УДК 004.032.26

ЭВОЛЮЦИОННАЯ НЕЙРОННАЯ МОДЕЛЬ ИЖИКЕВИЧА И ЕЕ ПРИМЕНЕНИЕ

Ю. М. Вувуникян[1], Ваньли Чэнь[2]

[1] Д. ф.-м. н., профессор, профессор кафедры фундаментальной и прикладной математики УО «Гродненский государственный университет имени Янки Купалы», Гродно, Беларусь, e-mail : vuv64@mail.ru
[2] Магистр ф.-м. н., аспирант кафедры фундаментальной и прикладной математики УО «Гродненский государственный университет имени Янки Купалы», Гродно, Беларусь, e-mail : wanli19930806@gmail.com

---

Реферат

Импульсные нейронные сети [1–3] являются искусственными нейронными сетями третьего поколения. В работе предлагается метод математического моделирования таких сетей с помощью эволюционных операторов с обобщенными импульсными характеристиками, которые свертываются с прямыми степенями входных воздействий.

Преимущество модели Ходжкина-Хаксли в том, что описание нейронов очень точное, но сложность высокая и она не подходит для использования в больших нейронных сетях. Также была предложена простая модель LIF для моделирования нейронов, но некоторые свойства нейронов в такой модели игнорируются из-за того, что такая модель слишком лаконична. Е.М. Ижикевич [3[упростил модель Ходжкина-Хаксли в 2003 году. В статье рассматривается построение импульсной нейронной сети на основе модели нейронов Ижикевича.

Ключевые слова: импульсные нейронные сети, математическое моделирование импульсных нейронных сетей, математическая модель Ходжкина-Хаксли, математическая модель LIF, математическая модель Ижикевича, эволюционный нелинейный оператор, система импульсных характеристик эволюционного оператора,

EVOLUTIONARY IZHIKEVICH NEURON MODEL AND ITS APPLICATION

Y. M. Vuvunikian, Wanli Chen

Abstract

Pulse neural networks [1–3] are third generation artificial neural networks. The paper proposes a method for mathematical modeling of such networks using evolutionary operators with generalized impulse responses that convolve with direct powers of input actions.

The advantage of the Hodgkin-Huxley model is that the description of neurons is very accurate, but the complexity is high and it is not suitable for use in large networks. A simple LIF model has also been proposed for modeling neurons, but some properties of neurons in such a model are ignored due to the fact that such a model is too concise. E.M. Izhikevich [3] simplified the Hodgkin-Huxley model in 2003. The paper considers the construction of impulse neuron network based on Izhikevich neuron model.

Keywords: impulse neural networks, mathematical modeling of impulse neural networks, Hodgkin-Huxley mathematical model, LIF mathematical model, Izhikevich mathematical model, evolutionary non-linear operator, evolutionary operator impulse characteristics.

Introduction

The Izhikevich model is defined by the following system:

$$\begin{cases} \dfrac{du}{dt} = a(bv - u) \\ u + \dfrac{dv}{dt} = \alpha v^2 + \beta v + g \end{cases}$$

With the auxiliary after-spike resetting:

$$\text{If } v \geq 30\ mV \text{, then } \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases}.$$

Here, u, v and g are functions of a temporary variable t. The variable v represents the membrane potential of the neuron and u represents a membrane recovery varidble, which accounts for the activation of K^+ ionic currents and inactivation of Na^+ ionic currents, and it provides negative feedback to v. After the spike reacher it apex $(+30mV)$, the membrane voltage and recovery variable are reset.

Main part. The evolution operator of the considered model

From the first equation of the system we obtain the following equality $u = ab\theta e^{-at} * v$, where * is the convolution operation and θ - Heaviside function. Substituting the obtained into the second equation of the system, we obtain the evolutionary integro-differential equation:

$$ab\theta e^{-at} * v + \frac{dv}{dt} = \alpha v^2 + \beta v + g,$$

which can be written in the following form:

$$(ab\theta e^{-at} + \delta' - \beta\delta) * v + S_2((-\alpha\delta \otimes \delta) * v^{\otimes 2}) = g,$$

where δ is the Dirac delta function, δ' is the generalized derivative of the function δ, S_2 is the second-order variable reduction operator.

