National Technical University of Ukraine
“Igor Sikorsky Kyiv Polytechnic Institute”
Institute of Physics and Technology

Lecture 16
Objects and Systems Identification Methods. Artificial Neural Networks

Dmytro Progonov,

PhD, Associate Professor,

Department Of Physics and Information Security Systems

Content

- A brief history of the field;
- The perceptron algorithm;
- Feedforward neural networks;
- Other kinds of neural networks;
- The backpropagation algorithm;
- Regularization methods for MLP;
- Bayesian inference for MLP.

A brief history of the field (1/3)

- ***1943*** – McCulloch and Pitts devised a *simple mathematical model of the neuron*, in which they approximated the output as weighted sum of inputs passed through a threshold function $y = \mathbb{I}(\sum_i w_i x_i > \theta)$ for some threshold θ;
- ***1957*** – Frank Rosenblatt *invented the perceptron learning algorithm*, which is a way to estimate the parameters of McCulloch-Pitts neuron;
- ***1960*** – Widrow and Hoff *invented adaline* (adaptive linear element);
- ***1969*** – Minsky and Papert *published a famous book called "Perceptrons"* in which *they showed* that *such linear model*, with no hidden layers, were very limited in their power, since they *cannot classify data that is not linearly separable*.

A brief history of the field (2/3)

- ***1986*** – Rumelhart, Hintin and Williams *discovered the backpropagation algorithm*, which allows one to fit models with hidden layers (Bryson and Ho in 1969, and Werbos in 1974 independently invented this algorithm earlier);
- ***1987*** – Sejnowski and Rosenberg *created the famous NETtalk system*, that learned a mapping from English words to phonetic symbols which could be fed into a speech synthesizer;
- ***1989*** – LeCun and others *created the famous LeNet system* to classify gray-scale images of handwritten digits from MNIST dataset;
- ***1992*** – *the support vector machine (SVM) was invented* that provide similar prediction accuracy to neural networks while being considerably easier to train (since they use a convex objective function).

A brief history of the field (3/3)

- ***2002*** – Hinton *invented the contrastive divergence training procedure*, which provided a way for the first time, to learn deep networks, by training one layer at a time in an unsupervised fashion;
- ***2009*** – Salakhutdinov and Hinton *discovered the deep Boltzmann machine (DBM)* and *created the procedure to training the deep networks with any numbers of layers*.

The perceptron algorithm

We now how to fit a binary logistic regression model in an online manner. Let us represent the most probable class label as follows

$$\boxed{\hat{y}_i = \underset{y\in\{0,1\}}{\operatorname{argmax}}\, p(y|\mathbf{x}_i, \boldsymbol{\theta})}$$

We replace $\mu_i = p(y = 1|\mathbf{x}_i, \boldsymbol{\theta}) = sigm(\boldsymbol{\theta}^T\mathbf{x}_i)$ in the gradient expression with $\hat{y}_i$. Thus the approximate gradient becomes

$$\boxed{\mathbf{g}_i \approx (\hat{y}_i - y_i)\mathbf{x}_i}$$

It will make the algebra simpler if we assume $y \in \{-1,1\}$ rather than $y \in \{0,1\}$. In this case our prediction becomes

$$\boxed{\hat{y}_i = sign\ [\boldsymbol{\theta}^T\mathbf{x}_i]}$$

Then if $\hat{y}_i y_i = (-1)$, we have made an error, and guessed the right label if $\hat{y}_i y_i = (+1)$. The weight update has the simple form

$$\boxed{\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} + \eta_k y_i \mathbf{x}_i = \boldsymbol{\theta}_{k-1} + y_i \mathbf{x}_i}$$

Feedforward neural networks (1/2)

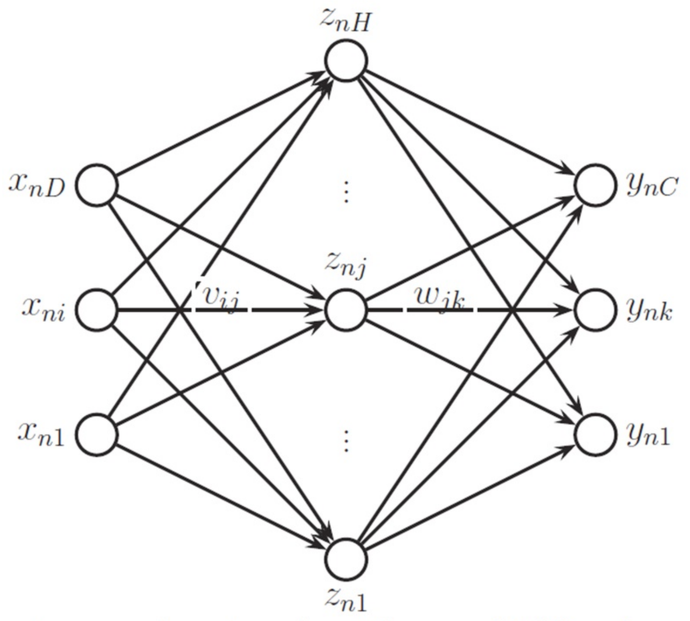
A (feedforward) ***neural network*** or ***multi-layer perceptron*** (***MLP***) is a series of logistic regression models stacked on top of each other, with the final layer being either another logistic regression of a linear regression model, depending on whether we are solving a classification or regression problem.

