ML for Econometricians
Deep Learning

Mattias Villani

Division of Statistics and Machine Learning
Department of Computer and Information Science
Linköping University

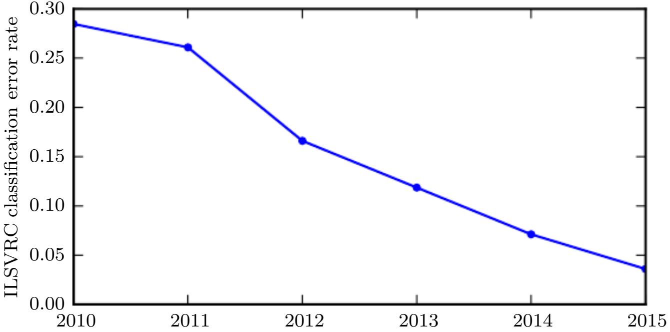
Lecture overview

- **Deep Learning - don't believe the hype?**
- **Deep Neural Networks**
- **Stochastic gradient descent**
- **Deep Learning in economic applications**

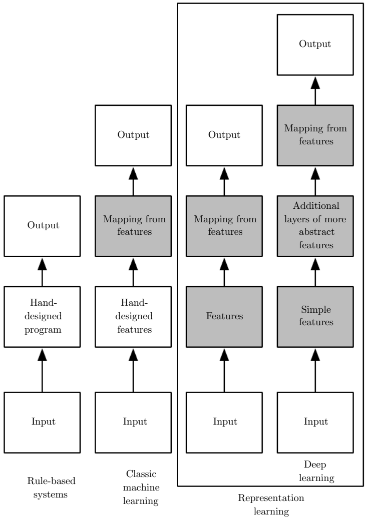
Deep Learning

- Deep (multi-layered) **neural networks** estimated with **stochastic gradient descent** and **back-propagation** to compute the gradient.
- **BIG HYPE!**
  - Reshaping the Machine Learning field.
  - ML conferences are booming.
  - Huge industry interest.
  - A lot of human capital allocated to Deep Learning.
- Very successful for **images** and **computer vision**. **Big boost in predictive performance** over previous methods.
- **Automatic feature construction**.
- The jury is still out for other types of data, for example text, or more structured statistical data.
- Why this comeback for neural networks?
  - Massive **cloud-sized datasets**. DNN need a lot of data to work well.
  - **GPU computing** makes it possible to go deep (many layers)
  - **Improved optimization** methods (kind of)
  - **Better** understanding of **network choices** (activation functions etc)
  - **Invariances** built in (e.g. convolutions for images)

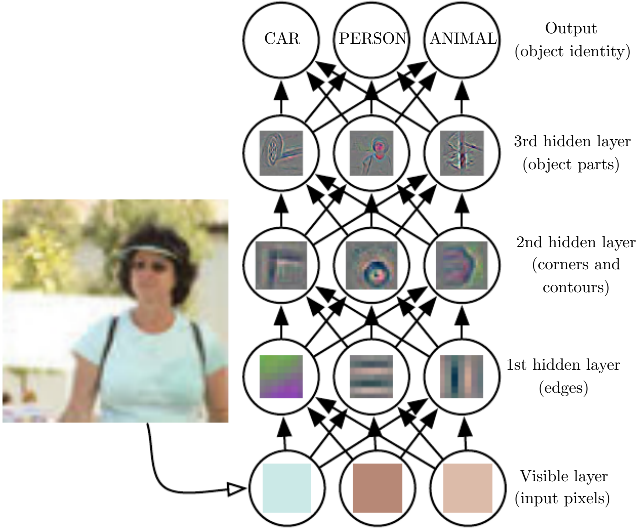
IMAGENET COMPETITIONS [1]



Automatic feature construction [1]



DNN and visual cortex layers [1]



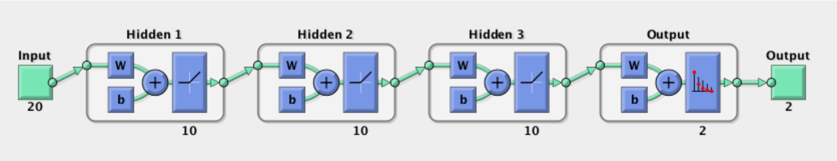
FEEDFORWARD NETWORKS

- Flow from inputs $\mathbf{x} \Rightarrow$ outputs $\mathbf{y}$ **without feedback loops**.
- Feedforward network with L **hidden layers**:

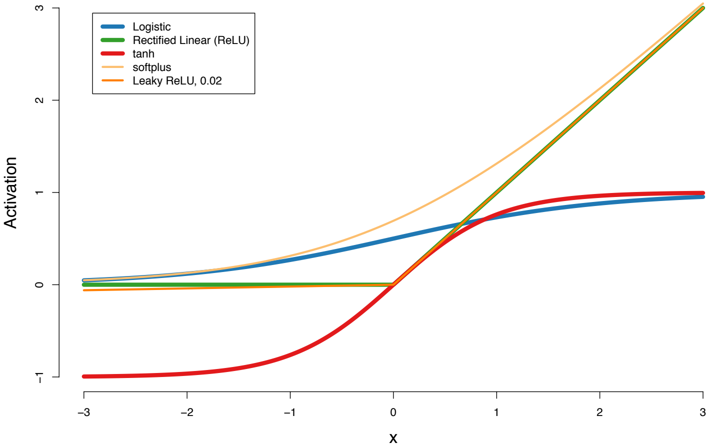
$$\mathbf{h}^{(1)} = g^{(1)}\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right)$$
$$\mathbf{h}^{(2)} = g^{(2)}\left(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}\right)$$
$$\vdots$$
$$\mathbf{h}^{(L)} = g^{(L)}\left(\mathbf{W}^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}\right)$$
$$\mathbf{y}|\mathbf{x} \sim p\left(\mathbf{y}|q(\mathbf{W}^{(L+1)}\mathbf{h}^{(L)} + \mathbf{b}^{(L+1)})\right)$$

where
  - $g^{(l)}$ is the (nonlinear) **activation function** at layer l, e.g. the logistic
  - $q(\cdot)$ is the **link function** for the output layer as in GLMs.
- **Network depth** - the number of layers.
- **Network width** - number of **neurons** (**units**) per layer.

DNN for binary classification with 20 inputs 3 layers each with 10 neurons



Activation functions



Gradient descent for Deep Neural Networks

- (scaled) Negative log-likelihood as **loss function**

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n} \log p(\mathbf{y}_i|\mathbf{x}_i, \theta)$$

- Gradient

$$g(\theta) = \nabla_\theta J(\theta) = -\frac{1}{n}\sum_{i=1}^{n} \nabla_\theta \log p(\mathbf{y}_i|\mathbf{x}_i, \theta)$$

- Gradient descent to find the minimum.

$$\theta^{(t)} = \theta^{(t-1)} - \epsilon g(\theta^{(t-1)})$$

- Gradient in DNN efficiently computed by **back-propagation** (chain rule + smart computations).

Gradient descent for Deep Neural Networks

- Gradient descent is still costly for large n.
- **Stochastic Gradient Decent** (**SGD**) uses an **unbiased estimator of the gradient**.
- Unbiased estimator of the gradient from a mini-batch of m observations selected by $\mathbf{u} = (u_1, ..., u_n)$

$$\hat{g}(\theta, \mathbf{u}) = -\frac{1}{m}\sum_{j=1}^{m} \nabla_\theta \log p(\mathbf{y}_j | \mathbf{x}_j, \theta)$$

- SGD

$$\theta^{(t)} = \theta^{(t-1)} - \epsilon_t \hat{g}(\theta^{(t-1)}, \mathbf{u}^{(t-1)})$$

- Will converge to local minima if [2]
  1. $\sum_{t=1}^{\infty} \epsilon_t = \infty$ and
  2. $\sum_{t=1}^{\infty} \epsilon_t^2 = 0$.
- Satisfied by for example $\epsilon_t = t^{-\kappa}$ for $\kappa \in (0.5, 1]$.

