Mathematical Foundations of Deep Learning

Ather Gattami
Senior Research Scientist
RISE SICS
Stockholm, Sweden

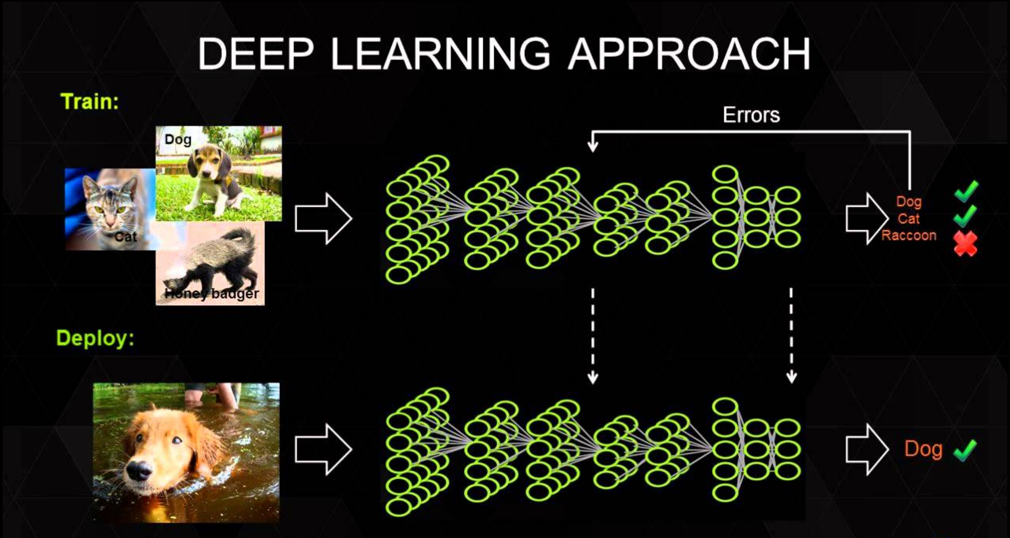
February 14, 2018

Outline

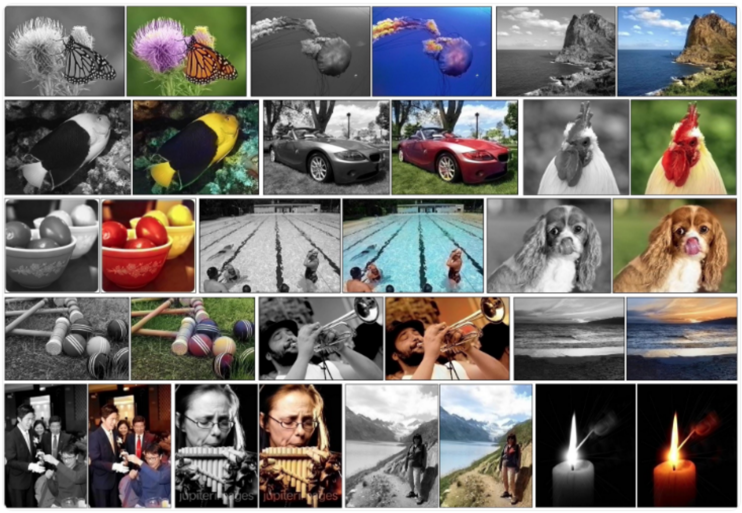
Applications

APPLICATIONS

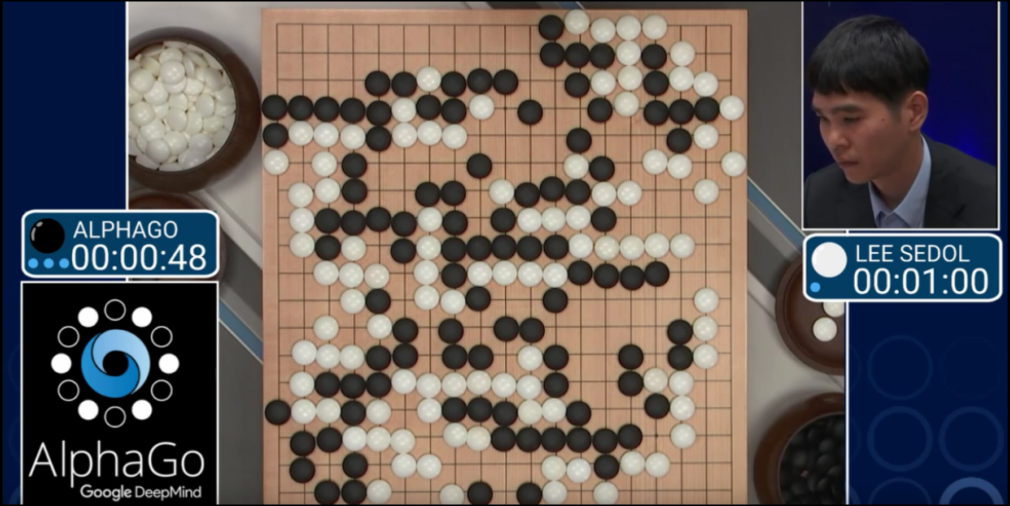
Image Classification



Colorization



AlphaGo

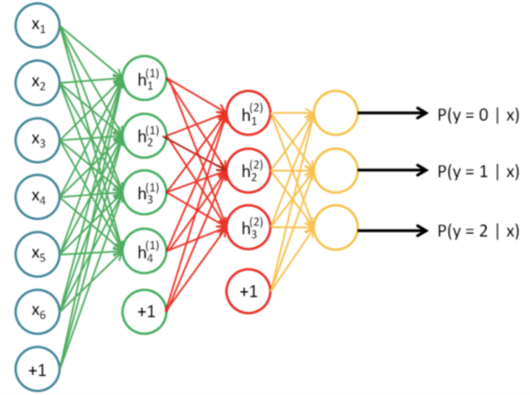


Video

Structure & Training

STRUCTURE & TRAINING

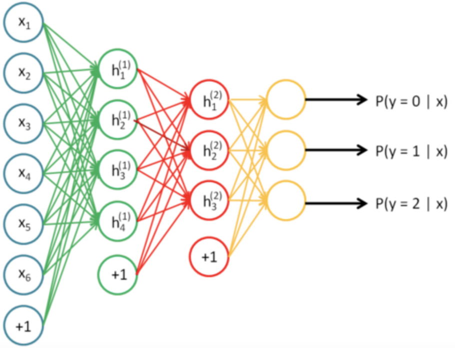
