KAIST CHEMISTRY

Neural Networks

Seongok Ryu

Department of Chemistry, KAIST

KAIST

Contents

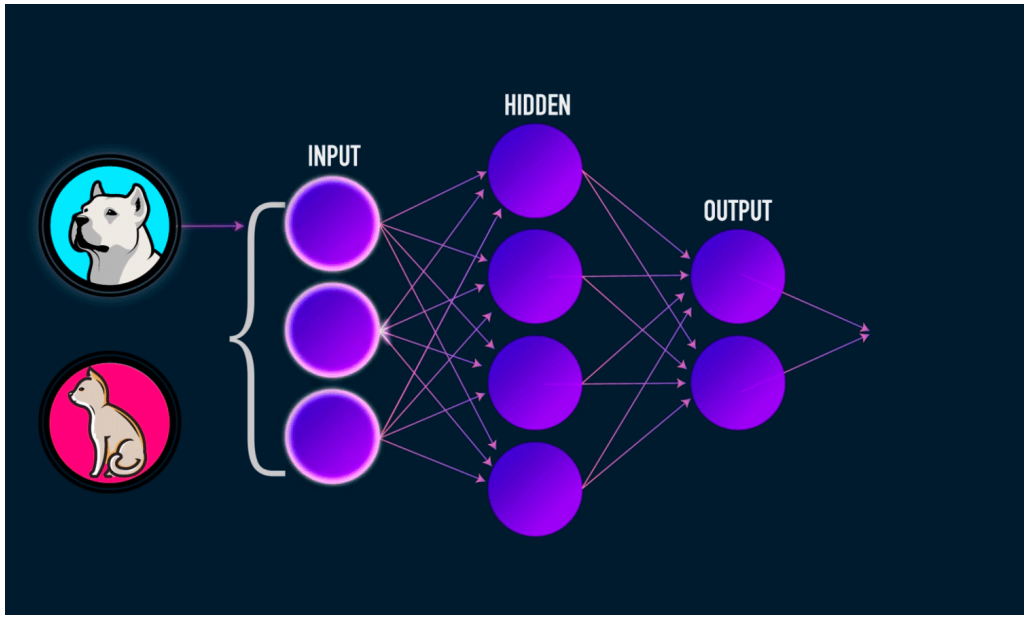
Review

Our aim of learning statistical model from data is to minimize the risk of our hypothesis by tuning model parameters.

$$\hat{h} = \underset{h \in \mathcal{H}}{\operatorname{argmin}} R_{emp}(h) = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L(h(x_i), y_i)$$

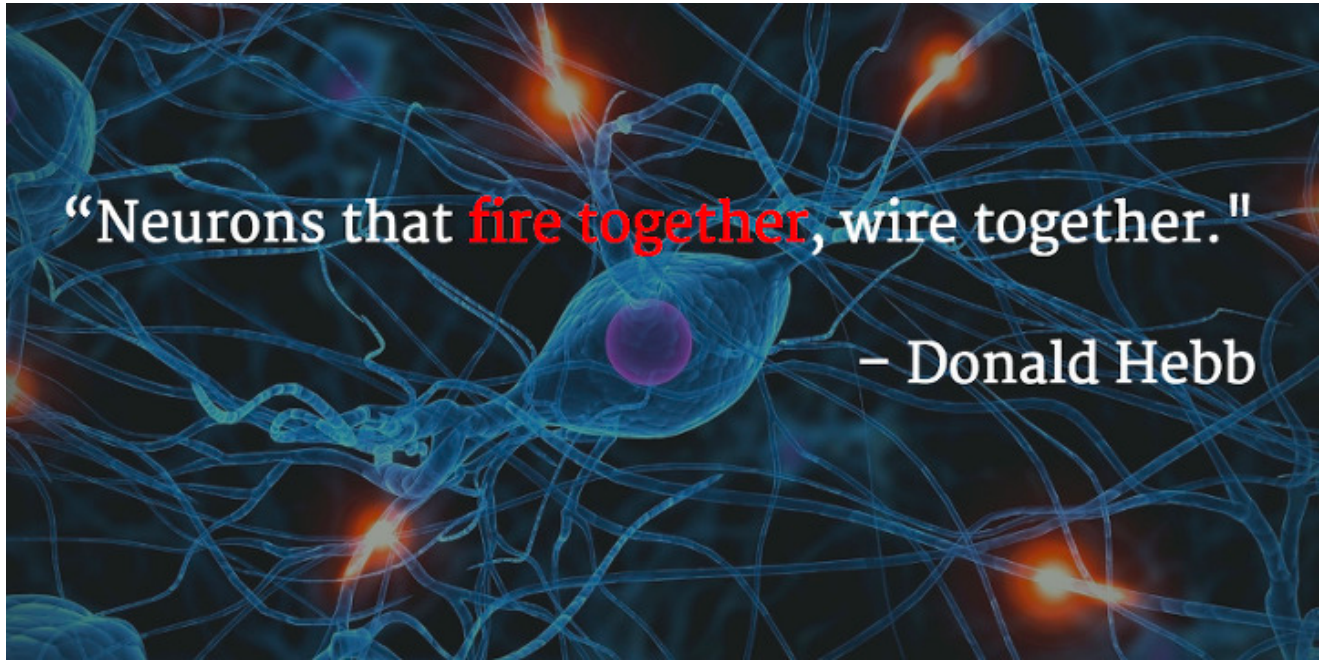
In previous sections, we do not discuss on how we can parameterize our hypothesis function, h.

From now on, we will investigate that *neural networks can parameterize any hypothesis in theoretical*.