For example, if *we have two layers*, and we *are solving a regression problem*, the model has the form:

$$\boxed{\begin{array}{c} p(y|\mathbf{x},\boldsymbol{\theta}) = \mathcal{N}(y|\mathbf{w}^T\mathbf{z}(\mathbf{x}),\sigma^2) \\ \mathbf{z}(\mathbf{x}) = g(\mathbf{V}\mathbf{x}) = [g({\mathbf{v}_1}^T\mathbf{x}),\cdots,g({\mathbf{v}_H}^T\mathbf{x})] \end{array}}$$

where $\mathbf{v}_j = \mathbf{V}_j$ is the j^{th} row of $\mathbf{V}$. Here $\mathbf{z}(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x},\mathbf{V})$ is called ***hidden layer*** of size H; $g(\cdot)$ is a non-linear ***activation*** or ***transfer function***.

Feedforward neural networks (2/2)



A neural network with one hidden layer.

To handle binary classification, we pass the output through a sigmoid, as in GLM:

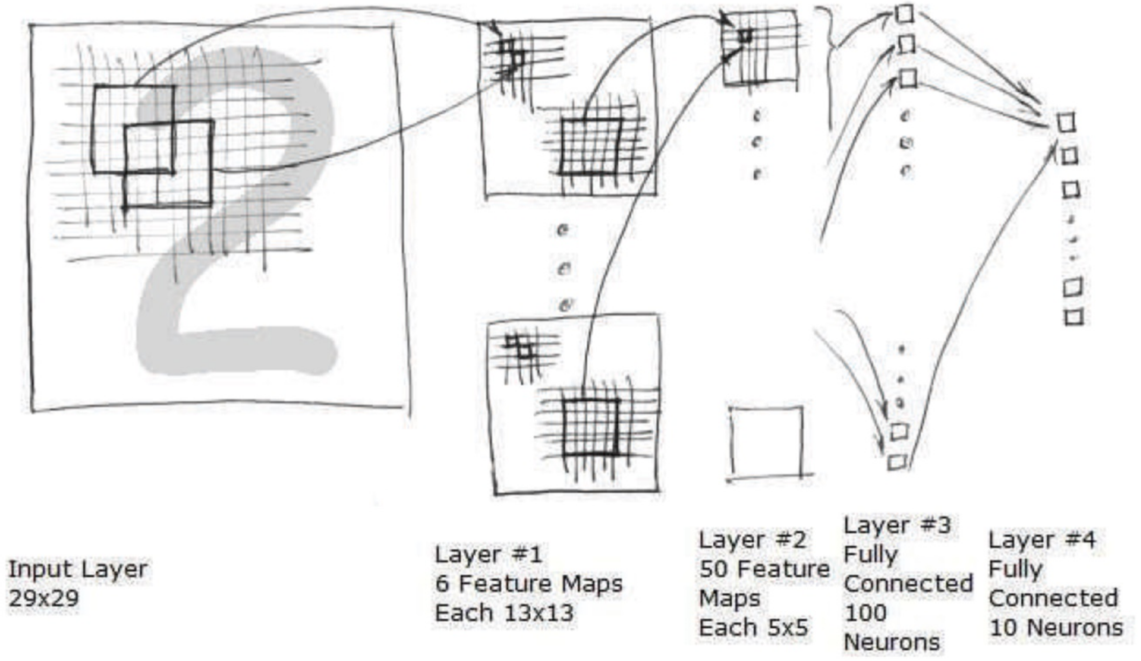
$$\boxed{p(y|\mathbf{x},\boldsymbol{\theta}) = Ber(y|sigm\,[\mathbf{w}^T\mathbf{z}(\mathbf{x})])}$$

We can easily extend the MLP to predict multiple outputs. For example, in regression case, we have

$$\boxed{p(y|\mathbf{x},\boldsymbol{\theta}) = \mathcal{N}(y|\mathbf{W}\boldsymbol{\phi}(\mathbf{x},\mathbf{V}),\sigma^2\mathbf{I})}$$

Other kinds of neural networks (1/2)

A form of MLP which is particularly well suited to 1D signals like speech or text, or 2D signals like images, is the ***convolutional neural network***. The purpose of the hidden units for such networks is to learn non-linear combinations of the original inputs; this is called ***feature extraction*** or ***feature construction***.



