Latent Variable Models (contd) and Deep Neural Networks

CS771: Introduction to Machine Learning

MLE for Latent Variable Models

Optimization is difficult in general because the objective has a complex form, complex derivatives, so FOO etc won't apply easily

- Original problem: $\Theta_{MLE} = \underset{\Theta}{\text{argmax}} \log p(\boldsymbol{X}|\Theta) = \underset{\Theta}{\text{argmax}} \log \sum_{\boldsymbol{z}} p(\boldsymbol{X}, \boldsymbol{Z}|\Theta)$
- The EM approach for solving MLE problem in LVMs

MAP for $\boldsymbol{\Theta}$ can also be done by adding a log prior term for $\boldsymbol{\Theta}$

$$\Theta_{MLE} = \underset{\Theta}{\text{argmax}} \, \mathbb{E}_{p(\boldsymbol{Z}|\Theta,\boldsymbol{X})}[\log p(\boldsymbol{X}, \boldsymbol{Z}|\Theta)]$$

- Can be easily shown (exercise) that original objective (log-lik) can be written as

$$\log p(\boldsymbol{X}|\Theta) = \mathcal{L}(q, \Theta) + KL(q||p_z)$$

Holds for any distribution $\boldsymbol{q(Z)}$ over $\boldsymbol{Z}$

$$\boxed{\log p(\boldsymbol{X}|\Theta) \geq \mathcal{L}(q, \Theta)}$$

$$\mathcal{L}(q, \Theta) = \sum_{\boldsymbol{Z}} q(\boldsymbol{Z}) \log \left\{ \frac{p(\boldsymbol{X}, \boldsymbol{Z}|\Theta)}{q(\boldsymbol{Z})} \right\}$$

Thus $\mathcal{L}(q, \Theta)$ is a lower bound on the log likelihood of the LVM

KL is always non-negative

$$KL(q||p_z) = -\sum_{\boldsymbol{Z}} q(\boldsymbol{Z}) \log \left\{ \frac{p(\boldsymbol{Z}|\boldsymbol{X}, \Theta)}{q(\boldsymbol{Z})} \right\}$$

EM maximizes this lower bound to get (an approximate) MLE

Maximizing the Lower Bound

The posterior distribution of Z given older parameters Θ^{old} (will need this posterior to get the expectation of CLL)

- As we saw, $\mathcal{L}(q,\Theta)$ depends on q and Θ. Consider ALT-OPT.
- Let's maximize $\mathcal{L}(q,\Theta)$ w.r.t. q with Θ fixed at Θ^{old}

Since $\log p(\boldsymbol{X}|\Theta) = \mathcal{L}(q,\Theta) + KL(q||p_z)$ is constant when Θ is held fixed at Θ^{old}

$$\hat{q} = \text{argmax}_q \mathcal{L}(q,\Theta^{\text{old}}) = \text{argmin}_q KL(q||p_z) = p_z = p(\boldsymbol{Z}|\boldsymbol{X},\Theta^{\text{old}})$$

- Now let's maximize $\mathcal{L}(q,\Theta)$ w.r.t. Θ with q fixed at $\hat{q} = p_z = p(\boldsymbol{Z}|\boldsymbol{X},\Theta^{\text{old}})$

$$\Theta^{\text{new}} = \text{argmax}_\Theta \mathcal{L}(\hat{q},\Theta) = \text{argmax}_\Theta \sum_{Z} p(\boldsymbol{Z}|\boldsymbol{X},\Theta^{\text{old}}) \log\left\{\frac{p(\boldsymbol{X},\boldsymbol{Z}|\Theta)}{p(\boldsymbol{Z}|\boldsymbol{X},\Theta^{\text{old}})}\right\}$$

$$= \text{argmax}_\Theta \sum_{Z} p(\boldsymbol{Z}|\boldsymbol{X},\Theta^{\text{old}}) \log p(\boldsymbol{X},\boldsymbol{Z}|\Theta)$$

Maximization of expected CLL w.r.t. the posterior distribution of Z given older parameters Θ^{old}

$$= \text{argmax}_\Theta \mathbb{E}_{p(\boldsymbol{Z}|\boldsymbol{X},\Theta^{\text{old}})}[\log p(\boldsymbol{X},\boldsymbol{Z}|\Theta)]$$