Neural Networks



Neural Networks

$$h_1^{(1)} = g(w_{11}^{(0)} x_1 + \cdots w_{1d_x}^{(0)} x_{d_x} + b_1^{(1)} \cdot 1)$$
$$h_2^{(1)} = g(w_{21}^{(0)} x_1 + \cdots w_{2d_x}^{(0)} x_{d_x} + b_2^{(0)} \cdot 1)$$
$$\vdots$$
$$h_{d_1}^{(1)} = g(w_{d_1 1}^{(0)} x_1 + \cdots w_{d_1 d_x}^{(0)} x_{d_x} + b_{d_1}^{(0)} \cdot 1)$$

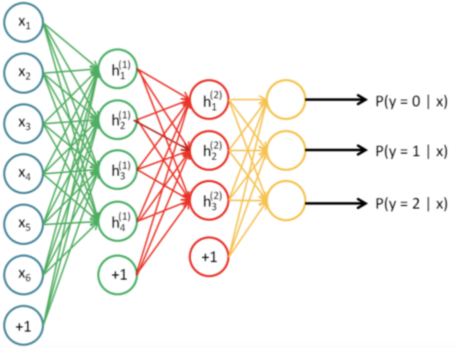
$$\mathbf{h^{(1)}} = \mathbf{g}(\mathbf{W^{(0)}x} + \mathbf{b^{(0)}})$$



