STAT406- Methods of Statistical Learning Lecture 18

Matias Salibian-Barrera

UBC - Sep / Dec 2019

TO COMPLETE YOUR REGISTRATION, PLEASE TELL US WHETHER OR NOT THIS IMAGE CONTAINS A STOP SIGN:

NO YES

ANSWER QUICKLY—OUR SELF-DRIVING CAR IS ALMOST AT THE INTERSECTION.

SO MUCH OF "AI" IS JUST FIGURING OUT WAYS TO OFFLOAD WORK ONTO RANDOM STRANGERS.

Boosting

- Boosting is fitting an **additive model**
- ... using a **forward search algorithm**
- ... and a **specific loss function**

Boosting

- Think of classifiers of the form

$$G(x) = \sum_{j=1}^{K} \beta_j f(\mathbf{x}, \gamma_j)$$

where $f(\mathbf{x}, \gamma_j)$ are simple base classifiers (e.g. trees)

Boosting

- Given a data set $(y_i, \mathbf{x}_i)$, $i = 1, \ldots, n$

$$\min_G \sum_{i=1}^{n} L\left(y_i, G(\mathbf{x}_i)\right) =$$

$$= \min_{\beta,\gamma} \sum_{i=1}^{n} L\left(y_i, \sum_{j=1}^{K} \beta_j f(\mathbf{x}_i, \gamma_j)\right)$$

where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_K)'$ and $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_K)'$

Boosting

- Find approximate solutions sequentially
- Start with $f_0(\mathbf{x}) = 0$
- `for(j in 1:K)`
- Find

$$(\beta_j, \gamma_j) = \arg\min_{\beta,\gamma} \sum_{i=1}^{n} L\left(y_i, f_{j-1}(\mathbf{x}_i) + \beta\, f(\mathbf{x}_i, \gamma)\right)$$

- Let $f_j(\mathbf{x}) = f_{j-1}(\mathbf{x}) + \beta_j\, f(\mathbf{x}, \gamma_j)$

Boosting

- AdaBoost uses the following loss function

$$L\left(y, G(\mathbf{x})\right) \;=\; \exp\left(-y\, G\left(\mathbf{x}\right)\right)$$

- At the j-th iteration we have

$$\arg\min_{\beta,\gamma} \sum_{i=1}^{n} \exp\Big(-y_i \left(f_{j-1}(\mathbf{x}_i) + \beta f(\mathbf{x}_i, \gamma)\right)\Big)$$

$$\arg\min_{\beta,\gamma} \sum_{i=1}^{n} w_i^{(l-1)} \exp\Big(-\beta\, y_i f(\mathbf{x}_i, \gamma)\Big)$$

Boosting

- For any $\beta > 0$ the solution is the classifier $f(\mathbf{x}, \gamma)$ that minimizes

$$\sum_{y_i \neq f(\mathbf{x}_i, \gamma)} w_i^{(j-1)} = \sum_{i=1}^{n} w_i^{(j-1)} I\left(y_i \neq f(\mathbf{x}_i, \gamma)\right)$$

  which is a weighted missclassification error

Boosting

- Similarly we obtain

$$\beta_j = \frac{1}{2}\log\left(\frac{1-e_j}{e_j}\right)$$

where

$$e_j = \sum_{i=1}^{n} w_i^{(j-1)} I\left(y_i \neq f(\mathbf{x}_i, \gamma_j)\right) \Big/ \sum_{i=1}^{n} w_i^{(j-1)}$$

