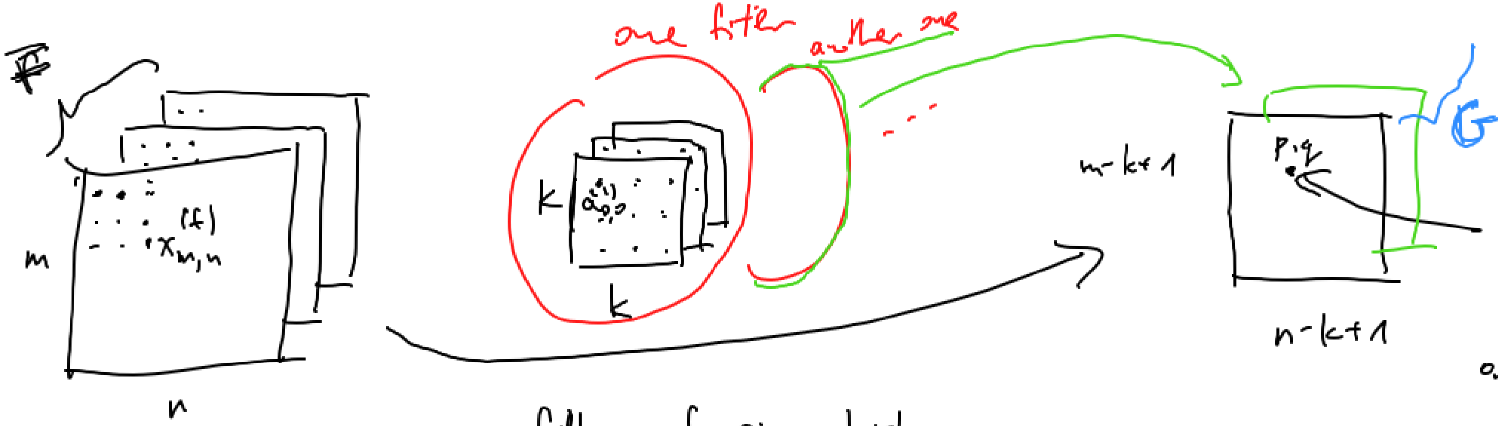


$$net_{p,q} = \sum_{f=1}^{F} \sum_{i,j=0}^{k-1} x^{(f)}_{m+i,n+j}\, a^{(f)}_{i,j} + b$$

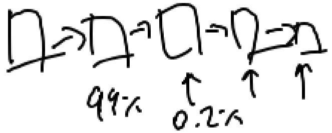
$$output_{p,q} = \varphi(net_{p,q})$$

filter of size $k \times k$,
but in fact it is
of size $\underbrace{F \times k \times k}_{\text{number of weight in the filter}} + 1$

$$\cancel{f * g^{(n)} = \sum f(n+m)\, g(m)}$$

ReLU

$$\varphi(x) = \max(x, 0)$$



Naively, how many operations do we need to calculate the output of the layer?

For each entry of the output: Fk^2 multiplications

entries in the output: $(m-k+1)(n-k+1)\cdot G \approx mnG$ (often)

(G channels)

$\rightarrow$ total $\approx Fk^2 mn \cdot G$

$128 \quad 3^2 \quad 100\ 100 \quad 128 \approx 10^3\, 10^4 \cdot 128 \approx 10^9$

Dropout

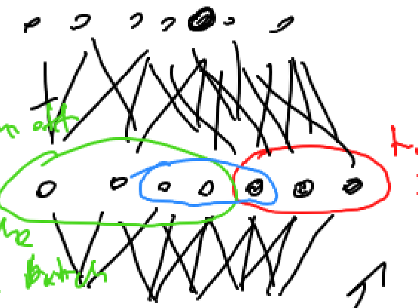
cheap (computationally) way of averaging multiple neural networks

3. layer

[2.5 layer]

2. layer

1 layer



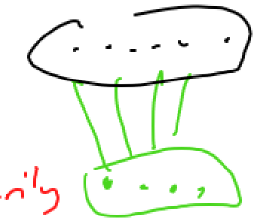
turn off

keep in the second batch

turn off temporarily

(for training only!)

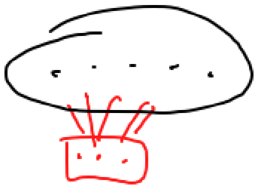
network I



For the prediction phase, we use all neurons

$\underbrace{\frac{4}{7}\,\frac{7}{4}(\text{output of } 4)}_{\text{this was used as input in training}} + \frac{3}{7}\,\underbrace{\frac{7}{3}(\text{output of } 3)} \rightarrow$ 3. layer

network II



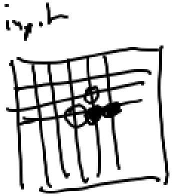
to compensate this turn off, one multiplies the output of the remaining neurons by a proper constant

