Lecture 2

Neural Network Design and Back Propagation Algorithm

Georgiy Platonov and Kamrul Hasan

June 2017

Representation Power of the Neural Networks

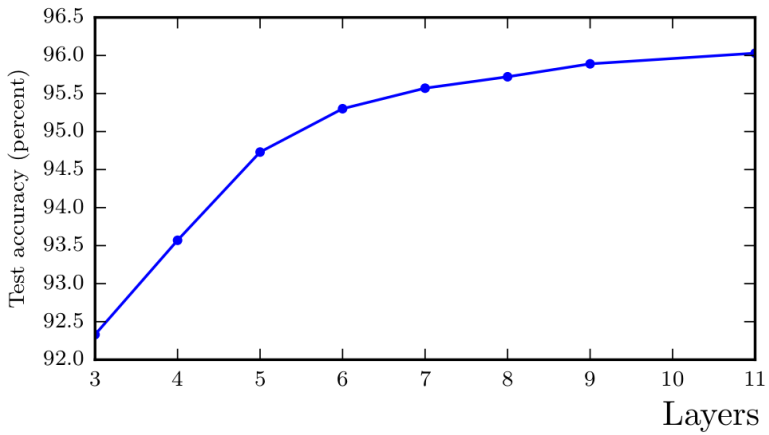
The neural networks are a very powerful tool for function approximation.

The Universal Approximation Theorem (Cybenko, 1989) states that the set of functions, represented by the single-hidden layer (with certain types of non-linear activations), feedforward networks, with linear outputs, is dense in the space of continuous functions over the n-dimensinal unit cube I^n.

This is a very broad class of functions, so we can can represent many things. Yet there are a couple of problems.

Network Design

- Controlling the model complexity
- Depth vs. width
- Choice of the activation functions
- Nontrivial connections



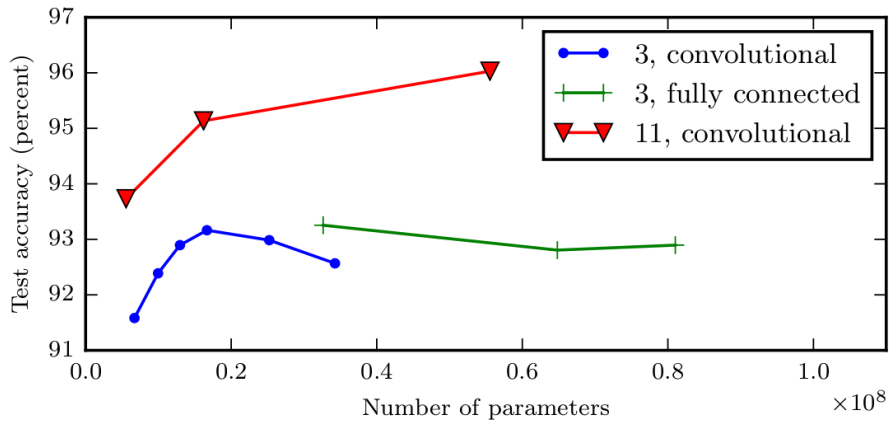


Figure: The scaling behaviour of the network

Network Training

The most popular algorithm for the NN learning are the gradient descent methods based on the back-propagation algorithm (AKA backprop).

Given the error $E(w)$, depending on the weights, we want to minimize it. The standard approach to function minimization is the gradient descent family of methods. The general idea is to move, iteratively, in the opposite direction to the the gradient vector in the argument space:

$$W(t+1) = W(t) - \eta \nabla E(W(t)),$$

where t is the iteration index.