Boosting

- We then update

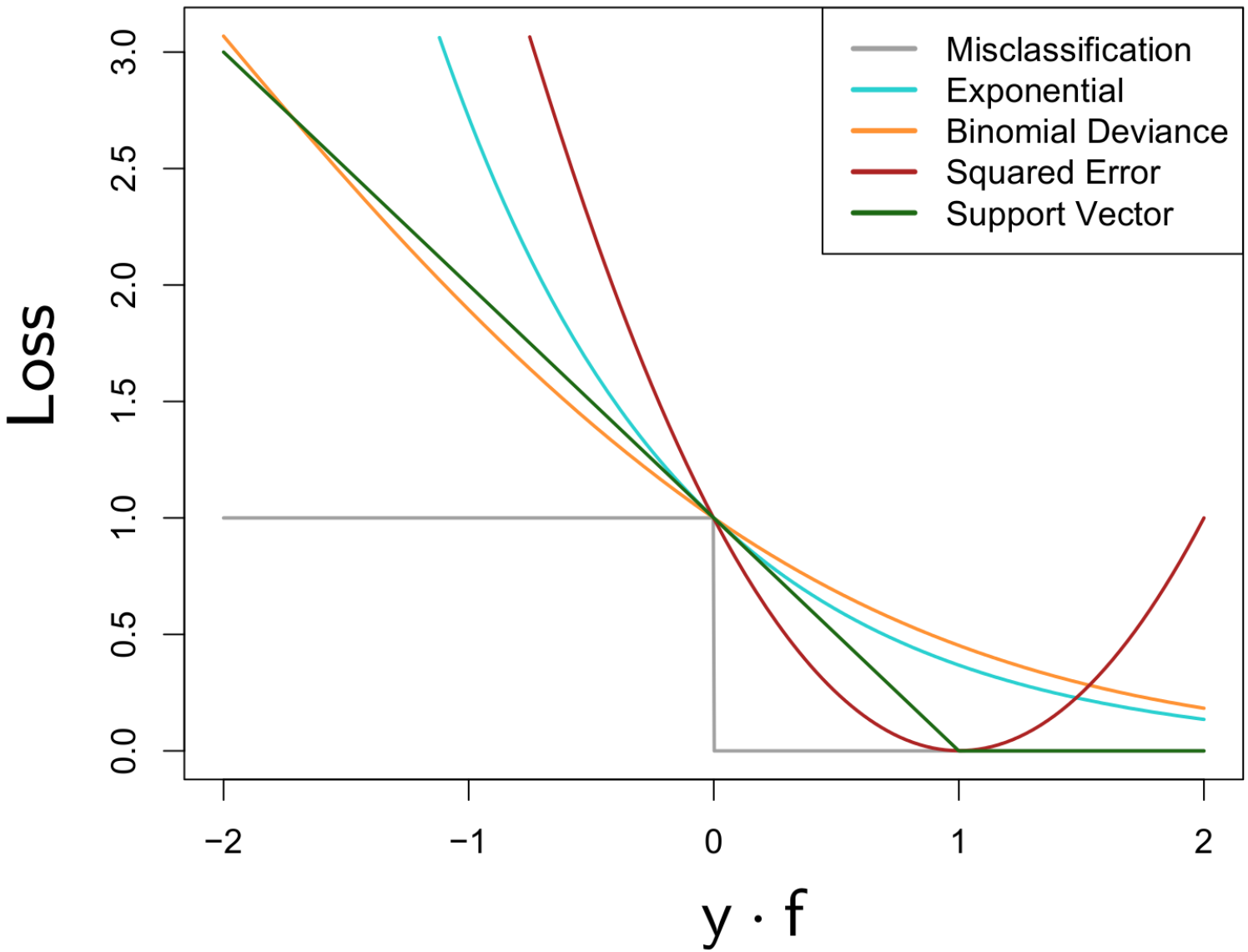
$$f_j(\mathbf{x}) \;=\; f_{j-1}(\mathbf{x}) + \beta_j\, f(\mathbf{x}, \gamma_j)$$

and hence

$$w_i^{(j+1)} \;=\; w_i^{(j)}\, \exp\left(-\beta_j\, y_i\, f(\mathbf{x}_i, \gamma_j)\right)$$
$$= \exp\left(-\beta_j\right)\, w_i^{(j)}\, \exp\left(-\alpha_j\, I\left(y_i \neq f(\mathbf{x}_i, \gamma_j)\right)\right)$$

where $\alpha_j = 2\,\beta_j$

Loss functions



Boosting

- The exponential loss penalizes misclassifications more than it approves correct classifications
- In particular, severe mistakes are very costly
- but the benefit of correct calls changes much more slowly

Boosting

- One can show that the "population" solution

$$\arg\min_{G(\mathbf{x})} E_{Y|\mathbf{X}=\mathbf{x}}\left[\exp\left(-Y\,G(\mathbf{x})\right)\right] =$$

$$= \frac{1}{2}\log\left(\frac{P(Y=1|\mathbf{X}=\mathbf{x})}{P(Y=-1|\mathbf{X}=\mathbf{x})}\right)$$

- The deviance loss also has the same "target" solution but grows slower - (so what?)