Neural Networks

$$h_1^{(2)} = g(w_{11}^{(1)} h_1^{(1)} + \cdots w_{1d_1}^{(1)} h_{d_1}^{(1)} + b_1^{(1)} \cdot 1)$$
$$h_2^{(2)} = g(w_{21}^{(1)} h_1^{(1)} + \cdots w_{2d_1}^{(1)} h_{d_1}^{(1)} + b_2^{(1)} \cdot 1)$$
$$\vdots$$
$$h_{d_2}^{(2)} = g(w_{d_2 1}^{(1)} h_1^{(1)} + \cdots w_{d_2 d_1}^{(1)} h_{d_1}^{(1)} + b_{d_2}^{(1)} \cdot 1)$$

$$\mathbf{h}^{(2)} = \mathbf{g}(\mathbf{W}^{(1)} \mathbf{h}^{(1)} + \mathbf{b}^{(1)})$$



Neural Networks

Similarly,

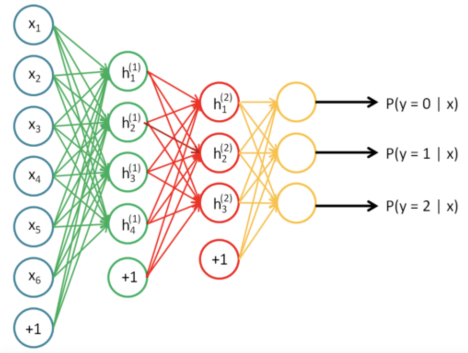
$$\mathbf{h^{(L)}} = \mathbf{g}(\mathbf{W^{(L-1)}} \mathbf{h^{(L-1)}} + \mathbf{b^{(L-1)}})$$

Fitting the neural network parameters to the pairs (x_i, y_i) is to minimize

$$J(W, b) = \sum_{i=1}^{N} \frac{1}{2} |h^{(L)}(W, b, x_i) - y_i|^2$$

Gradient descent to find the optimum values of W, b

$$W^{(i)} \leftarrow W^{(i)} - \alpha \frac{\partial J(W, b)}{\partial W^{(i)}}$$



Neural Networks

Forward-backward propagation to calculate the gradient $\frac{\partial J(W,b)}{\partial W^{(i)}}$.

Feedforward pass to caculate $h^{(1)}, \ldots, h^{(L)}$.

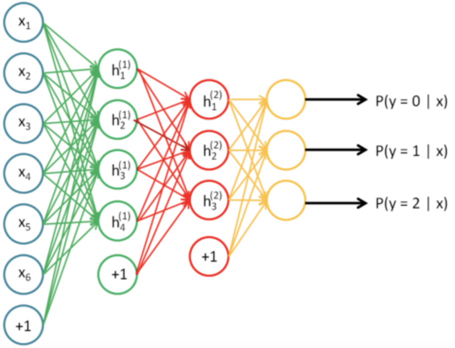
Compute

$$\delta^{(L)} = (h^{(L)} - y) \odot g'(W^{(L-1)} h^{(L-1)} + b^{(L-1)})$$

Backward propagation for $l = L-1, \ldots, 2$

$$\delta^{(l)} = ((W^{(l)})^\intercal \delta^{(l+1)}) \odot g'(W^{(l-1)} h^{(l-1)} + b^{(l-1)})$$

$$\frac{\partial J(W,b)}{\partial W^{(l)}} = \delta^{(l+1)} \cdot (h^{(l)})^\intercal$$



Complexity

COMPLEXITY

Neural Networks

Note that the optimization problem

$$\min_{W,b} \frac{1}{2}|h^{(L)}(W, b, x) - y|^2$$

is never convex unless $h^{(L)}(W, b, x)$ is linear in (W, b). Why?

Neural Networks

Note that the optimization problem

$$\min_{W,b} \frac{1}{2}|h^{(L)}(W,b,x) - y|^2$$

is never convex unless $h^{(L)}(W,b,x)$ is linear in (W,b). Why?

