

Hopfield Networks is all you need

Shabalin Evgeniy

13.10.2020

Classical Hopfield Networks

- The most popular Associative Memory model.
- Presented in 1982.
- Used to store and retrieve patterns.

Classical Hopfield Networks

The simplest associative memory: sum of the outer products of N patterns $\{x_i\}_{i=1}^N$ where $x_i \in \{-1, 1\}^d$

$$W = \sum_{i=1}^N x_i x_i^T$$

Classical Hopfield Networks

The simplest associative memory: sum of the outer products of N patterns $\{x_i\}_{i=1}^N$ where $x_i \in \{-1, 1\}^d$

$$W = \sum_{i=1}^N x_i x_i^T$$

From now on we denote the N stored patterns as $\{x_i\}_{i=1}^N$ and any state pattern or state as ξ

Update rules

Synchronous:

$$\xi^{t+1} = (W\xi^t - b) ,$$

where $b \in \mathbb{R}^d$ is a bias vector, which can be interpreted as threshold for every component.

We stop when $\xi^{t+1} = \xi^t$

Update rules

Synchronous:

$$\xi^{t+1} = (W\xi^t - b)$$

Asynchronous:

Update each component of ξ separately and then choose next component to update. This is equivalent to minimizing energy function

$$E = -\frac{1}{2}\xi^T W \xi + \xi^T b = -\frac{1}{2} \sum_{i=1}^d \sum_{j=1}^d \xi_i \xi_j w_{ij} + \sum_{i=1}^d \xi_i b_i$$

Convergence

The convergence properties are dependent on the structure of the weight matrix \mathbf{W} and the method by which the nodes are updated:

- For synchronous updates with $w_{ij} = w_{ji}$, the updates converge to a stable state or a limit cycle of length 2.
- For asynchronous updates with $w_{ii} \geq 0$ and $w_{ij} = w_{ji}$, the updates converge to a stable state.

Example of Hopfield Network



The weight matrix \mathbf{W} is the outer product of this black and white image x_{Homer} :

$$\mathbf{W} = x_{Homer} x_{Homer}^T, \quad x_{Homer} \in \{-1, 1\}^d,$$

where for this example $d = 64 \times 64$.

Example of Hopfield Network

Now we want from network to retrieve masked picture of Homer

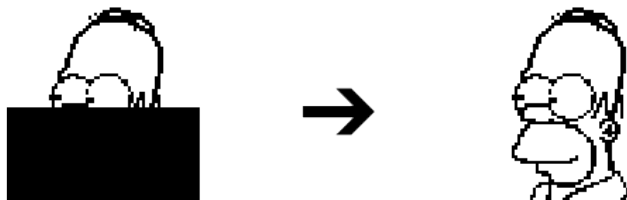


which is our initial state ξ . This initial state is updated via our simple update rule.

Note: in this example we do not use bias vector b

Example of Hopfield Network

So, it takes only update until the original image is restored.



Example of Hopfield Network

Now we want to store multiple patterns. Our weight matrix is now sum of outer products

$$\mathbf{W} = \sum_{i=1}^3 \mathbf{x}_i \mathbf{x}_i^T, \quad x_{Homer} \in \{-1, 1\}^d,$$



Example of Hopfield Network

Now we want to store multiple patterns. Our weight matrix is now sum of outer products

$$\mathbf{W} = \sum_{i=1}^3 \mathbf{x}_i \mathbf{x}_i^T, \quad x_{Homer} \in \{-1, 1\}^d,$$



Example of Hopfield Network

Let's try 6 patterns



Storage capacity

The storage capacity for retrieval of patterns **free of errors** is:

$$C \cong \frac{d}{2 \ln d}$$

Storage capacity

The storage capacity for retrieval of patterns **free of errors** is:

$$C \cong \frac{d}{2 \ln d}$$

The storage capacity for retrieval of patterns **with a small percentage of errors** is:

$$C \cong 0.14d$$

Storage capacity

The storage capacity for retrieval of patterns **free of errors** is:

$$C \cong \frac{d}{2 \ln d}$$

The storage capacity for retrieval of patterns **with a small percentage of errors** is:

$$C \cong 0.14d$$

In example above: $0.14 \cdot 64 \cdot 64 \approx 570$, this means that capacity of network is not directly responsible for retrieval errors

Modern Hopfield Networks

New energy function:

$$E = - \sum_i^N F(x_i^T \xi),$$

Where F is an interaction function

They choose a polynomial interaction function $F(z) = z^a$, $a \in \mathbb{N}$.

