Neural Networks

Tobias Scheffer
Overview

- Neural information processing.
- Deep learning.
- Feed-forward networks.
- Training feed-forward networks: back propagation.
- Parallel inference on GPUs.
- Outlook:
  - Convolutional neural networks,
  - Recurrent neural networks.
Learning Problems can be Impossible without the Right Features
Learning Problems can be Impossible without the Right Features
Learning Problems can be Impossible without the Right Features

Abstract features (higher level)

Raw data (low-level)
Learning Problems can be Impossible without the Right Features

- **Deep Learning:**
  - Neural network with many sequential layers.
  - Layer-wise transformation of the input into more abstract, more “semantic” attributes.
  - Raw input data (bitmap, spectral image of audio signal) is fed into network, no manual feature engineering.
  - Entire network is usually trained with stochastic gradient descent (back-propagation).
  - Hundreds of millions of parameters, hundreds of millions to billions of training data.

**Abstract features** (higher level)

**Raw data** (low-level)
Neural Networks

- Model of neural information processing
- Waves of popularity
  - ↓ Perceptron only linear classifier (Minsky, Papert, 69).
  - ↑ Multilayer perceptrons (90s).
  - ↓ Popularity of SVMs (late 90s).
  - ↑ Deep learning (late 2000s).
  - Now state of the art for Speech Recognition image classification, face recognition and other problems.
Deep Learning Records

- Neural networks best-performing algorithms for
  - Object classification (CIFAR/NORB/PASCAL VOC-Benchmarks)
  - Video classification (various benchmarks)
  - Sentiment analysis (MR Benchmark)
  - Pedestrian detection
  - Face recognition
  - Speech recognition
Neural Information Processing

Input signals

Weighted input signals are aggregated

Axon: output signal

Synaptic weights: strengthened and weakened by learning processes

Output signals are electric spikes

Connections to other nerve cells

Neural Information Processing

Input signals

Weighted input signals are aggregated

Axon: output signal

Synaptic weights: strengthened and weakened by learning processes

Output signals are electric spikes

Connections to other nerve cells
Neural Information Processing: Model

\[ h = x^T \theta + \theta_0 \]

Input vector \( x \)

Weight vector \( \theta \)

Output

\( \sigma(h) \)

Weighted input signals

Probability of an output spike
Overview

- Neural information processing.
- Deep learning.
- Feed-forward networks.
- Training feed-forward networks: back propagation.
- Parallel inference on GPUs.
- Outlook:
  - Convolutional neural networks,
  - Recurrent neural networks.
Feed Forward Networks

- Forward propagation:
  - Input vector $x^0$
  - Linear model: $h_k^i = \theta_k^i x^{i-1} + \theta_k^0$
  - Each unit has parameter vector $\theta_k^i = (\theta_{k1}^i \ldots \theta_{kn_i}^i)$
  - Layer $i$ has matrix of parameters $\theta^i = \begin{pmatrix} \theta_{11}^i & \ldots & \theta_{1n_i}^i \\ \vdots & \ddots & \vdots \\ \theta_{n_i1}^i & \ldots & \theta_{n_in_i}^i \end{pmatrix}$
Feed Forward Networks

- **Forward propagation:**
  - Input vector: $x^0$
  - Linear model: $h^i_k = \theta^i_k x^{i-1} + \theta^i_{k0}$
  - Activation function and propagation: $x^i = \sigma(h^i)$
  - Output vector: $x^d$

![Diagram of feed forward network with indices and equations]

- Input layer: $x^0_1 \ldots x^0_{n_0}$
- Hidden layers: $\theta^1$, $\theta^2$
- Output layer: $\theta^d$
- Index $i$ and $k$
Feed Forward Networks

- Bias unit
  - Linear modell: $h^i_k = \theta^i_k x^{i-1} + \theta^i_{k0}$
  - Constant element $\theta^i_{k0}$ is replaced by additional unit with constant output 1: $h^i_k = \theta^i_k x^{i-1}[1..n_k+1]$
Feed Forward Networks