$$= \text{argmax}_\Theta \mathcal{Q}(\Theta,\Theta^{\text{old}})$$

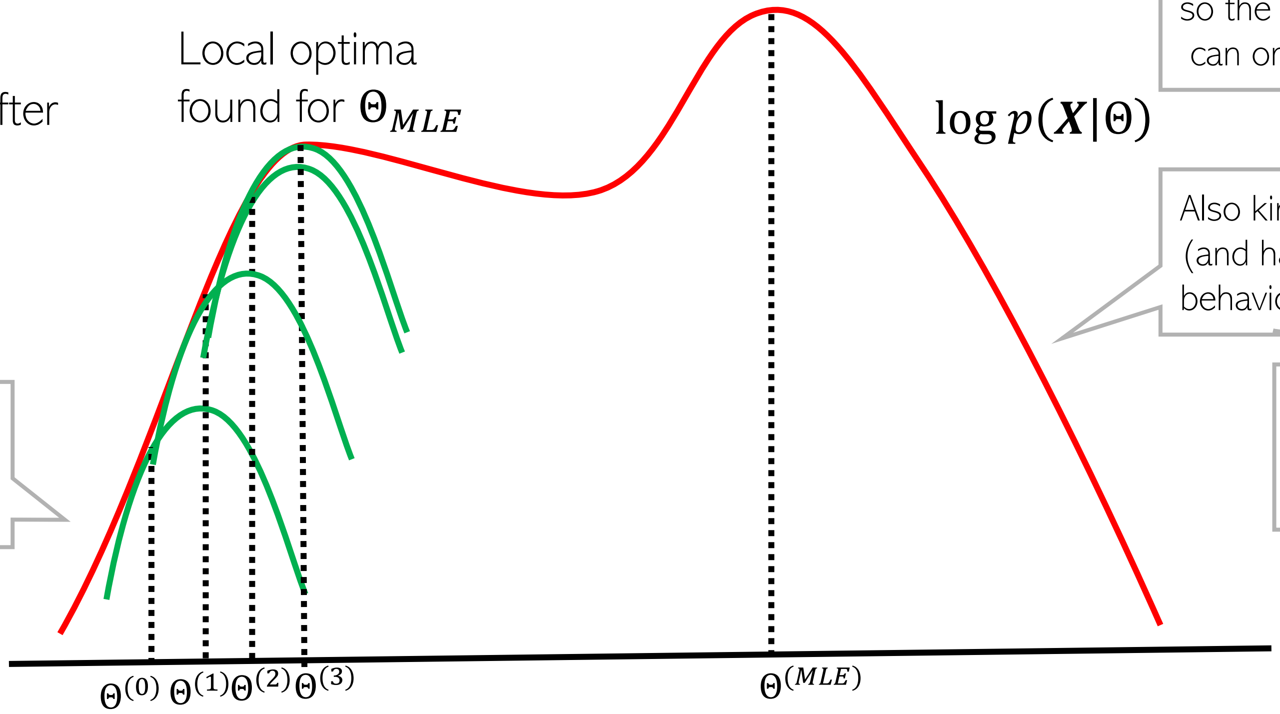
EM: An Illustration

Alternating between them until convergence to some local optima

- As we saw, EM maximizes the lower bound $\mathcal{L}(q, \Theta)$ to get $\Theta^{(MLE)}$ in two steps
- Step 1 sets $\hat{q} = p(\boldsymbol{Z}|\Theta, \boldsymbol{X})$ using Θ^{old}. KL becomes zero and $\mathcal{L}(\hat{q}, \Theta^{\text{old}}) = \log p(\boldsymbol{X}|\Theta^{\text{old}})$
- Step 2 maximizes $\mathcal{L}(\hat{q}, \Theta)$ w.r.t. Θ which gives the new Θ.

Note that Θ only changes in Step 2 so the objective $\log p(\boldsymbol{X}|\Theta)$ can only change in Step 2

Green curve: $\mathcal{L}(\hat{q}, \Theta)$ after setting q to $\hat{q}$



Good initialization matters; otherwise would converge to a poor local optima

Also kind of similar to Newton's method (and has second order like convergence behavior like Newton's in some cases)

Unlike Newton's method, we don't construct and optimize a quadratic approximation, but a lower bound

Even though original MLE problem $\text{argmax}_\Theta \log p(\boldsymbol{X}|\Theta)$ could be solved using gradient methods, EM often works faster and has cleaner updates

The EM Algorithm in its general form..

- Maximization of $\mathcal{L}(q, \Theta)$ w.r.t. q and Θ gives the EM algorithm (Dempster, Laird, Rubin, 1977)

The EM Algorithm

1. Initialize Θ as $\Theta^{(0)}$, set $t = 1$
2. Step 1: Compute posterior of latent variables given current parameters $\Theta^{(t-1)}$

$$p(\boldsymbol{z}_n^{(t)}|\boldsymbol{x}_n, \Theta^{(t-1)}) = \frac{p(\boldsymbol{z}_n^{(t)}|\Theta^{(t-1)})p(\boldsymbol{x}_n|\boldsymbol{z}_n^{(t)}, \Theta^{(t-1)})}{p(\boldsymbol{x}_n|\Theta^{(t-1)})} \propto \text{prior} \times \text{likelihood}$$

3. Step 2: Now maximize the expected complete data log-likelihood w.r.t. Θ

$$\Theta^{(t)} = \arg\max_{\Theta} \mathcal{Q}(\Theta, \Theta^{(t-1)}) = \arg\max_{\Theta} \sum_{n=1}^{N} \mathbb{E}_{p(\boldsymbol{z}_n^{(t)}|\boldsymbol{x}_n, \Theta^{(t-1)})}[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n^{(t)}|\Theta)]$$

4. If not yet converged, set $t = t + 1$ and go to step 2.

- Note: If we can take the MAP estimate $\hat{z}_n$ of z_n (not full posterior) in Step 2 and maximize the CLL in Step 3 using that, i.e., do $\operatorname{argmax}_{\Theta} \sum_{n=1}^{N} \left[\log p\left(x_n, \hat{z}_n^{(t)} \middle| \Theta\right)\right]$ this will be ALT-OPT

The Expected CLL

- Expected CLL in EM is given by (assume observations are i.i.d.)

$$\begin{aligned}\mathcal{Q}(\Theta, \Theta^{old}) &= \sum_{n=1}^{N} \mathbb{E}_{p(\boldsymbol{z}_n|\boldsymbol{x}_n,\Theta^{old})}[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n|\Theta)] \\ &= \sum_{n=1}^{N} \mathbb{E}_{p(\boldsymbol{z}_n|\boldsymbol{x}_n,\Theta^{old})}[\log p(\boldsymbol{x}_n|\boldsymbol{z}_n, \Theta) + \log p(\boldsymbol{z}_n|\Theta)]\end{aligned}$$

Was indeed the case of GMM: $p(\boldsymbol{z}_n|\Theta)$ was multinoulli, $p(\boldsymbol{x}_n|\boldsymbol{z}_n,\Theta)$ was Gaussian

- If $p(\boldsymbol{z}_n|\Theta)$ and $p(\boldsymbol{x}_n|\boldsymbol{z}_n,\Theta)$ are exponential family distributions, then $\mathcal{Q}(\Theta, \Theta^{\text{old}})$ has a very simple form

- In resulting expressions, replace terms containing $\boldsymbol{z}_n$'s by their respective expectations, e.g.,
  - $\boldsymbol{z}_n$ replaced by $\mathbb{E}_{p(\boldsymbol{z}_n|\boldsymbol{x}_n,\widehat{\Theta})}[\boldsymbol{z}_n]$
  - $\boldsymbol{z}_n\boldsymbol{z}_n^\top$ replaced by $\mathbb{E}_{p(\boldsymbol{z}_n|\boldsymbol{x}_n,\widehat{\Theta})}[\boldsymbol{z}_n\boldsymbol{z}_n^\top]$

- However, in some LVMs, these expectations are intractable to compute and need to be approximated (beyond the score of CS771)

Another LVM: Probabilistic PCA (PPCA)

- Assume $\boldsymbol{x}_n \in \mathbb{R}^D$ as a linear mapping of a latent var $\boldsymbol{z}_n \in \mathbb{R}^K$ + Gaussian noise

A "reverse" generative way of thinking about PCA (low-dim $\boldsymbol{z}_n$ generating high-dim $\boldsymbol{x}_n$

$D \times 1$ offset/bias

$D \times K$ matrix

Drawn from a zero-mean D-dim Gaussian $\mathcal{N}(\mathbf{0}, \sigma^2 I_D)$

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

This linear mapping can be replaced by more powerful nonlinear mapping of the form $\boldsymbol{x}_n = f(\boldsymbol{z}_n) + \boldsymbol{\epsilon}_n$ where f can be modeled using a deep neural net (e.g., models like variational autoencoders)

