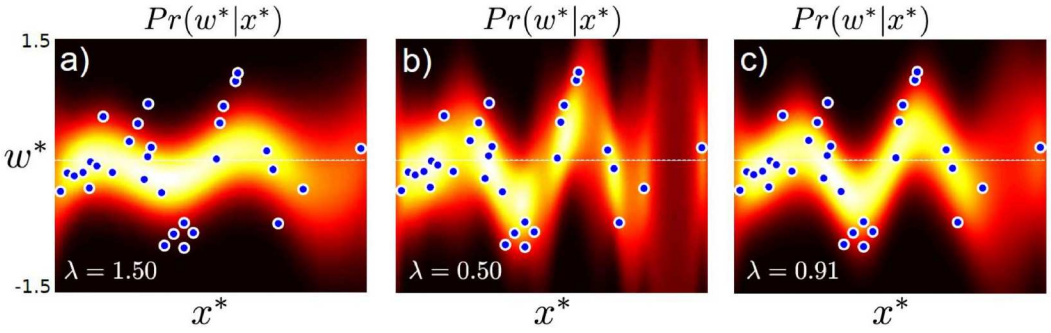
similar input → high response

Good kernel: close vectors in the input space have a high response and vica versa

Linear: $\mathrm{k}[\mathbf{x}_i, \mathbf{x}_j] = \mathbf{x}_i^T\mathbf{x}_j,$

Polynomial: $\mathrm{k}[\mathbf{x}_i, \mathbf{x}_j] = (\mathbf{x}_i^T\mathbf{x}_j + 1)^p$

RBF: $\mathrm{k}[\mathbf{x}_i, \mathbf{x}_j] = \exp\left[-0.5\left(\frac{(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j)}{\lambda^2}\right)\right]$



a) too high: all inputs are similar → no learning ability

b) too low: all inputs different → overfitting

Learning:

Inference:

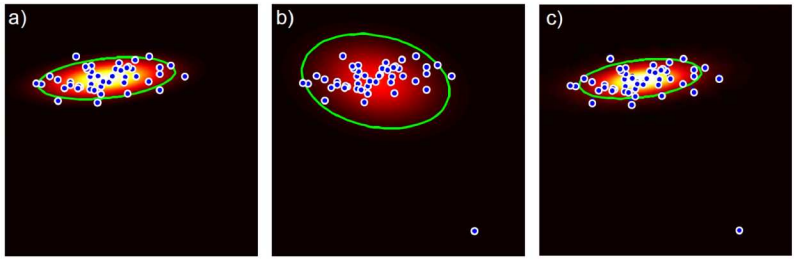
Fitting variance:

} same as non-linear regression

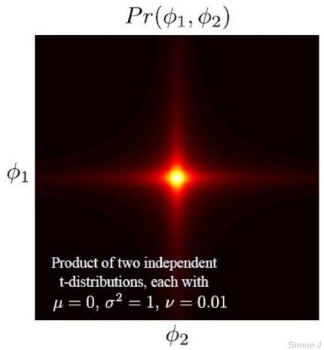
Sparse Linear Regression

Idea: Enforce a sparse solution forcing some coefficients ϕ to zero. Use a t-distribution as the prior to encourage sparsity.

Motivation:



Normal distribution | Normal distribution w/ one extra datapoint! | t-distribution



→ regularization encourages better generalization

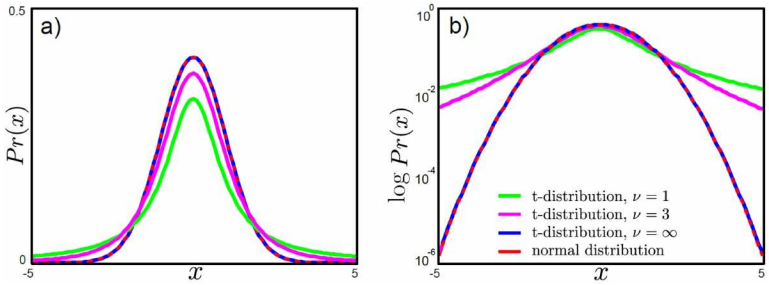
Student t-distribution

Univariate:

$$Pr(x) = \text{Stud}_x[\mu, \sigma^2, \nu] = \frac{\Gamma\left[\frac{\nu+1}{2}\right]}{\sqrt{\nu\pi\sigma^2}\Gamma\left[\frac{\nu}{2}\right]}\left(1+\frac{(x-\mu)^2}{\nu\sigma^2}\right)^{-\frac{\nu+1}{2}}$$

Multivariate:

$$Pr(\mathbf{x}) = \text{Stud}_{\mathbf{x}}[\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu] = \frac{\Gamma\left[\frac{\nu+D}{2}\right]}{(\nu\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}\Gamma\left[\frac{\nu}{2}\right]}\left(1+\frac{(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}{\nu}\right)^{-\frac{\nu+D}{2}}$$



Expressed as marginalization:

→ express t-distribution as gaussian mixture model

Define:

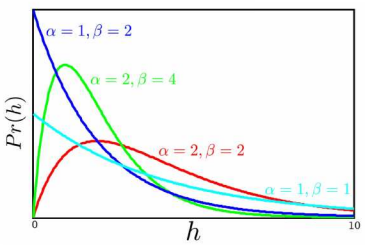
$$Pr(\mathbf{x}|h) = \text{Norm}_x[\boldsymbol{\mu}, \boldsymbol{\Sigma}/h]$$
$$Pr(h) = \text{Gam}_h[\nu/2, \nu/2]$$

Reformulation:

$$Pr(\mathbf{x}) = \int Pr(\mathbf{x}, h)dh = \int Pr(\mathbf{x}|h)Pr(h)dh$$
$$= \int \text{Norm}_x[\boldsymbol{\mu}, \boldsymbol{\Sigma}/h]\text{Gam}_h[\nu/2, \nu/2]dh$$
$$= \text{Stud}_{\mathbf{x}}[\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu].$$

Gamma distribution:

$$\text{Gam}_h[\alpha, \beta] = \frac{\beta^\alpha}{\Gamma[\alpha]}\exp[-\beta h]h^{\alpha-1}$$



Approximation:

$$Pr(\mathbf{w}|\mathbf{X},\sigma^2) \approx \max_{\mathbf{H}}\left[\text{Norm}_{\mathbf{w}}[\mathbf{0},\mathbf{X}^T\mathbf{H}^{-1}\mathbf{X}+\sigma^2\mathbf{I}]\prod_{d=1}^{D}\text{Gam}_{h_d}[\nu/2,\nu/2]\right]$$

Derivation:

$$\begin{aligned}
Pr(\mathbf{w}|\mathbf{X},\sigma^2) &= \int Pr(\mathbf{w},\boldsymbol{\phi}|\mathbf{X},\sigma^2)\,d\boldsymbol{\phi} \\
&= \int Pr(\mathbf{w}|\mathbf{X},\boldsymbol{\phi},\sigma^2)Pr(\boldsymbol{\phi})\,d\boldsymbol{\phi} \\
&= \int \text{Norm}_{\mathbf{w}}[\mathbf{X}^T\boldsymbol{\phi},\sigma^2\mathbf{I}]\int \text{Norm}_{\boldsymbol{\phi}}[0,\mathbf{H}^{-1}]\prod_{d=1}^{D}\text{Gam}_{h_d}[\nu/2,\nu/2]\,d\mathbf{H}d\boldsymbol{\phi} \\
&= \iint \text{Norm}_{\mathbf{w}}[\mathbf{X}^T\boldsymbol{\phi},\sigma^2\mathbf{I}]\text{Norm}_{\boldsymbol{\phi}}[0,\mathbf{H}^{-1}]\prod_{d=1}^{D}\text{Gam}_{h_d}[\nu/2,\nu/2]\,d\mathbf{H}d\boldsymbol{\phi} \\
&= \int \text{Norm}_{\mathbf{w}}[\mathbf{0},\mathbf{X}^T\mathbf{H}^{-1}\mathbf{X}+\sigma^2\mathbf{I}]\prod_{d=1}^{D}\text{Gam}_{h_d}[\nu/2,\nu/2]\,d\mathbf{H}.
\end{aligned}$$

Learning: Iteratively update H and θ

1. Initialize $H = I_d$ and $\theta > 1000$

Iterate:

2. Update posterior:

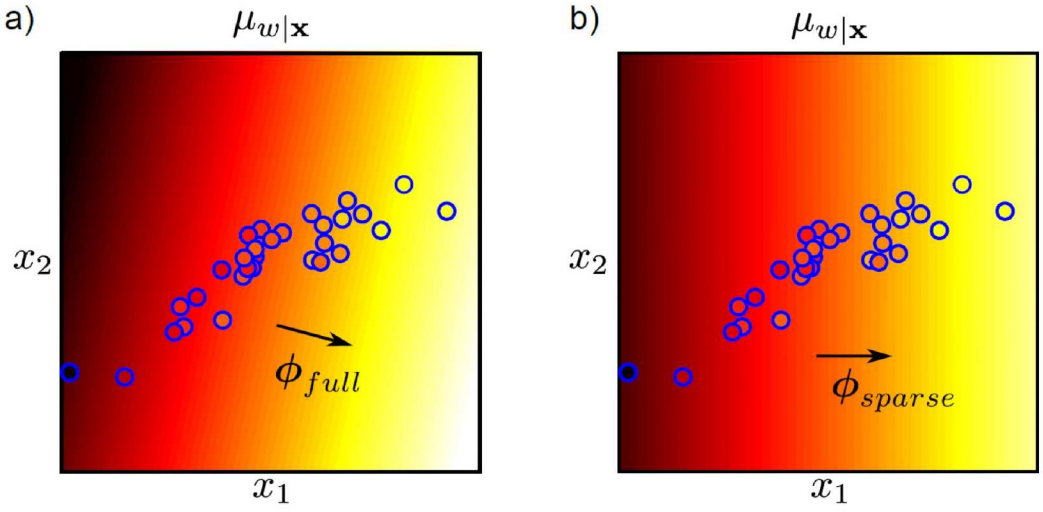
$$Pr(\boldsymbol{\phi}|\mathbf{X},\mathbf{w}) = \text{Norm}_{\boldsymbol{\phi}}\left[\frac{1}{\sigma^2}\mathbf{A}^{-1}\mathbf{X}\mathbf{w},\mathbf{A}^{-1}\right]$$

$$\begin{aligned}
\boldsymbol{\mu} &= \frac{1}{\sigma^2}\mathbf{A}^{-1}\mathbf{X}\mathbf{w} \\
\boldsymbol{\Sigma} &= \mathbf{A}^{-1},
\end{aligned} \qquad \mathbf{A} = \frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T + \mathbf{H}$$

3. Estimate H: $h_d^{new} = \dfrac{1 - h_d\Sigma_{dd} + \nu}{\mu_d^2 + \nu}$

4. Update posterior

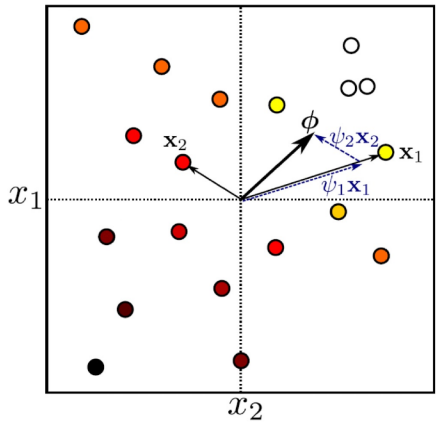
5. Estimate θ^2: $(\sigma^2)^{new} = \dfrac{1}{D - \sum_d(1 - h_d\Sigma_{dd})}(\mathbf{w} - \mathbf{X}\boldsymbol{\mu})^T(\mathbf{w} - \mathbf{X}\boldsymbol{\mu})$



Dual Linear Regression

Idea:

The gradient ϕ in the data space can be represented as a weighted sum of the data points. This reduces dimensionality when the the dim. of the feature space is larger than the number of data points:



$$\boldsymbol{\phi} = \mathbf{X}\boldsymbol{\psi}$$

Note: X only occurs in dot products → kernelized

Goal: Model $Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\theta}) = \text{Norm}_{\mathbf{w}}\left[\mathbf{X}^T\mathbf{X}\boldsymbol{\psi}, \sigma^2\mathbf{I}\right]$

Learning: Maximize log-likelihood

$$\hat{\boldsymbol{\psi}}, \hat{\sigma}^2 = \underset{\boldsymbol{\psi},\sigma^2}{\text{argmax}}\left[-\frac{I\log[2\pi]}{2} - \frac{I\log[\sigma]}{2} - \frac{(\mathbf{w}-\mathbf{X}^T\mathbf{X}\boldsymbol{\psi})^T(\mathbf{w}-\mathbf{X}^T\mathbf{X}\boldsymbol{\psi})}{2\sigma^2}\right]$$

Solution:

$$\begin{aligned}\hat{\boldsymbol{\psi}} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{w} \\ \hat{\sigma}^2 &= \frac{(\mathbf{w}-\mathbf{X}^T\mathbf{X}\boldsymbol{\psi})^T(\mathbf{w}-\mathbf{X}^T\mathbf{X}\boldsymbol{\psi})}{I}\end{aligned}$$

Bayesian case:

Goal: Model $Pr(\boldsymbol{\psi}|\mathbf{X}, \mathbf{w}, \sigma^2) = \frac{Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\psi}, \sigma^2)Pr(\boldsymbol{\psi})}{Pr(\mathbf{w}|\mathbf{X}, \sigma^2)}$

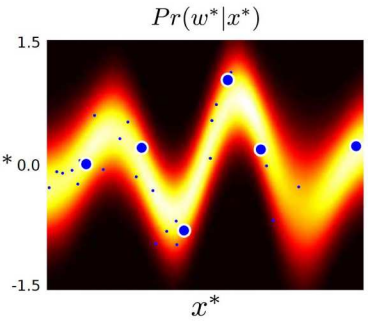
with:

$$\begin{aligned}Pr(\boldsymbol{\psi}) &= \text{Norm}_{\boldsymbol{\psi}}[\mathbf{0}, \sigma_p^2\mathbf{I}] \\ Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\theta}) &= \text{Norm}_{\mathbf{w}}\left[\mathbf{X}^T\mathbf{X}\boldsymbol{\psi}, \sigma^2\mathbf{I}\right]\end{aligned}$$

$$\Rightarrow \quad Pr(\boldsymbol{\psi}|\mathbf{X}, \mathbf{w}, \sigma^2) = \text{Norm}_{\boldsymbol{\psi}}\left[\frac{1}{\sigma^2}\mathbf{A}^{-1}\mathbf{X}^T\mathbf{X}\mathbf{w}, \mathbf{A}^{-1}\right]$$

$$\text{with: } \mathbf{A} = \frac{1}{\sigma^2}\mathbf{X}^T\mathbf{X}\mathbf{X}^T\mathbf{X} + \frac{1}{\sigma_p^2}\mathbf{I}$$

Relevance Vector Regression



Idea:

Combine dual and sparse regression to make the solution dependend on a sparse subset of the training examples.

Goal: Model

$$Pr(\mathbf{w}|\mathbf{X},\boldsymbol{\theta}) = \text{Norm}_{\mathbf{w}}\left[\mathbf{X}^T\mathbf{X}\boldsymbol{\psi}, \sigma^2\mathbf{I}\right]$$
$$Pr(\boldsymbol{\psi}) = \prod_{i=1}^{I} \text{Stud}_{\psi_i}[0,1,\nu]$$

Learning: same as sparse regression

$$Pr(\mathbf{w}|\mathbf{X},\sigma^2) \approx \max_{\mathbf{H}}\left[\text{Norm}_{\mathbf{w}}[\mathbf{0}, \mathbf{X}^T\mathbf{X}\mathbf{H}^{-1}\mathbf{X}^T\mathbf{X} + \sigma^2\mathbf{I}] \prod_{i=1}^{I} \text{Gam}_{h_i}[\nu/2, \nu/2]\right]$$

II Classification

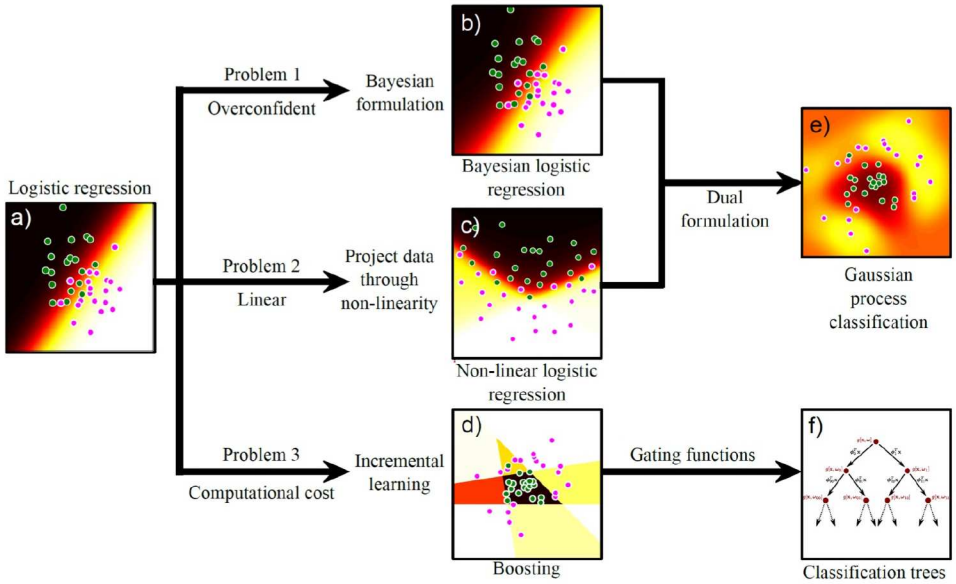
Task: (probabilistic view)

→ similar to regression

Model: $Pr(\mathbf{w}|\mathbf{x})$

→ w has changed to discrete class labels

Classification Models:



Logistic Regression

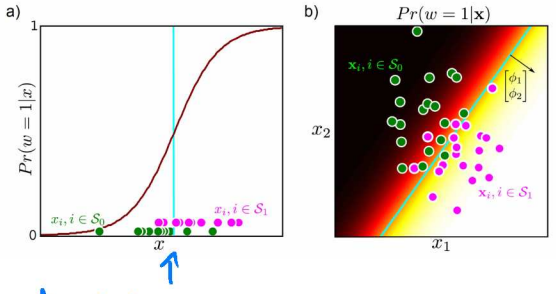
Idea: Regress to a linear separating function used for classifying instances into two classes. For this we model the prob. distribution as a Bernoulli distribution an estimate parameter λ.

Goal:

Model

$$Pr(w|\phi, \mathbf{x}) = \text{Bern}_w\left[\frac{1}{1+\exp[-\phi^T\mathbf{x}]}\right]$$

$\frac{1}{1+\exp[-\phi^T\mathbf{x}]}$ = sigmoid(a)



↑ decision boundary = 0.5

Learning: Maximize log-likelihood

→ there is no closed-form solution → iterative optimization

Log-likelihood:

$$L = \sum_{i=1}^{I} w_i \log\left[\frac{1}{1+\exp[-\phi^T\mathbf{x}_i]}\right] + \sum_{i=1}^{I}(1-w_i)\log\left[\frac{\exp[-\phi^T\mathbf{x}_i]}{1+\exp[-\phi^T\mathbf{x}_i]}\right]$$

Derivatives:

$$\frac{\partial L}{\partial \phi} = -\sum_{i=1}^{I}\left(\frac{1}{1+\exp[-\phi^T\mathbf{x}_i]} - w_i\right)\mathbf{x}_i = -\sum_{i=1}^{I}\left(\text{sig}[a_i] - w_i\right)\mathbf{x}_i$$

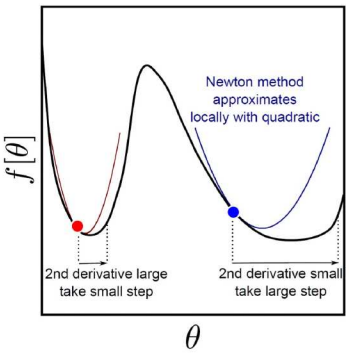
$$\frac{\partial^2 L}{\partial \phi^2} = -\sum_{i=1}^{I}\text{sig}[a_i](1-\text{sig}[a_i])\mathbf{x}_i\mathbf{x}_i^T$$

Newton's Method: (used for optimization)

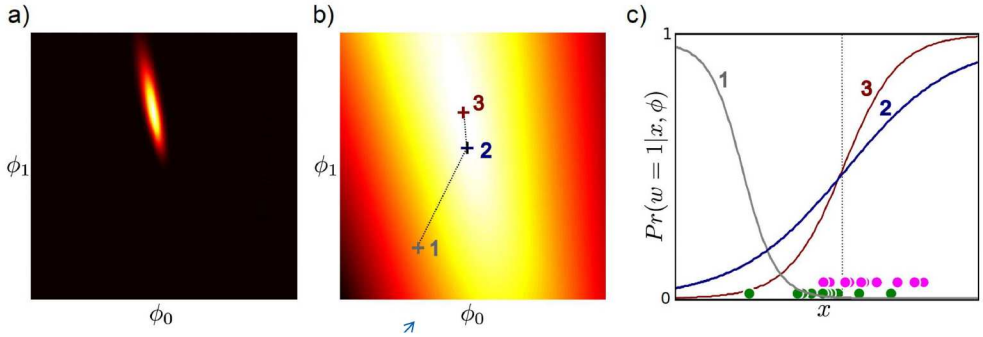
Goal: Find $\hat{\theta} = \underset{\theta}{\text{argmin}}\,[f[\theta]]$

Update:

$$\theta^{[t+1]} = \theta^{[t]} - \lambda\left(\frac{\partial^2 f}{\partial \theta^2}\right)^{-1}\frac{\partial f}{\partial \theta}$$



→ If hessian is positive definite, then the optimization domain is convex



Bayesian Logistic Regression

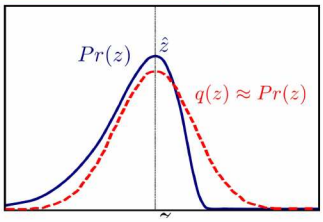
Idea: Apply bayesian regression to logistic regression The problem is, since the prior is a normal distribution and the likelihood is a Bernoulli distribution, there exists no closed form for the posterior. Thus, apply Laplace approximation to get a gaussian.

Goal: Model $$Pr(\phi|\mathbf{X},\mathbf{w}) = \frac{Pr(\mathbf{w}|\mathbf{X},\phi)Pr(\phi)}{Pr(\mathbf{w}|\mathbf{X})}$$

Prior: $$Pr(\phi) = \text{Norm}_{\phi}[\mathbf{0}, \sigma_p^2\mathbf{I}]$$

Likelihood: $$Pr(\mathbf{w}|\mathbf{X},\phi) = \prod_{i=1}^{I}\left(\frac{1}{1+\exp[-\phi^T\mathbf{x}_i]}\right)^{w_i}\left(\frac{\exp[-\phi^T\mathbf{x}_i]}{1+\exp[-\phi^T\mathbf{x}_i]}\right)^{1-w_i}$$

Laplace Approximation:

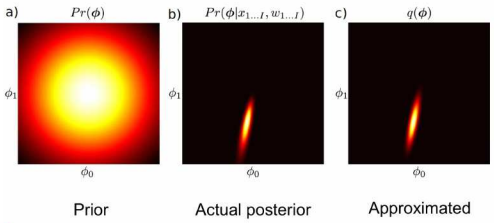


Approximation:

$$Pr(\phi|\mathbf{X},\mathbf{w}) \approx q(\phi) = \text{Norm}_{\phi}[\boldsymbol{\mu}, \boldsymbol{\Sigma}]$$

with: $$\boldsymbol{\mu} = \hat{\phi}$$

$$\boldsymbol{\Sigma} = -\left(\frac{\partial^2 L}{\partial \phi^2}\right)^{-1}\Big|_{\phi=\hat{\phi}}$$



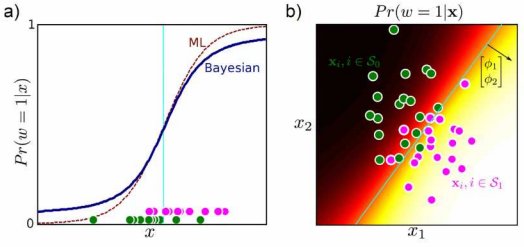
Learning: Find MAP solution

Optimize: $$L = \sum_{i=1}^{I}\log[Pr(w_i|\mathbf{x}_i,\phi)] + \log[Pr(\phi)]$$

using Newton's method:

$$\frac{\partial L}{\partial \phi} = -\sum_{i=1}^{I}(\text{sig}[a_i] - w_i)\mathbf{x}_i - \frac{\phi}{\sigma_p^2}$$

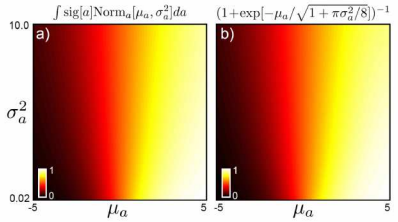
$$\frac{\partial^2 L}{\partial \phi^2} = -\sum_{i=1}^{I}\text{sig}[a_i](1-\text{sig}[a_i])\mathbf{x}_i\mathbf{x}_i^T - \frac{1}{\sigma_p^2}$$



Inference:

Compute: $$Pr(w^*|\mathbf{x}^*,\mathbf{X},\mathbf{w}) = \int Pr(w^*|\mathbf{x}^*,\phi)Pr(\phi|\mathbf{X},\mathbf{w})d\phi$$

$$\approx \frac{1}{1+\exp[-\mu_a/\sqrt{1+\pi\sigma_a^2/8}]}$$



→ intractable to compute integral

Approximation:

1. Laplace: $\approx \int Pr(w^*|\mathbf{x}^*,\phi)q(\phi)d\phi.$
2. Re-express: $$Pr(w^*|\mathbf{x}^*,\mathbf{X},\mathbf{w}) \approx \int Pr(w^*|a)Pr(a)da$$
3. Change variables: $$Pr(a) = Pr(\phi^T\mathbf{x}^*) = \text{Norm}_a[\boldsymbol{\mu}^T\mathbf{x}^*, \mathbf{x}^{*T}\boldsymbol{\Sigma}\mathbf{x}^*]$$
$$= \text{Norm}_a[\mu_a, \sigma_a^2],$$

Non-linear Logistic Regression

Idea: Use a non-linear embedding function before applying logistic regression

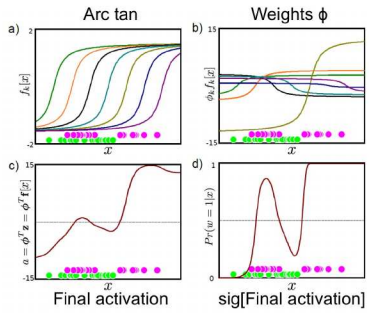
Goal: Model

$$Pr(w=1|\mathbf{x},\boldsymbol{\phi}) = \text{Bern}_w\left[\text{sig}[\boldsymbol{\phi}^T\mathbf{z}]\right] = \text{Bern}_w\left[\text{sig}[\boldsymbol{\phi}^T\mathbf{f}[\mathbf{x}]]\right]$$

Transformations:
→ the parameters of the transform are also learned

Arc tan: $z_k = \arctan[\boldsymbol{\alpha}_k^T\mathbf{x}]$

RBF: $z_k = \exp\left[-\frac{1}{\lambda_0}(\mathbf{x}-\boldsymbol{\alpha}_k)^T(\mathbf{x}-\boldsymbol{\alpha}_k)\right]$



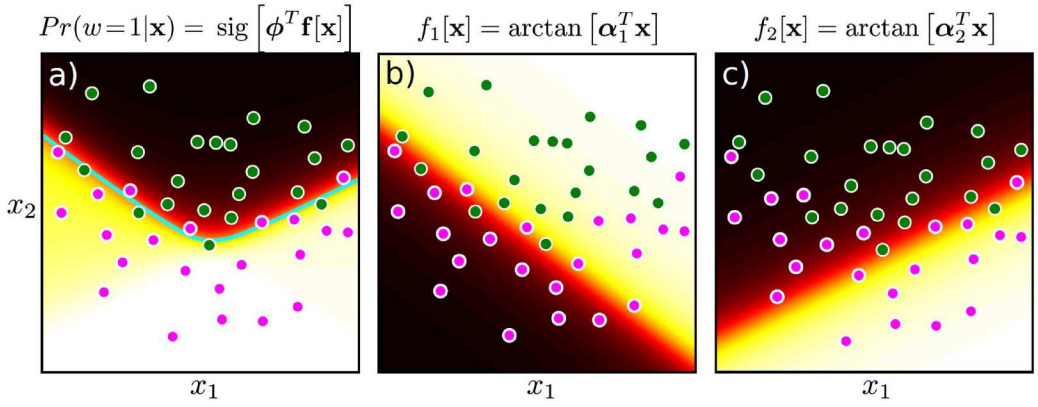
Learning: Optimize using Newton's method

Parameters: $\boldsymbol{\theta} = [\boldsymbol{\phi}^T, \boldsymbol{\alpha}_1^T, \boldsymbol{\alpha}_2^T, \ldots, \boldsymbol{\alpha}_K^T]^T$

Derivatives:

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = \sum_{i=1}^{I}(w_i - \text{sig}[a_i])\frac{\partial a_i}{\partial \boldsymbol{\theta}}$$

$$\frac{\partial^2 L}{\partial \boldsymbol{\theta}^2} = \sum_{i=1}^{I}\text{sig}[a_i](\text{sig}[a_i]-1)\frac{\partial a_i}{\partial \boldsymbol{\theta}}\frac{\partial a_i}{\partial \boldsymbol{\theta}}^T + (w_i - \text{sig}[a_i])\frac{\partial^2 a_i}{\partial \boldsymbol{\theta}^2}$$



Dual Logistic Regression:

Idea: Exact same as dual regression: $\boldsymbol{\phi} = \mathbf{X}\boldsymbol{\psi}$

Learning: Optimize using Newton's method

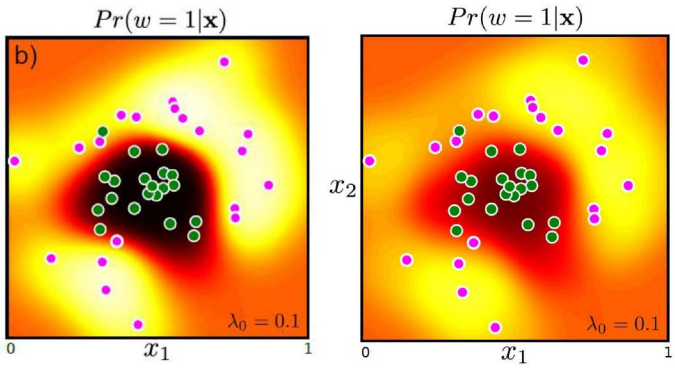
Likelihood:

$$Pr(\mathbf{w}|\mathbf{X},\boldsymbol{\psi}) = \prod_{i=1}^{I} \text{Bern}_{w_i}\left[\text{sig}[a_i]\right] = \prod_{i=1}^{I} \text{Bern}_{w_i}\left[\text{sig}[\boldsymbol{\psi}^T\mathbf{X}^T\mathbf{x}_i]\right]$$

Derivatives:

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{\psi}} &= -\sum_{i=1}^{I}\left(\text{sig}[a_i] - w_i\right)\mathbf{X}^T\mathbf{x}_i \\ \frac{\partial^2 L}{\partial \boldsymbol{\psi}^2} &= -\sum_{i=1}^{I}\text{sig}[a_i]\left(1-\text{sig}[a_i]\right)\mathbf{X}^T\mathbf{x}_i\mathbf{x}_i^T\mathbf{X}\end{aligned}$$

Bayesian: Gaussian Process Classification



Relevance Vector Classification

Idea: Use student-t distribution as the prior of the dual logistic regression to enforce a sparse solution:

$$Pr(\boldsymbol{\psi}) = \prod_{i=1}^{I} \text{Stud}_{\psi_i}[0, 1, \nu]$$

Goal: Model

$$\begin{aligned} &Pr(\mathbf{w}|\mathbf{X}) \\ &= \int Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\psi}) Pr(\boldsymbol{\psi})\, d\boldsymbol{\psi} \\ &= \int\int \prod_{i=1}^{I} \text{Bern}_{w_i}\Big[\text{sig}[\boldsymbol{\psi}^T \mathbf{K}[\mathbf{X}, \mathbf{x}_i]]\Big] \text{Norm}_{\boldsymbol{\psi}}[0, \mathbf{H}^{-1}]\, \text{Gam}_{h_i}[\nu/2, \nu/2]\, d\mathbf{H} d\boldsymbol{\psi} \end{aligned}$$

Prior:

$$\begin{aligned} Pr(\boldsymbol{\psi}) &= \prod_{i=1}^{I} \int \text{Norm}_{\psi_i}\left[0, \frac{1}{h_i}\right] \text{Gam}_{h_i}\left[\frac{\nu}{2}, \frac{\nu}{2}\right] dh_i \\ &= \int \text{Norm}_{\boldsymbol{\psi}}[0, \mathbf{H}^{-1}] \prod_{i=1}^{I} \text{Gam}_{h_i}[\nu/2, \nu/2]\, d\mathbf{H} \end{aligned}$$

Learning:

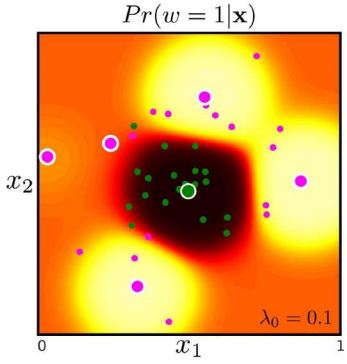
Alternatively update H and μ, Σ of the Laplace approximation:

Laplace approximation:

$$Pr(\mathbf{w}|\mathbf{X}) \approx \int \prod_{i=1}^{I} (2\pi)^{I/2} |\boldsymbol{\Sigma}|^{0.5} \text{Bern}_{w_i}[\text{sig}[\boldsymbol{\mu}^T \mathbf{K}[\mathbf{X}, \mathbf{x}_i]]] \text{Norm}_{\boldsymbol{\mu}}[0, \mathbf{H}^{-1}] \text{Gam}_{h_i}\left[\frac{\nu}{2}, \frac{\nu}{2}\right] d\mathbf{H}$$

Second approximation:

$$Pr(\mathbf{w}|\mathbf{X}) \approx \max_{\mathbf{H}} \left[\prod_{i=1}^{I} (2\pi)^{I/2} |\boldsymbol{\Sigma}|^{0.5} \text{Bern}_{w_i}[\text{sig}[\boldsymbol{\mu}^T \mathbf{K}[\mathbf{X}, \mathbf{x}_i]]] \text{Norm}_{\boldsymbol{\mu}}[0, \mathbf{H}^{-1}] \text{Gam}_{h_i}\left[\frac{\nu}{2}, \frac{\nu}{2}\right]\right]$$



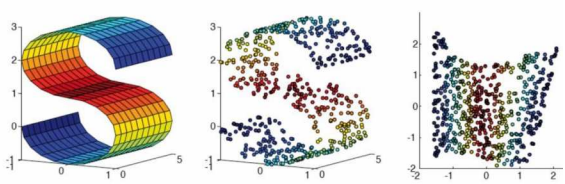
III Gaussian Processes and Latent Variable Models

Motivation: Dimensionality Reduction

1. Reduce dimensionality for more invariant representation
2. Make sampling more tractable
3. Representation where close points have similar semantic meaning
4. Data with structure:
   Usually have fewer distortions (e.g. rotation, translation) than dimensions. Thus, we can expect it to live on a lower dimensional manifold

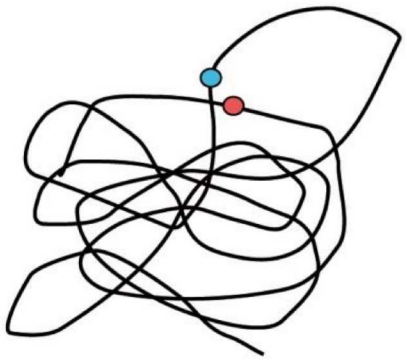
Spectral Methods (CU-1)

Mapping based on preserving the distance between local neighbors

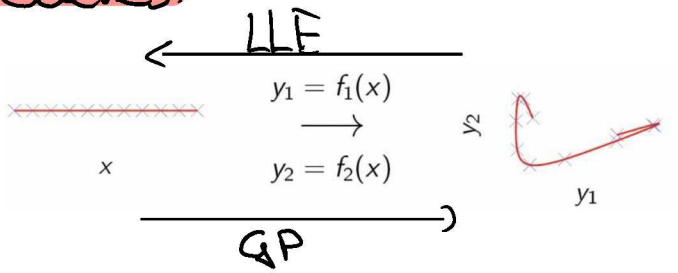


Problem:

Local distance might not be an appropriate metric for similarity
→ points are close in space but far away in the domain of interest



Approaches:



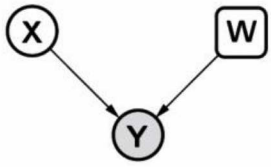
→ we want to learn how to generate data from the latent space

Linear Latent Variable Models

Idea: Represent Y through a lower dimensional set of latent variables X:

$$\mathbf{y}_{i,:} = \mathbf{W}\mathbf{x}_{i,:} + \boldsymbol{\epsilon}_{i,:} \quad \text{with} \quad \boldsymbol{\epsilon}_{i,:} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$$

Probabilistic PCA: $p(\mathbf{Y}|\mathbf{W}) = \prod_{i=1}^{n} \mathcal{N}\left(\mathbf{y}_{i,:}|\mathbf{0}, \mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I}\right)$



→ defines linear-gaussian relationship between x and y

Derivation:

Define gaussian prior over latent space and integrate out the latent variables

$$p(\mathbf{Y}|\mathbf{X},\mathbf{W}) = \prod_{i=1}^{n} \mathcal{N}(\mathbf{y}_{i,:}|\mathbf{W}\mathbf{x}_{i,:}, \sigma^2\mathbf{I}) \longrightarrow p(\mathbf{X}) = \prod_{i=1}^{n} \mathcal{N}(\mathbf{x}_{i,:}|\mathbf{0}, \mathbf{I}) \longrightarrow p(\mathbf{Y}|\mathbf{W}) = \prod_{i=1}^{n} \mathcal{N}\left(\mathbf{y}_{i,:}|\mathbf{0}, \mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I}\right)$$

Maximum Likelihood Solution (Solve for mapping)

$$p(\mathbf{Y}|\mathbf{W}) = \prod_{i=1}^{n} \mathcal{N}(\mathbf{y}_{i,:}|\mathbf{0}, \mathbf{C}), \quad \mathbf{C} = \mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I}$$

$$\log p(\mathbf{Y}|\mathbf{W}) = -\frac{n}{2}\log|\mathbf{C}| - \frac{1}{2}\mathrm{tr}\left(\mathbf{C}^{-1}\mathbf{Y}^\top\mathbf{Y}\right) + \text{const.}$$

If $\mathbf{U}_q$ are first q principal eigenvectors of $n^{-1}\mathbf{Y}^\top\mathbf{Y}$ and the corresponding eigenvalues are $\boldsymbol{\Lambda}_q$,

$$\mathbf{W} = \mathbf{U}_q\mathbf{L}\mathbf{R}^\top, \quad \mathbf{L} = (\boldsymbol{\Lambda}_q - \sigma^2\mathbf{I})^{\frac{1}{2}}$$

where $\mathbf{R}$ is an arbitrary rotation matrix.