Input Layer 29x29
Layer #1 6 Feature Maps Each 13x13
Layer #2 50 Feature Maps Each 5x5
Layer #3 Fully Connected 100 Neurons
Layer #4 Fully Connected 10 Neurons

The convolutional neural network from (Simard et al. 2003). Source: `http://www.codeproject.com/KB/library/NeuralNetRecognition.aspx` . Used with kind permission of Mike O'Neill.

Other kinds of neural networks (2/2)

If we allow feedback connections, the model is known as a ***recurrent neural networks*** (***RNN***); this defines a nonlinear dynamical system, but does not have a simple probabilistic interpretation.

Such *RNN* models are currently *the best approach for language modeling*, i.e. performing word prediction in natural language, significantly outperforming the standard n-gram-based methods.

One difficulty with *RNNs* is that they are hard to train, since they *suffer from the vanishing gradient problem*.

If we allow symmetric connections between the hidden units, the model is known as a ***Hopfield network*** or ***associative memory*** that can be used for unsupervised learning.

The backpropagation algorithm (1/7)

Unlike a GLM, *the NLL if an MLP is a non-convex function of its parameters*. Nevertheless, we can find a locally optimal ML or MAP estimate using standard gradient-based optimization methods.

We now discuss how to compute the gradient vector of the NLL by applying the chain rule of calculus. The resulting algorithm is known as ***backpropagation***.

For notational simplicity, we shall assume a model with just one hidden layer. Let $\mathbf{x}_n$ be the n'th input, $\mathbf{a}_n = \mathbf{V}\mathbf{x}_n$ be the pre-synaptic hidden layer, and $\mathbf{z}_n = g(\mathbf{a}_n)$ be the post-synaptic hidden layer, where g is some ***transfer function***. We typically use $g(a) = sigm(a)$, but we may also use $g(a) = tanh(a)$.

The backpropagation algorithm (2/7)

We now convert this hidden layer to the output layer as follows. Let $\mathbf{b}_n = \mathbf{W}\mathbf{z}_n$ be the pre-synaptic output layer, and $\hat{\mathbf{y}}_n = h(\mathbf{b}_n)$ be the post-synaptic output layer, where h is another nonlinearity, corresponding to the canonical link for the GLM.

For regression model, we use $h(\mathbf{b}) = \mathbf{b}$; for binary classification, we use $h(\mathbf{b}) = [sigm(b_1), \cdots, sigm(b_c)]$.

We can write the overall model as follows

$$\boxed{\mathbf{x}_n \xrightarrow{\mathbf{V}} \mathbf{a}_n \xrightarrow{g} \mathbf{z}_n \xrightarrow{\mathbf{W}} \mathbf{b}_n \xrightarrow{h} \hat{\mathbf{y}}_n}$$

The parameters of the model are $\boldsymbol{\theta} = (\mathbf{V}, \mathbf{W})$, the first and second layer weight matrices. Offset or bias terms can be accommodated by clamping an element of $\mathbf{x}_n$ and $\mathbf{z}_n$ to one.

The backpropagation algorithm (3/7)

In the regression case, with K outputs, the NLL is given by the squared error:

$$\boxed{J(\boldsymbol{\theta}) = -\sum_n \sum_k (\hat{y}_{nk}(\boldsymbol{\theta}) - y_{nk})^2}$$

In the classification case, with K classes, the NLL is given by the cross entropy:

$$\boxed{J(\boldsymbol{\theta}) = -\sum_n \sum_k y_{nk} \log[\hat{y}_{nk}(\boldsymbol{\theta})]}$$

Our task is to compute $\nabla_{\boldsymbol{\theta}} J$. We will derive this for each n separately; the overall gradient is obtained by summing over n.

The backpropagation algorithm (4/7)

Let us start by considering the output layer weights. We have

$$\boxed{\nabla_{\mathbf{w}_k} J_n = \frac{\partial J_n}{\partial b_{nk}} \nabla_{\mathbf{w}_k} b_{nk} = \frac{\partial J_n}{\partial b_{nk}} \mathbf{z}_n}$$

since $b_{nk} = \mathbf{w}_k{}^T \mathbf{z}_n$. Assuming h is the canonical link function for the output GLM, then

$$\boxed{\frac{\partial J_n}{\partial b_{nk}} \triangleq \delta_{nk}{}^w = \hat{y}_{nk} - y_{nk}}$$

which is the error signal. So the overall gradient is

$$\boxed{\nabla_{\mathbf{w}_k} J_n = \delta_{nk}{}^w \mathbf{z}_n}$$

which is the pre-synaptic input to the output layer, namely $\mathbf{z}_n$, times the error signal, namely $\delta_{nk}{}^w$.