- Equivalent to saying $p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{\mu}, \boldsymbol{W}, \sigma^2) = \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu} + \boldsymbol{W}\boldsymbol{z}_n, \sigma^2 I_D)$
- Assume a zero-mean K-dim Gaussian prior on $\boldsymbol{z}_n$, so $p(\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{z}_n|\mathbf{0}, I_K)$
- We would like to do MLE for $\Theta = (\boldsymbol{\mu}, \boldsymbol{W}, \boldsymbol{\sigma}^2)$
- ILL for this model $p(\boldsymbol{x}_n|\boldsymbol{\mu}, \boldsymbol{W}, \sigma^2)$ is also a Gaussian (thanks to Gaussian properties)

$$p(\boldsymbol{x}_n|\boldsymbol{\mu}, \boldsymbol{W}, \sigma^2) = \int p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{\mu}, \boldsymbol{W}, \sigma^2)p(\boldsymbol{z}_n)d\boldsymbol{z}_n = N(\boldsymbol{x}_n\,|\boldsymbol{\mu}, \boldsymbol{W}\boldsymbol{W}^\top + \sigma^2 I_D)$$

PRML 12.2.1

- Maximizing ILL w.r.t. $\Theta = (\boldsymbol{\mu}, \boldsymbol{W}, \boldsymbol{\sigma}^2)$ is possible but requires solving eig decomp. problem
- We can use ALT-OPT/EM to estimate $\Theta = (\boldsymbol{\mu}, \boldsymbol{W}, \boldsymbol{\sigma}^2)$ more efficiently without eig decomp.

Learning PPCA using EM

- Instead of maximizing the ILL $p(\boldsymbol{x}_n|\boldsymbol{\mu}, \boldsymbol{W}, \sigma^2) = N(\boldsymbol{x}_n\,|\boldsymbol{\mu}, \boldsymbol{W}\boldsymbol{W}^\top + \sigma^2 I_D)$, let's use ALT-OPT/EM
- EM will instead maximize expected CLL, with CLL (assume $\boldsymbol{\mu} = \mathbf{0}$) given by

$$\log p(X, Z|W, \sigma^2) = \log \prod_{n=1}^{N} p(\boldsymbol{x}_n, \boldsymbol{z}_n|\boldsymbol{W}, \sigma^2) = \log \prod_{n=1}^{N} p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{W}, \sigma^2)\, p(\boldsymbol{z}_n)$$

- Using $p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{W}, \sigma^2) = \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{W}\boldsymbol{z}_n, \sigma^2 I_D)$ and $p(\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{z}_n|\mathbf{0}, I_K)$

$$\text{CLL} = -\sum_{n=1}^{N}\left\{\frac{D}{2}\log\sigma^2 + \frac{1}{2\sigma^2}\|\boldsymbol{x}_n\|^2 - \frac{1}{\sigma^2}\boldsymbol{z}_n^\top \boldsymbol{W}^\top \boldsymbol{x}_n + \frac{1}{2\sigma^2}\text{trace}(\boldsymbol{z}_n\boldsymbol{z}_n^\top \boldsymbol{W}^\top \boldsymbol{W}) + \frac{1}{2}\text{trace}(\boldsymbol{z}_n\boldsymbol{z}_n^\top)\right\}$$

- Expected CLL will require $\mathbb{E}[\boldsymbol{z}_n]$ and $\mathbb{E}[\boldsymbol{z}_n\boldsymbol{z}_n^\top]$ w.r.t. conditional posterior of $\boldsymbol{z}_n$

Using the fact that $p(\boldsymbol{x}_n|\boldsymbol{z}_n)$ and $p(\boldsymbol{z}_n)$ are Gaussians and the CP is just the reverse conditional $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$ and must also be Gaussian

$$\begin{aligned} p(\mathbf{z}_n|\mathbf{x}_n, \mathbf{W}) &= \mathcal{N}(\mathbf{M}^{-1}\mathbf{W}^\top \mathbf{x}_n, \sigma^2\mathbf{M}^{-1}) \qquad \text{where } \mathbf{M} = \mathbf{W}^\top\mathbf{W} + \sigma^2\mathbf{I}_K \\ \mathbb{E}[\mathbf{z}_n] &= \mathbf{M}^{-1}\mathbf{W}^\top \mathbf{x}_n \\ \mathbb{E}[\mathbf{z}_n\mathbf{z}_n^\top] &= \mathbb{E}[\mathbf{z}_n]\mathbb{E}[\mathbf{z}_n]^\top + \text{cov}(\mathbf{z}_n) = \mathbb{E}[\mathbf{z}_n]\mathbb{E}[\mathbf{z}_n]^\top + \sigma^2\mathbf{M}^{-1} \end{aligned}$$

Learning PPCA using EM

- The EM algo for PPCA alternates between two steps
  - Compute CP of $\boldsymbol{z}_n$ given parameters $\boldsymbol{\Theta} = (\mathbf{W}, \sigma^2)$ and required expectatuions

$$
\begin{aligned}
p(\mathbf{z}_n|\mathbf{x}_n, \mathbf{W}) &= \mathcal{N}(\mathbf{M}^{-1}\mathbf{W}^\top \mathbf{x}_n, \sigma^2 \mathbf{M}^{-1}) \qquad \text{where } \mathbf{M} = \mathbf{W}^\top \mathbf{W} + \sigma^2 \mathbf{I}_K \\
\mathbb{E}[\mathbf{z}_n] &= \mathbf{M}^{-1}\mathbf{W}^\top \mathbf{x}_n \\
\mathbb{E}[\mathbf{z}_n \mathbf{z}_n^\top] &= \mathbb{E}[\mathbf{z}_n]\mathbb{E}[\mathbf{z}_n]^\top + \text{cov}(\mathbf{z}_n) = \mathbb{E}[\mathbf{z}_n]\mathbb{E}[\mathbf{z}_n]^\top + \sigma^2 \mathbf{M}^{-1}
\end{aligned}
$$