→ Number of dimension q is pre-defined
→ Solution is an eigenvalue problem
→ R is an identity matrix

Dual Probabilistic PCA: $p(\mathbf{Y}|\mathbf{X}) = \prod_{j=1}^{p} \mathcal{N}\left(\mathbf{y}_{:,j}|\mathbf{0}, \mathbf{X}\mathbf{X}^\top + \sigma^2\mathbf{I}\right)$

Derivation: similar to previous formulation

$$p(\mathbf{Y}|\mathbf{X},\mathbf{W}) = \prod_{i=1}^{n} \mathcal{N}(\mathbf{y}_{i,:}|\mathbf{W}\mathbf{x}_{i,:}, \sigma^2\mathbf{I}) \longrightarrow p(\mathbf{W}) = \prod_{i=1}^{p} \mathcal{N}(\mathbf{w}_{i,:}|\mathbf{0}, \mathbf{I}) \longrightarrow p(\mathbf{Y}|\mathbf{X}) = \prod_{j=1}^{p} \mathcal{N}\left(\mathbf{y}_{:,j}|\mathbf{0}, \mathbf{X}\mathbf{X}^\top + \sigma^2\mathbf{I}\right)$$

Maximum Likelihood Solution (Solve for latent positions)

$$p(\mathbf{Y}|\mathbf{X}) = \prod_{j=1}^{p} \mathcal{N}(\mathbf{y}_{:,j}|\mathbf{0}, \mathbf{K}), \quad \mathbf{K} = \mathbf{X}\mathbf{X}^\top + \sigma^2\mathbf{I}$$

$$\log p(\mathbf{Y}|\mathbf{X}) = -\frac{p}{2}\log|\mathbf{K}| - \frac{1}{2}\mathrm{tr}\left(\mathbf{K}^{-1}\mathbf{Y}\mathbf{Y}^\top\right) + \text{const.}$$

If $\mathbf{U}'_q$ are first q principal eigenvectors of $p^{-1}\mathbf{Y}\mathbf{Y}^\top$ and the corresponding eigenvalues are $\boldsymbol{\Lambda}_q$,

← only change

$$\mathbf{X} = \mathbf{U}'_q\mathbf{L}\mathbf{R}^\top, \quad \mathbf{L} = (\boldsymbol{\Lambda}_q - \sigma^2\mathbf{I})^{\frac{1}{2}}$$

where $\mathbf{R}$ is an arbitrary rotation matrix.

Equivalence of Models:

Solution probabilistic PCA: $\mathbf{Y}^\top \mathbf{Y}\mathbf{U}_q = \mathbf{U}_q\mathbf{\Lambda}_q \qquad \mathbf{W} = \mathbf{U}_q\mathbf{L}\mathbf{R}^\top$

Solution dual probabilistic PCA: $\mathbf{Y}\mathbf{Y}^\top \mathbf{U}'_q = \mathbf{U}'_q\mathbf{\Lambda}_q \qquad \mathbf{X} = \mathbf{U}'_q\mathbf{L}\mathbf{R}^\top$

Equivalence comes from: $\mathbf{U}_q = \mathbf{Y}^\top \mathbf{U}'_q\mathbf{\Lambda}_q^{-\frac{1}{2}}$

→ insert into solutions to see equivalence

Non-linear Latent Variable Models

GP-LVM: $p(\mathbf{Y}|\mathbf{X}) = \prod_{j=1}^{p} \mathcal{N}(\mathbf{y}_{:,j}|\mathbf{0}, \mathbf{K})$ K: non-linear kernel

→ non-linearity prevents solution via eigenvalue problem
→ optimize using gradients w.r.t. the parameters

Derivation:

Use dual formulation and replace the dot product with a kernel function

$$p(\mathbf{Y}|\mathbf{X}) = \prod_{j=1}^{p} \mathcal{N}\left(\mathbf{y}_{:,j}|\mathbf{0}, \mathbf{X}\mathbf{X}^\top + \sigma^2\mathbf{I}\right) \longrightarrow p(\mathbf{Y}|\mathbf{X}) = \prod_{j=1}^{p} \mathcal{N}(\mathbf{y}_{:,j}|\mathbf{0}, \mathbf{K}) \longrightarrow p(\mathbf{Y}|\mathbf{X}) = \prod_{j=1}^{p} \mathcal{N}(\mathbf{y}_{:,j}|\mathbf{0}, \mathbf{K})$$

$$\mathbf{K} = \mathbf{X}\mathbf{X}^\top + \sigma^2\mathbf{I}$$

Exponentiated Quadratic Covariance: $k(\mathbf{x}_{i,:}, \mathbf{x}_{j,:}) = \alpha \exp\left(-\frac{\|\mathbf{x}_{i,:} - \mathbf{x}_{j,:}\|_2^2}{2\ell^2}\right)$

Applications / Properties:

- Dimensionality reduction
- works well for data consisting of a small amount of points
- performance drops for data with a lot of variation
- powerful uncertainty handling

Character Animation

GPLVM learns how to embed a pose into a lower (2D) dimensional feature space

Problem: For synthesizing new poses we require the backward mapping which is unknown

Solution: Iterative optimization using a loss function

$$\arg\min_{\mathbf{x},\mathbf{q}} L_{IK}(\mathbf{x},\mathbf{y}(\mathbf{q})) \quad \text{s.t. } C(\mathbf{q}) = 0$$

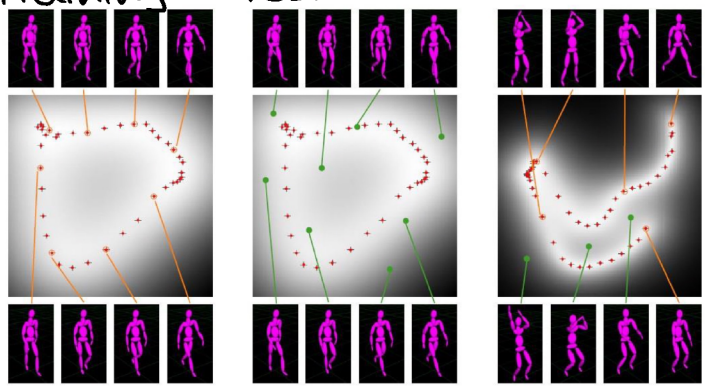
with: $$L_{IK}(\mathbf{x},\mathbf{y}) = \frac{||\mathbf{W}(\mathbf{y}-\mathbf{f}(\mathbf{x}))||^2}{2\sigma^2(\mathbf{x})} + \frac{D}{2}\ln\sigma^2(\mathbf{x}) + \frac{1}{2}||\mathbf{x}||^2$$

- Prefer latent variable which predicts y
- Prefer latent variable with high confidence
- Prediction of GPLVM becomes unreliable for large x

C: pose constraints, e.g. certain joint movements
W: additional scaling matrix
f, σ: Come from GPLVM Training

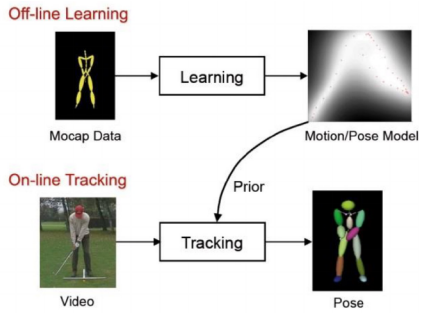
→ x and q are optimized

Test:



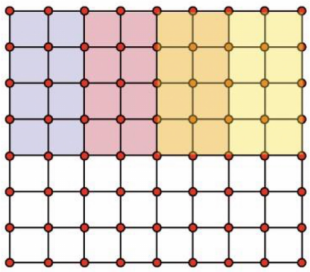
Generative Tracking

Learn off-line priors using a GPLVM and employ them for tracking



Non-rigid Shape Deformation

Idea: Model local deformations in small patches. Learn a single model and query it over the surface as a product of experts.



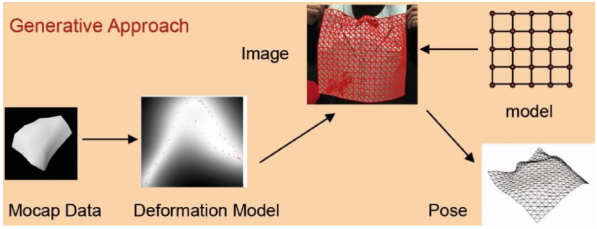
Tracking: $$p(\phi_t|\mathbf{I}_t,\mathbf{X},\mathbf{Y}) \propto p(\mathbf{I}_t|\phi_t)p(\mathbf{y}_t|\mathbf{x}_t,\mathbf{X},\mathbf{Y})p(\mathbf{x}_t)$$

(prior: $p(\mathbf{x}_t)$; GPLVM: $p(\mathbf{y}_t|\mathbf{x}_t,\mathbf{X},\mathbf{Y})$)

→ minimize posterior

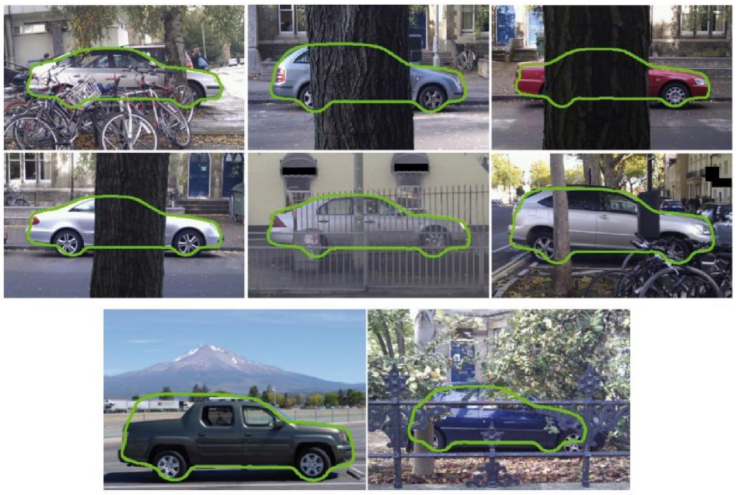
Likelihood: $$p(\mathbf{I}_t|\phi_t) = p(\mathbf{T}_t|\phi_t)p(\mathbf{E}_t|\phi_t)$$

$\mathbf{T}_t$: texture, $\mathbf{E}_t$: edge



GPLVM on Contours

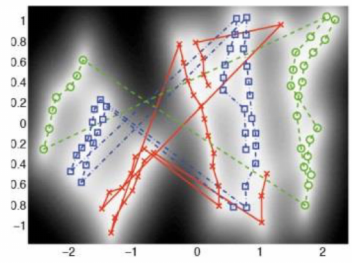
Define prior on the contours using GPLVM
→ strong against occlusion
→ unseen shapes are mapped



GPLVM with Dynamics

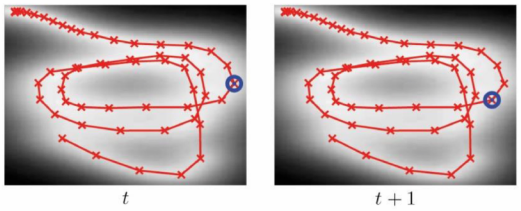
Problem: Optimization relies on non-convex function:

$$\mathcal{L} = \frac{p}{2}\ln|\mathbf{K}| + \frac{p}{2}tr(\mathbf{K}^{-1}\mathbf{Y}\mathbf{Y}^T)$$



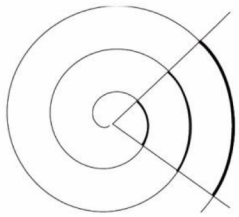
=> poor representation if initialized far from optimum

Idea: Instead represent relative positions by using an auto-regressive gaussian process mapping



Problem: Inner Grove distortion

Auto-regressive model forces spiral structure
→ constant velocity in org. space results in increasing velocity in embedding space



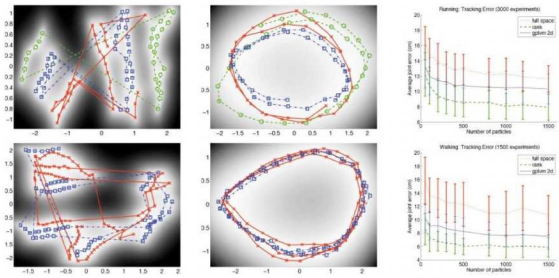
Continuous Dimensionality Reduction

Learn the latent space and its dimensionality

Idea: the dimensionality can be measured by the rank of XX^T. Since this is not differentiable, we choose a prior that encourages sparse eigenvalues

Prior: $\alpha \sum_{i=1}^{p} \phi(s_i)$ with: $\phi(s_i, r) = |s_i|^r$

s_i: eigenvalues



(GPLVM init PCA) (our method) (Error comparison)

Optimization

Initialize: $\mathbf{X}_{init} = \mathbf{Y}$

Objective:

$$\min_{\mathbf{y},\theta} p(\mathbf{Y}|\mathbf{X}, \theta) \quad \text{with: } E(\mathbf{X}) = \sum_i s_i^2.$$

$$\text{s. t.} \forall i \; s_i \geq 0, \quad E(\mathbf{Y}) - E(\mathbf{X}) = 0$$

sum of eigenvalues stay the same. Otherwise all points would be mapped to a single point

Choice of dimensionality:

$$Q = \text{argmax}_i \frac{s_i}{s_{i+1} + \epsilon}$$

where $\epsilon \ll 1$, and $s_1 \geq s_2 \geq \cdots \geq s_D$

→ biggest jump in the eigenvalues

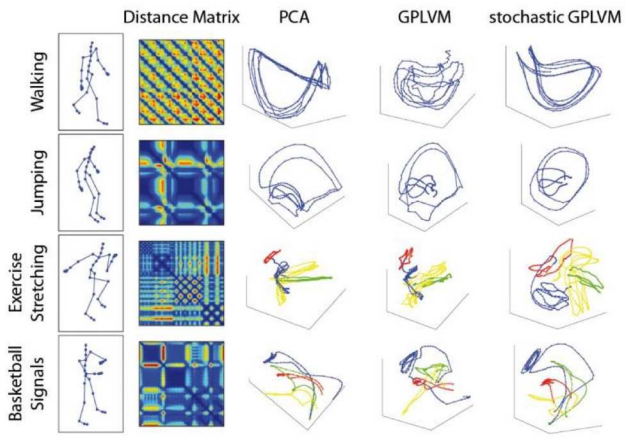
Stochastic Gradient Descent

→ more robust against local minima/maxima than maximum likelihood

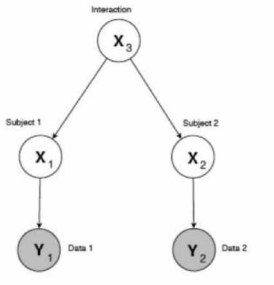
→ compute gradients in the local neighborhood

Algorithm 1: Stochastic GPLVM

Randomly initialize $\mathbf{X}$
Set θ with an initial guess
for $t = 1{:}T$
  randomly select $\mathbf{x}_r$
  find R neighbors around $\mathbf{x}_r$: $\mathbf{X}_R = \mathbf{X} \in \mathcal{R}$
  Compute $\frac{\partial L}{\partial \mathbf{X}_R}$ and $\frac{\partial L}{\partial \theta_R}$
  Update $\mathbf{X}$ and θ:
    $\Delta \mathbf{X}_t = \mu_X \cdot \Delta \mathbf{X}_{t-1} + \eta_X \cdot \frac{\partial L}{\partial \mathbf{X}_R}$
    $\mathbf{X}_t \leftarrow \mathbf{X}_{t-1} + \Delta \mathbf{X}_t$
    $\Delta \theta_t = \mu_\theta \cdot \Delta \theta_{t-1} + \eta_\theta \cdot \frac{\partial L}{\partial \theta_R}$
    $\theta_t \leftarrow \theta_{t-1} + \Delta \theta_t$

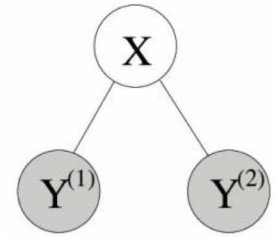


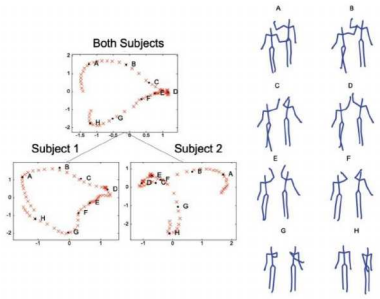
Hierarchical GP-LVM



By stacking GPs we can express more complex interactions and correlations between latent representations
→ e.g. same pose seen in different cameras, correlated subjects

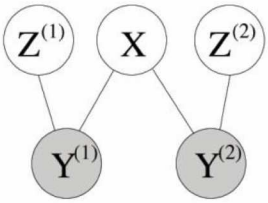
Multiple outputs:





→ expresses highly correlated views
→ does not capture different informations of views

Shared-Private Factorization:



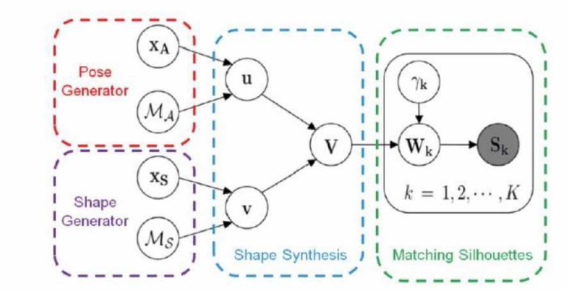
Model shared and private information separately

Optimization: $\mathcal{L} = L_{data} + L_{ortho} + L_{dim} + L_{energy}$

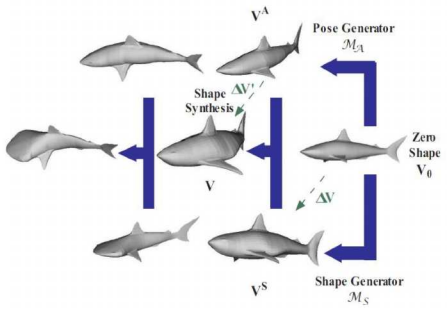
→ continuous dimensionality reduction

Orthogonality prior: $L_{ortho} = \alpha \sum_i \left(||\mathbf{X}^T \cdot \mathbf{Z}^{(i)}||_F^2 + \sum_{j>i} ||(\mathbf{Z}^{(i)})^T \cdot \mathbf{Z}^{(j)}||_F^2 \right)$

→ enforces that elements of $Z^{(i)}$ and X are independent
→ Different latent spaces do not capture redundant information



$$\mathbf{V} = \mathbf{V}^{\mathbf{A}} + \Delta\mathbf{V}' + \mathbf{n}_{\mathbf{V}}$$



Multilinear Models

Style-Content Separation:

$$\mathbf{y} = \sum_{ij} w_{ij} a_i b_j + \epsilon$$

emotion (↓ on a_i), shape (↑ on b_j)

Multi-linear Analysis:

$$\mathbf{y} = \sum_{ijk\cdots} w_{ijk\ldots} a_i b_j c_k \cdots + \epsilon$$

Non-linear Basis Function:

$$\mathbf{y} = \sum_{ij} w_{ij} a_i \phi_j(b) + \epsilon$$

Multi (non)-linear Models

GP-LVM:

$$\mathbf{y} = \sum_j w_j \phi_j(\mathbf{x}) + \epsilon = \mathbf{w}^T \Phi(\mathbf{x}) + \epsilon$$

with

$$E[\mathbf{y}, \mathbf{y}'] = \Phi(\mathbf{x})^T \Phi(\mathbf{x}') + \beta^{-1}\delta = k(\mathbf{x}, \mathbf{x}') + \beta^{-1}\delta$$

Multifactor Gaussian process:

$$\mathbf{y} = \sum_{i,j,k,\cdots} w_{ijk\ldots} \phi_i^{(1)} \phi_j^{(1)} \phi_k^{(1)} \cdots + \epsilon$$

with

$$E[\mathbf{y}, \mathbf{y}'] = \prod_i {\Phi^{(i)}}^T \Phi^{(i)} + \beta^{-1}\delta = \prod_i k_i(\mathbf{x}^{(i)}, \mathbf{x}^{(i)\prime}) + \beta^{-1}\delta$$

Continuous Character Control

Enforce inter-connectivity to ensure smooth transitions

Prior:

$$\ln p(\mathbf{X}) = w_c \sum_{i,j} \ln K_{ij}^d$$

with the **graph diffusion kernel** $\mathbf{K}^d$ obtain from

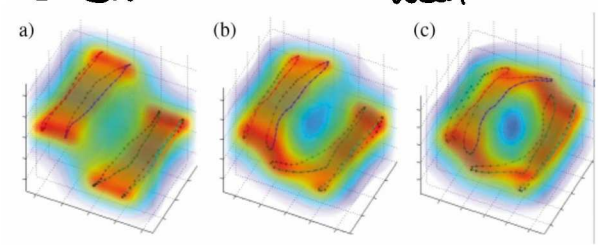
$$K_{ij}^d = \exp(\beta \mathbf{H}) \quad \text{with} \quad \mathbf{H} = -\mathbf{T}^{-1/2}\mathbf{L}\mathbf{T}^{-1/2}$$

the graph Laplacian, and $\mathbf{T}$ is a diagonal matrix with $T_{ii} = \sum_j w(\mathbf{x}_i, \mathbf{x}_j)$,

$$L_{ij} = \begin{cases} \sum_k w(\mathbf{x}_i, \mathbf{x}_k) & \text{if } i = j \\ -w(\mathbf{x}_i, \mathbf{x}_j) & \text{otherwise.} \end{cases}$$

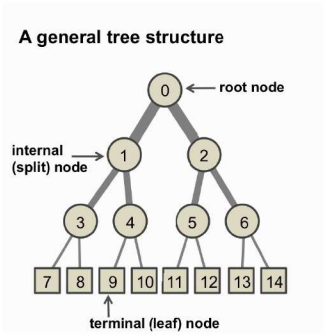
and $w(\mathbf{x}_i, \mathbf{x}_j) = ||\mathbf{x}_i - \mathbf{x}_j||^{-p}$ measures similarity.

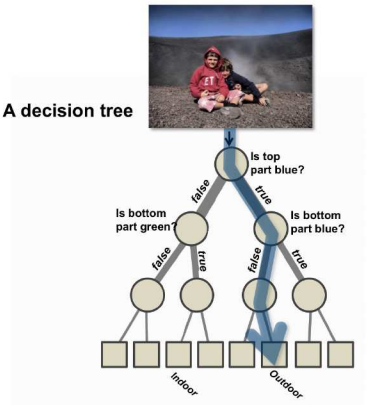
Problem: Wanted:

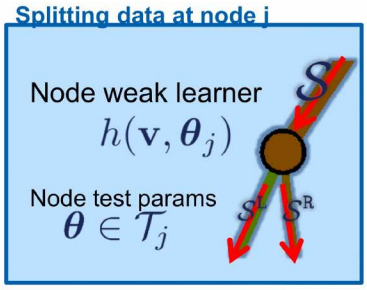


IV Random Forests

Tree Structure





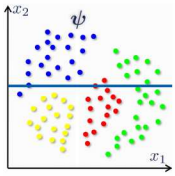


Internal node: Split the data on basis of a pre-defined splitting function. Each node can be seen as a weak learner.

Splitting functions:

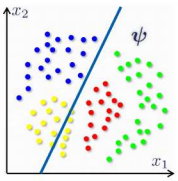
Axis-aligned:

$$h(v, \theta) = \phi(v) \cdot \psi > \tau$$
$$\phi(v) = (x_1, x_2)$$
$$\psi = (0, 1)$$



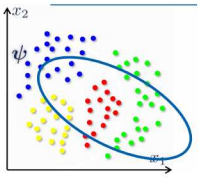
Oriented line:

$$h(v, \theta) = \phi(v) \cdot \psi > \tau$$
$$\phi(v) = (x_1, x_2)$$
$$\psi = (a, b) \in \mathbb{R}^2$$



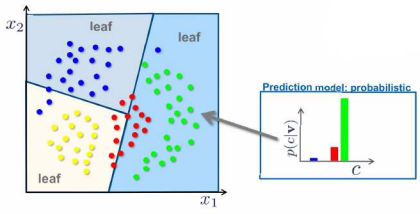
Conic section:

$$h(v, \theta) = \phi(v)^T \psi \phi(v) > \tau$$
$$\phi(v) = (x_1, x_2)$$
$$\psi = (a, b) \in \mathbb{R}^{3\times3}$$



Predictor Model (Leaf nodes)

Classification: Represent an area in the input space that is labeled according to the training data ending in the given leaf.

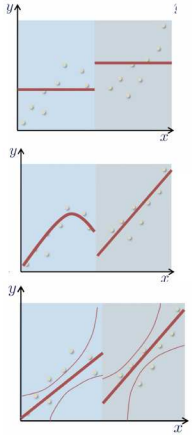


Regression:

Constant: $y = const$

Polynomial: $y = \sum_{i=0}^{n} w_i x^i$

Probabilistic-linear: $y = l_1 x + l_2$



Density: $\mathcal{N}(\mathbf{v}; \boldsymbol{\mu}_{l(\mathbf{v})}, \Lambda_{l(\mathbf{v})})$

→ each leaf represents a density through a gaussian

Randomness Model

1. Bagging

Train different trees on a subset of the training data and combine their results.

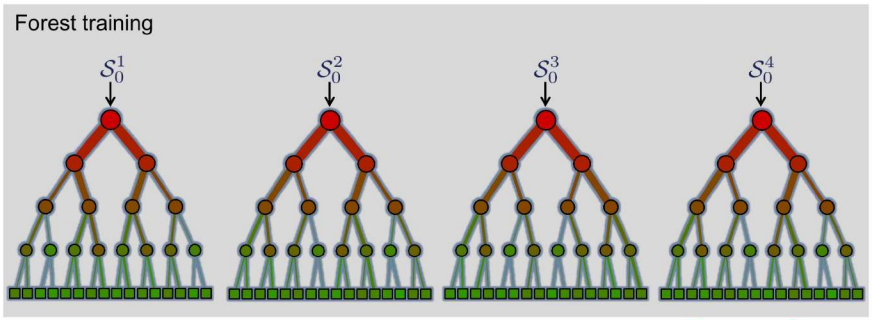
→ multiple smaller trees perform better than one large tree

→ faster training

→ better approximation of uncertainty

$\mathcal{S}_0$ The full training set

$\mathcal{S}_0^t \subset \mathcal{S}_0$ The randomly sampled subset of training data made available for the tree t



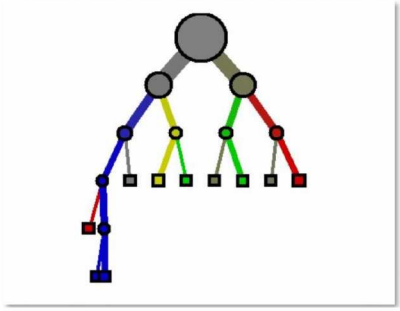
2. Randomized Node Optimization (RNO)

Introduce randomness control parameter ρ that determines how many sets of test parameters are tested to determine the splitting function

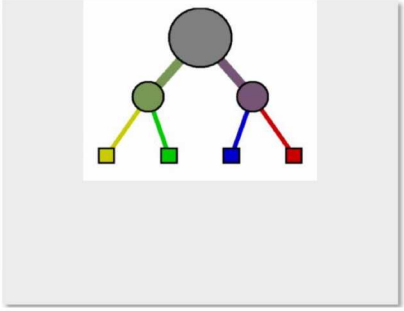
$\rho = |T|$: Take all possible parameters → no randomness

$\rho = 1$: Take one set of random parameters → full randomness

Small value of ρ; little tree correlation.



Large value of ρ; large tree correlation.



Training

For each node take a random set of parameters determined by ρ and take the parameters maximizing the information gain.