Hebbian Principle

- Learning algorithms implemented by *artificial neural networks* is originally inspired from the biological processes in brains.
- Hebbian theory (1940s): a neuro-scientific theory claiming that an increase in synaptic efficacy arises from a presynaptic cell's repeated and persistent stimulation of a postsynaptic cell.
- "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."
- *"Cells that fire together wire together."*



Hebbian Principle

- From the viewpoint of ANN, Hebbian principle can be described as a method of determining how to alter the weights between model neurons:

$$w_{ij} = x_i x_j$$

  where w_{ij} is the weights of the connection from neuron j to neuron i and x_i is the input for neuron i.
- Hebbian principle is often generalized as

$$\Delta w_i = \eta x_i y$$

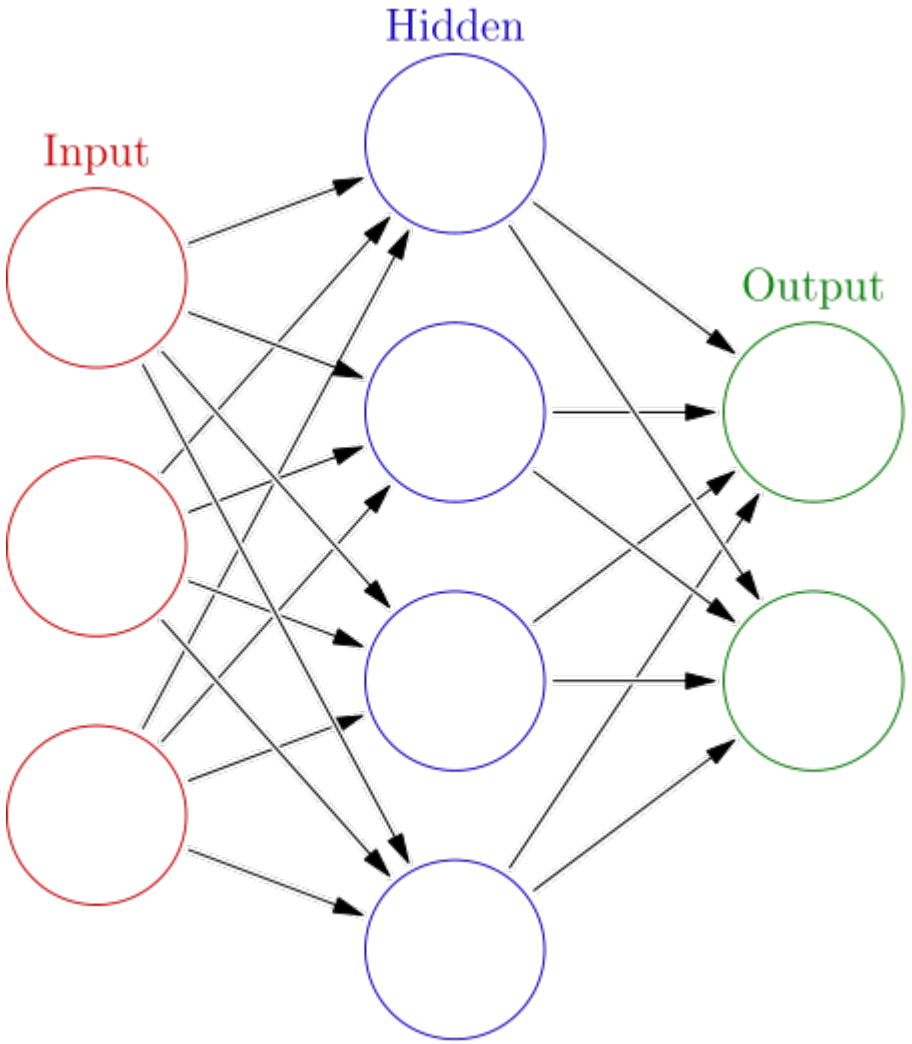
  or the change in the i-th synaptic weight w_i is equal to the learning rate η times the i-th input x_i times the post-synaptic response y.
- The original goal of the ANN approach was to solve the problem in the same way that a humain brain would. However, over time, attention moved to performing specific tasks, leading deviations from biology. In recent days, ANNs have been used on a variety of tasks, including computer vision, speech recognition, machine translation and so far.

Multi Layer Perceptron

- Multi-layer perceptron (MLP) is the basic building block of neural networks.
- The output $\mathbf{y} \in \mathbb{R}^{d_{out}}$ of the MLP for the input $\mathbf{x} \in \mathbb{R}^{\boldsymbol{d_{in}}}$ is given by

$$\mathbf{y} = \mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$$

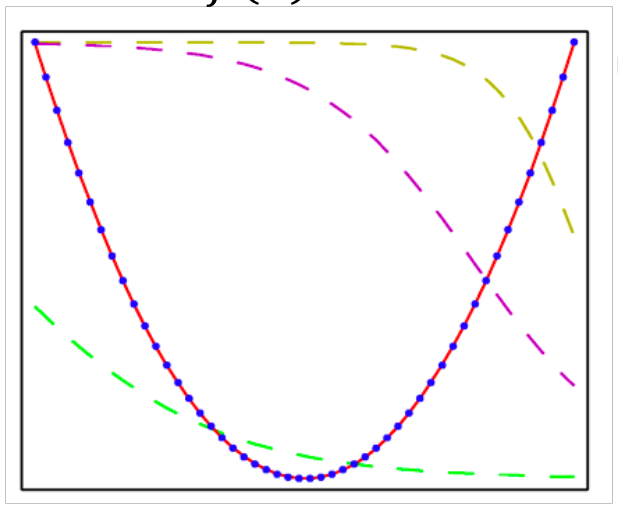
  where $\mathbf{W}_1 \in \mathbb{R}^{d_{in} \times d_h}$ and $\mathbf{W}_2 \in \mathbb{R}^{d_h \times d_{out}}$ are weight parameters, parameters, $\mathbf{b}_1 \in \mathbb{R}^{d_h}$ and $\mathbf{b}_2 \in \mathbb{R}^{d_{out}}$ are bias parameters, and $\sigma(\cdot)$ is an activation function.
- It should be emphasized that we must include the activation function to transform the input to the hidden vectors.



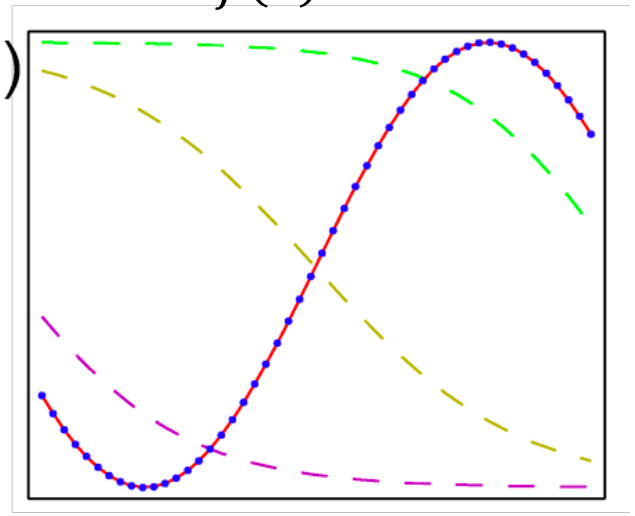
Multi Layer Perceptron

- If the activation function is omitted, overall operation will be a linear transformation. Therefore, it cannot represent any non-linear function.
- Figures in the right exemplify that some target functions approximated by MLPs with one hidden layers.
- We can check that MLPs can approximate functions that are very simple yet non-linear.

