

# Neural Networks

---

Shyue Ping Ong

Aiiso Yufeng Li Family Department of Chemical and Nano Engineering  
University of California, San Diego

<http://materialsvirtuallab.org>

# Overview

Preliminaries

Neural Networks

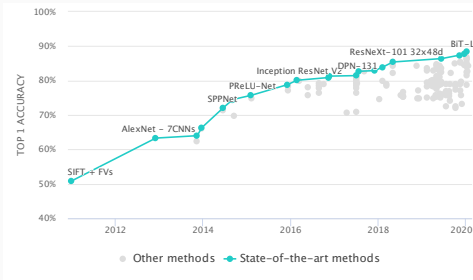
Applications

# Preliminaries

---

# Preliminaries

- Neural networks/deep learning has gotten a lot of hype in recent years.
- In many areas, they have outperformed many traditional ML methodologies.



# Neural Networks

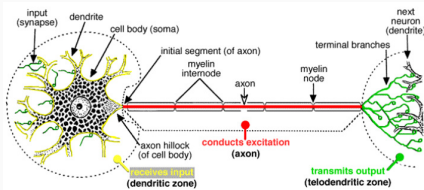
---

# Artificial Neural Network

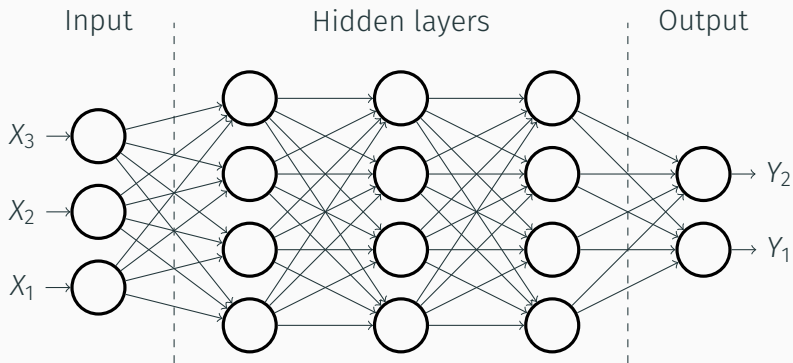
- An artificial neural network (ANN) is a learning algorithm that is (very) loosely based on the structure of the brain.
- Somewhat vaguely, you will also hear the terms “multi-layer perceptron” (MLP) used in place of ANN.

## Universal Approximation Theorem[1]

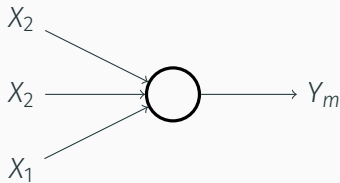
A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions under mild assumptions on the activation function.



# Neural Networks



# Neuron

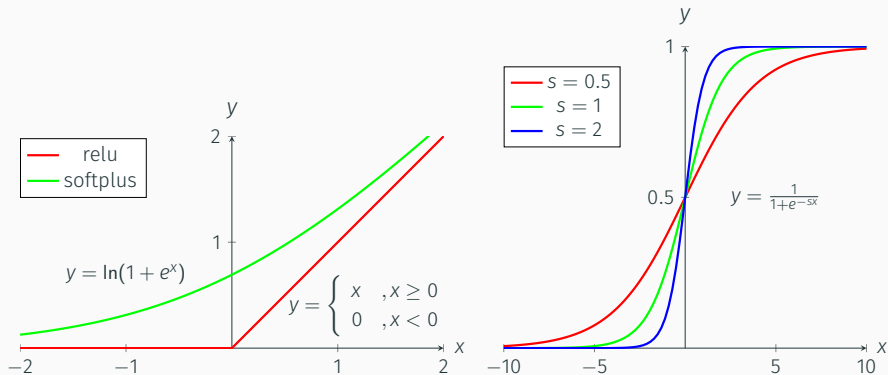


$$Y_m = \sigma(\alpha_{0m} + \alpha_m^T \mathbf{X})$$

- Output of each neuron is a linear function of the inputs.
- In hidden layers, the output is passed through an *activation function*  $\sigma$ .
- Common choices of  $\sigma$  include the rectified linear unit (RELU), sigmoid function, softplus, etc.



# Activation Functions



# Fitting a Neural Network

- Model *weights* ( $\alpha$  in the linear functions) are fitted by *back-propagation*, basically a form of gradient descent.
- Loss functions: squared error for regression, squared error or cross entropy for classification.
- To avoid overfitting, regularization (similar to ridge regression) is typically applied. E.g., *weight decay*:

$$J = \sum \alpha^2$$

# Parameter Decisions for Neural Networks

- Number of hidden units and layers: generally error on the side of having too many hidden units than too few - flexibility is needed to capture non-linearities in the data.
- Extra weights can be shrunk to zero with appropriate regularization.
- Learning rate is a key parameter in fitting ANNs as well as other models. The learning rate controls the rate of gradient descent. A high learning rate reduces training time, but also decreases accuracy. State-of-the-art is to use an adaptive learning rate.
- The error function is non-convex and contains multiple minima, i.e., final solution obtained depends on initial weights. Typical approach is to start with a number of random starting configurations and choose solution with lowest penalized error.
- Alternative is to average predictions over a number of ANN, i.e., ensemble models.