Boosting

- The shape of the exponential loss means that even if we have perfect classification for the training data, the objective function

$$\frac{1}{n}\sum_{i=1}^{n} L\left(y_i, G(\mathbf{x}_i)\right)$$

  may not have reached its minimum

- Thus the iterations continue...

Boosting

- Since we know what this method is estimating

$$\frac{1}{2}\log\left(\frac{P(Y=1|\mathbf{X}=\mathbf{x})}{P(Y=-1|\mathbf{X}=\mathbf{x})}\right)$$

... and we know what type of functions is attempting to use

$$\sum_{j=1}^{K}\beta_j f(\mathbf{x},\gamma_j) \approx \frac{1}{2}\log\left(\frac{P(Y=1|\mathbf{X}=\mathbf{x})}{P(Y=-1|\mathbf{X}=\mathbf{x})}\right)$$

Boosting

- ... we can understand when it works and when it may not work
- Note that the class of base classifiers $f(\mathbf{x}, \gamma)$ determines the type of log odds ratio we can model
- In particular, when using trees, the number of leaves (terminal nodes) determines the degree of interaction among the features that it may be able to capture

Gradient Boosting

- In many situations we want to find

$$\hat{f} = \arg\min_f \sum_{i=1}^{n} L\left(y_i, f(\mathbf{x}_i)\right)$$

which is similar to finding

$$\hat{\mathbf{h}} = \arg\min_{\mathbf{h}\in\mathbb{R}^n} \sum_{i=1}^{n} L\left(y_i, h_i\right)$$

$$\mathbf{h} = \begin{pmatrix} h_1 \\ \vdots \\ h_n \end{pmatrix} \quad \leftrightarrow \quad \begin{pmatrix} f(\mathbf{x}_1) \\ \vdots \\ f(\mathbf{x}_n) \end{pmatrix}$$

Gradient Boosting

- The problem is

$$\min_{\mathbf{h}\in\mathbb{R}^n} G(\mathbf{h}) = \min_{\mathbf{h}\in\mathbb{R}^n} \sum_{i=1}^{n} L(y_i, h_i)$$

- Many numerical optimization methods compute $\hat{\mathbf{h}}$ iteratively

$$\hat{\mathbf{h}} = \sum_{\ell=1}^{K} \mathbf{b}_\ell$$

where $\mathbf{b}_\ell \in \mathbb{R}^n$.

Gradient Boosting

- For example, gradient descent methods

$$\mathbf{b}_\ell = -\lambda_\ell \nabla G(\mathbf{h})|_{\mathbf{h}=\mathbf{h}_{\ell-1}}$$

where

$$\mathbf{h}_{\ell-1} = \sum_{j=1}^{\ell-1} \mathbf{b}_j$$

and λ_ℓ is the step size

Gradient Boosting

- We want $\hat{\mathbf{h}}$ to be based on a function $\hat{f}$

$$\sum_{\ell=1}^{K} \mathbf{b}_\ell = \hat{\mathbf{h}} = \begin{pmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{pmatrix} = \begin{pmatrix} \hat{f}(\mathbf{x}_1) \\ \hat{f}(\mathbf{x}_2) \\ \vdots \\ \hat{f}(\mathbf{x}_n) \end{pmatrix}$$