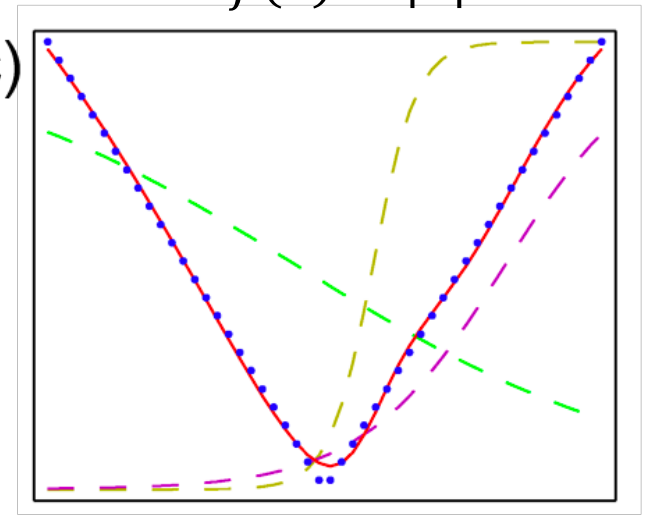
(a) $f(x) = x^2$



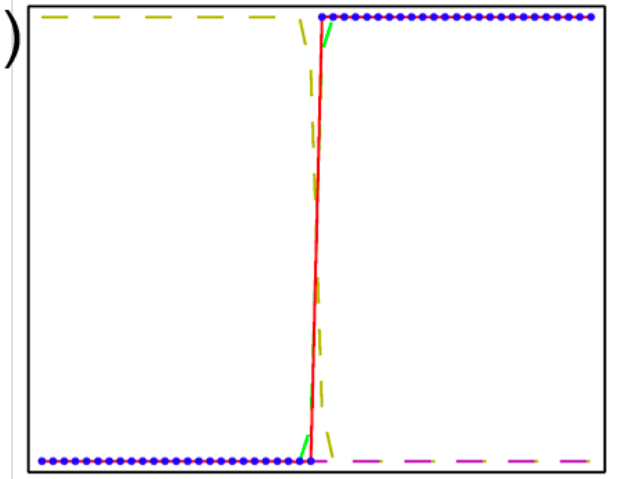
(b) $f(x) = \sin x$



(c) $f(x) = |x|$



(d) $f(x) = H(x)$, step function

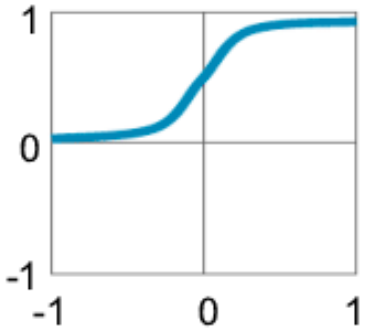


Multi Layer Perceptron

Examples of non-linear activation functions

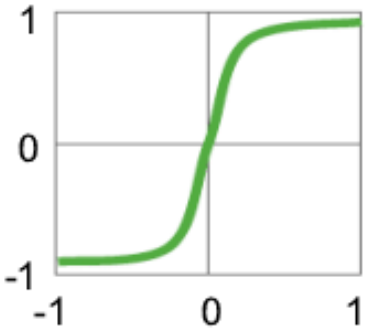
Traditional Non-Linear Activation Functions

Sigmoid



$y=1/(1+e^{-x})$

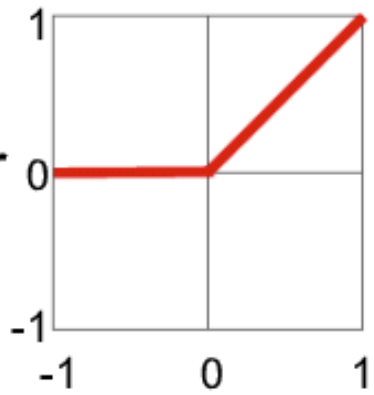
Hyperbolic Tangent



$y=(e^{x}-e^{-x})/(e^{x}+e^{-x})$

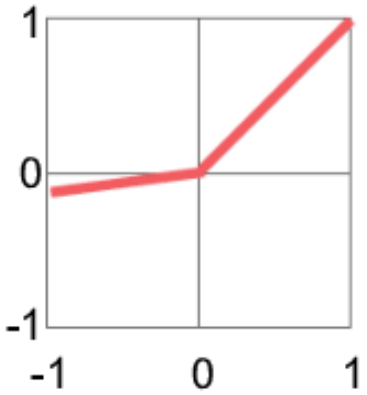
Modern Non-Linear Activation Functions

Rectified Linear Unit (ReLU)



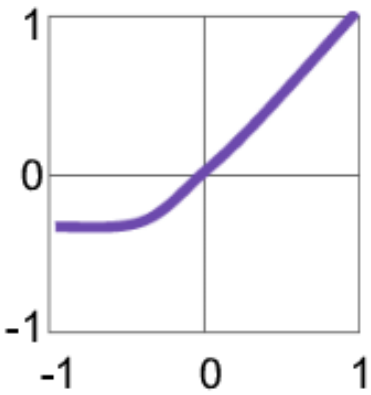
$y=\max(0,x)$

Leaky ReLU



$y=\max(\alpha x,x)$

Exponential LU



$$y=\begin{cases} x, & x\geq 0 \\ \alpha(e^{x}-1), & x<0 \end{cases}$$

α = small const. (e.g. 0.1)

Universal Approximation Theorem

- Why do we use neural networks to model our hypothesis in modern days?
- The answer is, MLPs can approximate any non-linear function and this fact is proven mathematically, so-called the *universal approximation theorem*.

https://en.wikipedia.org/wiki/Universal_approximation_theorem

Theorem 3.3.1 Universal Approximation Theorem. Let $\phi : \mathbb{R} \to \mathbb{R}$ be a non-constant, bounded and continuous function. Let I_m denote the m-dimensional unit hypercube $[0, 1]^m$. The space of real-valued continuous functions on I_m is denoted by $C(I_m)$. Then, given any $\epsilon > 0$ and any function $f \in C(I_m)$, there exist an interger N, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$ for $i = 1, ..., N$, such that we may define:

$$F(x) = \sum_{i=1}^{N} v_i \phi(w_i^T x + b_i) \tag{3.5}$$

as an approximate realization of the function f; that is,

$$|F(x) - f(x)| < \epsilon \tag{3.6}$$

for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$. This still holds when replacing I_m with any compacted subset of $\mathbb{R}^m$.