Network Training

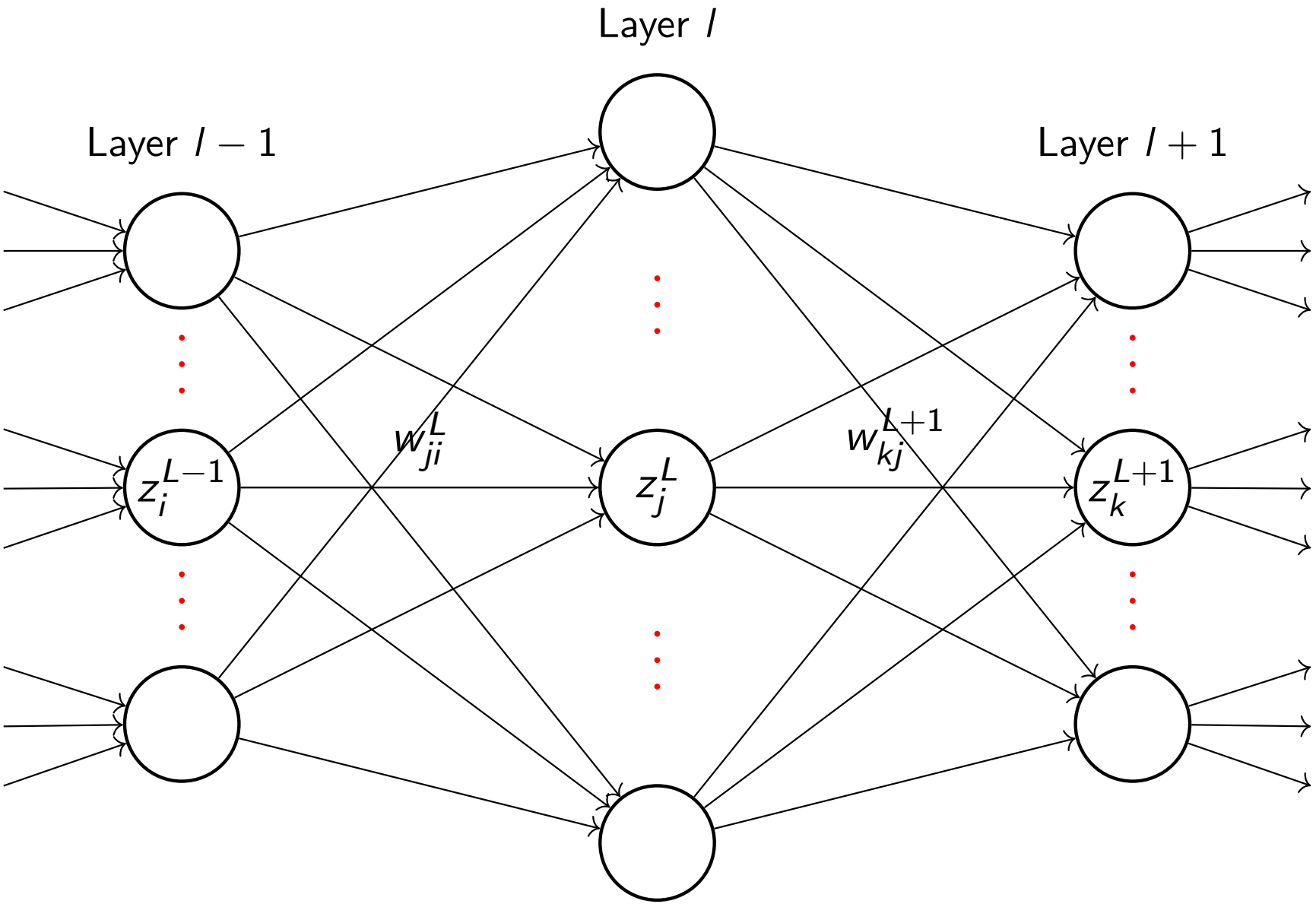
Let x be the input vector, y be the true output (label) corresponding to x and $\hat{y}$ be the actual network output. Then the error function can be expressed as

$$E = \frac{1}{2}\sum_n (\hat{y}_n - y_n)^2 = \sum_n E_n,$$

where $E_n = \frac{1}{2}(\hat{y}_n - y_n)^2$.

Following the Stochastic Gradient Descent (SGD) approach we will evaluate the gradient and update the weights at each training point x_n, so that instead of computing ∇E we will need ∇E_n.

Network Training



Network Learning

Notation:
We will index the neurons of the three layers with i, j and k. Let z_i^{l-1}, z_j^l and z_k^{l+1} be the output values of the corresponding neurons. In this way, $z^0 = x$ is the input to the network, and $z^L = \hat{y}$ is the network output.

Let $s_j^l = \sum_i w_{ji}^l z_i^{l-1} + w_{0i}^l$ be the weighted sum of the inputs to the neuron.

We will denote as $J_X(Y)$ the Jacobian of Y with respect to X.

Finally, the symbol $\circ$ stands for Hadamard, or element-wise, product of two vectors.