- Forward propagation per layer in matrix notation:
  - Linear model: $h^i = \theta^i x^{i-1}$
  - Activation function: $x^i = \sigma(h^i)$
Classification: Softmax Layer

- One output unit per class:
  - $x_k^d = \sigma_{sm}(h_k^d) = \frac{e^{h_k^d}}{\sum_{k'} e^{h_{k'}^d}}$
  - $x_k^d$: predicted probability for class $k$. 

\[ x_k^2 = \sigma(h_k^2) \]
\[ h_k^2 = \theta_k^2 x^1 + \theta_k^0 \]
\[ x_k^1 = \sigma(h_k^1) \]
\[ h_k^1 = \theta_k^1 x^0 + \theta_k^0 \]
\[ x_1^0 \ldots x_{n_0}^0 \]
Regression: Sigmoidal Activation

For target variable $k$:

- $x_k^d = \sigma_s(h_k^d) = \frac{1}{1+e^{-h_k^d}}$
- Logistic activation function.
Internal Units: Rectified Linear Units

For internal unit \((i, k)\):
- \(x_k^i = \sigma_{ReLU}(h_k^i) = \max(0, h_k^i)\)
- Leads to sparse activations and prevents the gradient from vanishing for deep networks.
Overview

- Neural information processing.
- Deep learning.
- Feed-forward networks.
- Training feed-forward networks: back propagation.
- Parallel inference on GPUs.
- Outlook:
  - Convolutional neural networks,
  - Recurrent neural networks.
Feed Forward Networks: Learning

- Stochastic gradient descent
- Loss function
  \[ \hat{R}(\theta) = \frac{1}{2m} \sum_{j=1}^{m} \ell(y_j, x^d) \]
- Gradient descent:
  \[ \theta' = \theta - \alpha \nabla \hat{R}(\theta) = \theta - \alpha \frac{\partial}{\partial \theta} \hat{R}(\theta) \]
  \[ = \theta - \frac{\alpha}{2m} \sum_{j=1}^{m} \frac{\partial}{\partial \theta} \ell(y_j, x^d) \]
- Stochastic gradient for instance \( x_j \):
  \[ \theta' = \theta - \alpha \nabla_{x_j} \hat{R}(\theta) \]
  \[ = \theta - \alpha \frac{\partial}{\partial \theta} \ell(y_j, x^d) \]
Learning: Back Propagation

- Stochastic gradient for output units for instance $x_j$:

$$\frac{\partial \ell(y_j, x^d)}{\partial \theta_k^d} = \frac{\partial \ell(y_j, x^d)}{\partial x^d} \frac{\partial x^d}{\partial h_k^d} \frac{\partial h_k^d}{\partial \theta_k^d}$$

$$= \frac{\partial \ell(y_j, x^d)}{\partial x^d} \frac{\partial \sigma(h_k^d)}{\partial h_k^d} x^{d-1}$$

$$= \delta_k^d x^{d-1}$$

- With

$$\delta_k^d = \frac{\partial \ell(y_j, x^d)}{\partial x^d} \frac{\partial \sigma(h_k^d)}{\partial h_k^d}$$
Learning: Back Propagation

- Stochastic gradient for hidden units for instance $x_j$:
  \[
  \frac{\partial \ell(y_j, x^d)}{\partial \theta_k^i} = \frac{\partial \ell(y_j, x^d)}{\partial h^i_k} \frac{\partial h^i_k}{\partial \theta_k^i} = \delta^i_k x^{i-1}
  \]