Thus, the left side of the considered equation represents a nonlinear evolution operator [4] of the second order with the following impulse characteristics:

$$a_1 = ab\theta e^{-at} + \delta' - \beta\delta, \quad a_2 = -\alpha\delta \otimes \delta \ .$$

The principle of image recognition by impulse neural network

Like the traditional ANN, when recognizing a picture, the size and data format of the picture are first converted. To put it simply, in ANN, we can perform convolution recognition by directly reading the data of the picture. In SNNs, it's similar, but with two differences. One is to perform pulse coding on the data of the picture, so that it becomes a pulse code that can be recognized by the pulse neural network; the other is to recognize the picture by using the excitement of neurons instead of performing convolution during recognition.

Bindsnet

BindsNET[2] is built on top of the PyTorch deep learning platform. It is used for the simulation of spiking neural networks (SNNs) and is geared towards machine learning and reinforcement learning.

BindsNET takes advantage of the torch.Tensor object to build spiking neurons and connections between them, and simulate them on CPUs or GPUs (for strong acceleration / parallelization) without any extra work. Recently, torchvision.datasets has been integrated into the library to allow the use of popular vision datasets in training SNNs for computer vision tasks. Neural network functionality contained in torch.nn.functional module is used to implement more complex connections between populations of spiking neurons.

At its core, BindsNET provides software objects and methods which support the simulation of groups of different types of neurons (bindsnet.network.nodes), as well as different types of connections between them (bindsnet.network.topology). These may be arbitrarily combined together under a single bindsnet.network.Network object, which is responsible for the coordination of the simulation logic of all underlying components. On creation of a network, the user can specify a simulation timestep constant, dt, which determines the granularity of the simulation. Choosing this parameter induces a trade-off between simulation speed and numerical precision: large values result in fast simulation, but poor simulation accuracy, and vice versa. Monitors (bindsnet.network.monitors) are available for recording state variables from arbitrary network components (e.g., the voltage v of a group of neurons).

Bindsnet installation

In the linux environment or google colab, use the command to install.

pip install https://github.com/BindsNET/bindsnet/archive/refs/tags/0.3.1.zip

pip install docker #Link to Docker repository,which installed BindsNET and all its dependencies.

Note that the latest version of bindsnet must be installed. If you do not specify the version to install, the data file will not be able to be loaded.

Izhikevich neuron mathematical modeling

Izhikevich neuron mathematical modeling function by python

```
class IzhikevichNodes(Nodes):
    def __init__(
        self,
        n: Optional[int] = None,
        shape: Optional[Iterable[int]] = None,
        traces: bool = False,
        traces_additive: bool = False,
        tc_trace: Union[float, torch.Tensor] = 20.0,
        trace_scale: Union[float, torch.Tensor] = 1.0,
        sum_input: bool = False,
        excitatory: float = 1,
        thresh: Union[float, torch.Tensor] = 45.0,
        rest: Union[float, torch.Tensor] = -65.0,
        lbound: float = None,
        **kwargs,
    ) -> None:
```