Stochastic Gradient Descent (SGD) algorithm

Algorithm 1: Stochastic Gradient Descent (SGD)

Input: data $\mathbf{y}$, likelihood function $p(\mathbf{y}|\theta)$, prior density $p(\theta)$, unbiased gradient estimator $\hat{g}(\theta, \mathbf{u})$, initial value $\theta^{(0)}$, random number generator for the subsampling indicators $\mathbf{u}$, subsample size m, step length sequence $\{\epsilon_t\}_{t\in\mathcal{T}}$, stopping criteria.

while *stopping criteria not met* **do**

  generate minibatch $\mathbf{u}^{(t)} \sim p(\mathbf{u})$

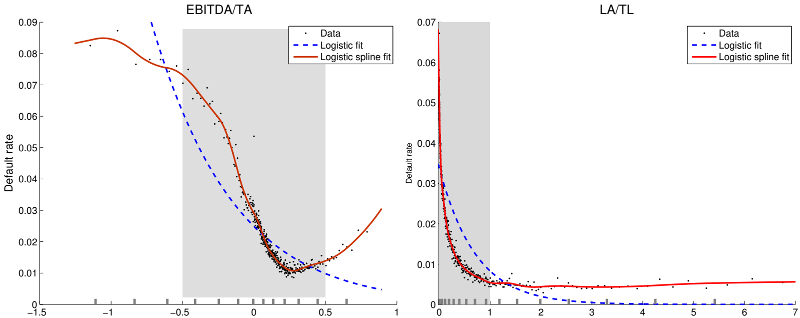
  set $\theta^{(t)} = \theta^{(t-1)} - \epsilon_t \hat{g}\left(\theta^{(t-1)}, \mathbf{u}^{(t)}\right)$

end

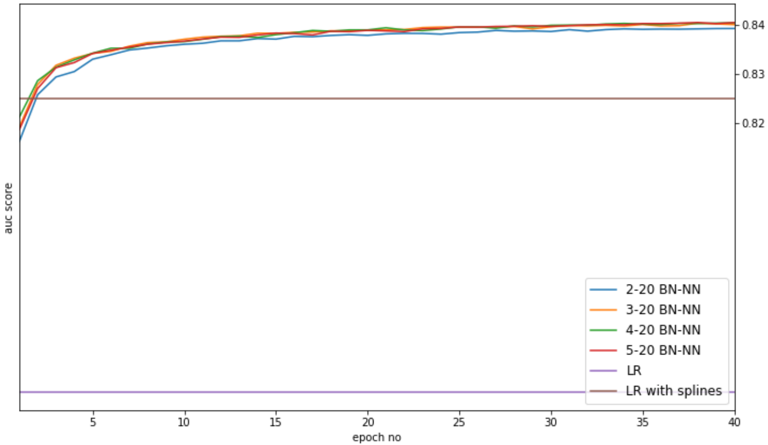
Output: terminal value $\theta^{(t_{\text{end}})}$

Deep learning for firm bankruptcy prediction

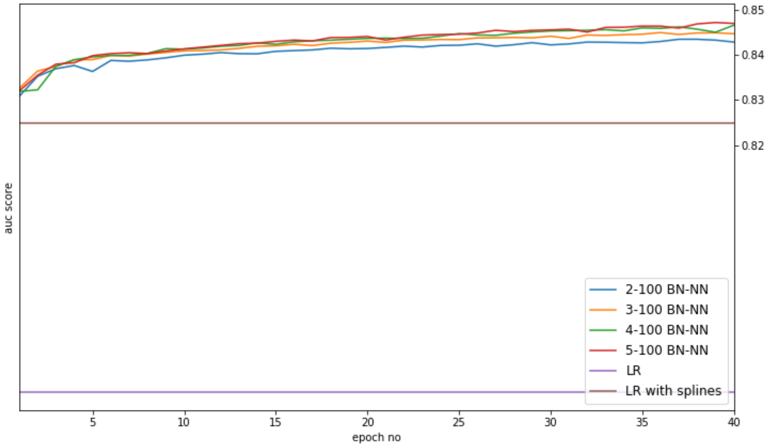
- Firm bankruptcy for Swedish firms.
  - $n = 4.7$ million observations
  - logistic regression using 8 covariates:
    - financial ratios (profits/assets, liquidity/debt etc)
    - macro variables (interest rate and GDP).
  - nonlinear: additive splines improve forecasting performance [3]