  With
  \[
  \delta^i_k = \frac{\partial \ell(y_j, x^d)}{\partial h^i_k}
  \]

  \[
  = \sum_{l=1}^{n_i+1} \frac{\partial \ell(y_j, x^d)}{\partial h^{i+1}_l} \frac{\partial h^{i+1}_l}{\partial x^i_k} \frac{\partial x^i_k}{\partial h^i_k}
  \]

  \[
  = \sum_{l=1}^{n_i+1} \delta^{i+1}_l \theta_{lk} \frac{\partial \sigma(h^i_k)}{\partial h^i_k}
  \]
Learning: Back Propagation

- Derivative of the loss function for classification.
- Softmax activation function:
  \[ x_k^d = \sigma_{sm}(h_k^d) = \frac{e^{h_k^d}}{\sum_k e^{h_k^d}} \]
  \[ \frac{\partial \sigma_{sm}(h_k^d)}{\partial h_k^d} = \sigma_{sm}(h_k^d)(1 - \sigma_{sm}(h_k^d)) \]
- Cost function:
  \[ \ell(y, x^d) = \sum_k y_k \log x_k^d \]
  \[ \frac{\partial \ell(y, x^d)}{\partial h_k^d} = x_k^d - y_k \]
Learning: Back Propagation

- Derivative of the loss function for regression.

- Sigmoidal activation function:
  - \( x_k^d = \sigma_s(h_k^d) = \frac{1}{1+e^{-h_k^d}} \)
  - \( \frac{\partial \sigma_s(h_k^d)}{\partial h_k^d} = \sigma_s(h_k^d)(1 - \sigma_s(h_k^d)) \)

- Cost function:
  - \( \ell(y, x^d) = \frac{1}{2} \sum_k (x_k^d - y_k)^2 \)
  - \( \frac{\partial \ell(y, x^d)}{\partial x_k^d} = x_k^d - y_k \)
Learning: Back Propagation

- **Derivative of the activation function for internal units.**
- **Rectified linear activation function:**
  - \( x_k^i = \sigma_{ReLU}(h_k^i) = \max(0, h_k^i) \)
  - \( \frac{\partial \sigma_{ReLU}(h_k^i)}{\partial h_k^i} = \begin{cases} 1 & \text{if } h_k^i > 0 \\ 0 & \text{otherwise} \end{cases} \)
Back Propagation: Algorithm

- Iterate over training instances \((x, y)\):
  - Forward propagation: for \(i=0\ldots d\):
    - For \(k=1\ldots n_i\): \(h_k^i = \theta_k^i x^{i-1} + \theta_{k0}^i\)
    - \(x^i = \sigma(h^i)\)
  - Back propagation:
    - For \(k=1\ldots n_i\): \(\delta_k^d = \frac{\partial}{\partial h_k^d} \sigma(h_k^d) \frac{\partial}{\partial x_k^d} \ell(y_k, x_k^d)\)
      \(\theta_k^d' = \theta_k^d - \alpha \delta_k^d x_k^{d-1}\)
    - For \(i=d-1\ldots 1\):
      - For \(k=1\ldots n_i\): \(\delta_k^i = \sigma'(h_k^i) \sum_j \delta_{j+1}^i \theta_{jk}^{i+1}\)
        \(\theta_k^i' = \theta_k^i - \alpha \delta_k^i x_k^{i-1}\)

- Until convergence
Back Propagation

- Loss function is not convex
  - Each permutation of hidden units is a local minimum.
  - Learned features (hidden units) may be ok, but not usually globally optimal.

- But:
  - Local minima can still be arbitrarily good.
  - Many local minima can be equally good.
  - Supervised learning often works with hundreds of layers and millions of training instances.
Regularization

- L2-regularized loss
  - $\hat{R}_2(\theta) = \frac{1}{2m} \sum_j (y_j - x_j^d)^2 + \frac{\eta}{2} \theta^T \theta$
  - Corresponds to normal prior on parameters.
- Gradient: $\nabla \hat{R}_2(\theta^i) = \frac{1}{m} \sum_j \delta_j^i x^i + \eta \theta$
- Update: $\theta' = \theta - \delta_j^i x - \eta \theta$
- Called weight decay.