Classification:

Information Gain:

$$I = H(\mathcal{S}_j) - \sum_{i=\text{L,R}} \frac{|\mathcal{S}_j^i|}{|\mathcal{S}_j|} H(\mathcal{S}_j^i)$$

Entropy:

$$H(\mathcal{S}) = -\sum_{c \in \mathcal{C}} p(c) \log(p(c))$$

Regression:

Information Gain:

$$I = \sum_{(x,y) \in \mathcal{S}_j} \log\left(\sigma_y(x)\right) - \sum_{i \in \{\text{L,R}\}} \left(\sum_{(x,y) \in \mathcal{S}_j^i} \log\left(\sigma_y(x)\right) \right)$$

Entropy:

$$H(\mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} \int_y p(y|x) \log p(y|x)\, dy \qquad p(y|x) \sim N\left(y; \overline{y}, \sigma_y^2(x)\right)$$

$$H(\mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} \frac{1}{2} \log\left((2\pi e)^2 \sigma_y^2(x)\right)$$ *Differential entropy of Gaussian*

CART-Algorithm Objective:

$$E = \sum_{(x,y) \in \mathcal{S}_j} (y - \overline{y}_j)^2 - \sum_{i \in \{\text{L,R}\}} \left(\sum_{(x,y) \in \mathcal{S}_j^i} (y - \overline{y}_j)^2 \right)$$

→ special instance of the more general objective function

Density:

Information Gain:

$$I = H(\mathcal{S}_j) - \sum_{i \in \{\text{L,R}\}} \frac{|\mathcal{S}_j^i|}{|\mathcal{S}|} H(\mathcal{S}_j^i)$$

Entropy:

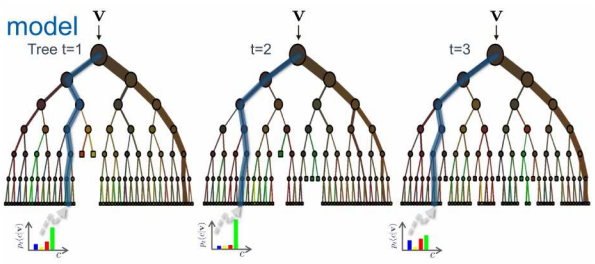
$$H(\mathcal{S}) = \frac{1}{2} \log\left((2\pi e)^d |\Lambda(\mathcal{S})|\right)$$ ← covariance matrix

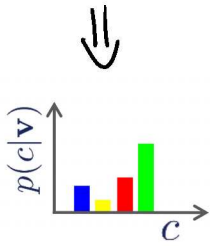
→ similar to regression since we estimate the mean and variance

Classification Forests

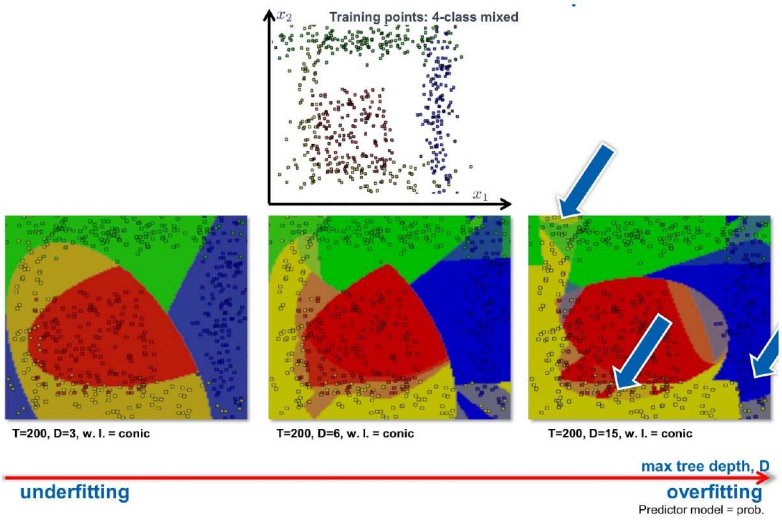
Ensemble Model:

$$p(c|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(c|\mathbf{v})$$

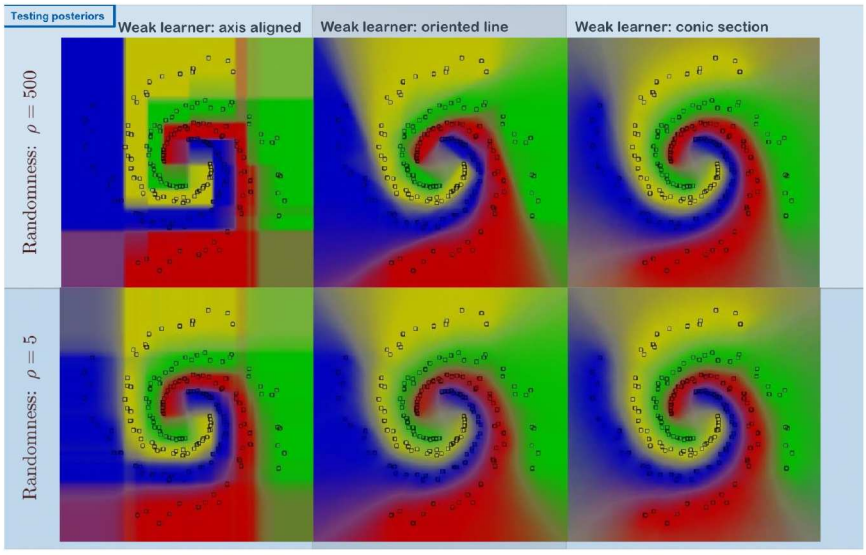




Effect of Tree Depth

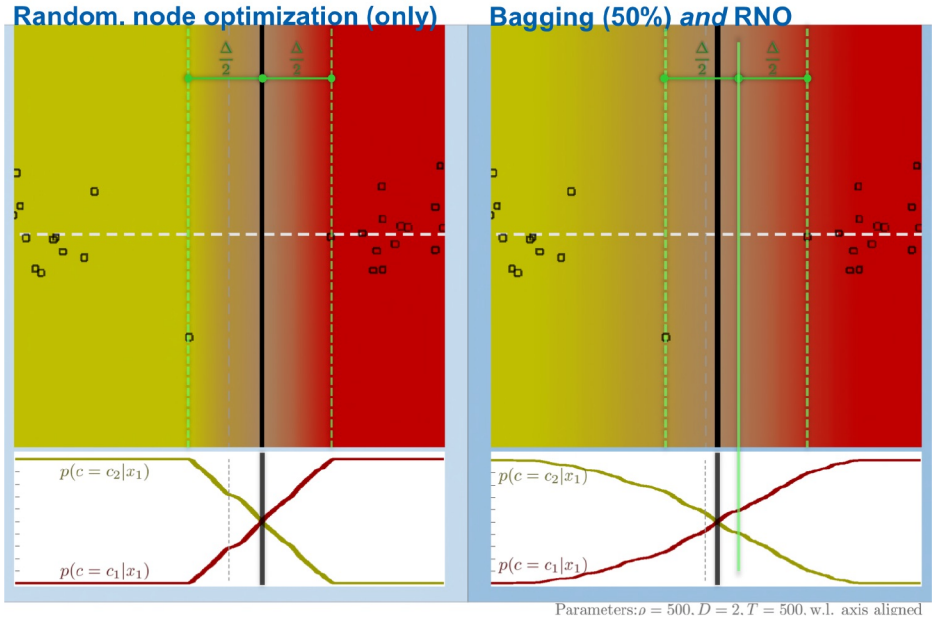


Effect of Learner Model and Randomness



Max-Margin Property

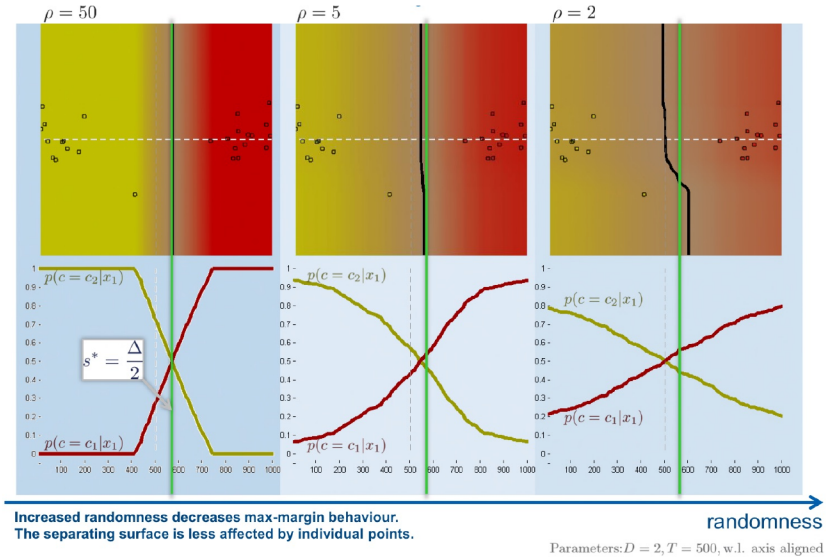
→ similar to SVM random forrest produce a distribution that places the hyperplane where $p(c|x)=0.5$ in the middle of the max-margin corridor



RNO: Hyperplanes lies perfectly in max-margin

Bagging: Since the "support vectors" are missing in some training sets the hyperplane shifts

Effect of Randomness:



Regression Forests

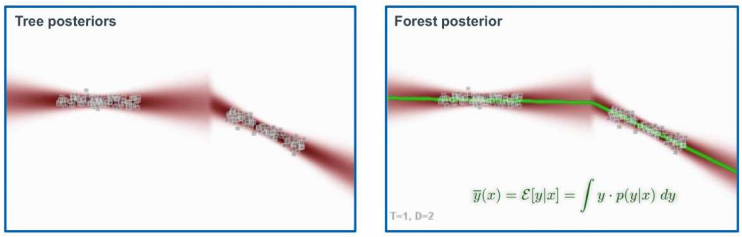
Ensemble Model: $$p(\mathbf{y}|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(\mathbf{y}|\mathbf{v})$$

Probabilistic, non-linear Regression

Single Tree:

→ Each internal node splits the feature space into two subspaces that are approximated by different functions

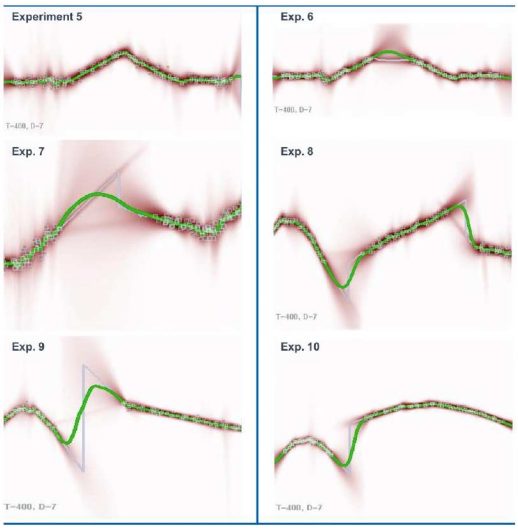
→ leads to artifacts

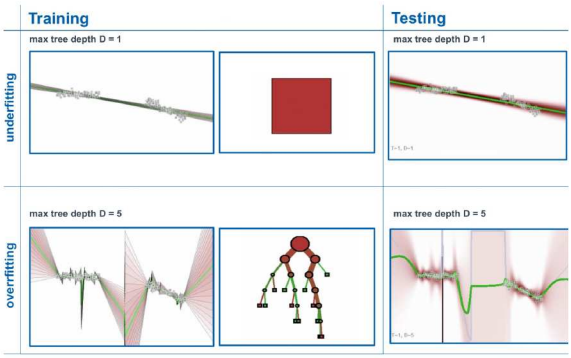


Forests:

→ smooth interpolation between gaps in the training data

→ uncertainty increases with the distance to the training data



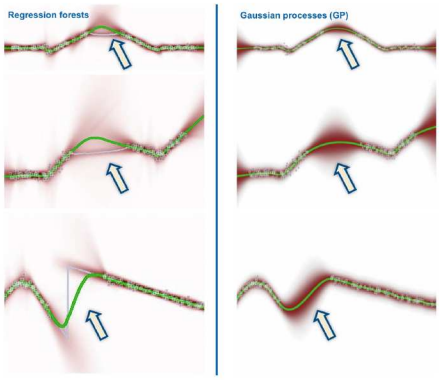


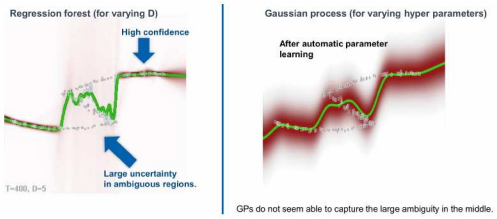
Comparison with Gaussian Processes

→ similar behavior of the mean

→ forest have multimodal distribution

→ quality and shape of the distribution solely depends on the leaf node





Dealing with Non-functions

→ forests are able to represent large ambiguities

Convergence Theorem

Assumptions:

1. Training samples are i.i.d.
2. Trees are trained independently
3. Number of splitting functions per node are sampled from Poisson distribution
4. Stopping criteria: minimum samples per leaf $k(n)$

Notation:

$\mathbb{E}[\Theta | X=x]$: Unknown mean prediction with $\mathbb{E}[\Theta^2] < \infty$
$A(n)$: Training data
Z_t: Randomness of a tree
X: Input
Θ: Predicted values
T: Number of trees

Then: (optimal regressor)

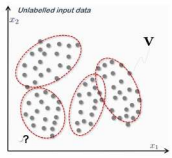
$$\mathbb{E}_{X,A(n),Z_t}\left[\left|\frac{1}{T}\sum_{t}^{T}\mathbb{E}\left[\theta|X,A(n),Z_t\right]-\mathbb{E}\left[\theta|X\right]\right|^2\right]\to 0$$

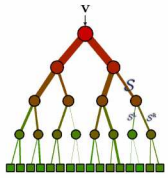
for $k_n \to \infty$ and $k_n/n \to 0$ as $n \to \infty$.

- $k_n \to \infty$: optimal classification → all samples in one leaf
- $k_n/n \to 0$: n grows faster → as sample grow, depths and leafs increase

→ As the amount of training data increases, the tree grows and approximates the optimal regressor

Density Forests





Task: Perform density estimation on unlabeled data to approximate the generative distribution $p(\mathbf{v})$

Tree Output: $$p_t(\mathbf{v}) = \frac{\pi_{l(\mathbf{v})}}{Z_t} \mathcal{N}(\mathbf{v}; \boldsymbol{\mu}_{l(\mathbf{v})}, \Lambda_{l(\mathbf{v})})$$

with $\pi_{l(\mathbf{v})} = |\mathcal{S}_{l(\mathbf{v})}|/|\mathcal{S}_0|$ → proportion of data points in leaf l

Z_t: Normalization constant

$\mathcal{N}(\mathbf{v}; \boldsymbol{\mu}_{l(\mathbf{v})}, \Lambda_{l(\mathbf{v})})$: Gaussian in leaf l

→ Piece-wise Gaussian defined by the leaf's boundaries

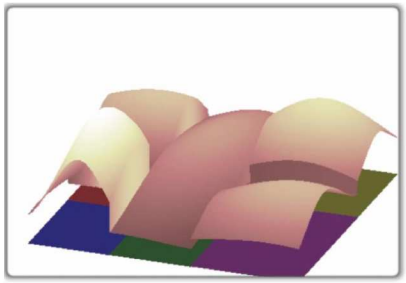
Partition function:

$$Z_t = \int_{\mathbf{v}} \left(\sum_l \pi_l \, \mathcal{N}(\mathbf{v}; \boldsymbol{\mu}_l, \Lambda_l) \, p(l|\mathbf{v}) \right) d\mathbf{v}$$

$$\Leftrightarrow Z_t = \int_{\mathbf{v}} \pi_{l(\mathbf{v})} \, \mathcal{N}\left(\mathbf{v}; \boldsymbol{\mu}_{l(\mathbf{v})}, \Lambda_{l(\mathbf{v})}\right) d\mathbf{v}$$

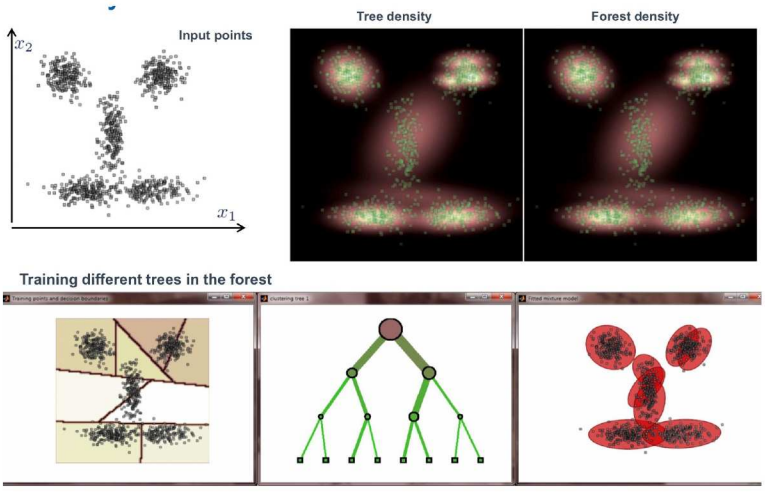
$p(l|v)$ becomes delta-dirac function since each point is assigned to a single leaf

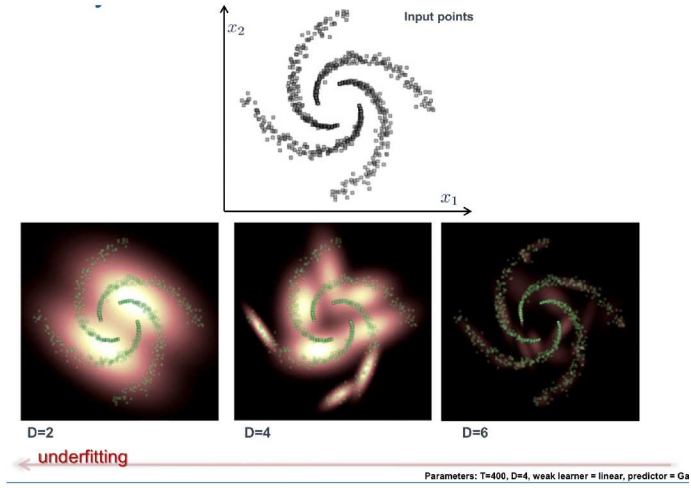
Approximation:



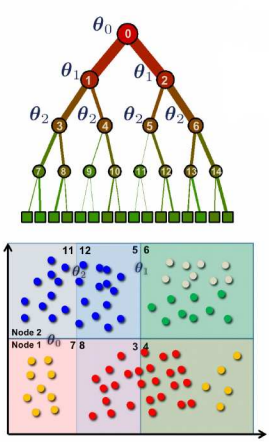
1. Compute via cummulative multi-variate gaussians
   → only possible with axis-aligned splitting functions
2. Grid-based numerical integration

$$Z_t \approx \Delta \sum_i \pi_{l(\mathbf{v}_i)} \; \mathcal{N}(\mathbf{v}_i; \boldsymbol{\mu}_{l(\mathbf{v}_i)}, \Lambda_{l(\mathbf{v}_i)})$$





Variants



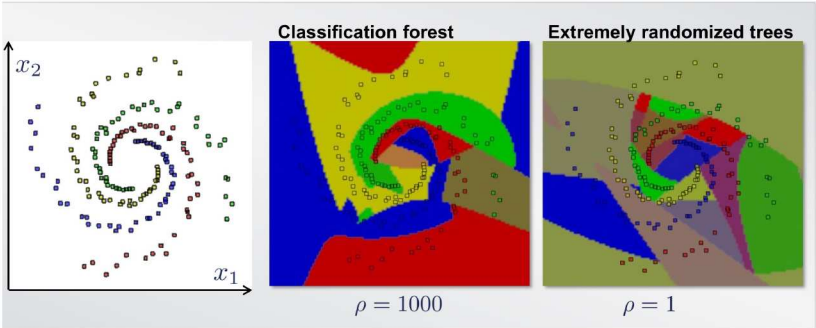
Ferns

All nodes on the same level perform the same test.

→ simpler but less rich and flexible
→ takes longer to converge (more trees)
→ underconfident

Extremly Randomized Trees

Forests with randomization parameter ρ set to 1

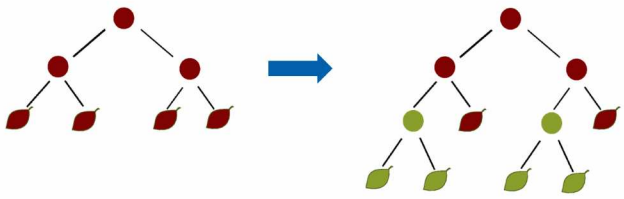


Properties:

Ferns and ERTs have less parameters and may over-fit less

Incremental Learning

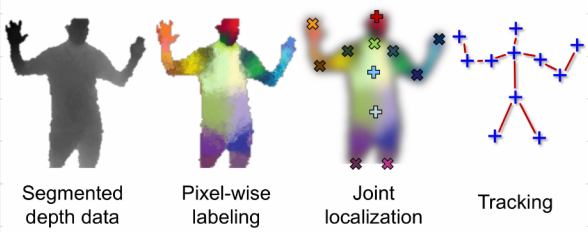
Instead of re-training as soon as new data arrives, continue training at the leafs:

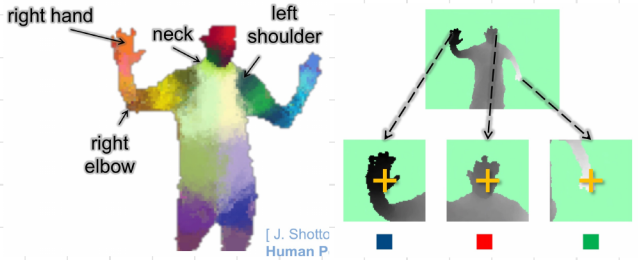


II Random Forests Applications

Pose Estimation

Pipeline:



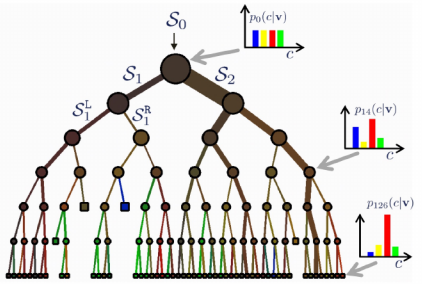


Body Part Classification

Classification:

Input: Sampled patches from an image with a foreground pixel in the center

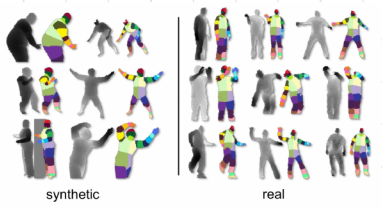
Task: Separate data based on class labels



Training data:

Real data: 500k MoCap frame → 100k redundant poses are removed

Synthetic data: 500k rendered images with body models

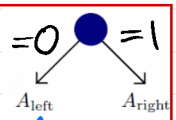


Splitting function:

→ compare depth to center
→ axis-alligned

$$A_L(\phi) = \{\mathcal{P} \in A_{node} | f_\phi(P) = 0\}$$

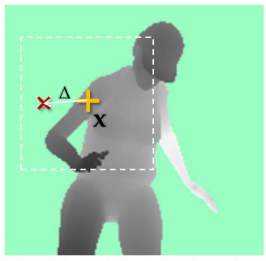
$$A_R(\phi) = \{\mathcal{P} \in A_{node} | f_\phi(P) = 1\}$$



$$f_\phi(\mathcal{P}(\boldsymbol{x})) = \begin{cases} 0 & \text{if } d(\boldsymbol{x}) - d\left(\boldsymbol{x} + \frac{\boldsymbol{v}}{d(\boldsymbol{x})}\right) < \tau \\ 1 & \text{otherwise.} \end{cases}$$

center

$$\phi = (\boldsymbol{v}, \tau)$$



Training: Take best test based on information gain

$$\phi^* = \underset{\phi \in \Phi}{\operatorname{argmax}}\ g(\phi, A_{node})$$

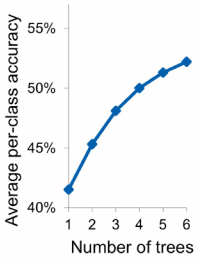
$$g(\phi, A_{node}) = \mathcal{H}(A_{node}) - \sum_{S \in \{L,R\}} \frac{|A_S(\phi)|}{|A_{node}|} \mathcal{H}(A_S(\phi))$$

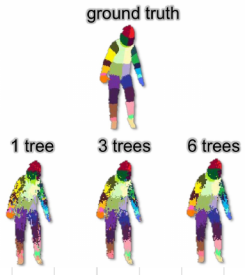
$$\mathcal{H}_c(A) = -\sum_c p(c|A) \log(p(c|A))$$

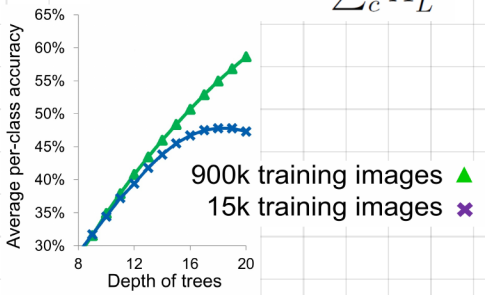
Stopping criteria: maximum depth, minimum samples

Leaves: Store class probabilities $p(c|L(\mathcal{P}(\boldsymbol{x}))) = \frac{A_L^c}{\sum_c A_L^c}$

Evaluation:







Body Pose Estimation with Pictorial Structure

Pictorial Structure:

Deformable tree structure with vertices indicating parts and edges the link between them
→ here joints are modeled with the nose as the anchor

Model distribution:

$$p(\mathcal{J}|\mathbf{I}) \propto \prod_{k} \phi_k(\mathbf{j}_k) \cdot \prod_{(k,l)\in E} \psi_{kl}(\mathbf{j}_k, \mathbf{j}_l)$$

gaussian

unary potential appearance

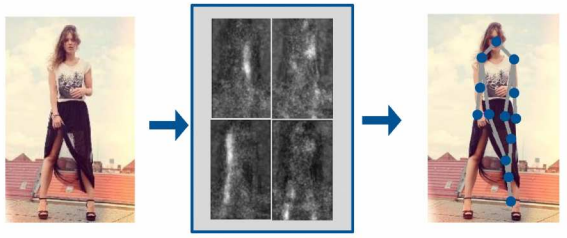
binary potential kinematic prior → defined by kinematic dam

Independent Part Detectors:

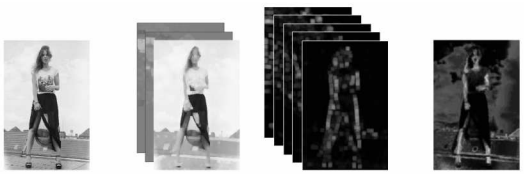
$$\phi_k(\mathbf{j}_k) = p(\mathbf{j}_k|\mathbf{I}, \mathcal{L})$$

→ estimate each body part independently
→ sampled patches do not take their spatial surroundings into account

$\mathcal{L}$: set of body parts



Features:

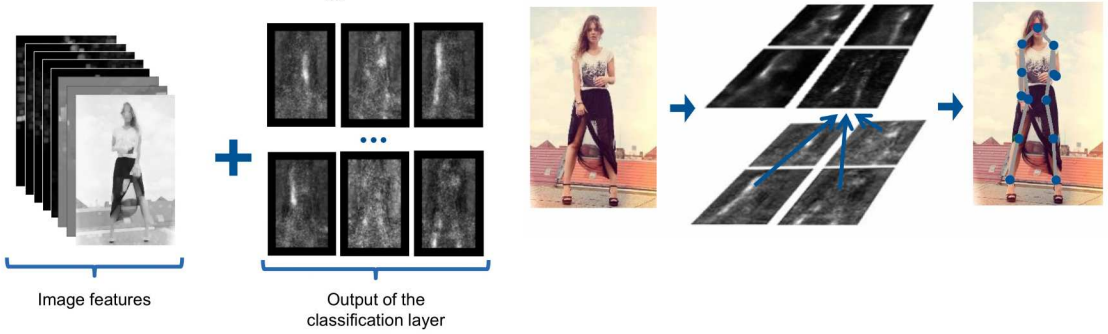


Dependent Part Detectors:

$$\phi_k(\mathbf{j}_k, \mathcal{L}) = p(\mathbf{j}_k|\mathbf{I}, \mathcal{L})$$

1. Compute independent part potentials
2. Compute unary potentials using the features from the first stage

Features:



Evaluation:

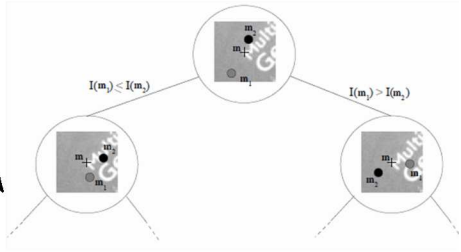
	Avg.	Nose	Should.	Hip	Elbow	Wrist	Knee	Ankle
Without Pictorial Structure								
Body Part Template	**31.15**	61.15	51.40	24.39	12.64	21.1	30.71	36.0
Body Joint Regression	**31.20**	70.97	46.45	22.00	10.20	17.87	35.80	34.97
Dep. Joint Regression	**40.61**	76.65	57.55	30.38	21.29	26.77	44.71	44.90
With Pictorial Structure								
Dep. Joint Regression	**49.21**	69.16	62.39	59.87	39.22	29.80	49.93	44.06

Keypoint Recognition

Task: Find a keypoint from a baseline image in a new image

Training data: Extract patches with prominent key-points
→ augment data for more robustness

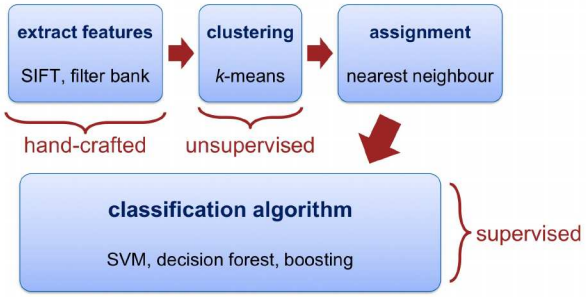
Classification: Forrest classifies patches as key-points



Splitting Function: pixel-wise comparison

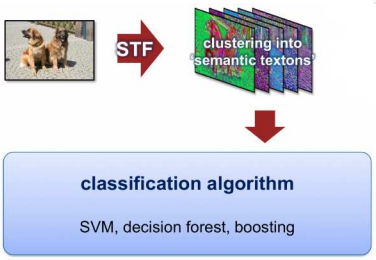
Object Recognition:

General Pipeline:

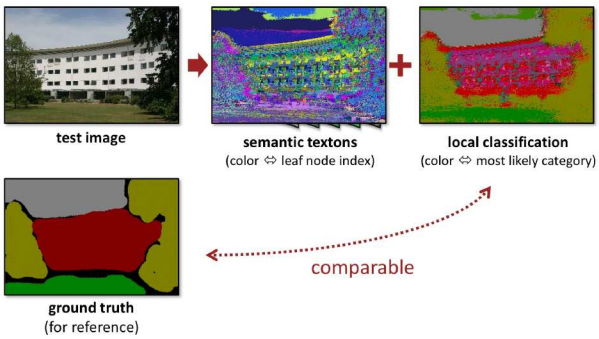


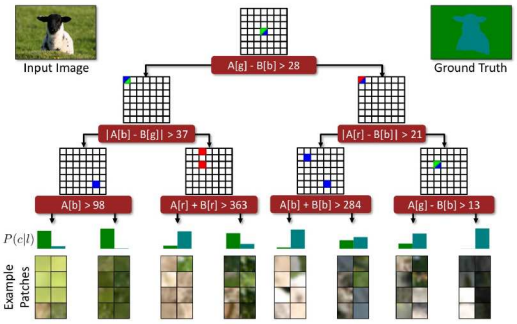
Semantic Texton Forest

Build codebook with trees which learns object category association



Evaluation:

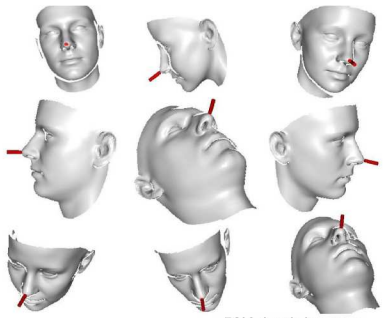




Head Pose Estimation

Approach: Build a regression forest to estimate the orientation of the nose:

$$\boldsymbol{\theta}_i = \{\theta_x, \theta_y, \theta_z, \theta_{yaw}, \theta_{pitch}, \theta_{roll}\}$$



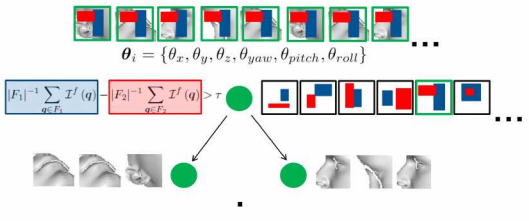
50K depth images
- ± 90° Yaw
- ± 50° Pitch
- ± 20° Roll
- Identity variations (not expression)

Training data: 3D renderings of a face model

Splitting function:

$$|F_1|^{-1} \sum_{\boldsymbol{q} \in F_1} \mathcal{I}^f(\boldsymbol{q}) - |F_2|^{-1} \sum_{\boldsymbol{q} \in F_2} \mathcal{I}^f(\boldsymbol{q}) > \tau$$

→ compare average intensity between two rectangles



Leaves: $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}; \overline{\boldsymbol{\theta}}, \boldsymbol{\Sigma})$

→ Multi-variate gaussian over head pose

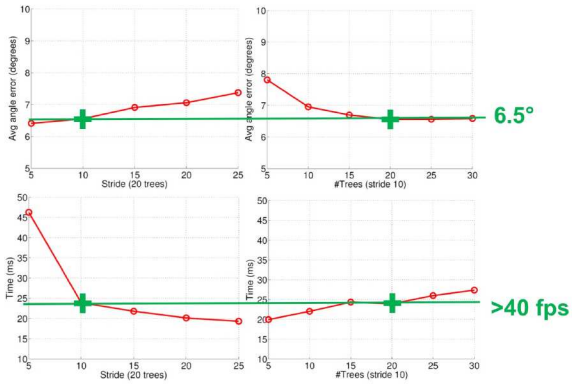
Training: Maximize information gain

$$\phi^* = \underset{\phi \in \Phi}{\operatorname{argmax}}\, g(\phi, A_{node})$$

$$g(\phi, A_{node}) = \mathcal{H}(A_{node}) - \sum_{S \in \{L,R\}} \frac{|A_S(\phi)|}{|A_{node}|} \mathcal{H}(A_S(\phi))$$

$$\mathcal{H}_r(A) = \sum_c \left(\sum_{\boldsymbol{d} \in D_c^A} \left\| \boldsymbol{d} - \frac{1}{|D_c^A|} \sum_{\boldsymbol{d}' \in D_c^A} \boldsymbol{d}' \right\|^2 \right) |c, \mathcal{P})) \, \mathrm{d}\boldsymbol{d}$$

Evaluation:

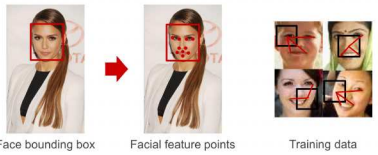


Facial Feature Detection

Task: Predict the position of fixed facial feature points:

Regression forest

Predict the relative position w.r.t the patch center: $\theta_i = \{\theta_i^1, \theta_i^2, \ldots \theta_i^K\}$

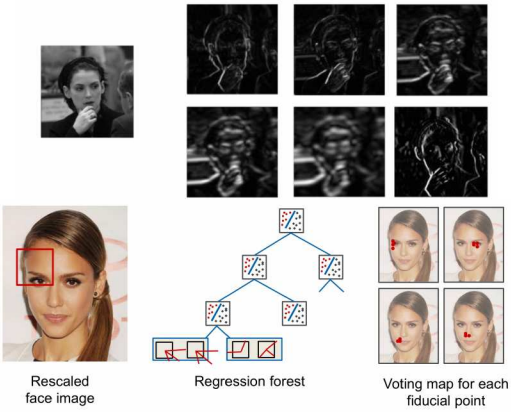


Problem: patches vote for points not visible
→ strong bias towards a neutral face

Splitting function: several feature channels using 24 Gabor wavelets

Testing: Weight vote based on the distance to the facial point:

$$\exp\left(-\frac{\|\overline{\theta^n}\|^2}{\gamma}\right)$$

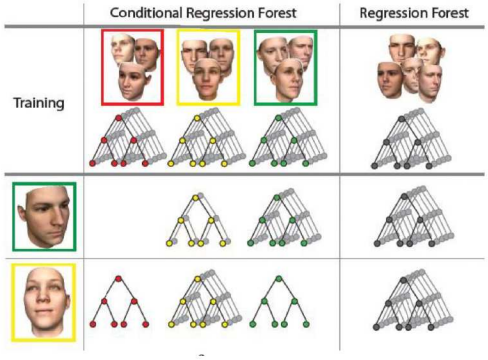


Conditional Regression Forest

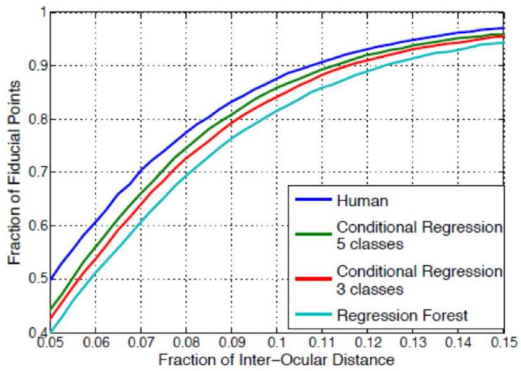
Cluster data first and train forests on the different clusters

Testing: $p(\boldsymbol{d}^n|\mathcal{P}) = \int p(\boldsymbol{d}^n|\omega, \mathcal{P}) p(\omega|\mathcal{P})\, \mathrm{d}\omega$

→ define forest dependent on cluster



Comparison:



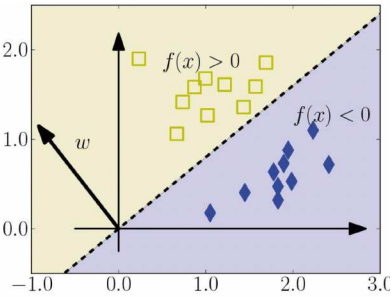
VI Maximum Margin Classification and kernels

Task: Linear Classification

Given: Data points: $X = \{x_1, \ldots, x_n\},\ x_i \in \mathbb{R}^d$

Class labels $Y = \{y_1, \ldots, y_n\},\ y_i \in \{+1, -1\}$

Find classification rule: $g : \mathbb{R}^d \to \{-1, +1\}$



With: $g(x) = \operatorname{sign} f(x)$ with $f : \mathbb{R}^d \to \mathbb{R}$

$$f(x) = a^1 x^1 + a^2 x^2 + \ldots a^n x^n + a^0$$

Simplified Notation: $f(x) = \langle \hat{w}, \hat{x} \rangle$

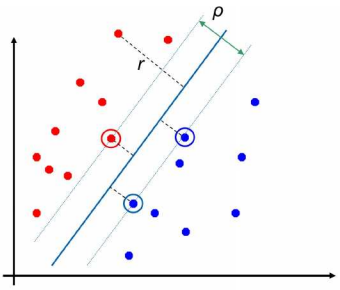
With: $\hat{x} = (1, x),\ \hat{w} = (a^0, \ldots, a^n)$

Definition: Support Vectors

Data points closest to the separating hyperplane

Definition: Classification Margin

Distance of data point to separator: $r = \dfrac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}$



Margin ρ is defined by the distance of the support vectors

→ maximizing this margin leads to a more robust classification and tends to generalize better to new data points

Maximum Margin Classification

$$\max_{\substack{w \in \mathbb{R}^d, \|w\|=1 \\ \gamma \in \mathbb{R}^+}} \gamma \qquad \text{subject to} \qquad y_i \langle w, x_i \rangle \geq \gamma \quad \text{for } i = 1, \ldots n.$$

$$\Leftrightarrow$$

$$\min_{w \in \mathbb{R}^d} \|w\|^2 \qquad \text{subject to} \qquad y_i \langle w, x_i \rangle \geq 1 \quad \text{for } i = 1, \ldots n.$$

Properties:

- objective function is convex and differentiable
  → no local minima
- constraints are all linear
- globally optimal w can be found in $O(nd^2 + d^3)$

Soft-Margin Classification

Often data is not linear separable. Thus allow violations of the constraints by introducing a slack variable ξ_i:

$$\min_{w\in\mathbb{R}^d,\xi_i\in\mathbb{R}^+} \|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

subject to

$$y_i\langle w, x_i\rangle \geq 1-\xi_i \qquad \text{for } i=1,\dots n.$$

ξ_i: Slack variable — **large:** heavy violations allowed → worse classification; **small:** similar to previous formulation

C: Regularization / trade-off parameter

- **small:** constraints are easily ignored
- **large:** constraints are hard to ignore
- **$C=\infty$:** hard-margin → no errors

Reformulation with Hinge loss:

$$\min_{w\in\mathbb{R}^d} \|w\|^2 + C\sum_{i=1}^{n}\left[1-y_i\langle w, x_i\rangle\right]_+$$

where $[t]_+ := \max\{0,t\}$.