Modern Hopfield Networks

New energy function:

$$E = - \sum_i^N F(\mathbf{x}_i^T \boldsymbol{\xi}),$$

The storage capacity for retrieval of patterns free of errors is:

$$C \cong \frac{1}{2(2a-3)!!} \frac{d^{a-1}}{\log(d)}$$

The storage capacity for retrieval of patterns with a small percentage of errors is:

$$C \cong \alpha_a d^{a-1}$$

Modern Hopfield Networks

Another energy function:

$$E = - \sum_i^N \exp(x_i^T \xi),$$

Which leads to storage capacity:

$$C \cong 2^{\frac{d}{2}}$$

New update rule

Denote $\xi[l]$ as l -th component of ξ

$$\xi^{new}[l] = \text{sgn} \left[-E \left(\xi^{(l+)} \right) + E \left(\xi^{(l-)} \right) \right],$$

where $\xi^{(l+)}[l] = 1$ and $\xi^{(l-)}[l] = -1$ and $\xi^{(l+)}[k] = \xi^{(l-)}[k] = \xi[k]$ for $k \neq l$.

New update rule

Denote $\xi[l]$ as l -th component of ξ

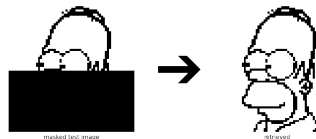
$$\xi^{new}[l] = \text{sgn} \left[-E \left(\xi^{(l+)} \right) + E \left(\xi^{(l-)} \right) \right],$$

where $\xi^{(l+)}[l] = 1$ and $\xi^{(l-)}[l] = -1$ and $\xi^{(l+)}[k] = \xi^{(l-)}[k] = \xi[k]$ for $k \neq l$.

Paper states that with this update rule energy function with $F(z) = e^z$ converges with high probability in just one asynchronous update

Retrieval ability

Now we can successfully retrieve our beloved Homer from 24 patterns



Continuous valued patterns

We want to deal not only with binary patterns but also with continuous valued ones. So we modify our energy function:

$$\text{lse}(\beta, \mathbf{z}) = \beta^{-1} \log \left(\sum_{l=1}^N \exp(\beta z_l) \right)$$

$$E = -\text{lse}(\beta, X^T \xi) + \frac{1}{2} \xi^T \xi + \beta^{-1} \log N + \frac{1}{2} M^2,$$

$X = \{\mathbf{x}_i\}_{i=1}^N$, M is the largest norm of all stored patterns.

Brand new update rule

Total energy is split into convex and concave terms: $E(\xi) = E_{\text{vex}}(\xi) + E_{\text{cave}}(\xi)$

- $E_{\text{vex}}(\xi) = \frac{1}{2}\xi^T \xi + \text{const}$ is convex term
- $E_{\text{cave}}(\xi) = -\text{lse}(\beta, X^T \xi)$ is concave term

And now we apply Concave-Convex-Procedure (CCCP)

Concave-Convex-Procedure (CCCP)

$$\nabla_{\xi} E_{\text{vex}}(\xi^{t+1}) = -\nabla_{\xi} E_{\text{cave}}(\xi^t)$$

$$\nabla_{\xi} \left(\frac{1}{2} \xi^T \xi + C \right) (\xi^{t+1}) = \nabla_{\xi} \text{lse}(\beta, X^T \xi^t)$$
$$(\xi^{t+1}) = X \text{Softmax}(\beta X^T \xi^t)$$

Therefore, The update rule for a state pattern ξ :

$$\xi^{\text{new}} = X \text{Softmax}(\beta X^T \xi^t)$$

Continuous retrieval

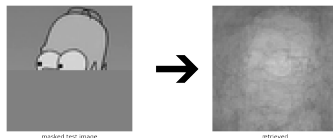
Now we conduct the same experiment as above, but now in continuous form:



It can be seen again that Homer is perfectly retrieved.