Universal Approximation Theorem

- Lin et. al. presented another interpretation of neural networks with the perspective of statistical physics.

> **Theorem 3.3.2** Let $\mathbf{f}$ be a neural network of the form $\mathbf{f} = \mathbf{W}_2\sigma\mathbf{W}_1$, where σ acts element-wise by applying some smooth non-linear function σ to each element. Let the input layer, hidden layer and output layer have sizes 2, 4, and 1, respectively. Then $\mathbf{f}$ can approximate a multiplication gate arbitrarily well.

- Let us first Taylor-expand the function σ around the origin:

$$\sigma(u) = \sigma_0 + \sigma_1 u + \sigma_2 \frac{u^2}{2} + \mathcal{O}(u^3)$$

- Without loss of generality, we can assume that $\sigma_2 \neq 0$: since σ is non-linear, it must have a non-zero second derivative at some point, so we can use the biases in $\mathbf{W}_1$ to shift the origin to this point to ensure $\sigma_2 \neq 0$.

Lin, Henry W., Max Tegmark, and David Rolnick. "Why does deep and cheap learning work so well?." *Journal of Statistical Physics* 168.6 (2017): 1223-1247.

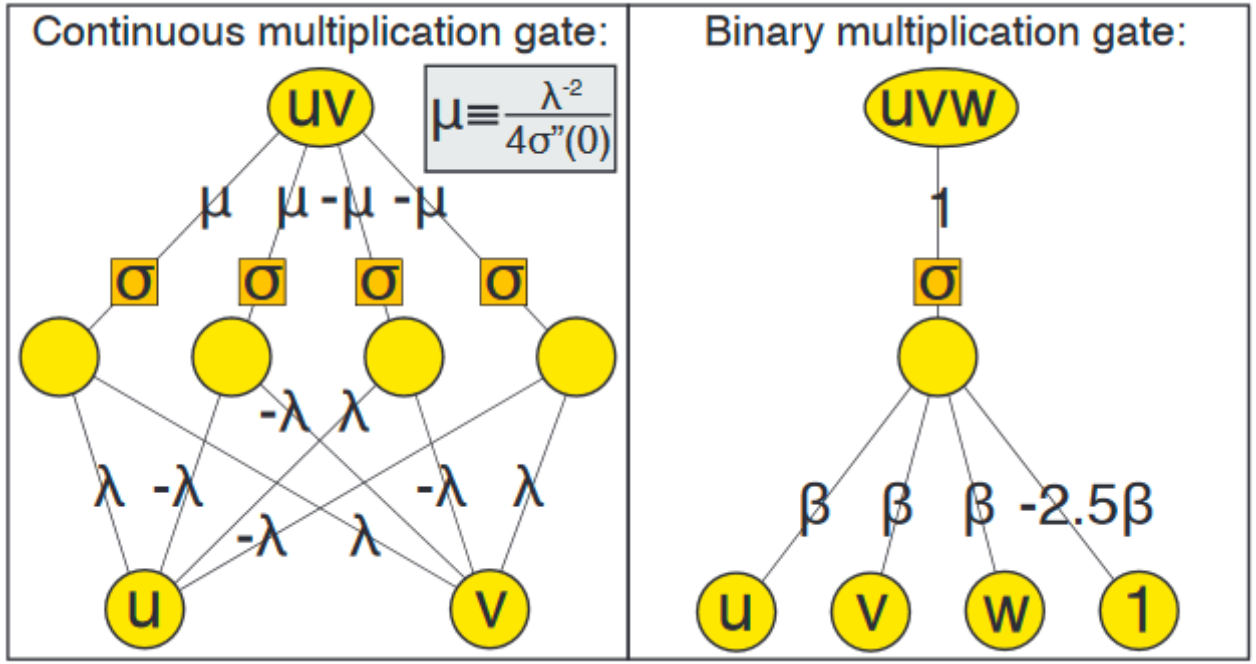
Universal Approximation Theorem

- Then, it now implies that

$$m(u,v) = \frac{\sigma(u+v) + \sigma(-u-v) - \sigma(u-v) - \sigma(-u+v)}{4\sigma_2} = uv[1 + \mathcal{O}(u^2+v^2)]$$

where we will term $m(u,v)$ the *multiplication approximator*.

- Taylor's theorem guarantees that $m(u,v)$ is an arbitrarily good approximation of uv for small $|u|$ and $|v|$.
- However, we can always make $|u|$ and $|v|$ arbitrarily small by scaling $\mathbf{W}_1 \to \lambda\mathbf{W}_1$ and the compensating by scaling $\mathbf{W}_2 \to \lambda^{-2}\mathbf{W}_2$.



Lin, Henry W., Max Tegmark, and David Rolnick. "Why does deep and cheap learning work so well?." *Journal of Statistical Physics* 168.6 (2017): 1223-1247.

Backpropagation and Stochastic Gradient Descent

- Our goal with statistical learning algorithms is to minimize the risk of our hypothesis parameterized by θ, i.e. weights and biases.

$$\hat{\theta} = \underset{\theta \in \Theta}{\operatorname{argmin}} R(h_\theta) = \underset{\theta \in \Theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L(f_\theta(x_i), y_i)$$

- We can cast this learning problem to optimization problems by adopting *stochastic gradient descent optimization*.