→ unconstrained optimization
→ non-differentiable
→ use stochastic gradient descent

Generalized Linear Classifier

To make the data linearly separable, we map the points into a feature space

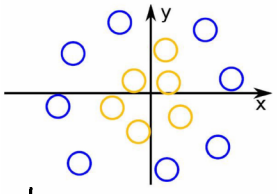
Feature Map: $\varphi : \mathbb{R}^k \to \mathbb{R}^m$

$$\min_{w\in\mathbb{R}^m,\xi_i\in\mathbb{R}^+} \|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

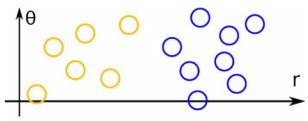
subject to

$$y_i\langle w, \varphi(x_i)\rangle \geq 1-\xi_i \qquad \text{for } i=1,\dots n.$$

cartesian:



polar:



Lemma 1:

Let: Data: $(x_i)_{i=1\dots n}$ with $x_i \neq x_j$ for $i\neq j$

Feature Map: $\varphi: \mathbb{R}^k \to \mathbb{R}^m$

Then: $\varphi(x_i)_{i=1\dots n}$ is linearly independent

$\Rightarrow \varphi(x_i)_{i=1\dots n}$ are linearly separable

Lemma 2: If we choose $m>n$ large enough, we can always find such a map φ

Representer Theorem (Dual formulation)

Theorem: w solves

$$(*) \quad \min_{w\in\mathbb{R}^m,\xi_i\in\mathbb{R}^+} \|w\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{subject to } y_i\langle w,\varphi(x_i)\rangle \geq 1-\xi_i$$

then $w = \sum_j \alpha_j\varphi(x_j)$ for some $\alpha \in \mathbb{R}^n$.

→ if we have a solution, we can convert it to the dual formulation

Proof: Show that we can convert every solution w

1. Define

Space of vectors spanned by the linear combinations of training samples:

$$V := \operatorname{span}\{\varphi(x_1),\ldots,\varphi(x_n)\} = \{\textstyle\sum_j \alpha_j\varphi(x_j) : \alpha \in \mathbb{R}^n\} \subset \mathbb{R}^m$$

Space of vectors that can not be represented through a linear combination of input vectors:

$$V^\perp := \{u \in \mathbb{R}^m : \langle u, v\rangle = 0 \text{ for all } v \in V\}$$

↑ orthogonal to all vectors in V

2. Decompose w as: $w = w_V + w_{V^\perp}$ with $w_V \in V$, $w_{V^\perp} \in V^\perp$

→ w can be written as a combination of V and its dual space

Then:

(I) from $\langle w_{V^\perp}, \varphi(x_i)\rangle = 0$ follows $\langle w_V, \varphi(x_i)\rangle = \langle w, \varphi(x_i)\rangle$

(II) from $\langle w_V, w_{V^\perp}\rangle = 0$ follows $\|w\|^2 = \|w_V\|^2 + \|w_{V^\perp}\|^2$

3. Show with contradiction proof that $w_{V^\perp}=0 \Rightarrow w \in V$

Assumption: $w_{V^\perp} \neq 0$

Then: $\|w_{V^\perp}\|^2 > 0 \Rightarrow \|w\|^2 > \|w_V\|^2$

$\Rightarrow$ w_V solves $(*)$ with same ξ_i but smaller norm. ↯

$\Rightarrow$ $w_{V^\perp} = 0$, ergo $w = w_V \in V$ □

Kernel Trick

Idea: Rewrite the optimization problem using the representer theorem. The inputs are solely used as inner products in this formulation. Instead of computing the embeddings, we can compute the inner product in the feature space through a kernel function.

Formulation:

$$\min_{\alpha_i \in \mathbb{R}, \xi_i \in \mathbb{R}^+} \sum_{j,k=1}^{n} \alpha_j \alpha_k k(x_j, x_k) + C \sum_{i=1}^{n} \xi_i$$

subject to

$$y_i \sum_{j=1}^{n} \alpha_j k(x_j, x_i) \geq 1 - \xi_i \qquad \text{for } i = 1, \dots n.$$

Evaluate new data: $f(x) = \langle w, \varphi(x) \rangle = \sum_{i=1}^{n} \alpha_i k(x_i, x)$

Training: Use kernel matrix $K_{ij} := k(x_j, x_i)$

Derivation:

Initial problem: $\min_{w \in \mathbb{R}^m, \xi_i \in \mathbb{R}^+} \|w\|^2 + C \sum_{i=1}^{n} \xi_i$ with $y_i \langle w, \varphi(x_i) \rangle \geq 1 - \xi_i$

1. Rewrite with representer theorem:

$$\min_{\alpha_i \in \mathbb{R}, \xi_i \in \mathbb{R}^+} \|\sum_{j=1}^{n} \alpha_j \varphi(x_j)\|^2 + C \sum_{i=1}^{n} \xi_i \text{ with } y_i \langle \sum_{j=1}^{n} \alpha_j \varphi(x_j), \varphi(x_i) \rangle \geq 1 - \xi_i$$

2. Apply $\|w\|^2 = \langle w, w \rangle$ and linearity

$$\min_{\alpha_i \in \mathbb{R}, \xi_i \in \mathbb{R}^+} \sum_{j,k=1}^{n} \alpha_j \alpha_k \langle \varphi(x_j), \varphi(x_k) \rangle + C \sum_{i=1}^{n} \xi_i \text{ with } y_i \sum_{j=1}^{n} \alpha_j \langle \varphi(x_j), \varphi(x_i) \rangle \geq 1 - \xi_i$$

3. Apply the kernel trick

$$\min_{\alpha_i \in \mathbb{R}, \xi_i \in \mathbb{R}^+} \sum_{j,k=1}^{n} \alpha_j \alpha_k k(x_j, x_k) + C \sum_{i=1}^{n} \xi_i \text{ with } y_i \sum_{j=1}^{n} \alpha_j k(x_j, x_i) \geq 1 - \xi_i$$

Dual Formulation: Dualization with Langrian multipliers $\forall i$

$$\min_{\alpha_i \in \mathbb{R}^+} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j k(x_i, x_j) + C \sum_{i=1}^{n} \alpha_i$$

subject to

$$\sum_{i=1}^{n} y_i \alpha_i = 0$$

Definition: Hilbert space

A hilbert space is an inner product: $\langle .,. \rangle_{\mathcal{H}} : H \times H \to \mathbb{R}$ such that:

1. symmetric: $\langle v, v' \rangle_{\mathcal{H}} = \langle v', v \rangle_{\mathcal{H}}$ for all $v, v' \in H$,
2. positive definite: $\langle v, v \rangle_{\mathcal{H}} \geq 0$ for all $v \in H$, where $\langle v, v \rangle_{\mathcal{H}} = 0$ only for $v = \vec{0} \in H$
3. bilinear: $\langle av, v' \rangle_{\mathcal{H}} = a \langle v, v' \rangle_{\mathcal{H}}$ for $v \in H, a \in \mathbb{R}$
   $\langle v + v', v'' \rangle_{\mathcal{H}} = \langle v, v'' \rangle_{\mathcal{H}} + \langle v', v'' \rangle_{\mathcal{H}}$

Definition: Kernel

A function $k: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a (positive definite) kernel function, if:

1. k is symmetric, i.e. $k(x, x') = k(x', x)$ for all $x, x' \in \mathcal{X}$.
2. For any set of points $x_1, \ldots, x_n \in \mathcal{X}$, the matrix
   $$K_{ij} = (k(x_i, x_j))_{i,j}$$
   is positive (semi-)definite, i.e. for all vectors $t \in \mathbb{R}^n$:
   $$\sum_{i,j=1}^{n} t_i K_{ij} t_j \geq 0.$$

→ kernels are a measurement of similarity

Theorem:

Let: kernel function: $k: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$

Then: There exists a hilbert space $\mathcal{H}$ and mapping $\varphi: \mathcal{X} \to \mathcal{H}$, such that:

$$k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$$

Non positive definiteness:

Dependend on the data, the optimization will most likely fail:

If $k(x, x')$ is not positive definite:

- the matrix $K = \big(k(x_i, x_j)\big)_{i,j}$ can have negative eigenvalues,
- let $\boldsymbol{v} = (v_1, \ldots, v_n)$ be an eigenvector with eigenvalue $\lambda < 0$,
- for $\alpha_i = y_i v_i$, we have $\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j k(x_i, x_j) = \lambda \|v\|^2$,
- for $\|v\| \to \infty$, we have $\lambda \|v\|^2 \to -\infty$, because $\lambda < 0$.

Constructing kernels

1. From scratch
→ we can explicitely show the underlying embedding function

For any $\varphi : \mathcal{X} \to \mathbb{R}^m$, $k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathbb{R}^m}$ is a kernel.
Example: $\varphi(x) = \big($"# of red pixels in image x", green,blue $\big)$.

Any norm $\|.\| : V \to \mathbb{R}^m$ that fulfills the *parallelogram equation*
- $\|x+y\|^2 + \|x-y\|^2 = 2\|x\|^2 + 2\|y\|^2$

induces a kernel by *polarization*:
- $k(x,y) := \frac{1}{2}\big(\|x+y\|^2 - \|x\|^2 - \|y\|^2\big)$.

Example: $\mathcal{X} =$ *time series with bounded values*, $\|x\|^2 = \sum_{t=1}^{\infty} \frac{1}{2^t} x_t$

If $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is *conditionally positive definite*, i.e.
- $\sum_{i,j=1}^n t_i d(x_i, x_j) t_j \geq 0$ for any $t \in \mathbb{R}^n$ with $\sum_i t_i = 0$

for $x_1, \ldots, x_n \in \mathcal{X}$ for any $n \in \mathbb{N}$, then
- $k(x,x') := \exp(-d(x,x'))$ is a positive kernel.

Example: $d(x,x') = \|x - x'\|_{L^2}^2$. $k(x,x') = e^{-\|x-x'\|_{L^2}^2}$

2. Construct from other kernels

Let: kernels k_1, k_2

Then:

1. $a k_1$ & $a + k_1$ are kernels for $a > 0$
2. $k_1 + k_2$ & $k_1 k_2$ are kernels
3. $\exp(k_1)$ is a kernel

Examples for $\mathbb{R}^d$:

Linear combination: $\sum_j \alpha_j k_j$ with $\alpha_j \geq 0$

Polynomials: $k(x,x') = (1 + \langle x, x' \rangle)^m$, $m > 0$

Gaussian: $k(x,x') = \exp\left(-\frac{\|x-x'\|^2}{2\sigma^2}\right)$ with $\sigma > 0$

Examples for other $\mathcal{X}$:

$k(h,h') = \sum_{i=1}^n \min(h_i, h_i')$ for n-bin histograms h, h'.

$k(p,p') = \exp(-KL(p,p'))$ with KL the symmetrized *KL-divergence* between positive probability distributions.

$k(s,s') = \exp(-D(s,s'))$ for *strings* s, s' and $D =$ *edit distance*

Advantages:

1. Memory usage

   Storing $\varphi(x_1), \ldots, \varphi(x_n)$ requires $O(nm)$ memory.
   Storing $k(x_1, x_1), \ldots, k(x_n, x_n)$ requires $O(n^2)$ memory.

2. Speed

   Computing the kernel function might be faster than embedding the inputs.

   Example: $\varphi : x \mapsto (\cos(x), \sin(x)) \in \mathbb{R}^2$

$$\begin{aligned}\langle \varphi(x_i), \varphi(x_j) \rangle &= \langle\ (\cos(x_i), \sin(x_i)), (\cos(x_j), \sin(x_j))\ \rangle \\ &= \cos(x_i)\cos(x_j) + \sin(x_i)\sin(x_j) \ = \cos(x_i - x_j)\end{aligned}$$

3. Flexibility

   → kernels measure similarity
   → kernels can be computed without knowing the mapping

VII SVM and Kernel Applications

Definition: Visual similarity

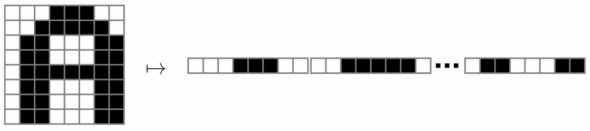
Visual similarity is invariant to:

1. Translations
2. (small) Rotations
3. (small) Changes
4. Blur
5. Brightness / Contrast
6. stroke width
7. Illumination
8. ...

Encoding Invariance

Problem: Kernels are typically not invariant to small changes in the images leading to an unreliable prediction.

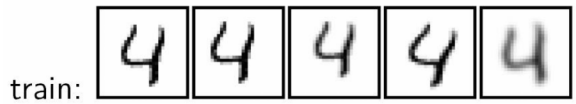
Data representation: flattened images



$\mapsto \quad 0,0,0,1,1,1,0,0,0,0,1,1,1,1,1,0,\cdots,0,1,1,0,0,0,1,1$

$N \times M$ pixels, pixels are values in $[0,255]$ or $[0,1] \Rightarrow$ images $x_1, \ldots, x_n$ are vectors in $\mathbb{R}^{NM}$

Visual similar images have a large distance:



$\|x_{tst} - x_i\|$ is large for all $i = 1, \ldots, n$,

$\Rightarrow \quad k(x_i, x_{tst}) = \exp\left(-\frac{1}{2\sigma^2}\|x_{tst} - x_i\|^2\right) \approx 0,$

$\Rightarrow \quad f(x_{tst}) = \sum_{j=1}^{n} \alpha_j y_j k(x_j, x_{tst}) \approx 0,$

$\Rightarrow$ Decision of the SVM classifier is unrealiable

→ SVMs are a more elaborate nearest neighbor search

→ small changes flip sign

Solutions:

1. Image Normalization

- Resize images to a fixed size
- Perform brightness / contrast normalization
- Translate images to the same center of gravity
- Rotate images to the same orientation
- Calculate a gradient / edge image

2. Invariant Representation

Transform images to a feature representation:

$$\psi : \{images\} \rightarrow \{feature\ representations\}$$

Examples: filter responses, e.g. Fourier, Gabor, ...
calculate **histograms** of properties, e.g. color histogram

3. Increase training data

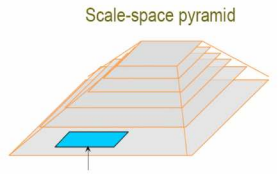
→ does not work for SVMs & Gaussian processes

Add modifications to existing data:

- Add noise
- Apply geometric transformations

Human Detection:

Pipeline:

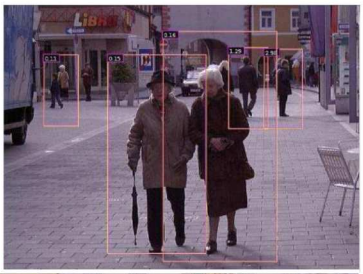


Learning:

Create normalised train image data set
↓
Encode images into feature spaces
↓
Learn binary classifier
↓
Object/Non-object decision

Detection:

Scan image at all scales and locations
↓
Run classifier to obtain object/non-object decisions
↓
Fuse multiple detections in 3-D position & scale space
↓
Object detections with bounding boxes



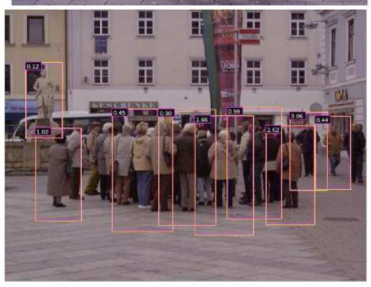
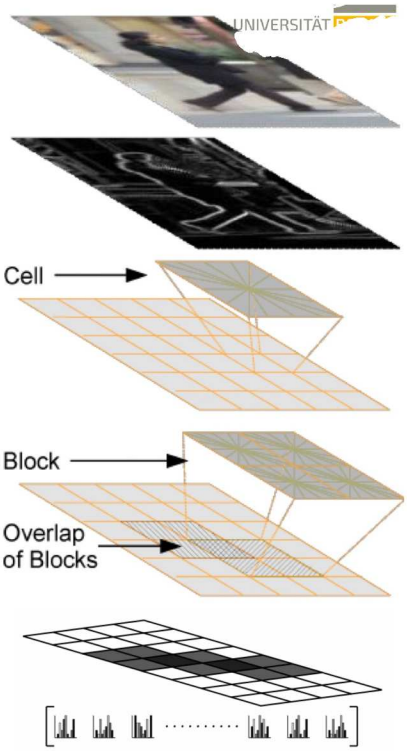


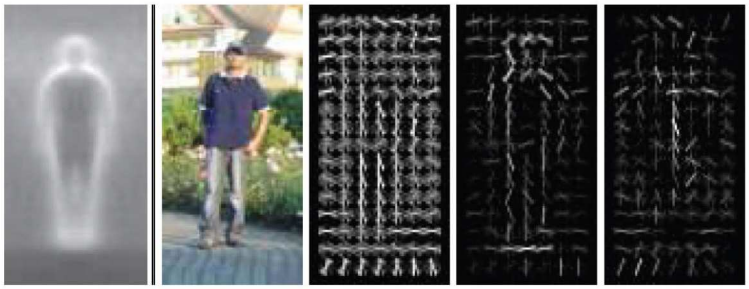
Image descriptor

1. Compute gradients in a region of 64×128
2. Compute histograms of cells of 8×8 pixels
   9 bins for gradient orientations filled with magnitude
   → linearly interpolate into bins
   → bilinearly interpolate spatial cells
3. Normalize histograms within overlapping blocks of 2×2 cells
   → concatenate cell histograms and normalize
   → each cell appears multiple times in the final descriptor
4. Concatenate block histograms



Results:

→ scale invariance through resizing and evaluation on multiple scales
→ rotation invariance is undesirable



Average gradient of training images | Test image | HOG | Pos. weights of SVM | Neg. weights of SVM

$$f(x) = \langle w, x \rangle$$

Local Representations

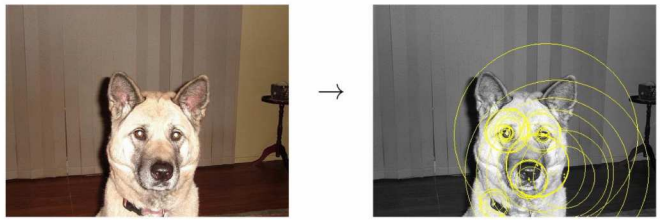
→ variations within a class may be huge
→ features may vary across regions
→ vast parts are background / non-significant

Interest Points:

Extract a set of interest points as an image representation:

$$I_i \mapsto (\theta_i, x_i, y_i, r_i, \alpha_i, d_i)_{i=1,\dots,n_i}$$

θ_j is confidence or "interestingness"
(x_j, y_j) is the center of the region
r_j is the scale of the region (e.g. radius of a circle
α_j is the orientation of the region



Original Approach:

An object is characterized by a few characteristic parts (5-10)

Problems:

- Detected interest points are low-level
- Similar regions can occur in different objects

Alternatives:

1. Find more high-level parts by grouping regions
2. Use statistics of many unreliable regions directly

Set Kernels

kernels working on a set of descriptors disregarding the spatial arrangements

$$I_i \mapsto D_i := \{d_i\}_{i=1,\dots,n_i}$$

→ typically build from simple kernels iterated over the set of descriptors

Averaging kernels:

$$k_{count}(D_i, D_j) := \sum_{i-1}^{n_i} \sum_{i-1}^{n_j} \delta_{d_i = d_j}$$

$$k_{sum}(D_i, D_j) := \frac{1}{n_i n_j} \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} k(d_i, d_j)$$

$$k_{prod}(D_i, D_j) := \left(\prod_{i=1}^{n_i} \prod_{j=1}^{n_j} k(d_i, d_j) \right)^{\frac{1}{n_i n_j}}$$

Matching kernels:

Instead of averaging, match against the highest response of the kernel:

$$d_{match}(D_i, D_j) := \frac{1}{2}\left[\hat{k}(D_i, D_j) + \hat{k}(D_j, D_i)\right]$$

$$\text{with} \quad \hat{k}(D_i, D_j) := \frac{1}{n_i}\sum_{i=1}^{n_i} \max_{j=1,\ldots,n_j} k(d_i, d_j)$$

→ not positive definite

Problems:

Set kernels are slow to compute: $O(N^2 n^2 d)$

kernel matrix of N samples: $O(N^2)$ kernel evaluations.
Each kernel evaluation: compare all pairs of elements in $D_i \times D_j$: $O(n_i \cdot n_j)$
Each comparison: at least $O(d)$: for d-dim vectors

Pyramid Match Kernel

→ positive definite in contrast to conventional match kernels
→ complexity is reduced to $O(nd \log R)$

$$K_\Delta\left(\overbrace{\Psi(\mathbf{X}), \Psi(\mathbf{Y})}^{\text{histogram pyramids}}\right) = \sum_{i=0}^{L} \underbrace{\frac{1}{2^i}}_{\substack{\text{measure of difficulty of} \\ \text{a match at level } i}} \Big(\underbrace{\mathcal{I}\left(H_i(\mathbf{X}), H_i(\mathbf{Y})\right) - \mathcal{I}\left(H_{i-1}(\mathbf{X}), H_{i-1}(\mathbf{Y})\right)}_{\text{number of newly matched pairs at level } i}\Big)$$

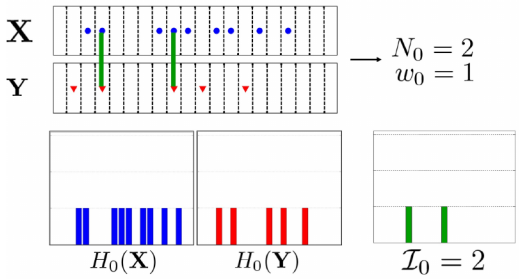
→ compute matches on different scales and count the number of matches

Histogram intersection:

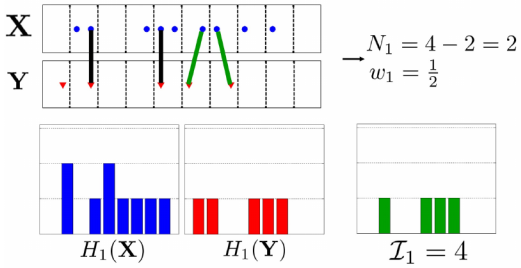
$$\mathcal{I}\left(H(\mathbf{X}), H(\mathbf{Y})\right) = \sum_{j=1}^{r} \min\left(H(\mathbf{X})_j, H(\mathbf{Y})_j\right)$$

Example:

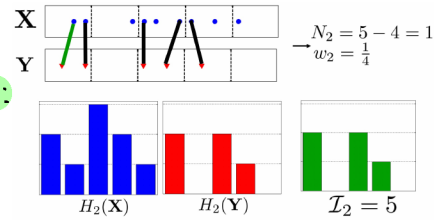
Level 0:



Level 1:



Level 2:



Pyramid match:

$$K_\Delta = \sum_{i=0}^{L} w_i N_i$$
$$= 1(2) + \tfrac{1}{2}(2) + \tfrac{1}{4}(1) = 3.25$$

Optimal match:

$$K = \max_{\pi:\mathbf{X}\to\mathbf{Y}} \sum_{\mathbf{x}_i \in \mathbf{X}} \mathcal{S}(\mathbf{x}_i, \pi(\mathbf{x}_i))$$
$$= 1(2) + \tfrac{1}{2}(3) = 3.5$$

Bag of Visual Words Representation

Discretize the space by using a codebook of descriptors that each data point is matched to.
→ images are represented by histograms

Histogram kernels:

Unnormalized:

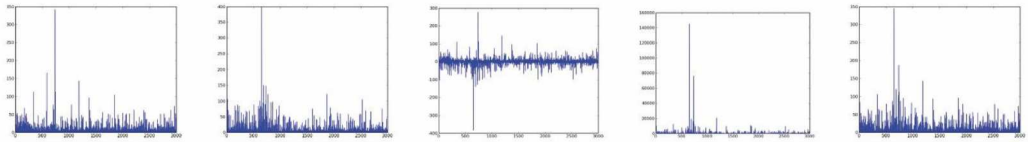
$k(h, h') = \sum_j h_j h'_j$ *linear kernel*
$k(h, h') = (c + \sum_j h_j h'_j)^d$ *polynomial kernel*
$k(h, h') = \exp\left(-\frac{1}{\gamma}\sum_j \|h_j - h'_j\|^2\right)$ *Gaussian kerne*

Normalized:

$k_{HI}(h, h') = \sum_j \min(h_j, h'_j)$.
$k_{bhattacharya}(h, h') = \sum_j \sqrt{h_j h'_j}$
$k_{symKL}(h, h') = \exp\left(-\frac{1}{2}(KL(h|h') + KL(h'|h))\right)$
where $KL(h|h') = \sum_j h_j \log \frac{h_j}{h'_j}$
$k_{\chi^2}(h, h') = \exp\left(-\frac{1}{\gamma}\chi^2(h, h')\right)$ with $\chi^2(h, h') = \sum_j \frac{(h_j - h'_j)^2}{h_j + h'_i}$

Imbalanced Representation:

The histograms consist of a few very large representations that dominate the representation



h_j h'_j $h_j - h'_j$ $(h_j - h'_j)^2$ $\frac{(h_j - h'_j)^2}{h_j + h'_j}$

→ L^2 or gaussian kernels create histograms where ~3 bins make up 25%.

