# Chapter 7. Neural Networks

Wei Pan

Division of Biostatistics, School of Public Health, University of Minnesota, Minneapolis, MN 55455
Email: weip@biostat.umn.edu

PubH 7475/8475

# Introduction

- Chapter 11. only on Feedforward NN (FNN).
  Also called Fully Connected NN (FCN) and Multi-Layer Perceptron (MLP).
  Related to projection pursuit regression (§11.2):
  $f(x) = \sum_{m=1}^{M} g_m(w_m' x)$,
  where each $w_m$ is a vector of weights and $g_m$ is a smooth nonparametric function; to be estimated. really?
  GAM: $f(x) = \sum_{m=1}^{p} g_m(x_m)$ (as if $w_m = e_m$) (§9.1).
- Here: + CNN; later recurrent NNs (for seq data).
  Autoencoders (unsupervised) ...?
  Goodfellow, Bengio, Courville (2016). *Deep Learning*.
  `http://www.deeplearningbook.org/`
- Two high waves in 1960s and late 1980s-90s.
- McCulloch & Pitts model (1943):
  $n_j(t) = I(\sum_{i \to j} w_{ij} n_i(t-1) > \theta_j)$.
  $w_{ij}$ can be $> 0$ (excitatory) or $< 0$ (inhibitory).

- A biological neuron vs an artificial neuron (perceptron).
  Google: images biological neural network tutorial
  Minsky & Papert's (1969) XOR problem:
  $XOR(X_1, X_2) = 1$ if $X_1 \neq X_2$; $= 0$ o/w. $X_1, X_2 \in \{0, 1\}$.
  Perceptron: $f = I(\alpha_0 + \alpha' X > 0)$.
- Feldman's (1985) "one hundred step program": at most 100 steps within a human reaction time.
  because a human can recognize another person in 100 ms, while the processing time of a neuron is 1ms. $\Longrightarrow$ human brain works in a massively parallel and distributed way.
- Cognitive science: human vision is performed in a series of layers in the brain.
- Human can learn.
- Hebb (1949) model:
  $w_{ij} \leftarrow w_{ij} + \eta y_i y_j$,
  reinforcing learning by simultaneous activations.

# Feed-forward NNs

- Fig 11.2. Input: $X$
- A (hidden) layer: for $m = 1, ..., M$,
  $Z_m = \sigma(\alpha_{0m} + \alpha_m' X)$, $Z = (Z_1, ..., Z_M)'$.
  **activation function:** $\sigma(v) = 1/(1 + \exp(-v))$, sigmoid (or logit$^{-1}$); Q: what is each $Z_m$?
  hyperbolic tangent: $tanh(v) = 2\sigma(v) - 1$.
- ...(may have multiple (hidden) layers)...
- Output: $f_1(X), ..., f_K(X)$.
  $T_k = \beta_{0k} + \beta_k' Z$, $T = (T_1, ..., T_K)'$,
  $f_k(X) = g_k(T)$.
  regression: $g_k(T) = T_k$;
  classification: $g_k(T) = \exp(T_k)/\sum_{j=1}^{K} \exp(T_j)$; softmax or multi-logit$^{-1}$ function.
- More generally, $L$-hidden layers: $f(W_L\sigma_0(...\sigma_0(W_1X)))$.
  $W_j$: $p_j \times p_{j-1}$ (unknown) weight parameter matrix.
  **DL**: large $L$.

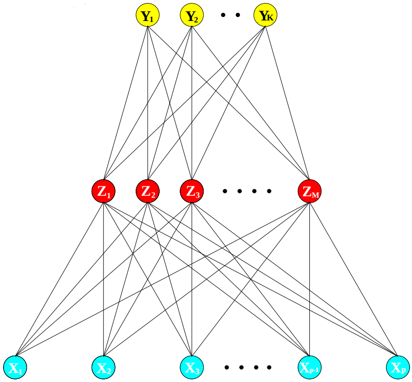



**FIGURE 11.2.** *Schematic of a single hidden layer, feed-forward neural network.*

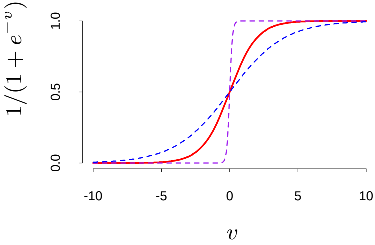



**FIGURE 11.3.** *Plot of the sigmoid function* $\sigma(v) = 1/(1+\exp(-v))$ *(red curve), commonly used in the hidden layer of a neural network. Included are* $\sigma(sv)$ *for* $s = \frac{1}{2}$ *(blue curve) and* $s = 10$ *(purple curve). The scale parameter* $s$ *controls the activation rate, and we can see that large* $s$ *amounts to a hard activation at* $v = 0$*. Note that* $\sigma(s(v - v_0))$ *shifts the activation threshold from* 0 *to* $v_0$*.*