$\frac{7}{4}\cdot$(output of the remaining 4 neurons) $\rightarrow$ 3. layer

$\frac{7}{3}$(— " — 3 neurons) $\rightarrow$ 3. layer

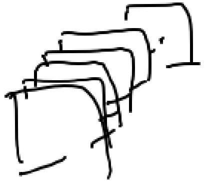
DeepForest   predicting moves in Go

input



$19^2 = 361$ intersection

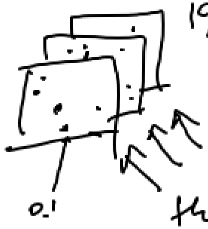
transformed →



- white stones,
- black stones
- empty intersections

CNN → same padding without strides or pooling →

3   19x19



0.1

probability the distribution of the moves made
- now
- next
- afterwards

90' — program + hardware   won with the best human chess player (Kasparov)

2017' — program + -||-   won with -||- Go

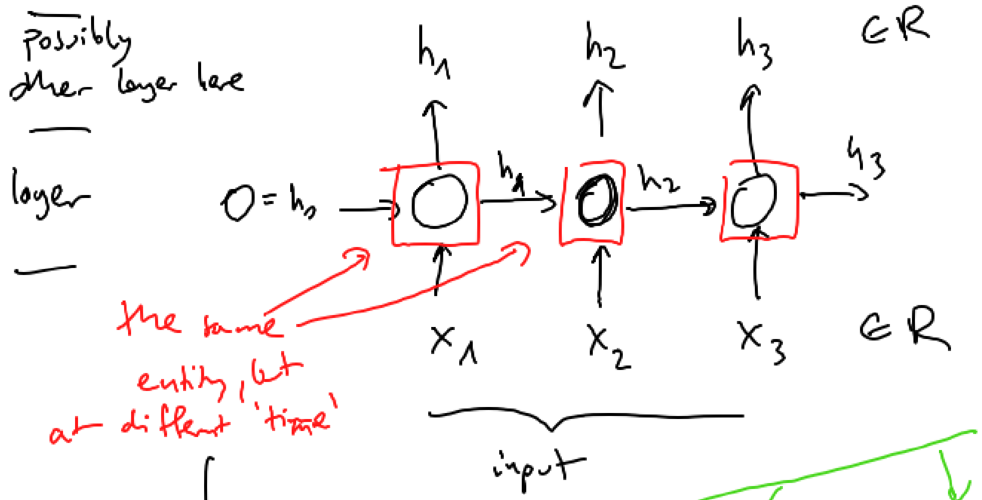
← AlphaGo

- purely CNN
- same size

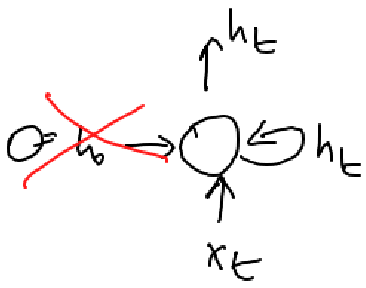
Recurrent Neural Networks (RNN)

one layer, one cell, input = sequence of length 3

Simple RNN:



unrolled picture

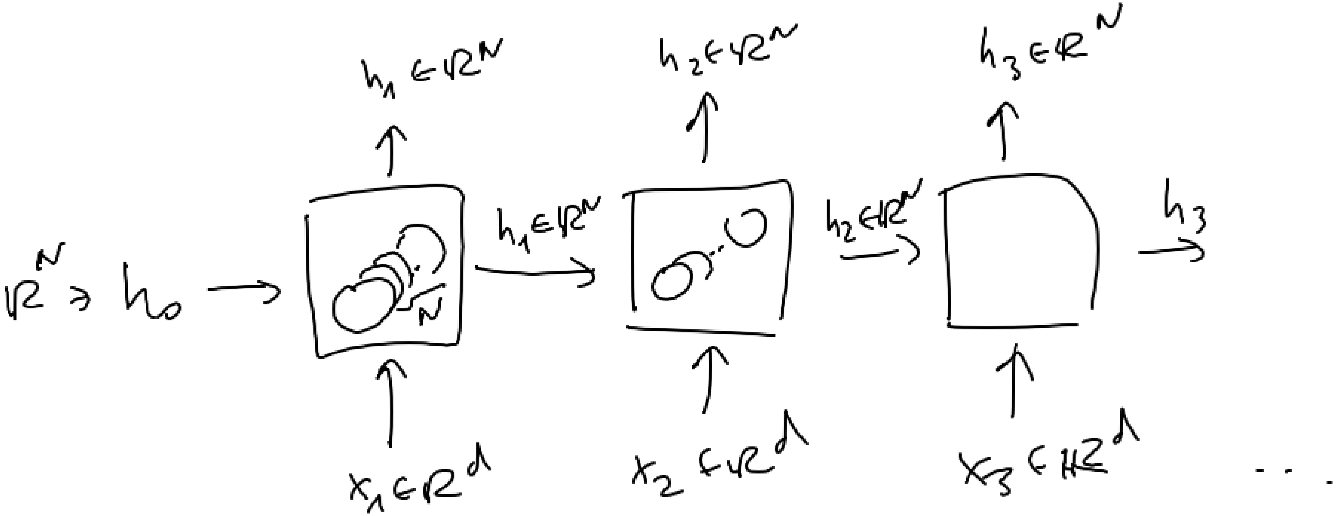


trainable parameters (weights)

$$h_t = \sigma\left(W \cdot \underset{\mathbb{R}}{x_t} + U \cdot \underbrace{h_{t-1}}_{\mathbb{R}} + b \right)$$