... so that we can use $\hat{f}$ on new $\mathbf{x}$'s

Gradient Boosting

- This suggests that each update

$$\mathbf{b}_\ell = -\lambda_\ell \nabla G(\mathbf{h})|_{\mathbf{h}=\mathbf{h}_{\ell-1}}$$

be approximated by a function instead

$$\mathbf{b}_\ell = \begin{pmatrix} b_{\ell,1} \\ b_{\ell,2} \\ \vdots \\ b_{\ell,n} \end{pmatrix} \approx \begin{pmatrix} T(\mathbf{x}_1, \theta_\ell) \\ T(\mathbf{x}_2, \theta_\ell) \\ \vdots \\ T(\mathbf{x}_n, \theta_\ell) \end{pmatrix}$$

Gradient Boosting

- And then, our $\hat{f}$ would be

$$\hat{f}(\cdot) = \sum_{\ell=1}^{K} -\lambda_\ell T(\cdot, \boldsymbol{\theta}_\ell)$$

Gradient Boosting

- At each step

$$\hat{\theta}_\ell = \arg\min_{\Theta} \sum_{i=1}^{n} \left(g_{i,\ell} - T\left(\mathbf{x}_i, \Theta\right)\right)^2$$

where

$$g_{i,\ell} = -\frac{\partial G}{\partial h_i}(\mathbf{h})\Big|_{\mathbf{h}=\mathbf{h}_{\ell-1}}$$

and $T(\mathbf{x}, \cdot)$ is a family of predictors

Gradient Boosting

- As with gradient descent, we can fine-tune $T(\mathbf{x}, \hat{\theta}_\ell)$
- If $T(\cdot, \theta)$ is a tree, we can use only the regions and adjust their values with respect to G
- Find the step size optimizing with respect to G
- Maybe use a shrinkage / decay factor

Gradient Tree Boosting

For, example, when

$$G(\mathbf{h}) = \sum_{i=1}^{n} L\left(y_i, h_i\right)$$

and $T(\cdot, \theta)$ are regression trees...

Gradient Tree Boosting

- Initialize

$$\hat{\gamma}_0 = \arg\min_{\gamma \in \mathbb{R}} \sum_{i=1}^{n} L(y_i, \gamma) \ , \quad \hat{\mathbf{h}}_0 = (\hat{\gamma}_0, \ldots, \hat{\gamma}_0)^\top$$

- For $j = 1, 2, ..., K$
  - Let

$$g_{i,j} = - \left. \frac{\partial L(y_i, h)}{\partial h} \right|_{\mathbf{h}=\mathbf{h}_{j-1}(\mathbf{x}_i)} \qquad 1 \leq i \leq n$$

  - Fit a regression tree to $g_{1,j}$, $g_{2,j}$, ..., $g_{n,j}$, obtain regions $R_{1,j}$, ..., $R_{M_j,j}$

Gradient Tree Boosting

- Continue "For $j = 1, 2, ..., K$"
  - For each region $R_{s,j}$, $1 \leq s \leq M_j$

$$\hat{\gamma}_{s,j} = \arg\min_{\gamma} \sum_{\mathbf{x}_i \in R_{s,j}} L(y_i, \mathbf{h}_{j-1}(\mathbf{x}_i) + \gamma)$$

  - Let

$$\hat{\mathbf{h}}_j(\cdot) = \hat{\mathbf{h}}_{j-1}(\cdot) + \sum_{s=1}^{M_j} \hat{\gamma}_{s,j} \mathbf{I}\left(\cdot \in R_{s,j}\right)$$

Gradient Tree Boosting

- Shrinkage:

$$\mathbf{h}_j(\cdot) = \mathbf{h}_{j-1}(\cdot) + \tau \sum_{s=1}^{M_j} \gamma_{s,j} \mathbf{I}\left(\cdot \in R_{s,j}\right)$$

  where $\tau \in (0, 1)$

- τ is the "rate of learning"
- Small values of τ require more iterations (K)
- This approach works very well in practice. More work is needed.