# Software packages for Neural Networks

- Scikit-learn has a basic implementation of a multi-layer perceptron. For this course, we will use this to avoid overloading students with different software packages.
- However, you should be aware that if you are into serious work with ANNs, alternative software are available that offer more features and efficiency. Note that these packages are not limited to ANNs, but ANNs are the most common use case.
  - **Tensorflow**. Open-source package by Google. Very powerful and efficient, but a very steep learning curve. Highly recommend that you use the Keras package (already integrated into TF2) which provides a high level API to it similar to scikit-learn in some ways.
  - **Pytorch**. Open-source package by Facebook. Equally powerful as TF but also steep learning curve.

# ANNs in Scikit-learn and TF2.0

```
from __future__ import annotations
```

```
from sklearn.neural_network import MLPRegressor
```

```
nn = MLPRegressor(hidden_layer_sizes=(5, 3), alpha=1e-4, max_iter=200, learning_rate_init=0.001)  
nn.fit(x, y_reg)
```

```
# Equivalent Tensorflow 2.0
```

```
from tensorflow.keras import Input, Model
```

```
from tensorflow.keras.layers import Dense
```

```
from tensorflow.keras.optimizers import Adam
```

```
x_in = Input(shape=(x.shape[1],))
```

```
x_h1 = Dense(5, activation="relu")(x_in)
```

```
x_h2 = Dense(3, activation="relu")(x_h1)
```

```
x_out = Dense(1)(x_h2)
```

```
model = Model(inputs=x_in, outputs=x_out)
```

```
opt = Adam(learning_rate=0.01)
```

```
model.compile(optimizer=opt, loss="mse")
```

```
model.fit(x, y_reg, epochs=200, batch_size=200)
```

# Beyond the Feedforward Neural Network

- ANNs can be an entire course in itself. Here, we cover some basic variations.

**Convolutional NNs** Uses convolutional operations to achieve translational invariance. Especially important in image processing applications. In modeling crystals with periodic boundary conditions, this can also be a useful feature.

**Recurrent NNs** Connections between nodes form a directed graph along a temporal sequence. Exhibits temporal dynamic behavior and can handle variable length sequences of inputs. Applications: handwriting recognition or speech recognition.

# Applications

---

# Example Application: NN for interatomic potentials

- Atom-centered symmetry functions (ACSF)[2] to represent the atomic local environments.

$$G_i^{\text{atom,rad}} = \sum_{j \neq i}^{N_{\text{atom}}} e^{-\eta(R_{ij}-R_s)^2} \cdot f_c(R_{ij}),$$

$$G_i^{\text{atom,ang}} = 2^{1-\zeta} \sum_{j,k \neq i}^{N_{\text{atom}}} (1 + \lambda \cos \theta_{ijk})^\zeta \cdot e^{-\eta'(R_{ij}^2 + R_{ik}^2 + R_{jk}^2)} \cdot f_c(R_{ij}) \cdot f_c(R_{ik}) \cdot f_c(R_{jk}),$$

where  $R_{ij}$  is the distance between atom  $i$  and neighbor  $j$ ;  $\eta$  is the width of the Gaussian;  $R_s$  is the position shift over all neighboring atoms;  $\zeta$  controls the angular resolution.  $f_c(R_{ij})$  is a cutoff function.

- Fully connected ANNs describe the PES with respect to ACSFs.[3]
- Has been developed for Si,  $\text{TiO}_2$ , water, metal-organic frameworks, ZnO, LiPO<sub>4</sub>, etc.



# Example Application: NNP for Si

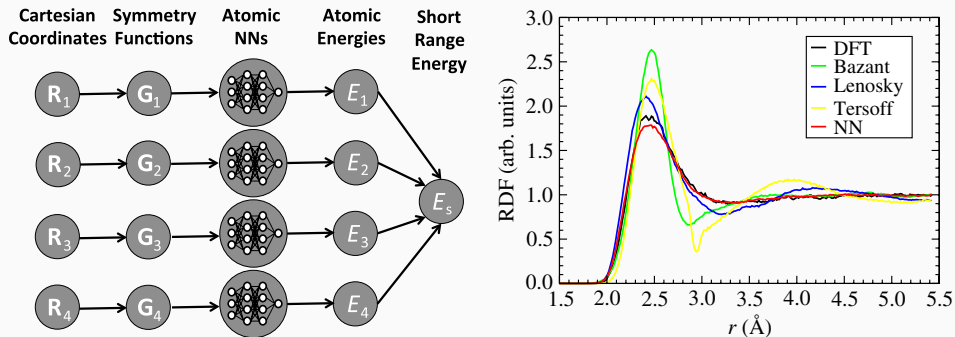


Figure 1: Left: Schematic of NNP architecture. Right: Comparison of radial distribution functions from MD simulations using various interatomic potentials for Si.[4]