$W, U, b \in \mathbb{R}$

a more general case: $x_t \in \mathbb{R}^d$, N cells (neurons) in a layer



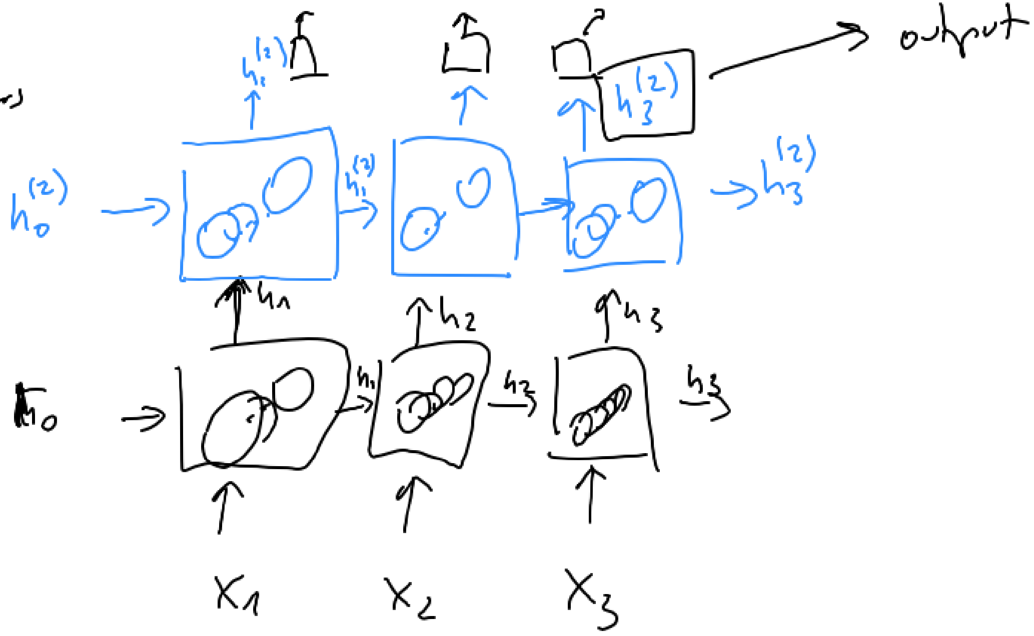
$$\underset{\mathbb{R}^N}{\underset{\cap}{h_t}} = \sigma\Big(\underset{N\times d}{W} \cdot \underset{\mathbb{R}^d}{\underset{\cap}{x_t}} + \underset{N\times N}{U} \cdot \underset{\mathbb{R}^N}{\underset{\cap}{h_{t-1}}} + \underset{\in\mathbb{R}^N}{b}\Big)$$

trainable weights

h is both the output and the 'hidden state'
This hidden state is shared between the neurons in the layer.

these can be stacked together

possible dense layers



$\underbrace{x_1, x_2, x_3}_{\text{feed the the network, get some output } y}, \underbrace{x_4}_{\text{the expected output}}$

He . . . . $\boxed{\text{his}}$

She $\boxed{\text{her}}$

LSTM (Long-Short Term Memory) 95-97-99 Hochreiter, Schmidhuber

$$\mathbb{R}^N \ni f_t = \sigma_g(\underbrace{W_f}_{N\times d} \cdot \underbrace{x_t}_{\in\mathbb{R}^d} + \underbrace{U_f}_{N\times N} \underbrace{h_{t-1}}_{\in\mathbb{R}^N} + \underbrace{b_f}_{\in\mathbb{R}^N})$$ forget gate

σ_g = sigmoid

σ_c = tanh

$$\mathbb{R}^N \ni i_t = \sigma_g(\underbrace{W_i}\cdot x_t + \underbrace{U_i} h_{t-1} + \underbrace{b_i})$$ input gate

$$o_t = \sigma_g(\underbrace{W_o} x_t + \underbrace{U_o} h_{t-1} + \underbrace{b_o})$$ output gate

$$\tilde{c}_t = \sigma_c(\underbrace{W_c} x_t + \underbrace{U_c} h_{t-1} + \underbrace{b_c}) \in\mathbb{R}^N$$

weights

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$ memory cell

$\odot$ coordinate-wise multiplication

$$h_t = o_t \odot \sigma_h(c_t) \in\mathbb{R}^N$$ output

jth coordinate

If $f_t^{(j)} \simeq 1$ and $i_t(j) \simeq 0$, then $c_t^{(j)} \simeq c_{t-1}(j)$ (old value is kept) $c \to \to \to \to \to \ldots$

$f_t^{(j)} \simeq 0$ and $i_t(j) \simeq 1$, then $c_t(j) \simeq \tilde{c}_t(j)$ (old value is discarded)