Neural Networks

- Neural networks are flexible regression models
- Hybrid (or particular case) of non-parametric regression and projection-pursuit
- Versatile, computationally very costly and fragile

Neural Networks

- Fitting them can almost be considered an art...
- Recently they've made a come back under the umbrella of "deep learning"

`http://www.youtube.com/watch?v=VdIURAu1-aU`

`http://lmgtfy.com/?q=deep+learning`

Neural Networks

- Neural networks (NN) build flexible regression models
- They use an ordered sequence of unobserved units that are transformed linear combinations of units appearing in previous levels of the network
- We'll focus on single-layer NNs (to simplify the discussion)

Neural Networks

- Let $\mathbf{X} \in \mathbb{R}^p$ be a generic vector of explanatory variables
- Build a new set of explanatory variables

$$Z_m = \sigma\left(\alpha_{0,m} + \boldsymbol{\alpha}'_m \mathbf{X}\right), \quad m = 1, 2, \ldots, M$$

(These form the hidden layer)

Neural Networks

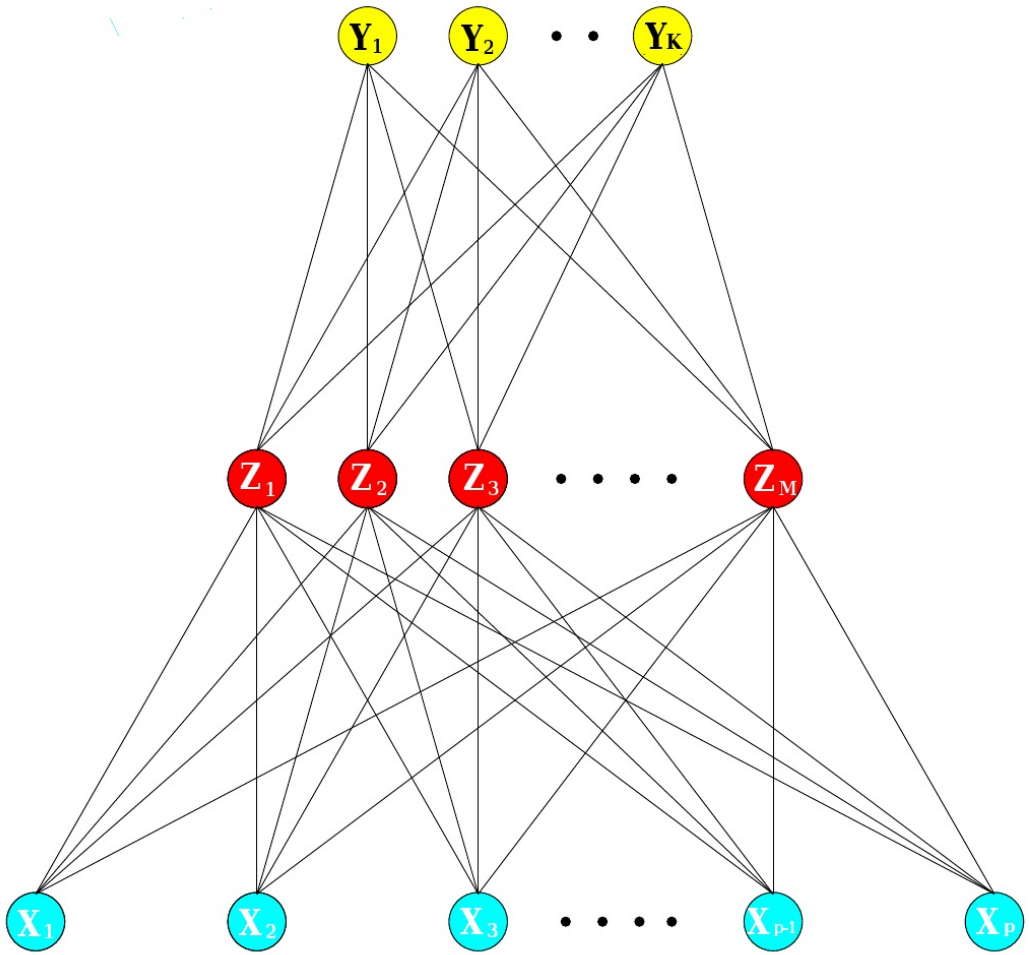
- The output variables are then linear combinations of the Z_m's