Note: This approach does not assume/ensure that $\boldsymbol{W}$ is orthonormal

  - Maximize the expected CLL $\mathbb{E}[\log p(\boldsymbol{X}, \boldsymbol{Z}|\mathbf{W}, \sigma^2)]$ w.r.t. $\boldsymbol{W}$ and $\boldsymbol{\sigma}^2$

$$
\mathbb{E}[\text{CLL}] = -\sum_{n=1}^{N} \left\{ \frac{D}{2} \log \sigma^2 + \frac{1}{2\sigma^2} ||\boldsymbol{x}_n||^2 - \frac{1}{\sigma^2} \mathbb{E}[\boldsymbol{z}_n^\top] \boldsymbol{W}^\top \boldsymbol{x}_n + \frac{1}{2\sigma^2} \text{trace}(\mathbb{E}[\boldsymbol{z}_n \boldsymbol{z}_n^\top] \boldsymbol{W}^\top \boldsymbol{W}) + \frac{1}{2} \text{trace}(\mathbb{E}[\boldsymbol{z}_n \boldsymbol{z}_n^\top]) \right\}
$$

Unlike standard PCA (non-probabilistic), no eigendecomposition needed to estimate $\boldsymbol{W}$

$$
\begin{aligned}
\mathbf{W}_{new} &= \left[\sum_{n=1}^{N} \mathbf{x}_n \mathbb{E}[\mathbf{z}_n]^\top\right] \left[\sum_{n=1}^{N} \mathbb{E}[\mathbf{z}_n \mathbf{z}_n^\top]\right]^{-1} \\
\sigma^2_{new} &= \frac{1}{ND} \sum_{n=1}^{N} \left\{ ||\mathbf{x}_n||^2 - 2\mathbb{E}[\mathbf{z}_n]^\top \mathbf{W}_{new}^\top \mathbf{x}_n + \text{tr}\left(\mathbb{E}[\mathbf{z}_n \mathbf{z}_n^\top] \mathbf{W}_{new}^\top \mathbf{W}_{new}\right) \right\}
\end{aligned}
$$

Note: setting $\boldsymbol{\sigma}^2 = \mathbf{0}$ makes it equivalent to standard PCA without orthonormality constraint, but EM is more efficient since no eigendecomposition is needed

- Will get ALT-OPT if we use mode of the CP as $\hat{\boldsymbol{z}}_n$ in the CLL

Generative Models can generate synthetic data!

- Once parameters $\Theta = (\boldsymbol{\mu}, \boldsymbol{W}, \boldsymbol{\sigma}^2)$ are learned, we can even generate new data, e.g.,
  - Generate a random $\boldsymbol{z}_n$ from $\mathcal{N}(\mathbf{0}, I_K)$
  - Generate $\boldsymbol{x}_n$ condition on $\boldsymbol{z}_n$ from $\mathcal{N}(\boldsymbol{\mu} + \boldsymbol{W}\boldsymbol{z}_n, \sigma^2 I_D)$

In addition to, of course, reducing the data dimensionality

(a) Training data (b) Random samples

Generated using a more sophisticated generative model, not PPCA (but similar in formulation)

Methods such as variational autoencoders, GAN, diffusion models, etc are based on similar ideas

EM: Some Comments

- Good initialization is important
- The E and M steps may not always be possible to perform exactly. Some reasons
  - CP of latent variables $p(\boldsymbol{Z}|\boldsymbol{X},\Theta)$ may not be easy to find and may require approx.
  - Even if $p(\boldsymbol{Z}|\boldsymbol{X},\Theta)$ is easy, expected CLL, i.e., $\mathbb{E}[\log p(\boldsymbol{X},\boldsymbol{Z}|\Theta)]$ may still not be tractable

Monte-Carlo EM

$$\mathbb{E}[\log p(\mathbf{X},\mathbf{Z}|\Theta)] = \int \log p(\mathbf{X},\mathbf{Z}|\Theta)p(\mathbf{Z}|\mathbf{X},\Theta)d\mathbf{Z}$$

..and may need to be approximated, e.g., using Monte-Carlo expectation

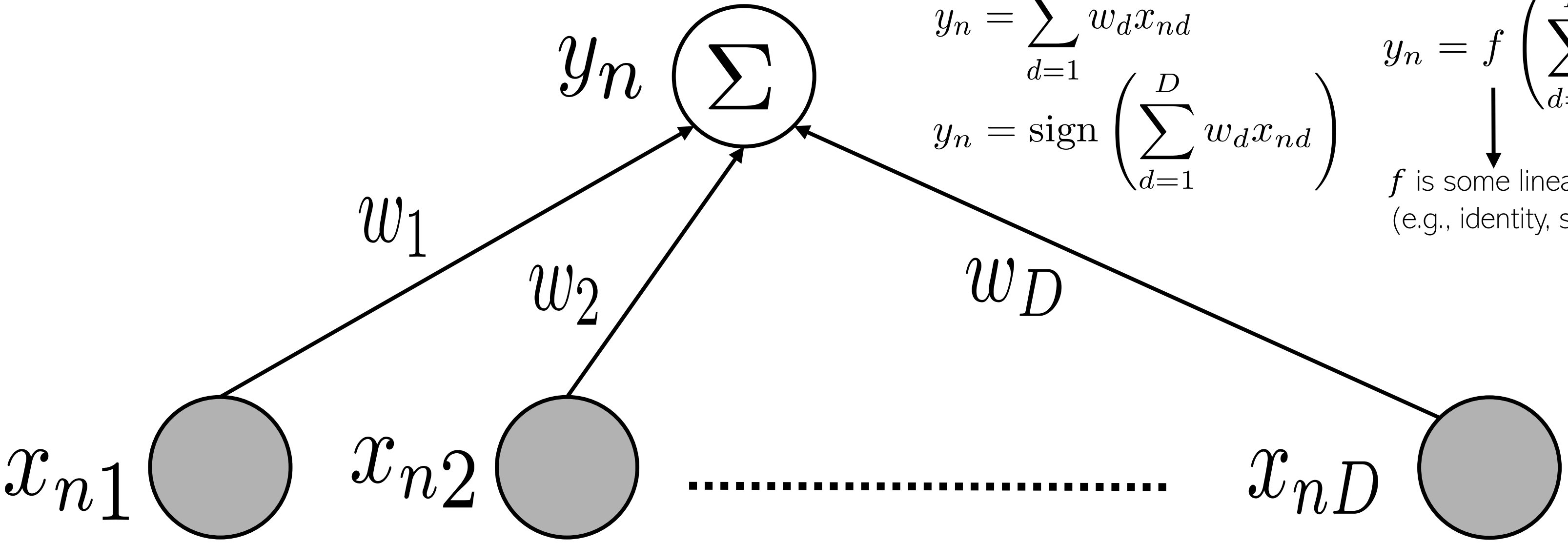
Gradient methods may still be needed for this step