The backpropagation algorithm (5/7)

For the input layer weights, we have

$$\boxed{\nabla_{\mathbf{v}_j} J_n = \frac{\partial J_n}{\partial a_{nj}} \nabla_{\mathbf{v}_j} a_{nj} \triangleq \delta_{nj}{}^{v} \mathbf{x}_n}$$

where we exploited the fact that $a_{nj} = \mathbf{v}_j{}^T \mathbf{x}_n$. All the remains is to compute the first level error signal$\delta_{nj}{}^{v}$. We have

$$\boxed{\delta_{nj}{}^{v} = \frac{\partial J_n}{\partial a_{nj}} = \sum_{k=1}^{K} \frac{\partial J_n}{\partial b_{nk}} \frac{\partial b_{nk}}{\partial a_{nj}} = \sum_{k=1}^{K} \delta_{nk}{}^{w} \frac{\partial b_{nk}}{\partial a_{nj}}}$$

Now

$$\boxed{b_{nk} = \sum_{j} w_{kj} g(a_{nj})}$$

so

$$\boxed{\frac{\partial b_{nk}}{\partial a_{nj}} = w_{kj} \acute{g}(a_{nj})}$$

The backpropagation algorithm (6/7)

The $\acute{g}(a) = \frac{d}{da} g(a)$. For than units, $\acute{g}(a) = \frac{d}{da} tanh(a) = sech^2(a)$, and for sigmoid units, $\acute{g}(a) = \frac{d}{da}\sigma(a) = \sigma(a)\big(1-\sigma(a)\big)$. Hence

$$\boxed{{\delta_{nj}}^{v} = \sum_{k=1}^{K} {\delta_{nk}}^{w} w_{kj} \acute{g}(a_{nj})}$$

Thus the layer 1 errors can be computed by passing the layer 2 errors back through the $\mathbf{W}$ matrix; hence the term "***backpropagation***". The key property is that we can compute the gradient locally: each node only needs to know about it immediate neighbors.

The backpropagation algorithm (7/7)

Putting it all together, we can compute all the gradients as follows: we first perform a forward pass to compute $\mathbf{a}_n$, $\mathbf{z}_n$, $\mathbf{b}_n$ and $\hat{\mathbf{y}}_n$. We then compute the error for the output layer, ${\boldsymbol{\delta}_n}^{(2)} = \hat{\mathbf{y}}_n - \mathbf{y}_n$, which we pass backwards through $\mathbf{W}$ and compute the error for the hidden layer, ${\boldsymbol{\delta}_n}^{(1)}$. We then compute the overall gradient vector by stacking the two components vector as follows

$$\boxed{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \left[\sum_n {\boldsymbol{\delta}_n}^{v} \mathbf{x}_n \,;\, \sum_n {\boldsymbol{\delta}_n}^{w} \mathbf{z}_n\right]}$$

Regularization methods for MLP

- ***Early stopping*** – means stopping the training procedure when the error on the validation set first start to increase;
- ***Weight decay*** – to impose a prior on the parameters, and then use MAP estimation. It is standard to use a $\mathcal{N}(0, \alpha^{-1}\mathbf{I})$ prior (equivalent to ℓ_2 regularization), where α is the precision (strength) of the prior;
- ***Weight pruning*** – to encourage sparsity with usage of sparsity-promoting techniques;
- ***Soft weight sharing*** – to encourage similar weights to share statistical strength. Parameters that are assigned to the same cluster will share the same mean and variance and thus will have similar values (assuming the variance for that cluster is low)

Bayesian inference for MLP

- Integrating out the parameters instead of optimizing them is much stronger form of regularization than MAP estimation;
- We can use Bayesian model selection to determine things like the hyper-parameters settings and the number of hidden units. This is likely to be much faster than cross validation, especially if we have many hyper-parameters;
- Modelling uncertainty in the parameters will induce uncertainty in our predictive distributions, which is important for certain problems such as active learning and risk-averse decision making;
- We can use online inference methods, such as extended Kalman filter, to do online learning.

Conclusion

- A brief history of the field the artificial neural network was shown;
- The core perceptron algorithm as well as feedforward neural networks were presented;
- Special kinds of neural networks were considered;
- The backpropagation algorithm and regularization methods for MLP were described.