$$T_k = \beta_{0,k} + \beta_k' \mathbf{Z}, \quad k = 1, 2, \ldots, K$$

- The output variables may themselves be transformed again

$$f_k = f_k(\mathbf{X}) = g_k(\mathbf{T})$$

Neural Networks



Neural Networks

- The "activation function" is usually

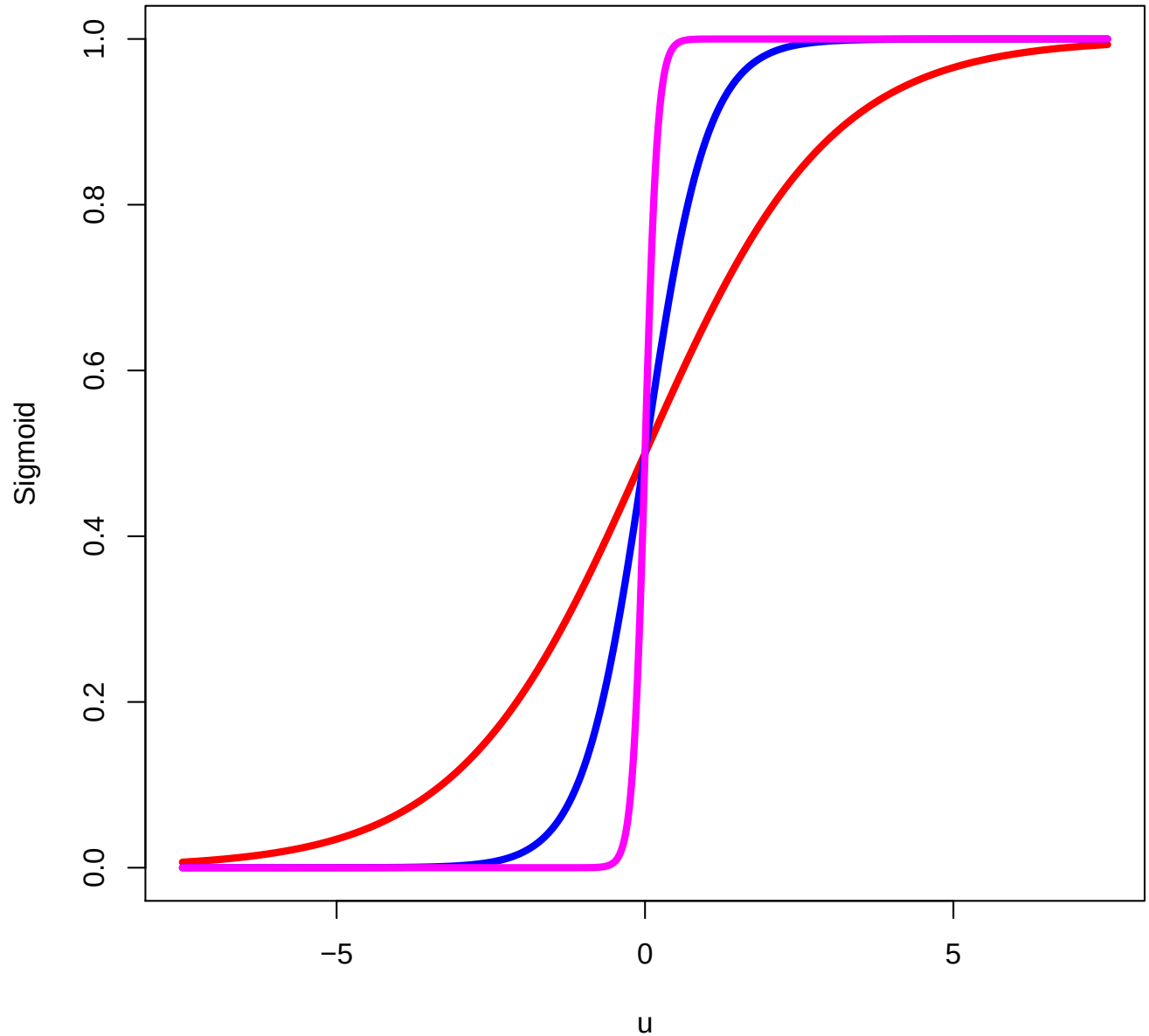
$$\sigma(u) = \frac{1}{1+\exp(-u)} = \frac{\exp(u)}{1+\exp(u)}$$