  - Maximization of the expected CLL may not be possible in closed form
- EM works even if the M step is only solved approximately (Generalized EM)
- Other advanced probabilistic inference algorithms are based on ideas similar to EM
  - E.g., Variational Bayesian inference a.k.a. Variational Inference (VI)
- EM is also related to non-convex optimization algorithms Majorization-Maximization (MM)
  - MM maximizes a difficult-to-optimize objective function by iteratively constructing surrogate functions that are easier to maximize (in EM, the surrogate function was the CLL)

Deep Neural Nets

Limitation of Linear Models

- Linear models: Output produced by taking a linear combination of input features



$$y_n = \sum_{d=1}^{D} w_d x_{nd}$$

$$y_n = \text{sign}\left(\sum_{d=1}^{D} w_d x_{nd}\right)$$

$$y_n = f\left(\sum_{d=1}^{D} w_d x_{nd}\right)$$

f is some linear/nonlinear function (e.g., identity, sign, or sigmoid)

Linear regression, logistic regression, SVM, etc

- A basic unit of the form $y = f(\mathbf{w}^\top \mathbf{x})$ is known as the "Perceptron" (not to be confused with the Perceptron "algorithm", which learns a linear classification model)

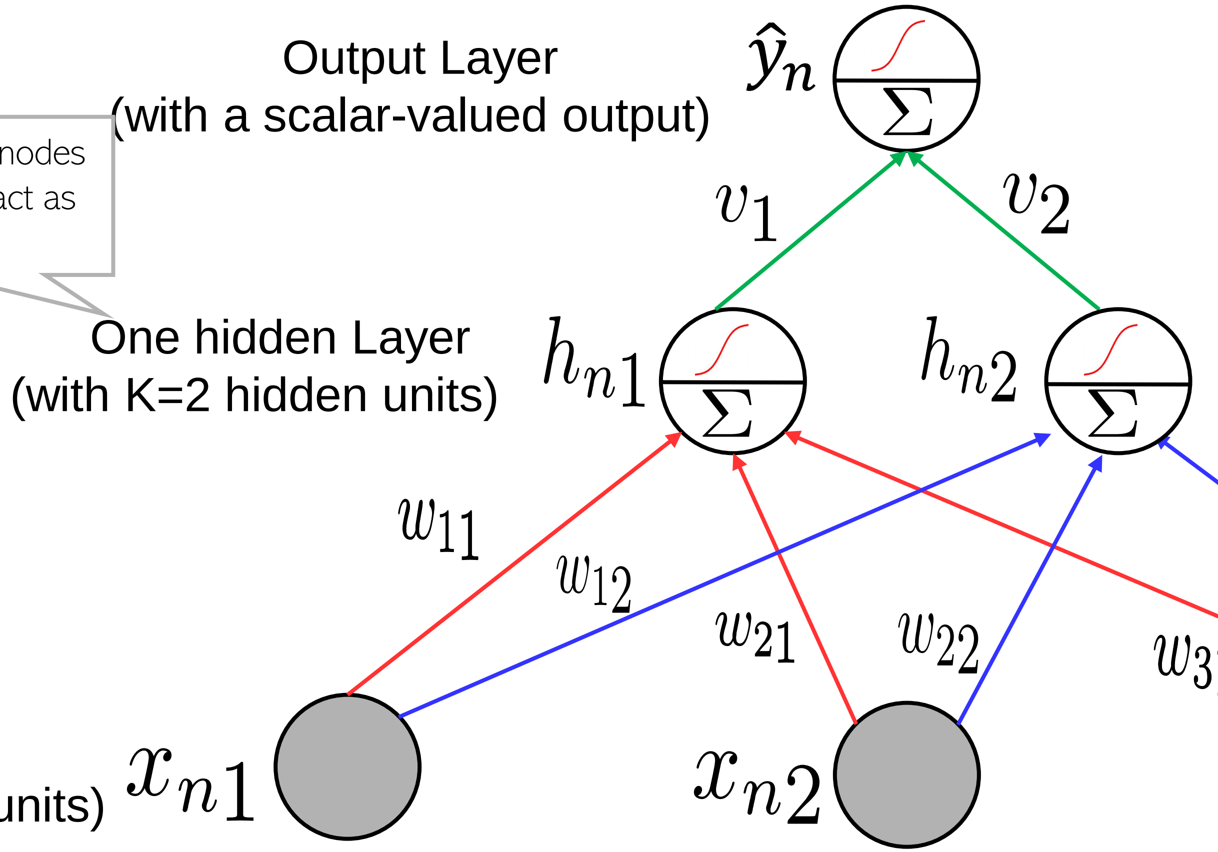
Although can kernelize to make them nonlinear

- This can't however learn nonlinear functions or nonlinear decision boundaries

Neural Networks: Multi-layer Perceptron (MLP)

- An MLP is a network containing several Perceptron units across many layers
- An MLP consists of an input layer, an output layer, and one or more hidden layers

Hidden layer units/nodes (a.k.a. "neurons") act as new features



$$y_n = o\left(\sum_{k=1}^{K} v_k h_{nk}\right)$$

Learnable weights

$$h_{nk} = g\left(\sum_{d=1}^{D} w_{dk} x_{nd}\right)$$

Can think of this model as a combination of two predictions h_{n1} and h_{n2} of two simple functions (red and blue weights)

The effective x to y mapping is nonlinear (will see justification shortly)

Input layer units/nodes denote the original features of input x_n

MLP is also called feedforward fully-connected network

Illustration: Neural Net with Single Hidden Layer

- Compute K pre-activations for each input $\boldsymbol{x}_n$

A linear model with learnable weight vec $\boldsymbol{w}_k$

$$z_{nk} = \boldsymbol{w}_k^\top \boldsymbol{x}_n = \sum_{d=1}^{D} w_{dk} x_{nd} \qquad (k = 1,2,\ldots,K)$$