Continuous retrieval

Remember, we had β in our Softmax. It is an inverse temperature.
Now we use $\beta = 0.5$ instead of $\beta = 8$



We can see so called metastable state.

Continuous retrieval



masked



retrieved

$\text{beta} = 0.25$



masked



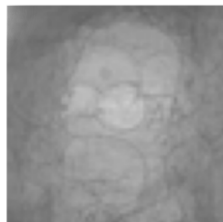
retrieved

$\text{beta} = 0.50$

Continuous retrieval



masked



retrieved

$\text{beta} = 1.00$



masked



retrieved

$\text{beta} = 2.00$

Continuous retrieval



masked



$\text{beta} = 4.00$



retrieved



masked



$\text{beta} = 8.00$



retrieved

Self-attention of Transformer

For S states $\Xi = (\xi_1, \dots, \xi_S)$ we can generalize our update rule to:

$$\Xi^{new} = X \text{Softmax}(\beta X^T \Xi^t)$$

and rewrite our variables in terms of different associative spaces:

$$Q = \Xi = RW_Q$$

$$K = X^T = YW_K$$

$$\beta = \frac{1}{\sqrt{d_k}}$$

Self-attention of Transformer

So, we obtain:

$$(Q^{new})^T = K^T \text{Softmax} \left(\frac{1}{\sqrt{d_k}} K Q^T \right)$$

$$Q^{new} = \text{Softmax} \left(\frac{1}{\sqrt{d_k}} Q K^T \right) K$$

Self-attention of Transformer

$$Q^{new} = \text{Softmax} \left(\frac{1}{\sqrt{d_k}} QK^T \right) K$$

Now we also want to map out output to another space via another projection matrix:

$$Z = Q^{new} W_V = \text{Softmax} \left(\frac{1}{\sqrt{d_k}} QK^T \right) KW_V = \text{Softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

Self-attention of Transformer

To sum up everything:

We have raw inputs $Y = \{y_i\}_{i=1}^N$ and raw states $R = \{\xi_i\}_{i=1}^S$ and 3 projection matrices W_K , W_Q and W_V . And this results to

$$Z = \text{Softmax} \left(\beta \cdot R W_Q W_K^T Y^T \right) Y W_K W_V$$

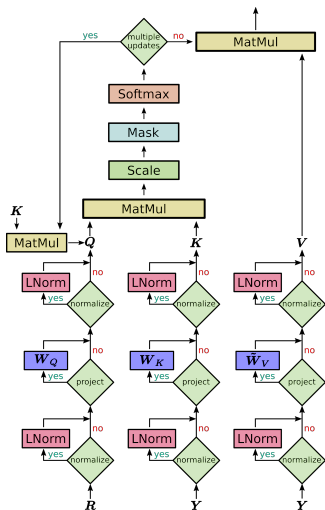
This is the fundament of a new PyTorch Hopfield layer.

Hopfield layer

Functionalities of Hopfield layer compared to self-attention:

- Association of two sets
- Variable β that determines the kind of fixed points
- Multiple updates for precise fixed points
- Dimension of the associative space for controlling the storage capacity
- Static patterns for fixed pattern search
- Pattern normalization to control the fixed point dynamics by norm and shift of the patterns

Hopfield layer



Hopfield layer

$$Z = \text{Softmax} \left(\beta \cdot R W_Q W_K^T Y^T \right) Y W_K W_V$$

For more flexibility we can denote

$$\tilde{W}_V = W_K W_V$$

Hopfield layer

$$Z = \text{Softmax} \left(\beta \cdot R W_Q W_K^T Y^T \right) Y W_K W_V$$

For more flexibility we can denote

$$\tilde{W}_V = W_K W_V$$

$$\mathbf{Z} = \text{softmax} \left(\beta \quad \mathbf{R} \quad \mathbf{W}_Q \quad \mathbf{W}_K^T \quad \mathbf{Y}^T \right) \quad \mathbf{Y} \quad \mathbf{W}_K \quad \mathbf{W}_V$$