- For continuous responses we set $K = 1$ and $g_k(\mathbf{T}) = T_k$
- For categorical responses K is the number of classes and

$$g_k(\mathbf{T}) = \frac{\exp(T_k)}{\sum_{s=1}^{K}\exp(T_s)}$$

(aka "soft-max" outputs)

“Activation function”



Neural Networks

- How do we estimate ("learn") the α's and β's?
- For continuous responses

$$\min_{\alpha,\beta} \sum_{i=1}^{n} (y_i - f_1(\mathbf{x}_i))^2$$

$$\min_{\alpha,\beta} \sum_{i=1}^{n} \left(y_i - \beta_0 - \beta'\mathbf{Z}\right)^2$$

$$\min_{\alpha,\beta} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{s=1}^{M} \beta_s \sigma\left(\alpha_{0,s} + \alpha'_m \mathbf{x}_i\right) \right)^2$$

Neural Networks

- For categorical responses we use the deviance (cross-entropy) function

$$\min_{\alpha,\beta} \sum_{i=1}^{n}\sum_{k=1}^{K} y_{i,k} \log(f_k(\mathbf{x}_i))$$

Neural Networks

- Soft-max outputs and cross-entropy loss = logistic regression model on the variables in the hidden layer + MLE estimation
- By adding variables in the hidden layer the model becomes more flexible