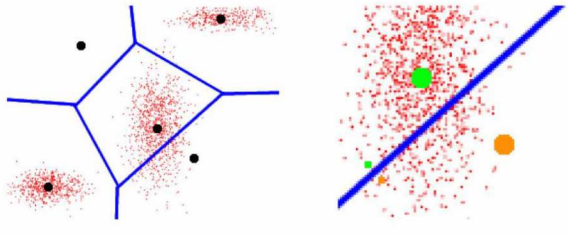
Solutions:

1. Robust kernels: $k_{HI}(h, h') = \sum_j \min(h_j, h'_j)$ $k_{\chi^2}(h, h') = \exp\left(-\frac{1}{\gamma}\sum_j \frac{(h_j - h'_j)^2}{h_j + h'_j}\right)$
   → these kernels work especially well for computer vision tasks
2. Normalize each histogram: $\hat{h}_j := \frac{1}{n_1} h_j$ with $n_1 = \sum_j h_j$
   $\hat{h}_j := \frac{1}{n_2} h_j$ with $n_2 = \sqrt{\sum_j h_j^2}$
3. Non-linear preprocessing: $\hat{h}_j := \sqrt{h_j}$, $\hat{h}_j := \sqrt[3]{h_j}$, $\hat{h}_j := \begin{cases} 1 & \text{for } h_j > \theta_j \\ 0 & \text{else.} \end{cases}$

Problems:

Quantization Artifacts

→similar features are assigned to different bins



Solutions:

1. **Soft quantisation**

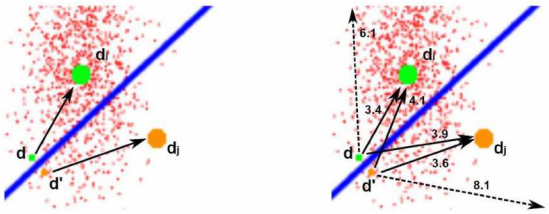
Perform a soft-assignment to the k-nearest neighbors

Weight: $a_k = \exp(-\lambda \| d - d_k \|^2),\ \tilde{a}_k = a_k / \sum_l a_l$

$h(d) = (0, 0, 1, 0, \dots)$ $\quad$ $h(d) = (0.1, 0.4, 0.5, 0.0, \dots)$

$h(d') = (0, 1, 0, 0, \dots)$ $\quad$ $h(d') = (0.0, 0.5, 0.4, 0.1, \dots)$

$k_{HI}(h(d), h(d')) = 0$ $\quad$ $k_{HI}(h(d), h(d')) = 0.8$



2. **Gaussian mixture models**

Represent the training data through a mixture of gaussians

$$p_{GMM}(d) = \sum_{k=1}^{K} a_k \mathcal{G}(d; \mu_k, \Sigma_k) \quad \text{with} \sum_k a_k = 1.$$

$$\text{with: } \mathcal{G}(d; \mu_k, \Sigma_k) = \frac{1}{(2\pi|\Sigma|)^{m/2}} e^{-\frac{1}{2}(d-\mu_k)^t \Sigma_k^{-1} (d-\mu_k)}$$

Learning: Expectation maximization to learn the maximum-likelihood parameters:

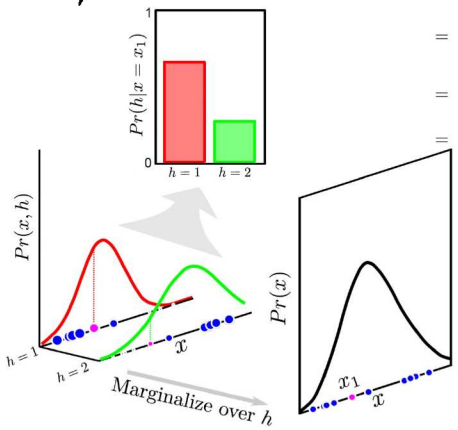
$$\lambda = \{a_k, \mu_k, \Sigma_k\}_k$$

Compare images w.r.t a GMM:

E-step: Compute the probability that the k-th gaussian is responsible for the i-th data point

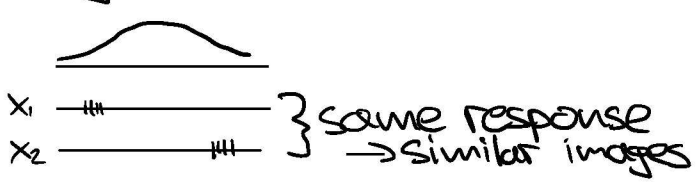
Iterate and average over every data point:

$$\begin{aligned} & Pr(h_i = k | \mathbf{x}_i, \boldsymbol{\theta}^{[t]}) \\ &= \frac{Pr(\mathbf{x}_i | h_i = k, \boldsymbol{\theta}^{[t]}) Pr(h_i = k | \boldsymbol{\theta}^{[t]})}{\sum_{j=1}^{K} Pr(\mathbf{x}_i | h_i = j, \boldsymbol{\theta}^{[t]}) Pr(h_i = j | \boldsymbol{\theta}^{[t]})} \\ &= \frac{\lambda_k \text{Norm}_{\mathbf{x}_i}[\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k]}{\sum_{j=1}^{K} \lambda_j \text{Norm}_{\mathbf{x}_i}[\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j]} \\ &= r_{ik} \end{aligned}$$



Problem:

Symmetries can not be distinguished



Fisher kernels: $k(x, x') := G_\lambda(x)^t F_\lambda^{-1} G_\lambda(x')$

Log-likelihood gradient vector:

$$G_\lambda(x) := \frac{1}{m} \sum_{i=1}^{m} \nabla_\lambda \log p_{GMM}(d_i)$$

$\lambda = \{a_k, \mu_k, \Sigma_k\}_k$

Fisher information matrix:

typical diagonal

$$F_\lambda := \mathbb{E}_{d \sim p_{GMM}(d)} \left\{ [\nabla_\lambda \log p_{GMM}(d)][\nabla_\lambda \log p_{GMM}(d)]^t \right\}$$

Compute the gradients w.r.t. the maximum-likelihood parameters
→ how does the GMM for the input image

Fisher vector:

$$\begin{aligned}
\mathcal{G}^X_{\alpha_k} &= \frac{1}{\sqrt{w_k}} \sum_{t=1}^{T} (\gamma_t(k) - w_k), \\
\mathcal{G}^X_{\mu_k} &= \frac{1}{\sqrt{w_k}} \sum_{t=1}^{T} \gamma_t(k) \left(\frac{x_t - \mu_k}{\sigma_k} \right), \\
\mathcal{G}^X_{\sigma_k} &= \frac{1}{\sqrt{w_k}} \sum_{t=1}^{T} \gamma_t(k) \frac{1}{\sqrt{2}} \left[\frac{(x_t - \mu_k)^2}{\sigma_k^2} - 1 \right]
\end{aligned}$$

→ Derivatives of the maximum likelihood parameters w.r.t the data points X_t
→ explicit feature map of inputs

Derivation:

1. Reparametrize

$$u_\lambda(x) = \sum_{k=1}^{K} w_k u_k(x), \qquad u_\lambda(x) = \sum_{k=1}^{K} w_k u_k(x),$$

$$u_k(x) = \frac{1}{(2\pi)^{D/2} |\Sigma_k|^{1/2}} \exp\left\{ -\frac{1}{2} (x - \mu_k)' \Sigma_k^{-1} (x - \mu_k) \right\}, \longrightarrow u_k(x) = \frac{1}{(2\pi)^{D/2} |\Sigma_k|^{1/2}} \exp\left\{ -\frac{1}{2} (x - \mu_k)' \Sigma_k^{-1} (x - \mu_k) \right\},$$

$$\forall_k : w_k \geq 0, \qquad \sum_{k=1}^{K} w_k = 1, \qquad w_k = \frac{\exp(\alpha_k)}{\sum_{j=1}^{K} \exp(\alpha_j)}$$

→ eliminate constraints by parametrizing w_k through α_k

2. Derivatives:

$$\begin{aligned}
\nabla_{\alpha_k} \log u_\lambda(x_t) &= \gamma_t(k) - w_k, \\
\nabla_{\mu_k} \log u_\lambda(x_t) &= \gamma_t(k) \left(\frac{x_t - \mu_k}{\sigma_k^2} \right), \\
\nabla_{\sigma_k} \log u_\lambda(x_t) &= \gamma_t(k) \left[\frac{(x_t - \mu_k)^2}{\sigma_k^3} - \frac{1}{\sigma_k} \right]
\end{aligned}$$

$$\gamma_t(k) = \frac{w_k u_k(x_t)}{\sum_{j=1}^{K} w_j u_j(x_t)}$$

No geometric information

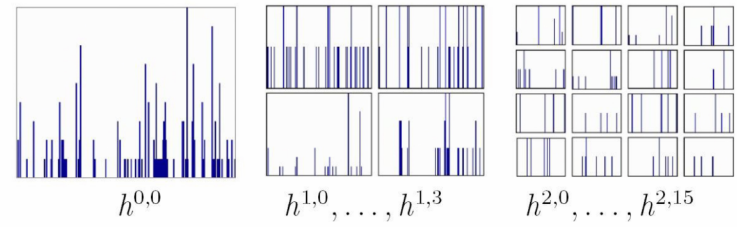
→ Bag of visual words do not incorporate spatial arrangements

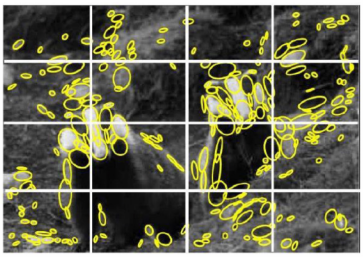
→ important geometrical features are lost

Solution:

Spatial Pyramid features

→ divide the image iteratively into subregions and create a bag of features locally





Combine per-level kernel:

$$k_{pyramid}(h, h') = \frac{1}{2^L} \sum_{l,k} 2^l \, k(h^{l,k}, h'^{l,k})$$

Definition: Approximate explicit feature maps

Explicit embedding may improve efficiency of non-linear SVMs. Since an exact embedding is often unable to obtain, we can approximate the feature map:

Approximate $\varphi : \mathcal{X} \to \mathcal{H}$ by explicit $\tilde{\varphi} : \mathcal{X} \to \mathbb{R}^D$ such that

$$k(x, x') \approx \langle \tilde{\varphi}(x), \tilde{\varphi}(x') \rangle$$

Histogram Intersection:

$$k_{HI}(x, x') = \sum_{i=1}^{d} \min(x_i, x'_i) \qquad \text{for } x = (x_1, \ldots, x_d) \in [0,1]^d$$

Let $\varphi : [0,1] \to \{f : [0,1] \to [0,1]\}$: $\varphi(a) = H_a(x) = [\![x < a]\!]$. Then

$$\min(a, b) = \int_0^1 H_a(x) H_b(x) \, dx$$

Approximate H by finite-dimensional $h : [0,1] \to \mathbb{R}^m$

$$h(a) = \frac{1}{\sqrt{m}} (\underbrace{1, 1, \ldots, 1}_{k}, 0, \ldots, 0) \quad \text{for } \frac{k}{m} \le a < \frac{k+1}{m}$$

Then $k_{HI}(x, x') \approx \sum_{i=1}^{d} \langle h(x_i), h(x'_i) \rangle_{\mathbb{R}^m}$

χ^2 Kernel:

$$k_{\chi^2}(x, x') = \sum_{i=1}^{d} \frac{2 x_i x'_i}{x_i + x'_i} \quad \approx \quad \langle \tilde{\varphi}(x), \tilde{\varphi}(x') \rangle \quad \text{for}$$

$$\tilde{\varphi}(x) := \big(h(x_1), \ldots, h(x_d)\big) \in \mathbb{R}^{3d}, \quad \text{with}$$

$$h(a) := \Big(0.8\sqrt{a},\ 0.6\sqrt{a}\cos(0.6 \log a),\ 0.6\sqrt{a}\sin(0.6 \log a)\Big) \in \mathbb{R}^3$$

Kernel Selection

Properties:

1. Visual similar → high response
   Visual dissimilar → low response
2. Integrate domain knowledge, e.g. invariance
3. If there are classes the kernel matrix should have a block structure

Observations

1. Most pixel-based kernels are too strict
2. To incorporat invariance use an invariant representation and normalize
3. Combination of kernels that focus on different features typically work better than single kernels
4. histogram intersection & χ^2 kernels work best on histograms

Determine kernel parameters

Use k-fold cross validation

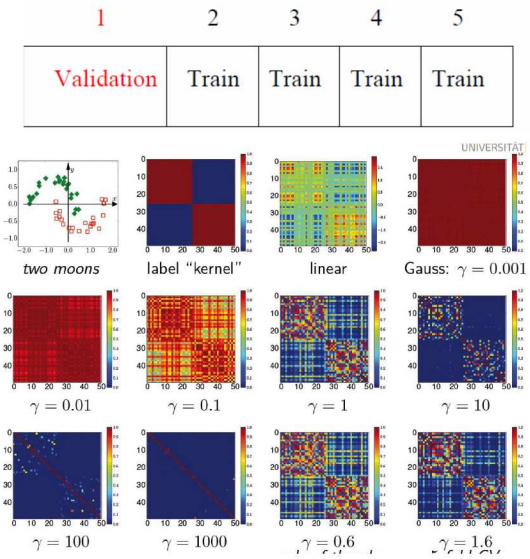
Gaussian kernels:

For general Gaussian kernels:

$$k(x, x') = \exp(-\frac{1}{2\gamma} d^2(x, x'))$$

with any distance d, set

$$\gamma \approx \text{median}_{i,j=1,\dots,n}\, d(x_i, x_j).$$



Combining kernels:

Product: $k = k_1 k_2$

Convex combination: $k = \sum_{j=1}^{K} \beta_j k_j$ with $\beta_j \geq 0$, $\sum_{j=1}^{K} \beta = 1.$

Multiple Kernel Learning:

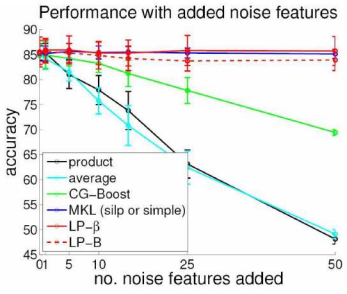
$$\min_{\substack{v_j \in \mathcal{H}_j \\ \sum_j \beta_j = 1 \\ \beta_j \geq 0 \\ \xi_i \in \mathbb{R}^+}} \sum_j \frac{1}{\beta_j} \|v_j\|^2_{\mathcal{H}_j} + C \sum_i \xi_i \quad \text{subject to}$$

$$y_i \sum_j \langle v_j, \varphi_j(x_i) \rangle_{\mathcal{H}_j} \geq 1 - \xi_i \qquad \text{for } i = 1, \dots n.$$

→ learn jointly kernel weights and SVM classifier

Evaluation:

Single features			Combination methods		
Method	Accuracy	Time	Method	Accuracy	Time
Colour	60.9 ± 2.1	3 s	product	85.5 ± 1.2	2 s
Shape	70.2 ± 1.3	4 s	averaging	84.9 ± 1.9	10 s
Texture	63.7 ± 2.7	3 s	CG-Boost	84.8 ± 2.2	1225 s
HOG	58.5 ± 4.5	4 s	MKL (SILP)	85.2 ± 1.5	97 s
HSV	61.3 ± 0.7	3 s	MKL (Simple)	85.2 ± 1.5	152 s
siftint	70.6 ± 1.6	4 s	LP-β	85.5 ± 3.0	80 s
siftbdy	59.4 ± 3.3	5 s	LP-B	85.4 ± 2.4	98 s



Multiclass Classification

Problem: Training samples $\{x_1, \ldots, x_n\} \subset \mathcal{X}$,
Training labels $\{y_1, \ldots, y_n\} \subset \{1, \ldots, M\}$,
Task: learn a prediction function $f : \mathcal{X} \to \{1, \ldots, M\}$.

One-versus-rest SVM

Train a single SVM per class

Classification: $f(x) = \operatorname{argmax}_{m=1,\ldots,M} F_m(x)$

Advantage: easy to implement, works well.
Disadvantage: with many classes, training sets become unbalanced.

All-versus-all SVM

Train a SVM for each pair of classes
$\rightarrow \frac{n(n-1)}{2}$ SVMs

Classify by voting:

$$f(x) = \operatorname*{argmax}_{m=1,\ldots,M} \#\{i \in \{1, \ldots, M\} : F_{m,i}(x) > 0\},$$

(writing $F_{j,i} = -F_{i,j}$ for $j > i$ and $F_{j,j} = 0$)

Advantage: SVM problems stay small and balanced, also works well.
Disadvantage: Number of classifiers grows quadratically in classes.

Support Vector Regression:

$$\min_{\substack{w \in \mathcal{H}, \\ \xi_1,\ldots,\xi_n \in \mathbb{R}^+, \\ \xi'_1,\ldots,\xi'_n \in \mathbb{R}^+}} \quad \|w\|^2 + C \sum_{i=1}^{n} (\xi_i + \xi'_i)$$

subject to

$$\begin{aligned} y_i - \langle w, \varphi(x_i) \rangle &\leq \epsilon + \xi_i, && \text{for } i = 1, \ldots, n, \\ \langle w, \varphi(x_i) \rangle - y_i &\leq \epsilon + \xi'_i, && \text{for } i = 1, \ldots, n. \end{aligned}$$

Dual formulation:

$$\min_{\substack{\alpha_1,\ldots,\alpha_n \in \mathbb{R}, \\ \xi_1,\ldots,\xi_n \in \mathbb{R}^+, \\ \xi'_1,\ldots,\xi'_n \in \mathbb{R}^+}} \quad \sum_{i,j=1}^{n} \alpha_i \alpha_j k(x_i, x_j) + C \sum_{i=1}^{n} (\xi_i + \xi'_i)$$

subject to

$$\begin{aligned} y_i - \textstyle\sum_j \alpha_j k(x_j, x_i) &\leq \epsilon + \xi_i, && \text{for } i = 1, \ldots, n, \\ \textstyle\sum_j \alpha_j k(x_j, x_i) - y_i &\leq \epsilon + \xi'_i, && \text{for } i = 1, \ldots, n. \end{aligned}$$

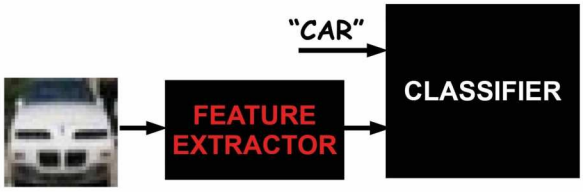
Regression function:

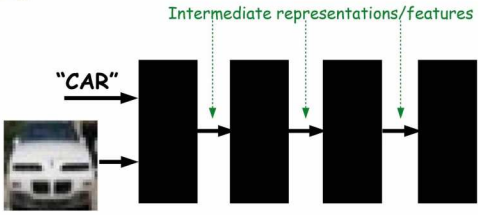
$$f(x) = \langle w, \varphi(x) \rangle = \textstyle\sum_j \alpha_j k(x_j, x)$$

VIII Neural Networks

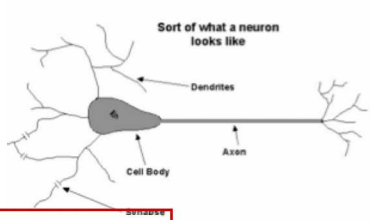
Key ideas:

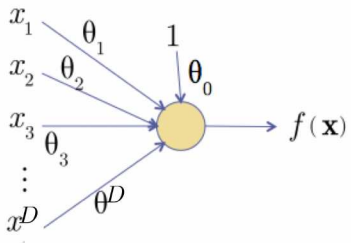
1. Learn features from data
2. Use differentiable functions
3. End-to-end learning:
   no distinction between classifier and feature extractor
4. Deep architecture:
   Concatenation of simpler blocks
5. Feature detection at different levels:
   low level features in early layers
   high level features in later layers





McCullough-Pitts

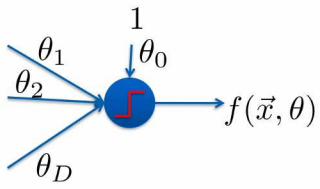




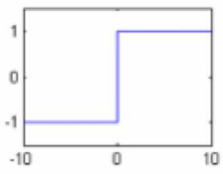
Output:

$$f(x,\theta) = \sum_{d=1}^{D} \theta_d x_d + \theta_0$$

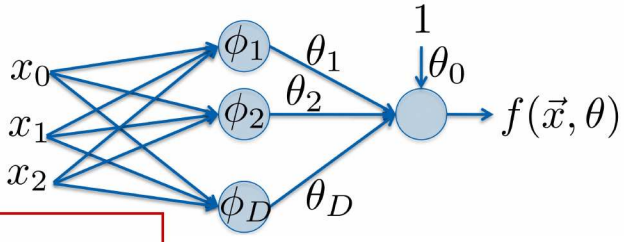
Perceptron:



$$g(z) = \begin{cases} -1 \text{ when } z < 0 \\ +1 \text{ when } z \geq 0 \end{cases}$$



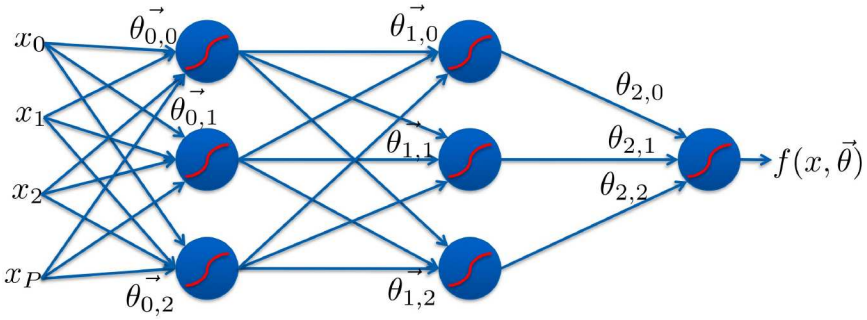
Fully-connected Layer:



Output:

$$f(\vec{x},\theta) = \sum_{n=0}^{N-1} \sum_{d=1}^{D} \theta_d \phi_d(x_n) + \theta_0$$

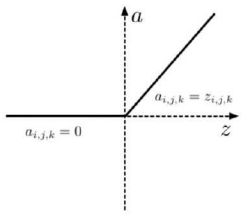
Multilayer Network:



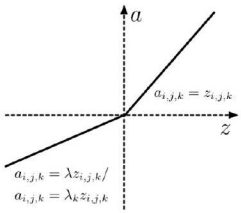
Activation Functions

Linear: $f(\vec{x}, \theta) = \theta^T \vec{x}$

Rectified Linear Unit: $f(y) = \max(0, y)$



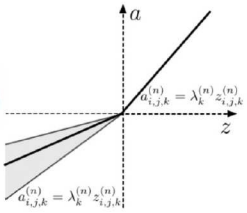
(a) ReLU



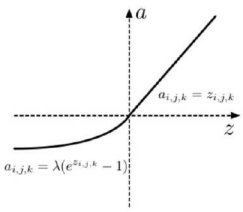
(b) LReLU/PReLU

Gelu: $\text{GELU}(x) = xP(X \leq x) = x\Phi(x)$

Approximation: $0.5x(1 + \tanh[\sqrt{2/\pi}(x + 0.044715x^3)])$

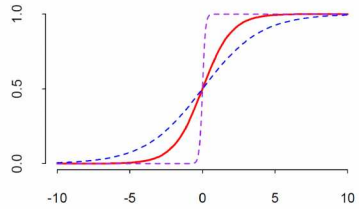


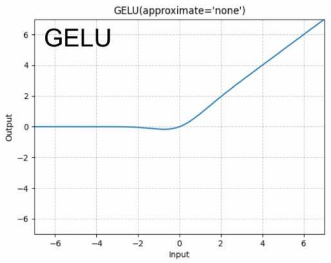
(c) RReLU



(d) ELU

Sigmoid: $g(z) = (1 + exp(-z))^{-1}$





Softmax: $P(y = c|x) = \frac{e^{O_c^{(L)}}}{\sum_{\tilde{c}} e^{O_{\tilde{c}}^{(L)}}}$

→ used to predict probabilities

Loss functions

Empirical Risk:

Cross Entropy: $J = -\frac{1}{N} \sum_n \sum_c Y_{nc} log(P(y = c|x_n))$

→ used for multi-class prediction

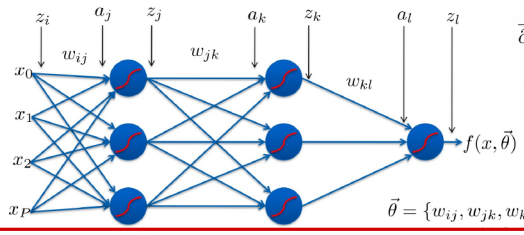
Error Backpropagation:

Backpropagate the error by computing the gradient of the error function w.r.t the weights to update the weights to minimize the error

Model:

$$a_j = \sum_i w_{ij} z_i \qquad a_k = \sum_j w_{jk} z_j \qquad a_l = \sum_k w_{kl} z_k$$

$$z_j = g(a_j) \qquad z_k = g(a_k) \qquad z_l = g(a_l)$$



$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right] = \frac{1}{N}\sum_n [-(y_n - z_{l,n})g'(a_{l,n})]\, z_{k,n} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{k,n}}\right]\left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right] = \frac{1}{N}\sum_n \left[\sum_l \delta_{l,n} w_{kl} g'(a_{k,n})\right] z_{j,n} = \frac{1}{N}\sum_n \delta_{k,n} z_{j,n}$$

$$\frac{\partial R}{\partial w_{ij}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{j,n}}\right]\left[\frac{\partial a_{j,n}}{\partial w_{ij}}\right] = \frac{1}{N}\sum_n \left[\sum_k \delta_{k,n} w_{jk} g'(a_{j,n})\right] z_{i,n} = \frac{1}{N}\sum_n \delta_{j,n} z_{i,n}$$

Error:

$$R(\theta) = \frac{1}{N}\sum_{n=0}^{N} L(y_n - f(x_n)) \qquad \text{Empirical Risk Function}$$

$$= \frac{1}{N}\sum_{n=0}^{N} \frac{1}{2}(y_n - f(x_n))^2$$

$$= \frac{1}{N}\sum_{n=0}^{N} \frac{1}{2}\left(y_n - g\left(\sum_k w_{kl} g\left(\sum_j w_{jk} g\left(\sum_i w_{ij} x_{n,i}\right)\right)\right)\right)^2$$

Error Function:

$${}^{p}E = \frac{1}{2}\sum_{m=1}^{M} ({}^{p}\hat{y}_m - {}^{p}y_m)^2$$

Weight Change:

$$\Delta w_{ij} = \eta \cdot \delta_j \cdot \widetilde{out}_i$$

↘ Delta-rule

Output Neuron:

$$\delta_m = (\hat{y}_m - y_m) \cdot f'(net_m)$$

Hidden Neuron:

$$\delta_h = \left(\sum_{k=1}^{K} \delta_k w_{hk}\right) \cdot f'(net_h)$$

Derivation:

Calculate the gradient w.r.t the error function to minimize the error through gradient descent

Output Neuron:

$$\Delta W_{nm} = -\eta \frac{\partial {}^{p}E(W_{nm})}{\partial W_{nm}} \qquad \text{| Definition}$$

$$\frac{\partial {}^{p}E}{\partial {}^{p}net_m} \cdot \frac{\partial net_m}{\partial W_{nm}} \qquad \text{| Chain rule}$$

$$\frac{\partial {}^{p}E}{\partial {}^{p}y_m} \cdot \frac{\partial f({}^{p}net_m)}{\partial {}^{p}net_m} \qquad \text{| Chain rule}$$

$$\delta_m = ({}^{p}\hat{y}_m - {}^{p}y_m) \cdot f'({}^{p}net_m)$$

$$\Delta W_{nm} = \eta \cdot ({}^{p}\hat{y}_m - {}^{p}y_m) \cdot f'({}^{p}net_m) \cdot {}^{p}\widetilde{out}_m$$

Hidden Neurons:

$$\Delta w_{hm} = -\eta \frac{\partial^p E(w_{hm})}{\partial w_{hm}}$$

$$\frac{\partial^p E(w_{gh})}{\partial^p net_h} \cdot \frac{\partial^p net_h}{\partial w_{gh}} \quad \text{| chain rule}$$

| only input g contributes

$$-{}^p\delta_h = \frac{\partial^p E}{\partial out_h} \cdot \frac{\partial out_h}{\partial^p net_h} \quad \text{| chain rule}$$

$$\frac{\partial f({}^p net_h)}{\partial^p net}$$

$$\sum_{k=1}^{K} \frac{\partial E^p}{\partial net_k} \cdot \frac{\partial net_k}{\partial out_h}$$

$$\frac{\partial \left(\sum_{j=0}^{H} {}^p out_j \, w_{jk}\right)}{\partial w_{jk}}$$

| only output h contributes

$$-\sum_{k=1}^{K} \delta_k \cdot w_{hk}$$

| contributes of h in next layer

$$\delta_h = \left(\sum_{k=1}^{K} \delta_h \cdot w_{hk}\right) \cdot f'({}^p net_h)$$

$$\Delta w_{hm} = \eta \cdot \left(\sum_{k=1}^{K} \delta_h \cdot w_{hk}\right) \cdot f'({}^p net_h) \cdot {}^p out_g$$

Stochastic Gradient Descent

Algorithm to optimize a network by minimizing the error:

while θ_t not converged **do**
- $t \leftarrow t + 1$
- $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
- $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
- $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
- $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
- $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
- $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)

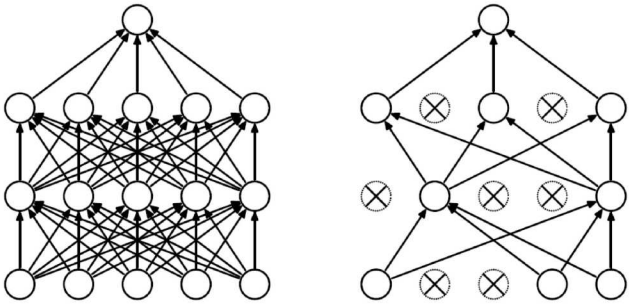
end while

Dropout

Systematically mask out layer outputs to encourage more sparse solutions and prevent overfitting
→ prevent units to rely too much on a single unit.

Training: Mask units
Test: Use scaled weights
Inverted Dropout:

Scale at training but not at test time



(a) Standard Neural Net (b) After applying dropout.

Bayesian Neural Network

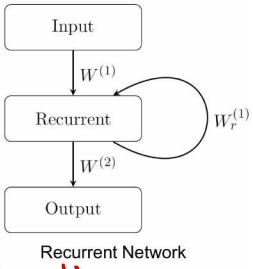
Impose a prior distribution on the weights by adding a term to the error similar to L2 regularization:

$$R(\theta) = \frac{1}{N}\sum_{n=0}^{N} L(y_n - f(x_n)) + \lambda||\theta||^2$$

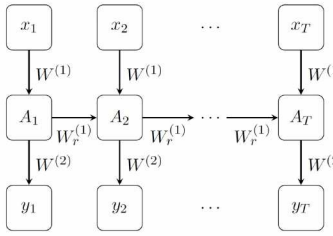
→ Estimation becomes Maximum-A-Posteriori estimation

Recurrent Neural Network

Model output is fed back and used as an additional input



Recurrent Network



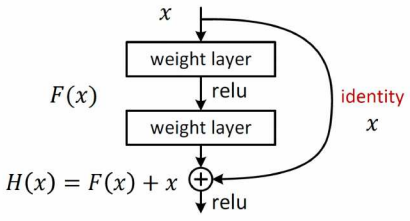
Unfolded Recurrent Network

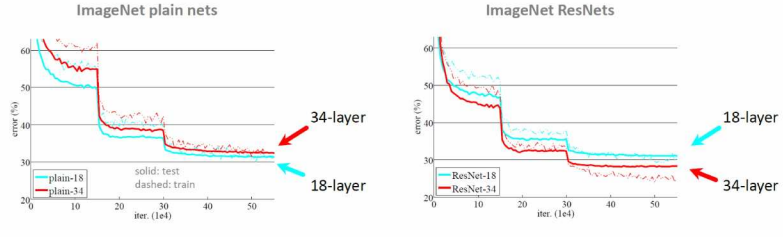
Training: Backpropagation through time
→ Unfold network in time and backpropagate gradients

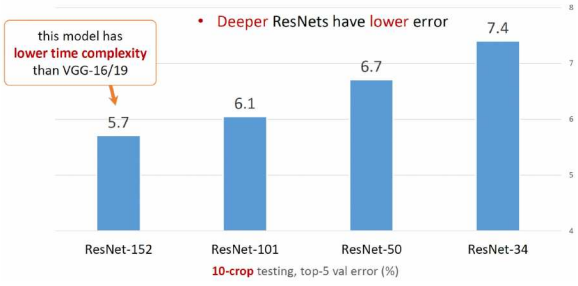
Residual Connections

Add skip layers to let information flow past layers. This also counteracts vanishing gradients which allows the training of larger networks.

→ ResNet first very deep network

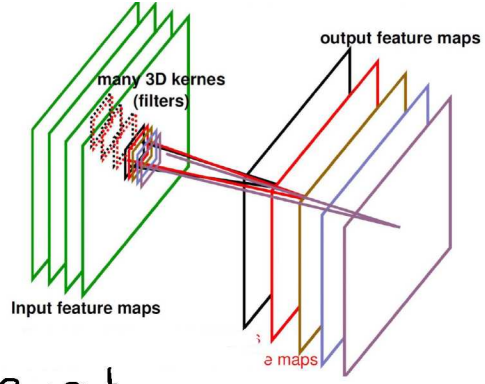






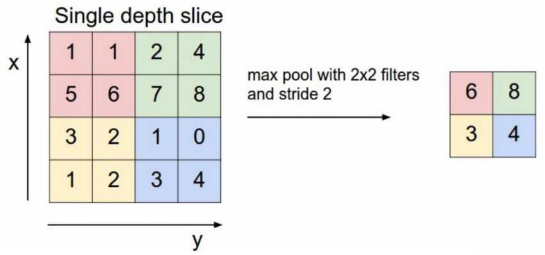
Convolutional Neural Networks

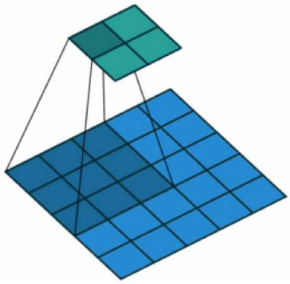
Locally connected neural network. A filter extracts feature from a patch.



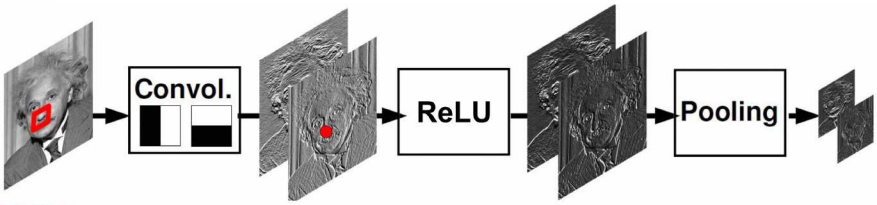
Convolution: $$(I * K)[x, y] = \sum_a \sum_b K[a, b] \cdot I[x - a, y - b]$$

Pooling: Pool layer outputs to implement a spatial invariance to features

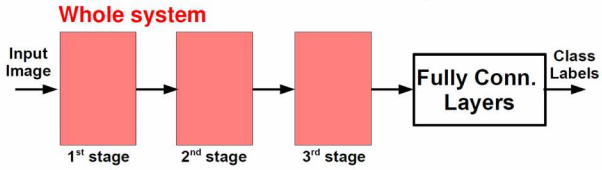




Convolutional Block:



Typical Setup:



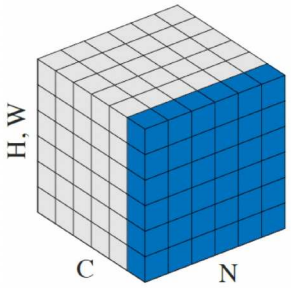
Normalization:

→ Counteract overfitting and improve performance

Batch Normalization:

Normalize channel-wise across a mini-batch

→ parameters can be learned with a higher learning rate



Training:

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

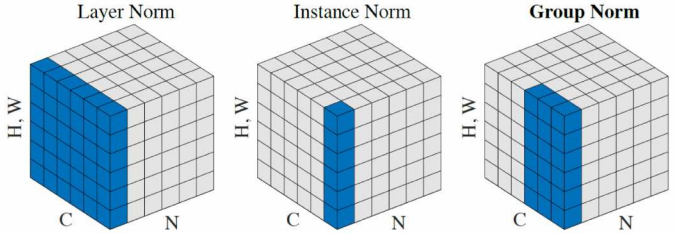
Testing:

6: Train $N^{\mathrm{tr}}_{\mathrm{BN}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: $N^{\mathrm{inf}}_{\mathrm{BN}} \leftarrow N^{\mathrm{tr}}_{\mathrm{BN}}$ // Inference BN network with frozen // parameters
8: **for** $k = 1 \ldots K$ **do**
9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu^{(k)}_{\mathcal{B}}$, etc.
10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma^2_{\mathcal{B}}]$$

11: In $N^{\mathrm{inf}}_{\mathrm{BN}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$
12: **end for**

Other Normalization:

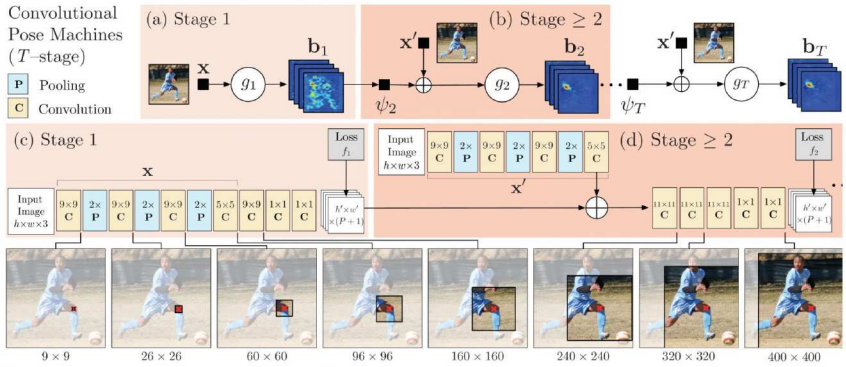


→ Group norm works well for small batch size

IX Neural Network Applications

Pose Estimation (Convolutional Pose Machine)

Approach:



Sequential Prediction:

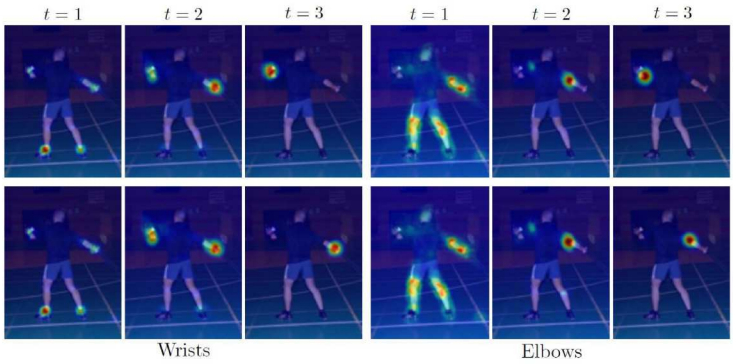
Compute a 2D belief map for each part and iteratively refine it to receive an accurate map

Stage 1: Keypoint Localization

Taking only the image as an input the first stage extracts locally interesting key points

Stage >1:

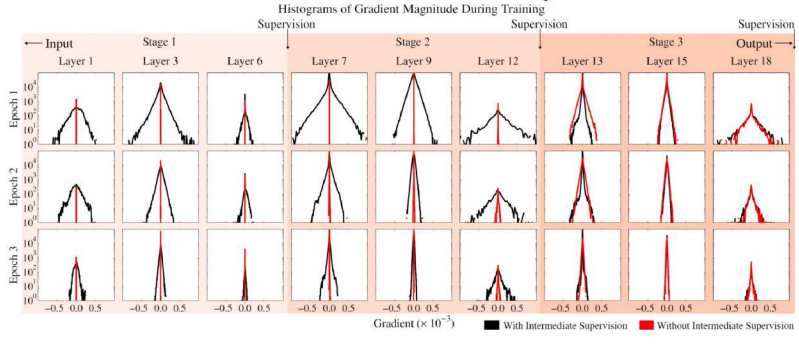
Use the belief features and input to refine the belief map
→ large kernels encourage a larger receptive field
→ spatial context between joints is used



Training: Minimize L2 loss between predicted and optimal belief map at each stage:

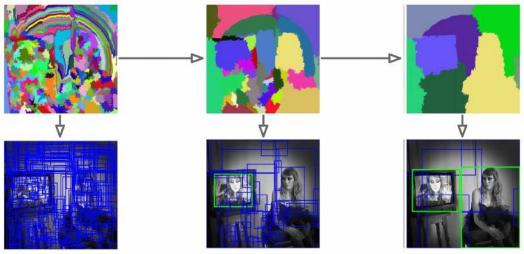
$$f_t = \sum_{p=1}^{P+1} \sum_{z \in Z} \| b_t^p(z) - b_*^p(z) \|_2^2$$

→ counteract vanishing gradients



Object Detection

Regions with CNN (R-CNN)



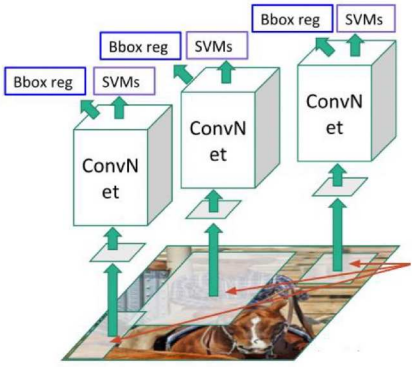
Approach:

1. **Region proposal**

   Use selective search to retrieve 22k region proposal
   → Method is agnostic to the particular region proposal method

2. **Feature extraction**

   Using a large extract a 4096-dimensional feature vector for each region
   → images are warped to a common size



3. **Classification**

   Using a linear SVM, each region is classified.