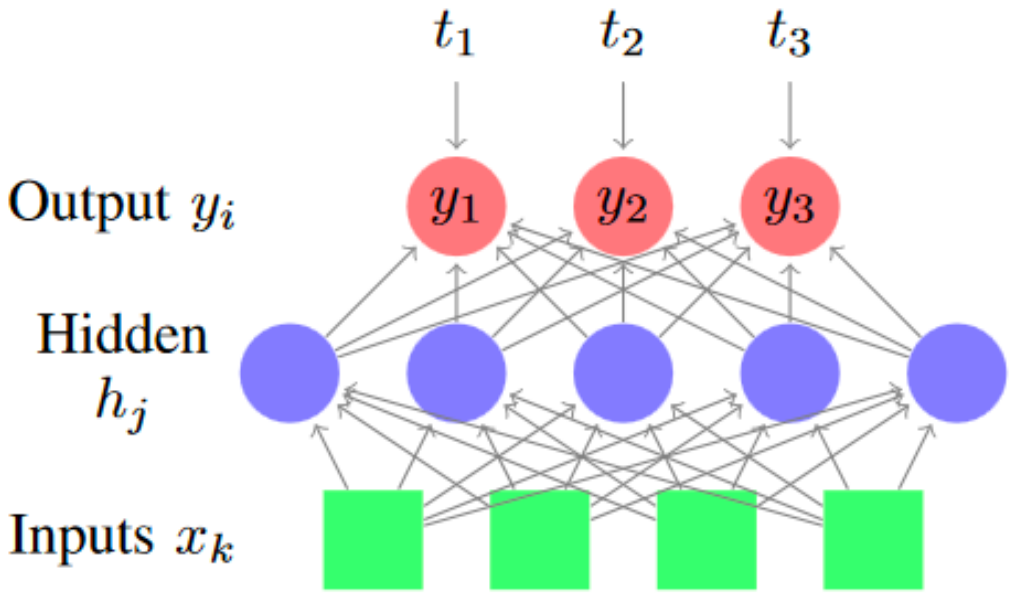
Backpropagation and Stochastic Gradient Descent

- Let us consider a classification model that consists of one-hidden layer.

  Firstly, the minimization objective with cross-entropy losses is given by:

$$L = -\sum_{i=1}^{n} t_i \log y_i + (1 - t_i) \log(1 - y_i)$$

where t_i is the categorical labels and y_i is the output of the model, which is the sigmoid-activated output of logits $s_i = \sum_{j=1} w_{ji} h_j$



Sadowski, Peter. "Notes on backpropagation." *homepage: https://www. ics. uci. edu/pjsadows/notes. pdf (online)* (2016).

Backpropagation and Stochastic Gradient Descent

- In order to minimize the risk, we can compute the derivative of the risk w.r.t. the each weight connecting the hidden units to the output units by using the chain rule:

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial y_i}\frac{\partial y_i}{\partial s_i}\frac{\partial s_i}{\partial w_{ji}}$$

- Examining each factor in turn,

$$\frac{\partial L}{\partial y_i} = \frac{-t_i}{y_i} + \frac{1-t_i}{1-y_i} = \frac{y_i - t_i}{y_i(1-y_i)}$$

$$\frac{\partial y_i}{\partial s_i} = y_i(1-y_i)$$

$$\frac{\partial s_i}{\partial w_{ji}} = h_j$$

- The above equation tells us that the gradients of the risk w.r.t. the weights in the last layer. However, computing the gradients w.r.t. the weights in lower layers requires another application of the chain rule.

Sadowski, Peter. "Notes on backpropagation." *homepage: https://www. ics. uci. edu/pjsadows/notes. pdf (online)* (2016).

Backpropagation and Stochastic Gradient Descent

- A weight w_{kj}^1 connecting to the input unit k to hidden unit j has a gradient

$$\frac{\partial L}{\partial w_{kj}^1} = \frac{\partial L}{\partial s_j^1}\frac{\partial s_j^1}{\partial w_{kj}^1} = \sum_{i=1}^{n}(y_i - t_i)\, w_{ji}\left(h_j(1-h_j)\right)x_k$$

- With

$$\frac{\partial L}{\partial s_j^1} = \sum_{i=1}^{n}\frac{\partial L}{\partial s_i}\frac{\partial s_i}{\partial h_j}\frac{\partial h_j}{\partial s_j^1} = \sum_{i=1}^{n}(y_i - t_i)w_{ji}\left(h_j(1-h_j)\right)$$

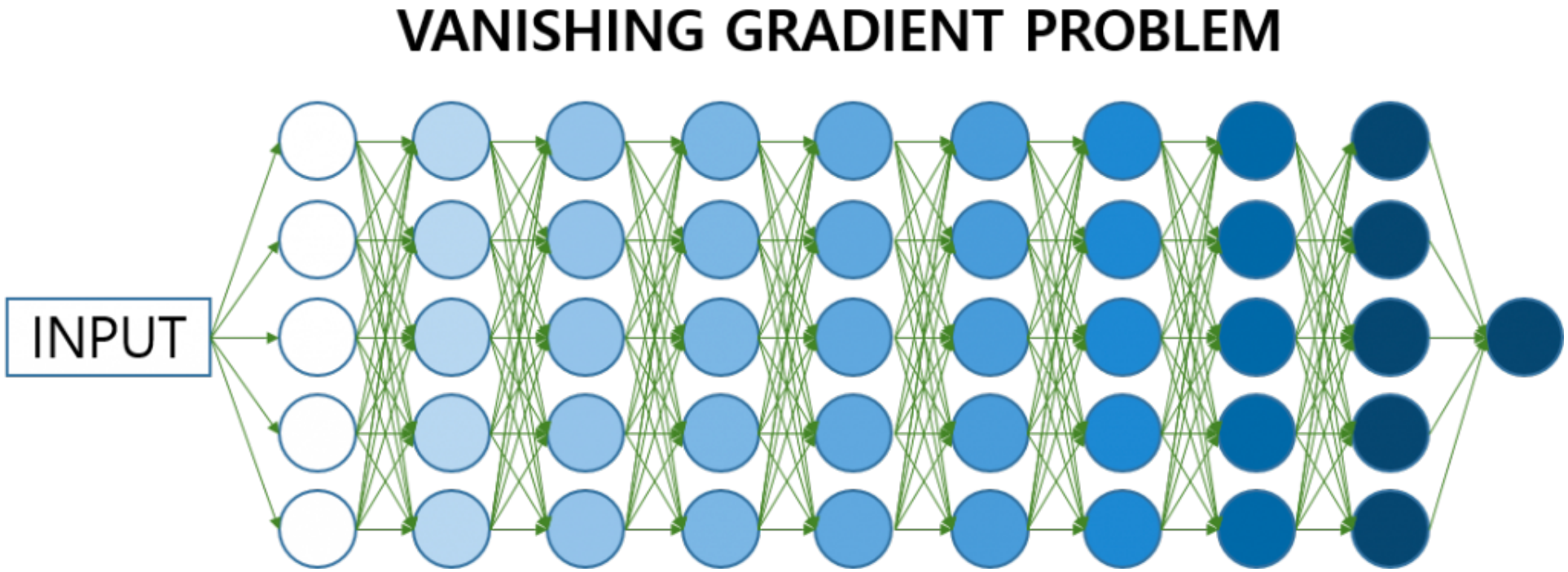
  where j indexes the hidden units, $s_j^1 = \sum_{k=1} w_{kj}^1\, x_k$ is the weighted input summation at hidden unit j, and h_j is the activation at unit j.