- How to fit the model?
- Given training data: $(Y_i, X_i)$, $i = 1, ..., n$.
- For regression, minimize
  $R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{n} (Y_{ik} - f_k(X_i))^2$.
- For classification, minimize
  $R(\theta) = -\sum_{k=1}^{K} \sum_{i=1}^{n} Y_{ik} \log f_k(X_i)$.
  And $G(x) = \arg\max f_k(x)$.
- Can use other loss functions.
- How to minimize $R(\theta)$?
  Gradient descent, called back-propagation.
  §11.4
  Very popular and appealing! recall Hebb model
- Other algorithms: Newton's, conjugate-gradient, ...

# Back-propagation algorithm

- Given: training data $(Y_i, X_i)$, $i = 1, ..., n$.
- Goal: estimate $\alpha$'s and $\beta$'s.
  Consider $R(\theta) = \sum_i \sum_k (Y_{ik} - f_k(X_i))^2 := \sum_i R_i := \sum_i r_i^2$.
- NN: input $X_i$, output $(f_1(X_i), ..., f_K(X_i))'$.
  $Z_{mi} = \sigma(\alpha_{0m} + \alpha_m' X_i)$, $Z_i = (Z_{1i}, ..., Z_{Mi})'$,
  $T_{ki} = \beta_{0k} + \beta_k' Z_i$, $T_i = (T_{1i}, ..., T_{Ki})'$,
  $f_k(X_i) = g_k(T_i) = T_{ki}$.
- Chain rule:

$$\frac{\partial R_i}{\partial \beta_{km}} = \frac{\partial R_i}{\partial r_i} \frac{\partial r_i}{\partial g_k} \frac{\partial g_k}{\partial T_i} \frac{\partial T_i}{\partial \beta_{km}}$$

$$\frac{\partial R_i}{\partial \beta_{km}} = -2(Y_{ik} - f_k(X_i)) g_k'(\beta_k' Z_i) Z_{mi} := \delta_{ki} Z_{mi},$$

# Back-propagation algorithm (cont'ed)

- $$\frac{\partial R_i}{\partial \alpha_{ml}} = \frac{\partial R_i}{\partial r_i}\frac{\partial r_i}{\partial g_k}\frac{\partial g_k}{\partial T_i}\frac{\partial T_i}{\partial Z_i}\frac{\partial Z_i}{\partial \alpha_{ml}}$$

  $$\frac{\partial R_i}{\partial \alpha_{ml}} = -\sum_k 2(Y_{ik} - f_k(X_i))g_k'(\beta_k' Z_i)\beta_{km}\sigma'(\alpha_m' X_i)X_{il} := s_{mi}X_{il}.$$

  where $\delta_{ki}$, $s_{mi}$ are "errors" from the current model.
- Update at step $r+1$:

  $$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_i \left.\frac{\partial R_i}{\partial \beta_{km}}\right|_{\beta^{(r)},\alpha^{(r)}}, \quad \alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_i \left.\frac{\partial R_i}{\partial \alpha_{ml}}\right|_{\beta^{(r)},\alpha^{(r)}}.$$

  $\gamma_r$: **learning rate**; a tuning parameter; can be fixed or selected/decayed. too large/small then ...
- training **epoch**: a cycle of updating

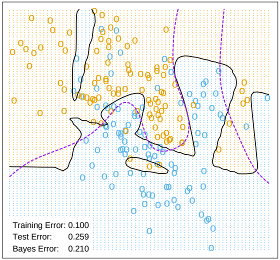
# Some issues

- Starting values:
  Existence of many local minima and saddle points.
  Multiple (random) initializations; model averaging, ...
  Data preprocessing: centering at 0 and scaling;
  batch normalization; Glorot-normal distribution ....
- Stochastic gradient descent (**SGD**): use a minibatch (i.e. a random subset) of the training data for a few iterations; minimbatch size: 32 or 64 or 128 or ..., a tuning parameter.
- +: simple and intuitive; -: slow
- Modifications: SGD + Momentum
  SGD: $x_{t+1} = x_t - \gamma \nabla f(x_t)$.
  SGD+M: $v_{t+1} = \rho v_t + \nabla f(x_t)$, $x_{t+1} = x_t - \gamma v_{t+1}$.
  ... (AdaGrad, RMSProp) ... **Adam**, default (now!)