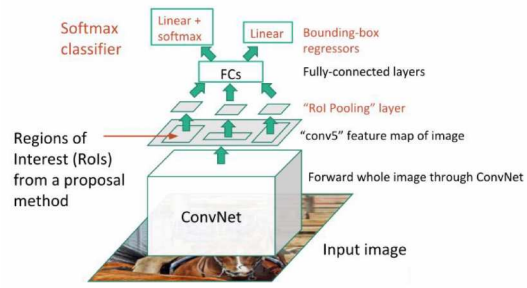
4. **Bounding box refinement**

   With the feature vector the bounding box is refined through a linear regressor.

Problems:

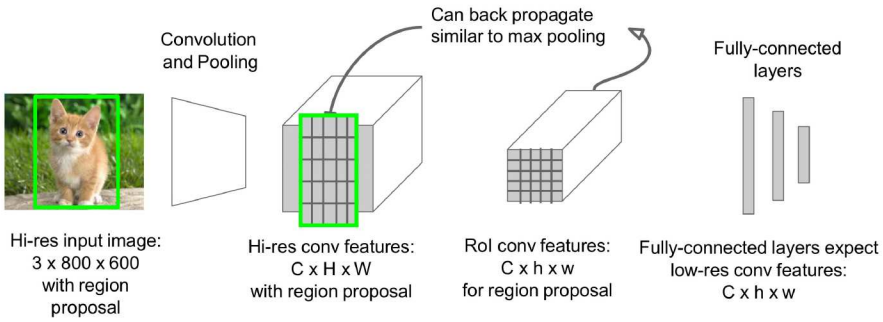
- not trained end-to-end
- slow for inference
- expensive to train

Fast R-CNN



Approach:

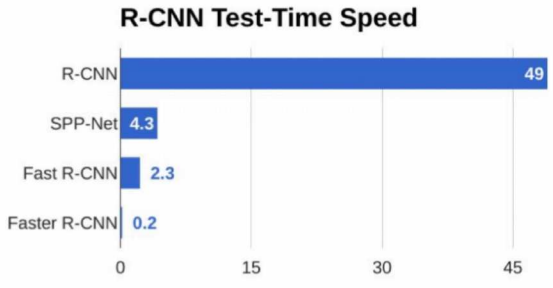
1. Region Proposal
2. Feature extraction
   Extract a feature vector for the complete image
3. Region of Interest Pooling
   For each RoI max-pool within the window to a fixed size



4. Output heads
   The model finally outputs a detection probability for an object class using the softmax function and additionally refines the bounding box in a linear layer

Training: Pre-training + fine tuning
Detection head: Negative log-likelihood
BB Regression: L1 loss

Evaluation:



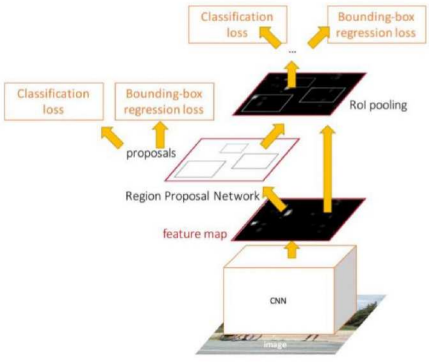
Faster R-CNN

Add a region proposal network to further improve computational efficiency

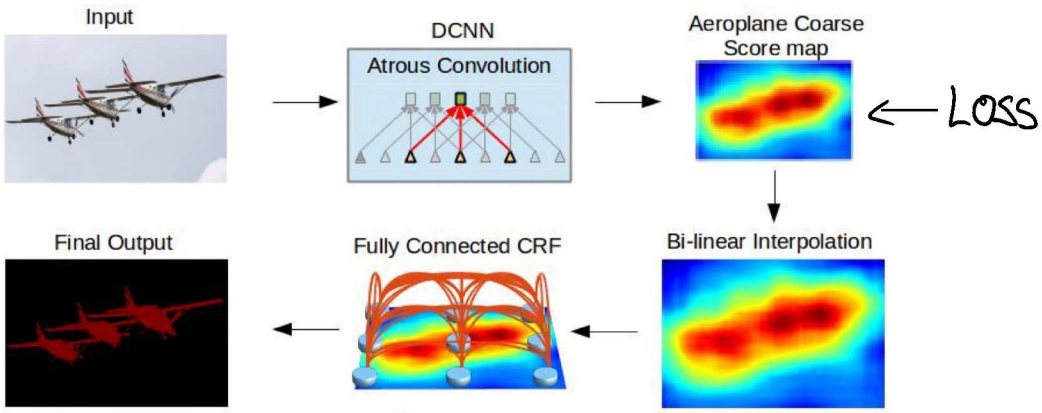
Training:

Jointly train with 4 losses:

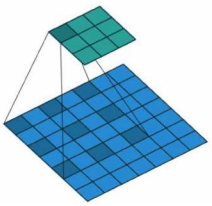
1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates



DeepLab:



Atrous convolution:



Conditional Random Field:

Iteratively refine the segmentation to create a tight fitting segmentation
→ CNNs are not able to draw sharp borders

Model conditional distribution:

$$p(Y = y|X = x) = \frac{1}{Z(x)} \prod_{F \in \mathcal{F}} \psi_F(y_F; x_F)$$

$$Z(x) = \sum_{y \in \mathcal{Y}} \prod_{F \in \mathcal{F}} \psi_F(y_F; x_F$$

Pixels form the random variables

Energy function:

$$E(\mathbf{x}) = \sum_i \psi_u(x_i) + \sum_{i<j} \psi_p(x_i, x_j),$$

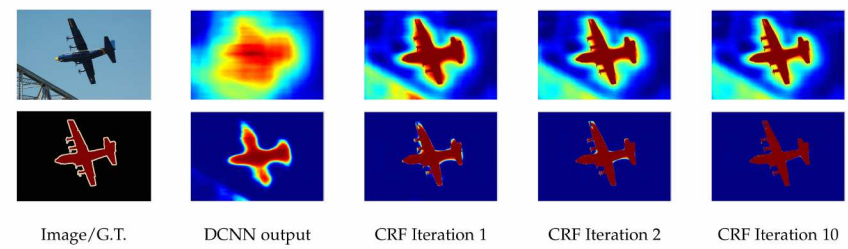
→ to be minimized

Unary potential: Inverse likelihood of a label
→ obtained from the CNN

Pairwise potential:

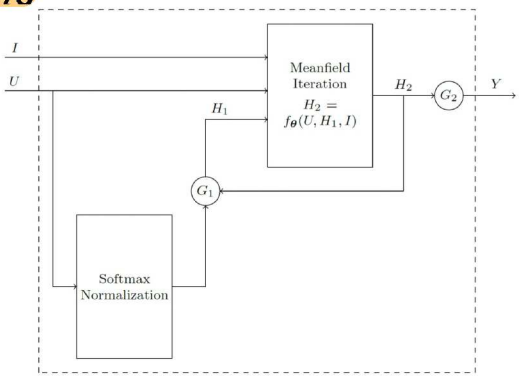
$$\theta_{ij}(x_i, x_j) = \mu(x_i, x_j) \left[w_1 \exp\left(-\frac{||p_i - p_j||^2}{2\sigma_\alpha^2} - \frac{||I_i - I_j||^2}{2\sigma_\beta^2} \right) + w_2 \exp\left(-\frac{||p_i - p_j||^2}{2\sigma_\gamma^2} \right) \right] \quad (3$$

→ bi-linear filtering, gaussian on color and position and solely on position



Conditional Random Fields as RNN

Reformulate the iterative mean-field approximation as a RNN to include it into training



Mean-Field Approximation

→ special case of variational inference

Approx. Distribution: $Q(\theta) = \prod_{i=1}^{D} Q_i(\theta_i).$

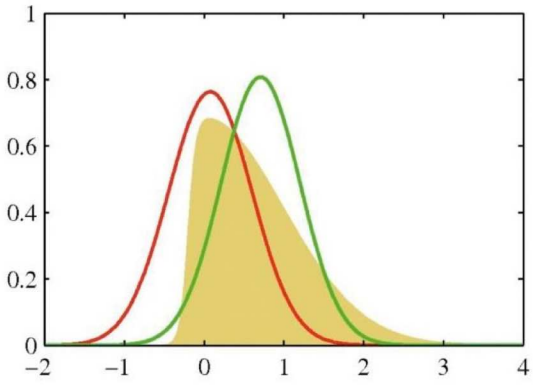
→ fully factorized over the pixels

Variational Lower Bound:

$$\begin{aligned}\mathcal{L}(Q) &= \int Q(\theta) \ln \frac{P(\mathcal{D},\theta)}{Q(\theta)} d\theta = \int Q(\theta) \ln p(\mathcal{D},\theta) d\theta + \int Q(\theta) \ln \frac{1}{Q(\theta)} d\theta \\ &= \int Q_j(\theta_j) \underbrace{\int \ln P(\mathcal{D},\theta) \prod_{i\neq j} Q_i(\theta_i) d\theta_i}_{\mathbb{E}_{i\neq j}[\ln P(\mathcal{D},\theta)]} d\theta_j + \sum_i \int Q_i(\theta_i) \ln \frac{1}{Q_i(\theta_i)} d\theta_i\end{aligned}$$

Compute expectation over pixels:

$$\begin{aligned}\sum_X p(X)\log p(X) &= \sum_X p(X)\log(\prod_i p(x_i)) \\ &= \sum_X p(X)\sum_i \log(p(x_i)) \\ &= \sum_i \sum_X p(X)\log p(x_i) \\ &= \sum_i \sum_X \prod_j p(x_j)\log p(x_i) \\ &= \sum_i \sum_X p(x_i)\log p(x_i)\prod_{j\neq i} p(x_j) \\ &= \sum_i \sum_{x_i} p(x_i)\log p(x_i) \sum_{x_j|j\neq i}\prod_{j\neq i} p(x_j) \\ &= \sum_i \sum_{x_i} p(x_i)\log p(x_i).\end{aligned}$$



Learning:

In each iteration:

1. Fix parameters for all $Q_{i\neq j}$
2. Maximize Q_j w.r.t. the other distributions of the neighboring pixels

General form: $Q_j^*(\theta_j) = \frac{\exp\left(\mathbb{E}_{i\neq j}[\ln P(\mathcal{D},\theta)]\right)}{\int \exp\left(\mathbb{E}_{i\neq j}[\ln P(\mathcal{D},\theta)]\right) d\theta_j}$

→ maximized bound

Variational Inference

Task: Approximate intractable distribution $P(\theta|D)$ with a simple, tractable distribution $Q(\theta)$

Goal: Maximize the variational lower bound
$\Leftrightarrow$ Minimize the KL-divergence
→ minimize the difference between the distributions

Variational Lower Bound:

$$\ln P(\mathcal{D}) \geq \ln P(\mathcal{D}) - \mathrm{KL}(Q(\theta)||P(\theta|\mathcal{D}))$$

with: $\mathrm{KL}(Q||P) = \int Q(x) \ln \frac{Q(x)}{P(x)} dx$

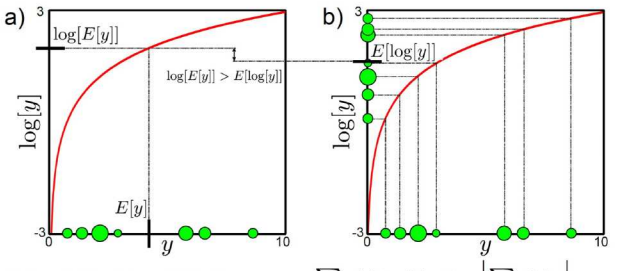
Derivation:

$$\begin{aligned}
\ln P(\mathcal{D}) &= \ln \int P(\mathcal{D}|\theta)P(\theta)d\theta = \ln \int Q(\theta)\frac{P(\mathcal{D},\theta)}{Q(\theta)}d\theta \\
&\geq \int Q(\theta) \ln \frac{P(\mathcal{D},\theta)}{Q(\theta)}d\theta = \int Q(\theta) \ln P(\mathcal{D},\theta)d\theta + \underbrace{\int Q(\theta) \ln \frac{1}{Q(\theta)}d\theta}_{\text{Entropy Functional}} \\
&\qquad \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{Variational Lower Bound}} \\
&= \ln P(\mathcal{D}) - \mathrm{KL}(Q(\theta)||P(\theta|\mathcal{D}))
\end{aligned}$$

Lower Bound for EM:

$$\begin{aligned}
&\int Q(\theta) \log \frac{P(D,\theta)}{Q(\theta)}d\theta \\
&= \int Q(\theta) \log \frac{P(\theta|D)P(D)}{Q(\theta)}d\theta \\
&= \int Q(\theta) \log P(D)d\theta + \int Q(\theta) \log \frac{P(\theta|D)}{Q(\theta)}d\theta \\
&= \log P(D) \int Q(\theta)d\theta - \int Q(\theta) \log \frac{Q(\theta)}{P(\theta|D)}d\theta \\
&= \log P(D) - \int Q(\theta) \log \frac{Q(\theta)}{P(\theta|D)}d\theta \leftarrow \text{KL divergence}
\end{aligned}$$

Jensen Inequality: The expectation of a logarithm is always smaller/equal as the logarithm of the expectation value

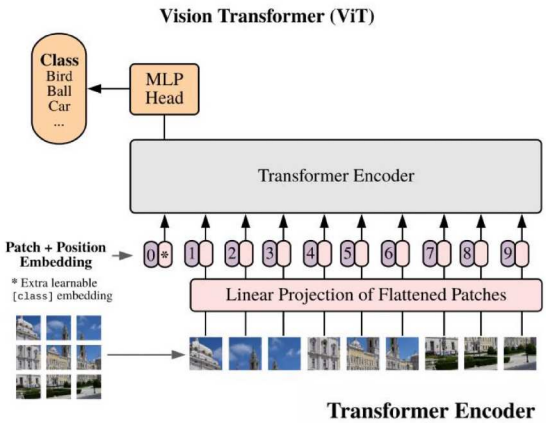


$$E\left[\log[y]\right] \leq \log\left[E[y]\right] \quad \text{or...} \quad \sum_y q(y)\log[y] \leq \log\left[\sum_y q(y)y\right]$$

similarly $\int Pr(y)\log[y]dy \leq \log\left[\int yPr(y)dy\right]$

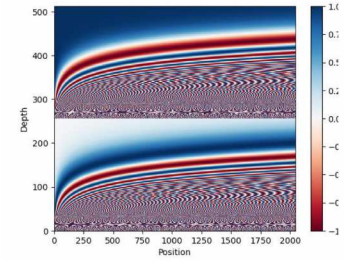
Vision Transformer

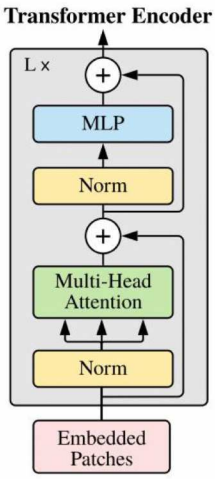
Split the images into patches that can be used as tokens for a transformer architecture



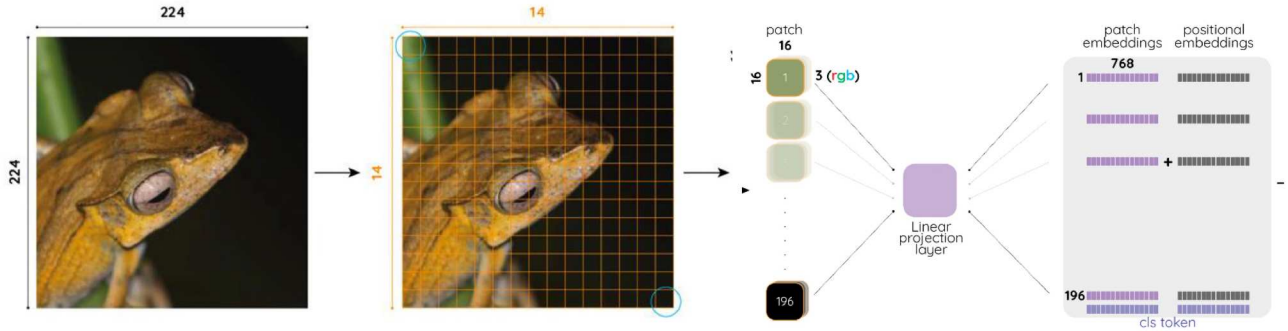
Patch Embedding:

1. Extract 16×16 patches.
2. Embed using convolution
   Convolution: channel out: 768
   kernel size: (16,16)
   stride: (16,16)
3. Encode with positional embedding
   → transformer is invariant to sequence order





$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



Attention Mechanism:

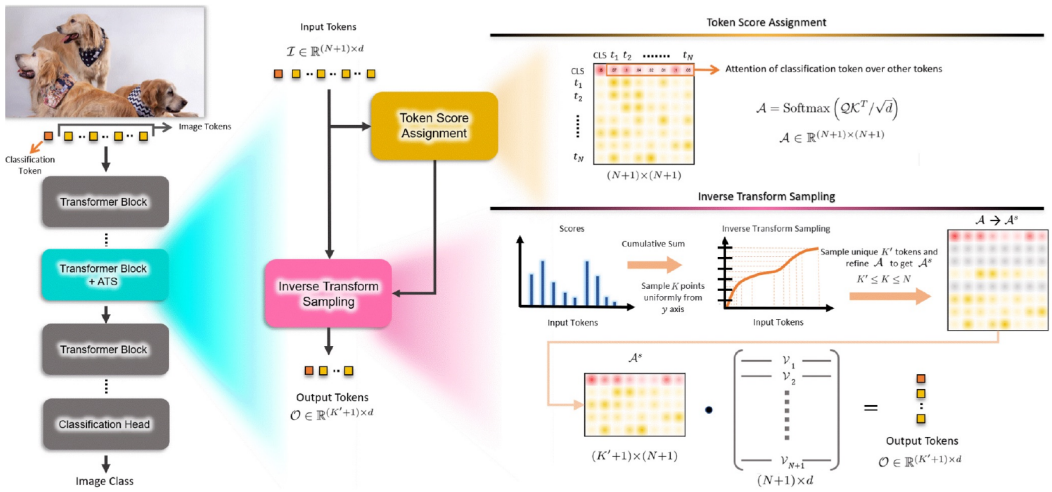
→ scaled dot-product attention

$$\mathcal{Q} \in \mathbb{R}^{(N+1)\times d}$$
$$\mathcal{K} \in \mathbb{R}^{(N+1)\times d}$$
$$\mathcal{V} \in \mathbb{R}^{(N+1)\times d}$$
$$\mathcal{A} = \text{Softmax}\left(\mathcal{Q}\mathcal{K}^T/\sqrt{d}\right)$$
$$\mathcal{A} \in \mathbb{R}^{(N+1)\times(N+1)}$$
$$\mathcal{O} = \mathcal{A}\mathcal{V}$$

Adaptive Token Sampling

Sample only tokens that are important to the inference
→ top-k might disregard important patches
→ attention is distributed across several patches

Approach:



Token Score Assignment

Add an additional classification token that can be attended by the patch queries
→ attention weight indicates importance of the patch for the classification
→ self attention for classification token is removed

Scores:

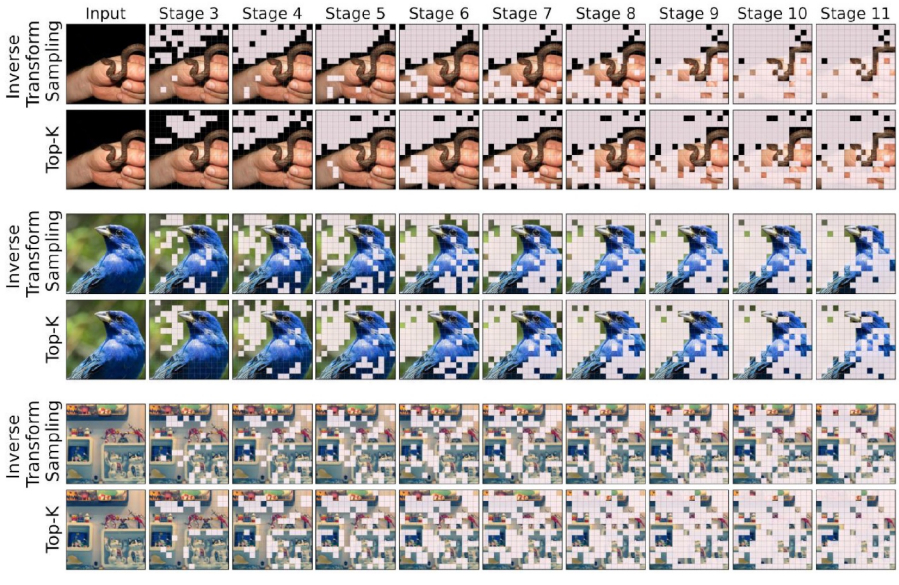
$$\mathcal{S}_j = \frac{\mathcal{A}_{1,j} \times ||\mathcal{V}_j||}{\sum_{i=2} \mathcal{A}_{1,i} \times ||\mathcal{V}_i||}$$

Inverse Transform Sampling

Sample from the above score distribution:

1. Sample K points uniformly from y axis
2. Sub-sample K' unique points
3. Reduce tokens

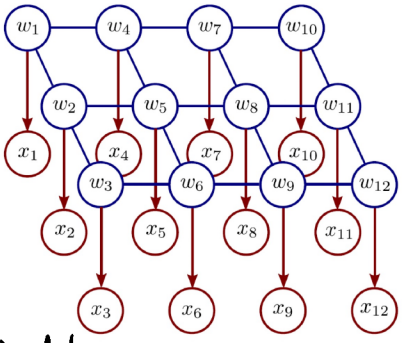
Results:



X Generative Models

Markov Random Fields

Models the joint distribution of random variables as a product of potential functions
→ here the potential functions are defined pair-wise in the neighborhood



Original image, **w**

Observed image, **x**

Components:

Sites: $\mathcal{S} = \{1 \dots N\}$

Random variables: $\{w_n\}_{n=1}^{N}$

Neighbors: $\{\mathcal{N}_n\}_{n=1}^{N}$

Clique: $\mathcal{C}_j \subset \{1, \dots, N\}$
→ set of inter-dependent variables

Potential function: $$Pr(\mathbf{w}) = \frac{1}{Z}\prod_{j=1}^{J}\phi_j[\mathbf{w}_{\mathcal{C}_j}]$$

→ prior on the variables

MAP Inference: Maximize the posterior $Pr(w_{1\dots N}|\mathbf{x}_{1\dots N})$

$$\hat{w}_{1\dots N} = \underset{w_{1\dots N}}{\operatorname{argmin}}\left[\sum_{n=1}^{N} U_n(w_n) + \sum_{(m,n)\in\mathcal{C}} P_{mn}(w_m, w_n)\right]$$

U_n: unary potential at pixel n
→ cost to observe pixel n given w_n
→ negative log-likelihood

P_{mn}: Pair-wise potential
→ cost for placing labels w_m and w_n at neighboring pixel locations

Derivation:

$$\begin{aligned}
\hat{w}_{1\dots N} &= \underset{w_{1\dots N}}{\operatorname{argmax}}\,[Pr(w_{1\dots N}|\mathbf{x}_{1\dots N})] \\
&= \underset{w_{1\dots N}}{\operatorname{argmax}}\left[\prod_{n=1}^{N} Pr(x_n|w_n)Pr(w_{1\dots N})\right] \\
&= \underset{w_{1\dots N}}{\operatorname{argmax}}\left[\sum_{n=1}^{N} \log[Pr(x_n|w_n)] + \log[Pr(w_{1\dots N})]\right] \\
&= \underset{w_{1\dots N}}{\operatorname{argmax}}\left[\sum_{n=1}^{N} \log[Pr(x_n|w_n)] - \sum_{(m,n)\in\mathcal{C}} \psi[w_m, w_n, \boldsymbol{\theta}]\right] \\
&= \underset{w_{1\dots N}}{\operatorname{argmin}}\left[\sum_{n=1}^{N} -\log[Pr(x_n|w_n)] + \sum_{(m,n)\in\mathcal{C}} \psi[w_m, w_n, \boldsymbol{\theta}]\right] \\
&= \underset{w_{1\dots N}}{\operatorname{argmin}}\left[\sum_{n=1}^{N} U_n(w_n) + \sum_{(m,n)\in\mathcal{C}} P_{mn}(w_m, w_n)\right],
\end{aligned}$$

Maximum Likelihood Learning

Let: Binary Pairwise MRF: $P_\theta(\mathbf{x}) = \frac{1}{\mathcal{Z}(\theta)} \exp\left(\sum_{ij \in E} x_i x_j \theta_{ij} + \sum_{i \in V} x_i \theta_i\right)$

Training samples: $\mathcal{D} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\}$

Task: Learn model parameters θ by maximizing the log-likelihood using its derivative

Likelihood: $L(\theta) = \frac{1}{N} \sum_{n=1}^{N} \log P_\theta(\mathbf{x}^{(n)})$

Derivative: $\frac{\partial L(\theta)}{\partial \theta_{ij}} = \frac{1}{N} \sum_n [x_i^{(n)} x_j^{(n)}] - \sum_{\mathbf{x}} [x_i x_j P_\theta(\mathbf{x})] = \mathrm{E}_{P_{data}}[x_i x_j] - \mathrm{E}_{P_\theta}[x_i x_j]$

→ second part can be approximated

exponential many configurations → hard to compute

Derivation:

$$\frac{\partial L(\theta)}{\partial \theta_{ij}} = \frac{\partial}{\partial \theta_{ij}} \frac{1}{N} \sum_{n=1}^{N} \log P_\theta(x^{(n)})$$

$$= \frac{\partial}{\partial \theta_{ij}} \left(\frac{1}{N} \sum_{n=1}^{N} \log\left(\exp\left(\sum_{i,j \in E} x_i x_j \theta_{ij} + \sum_{i \in V} x_i \theta_i \right) \right) - \log(Z(\theta)) \right)$$

$$= \frac{1}{N} \sum_{n=1}^{N} x_i^{(n)} x_j^{(n)} - \frac{\partial}{\partial \theta_{ij}} \log(Z(\theta))$$

$$= \frac{1}{N} \sum_{n=1}^{N} x_i^{(n)} x_j^{(n)} - \frac{1}{Z(\theta)} \frac{\partial}{\partial \theta_{ij}} \sum_{\mathbf{x}} \exp\left(\sum_{i,j \in E} x_i x_j \theta_{ij} + \sum_{i \in V} x_i \theta_i \right)$$

$P_\theta(x)$

$$= \frac{1}{N} \sum_{n=1}^{N} x_i^{(n)} x_j^{(n)} - \sum_{\mathbf{x}} \frac{1}{Z(\theta)} \exp\left(\sum_{i,j \in E} x_i x_j \theta_{ij} + \sum_{i \in V} x_i \theta_i \right) \cdot \frac{\partial}{\partial \theta_{ij}} \left(\sum_{i,j \in E} x_i x_j \theta_{ij} + \sum_{i \in V} x_i \theta_i \right)$$

$$= \frac{1}{N} \sum_{n=1}^{N} x_i^{(n)} x_j^{(n)} - \sum_{\mathbf{x}} x_i x_j P_\theta(\mathbf{x})$$

Denoising with MRFs

Prior: $Pr(w_{1...N}) = \frac{1}{Z} \exp\left[- \sum_{(m,n) \in \mathcal{C}} \psi[w_m, w_n, \boldsymbol{\theta}] \right]$

$\psi[\bullet] = -\log[\phi[\bullet]]$

$\psi[w_m = j, w_n = k, \boldsymbol{\theta}] = \theta_{jk}$

→ encourages smooth labels across neighbors

Likelihood:

$$Pr(x_n | w_n = 0) = \mathrm{Bern}_{x_n}[\rho]$$
$$Pr(x_n | w_n = 1) = \mathrm{Bern}_{x_n}[1 - \rho]$$

ρ: probability that pixel is flipped

Inference: $Pr(w_{1...N} | x_{1...N}) = \frac{\prod_{n=1}^{N} Pr(x_n | w_n) Pr(w_{1...N})}{Pr(x_{1...N})}$

Restricted Boltzmann Machines

Allows tractable mapping between hidden variables and images and vice versa

Components:

Visible variables: $\mathbf{v} \in \{0,1\}^D$

Hidden variables: $\mathbf{h} \in \{0,1\}^F$.

→ fully-connected to visible variables

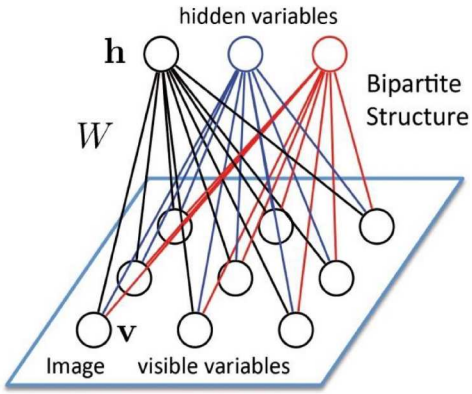
→ no connection in-between

Model parameters: $\theta = \{W, a, b\}$

W: edge weights

a: additional weight for pixels

b: additional parameter for hidden variables



Energy function:

$$E(\mathbf{v}, \mathbf{h}; \theta) = -\sum_{ij} W_{ij} v_i h_j - \sum_i b_i v_i - \sum_j a_j h_j$$

Joint distribution:

$$P_\theta(\mathbf{v}, \mathbf{h}) = \frac{1}{\mathcal{Z}(\theta)} \exp\left(-E(\mathbf{v}, \mathbf{h}; \theta)\right) = \underbrace{\frac{1}{\mathcal{Z}(\theta)}}_{\text{partition function}} \prod_{ij} \underbrace{e^{W_{ij} v_i h_j}}_{\text{potential functions}} \prod_i e^{b_i v_i} \prod_j e^{a_j h_j}$$

$$\mathcal{Z}(\theta) = \sum_{\mathbf{h},\mathbf{v}} \exp\left(-E(\mathbf{v}, \mathbf{h}; \theta)\right)$$

Generative distribution:

$$P_\theta(\mathbf{v}) = \sum_{\mathbf{h}} P_\theta(\mathbf{v}, \mathbf{h}) = \frac{1}{\mathcal{Z}(\theta)} \prod_i \exp(b_i v_i) \prod_j \underbrace{\left(1 + \exp(a_j + \sum_i W_{ij} v_i)\right)}_{\text{Product of experts}}$$

→ hidden variables can be marginalized out

$\prod_i \sum_{h_j} \exp(a_i h_j + \sum_k W_{ik} v_i k_i)$

Derivation:

$$P(\mathbf{v}; \theta) = \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}} \exp\left(\mathbf{v}^\top W \mathbf{h} + \mathbf{b}^\top \mathbf{v} + \mathbf{a}^\top \mathbf{h}\right) \quad \text{| insert energy function}$$

$$= \frac{1}{\mathcal{Z}(\theta)} \exp(\mathbf{b}^\top \mathbf{v}) \prod_{j=1}^{F} \sum_{h_j \in \{0,1\}} \exp\left(a_j h_j + \sum_{i=1}^{D} W_{ij} v_i h_j\right) \quad \text{| explicit form of dot-products}$$

$$= \frac{1}{\mathcal{Z}(\theta)} \exp(\mathbf{b}^\top \mathbf{v}) \prod_{j=1}^{F} \left(1 + \exp\left(a_j + \sum_{i=1}^{D} W_{ij} v_i\right)\right).$$

$\exp(0)$ $h_j = 1$

Goals:

Model P(h|v):

$$P(\mathbf{h}|\mathbf{v}) = \prod_j P(h_j|\mathbf{v}) \quad \text{with:} \quad P(h_j = 1|\mathbf{v}) = \frac{1}{1 + \exp(-\sum_i W_{ij} v_i - a_j)}$$

Derivation:

$$p(h|v) = \frac{p(h,v)}{p(v)} = \frac{\prod_j \exp(a_j h_j + \sum_i W_{ij} v_i h_j)}{\prod_j (1 + \exp(a_j + \sum_i W_{ij} v_i))}$$

$$= \prod_j \frac{1}{\frac{\exp(a_j + \sum_i W_{ij} v_i)}{\exp(a_j h_j + \sum_i W_{ij} v_i h_j)} + \frac{1}{\exp(a_j h_j + \sum_i W_{ij} v_i h_j)}}$$

$$p(h_j = 1|v) = \frac{1}{\frac{\exp(a_j + \sum_i W_{ij} v_i)}{\exp(a_j + \sum_i W_{ij} v_i)} + \frac{1}{\exp(a_j + \sum_i W_{ij} v_i)}}$$

$$= \frac{1}{1 + \exp(-a_j - \sum_i W_{ij} v_i)}$$

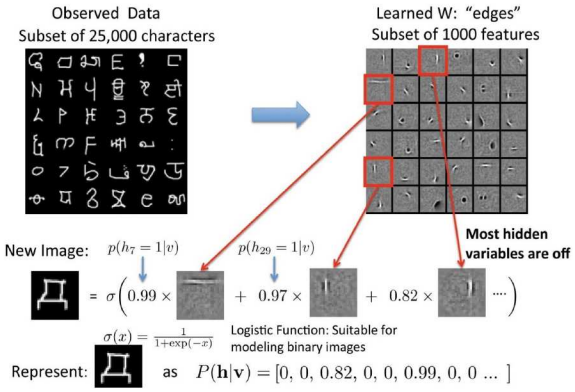
Model P(v|h):

$$P(\mathbf{v}|\mathbf{h}) = \prod_i P(v_i|\mathbf{h}) \quad \text{with:} \quad P(v_i = 1|\mathbf{h}) = \frac{1}{1 + \exp(-\sum_j W_{ij} h_j - b_i)}$$

Interpretation:

Shapes in an image are represented as a linear combination of basis shapes.