$$\begin{bmatrix} \text{blue} & \text{blue} & \text{blue} & \text{blue} \\ \text{blue} & \text{blue} & \text{blue} & \text{blue} \end{bmatrix} = \text{softmax} \left(\text{yellow} \begin{bmatrix} \text{green} & \text{green} \\ \text{green} & \text{green} \end{bmatrix} \begin{bmatrix} \text{green} & \text{green} & \text{green} \\ \text{green} & \text{green} & \text{green} \\ \text{green} & \text{green} & \text{green} \end{bmatrix} \begin{bmatrix} \text{purple} & \text{purple} & \text{purple} \\ \text{purple} & \text{purple} & \text{purple} \\ \text{purple} & \text{purple} & \text{purple} \end{bmatrix} \begin{bmatrix} \text{pink} \\ \text{pink} \\ \text{pink} \end{bmatrix} \right) \begin{bmatrix} \text{pink} & \text{pink} & \text{pink} \\ \text{pink} & \text{pink} & \text{pink} \end{bmatrix} \begin{bmatrix} \text{purple} & \text{purple} & \text{purple} \\ \text{purple} & \text{purple} & \text{purple} \\ \text{purple} & \text{purple} & \text{purple} \end{bmatrix} \begin{bmatrix} \text{blue} & \text{blue} & \text{blue} & \text{blue} \\ \text{blue} & \text{blue} & \text{blue} & \text{blue} \\ \text{blue} & \text{blue} & \text{blue} & \text{blue} \end{bmatrix}$$

Retrieval with Hopfield layer

We have raw patterns and states, so no weight matrices required

$$\mathbf{Z} = \text{softmax} \left(\beta \mathbf{R} \mathbf{Y}^T \right) \mathbf{Y}$$

$$\begin{bmatrix} \text{blue} & \text{blue} & \text{blue} & \text{blue} \end{bmatrix} = \text{softmax} \left(\begin{bmatrix} \text{yellow} \end{bmatrix} \begin{bmatrix} \text{green} & \text{green} & \text{green} & \text{green} \end{bmatrix} \begin{bmatrix} \text{pink} & \text{pink} \\ \text{pink} & \text{pink} \\ \text{pink} & \text{pink} \\ \text{pink} & \text{pink} \end{bmatrix} \right) \begin{bmatrix} \text{pink} & \text{pink} & \text{pink} & \text{pink} \\ \text{pink} & \text{pink} & \text{pink} & \text{pink} \end{bmatrix}$$

$$\begin{bmatrix} \text{Homer Simpson} \end{bmatrix} = \text{softmax} \left(\begin{bmatrix} \text{yellow} \end{bmatrix} \begin{bmatrix} \text{Homer Simpson} \end{bmatrix} \begin{bmatrix} \text{grid of 20 small Homer Simpson images} \end{bmatrix} \right) \begin{bmatrix} \text{grid of 20 small Homer Simpson images} \end{bmatrix}$$

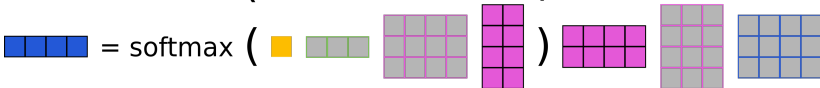
Hopfield lookup

A variant of our Hopfield-based modules which employs a trainable but input independent lookup mechanism. We train one or multiple stored patterns and pattern projections.

$$\mathbf{Z} = \text{softmax} \left(\beta \mathbf{R} \mathbf{W}_K^T \right) \mathbf{W}_K \mathbf{W}_V$$

Hopfield pooling

We consider the Hopfield layer as a pooling layer if only one static state pattern (query) exists. Then, it is de facto a pooling over the sequence.

$$\mathbf{Z} = \text{softmax} \left(\beta \quad \mathbf{Q} \quad \mathbf{W}_K^T \quad \mathbf{Y}^T \right) \quad \mathbf{Y} \quad \mathbf{W}_K \quad \mathbf{W}_V$$


The diagram illustrates the Hopfield pooling operation. The inputs to the softmax function are represented by colored grids: a blue 1x4 grid for β , a yellow 1x1 grid for \mathbf{Q} , a green 1x4 grid for the sequence, a grey 4x4 grid for \mathbf{W}_K^T , a pink 4x4 grid for \mathbf{Y}^T , a pink 4x4 grid for \mathbf{Y} , a grey 4x4 grid for \mathbf{W}_K , and a grey 4x4 grid for \mathbf{W}_V .