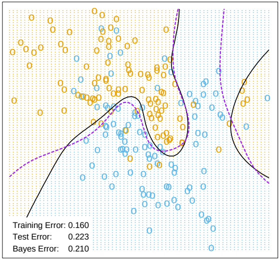
# Some issues (cont'd)

- Over-fitting? Universal Approx Thm
  If add more units or layers, then...
  1) Early stopping!
  2) Regularization: add a penalty term , e.g. Ridge; use $R(\theta) + \lambda J(\theta)$ with $J(\theta) = \sum_{km} \beta_{km}^2 + \sum_{ml} \alpha_{ml}^2$;
  called **weight decay**; Fig 11.4.
  Performance: Figs 11.6-8
  3) Regularization: **Dropout** (randomly) a subset/proportion of nodes/units or connections during training; an ensemble; more robust.
- A main technical issue with a deep NN: gradients vanishing or exploding, why?
  use **ReLU**: $\sigma(x) = \max(0, x)$; batch normalization; ....
- **Transfer learning**: reusing trained networks: why?
  `http://jmlr.org/proceedings/papers/v32/donahue14.pdf`
- Example code: ex7.1.r

Neural Network - 10 Units, No Weight Decay




Neural Network - 10 Units, Weight Decay=0.02



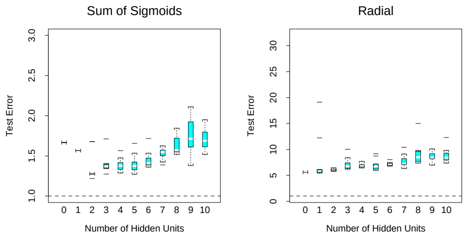



**FIGURE 11.6.** *Boxplots of test error, for simulated data example, relative to the Bayes error (broken horizontal line). True function is a sum of two sigmoids on the left, and a radial function is on the right. The test error is displayed for* 10 *different starting weights, for a single hidden layer neural network with the number of units as indicated.*

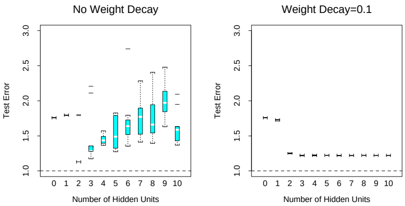



**FIGURE 11.7.** *Boxplots of test error, for simulated data example, relative to the Bayes error. True function is a sum of two sigmoids. The test error is displayed for ten different starting weights, for a single hidden layer neural network with the number units as indicated. The two panels represent no weight decay (left) and strong weight decay* $\lambda = 0.1$ *(right).*

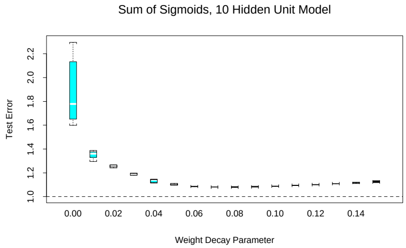
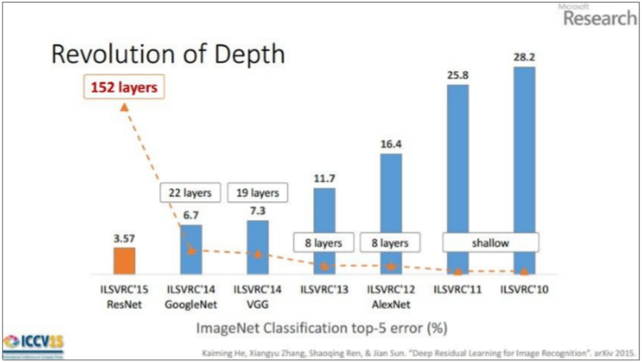



**FIGURE 11.8.** *Boxplots of test error, for simulated data example. True function is a sum of two sigmoids. The test error is displayed for ten different starting weights, for a single hidden layer neural network with ten hidden units and weight decay parameter value as indicated.*