Instantiates a layer of Izhikevich neurons.

```
        :param n: The number of neurons in the layer.
        :param shape: The dimensionality of the layer.
        :param traces: Whether to record spike traces.
        :param traces_additive: Whether to record spike traces additively.
        :param tc_trace: Time constant of spike trace decay.
        :param trace_scale: Scaling factor for spike trace.
        :param sum_input: Whether to sum all inputs.
        :param excitatory: Percent of excitatory (vs. inhibitory) neurons in the layer;
          in range ``[0, 1]``.
        :param thresh: Spike threshold voltage.
        :param rest: Resting membrane voltage.
        :param lbound: Lower bound of the voltage.
        """
        super().__init__(
            n=n,
            shape=shape,
            traces=traces,
            traces_additive=traces_additive,
            tc_trace=tc_trace,
            trace_scale=trace_scale,
            sum_input=sum_input,
        )
        self.register_buffer("rest", torch.tensor(rest))  # Rest voltage.
        self.register_buffer("thresh", torch.tensor(thresh))  # Spike threshold voltage.
        self.lbound = lbound
        self.register_buffer("r", None)
        self.register_buffer("a", None)
        self.register_buffer("b", None)
```

```
self.register_buffer("c", None)
self.register_buffer("d", None)
self.register_buffer("S", None)
self.register_buffer("excitatory", None)
if excitatory > 1:
    excitatory = 1
elif excitatory < 0:
    excitatory = 0
if excitatory == 1:
    self.r = torch.rand(n)
    self.a = 0.02 * torch.ones(n)
    self.b = 0.2 * torch.ones(n)
    self.c = -65.0 + 15 * (self.r**2)
    self.d = 8 - 6 * (self.r**2)
    self.S = 0.5 * torch.rand(n, n)
    self.excitatory = torch.ones(n).byte()
elif excitatory == 0:
    self.r = torch.rand(n)
    self.a = 0.02 + 0.08 * self.r
    self.b = 0.25 - 0.05 * self.r
    self.c = -65.0 * torch.ones(n)
    self.d = 2 * torch.ones(n)
    self.S = -torch.rand(n, n)
    self.excitatory = torch.zeros(n).byte()
else:
    self.excitatory = torch.zeros(n).byte()
    ex = int(n * excitatory)
    inh = n - ex
    # init
    self.r = torch.zeros(n)
    self.a = torch.zeros(n)
    self.b = torch.zeros(n)
    self.c = torch.zeros(n)
    self.d = torch.zeros(n)
    self.S = torch.zeros(n, n)
    # excitatory
    self.r[:ex] = torch.rand(ex)
    self.a[:ex] = 0.02 * torch.ones(ex)
    self.b[:ex] = 0.2 * torch.ones(ex)
    self.c[:ex] = -65.0 + 15 * self.r[:ex] ** 2
    self.d[:ex] = 8 - 6 * self.r[:ex] ** 2
    self.S[:, :ex] = 0.5 * torch.rand(n, ex)
    self.excitatory[:ex] = 1
    # inhibitory
    self.r[ex:] = torch.rand(inh)
    self.a[ex:] = 0.02 + 0.08 * self.r[ex:]
```

```
            self.b[ex:] = 0.25 - 0.05 * self.r[ex:]
            self.c[ex:] = -65.0 * torch.ones(inh)
            self.d[ex:] = 2 * torch.ones(inh)
            self.S[:, ex:] = -torch.rand(n, inh)
            self.excitatory[ex:] = 0
        self.register_buffer("v", self.rest * torch.ones(n))  # Neuron voltages.
        self.register_buffer("u", self.b * self.v)  # Neuron recovery.
    def forward(self, x: torch.Tensor) -> None:
```

Runs a single simulation step.

```
        :param x: Inputs to the layer.
        """
        # Voltage and recovery reset.
        self.v = torch.where(self.s, self.c, self.v)
        self.u = torch.where(self.s, self.u + self.d, self.u)
        # Add inter-columnar input.
        if self.s.any():
            x += torch.cat(
                [self.S[:, self.s[i]].sum(dim=1)[None] for i in range(self.s.shape[0])],
                dim=0,
            )
        # Apply v and u updates.
        self.v += self.dt * 0.5 * (0.04 * self.v**2 + 5 * self.v + 140 - self.u + x)
        self.v += self.dt * 0.5 * (0.04 * self.v**2 + 5 * self.v + 140 - self.u + x)
        self.u += self.dt * self.a * (self.b * self.v - self.u)
        # Voltage clipping to lower bound.
        if self.lbound is not None:
            self.v.masked_fill_(self.v < self.lbound, self.lbound)
        # Check for spiking neurons.
        self.s = self.v >= self.thresh
        super().forward(x)
def reset_state_variables(self) -> None:
```

Resets relevant state variables.

```
        super().reset_state_variables()
        self.v.fill_(self.rest)  # Neuron voltages.
        self.u = self.b * self.v  # Neuron recovery.
    def set_batch_size(self, batch_size) -> None:
        """
        Sets mini-batch size. Called when layer is added to a network.
        :param batch_size: Mini-batch size.
        """
        super().set_batch_size(batch_size=batch_size)
        self.v = self.rest * torch.ones(batch_size, *self.shape, device=self.v.device)
        self.u = self.b * self.v
```

Add Izhikevich neuron node

class-bindsnet.network.nodes.IzhikevichNodes(n:Optional[int]=None,shape:Optional[Iterable[int]]=None, traces: bool=False, traces_additive: bool = False, tc_trace: Union[float, torch.Tensor] = 20.0, trace_scale: Union[float, torch.Tensor] = 1.0, sum_input: bool = False, excitatory: float = 1, thresh: Union[float, torch.Tensor] = 45.0, rest: Union[float, torch.Tensor] = -65.0, lbound: float = None, **kwargs)

The meanings of the parameters are:

n – The number of neurons in the layer.

shape – The dimensionality of the layer.

traces – Whether to record spike traces.

traces_additive – Whether to record spike traces additively.

tc_trace – Time constant of spike trace decay.

trace_scale – Scaling factor for spike trace.

sum_input – Whether to sum all inputs.

excitatory – Percent of excitatory (vs. inhibitory) neurons in the layer; in range [0, 1].

thresh – Spike threshold voltage.

rest – Resting membrane voltage.

lbound – Lower bound of the voltage.