The minimization is equivalent to the optimization problem

$$\begin{aligned} \min_{W,b,\gamma} \quad & \gamma \\ \text{subject to} \quad & h^{(L)}(W,b,x) - y \leq \gamma \\ & h^{(L)}(W,b,x) - y \geq -\gamma \end{aligned}$$

Neural Networks

Challenge: Study the quality of gradient descent based optimization for the activation (ReLU) function $\max(0, x)$

$$\min_{w,b} \sum_{i=1}^{N} \frac{1}{2} |\max(0, w^\intercal x_i + b) - y_i|^2$$

We can rewrite as

$$\begin{aligned} \min_{w,b,\gamma} \quad & \sum_{i=1}^{N} \frac{1}{2} \gamma_i^2 \\ \text{subject to} \quad & \max(0, w^\intercal x_i + b) - y_i \leq \gamma_i \\ & \max(0, w^\intercal x_i + b) - y_i \geq -\gamma_i \\ & i = 1, ..., N \end{aligned}$$

Mathematical foundations

MATHEMATICAL FOUNDATIONS OF NEURAL NETWORKS

Neural Networks

But how good are neural networks as function approximators?

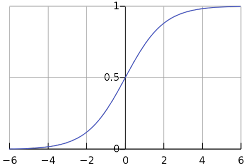
Neural Networks

But how good are neural networks as function approximators?

Neural Networks are universal approximators!

Let σ denote the sigmoidal function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



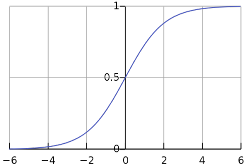
Neural Networks

But how good are neural networks as function approximators?

Neural Networks are universal approximators!

Let σ denote the sigmoidal function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Theorem (Cybenko, 1989)

The set of functions of the form

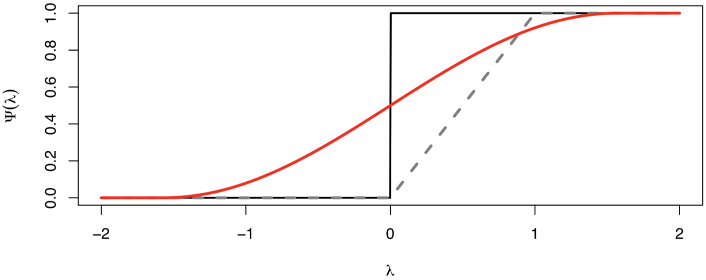
$$\sum_{j=1}^{N} w_j^{(2)} \sigma((w^{(1)})^\intercal x + b_j)$$

where $w^{(1)} \in \mathbb{R}^n$, *and* $w_j^{(2)}, b_j \in \mathbb{R}$, *are dense in the space of continuous functions in the range* $x \in [-1, 1]^n$.

Neural Networks

Definition: A function $\Psi : \mathbb{R} \to [0, 1]$ is a squashing function if it is non-decreasing, $\lim_{\lambda \to \infty} \Psi(\lambda) = 1$ and $\lim_{\lambda \to -\infty} \Psi(\lambda) \to 0$.

Squashing functions



Neural Networks

Let M^r be the set of Borel mearsurable functions $f : \mathbb{R}^r \to \mathbb{R}$, and let

$$\mathbf{A}^r = \{A(x) \mid A : \mathbb{R}^r \to \mathbb{R},\ A(x) = w^\intercal x + b\}$$

For any Borel measurable mapping $G : \mathbb{R} \to \mathbb{R}$, define

$$\Sigma^r(G) =$$

$$= \{f : \mathbb{R}^r \to \mathbb{R} \mid f(x) = \sum_{i=1}^{q} \beta_i G(A_j(x)), x \in \mathbb{R}^r, \beta_j \in \mathbb{R}, A_j \in \mathbf{A}^r, q \in \mathbb{N}\}$$

Neural Networks

English

There is a single hidden layer feedforward network that approximates any measurable function to any desired degree of accuracy on some compact set K.

Math

For every function g in M^r there is a compact subset K of $\mathbb{R}^r$ and an $f \in \Sigma^r(\psi)$ such that for any $\epsilon > 0$ we have $\mu(K) > 1 - \epsilon$ and for every $x \in K$ we have $|f(x) - g(x)| < \epsilon$, regardless of ψ, r, or the measure μ. (Hornik, 1989)

Neural Networks

English

Functions with finite support can be approximated exactly with a single hidden layer.