Immune Repertoire Classification

The immune repertoire of an individual consists of an immensely large number of immune repertoire receptors. These receptors can be represented as amino acid sequences, some of which bind to a single specific pathogen.

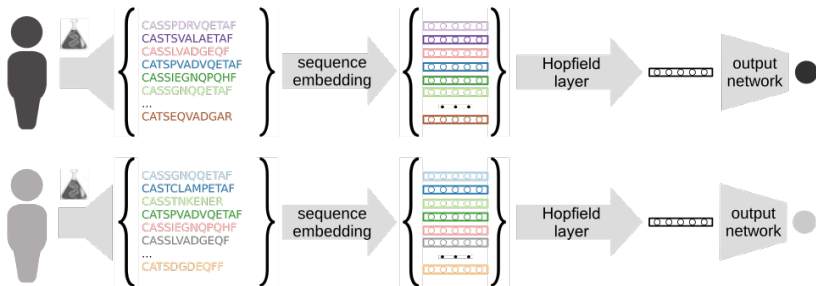
Problems:

- Sequences have variable length and consist up to 20 unique letters
- Each human has about $10^7 - 10^8$ unique immune receptors
- Only few of them (like 10^2) bind to a pathogen
- Overall there are more than 10^{14} unique receptors

Immune Repertoire Classification

Based on modern Hopfield networks, a method called **DeepRC** was designed, which consists of three parts:

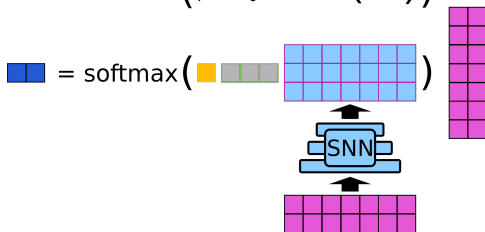
- A sequence-embedding neural network to supply a fixed-sized sequence-representation
- A Hopfield layer part for sequence-attention
- An output neural network and/or fully connected output layer



DeepRC

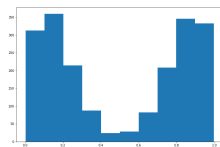
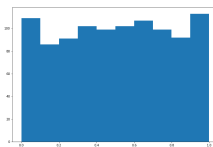
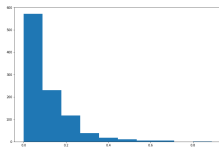
- Output of the sequence-embedding NN Y^T acts as values V
- A second neural network, e.g. a self-normalizing neural network (SNN), shares its first layers with the sequence-embedding neural network and outputs the keys K
- The query vector Q is learned and represents the variable binding sub-sequence we look for, like in pooling layer

$$\mathbf{Z} = \text{softmax}(\beta \mathbf{Q} \text{ SNN}(\mathbf{Y}^T)) \mathbf{Y}$$



Back to attention

As we mentioned before, metastable states are not always bad.
Let's look at some distributions.

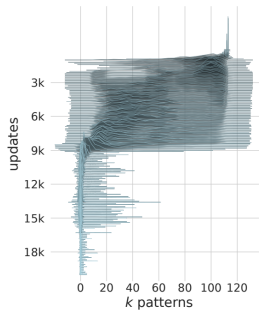


Attention layers of BERT

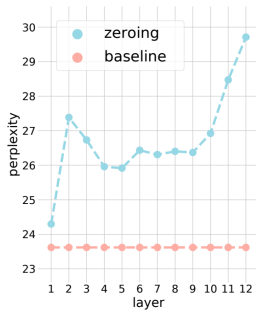
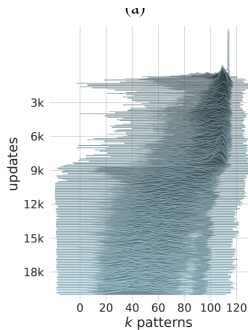


Attention layers of BERT

Head 2 in layer 6



Head 3 in layer 12



Questions

- Write an exponential energy function and an update rule for it.
- How can Hopfield Layer be used?
- What is a metastable state, how do metastable states differ and what do they represent?

References

- <https://ml-jku.github.io/hopfield-layers/>
- <https://arxiv.org/pdf/2008.02217.pdf>
- <https://www.youtube.com/watch?v=nv6oFDp6rNQ>