Conclusion

Although the establishment of the Izhikevich mathematical model by bindsnet is relatively complicated, the training results can clearly explain to us what is the action of each step in the process of training the spiking neural network model, which is also the role of the monitor in the spiking neural network model, so that Because we have a clear observation of each step, it is beneficial to the optimization of the subsequent neural network model and the adjustment of parameters.

References

1. Izhikevich, E. M. Simple model of spiking neurons/ E. M. Izhikevich // IEEE Transactions on Neural Networks, vol. 14, no. 6, pp. 1569-1572, Nov. 2003, doi: 10.1109/TNN.2003.820440.

2. Maass, Wolfgang (1997). Networks of spiking neurons: The third generation of neural network model/ Wolfgang Maass // – Neural Networks. 10 (9). – P. 1659–1671.

3. Panda, P. Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition / P. Panda, K. Roy // 2016 International Joint Conference on Neural Networks (IJCNN). IEEE, 2016. – P. 299-306.

4. Вувуникян, Ю.М. Эволюционные операторы с обобщёнными импульсными и спектральными характеристиками: монография / Ю.М. Вувуникян // Гродно: ГрГУ, 2007. – 224 с.

5. Вувуникян, Ю. М., Последовательное соединение мультиполярных эволюционных операторов с обобщенными импульсными характеристиками / Ю.М. Вувуникян, Ваньли Чэнь // XIII Белорусская математическая конференция: материалы Международной научной конференции, Минск, 22–25 ноября 2021 г. : в 2 ч. / сост. В. В. Лепин ; Национальная академия наук Беларуси, Институт математики, Белорусский государственный университет. – Минск: Беларуская навука, 2021. – Ч. 1. – С. 18–19. ISBN 978-985-08-2807-1.

6. Вувуникян, Ю.М. Операторное моделирование импульсной нейронной сети и прямое произведение реакций системных операторов. / Ю. М. Вувуникян, Ваньли Чэнь // Системы компьютерной математики и их приложения (СКМП-2022): сб. материалов XXIII Междунар. конф., Смоленск, 27-28 мая 2022 года. Вып. 23. – Смоленск : СмолГУ, 2022. – С. 202 – 210.

7. Вувуникян, Ю. М. Методы прямого обучения глубоких импульсных нейронных сетей / Ю.М. Вувуникян, Ваньли Чэнь // Материалы Международного конгресса по информатике: информационные системы и технологии (CSIST'2022). – Минск, 27 – 28 октября 2022 года. – Минск: БГУ, 2022. – С. 112–116.

УДК 664

ИМПОРТОЗАМЕЩЕНИЕ В ПИЩЕВОЙ ПРОМЫШЛЕННОСТИ В УСЛОВИЯХ САНКЦИОННОГО ВОЗДЕЙСТВИЯ НА РФ

Гавловская Г.В.[1], Самсоненко М.Р.[2]

[1] к.э.н., доцент Федеральное государственное автономное образовательное учреждение высшего образования «Российский университет дружбы народов» г. Москва, Российская Федерация

[2] аспирант Федеральное государственное автономное образовательное учреждение высшего образования «Российский университет дружбы народов» г. Москва, Российская Федерация

Аннотация

В статье рассмотрены особенности проведения Россией политики импортозамещения в пищевой промышленности в условиях антироссийских санкций. Определена актуальность политики импортозамещения в пищевой промышленности России. Проведен анализ состояния современного комплекса отечественной пищевой промышленности, в том числе его структура, связь отрасли с другими отраслями национальной экономики, а также выделены ключевые проблемы в отрасли: внешние (нестабильность цен на продовольственное сырье на мировых и национальном рынках, изменение мировых логистических каналов, санкционное давление на российскую экономику) и внутренние (отсталость материально-технической базы, зависимость от импортных компонентов и оборудования, недостаточные инвестиции в отрасли и пр.). Выделены направления реализации политики импортозамещения в пищевой промышленности и проведен их краткий анализ. Сделан вывод об изменениях в пищевой промышленности в результате проводимых государством мер по поддержке предприятий пищевой отрасли. Выявлено, что наиболее отчетливые проблемы зависимости от импортного сырья наблюдаются в мясной и маслодельной отрасли пищевой промышленности. Сделан акцент на важности проведения мероприятий по дальнейшему развитию пищевой отрасли, в том числе: снижение зависимости от импортного сырья, привлечение финансовых средств отечественных и зарубежных инвесторов в отрасль, проведение мероприятий