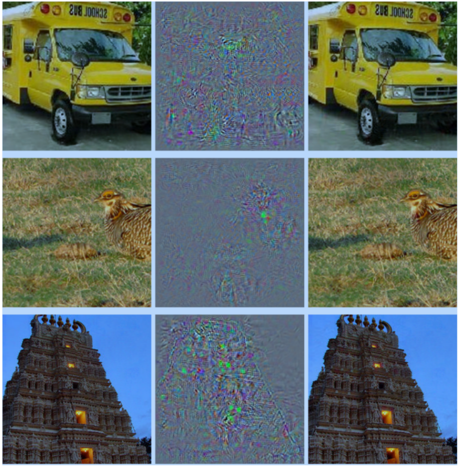
Math

Let $\{x_1, ..., x_n\}$ be a set of distinct points in $\mathbb{R}^r$ and let $g : \mathbb{R}^r \to \mathbb{R}$ be an arbitrary function. If Ψ achieves 0 and 1, then there is a function $f \in \Sigma^r(\Psi)$ with n hidden units such that $f(x_i) = g(x_i)$ for all i.

Sensitivity of Neural Networks

SENSITIVITY OF NEURAL NETWORKS

Sensitivity of Neural Networks



[Intriguing properties of neural networks]

Neural Networks

Robust training:

Fitting the neural network parameters to the pairs $(x_i + \epsilon_i, y_i)$ to minimize

$$J(W,b) = \max_{|\epsilon| \leq c} \sum_{i=1}^{N} \frac{1}{2} |h^{(L)}(W, b, x_i + \epsilon_i) - y_i|^2$$

subject to

$$h^{(L)} = g(W^{(L-1)} h^{(L-1)} + b^{(L-1)}), \qquad h^{(0)} = x_i + \epsilon_i$$

Time Series Prediction

TIME SERIES PREDICTION

Times Series Prediction and Neural Networks

Consider the time series give by T data samples

$$(x_1, y_1), (x_2, y_2), ..., (x_t, y_t), ..., (x_T, y_T)$$

Suppose that

$$h_t = f(x_t, h_{t-1})$$
$$y_t = g(h_t)$$

for some measurable functions $f : \mathbb{R}^{m+n} \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}^p$.

Approximate f and g with neural networks.

Recurrent Neural Networks

Approximate f and g with neural networks

$$h_{t+1} = \sigma_h(W_{xh}x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_{hy}h_t + b_y)$$

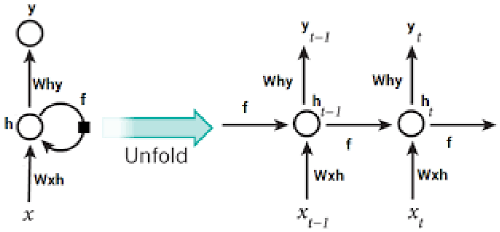


Figure: Is this structure good enough?

Recurrent Neural Networks

$$\min_{W,U,b} \sum_{t=1}^{T} |\sigma_y(W_{hy}h_t + b_y) - y_t|^2$$

subject to

$$h_{t+1} = \sigma_h(W_{xh}x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_{hy}h_t + b_y)$$

Forward-backward propagation over *layers* and *time*.

Unfolding over time gives a chain of T hidden layers.

Important constraint: The weights $W_{xh}, U_h, b_h, W_{hy}, b_y$ are *identical* for each layer.

Recurrent Neural Networks

Solution: Forward pass $h_1, ..., h_T$.

Consider each parameter as if it was *different* for each layer:

$$h_{t+1} = \sigma_h(W_{xh}^{(t)} x_t + U_h^{(t)} h_{t-1} + b_h^{(t)})$$
$$y_t = \sigma_y(W_{hy}^{(t)} h_t + b_y^{(t)})$$

This is called Back-Propagation Through Time (BPTT).

Long Short Term Memory

However, there is a problem of *exploding/vanishing gradient* in RNN.

Suppose that $\sigma_h = \sigma_y = \mathbf{id}$ (or ReLU) in

$$h_{t+1} = \sigma_h(W_{xh}x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_{hy}h_t + b_y)$$

Then we see that

$$h_t = U_h^{t-1} h_1 + \cdots$$

One solution is to introduce **Long Short Term Memory (LSTM)**.