- Additional regularization schemes:
  - Early stopping (outdated): Stop before convergence.
  - Delete units with small weights.
  - Dropout: During training, set some units' output to zero at random.
Regularization: Dropout

- In complex networks, complex co-adaptation relationships can form between units.
  - Not robust for new data.
- Dropout: In each training set, draw a fraction of units at random and set their output to zero.
- At application time, use all units.
- Improves overall robustness: each unit has to function within varying combinations of units.
Overview

- Neural information processing.
- Deep learning.
- Feed-forward networks.
- Training feed-forward networks: back propagation.
- Parallel inference on GPUs.
- Outlook:
  - Convolutional neural networks,
  - Recurrent neural networks.
Parallel Inference

- Both forward and backward propagation can be made much faster by parallel computation.
- GPUs are particularly suited.
- Pipelining in single-core CPU can be exploited.
- Forward- and backward-propagation can be written as matrix multiplications.
- Columns of the weight matrix can be processed in parallel.
Parallel Inference

- Forward propagation per layer in matrix notation:
  \[ h^i = \theta^i x^{i-1} \]
  \[
  \begin{bmatrix}
  h_1^i \\
  \vdots \\
  h_{n_i}^i
  \end{bmatrix} =
  \begin{bmatrix}
  \theta_{11}^i & \cdots & \theta_{1n_i-1}^i \\
  \vdots & \ddots & \vdots \\
  \theta_{n_i1}^i & \cdots & \theta_{n_in_i-1}^i
  \end{bmatrix}
  \begin{bmatrix}
  x_1^{i-1} \\
  \vdots \\
  x_{n_i-1}^{i-1}
  \end{bmatrix}
  \]

- Use vector coprocessor / GPU row-by-column multiplications.
- Split up rows of \( \theta \) between multiple cores.
Software Packages

- Caffe: allows to easily apply deep learning with standard units, loss functions, learning techniques.
- Deep learning frameworks that allow development of new architectures and learning techniques.
  - Torch,
  - Theano (+Lasagne),
  - TensorFlow (+Keras),
  - CNTK.
Overview

- Neural information processing.
- Deep learning.
- Feed-forward networks.
- Training feed-forward networks: back propagation.
- Parallel inference on GPUs.

**Outlook:**

- Convolutional neural networks,
- Recurrent neural networks.
Convolution

- Filter unit takes window of input, produces response.
- For instance, a filter can detect a specific type of patterns.
- Same filter is applied in parallel to all window positions of the input.
- Each unit produces output of same filter for different position.
- This produces a response image.

\[ \theta^1 = \text{Convolution} \]

\[ \text{Filter unit takes window of input, produces response.} \]
\[ \text{For instance, a filter can detect a specific type of patterns.} \]
\[ \text{Same filter is applied in parallel to all window positions of the input.} \]
\[ \text{Each unit produces output of same filter for different position.} \]
\[ \text{This produces a response image.} \]
Convolutional Layer

- Applies multiple filters to all positions of input from layer below.
- Creates a bank of response images.

Response image of filter 1
= filter 1 applied to all image positions
Convolutional Neural Networks

- Each network layer transforms the image into a more abstract feature representation.
- Top-most layer can be softmax units for classification.

Image representation on network layer $i$ (3D tensor)

Transformation: matrix product + activation function

Image representation on network layer $i+1$
Convolutional Neural Networks

- Widely used in image processing; e.g.,
  - ImageNet (general image classification),
  - DeepFace (face recognition).

Image representation on network layer $i$ (3D tensor)

Transformation: matrix product + activation function

Image representation on network layer $i+1$
Overview

- Neural information processing.
- Deep learning.
- Feed-forward networks.
- Training feed-forward networks: back propagation.
- Parallel inference on GPUs.
- Outlook:
  - Convolutional neural networks,
  - Recurrent neural networks.
Recurrence Neural Networks

- **Input:** time series $x_1^0, \ldots, x_T^0$.
- **Output can be:**
  - One output (vector) for entire time series: $x^d$.
  - One output at each time step: $x_1^d, \ldots, x_T^d$. 
Recurrent Neural Networks

- Hidden layer propagates information to itself.
- Hidden layer activation stores context information.

