Neural Networks

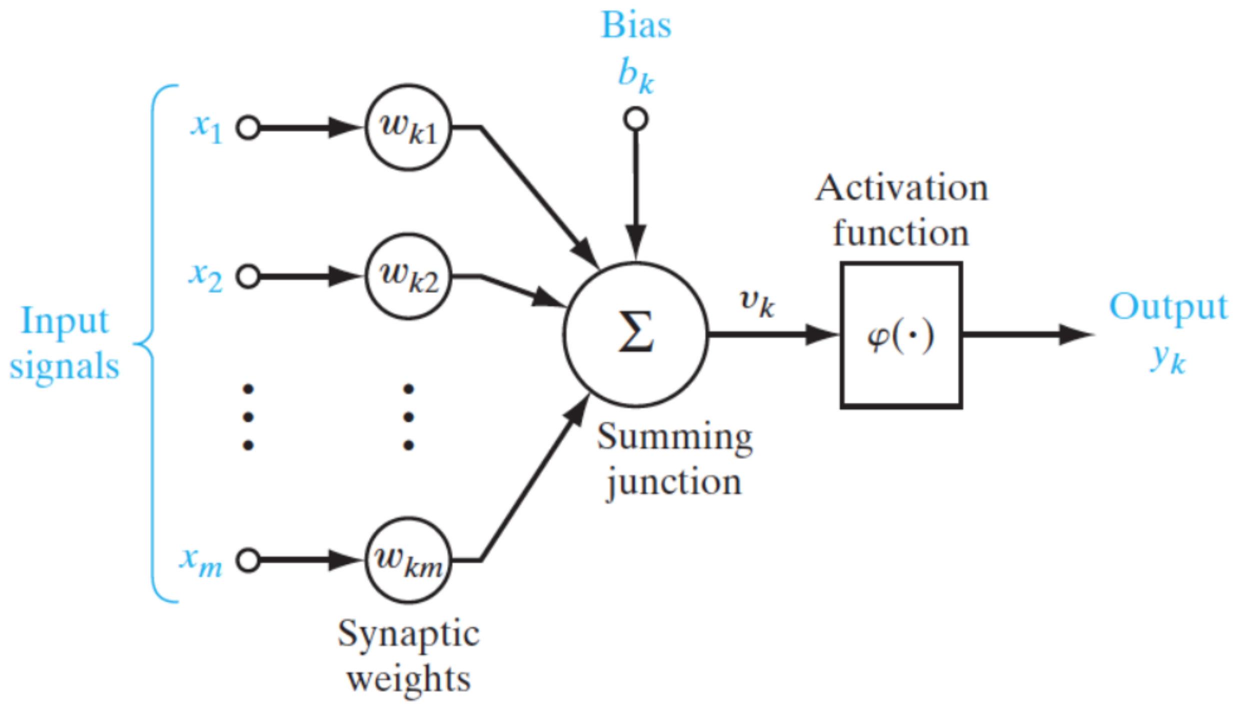
Jingwei Tang @ CGL

01.04.2019 – 05.04.2019

jingwei.tang@inf.ethz.ch

1. **Recap of ANN architectures.**
2. **Solving One Exam Question.**
3. **PyTorch Demos.**

Neurons



Haykin, Simon S., et al. Neural networks and learning machines. Vol. 3. Upper Saddle River: Pearson, 2009.

Linear Units:

$$g: \mathbb{R}^m \to \mathbb{R},$$

$$g(\boldsymbol{x}) = \sum_{j=1}^{m} w_j x_j + b$$

Activation function:

$$\phi: \mathbb{R} \to \mathbb{R}$$

Output:

$$y = \phi(g(\boldsymbol{x}))$$

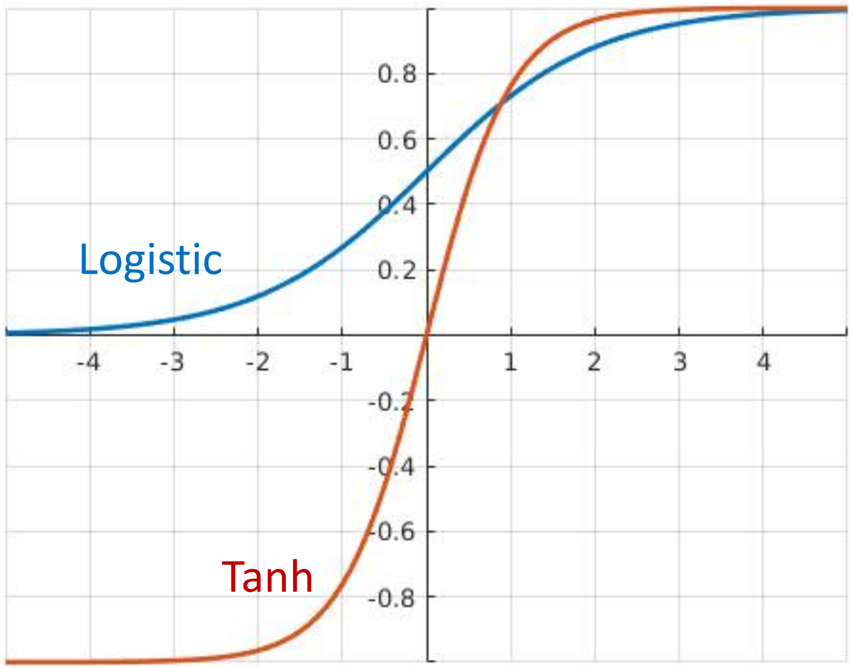
Activation Functions

- **Sigmoid Units:**

Logistic function: $\sigma(z) = \frac{1}{1+e^{-z}} \in (0,1)$

Tanh function: $\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \in (-1,1)$