Long Short Term Memory

LONG SHORT TERM MEMORY

Applications of LSTM

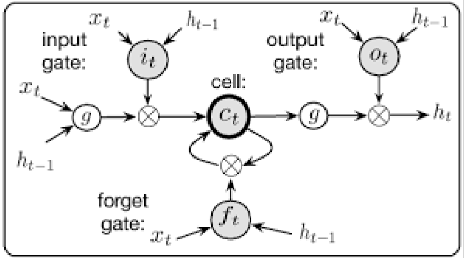
- Handwriting recognition
- Speech recognition
- Handwriting generation
- Machine translation
- Image captioning
- Text parsing
- Prediction of stock price evolution

Long Short Term Memory

The solution is to introduce Long Short Term Memory (LSTM)

$o_t = y_t$ in the figure below.

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$
$$y_t = \sigma(W_y x_t + U_y h_{t-1} + b_y)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$
$$h_t = y_t \odot \sigma_h(c_t)$$



with $c_0 = 0$ and $h_0 = 0$.

Remark: $h_0 = 0$ is a rough approximation.

Long Short Term Memory

Consider

$$h_k = Ah_{k-1} + Bx_k$$
$$y_k = Ch_k$$

Then,

$$h_k = A^k h_0 + \sum_{t=1}^{k} A^{k-t} B x_t$$

LSTM should be used cautiously for regression

Simulations

SIMULATIONS

Example

Consider the dynamic system

$$y_t = (x_t + y_{t-1}) \mod (2)$$

y_0 and the input $x_t \in \{0, 1\}$ are randomly generated for $t = 1, ..., T$, $T = 10^5$.

The memory in the dynamic system is of order 1

$$y_t = f(x_t, y_{t-1})$$
$$y_t = g(y_t)$$

with $g = \mathbf{id}$.

Example

f can be approximated by a *static* neural network with L layers, with input (x_t, y_{t-1}) and output y_t.

Training of the static neural network is given by

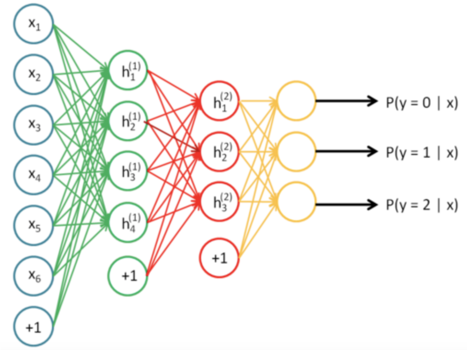
$$\min_{W,b} \sum_{t=1}^{T} |h^{(L)}(W, b, x_t, y_{t-1}) - y_t|^2$$

Training: 70% of the data samples.

Test: 30% of the remaining data.

Prediction accuracy: $\approx \mathbf{50}\%$.

Example



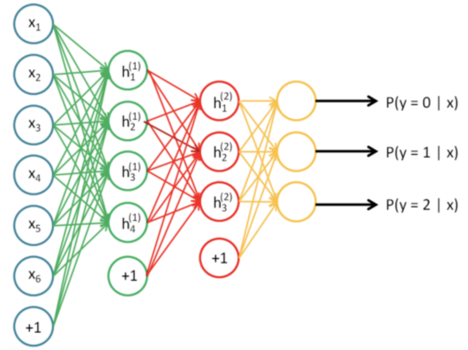
Introduce memory in the static neural network

$$y_t = f(x_t, y_{t-1}, y_{t-2}, \ldots, , y_{t-M})$$

$M = 1000$

Prediction accuracy is $\approx \mathbf{60}\%$.

Example



Introduce memory in the static neural network

$$y_t = f(x_t, y_{t-1}, y_{t-2}, \dots, , y_{t-M})$$

$M = 1000$

Prediction accuracy is $\approx \mathbf{60\%}$.

Prediction accuracy with LSTM is 100%!

End of Presentation

QUESTIONS?