Output: $x_1^d \ldots x_{t-1}^d \ x_t^d \ x_{t+1}^d \ldots \ x_T^d$

Input: $x_1^0 \ldots x_{t-1}^0 \ x_t^0 \ x_{t+1}^0 \ldots \ x_T^0$. 
Recurrent Neural Networks

- Propagation:
  \[ h^1_t = \theta^1 \left( \begin{array}{c} x^0_t \\ x^1_{t-1} \end{array} \right); \ x^1_t = \sigma(h^1_t) \]

- Output \( x^d_1, x^d_{t-1}, x^d_t, x^d_{t+1}, \ldots, x^d_T \)

- Input \( x^0_1, x^0_{t-1}, x^0_t, x^0_{t+1}, \ldots, x^0_T \).
Recurrent Neural Networks

- Identical network “unfolded” in time.
- Units on hidden layer propagate to the right.
- Hidden layer activation stores context information.

Output: \[ x^d_1 \ldots x^d_{t-1} x^d_t x^d_{t+1} \ldots x^d_T \]

Input: \[ x^0_1 \ldots x^0_{t-1} x^0_t x^0_{t+1} \ldots x^0_T \]
Recurrent Neural Networks

- Forward propagation

- Output \( x_1^d \ldots x_{t-1}^d x_t^d x_{t+1}^d \ldots x_T^d \)

- Input \( x_1^0 \ldots x_{t-1}^0 x_t^0 x_{t+1}^0 \ldots x_T^0 \)
Recurrent Neural Networks

- Back propagation through time.

- Output $\mathbf{x}_1^d \ldots \mathbf{x}_{t-1}^d \mathbf{x}_t^d \mathbf{x}_{t+1}^d \ldots \mathbf{x}_T^d$.

- Input $\mathbf{x}_1^0 \ldots \mathbf{x}_{t-1}^0 \mathbf{x}_t^0 \mathbf{x}_{t+1}^0 \ldots \mathbf{x}_T^0$. 
Deep Recurrent Neural Networks

- **Output**

  \[ \mathbf{x}_t \downarrow \mathbf{x}_t^d \quad \mathbf{x}_{t+1}^d \quad \ldots \quad \mathbf{x}_T^d \]

  \[
  \begin{array}{cccc}
  \theta^3 & \quad \theta^3 & \quad \theta^3 \quad \ldots \quad \theta^3 \\
  \theta^2 & \quad \theta^2 & \quad \theta^2 \quad \ldots \quad \theta^2 \\
  \theta^1 & \quad \theta^1 & \quad \theta^1 \quad \ldots \quad \theta^1 \\
  \end{array}
  \]

  Many paths through which long-term dependencies can be propagated through the network

- **Input**

  \[ \mathbf{x}_1^0 \quad \ldots \quad \mathbf{x}_{t-1}^0 \quad \mathbf{x}_t^0 \quad \mathbf{x}_{t+1}^0 \quad \ldots \quad \mathbf{x}_T^0 \]
Recurrent Neural Networks

- Widely used in language processing:
  - Speech recognition,
  - Machine translation,

- Output \( x_1^d \ldots x_{t-1}^d x_t^d x_{t+1}^d \ldots x_T^d \)

- Input \( x_1^0 \ldots x_{t-1}^0 x_t^0 x_{t+1}^0 \ldots x_T^0 \)
Summary

- Computational model of neural information processing.
- Feed-forward networks: layer-wise matrix multiplication + activation function.
- Back propagation: stochastic gradient descent. Gradient computation by layer-wise matrix multiplication + derivative of activation function.
- Recurrent neural networks: state information passed on to next time step.