# Example Application: NNP-computed phase diagram for amorphous Li-Si

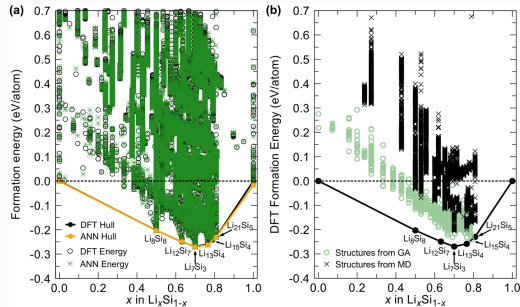
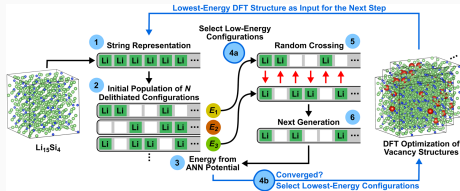
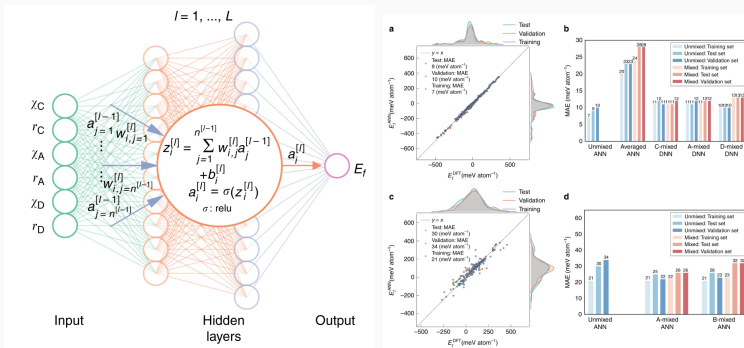


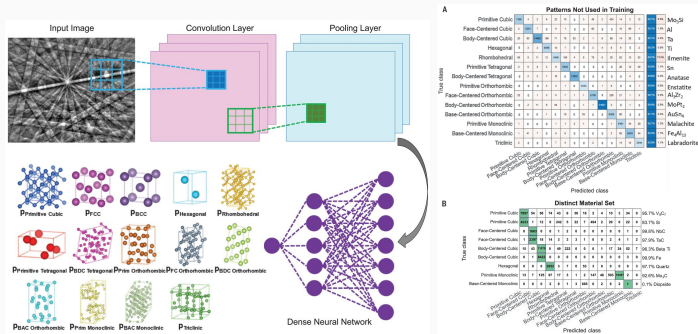
Figure 2: From ref [5]

# Example application: ANNs to predict crystal stability






**Figure 3:** ANNs predict formation energies of garnets and perovskite crystals from Pauling electronegativity and ionic radii.[6]

# Example application: CNNs for determination of crystal symmetry from electron diffraction







**Figure 4:** CNNs to determine crystal symmetry from electron diffraction patterns with > 90% accuracy.[7]

# Bibliography i

-  Balázs Csanád Csáji.  
***Approximation with Artificial Neural Networks.***  
PhD thesis, Eötvös Loránd University, 2001.
-  Jörg Behler.  
**Atom-centered symmetry functions for constructing high-dimensional neural network potentials.**  
*Journal of Chemical Physics*, 134(7), 2011.
-  Jörg Behler.  
**High-Dimensional Neural Network Potentials for Complex Systems.**  
*Angewandte Chemie International Edition*, pages n/a–n/a.

# Bibliography ii

-  Jörg Behler and Michele Parrinello.  
**Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces.**  
*Physical Review Letters*, 98(14):146401, April 2007.
-  Nongnuch Artrith, Alexander Urban, and Gerbrand Ceder.  
**Constructing first-principles phase diagrams of amorphous LixSi using machine-learning-assisted sampling with an evolutionary algorithm.**  
*The Journal of Chemical Physics*, 148(24):241711, March 2018.
-  Weike Ye, Chi Chen, Zhenbin Wang, Iek-Heng Chu, and Shyue Ping Ong.  
**Deep neural networks for accurate predictions of crystal stability.**  
*Nature Communications*, 9(1):3800, December 2018.

 Kevin Kaufmann, Chaoyi Zhu, Alexander S. Rosengarten, Daniel Maryanovsky, Tyler J. Harrington, Eduardo Marin, and Kenneth S. Vecchio.

**Crystal symmetry determination in electron diffraction using machine learning.**

*Science*, 367(6477):564–568, January 2020.

The End