Network Training

We need to calculate $\frac{\partial E_n}{\partial w_{ji}^l}$. Applying the chain rule yields

$$\frac{\partial E_n}{\partial w_{ji}^l} = \frac{\partial E_n}{\partial z_j^l}\frac{\partial z_j^l}{\partial s_j^l}\frac{\partial s_j^l}{\partial w_{ji}^l} = \delta_j^l (z_j^l)' z_i^{l-1}$$

where $\delta_j^l = \frac{\partial E_n}{\partial z_j^l}$. For the last layer, finding δ^L is easy (See the next slide).

But how do we get δ_j^l for the hidden layers? By applying the chain rule again!

$$\delta_j^l = \frac{\partial E_n}{\partial z_j^l} = \frac{\partial E_n}{\partial z^{l+1}}\frac{\partial z^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1}\frac{\partial z_k^{l+1}}{\partial z_j^l},$$

where

$$\delta_k^{l+1} = \frac{\partial E_n}{\partial z_k^{l+1}}$$

and

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = \frac{dz_k^{l+1}}{ds_k^{l+1}}\frac{\partial s_k^{l+1}}{\partial z_j^l} = (z_k^{l+1})' w_{kj}^{l+1}$$

Network Training

Combining these expressions we get

$$\frac{\partial E_n}{\partial w_{ji}^l} = (z_j^l)' z_i^{l-1} \sum_k \delta_k^{l+1} (z_k^{l+1})' w_{kj}^{l+1}$$

for the hidden layers, and

$$\frac{\partial E_n}{\partial w_{ji}^L} = (z_j^L)' z_i^{L-1} \delta_j^L$$

for the output layer. Regarding the $\delta^L = \frac{\partial E_n}{\partial z^L}$, we can calculate it directly since

$$E_n = \frac{1}{2}(\hat{y}_n - y_n)^2 = \frac{1}{2}(z^L - y_n)^2$$

which gives us

$$\frac{\partial E_n}{\partial z^L} = z^L - y_n$$

Network Training

Now let's evaluate the matrix version of these equations:

$$
\begin{aligned}
\delta_j^l &= \frac{\partial E_n}{\partial z^{l+1}} \frac{\partial z^{l+1}}{\partial s^{l+1}} \frac{\partial s^{l+1}}{\partial z_j^l} \\
&= (\delta^{l+1})^T J_{s^{l+1}}(z^{l+1}) \frac{\partial s^{l+1}}{\partial z_j^l} \\
&= (\frac{\partial s^{l+1}}{\partial z_j^l})^T J_{s^{l+1}}(z^{l+1}) \delta^{l+1} \\
&= (\frac{\partial s^{l+1}}{\partial z_j^l})^T (\mathbf{z'^{l+1}} \circ \delta^{l+1}) \\
&= (J_{z^l}(s^{l+1}))_{j:} (\mathbf{z^{l+1}} \circ \delta^{l+1}) \\
&= (J_{z^l}(s^{l+1})(\mathbf{z^{l+1}} \circ \delta^{l+1}))_j
\end{aligned}
$$

This result gives us that $\delta^l = J_{z^l}(s^{l+1})(\mathbf{z'^{l+1}} \circ \delta^{l+1})$.

Network Training

Next, since

$$\frac{\partial E_n}{\partial w_{ji}^l} = (z_j^l)' z_i^{l-1} \delta_j^l = (\mathbf{z}'^l \circ \delta^l)_j z_i^{l-1} = ((\mathbf{z}'^l \circ \delta^l)(z^{l-1})^T)_{ji},$$

we finally obtain

$$\frac{\partial E_n}{\partial w^l} = (\mathbf{z}'^l \circ \delta^l)(z^{l-1})^T,$$

where the last equation holds true for any layer $l \in \{1, ..., L\}$.

Network Training. Practical considerations

The time complexity of the gradient computation using backpropagation is $\mathcal{O}(|W|^2)$ for one training example. Not bad! :)

No time bounds for the whole learning process, though :(

No global minimum convergence guarantees as well :((

Multiple runs with randomly initialized parameters might help to avoid local minima and flat regions. Adaptive gradient descent methods can help to speed up learning.

Regularization in Neural Networks

Regularization is a technique of imposing certain constraints on the solutions to the problem. This can lead improvements of the learning process and control the model complexity.

Popular types of regularization include

L_2 regularization (AKA weight decay): $\hat{E} = E + \lambda \| W \|_2^2$

and

L_1 regularization: $\hat{E} = E + \lambda \| W \|_1$

Regularization in Neural Networks

Some additional methods for regularization:

- Dataset augmentation/Noise addition
- Early stopping
- Parameter Sharing
- Ensembles
- Dropout