The weight matrix of a given hidden variable represents an edge.



Learning: Maximum likelihood

Maximize the (penalized) log-likelihood:

$$L(\theta) = \frac{1}{N}\sum_{n=1}^{N} \log P_\theta(\mathbf{v}^{(n)}) - \frac{\lambda}{N}||W||_F^2$$

Derivative:

$$\frac{\partial L(\theta)}{\partial W_{ij}} = \frac{1}{N}\sum_{n=1}^{N} \frac{\partial}{\partial W_{ij}} \log\left(\sum_{\mathbf{h}} \exp\left[\mathbf{v}^{(n)\top} W\mathbf{h} + \mathbf{a}^\top \mathbf{h} + \mathbf{b}^\top \mathbf{v}^{(n)}\right]\right) - \frac{\partial}{\partial W_{ij}} \log \mathcal{Z}(\theta) - \frac{2\lambda}{N} W_{ij}$$
$$= \mathrm{E}_{P_{data}}[v_i h_j] - \mathrm{E}_{P_\theta}[v_i h_j] - \frac{2\lambda}{N} W_{ij}$$

$$P_{data}(\mathbf{v}, \mathbf{h}; \theta) = P(\mathbf{h}|\mathbf{v}; \theta) P_{data}(\mathbf{v})$$
$$P_{data}(\mathbf{v}) = \frac{1}{N}\sum_{n} \delta(\mathbf{v} - \mathbf{v}^{(n)})$$

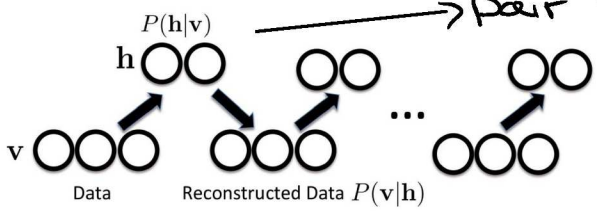
expensive to compute but can be approximated

→ derivation same as for MRFs

Contrastive Divergence:

Approximate maximum likelihood learning
→ similar to Monte Carlo sampling

Start from a given image and iteratively compute images and hidden variables to sample.
→ during training we only require a single iteration.



$$P(\mathbf{h}|\mathbf{v}) = \prod_j P(h_j|\mathbf{v}) \quad P(h_j = 1|\mathbf{v}) = \frac{1}{1 + \exp(-\sum_i W_{ij} v_i - a_j)}$$
$$P(\mathbf{v}|\mathbf{h}) = \prod_i P(v_i|\mathbf{h}) \quad P(v_i = 1|\mathbf{h}) = \frac{1}{1 + \exp(-\sum_j W_{ij} h_j - b_i)}$$

Weight update:

$$\Delta W_{ij} = \mathrm{E}_{P_{data}}[v_i h_j] - \mathrm{E}_{P_1}[v_i h_j]$$

Gaussian-Bernoulli RBM

Generate intensity values for multiple channels
→ image values are modeled as gaussians

Energy function: $$E(\mathbf{v}, \mathbf{h}; \theta) = \sum_i \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{ij} W_{ij} h_j \frac{v_i}{\sigma_i} - \sum_j a_j h_j$$

Model image:
→ gaussian

$$P(v_i = x|\mathbf{h}) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x - b_i - \sigma_i \sum_j W_{ij} h_j)^2}{2\sigma_i^2}\right)$$

Model hidden variable:
→ bernoulli

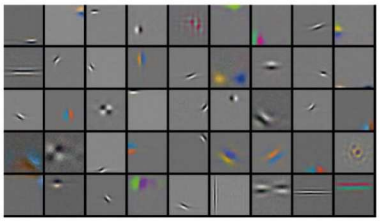
$$P(h_j = 1|\mathbf{v}) = \frac{1}{1 + \exp(-\sum_i W_{ij} \frac{v_i}{\sigma_i} - a_j)}$$

Interpretation: variant of a gaussian mixture model

Images: Gaussian-Bernoulli RBM

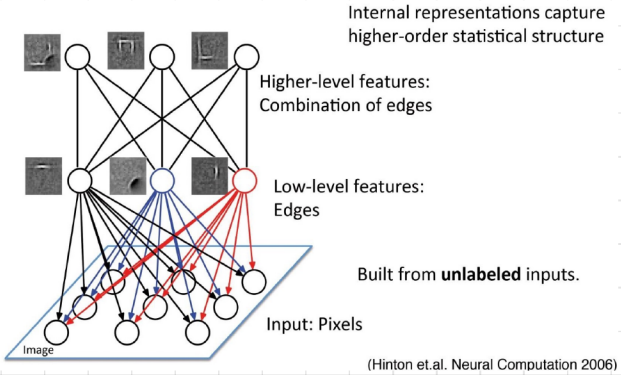
4 million **unlabelled** images

Learned features (out of 10,000)



Deep Belief Networks

RBMs are limited in their capacity. DBNs add additional layers on top to capture more complex relations



(Hinton et.al. Neural Computation 2006)

Architecture:

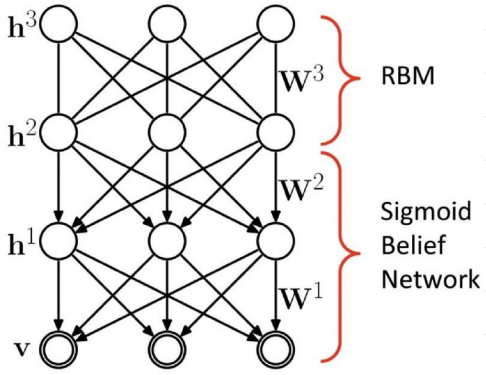
highest layer: undirected edges

other layers: directed edges

→ allows efficient factorization and computation

Joint distribution:

$$P(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3) = P(\mathbf{v}|\mathbf{h}^1)P(\mathbf{h}^1|\mathbf{h}^2)P(\mathbf{h}^2, \mathbf{h}^3)$$



Training: Layer-wise training

Train each layer as a RBM

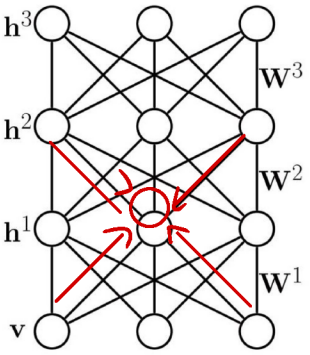
→ freeze layer afterwards

Deep Boltzmann Machine

Similar to DBN but with undirected edges

Generative distribution:

$$P_\theta(\mathbf{v}) = \frac{P^*(\mathbf{v})}{\mathcal{Z}(\theta)} = \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3} \exp\left[\mathbf{v}^\top W^1 \mathbf{h}^1 + \mathbf{h}^{1\top} W^2 \mathbf{h}^2 + \mathbf{h}^{2\top} W^3 \mathbf{h}^3\right]$$



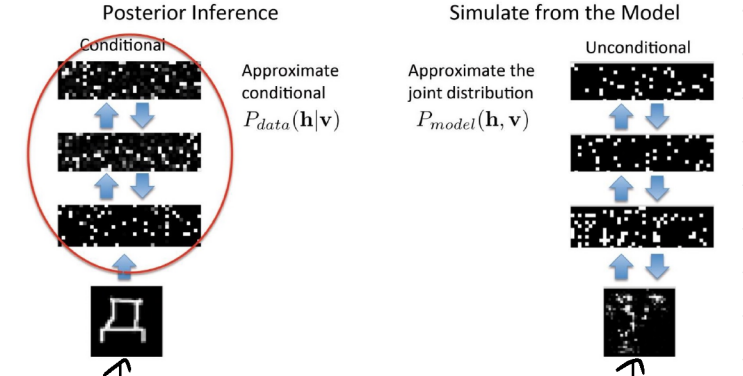
Problem: Intractable to compute hidden variables

→ not independent of later variables

$$\frac{\partial \log P_\theta(\mathbf{v})}{\partial W^1} = \mathrm{E}_{P_{data}}[\mathbf{v}\mathbf{h}^{1\top}] - \mathrm{E}_{P_\theta}[\mathbf{v}\mathbf{h}^{1\top}]$$

now both are intractable

Approximation:



Mean-field approximation

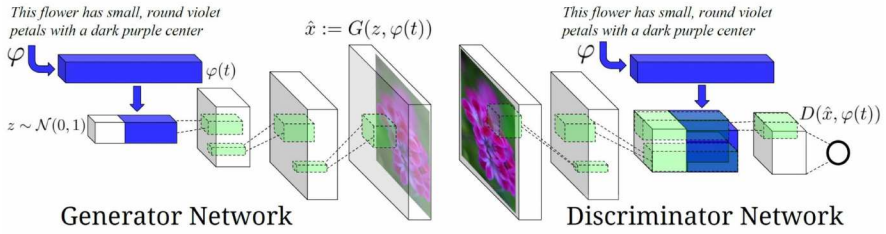
Markov chain approximation

Generative Adversarial Network (GAN)

Architecture:

Generator Network $g[z_j, \theta]$
- Generates (fake) new data
- fed with noise and additional condition ($N(0,I)$)

Discriminator Network $f[\mathbf{x}, \boldsymbol{\phi}]$
- Decides whether the data is real or not

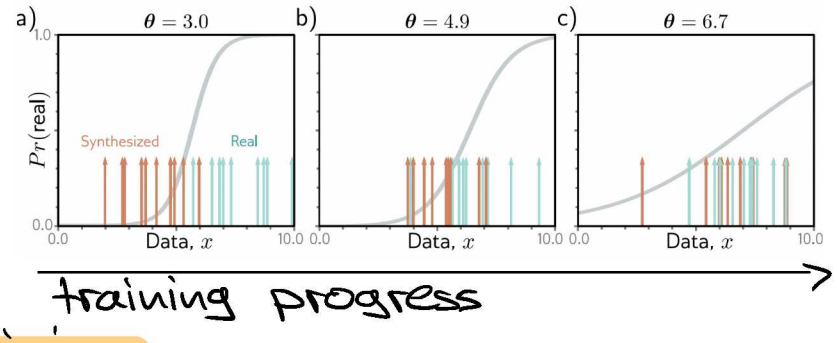


Training principle:

The two models are trained alternatively

Generator: Generate more realistic images to make the discriminator less confident

Discriminator: Learn to distinguish between synthetic and real images



Training:

Discriminator Loss: Binary Cross-Entropy

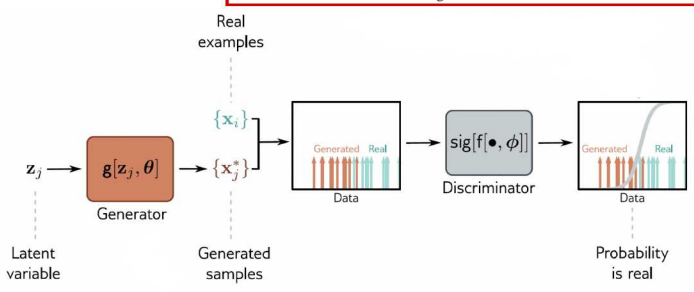
$$L[\boldsymbol{\phi}] = \underbrace{\sum_j -\log\Big[1 - \text{sig}[f[\mathbf{g}[\mathbf{z}_j, \boldsymbol{\theta}], \boldsymbol{\phi}]]\Big]}_{\text{loss synthetic}} - \underbrace{\sum_i \log\Big[\text{sig}[f[\mathbf{x}_i, \boldsymbol{\phi}]]\Big]}_{\text{loss real}}$$

real samples = 1, synthetic samples = 0

Generator Loss:

$$L[\boldsymbol{\theta}] = \sum_j \log\Big[1 - \text{sig}[f[\mathbf{g}[\mathbf{z}_j, \boldsymbol{\theta}], \boldsymbol{\phi}]]\Big]$$

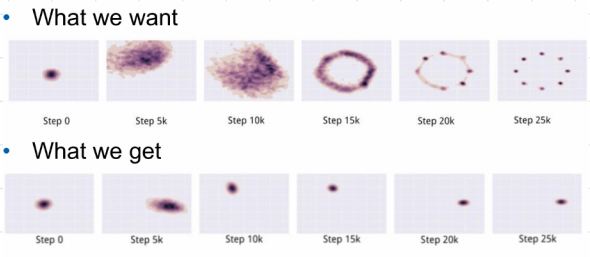
Pipeline:



Problems:

1. Mode collapse

The generator learns to generate a small subset very realistically



Theoretical Analysis:

The mode loss is a direct consequence of the loss function:

1. Reformulate the loss function

Assumption:

Pr(x*): distribution over generated samples
Pr(x): distribution over training data are known.

$$
\begin{aligned}
L[\phi] &= \frac{1}{J}\sum_{j=1}^{J}\Big(\log\Big(1-\text{sig}[\text{f}[\mathbf{x}_j^*,\phi]]\Big)\Big) + \frac{1}{I}\sum_{i=1}^{I}\Big(\log\Big[\text{sig}[\text{f}[\mathbf{x}_i,\phi]]\Big]\Big) \\
&\approx \mathbb{E}_{\mathbf{x}^*}\Big[\log\Big[1-\text{sig}[\text{f}[\mathbf{x}^*,\phi]]\Big]\Big] + \mathbb{E}_{\mathbf{x}}\Big[\log\Big[\text{sig}[\text{f}[\mathbf{x},\phi]]\Big]\Big] \\
&= \int Pr(\mathbf{x}^*)\log\Big[1-\text{sig}[\text{f}[\mathbf{x}^*,\phi]]\Big]d\mathbf{x}^* + \int Pr(\mathbf{x})\log\Big[\text{sig}[\text{f}[\mathbf{x},\phi]]\Big]d\mathbf{x}
\end{aligned}
$$

for $\to\infty$ | Rewrite expectation

2. Define an optimal discriminator

$$
Pr(\mathbf{x}\text{ is real}) = \text{sig}\big[\text{f}[\mathbf{x},\phi]\big] = \frac{Pr(\mathbf{x})}{Pr(\mathbf{x}^*)+Pr(\mathbf{x})}
$$

Insert:

$$
\begin{aligned}
L[\phi] &= \int Pr(\mathbf{x}^*)\log\left[1-\frac{Pr(\mathbf{x})}{Pr(\mathbf{x}^*)+Pr(\mathbf{x})}\right]d\mathbf{x}^* + \int Pr(\mathbf{x})\log\left[\frac{Pr(\mathbf{x})}{Pr(\mathbf{x}^*)+Pr(\mathbf{x})}\right]d\mathbf{x} \\
&= \int Pr(\mathbf{x}^*)\log\left[\frac{Pr(\mathbf{x}^*)}{Pr(\mathbf{x}^*)+Pr(\mathbf{x})}\right]d\mathbf{x}^* + \int Pr(\mathbf{x})\log\left[\frac{Pr(\mathbf{x})}{Pr(\mathbf{x}^*)+Pr(\mathbf{x})}\right]d\mathbf{x}.
\end{aligned}
$$

3. Comparison to Jenson-Shannon divergence

$$
\begin{aligned}
&D_{JS}\Big[Pr(\mathbf{x}^*)\,||\,Pr(\mathbf{x})\Big] \\
&= \frac{1}{2}D_{KL}\left[Pr(\mathbf{x}^*)\middle\|\frac{Pr(\mathbf{x}^*)+Pr(\mathbf{x})}{2}\right] + \frac{1}{2}D_{KL}\left[Pr(\mathbf{x})\middle\|\frac{Pr(\mathbf{x}^*)+Pr(\mathbf{x})}{2}\right] \\
&= \frac{1}{2}\underbrace{\int Pr(\mathbf{x}^*)\log\left[\frac{2Pr(\mathbf{x}^*)}{Pr(\mathbf{x}^*)+Pr(\mathbf{x})}\right]d\mathbf{x}}_{\text{quality}} + \frac{1}{2}\underbrace{\int Pr(\mathbf{x})\log\left[\frac{2Pr(\mathbf{x})}{Pr(\mathbf{x}^*)+Pr(\mathbf{x})}\right]d\mathbf{x}}_{\text{coverage}}.
\end{aligned}
$$

"for every synthetic image how close is a real image"

"for every real image, how close is a synthetic image"

⇒ the generator loss only employs the quality term

2. Vanishing gradients in the generator
If images are too easy/hard to classify, gradients vanish and make learning impossible

Wasserstein GAN

Idea: Limit the learning abilities of the discriminator, such that it does not get too strong too fast to prevent vanishing gradients

Method: Enforce that the the discriminator is Lipschitz continuous:

$$|D(x_2) - D(x_1)| \leq |x_2 - x_1|$$

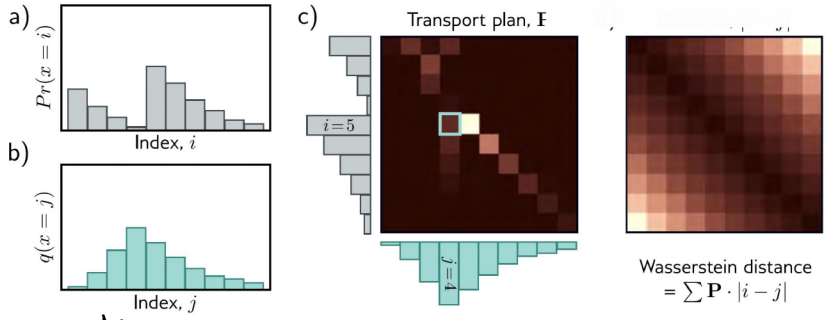
Regularization: Lipschitz penalty

$$R_{LP} = \mathbb{E}_{(\hat{x},h)\sim\mathbb{P}_{\hat{x},h}}[\max(0, \|\nabla_{\hat{x},h} D(\hat{x},h)\|_2 - 1)^2]$$

→ enforces the gradient norm to be <1
→ limits learning "speed"

Wasserstein distance: $D_w\left[Pr(x)||q(x)\right] = \min_{\mathbf{P}}\left[\sum_{i,j} P_{ij}|i-j|\right]$ such that:

$$\begin{aligned} \textstyle\sum_j P_{ij} &= Pr(x=i) \\ \textstyle\sum_i P_{ij} &= q(x=j) \\ P_{ij} &\geq 0 \end{aligned}$$



Intuition:
Transport plan defines how much probability mass is moved from i to j

Solution:

Primal problem:

$$D_w\left[Pr(\mathbf{x}), q(\mathbf{x}^*)\right] = \min_{\pi[\bullet,\bullet]} \int\int \pi(\mathbf{x}_1, \mathbf{x}_2)||\mathbf{x}_1 - \mathbf{x}_2|| d\mathbf{x}_1 d\mathbf{x}_2$$

Dual problem:

$$D_w\left[Pr(\mathbf{x}), q(\mathbf{x}^*)\right] = \max_{\mathrm{f}[\mathbf{x}]} \left[\int Pr(\mathbf{x})\mathrm{f}[\mathbf{x}]d\mathbf{x} - \int Pr(\mathbf{x}^*)\mathrm{f}[\mathbf{x}]d\mathbf{x}\right]$$

subject to: $||\mathrm{f}[z_1] - \mathrm{f}[z_2]|| \leq \beta||z_1 - z_2||$

$\beta=1$: lipschitz conti

Application in GAN

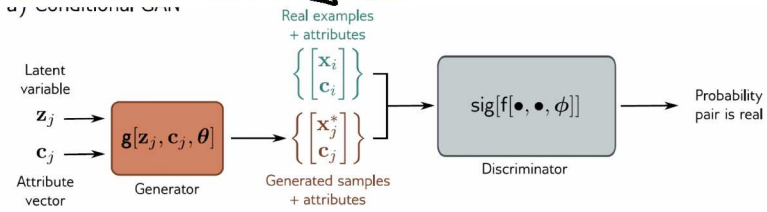
a) Generator distribution
b) "optimal" training data distribution

Then:
Solving Wasserstein distance
⇔ Training goal of a Gan
→ Approximate generative distribution

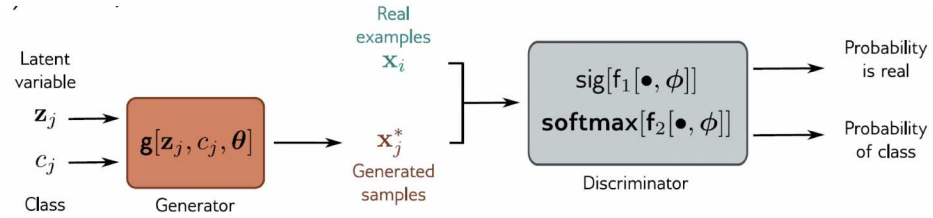
Conditional GANs

Input an additional condition to generate images based on a class.

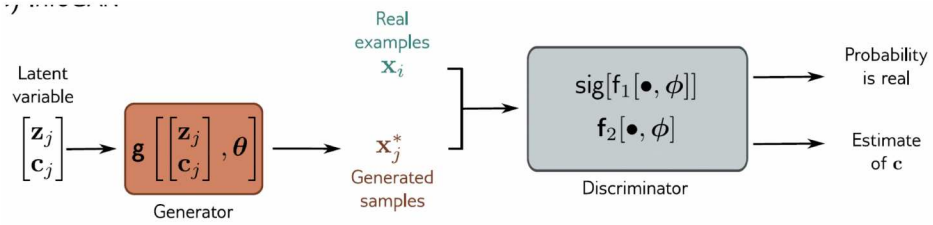
Conditional GAN:



Auxiliary Classifier GAN:

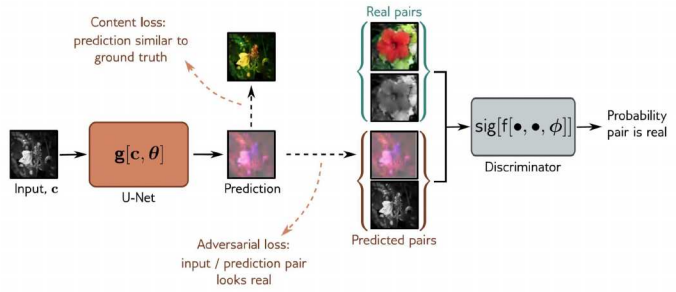


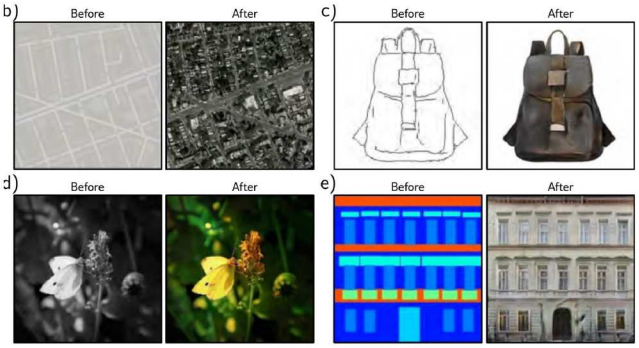
InfoGAN:



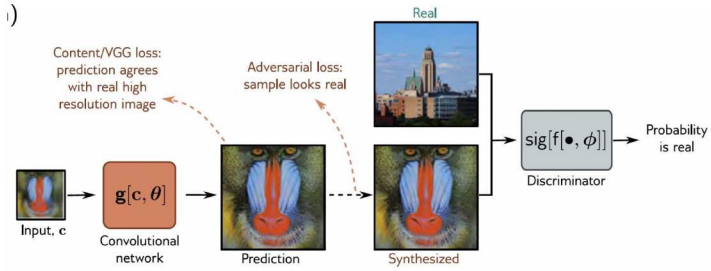
Applications:

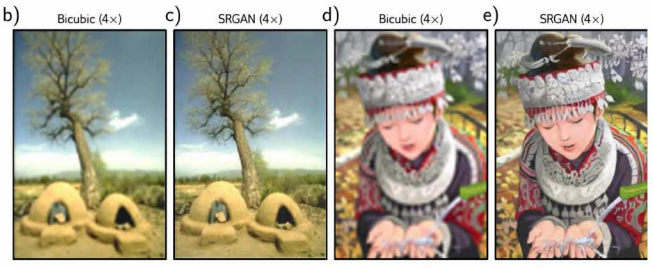
Image Translation:



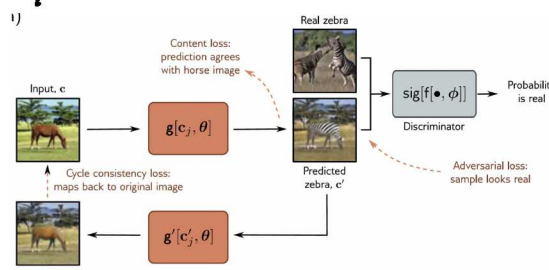


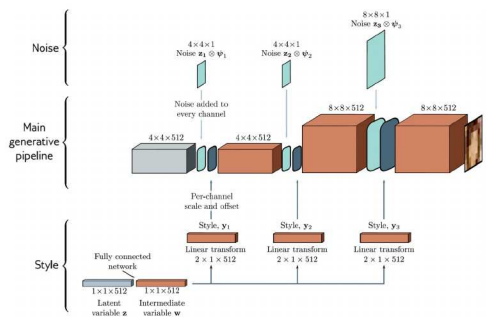
Super-Resolution:



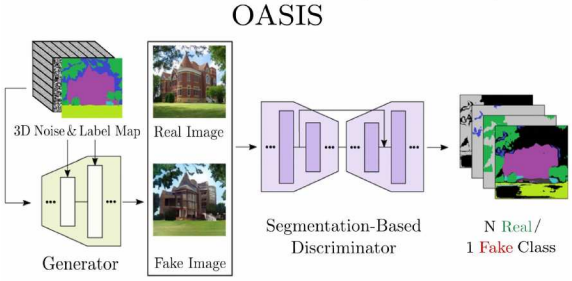


Style Transfer:

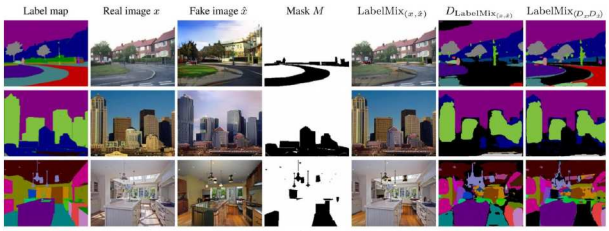




Semantic Image Synthesis:



Label mix: $M \odot x + (1 - M) \odot \hat{x}$

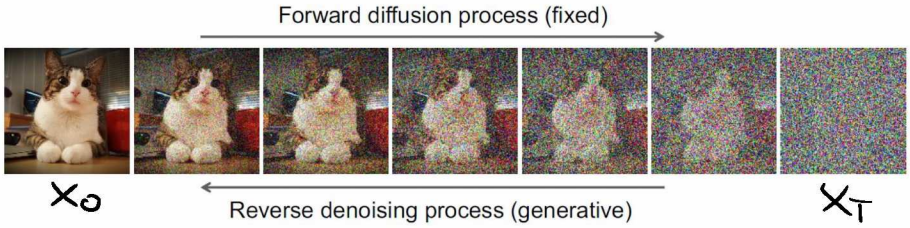


$$\mathcal{L}_{cons} = \Big\| D_{\text{logits}}\big(\text{LabelMix}(x, \hat{x}, M)\big) - \text{LabelMix}\big(D_{\text{logits}}(x), D_{\text{logits}}(\hat{x}), M\big)\Big\|^2$$

[V. Suhsko et al. **OASIS: Only Adversarial Supervision for Semantic Image Synthesis.** IJCV 2022]

Diffusion Models

Idea: If we gradually add noise to an image, we can reverse this process and iteratively denoise the image



Forward process: Gradually add noise

In each iteration: $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \Rightarrow q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$

Closed-form: $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s) \Rightarrow q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}))$

β_t is designed, such that: $\bar{\alpha}_T \to 0$ and $q(\mathbf{x}_T|\mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$

Sampling: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\,\epsilon$

Generative Process:

1. **Sample initial noise:** $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$
2. **Iteratively remove noise:** $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \sigma_t^2\mathbf{I})$
   → the network learns to predict the noise to be removed in each step

   learned by the network

$\Rightarrow$ $p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$

Problem:

marginalization over all images

$q(\mathbf{x}_{t-1}|\mathbf{x}_t) \propto q(\mathbf{x}_{t-1})q(\mathbf{x}_t|\mathbf{x}_{t-1})$ is in general intractable

$\Rightarrow$ Assuming β_t is small, we can approximate with a normal distribution

$\Rightarrow$ Requires many steps $\sim$1000

Training: The models are trained by minimizing the evidence lower bound. This is simplified to get the final loss below.