- Apply nonlinear activation on each pre-act

Called a hidden unit

$$h_{nk} = g(z_{nk}) \qquad (k = 1,2,\ldots,K)$$

- Apply a linear model with $\boldsymbol{h}_n$ acting as features

A linear model with learnable weight vec $\boldsymbol{v}$

Score of the input

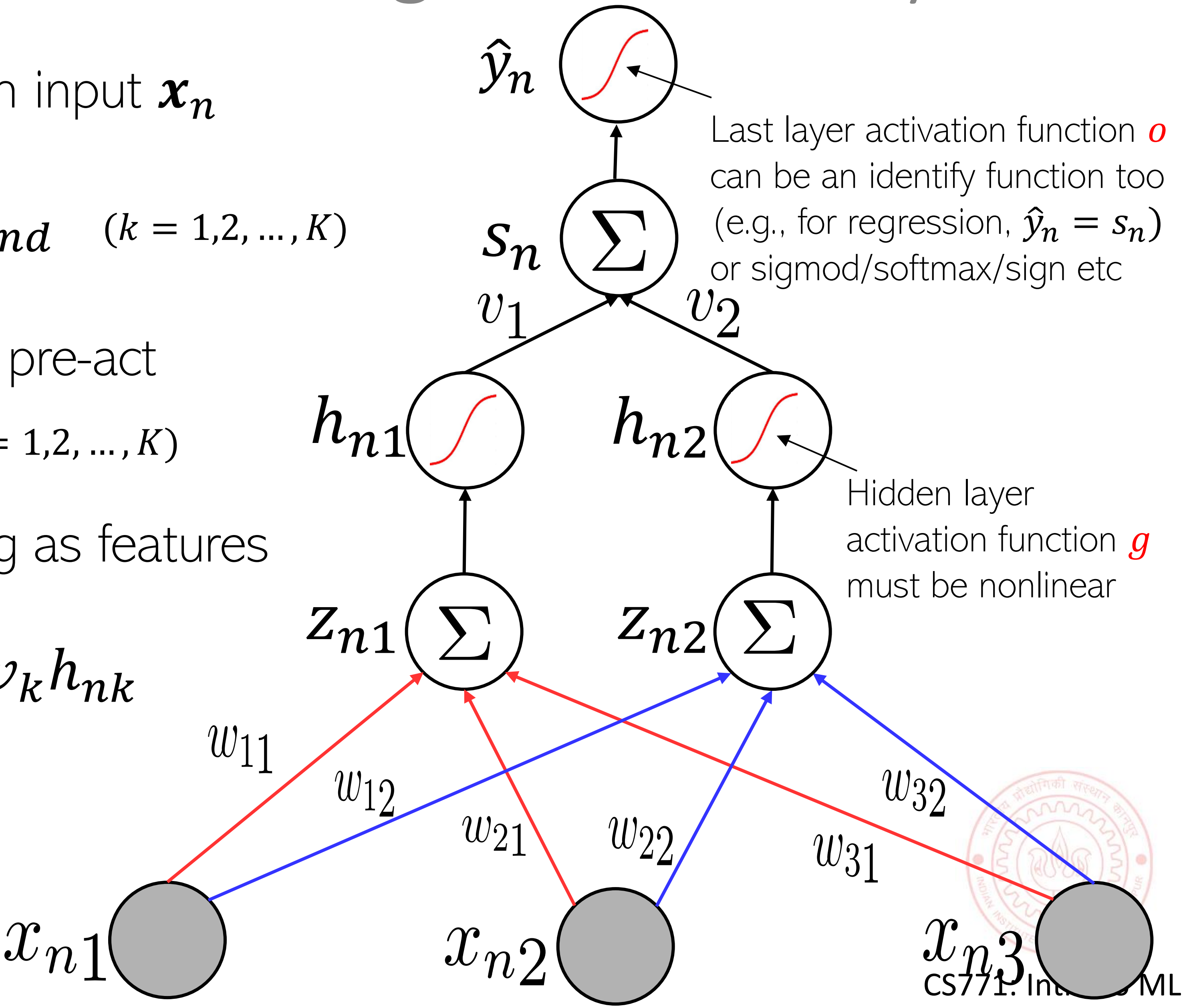
$$s_n = \boldsymbol{v}^\top \boldsymbol{h}_n = \sum_{k=1}^{K} v_k h_{nk}$$

- Finally, output is produced as

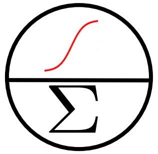
Score converted to the actual prediction

$$\hat{y}_n = o(s_n)$$

- Loss: $\mathcal{L}(\boldsymbol{W}, \boldsymbol{v}) = \sum_{n=1}^{N} \ell(y_n, \hat{y}_n)$

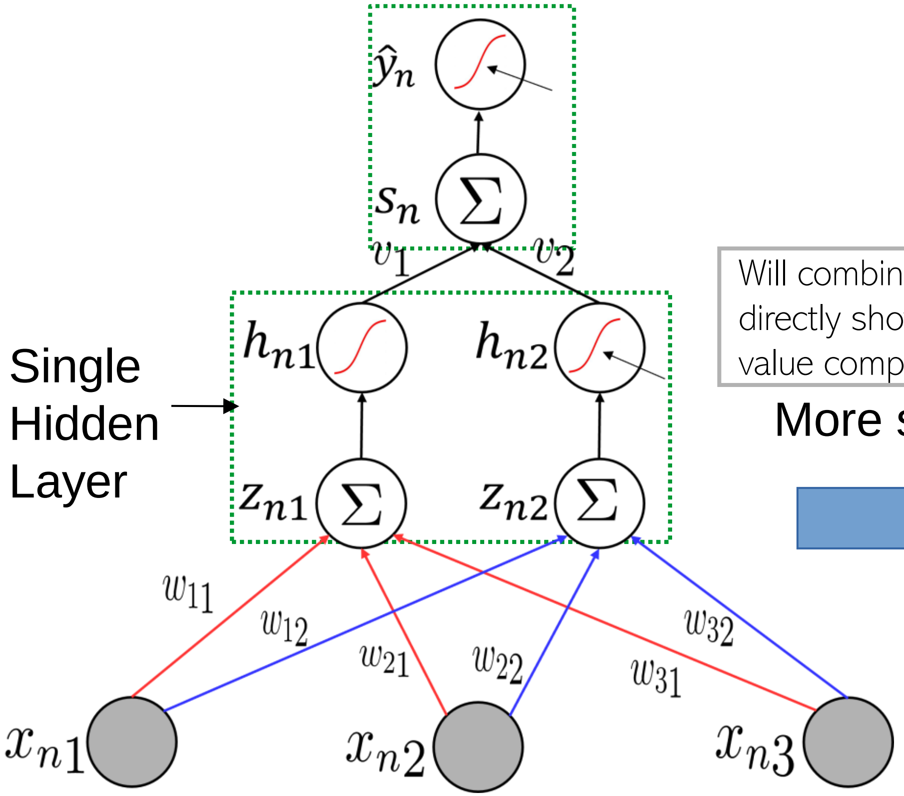


Neural Nets: A Compact Illustration



Will denote a linear combination of inputs followed by a nonlinear operation on the result