- By recursively computing the gradients of the error w.r.t. the activity of each neuron, we can compute the gradients for all weights in a network.

Sadowski, Peter. "Notes on backpropagation." *homepage: https://www. ics. uci. edu/pjsadows/notes. pdf (online)* (2016).

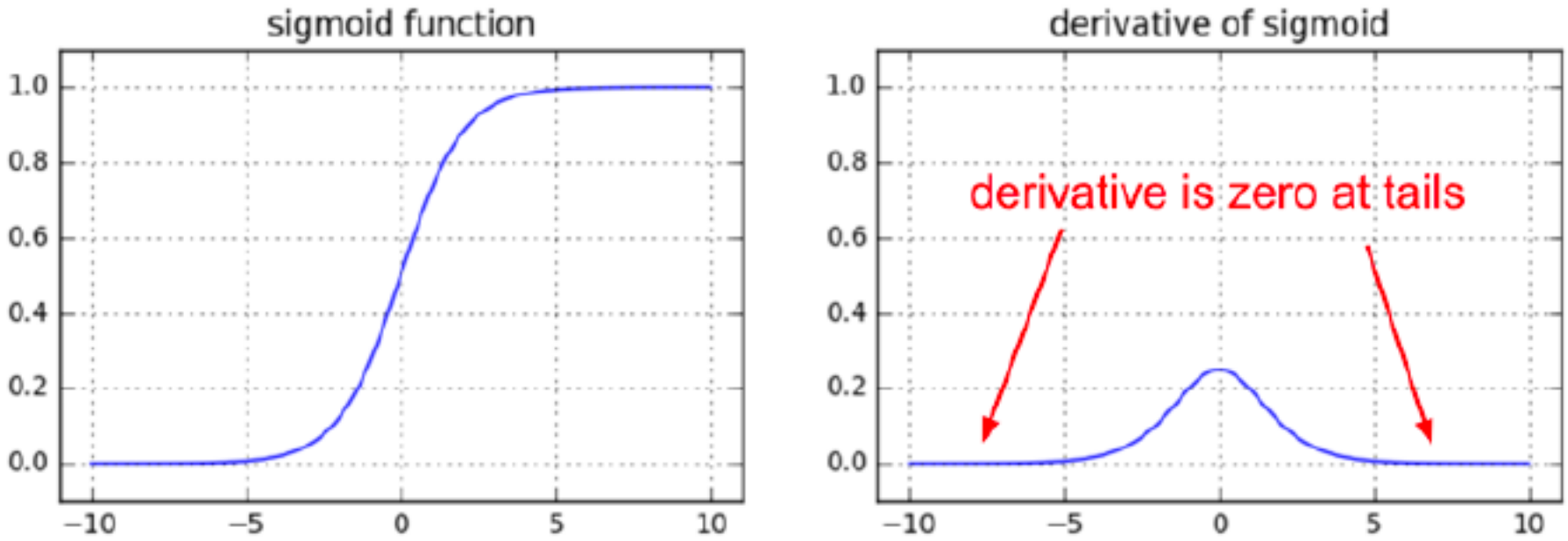
Vanishing Gradient Problems

- One of the obstacles that makes using neural networks be troublesome is *vanishing gradient problems*.
- In the previous slides, we looked how the parameters are optimized by applying a backpropagation algorithm.
- However, terms $\frac{\partial h_j}{\partial s_j}$ are always smaller than zero, when simgoid or tanh non-linearities are adopted.



Vanishing Gradient Problems

- The below figure shows the behaviour of the sigmoid activation and its derivative.
- The value of derivative is maximum ¼ at $x = 0$ and continuosly decreases to zero as x value moves far from the zero.
- This indicates that the total derivative represented by the chain rule becomes small as the network goes deeper.



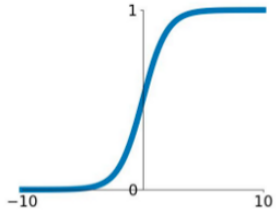
Vanishing Gradient Problems

- To address this vanishing gradient problems, a variety of activation functions have been suggested.
- For example, the Rectified Linear Unit (ReLU) has its derivative value 0 for negative domain and 1 for positive domain, which prevents vanishing gradient problem happened by the product of derivative values smaller than 1.

Activation Functions

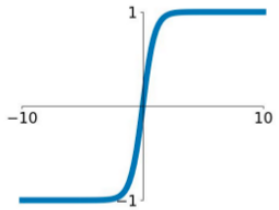
Sigmoid

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



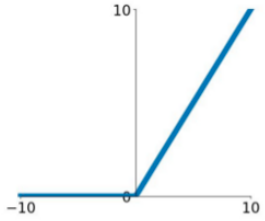
tanh

$$\tanh(x)$$



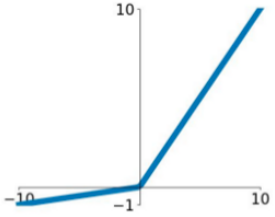
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