Loss:

$$L[\phi] = \sum_{i=1}^{I}\sum_{t=1}^{T}\left\|\mathbf{g}_t[\mathbf{z}_{it},\phi_t]-\epsilon_{it}\right\|^2$$
$$= \sum_{i=1}^{I}\sum_{t=1}^{T}\left\|\mathbf{g}_t\left[\sqrt{\alpha_t}\cdot\mathbf{x}_i+\sqrt{1-\alpha_t}\cdot\epsilon_{it},\phi_t\right]-\epsilon_{it}\right\|^2$$

Algorithm:

Algorithm 18.1: Diffusion model training

Input: Training data $\mathbf{x}$
Output: Model parameters ϕ_t
repeat
  for $i \in \mathcal{B}$ **do** // For every training example index in batch
    $t \sim \text{Uniform}[1,\ldots T]$ // Sample random timestep
    $\epsilon \sim \text{Norm}[\mathbf{0},\mathbf{I}]$ // Sample noise
    $\ell_i = \left\|\mathbf{g}_t\left[\sqrt{\alpha_t}\mathbf{x}_i+\sqrt{1-\alpha_t}\epsilon,\phi_t\right]-\epsilon\right\|^2$ // Compute individual loss
  Accumulate losses for batch and take gradient step
until converged

Derivation:

1. **Start with maximizing the log-likelihood**

1.1 **Maximize likelihood:** $Pr(\mathbf{x},\mathbf{z}_{1\ldots T}|\phi_{1\ldots T}) = Pr(\mathbf{x}|\mathbf{z}_1,\phi_1)\prod_{t=2}^{T}Pr(\mathbf{z}_{t-1}|\mathbf{z}_t,\phi_t)\cdot Pr(\mathbf{z}_T)$

1.2 **Marginalize over latent variables:** $Pr(\mathbf{x}|\phi_{1\ldots T}) = \int Pr(\mathbf{x},\mathbf{z}_{1\ldots T}|\phi_{1\ldots T})d\mathbf{z}_{1\ldots T}$

1.3 **Apply log-trick:** $\hat{\phi}_{1\ldots T} = \underset{\phi_{1\ldots T}}{\text{argmax}}\left[\sum_{i=1}^{I}\left[\log\left[Pr(\mathbf{x}_i|\phi_{1\ldots T})\right]\right]\right]$

2. **Define lower bound through Jensen's inequality**

$$\log\left[Pr(\mathbf{x}|\phi_{1\ldots T})\right] = \log\left[\int Pr(\mathbf{x},\mathbf{z}_{1\ldots T}|\phi_{1\ldots T})d\mathbf{z}_{1\ldots T}\right]$$
$$= \log\left[\int q(\mathbf{z}_{1\ldots T}|\mathbf{x})\frac{Pr(\mathbf{x},\mathbf{z}_{1\ldots T}|\phi_{1\ldots T})}{q(\mathbf{z}_{1\ldots T}|\mathbf{x})}d\mathbf{z}_{1\ldots T}\right]$$
$$\geq \int q(\mathbf{z}_{1\ldots T}|\mathbf{x})\log\left[\frac{Pr(\mathbf{x},\mathbf{z}_{1\ldots T}|\phi_{1\ldots T})}{q(\mathbf{z}_{1\ldots T}|\mathbf{x})}\right]d\mathbf{z}_{1\ldots T} \leftarrow \text{ELBO}$$

3. **Simplify the ELBO**

3.1 **Factorize forward & backward process**

$$\log\left[\frac{Pr(\mathbf{x},\mathbf{z}_{1\ldots T}|\phi_{1\ldots T})}{q(\mathbf{z}_{1\ldots T}|\mathbf{x})}\right] = \log\left[\frac{Pr(\mathbf{x}|\mathbf{z}_1,\phi_1)\prod_{t=2}^{T}Pr(\mathbf{z}_{t-1}|\mathbf{z}_t,\phi_t)\cdot Pr(\mathbf{z}_T)}{q(\mathbf{z}_1|\mathbf{x})\prod_{t=2}^{T}q(\mathbf{z}_t|\mathbf{z}_{t-1})}\right] \leftarrow \text{reverse} \leftarrow \text{forward}$$
$$= \log\left[\frac{Pr(\mathbf{x}|\mathbf{z}_1,\phi_1)}{q(\mathbf{z}_1|\mathbf{x})}\right] + \log\left[\frac{\prod_{t=2}^{T}Pr(\mathbf{z}_{t-1}|\mathbf{z}_t,\phi_t)}{\prod_{t=2}^{T}q(\mathbf{z}_t|\mathbf{z}_{t-1})}\right] + \log\left[Pr(\mathbf{z}_T)\right]$$

3.2 **Apply Markov property and eliminate variables**

$$= \log\left[\frac{Pr(\mathbf{x}|\mathbf{z}_1,\phi_1)}{q(\mathbf{z}_1|\mathbf{x})}\right] + \log\left[\frac{\prod_{t=2}^{T}Pr(\mathbf{z}_{t-1}|\mathbf{z}_t,\phi_t)}{\prod_{t=2}^{T}q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})}\cdot\frac{q(\mathbf{z}_{t-1}|\mathbf{x})}{q(\mathbf{z}_t|\mathbf{x})}\right] + \log\left[Pr(\mathbf{z}_T)\right] \leftarrow \text{Markov}$$
$$= \log\left[Pr(\mathbf{x}|\mathbf{z}_1,\phi_1)\right] + \log\left[\frac{\prod_{t=2}^{T}Pr(\mathbf{z}_{t-1}|\mathbf{z}_t,\phi_t)}{\prod_{t=2}^{T}q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})}\right] + \log\left[\frac{Pr(\mathbf{z}_T)}{q(\mathbf{z}_T|\mathbf{x})}\right] \leftarrow \to 0$$
$$\approx \log\left[Pr(\mathbf{x}|\mathbf{z}_1,\phi_1)\right] + \sum_{t=2}^{T}\log\left[\frac{Pr(\mathbf{z}_{t-1}|\mathbf{z}_t,\phi_t)}{q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})}\right]$$

$\prod_t \frac{q(\mathbf{z}_{t-1}|\mathbf{x})}{q(\mathbf{z}_t|\mathbf{x})} = \frac{q(\mathbf{z}_1|\mathbf{x})}{q(\mathbf{z}_2|\mathbf{x})}\cdot\frac{q(\mathbf{z}_2|\mathbf{x})}{q(\mathbf{z}_3|\mathbf{x})}\cdots = \frac{q(\mathbf{z}_1|\mathbf{x})}{q(\mathbf{z}_T|\mathbf{x})}$

Markov: $q(\mathbf{z}_t|\mathbf{z}_{t-1}) = q(\mathbf{z}_t|\mathbf{z}_{t-1},\mathbf{x}) = \frac{q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})q(\mathbf{z}_t|\mathbf{x})}{q(\mathbf{z}_{t-1}|\mathbf{x})}$

3.3 **Plug into ELBO and simplify**

$$\approx \int q(\mathbf{z}_{1\ldots T}|\mathbf{x})\left(\log\left[Pr(\mathbf{x}|\mathbf{z}_1,\phi_1)\right] + \sum_{t=2}^{T}\log\left[\frac{Pr(\mathbf{z}_{t-1}|\mathbf{z}_t,\phi_t)}{q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})}\right]\right)d\mathbf{z}_{1\ldots T}$$
$$= \mathbb{E}_{q(\mathbf{z}_1|\mathbf{x})}\left[\log\left[Pr(\mathbf{x}|\mathbf{z}_1,\phi_1)\right]\right] - \sum_{t=2}^{T}\mathbb{E}_{q(\mathbf{z}_t|\mathbf{x})}\left[D_{KL}\left[q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})\|Pr(\mathbf{z}_{t-1}|\mathbf{z}_t,\phi_t)\right]\right]$$

4. Insert definitions and rewrite KL divergence

Definitions:

$$Pr(\mathbf{x}|\mathbf{z}_1,\boldsymbol{\phi}_1) = \text{Norm}_{\mathbf{x}}\Big[\mathbf{f}_1[\mathbf{z}_1,\boldsymbol{\phi}_1],\sigma_1^2\mathbf{I}\Big]$$

$$Pr(\mathbf{z}_{t-1}|\mathbf{z}_t,\boldsymbol{\phi}_t) = \text{Norm}_{\mathbf{z}_{t-1}}\Big[\mathbf{f}_t[\mathbf{z}_t,\boldsymbol{\phi}_t],\sigma_t^2\mathbf{I}\Big]$$

$$q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x}) = \text{Norm}_{\mathbf{z}_{t-1}}\left[\frac{(1-\alpha_{t-1})}{1-\alpha_t}\sqrt{1-\beta_t}\mathbf{z}_t + \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\alpha_t}\mathbf{x}, \frac{\beta_t(1-\alpha_{t-1})}{1-\alpha_t}\mathbf{I}\right]$$

KL divergence for gaussians:

$$D_{KL}\Big[q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})\Big|\Big|Pr(\mathbf{z}_{t-1}|\mathbf{z}_t,\boldsymbol{\phi}_t)\Big] =$$

$$\frac{1}{2\sigma_t^2}\left\|\frac{(1-\alpha_{t-1})}{1-\alpha_t}\sqrt{1-\beta_t}\mathbf{z}_t + \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\alpha_t}\mathbf{x} - \mathbf{f}_t[\mathbf{z}_t,\boldsymbol{\phi}_t]\right\|^2$$

→ is again a gaussian

5. Define the loss and rewrite for training

5.1 Full loss term

$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I}\Bigg(\overbrace{-\log\Big[\text{Norm}_{\mathbf{x}_i}\left[\mathbf{f}_1[\mathbf{z}_{i1},\boldsymbol{\phi}_1],\sigma_1^2\mathbf{I}\right]\Big]}^{\text{reconstruction term}}$$
$$+\sum_{t=2}^{T}\frac{1}{2\sigma_t^2}\Bigg\|\underbrace{\frac{(1-\alpha_{t-1})}{1-\alpha_t}\sqrt{1-\beta_t}\mathbf{z}_{it} + \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\alpha_t}\mathbf{x}_i}_{\text{target, mean of } q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})} - \underbrace{\mathbf{f}_t[\mathbf{z}_{it},\boldsymbol{\phi}_t]}_{\text{predicted } \mathbf{z}_{t-1}}\Bigg\|^2\Bigg)$$

5.2 Reformulate loss

$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I} -\log\Big[\text{Norm}_{\mathbf{x}_i}\left[\mathbf{f}_1[\mathbf{z}_{i1},\boldsymbol{\phi}_1],\sigma_1^2\mathbf{I}\right]\Big]$$
$$+\sum_{t=2}^{T}\frac{1}{2\sigma_t^2}\left\|\left(\frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_{it} - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\boldsymbol{\epsilon}\right) - \mathbf{f}_t[\mathbf{z}_{it},\boldsymbol{\phi}_t]\right\|^2$$

5.3 Reparametrize s.t. the model predicts noise ϵ instead of latent variables

$$\mathbf{f}_t[\mathbf{z}_t,\boldsymbol{\phi}_t] = \frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\mathbf{g}_t[\mathbf{z}_t,\boldsymbol{\phi}_t]$$

$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I}\sum_{t=1}^{T}\frac{\beta_t^2}{(1-\alpha_t)(1-\beta_t)2\sigma_t^2}\left\|\mathbf{g}_t[\mathbf{z}_{it},\boldsymbol{\phi}_t] - \boldsymbol{\epsilon}_{it}\right\|^2$$

5.4 Ignore the scaling factor

$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I}\sum_{t=1}^{T}\left\|\mathbf{g}_t[\mathbf{z}_{it},\boldsymbol{\phi}_t] - \boldsymbol{\epsilon}_{it}\right\|^2$$
$$= \sum_{i=1}^{I}\sum_{t=1}^{T}\left\|\mathbf{g}_t\left[\sqrt{\alpha_t}\cdot\mathbf{x}_i + \sqrt{1-\alpha_t}\cdot\boldsymbol{\epsilon}_{it},\boldsymbol{\phi}_t\right] - \boldsymbol{\epsilon}_{it}\right\|^2$$

Conditional Diffusion Distribution $q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x})$

Tractable distribution used for training

$$q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x}) = \text{Norm}_{\mathbf{z}_{t-1}}\left[\frac{(1-\alpha_{t-1})}{1-\alpha_t}\sqrt{1-\beta_t}\mathbf{z}_t + \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\alpha_t}\mathbf{x}, \frac{\beta_t(1-\alpha_{t-1})}{1-\alpha_t}\mathbf{I}\right]$$

Derivation:

1. Rewrite distribution

$$\begin{aligned}
& q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x}) \\
\propto\ & q(\mathbf{z}_t|\mathbf{z}_{t-1})q(\mathbf{z}_{t-1}|\mathbf{x}) \\
=\ & \text{Norm}_{\mathbf{z}_t}\left[\sqrt{1-\beta_t}\cdot\mathbf{z}_{t-1}, \beta_t\mathbf{I}\right]\text{Norm}_{\mathbf{z}_{t-1}}\left[\sqrt{\alpha_{t-1}}\cdot\mathbf{x}, (1-\alpha_{t-1})\mathbf{I}\right] \\
\propto\ & \text{Norm}_{\mathbf{z}_{t-1}}\left[\frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_t, \frac{\beta_t}{1-\beta_t}\mathbf{I}\right]\text{Norm}_{\mathbf{z}_{t-1}}\left[\sqrt{\alpha_{t-1}}\cdot\mathbf{x}, (1-\alpha_{t-1})\mathbf{I}\right]
\end{aligned}$$

← exchange variables

with: $\text{Norm}_{\mathbf{v}}[\mathbf{A}\mathbf{w}, \mathbf{B}] \propto \text{Norm}_{\mathbf{w}}\left[(\mathbf{A}^T\mathbf{B}^{-1}\mathbf{A})^{-1}\mathbf{A}^T\mathbf{B}^{-1}\mathbf{v}, (\mathbf{A}^T\mathbf{B}^{-1}\mathbf{A})^{-1}\right]$

V: z_t , W: z_{t-1} , $A = \sqrt{1-\beta_t}I$, $B = \beta_t I$

2. Apply product rule for gaussians

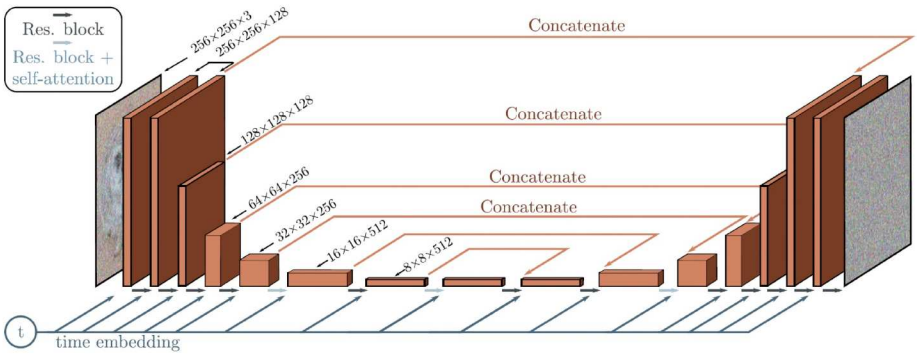
$$q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x}) = \text{Norm}_{\mathbf{z}_{t-1}}\left[\frac{(1-\alpha_{t-1})}{1-\alpha_t}\sqrt{1-\beta_t}\mathbf{z}_t + \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\alpha_t}\mathbf{x}, \frac{\beta_t(1-\alpha_{t-1})}{1-\alpha_t}\mathbf{I}\right]$$

with: $\text{Norm}_{\mathbf{w}}[\mathbf{a}, \mathbf{A}]\cdot\text{Norm}_{\mathbf{w}}[\mathbf{b}, \mathbf{B}] \propto \text{Norm}_{\mathbf{w}}\left[(\mathbf{A}^{-1}+\mathbf{B}^{-1})^{-1}(\mathbf{A}^{-1}\mathbf{a}+\mathbf{B}^{-1}\mathbf{b}), (\mathbf{A}^{-1}+\mathbf{B}^{-1})^{-1}\right]$

Generation:

Algorithm 18.2: Sampling

Input: Model, $\mathbf{g}_t[\bullet, \boldsymbol{\phi}_t]$
Output: Sample, $\mathbf{x}$
$\mathbf{z}_T \sim \text{Norm}_{\mathbf{z}}[\mathbf{0}, \mathbf{I}]$ // Sample last latent variable
for $t = T \ldots 2$ **do**
  $\hat{\mathbf{z}}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\mathbf{g}_t[\mathbf{z}_t, \boldsymbol{\phi}_t]$ // Predict previous latent variable
  $\boldsymbol{\epsilon} \sim \text{Norm}_{\boldsymbol{\epsilon}}[\mathbf{0}, \mathbf{I}]$ // Draw new noise vector
  $\mathbf{z}_{t-1} = \hat{\mathbf{z}}_{t-1} + \sigma_t\boldsymbol{\epsilon}$ // Add noise to previous latent variable
$\mathbf{x} = \frac{1}{\sqrt{1-\beta_1}}\mathbf{z}_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}\sqrt{1-\beta_1}}\mathbf{g}_1[\mathbf{z}_1, \boldsymbol{\phi}_1]$ // Generate sample from $\mathbf{z}_1$ without noise



XI Structured Learning and Prediction

Definition: Structured Learning
Output are complex (structured/discrete) objects

$$f : \mathcal{X} \to \mathcal{Y}.$$

Definition: Structured Data
Data consists of structured parts and not the parts themselves contain information

Similarity Measures for Conditional Distributions

KL-Divergence:

$$\mathrm{KL}_{cond}(p||d)(x) := \sum_{y \in \mathcal{Y}} d(y|x) \log \frac{d(y|x)}{p(y|x,w)}$$

Expected KL:

$$\begin{aligned}\mathrm{KL}_{tot}(p||d) &:= \mathbb{E}_{x \sim d(x)} \Big\{ \mathrm{KL}_{\mathrm{cond}}(p||d)(x) \Big\} \\ &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} d(x,y) \log \frac{d(y|x)}{p(y|x,w)}\end{aligned}$$

→ conditioned on all data

Learning Techniques:

Maximum Likelihood:

$$\begin{aligned} w^* &= \operatorname*{argmax}_{w \in \mathbb{R}^D} \; p(y^1, \dots, y^N | x^1, \dots, x^N, w) \\ &\overset{i.i.d.}{=} \operatorname*{argmax}_{w \in \mathbb{R}^D} \; \prod_{n=1}^{N} p(y^n | x^n, w) \\ &\overset{-\log(\cdot)}{=} \operatorname*{argmin}_{w \in \mathbb{R}^D} \; \underbrace{-\sum_{n=1}^{N} \log p(y^n | x^n, w)}_{\text{negative conditional log-likelihood (of } \mathcal{D})} \end{aligned}$$

Best Approximation:
→ minimize total KL-divergence

$$\begin{aligned} w^* &= \operatorname*{argmin}_{w \in \mathbb{R}^D} \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} d(x,y) \log \frac{d(y|x)}{p(y|x,w)} \\ &\overset{\text{drop const.}}{=} \operatorname*{argmin}_{w \in \mathbb{R}^D} - \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} d(x,y) \log p(y|x,w) \\ &\overset{(x^n,y^n) \sim d(x,y)}{\approx} \operatorname*{argmin}_{w \in \mathbb{R}^D} \; \underbrace{-\sum_{n=1}^{N} \log p(y^n | x^n, w)}_{\text{negative conditional log-likelihood (of } \mathcal{D})} \end{aligned}$$

MAP Estimation:

Maximize posterior $p(w|\mathcal{D}) \overset{\text{Bayes}}{=} \dfrac{p(x^1, y^1, \dots, x^n, y^n | w) p(w)}{p(\mathcal{D})} \overset{i.i.d.}{=} p(w) \prod_{n=1}^{N} \dfrac{p(y^n|x^n, w)}{p(y^n|x^n)}$

$$\begin{aligned} w^* &= \operatorname*{argmax}_{w \in \mathbb{R}^D} p(w|\mathcal{D}) = \operatorname*{argmin}_{w \in \mathbb{R}^D} \big[-\log p(w|\mathcal{D}) \big] \\ &= \operatorname*{argmin}_{w \in \mathbb{R}^D} \Big[-\log p(w) - \sum_{n=1}^{N} \log p(y^n|x^n, w) + \underbrace{\log p(y^n|x^n)}_{\text{indep. of } w} \Big] \\ &= \operatorname*{argmin}_{w \in \mathbb{R}^D} \Big[-\log p(w) - \sum_{n=1}^{N} \log p(y^n|x^n, w) \Big] \end{aligned}$$

Choice prior belief

Constant: $p(w) :\equiv \text{const.}$

$$w^* = \operatorname*{argmin}_{w \in \mathbb{R}^D} \Big[\underbrace{-\sum_{n=1}^{N} \log p(y^n|x^n, w)}_{\text{negative conditional log-likelihood}} + \text{const.} \Big]$$

Gaussian: $p(w) := const. \cdot e^{-\frac{1}{2\sigma^2}\|w\|^2}$

$$w^* = \operatorname*{argmin}_{w \in \mathbb{R}^D} \Big[\underbrace{\frac{1}{2\sigma^2}\|w\|^2 - \sum_{n=1}^{N} \log p(y^n|x^n, w)}_{\text{regularized negative conditional log-likelihood}} + \text{const.} \Big]$$

Loss Minimization:

Let $d(x, y)$ be the (unknown) true data distribution.
Let $\mathcal{D} = \{(x^1, y^1), \ldots, (x^N, y^N)\}$ be i.i.d. samples from $d(x, y)$.
Let $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$ be a feature function.
Let $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be a loss function.

- *Find a weight vector w^* that leads to minimal expected loss*

$$\mathbb{E}_{(x,y)\sim d(x,y)}\{\Delta(y, f(x))\}$$

for $f(x) = \text{argmax}_{y\in\mathcal{Y}} \langle w, \phi(x, y)\rangle$.

Conditional Random Field Learning

CRF learning is typically a gradient-based optimization:
→ all approaches turn out to be very similar

CRF Model:
a set of i.i.d. samples $\mathcal{D} = \{(x^n, y^n)\}_{n=1,\ldots,N}, \quad (x^n, y^n) \sim d(x, y)$
feature functions $(\phi_1(x, y), \ldots, \phi_D(x, y)) \equiv: \phi(x, y)$
parametrized family $p(y|x, w) = \frac{1}{Z(x,w)} \exp(\langle w, \phi(x, y)\rangle)$

Feature functions:

$\phi_i(y_i, x)$: local representation, high-dimensional
→ $\langle w_i, \phi_i(y_i, x)\rangle$: local classifier

$\phi_{i,j}(y_i, y_j)$: prior knowledge, low-dimensional
→ $\langle w_{ij}, \phi_{ij}(y_i, y_j)\rangle$: penalize outliers

Learning:
unary w_i: learn local classifiers and their importance
binary w_{ij}: learn importance of smoothing/penalization

Goal: Find optimal w for $p(y|x,w)$ using $\mathcal{D}$

Minimize:

$$\mathcal{L}(w) = \frac{1}{2\sigma^2}\|w\|^2 - \sum_{n=1}^{N}\left[\langle w, \phi(x^n, y^n)\rangle - \log \sum_{y\in\mathcal{Y}} e^{\langle w,\phi(x^n,y)\rangle}\right]$$

→ negative regularized conditional log-likelihood
→ differentiable and convex

Derivation: MAP estimation with gaussian prior

Gradient:

$$\nabla_w \mathcal{L}(w) = \frac{1}{\sigma^2} w - \sum_{n=1}^{N}\left[\phi(x^n, y^n) - \mathbb{E}_{y\sim p(y|x^n,w)}\phi(x^n, y)\right]$$

Derivation:

$$\begin{aligned}\nabla_w \mathcal{L}(w) &= \frac{1}{\sigma^2} w - \sum_{n=1}^{N}\left[\phi(x^n, y^n) - \frac{\sum_{y\in\mathcal{Y}} e^{\langle w,\phi(x^n,y)\rangle}\phi(x^n, y)}{\sum_{\bar{y}\in\mathcal{Y}} e^{\langle w,\phi(x^n,\bar{y})\rangle}}\right] \\ &= \frac{1}{\sigma^2} w - \sum_{n=1}^{N}\left[\phi(x^n, y^n) - \sum_{y\in\mathcal{Y}} p(y|x^n, w)\phi(x^n, y)\right] \\ &= \frac{1}{\sigma^2} w - \sum_{n=1}^{N}\left[\phi(x^n, y^n) - \mathbb{E}_{y\sim p(y|x^n,w)}\phi(x^n, y)\right]\end{aligned}$$

Hessian:

$$\Delta\mathcal{L}(w) = \frac{1}{\sigma^2} Id_{D\times D} + \sum_{n=1}^{N}[\mathbb{E}_{y\sim p(y|x^n,w)}\phi(x^n, y)][\mathbb{E}_{y\sim p(y|x^n,w)}\phi(x^n, y)]^\top$$

→ positive definite

Problems: The main problem of this learning method is the computational costs:

$$\nabla_w \mathcal{L}(w) = \frac{1}{\sigma^2} w - \sum_{n=1}^{N}\left[\phi(x^n, y^n) - \mathbb{E}_{y\sim p(y|x^n,w)}\phi(x^n, y)\right]$$

← Y is very large

Computing the Gradient (naive): $O(K^M ND)$

$$\mathcal{L}(w) = \frac{1}{2\sigma^2}\|w\|^2 - \sum_{n=1}^{N}\left[\langle w, \phi(x^n, y^n)\rangle - \log Z(x^n, w)\right]$$

Line Search (naive): $O(K^M ND)$ per evaluation of $\mathcal{L}$

N: number of samples
D: dimension of feature space
M: number of output nodes ≈ 100s to 1,000,000s
K: number of possible labels of each output nodes ≈ 2 to 100s

Solutions:

problem	"solution"	method(s)
$\lvert\mathcal{Y}\rvert$ too large	exploit structure	(loopy) belief propagation
	smart sampling	contrastive divergence
	use approximate $\mathcal{L}$	e.g. pseudo-likelihood
N too large	mini-batches	stochastic gradient descent
D too large	trained ϕ_{unary}	two-stage training

Exploit Structure:

Decompose joint distribution into marginal distributions over local neighbors:

$$\phi(x,y) = \left(\phi_F(x,y_F)\right)_{F\in\mathcal{F}}$$

$$\begin{aligned}\mathbb{E}_{y\sim p(y|x,w)}\phi(x,y) &= \left(\mathbb{E}_{y\sim p(y|x,w)}\phi_F(x,y_F)\right)_{F\in\mathcal{F}} \\ &= \left(\mathbb{E}_{y_F\sim p(y_F|x,w)}\phi_F(x,y_F)\right)_{F\in\mathcal{F}}\end{aligned}$$

$$\mathbb{E}_{y_F\sim p(y_F|x,w)}\phi_F(x,y_F) = \underbrace{\sum_{y_F\in\mathcal{Y}_F}}_{K^{|F|}\text{ terms}} \underbrace{p(y_F|x,w)}_{\text{factor marginals}}\ \phi_F(x,y_F)$$

Computing the Gradient: ~~$O(K^M nd)$~~, $O(MK^{|F_{max}|}ND)$

Line Search: ~~$O(K^M nd)$~~, $O(MK^{|F_{max}|}ND)$ per evaluation of $\mathcal{L}$

Mini-Batches (SGD):

Compute the gradient for a small subset than all training data.

→ requires extra step size parameter

→ requires more iterations but each one is faster

$$\tilde{\nabla}_w\,\mathcal{L}(w) = \frac{1}{\sigma^2}w - \sum_{(x^n,y^n)\in\mathcal{D}'}\left[\phi(x^n,y^n) - \mathbb{E}_{y\sim p(y|x^n,w)}\phi(x^n,y)\right]$$

Computing the Gradient: ~~$O(K^M nd)$~~, $O(MK^2ND)$

Line Search: ~~$O(K^M nd)$~~, $O(MK^2ND)$ per evaluation of $\mathcal{L}$

Trained ϕ_{unary}:

Split training into training the local classifiers and their importance

→ faster due to lower dimensional feature space

→ stronger classifiers

→ CRF training can't fix bad classifier

Two-stage Training:

1. Pre-train $f_i^Y(x) = \log p(y_i|x)$
2. Use $\tilde{\phi}_i(y_i,x) = f_i^Y(x) \in \mathbb{R}^k$ and keep $\phi_{ij}(y_i,y_j)$
3. Perform CRF learning

Structured SVM

Regularized Risk Minimization:

Solve for: $\min_w \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N}\left[\max_{y\in\mathcal{Y}} \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle\right]$

→ unconstrained and convex
→ non-differentiable objective

Derivation:

1. Define optimization problem

Minimize $\mathbb{E}_{(x,y)\sim d(x,y)}\ \Delta(\ y, \operatorname{argmax}_y g(x, y; w)\ )$. Δ: loss

with: $g(x, y; w) := \langle w, \phi(x, y)\rangle$

Problems:

1. $d(x,y)$ unknown
2. $\operatorname{argmax}_y g(x, y; w)$ maps into a discrete space
   → $\Delta(\ y, \operatorname{argmax}_y g(x, y; w))$ is discontinuous, piecewise constant

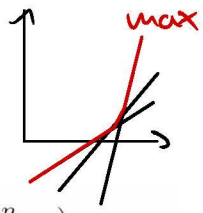
2. Replace expectation with an empirical statement and add regularizer
   → computation over training samples

$$\min_w \quad \lambda\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N}\Delta(\ y^n, \operatorname{argmax}_y g(x^n, y; w)\ ).$$

3. Approximate loss through an upper bound
   → hinge loss
   → continuous and convex

$$\min_w \quad \lambda\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N}\ell(x^n, y^n, w)$$

with: $\ell(x^n, y^n, w) := \max_{y\in\mathcal{Y}}\left[\ \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle\ \right]$



Proof upper bound:

$$\begin{aligned}\Delta(y^n, \bar{y}) &\le \Delta(y^n, \bar{y}) + g(x^n, \bar{y}, w) - g(x^n, y^n, w)\\ &\le \max_{y\in\mathcal{Y}}\left[\Delta(y^n, y) + g(x^n, y, w) - g(x^n, y^n, w)\right]\end{aligned}$$

Equivalent Formulations:

1. $$\min_{w,\xi} \quad \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N}\xi^n$$
   subject to, for $n = 1, \ldots, N$,
   $$\max_{y\in\mathcal{Y}}\left[\Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle\right] \le \xi^n$$
   → # non-linear constraints
   → differentiable

2. $$\min_{w,\xi} \quad \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N}\xi^n$$
   subject to, for $n = 1, \ldots, N$,
   $$\Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle \le \xi^n, \quad \text{for all } y \in \mathcal{Y}$$
   → linear constraints

3. Solve
   $$\min_{w\in\mathbb{R}^D, \xi\in\mathbb{R}^n_+} \|w\|^2 + \frac{C}{N}\sum_{n=1}^{N}\xi^n$$
   subject to, for $i = 1, \ldots n$, for all $y \in \mathcal{Y}$,
   $$\langle w, \delta\phi(x^n, y^n, y)\rangle \ge \Delta(y^n, y) - \xi^n.$$
   → linear constraints
   → large amount of constraints
   → quadratic objective
   → solvable with QP

$\delta\phi(x^n, y^n, y) := \phi(x^n, y^n) - \phi(x^n, y)$

4.

$$\min_{w\in\mathbb{R}^D,\xi\in\mathbb{R}_+} \quad \frac{1}{2}\|w\|^2 + C\xi$$

subject to, for all $(\hat{y}^1,\ldots,\hat{y}^N) \in \mathcal{Y}\times\cdots\times\mathcal{Y}$,

$$\sum_{n=1}^{N}\left[\Delta(y^n,\hat{y}^n) + \langle w,\phi(x^n,\hat{y}^n)\rangle - \langle w,\phi(x^n,y^n)\rangle\right] \le N\xi,$$

→ average over constraints
→ use one slack variable

Optimization: Since the objective is not differentiable, we use sub-gradients
→ here we have again the same problems as for the CRF

Subgradient Descent:

input training pairs $\{(x^1,y^1),\ldots,(x^n,y^n)\}\subset\mathcal{X}\times\mathcal{Y}$,
input feature map $\phi(x,y)$, loss function $\Delta(y,y')$, regularizer C,
input number of iterations T, stepsizes η_t for $t=1,\ldots,T$

$w \leftarrow \vec{0}$
for t=1,...,T **do**
  for i=1,...,n **do**
    $\hat{y} \leftarrow \operatorname{argmax}_{y\in\mathcal{Y}} \Delta(y^n,y) + \langle w,\phi(x^n,y)\rangle - \langle w,\phi(x^n,y^n)\rangle$
    $v^n \leftarrow \phi(x^n,\hat{y}) - \phi(x^n,y^n)$
  end for
  $w \leftarrow w - \eta_t(w - \frac{C}{N}\sum_n v^n)$
end for

output prediction function $f(x) = \operatorname{argmax}_{y\in\mathcal{Y}}\langle w,\phi(x,y)\rangle$.

Stochastic Subgradient Descent:
→ counteract large amounts of data

input training pairs $\{(x^1,y^1),\ldots,(x^n,y^n)\}\subset\mathcal{X}\times\mathcal{Y}$,
input feature map $\phi(x,y)$, loss function $\Delta(y,y')$, regularizer C,
input number of iterations T, stepsizes η_t for $t=1,\ldots,T$

$w \leftarrow \vec{0}$
for t=1,...,T **do**
  (x^n,y^n) ← randomly chosen training example pair
  $\hat{y} \leftarrow \operatorname{argmax}_{y\in\mathcal{Y}} \Delta(y^n,y) + \langle w,\phi(x^n,y)\rangle - \langle w,\phi(x^n,y^n)\rangle$
  $w \leftarrow w - \eta_t(w - C[\phi(x^n,\hat{y}) - \phi(x^n,y^n)])$
end for

output prediction function $f(x) = \operatorname{argmax}_{y\in\mathcal{Y}}\langle w,\phi(x,y)\rangle$.

Working Set Training:
Iteratively solve with only a subset of constraints that are most violated
→ uses the third formulation

input training pairs $\{(x^1,y^1),\ldots,(x^n,y^n)\}\subset\mathcal{X}\times\mathcal{Y}$,
input feature map $\phi(x,y)$, loss function $\Delta(y,y')$, regularizer C

$S \leftarrow \emptyset$
repeat
  $(w,\xi) \leftarrow$ *solution to QP only with constraints from* S
  for i=1,...,n **do**
    $\hat{y} \leftarrow \operatorname{argmax}_{y\in\mathcal{Y}} \Delta(y^n,y) + \langle w,\phi(x^n,y)\rangle$
    if $\hat{y} \neq y^n$ **then**
      $S \leftarrow S \cup \{(x^n,\hat{y})\}$
    end if
  end for
until S doesn't change anymore.

output prediction function $f(x) = \operatorname{argmax}_{y\in\mathcal{Y}}\langle w,\phi(x,y)\rangle$.

Start with working set $S=\emptyset$ (no constraints)
Repeat until convergence:
- Solve S-SVM training problem with constraints from S
- Check, if solution violates any of the *full* constraint set
  - if no: we found the optimal solution, *terminate*.
  - if yes: add most violated constraints to S, *iterate*.

Working Set on One-Slack formulation:

input training pairs $\{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$,
input feature map $\phi(x, y)$, loss function $\Delta(y, y')$, regularizer C

1: $S \leftarrow \emptyset$
2: **repeat**
3: $(w, \xi) \leftarrow$ *solution to QP only with constraints from* S
4: **for** i=1,...,n **do**
5: $\hat{y}^n \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \Delta(y^n, y) + \langle w, \phi(x^n, y) \rangle$
6: **end for**
7: $S \leftarrow S \cup \{((x^1, \ldots, x^n), (\hat{y}^1, \ldots, \hat{y}^n))\}$
8: **until** S doesn't change anymore.

output prediction function $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \phi(x, y) \rangle$.

Dual Structured SVM

$$\max_{\alpha \in \mathbb{R}_+^{n|\mathcal{Y}|}} \sum_{\substack{n=1,\ldots,n \\ y \in \mathcal{Y}}} \alpha_{ny} \Delta(y^n, y) - \frac{1}{2} \sum_{\substack{y, \bar{y} \in \mathcal{Y} \\ n, \bar{n} = 1, \ldots, N}} \alpha_{ny} \alpha_{\bar{n}\bar{y}} \left\langle \delta\phi(x^n, y^n, y), \delta\phi(x^{\bar{n}}, y^{\bar{n}}, \bar{y}) \right\rangle$$

subject to, for $n = 1, \ldots, N$,

$$\sum_{y \in \mathcal{Y}} \alpha_{ny} \leq \frac{C}{N}.$$

Kernelized:

$$\max_{\alpha \in \mathbb{R}_+^{n|\mathcal{Y}|}} \sum_{\substack{i=1,\ldots,n \\ y \in \mathcal{Y}}} \alpha_{iy} \Delta(y^n, y) - \frac{1}{2} \sum_{\substack{y, \bar{y} \in \mathcal{Y} \\ i, \bar{i} = 1, \ldots, n}} \alpha_{iy} \alpha_{\bar{i}\bar{y}} K_{i\bar{i}y\bar{y}}$$

subject to, for $i = 1, \ldots, n$,

$$\sum_{y \in \mathcal{Y}} \alpha_{iy} \leq \frac{C}{N}.$$

Multiclass SVM
→ in practice does not bring an advantage

- $\mathcal{Y} = \{1, 2, \ldots, K\}, \quad \Delta(y, y') = \begin{cases} 1 & \text{for } y \neq y' \\ 0 & \text{otherwise} \end{cases}$
- $\phi(x, y) = \Big([\![y = 1]\!]\phi(x),\ [\![y = 2]\!]\phi(x),\ \ldots,\ [\![y = K]\!]\phi(x) \Big)$

Solve: $\min_{w, \xi} \frac{1}{2}\|w\|^2 + \frac{C}{N} \sum_{n=1}^{N} \xi^n$

subject to, for $i = 1, \ldots, n$,

$$\langle w, \phi(x^n, y^n) \rangle - \langle w, \phi(x^n, y) \rangle \geq 1 - \xi^n \quad \text{for all } y \in \mathcal{Y} \setminus \{y^n\}.$$

Classification: $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \ \langle w, \phi(x, y) \rangle$.

Beyond Fully-Supervised Learning

Marginalization over Latent Variables

Construct conditional likelihood as usual:

$$p(y, z | x, w) = \frac{1}{Z(x, w)} \exp(\langle w, \phi(x, y, z) \rangle)$$

Derive $p(y|x, w)$ by marginalizing over z:

$$p(y | x, w) = \frac{1}{Z(x, w)} \sum_{z \in \mathcal{Z}} p(y, z | x, w)$$

Three types of variables:
- $x \in \mathcal{X}$ always observed,
- $y \in \mathcal{Y}$ observed only in training,
- $z \in \mathcal{Z}$ never observed (latent).

Decision function: $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \ \max_{z \in \mathcal{Z}} \ \langle w, \phi(x, y, z) \rangle$

Maximum Margin Training with Maximization over Latent Variables

Solve: $\min_{w, \xi} \frac{1}{2}\|w\|^2 + \frac{C}{N} \sum_{n=1}^{N} \xi^n$

subject to, for $n = 1, \ldots, N$, for all $y \in \mathcal{Y}$

$$\Delta(y^n, y) + \max_{z \in \mathcal{Z}} \langle w, \phi(x^n, y, z) \rangle - \max_{z \in \mathcal{Z}} \langle w, \phi(x^n, y^n, z) \rangle \ \leq \xi^n$$