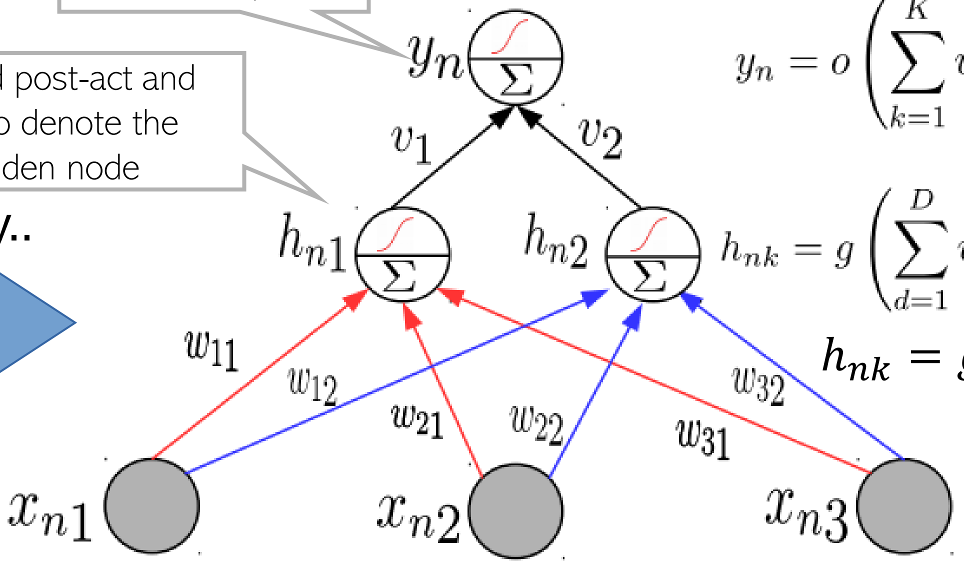
- Note: Hidden layer pre-act z_{nk} and post-act h_{nk} will be shown together for brevity



Will combine pre-act and post-act and directly show only h_{nk} to denote the value computed by a hidden node

More succinctly..

Will directly show the final output



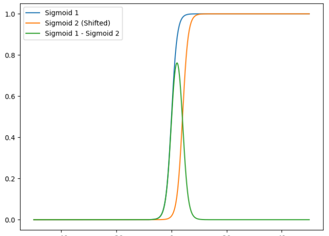
$$y_n = o\left(\sum_{k=1}^{K} v_k h_{nk}\right)$$

$$h_{nk} = g\left(\sum_{d=1}^{D} w_{dk} x_{nd}\right)$$

$$h_{nk} = g(\boldsymbol{w}_k^\top \boldsymbol{x}_n)$$

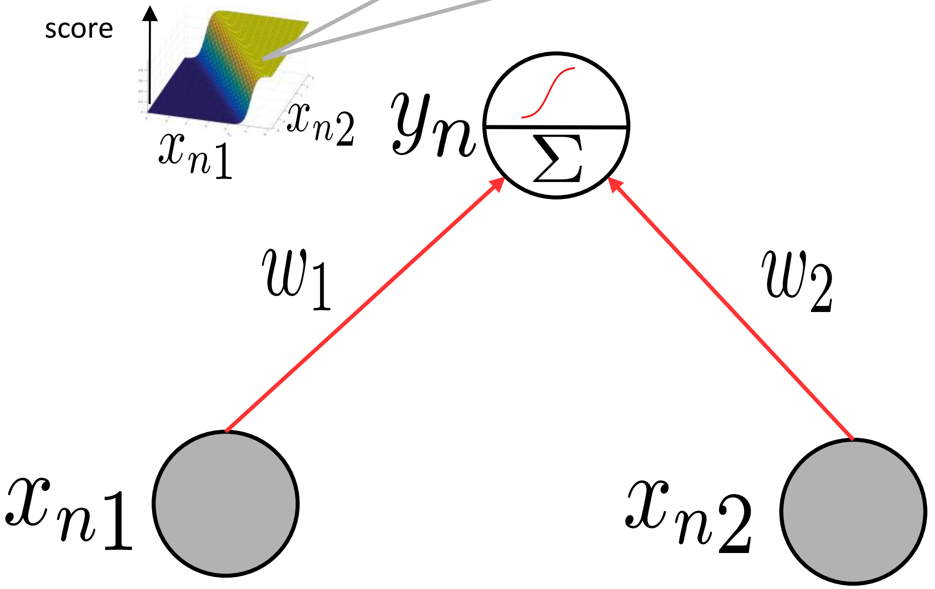
- Denoting $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_K]$, $\boldsymbol{w}_k \in \mathbb{R}^D$, $\boldsymbol{h}_n = g(\boldsymbol{W}^\top \boldsymbol{x}_n) \in \mathbb{R}^K$ $(K = 2, D = 3$ above). Note: g applied elementwise on pre-activation vector $\boldsymbol{z}_n = \boldsymbol{W}^\top \boldsymbol{x}_n$

MLP Can Learn Any Nonlinear Function



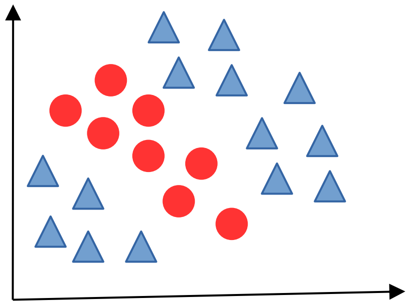
- An MLP can be seen as a composition of multiple linear models combined nonlinearly

Score monotonically increases. One-sided increase (not ideal for learning nonlinear decision boundaries). Just a linear model