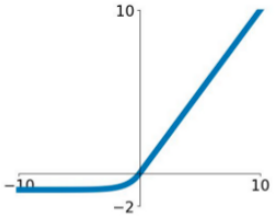


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

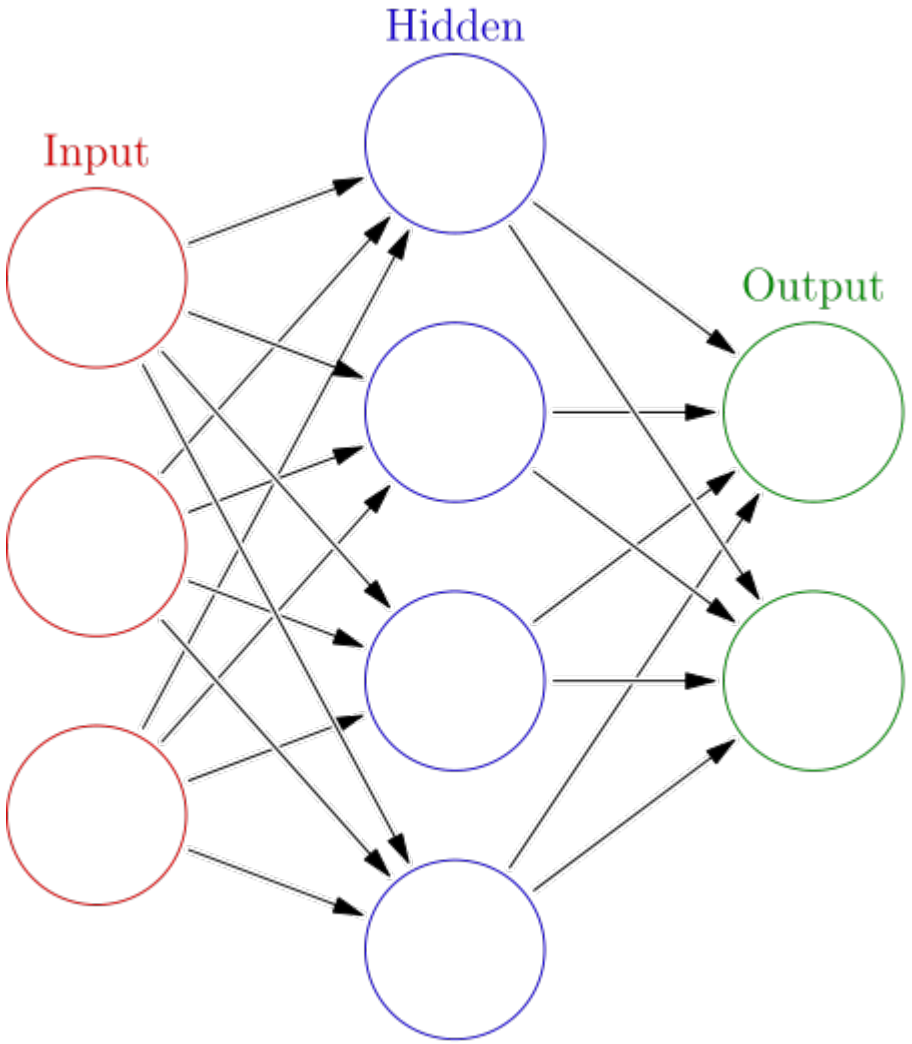


Importance of Inductive Biases

- The output $\mathbf{y} \in \mathbb{R}^{d_{out}}$ of the MLP for the input $\mathbf{x} \in \mathbb{R}^{d_{in}}$ is given by

$$\mathbf{y} = \mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$$

  where $\mathbf{W}_1 \in \mathbb{R}^{d_{in} \times d_h}$ and $\mathbf{W}_2 \in \mathbb{R}^{d_h \times d_{out}}$ are weight parameters, parameters, $\mathbf{b}_1 \in \mathbb{R}^{d_h}$ and $\mathbf{b}_2 \in \mathbb{R}^{d_{out}}$ are bias parameters, and $\sigma(\cdot)$ is an activation function.
- In this case, the number of parameters is $(d_{in} \times d_h + d_h + d_h \times d_{out} + d_{out})$
- Therefore, using MLPs requires too many number of parameters and efficient design of a model architecture is necessitated.



Importance of Inductive Biases

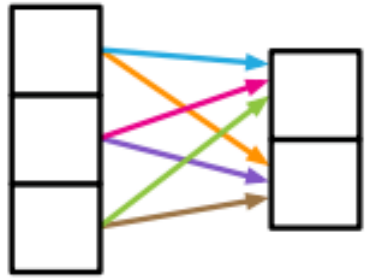
- Efficient design of a model architecture may correspond to the use of operations suitable for a domain of given problem.
- Convolution: pixel values are locally correlated.
- Recurrence: words in a sentence have a sequential relationships.
- Graph convolution: atoms in a molecules are explicitly connected.
- Inductive bias refers to designing a model architecture with a given prioritization.

Component	Entities	Relations	Rel. inductive bias	Invariance
Fully connected	Units	All-to-all	Weak	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation
Graph network	Nodes	Edges	Arbitrary	Node, edge permutations

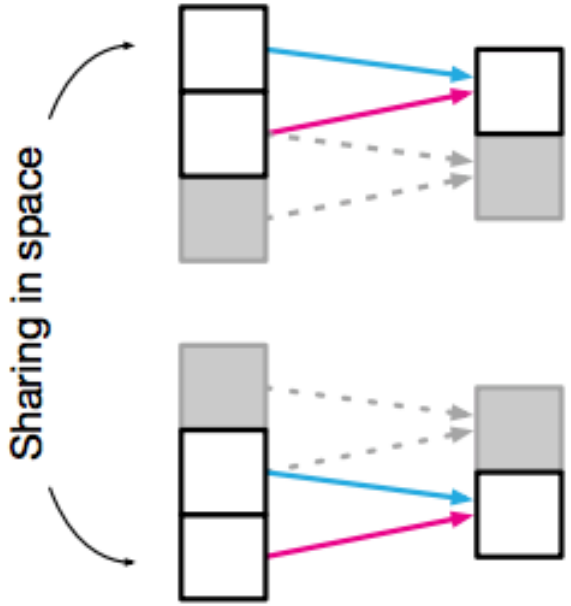
Battaglia, Peter W., et al. "Relational inductive biases, deep learning, and graph networks." *arXiv preprint arXiv:1806.01261*(2018).