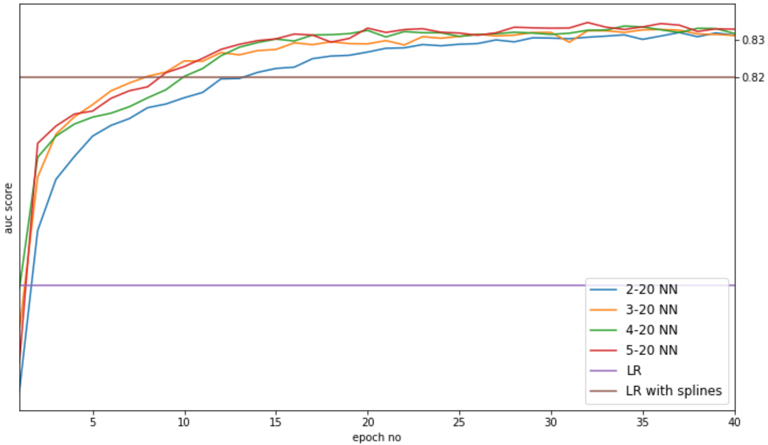
DL FOR FIRM BANKRUPTCY - IN-SAMPLE AUC



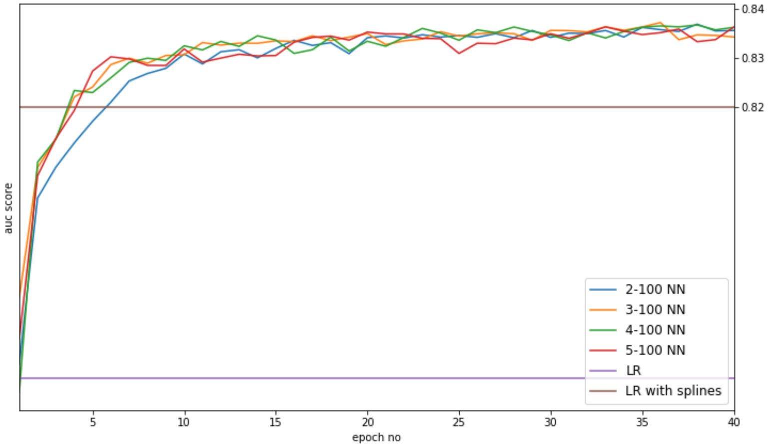
DL for firm bankruptcy - in-sample AUC



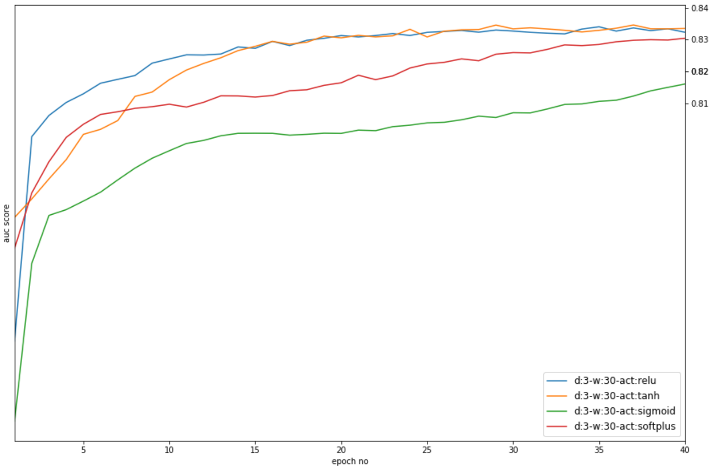
DL for firm bankruptcy - out-of-sample AUC



DL for firm bankruptcy - out-of-sample AUC



DL for firm bankruptcy - choice of activation



Software

- Matlab: **Neural Network Toolbox** (`patternet` function)
- R: **mxnet**
- Python: **sklearn.neural_network**
- Google's **TensorFlow** for serious use. Excellent GPU-support. Efficient use of computational graph. Start-up cost.
- Theano, Caffe, Torch, Mathematica etc etc
- See Wiki page for a comparison.

- I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
- H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.
- P. Giordani, T. Jacobson, E. Von Schedvin, and M. Villani, "Taking the twists into account: Predicting firm bankruptcy risk with splines of financial ratios," *Journal of Financial and Quantitative Analysis*, vol. 49, no. 4, pp. 1071–1099, 2014.