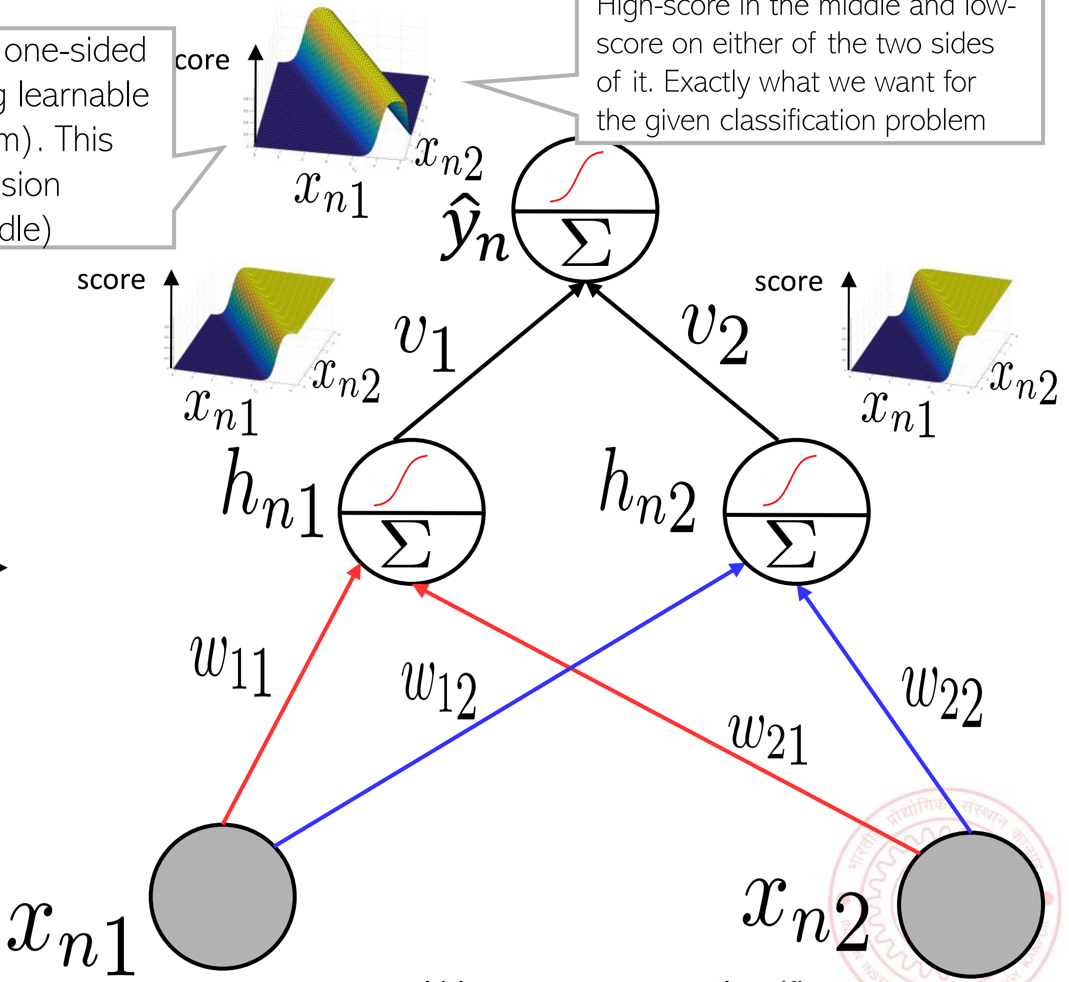
Standard Single “Perceptron” Classifier (no hidden units)

Obtained by composing the two one-sided increasing score functions (using learnable weights $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$ to “add“ them). This can now learn the nonlinear decision boundary (high score in the middle)



A nonlinear classification problem

High-score in the middle and low-score on either of the two sides of it. Exactly what we want for the given classification problem

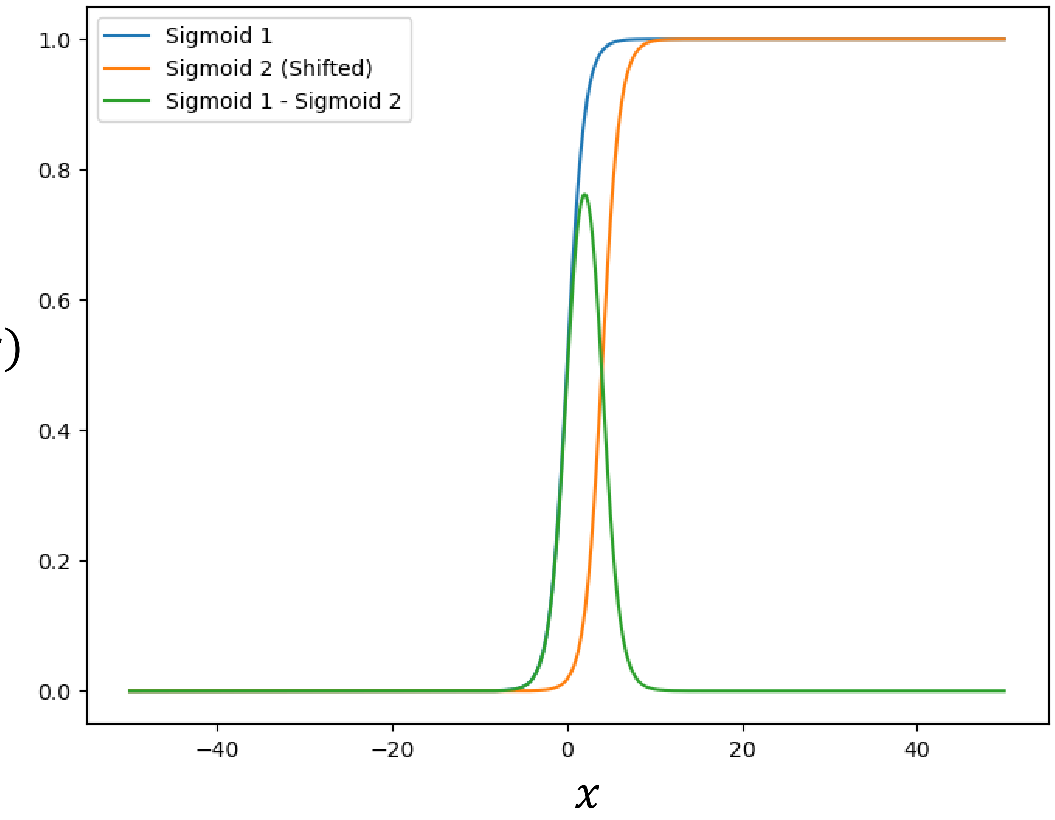


A Multi-layer Perceptron Classifier
(one hidden layer with 2 units)
Capable of learning nonlinear boundaries

A single hidden layer MLP with sufficiently large number of hidden units can approximate any function (Hornik, 1991)

Superposition of two linear models = Nonlinear model

$p(y = \text{red}|x)$



Two sigmoids (blue and orange) can be combined via a shift and a subtraction operation to result in a nonlinear separation boundary

Likewise, more than two sigmoids can be combined to learn even more sophisticated separation boundaries

Nonlinear separation boundary