Neural Networks

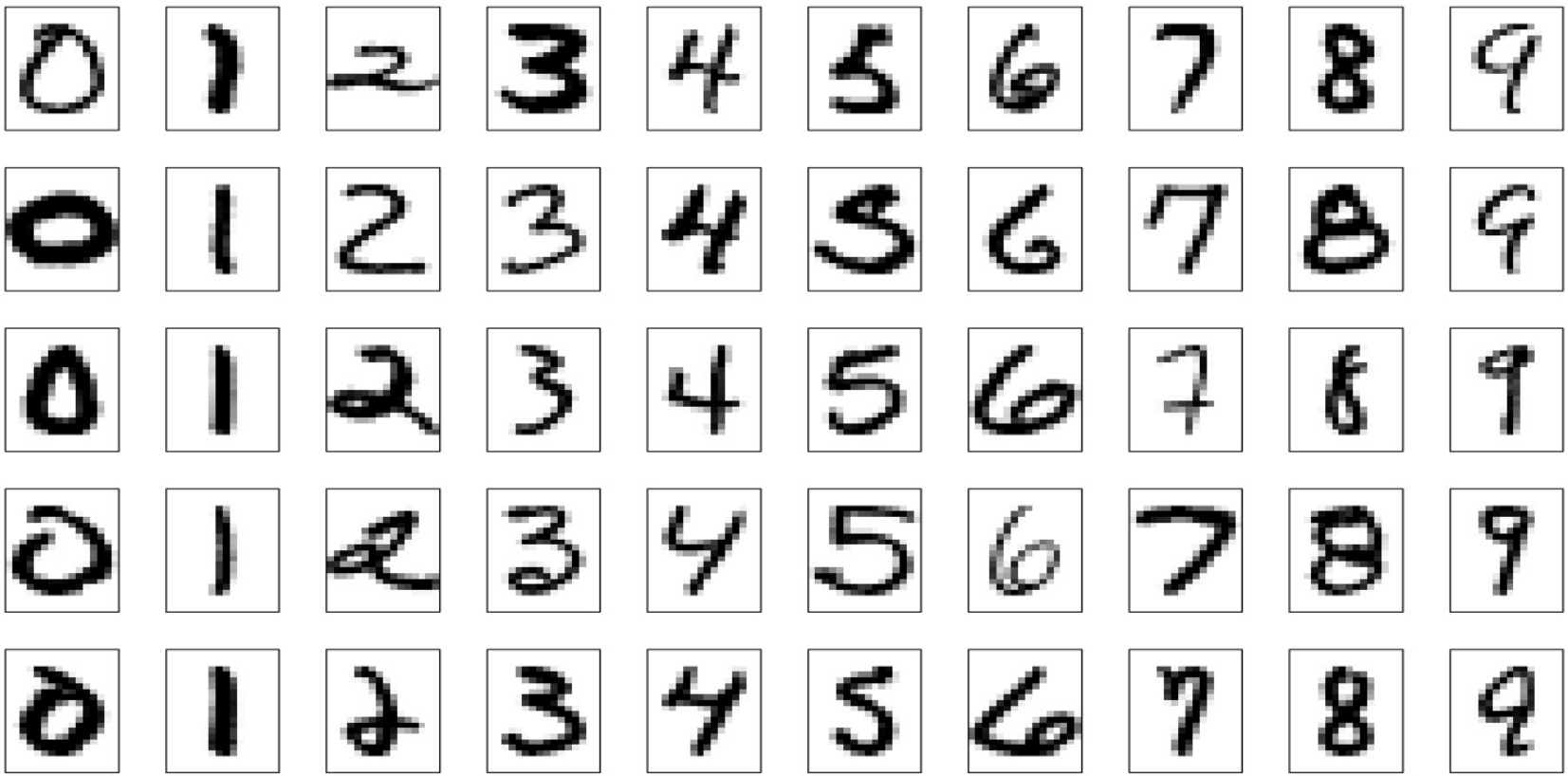
- Overfitting / regularization

$$\min_{\alpha,\beta} \sum_{i=1}^{n}\sum_{k=1}^{K} y_{i,k} \log(f_k(\mathbf{x}_i)) + \lambda P(\alpha,\beta)$$

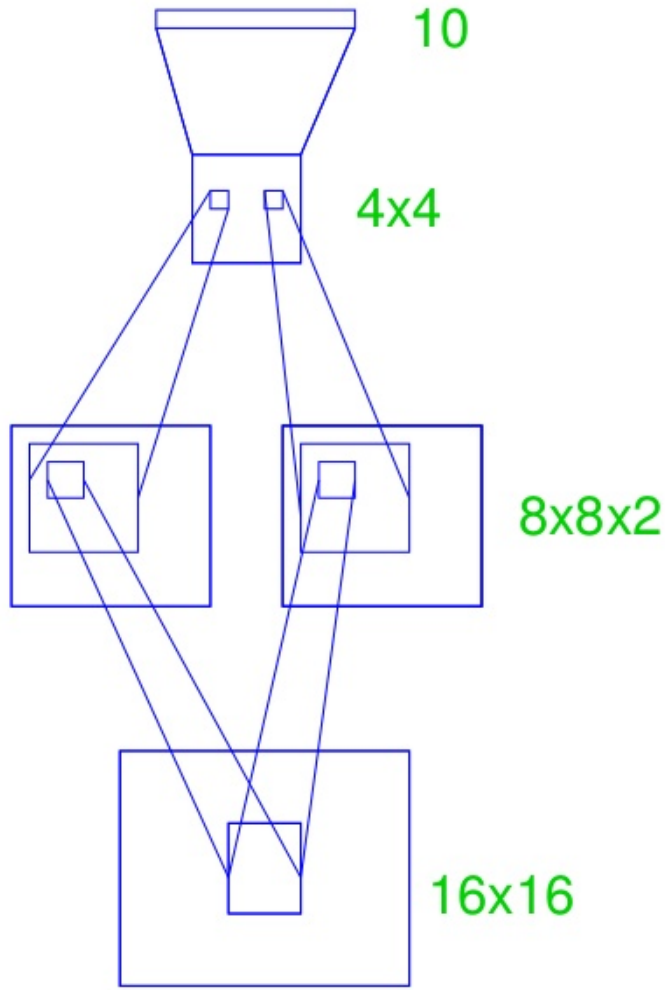
$$P(\alpha,\beta) = \|\alpha\|_2^2 + \|\beta\|_2^2$$

- A.K.A. as “weight decay”
- Since random starts are needed, the scale of the input variables becomes a potentially important issue

Feature “discovery”



Feature “discovery”



Net-4

Shared Weights

Neural Networks - ISOLET

```
> # ISOLET EXAMPLE
>
> # GitHub notes
>
```