# Current and future ...

- **Deep learning**: deep NNs (Wikipedia; google)
  Facebook hired Yann LeCun at NYU;
  Google hired Geoffrey Hinton at U Toronto;
  Bengio stays in U Montreal;
  Baidu hired Andrew Ng, who recently left; ...
- Impressive applications: imaging recognition (Krizhevsky et al); playing the game of Go (Silver et al 2016, *Nature*); ...
- Keys: AlexNet (Krizhevsky et al),
  "60 million parameters ... of five convolutional layers ... three fully-connected layers witha final 1000-way softmax."
  "there are roughly 1.2 million training images, 50,000 validation images, and 150,000 testing images."
  Needs **regularization** too!
- Qs: another wave? yes! just check constantly appearing papers on arXiv, ICLR, NeurIPS, ...




(slide from Kaiming He's recent presentation)

# Convolutional NNs

- LeCun et al (1998, *Proc of the IEEE*);
- Keys: "to ensure some degree of shift, scale, and distortion invariance: *local receptive fields, shared weights ... and spatial or temporal sub-sampling*."
- "Local correlations are the reasons for the well-known advantages of extracting and combining *local* features ..."
- Hubel and Wiesel (1962): locally-sensitive, orientation-selective neurons in the cat's visual system.
- New: a convolution layer uses rectified linear function,

$$\text{ReLU}(x) = \max(0, x).$$

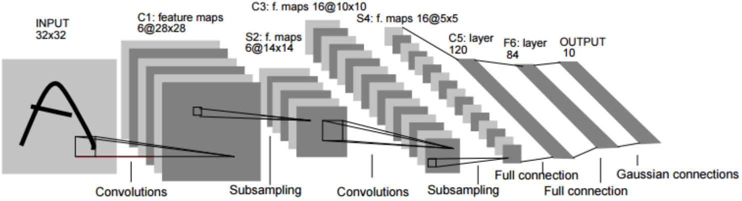



Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Figure: LeCun et al 1998, *Proc of the IEEE*.

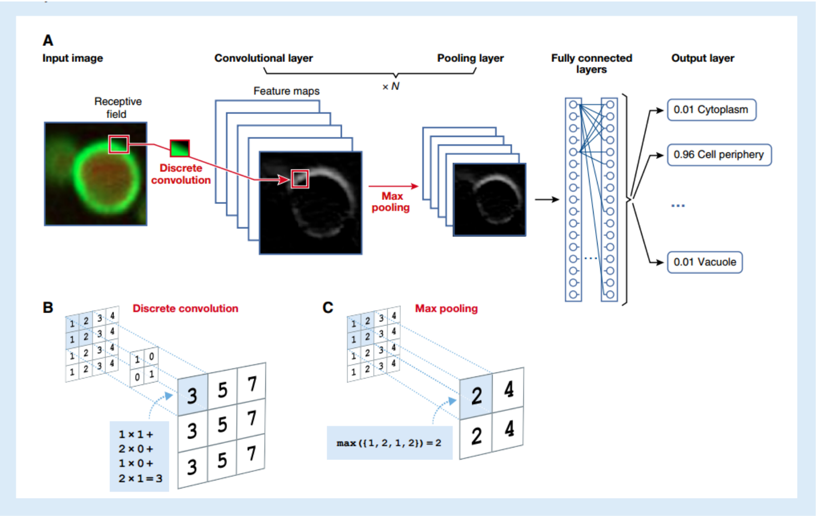



Figure: Angermueller et al 2016, *Mol Sys Biol*.

# Resources

- Today's "standards": mostly in Python
  1. Caffe (UC Berkeley) $\Longrightarrow$ Caffe2 (Facebook);
  2. Torch (NYU/Facebook) $\Longrightarrow$ PyTorch (Facebook);
  3. Theano(U Montreal) $\Longrightarrow$ TensorFlow (Google);

  3b. Keras: on top of TensorFlow.

  Others: MXNet (Amazon), Paddle (Baidu), CNTK (Microsoft)...
- CPU vs GPU
- Matlab: `ConvNet`, `DeepLearnToolBox`, `MatConvNet`, ...
- Java: `Deeplearning4j`, ...
- R packages: `deepnet`, `darch`, `mxnet`, `h2o`, ...

  **now**: R-Keras