Importance of Inductive Biases

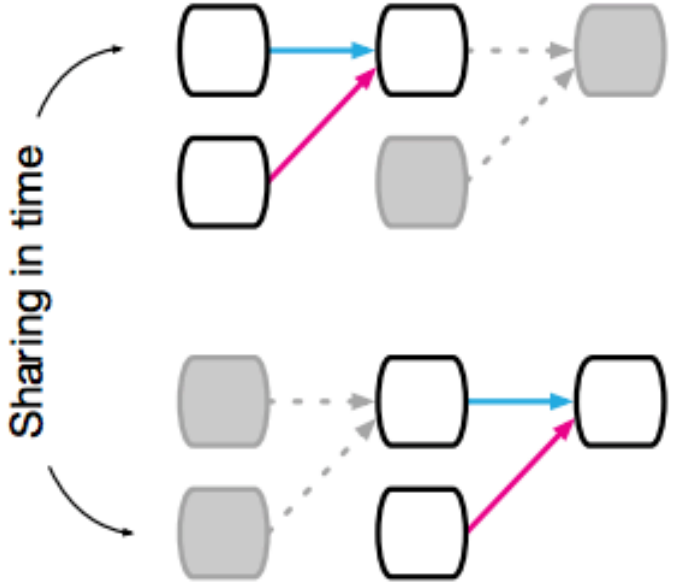
Component	Entities	Relations	Rel. inductive bias	Invariance
Fully connected	Units	All-to-all	Weak	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation
Graph network	Nodes	Edges	Arbitrary	Node, edge permutations



(a) Fully connected



(b) Convolutional



(c) Recurrent

Battaglia, Peter W., et al. "Relational inductive biases, deep learning, and graph networks." *arXiv preprint arXiv:1806.01261*(2018).

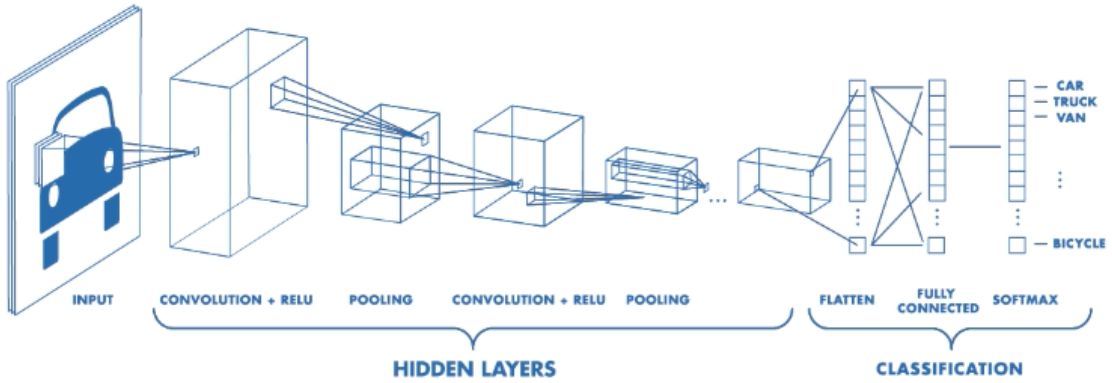
Importance of Inductive Biases

- An inductive bias allows a learning algorithm to prioritize one solution over others, independent of the observed data.
- In Bayesian interpretation, inductive biases are typically expressed through the choice and parameterization of the prior distribution.
- In other contexts, an inductive bias might be a regularization term added to avoid over-fitting, or it might be encoded in the architecture of the algorithm itself.
- Inductive biases can express assumptions about either the data-generating process or the space of solutions.
- This can be interpreted as an assumption about the learning process: that searching for good solutions is easier when there is less ambiguity among solutions.

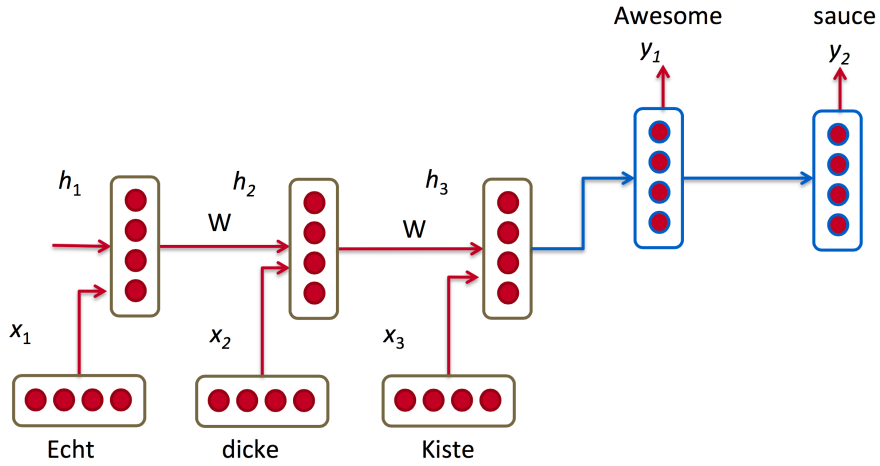
Battaglia, Peter W., et al. "Relational inductive biases, deep learning, and graph networks." *arXiv preprint arXiv:1806.01261*(2018).

Importance of Inductive Biases

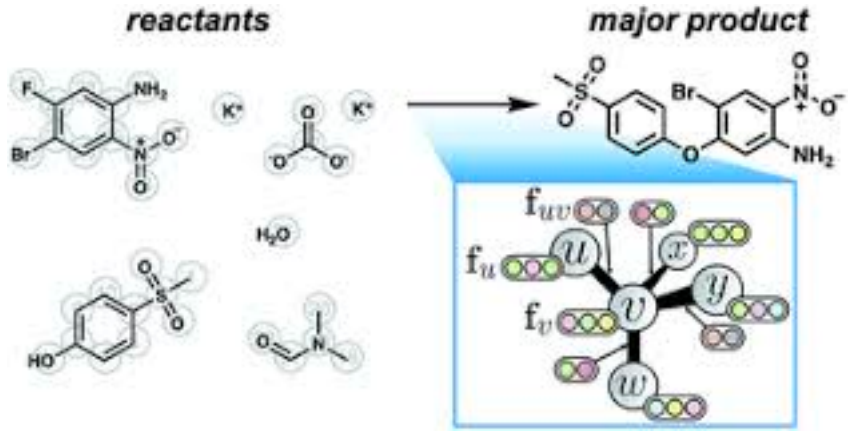
CNNs for computer vision tasks



RNNs for language processing tasks



GNNs for molecular applications



We will investigate in next chapters.