Vanishing gradients



Activation Functions

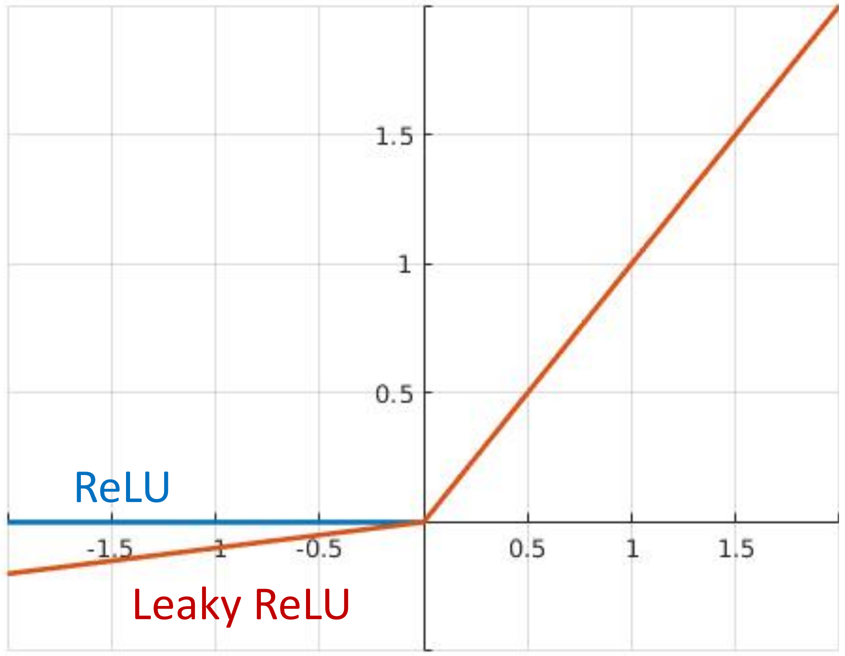
- **Rectified Linear Unit (ReLU):**

$$(z)_+ = \max(0, z)$$

- **Leaky ReLU:**

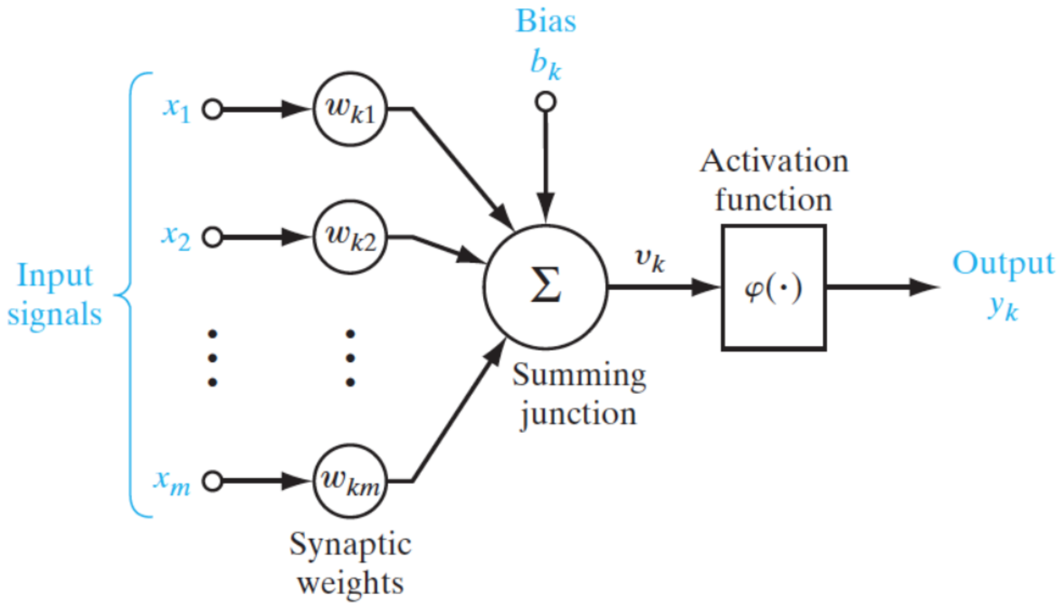
$$(z)_{+'} = \begin{cases} \alpha z, & z < 0 \\ z, & z \geq 0 \end{cases}$$

With α being a small value (0.001).



[Activation function comparisons]

Single Neuron as Binary Classifier



- **Binary Softmax classifier:**

- Use sigmoid activation function.

- Interpret $\sigma(\sum_j w_j x_j + b)$ to be $P(y = 1|x; w)$.

- **Binary SVM classifier:**

- Use an extra max-margin hinge loss to the output.

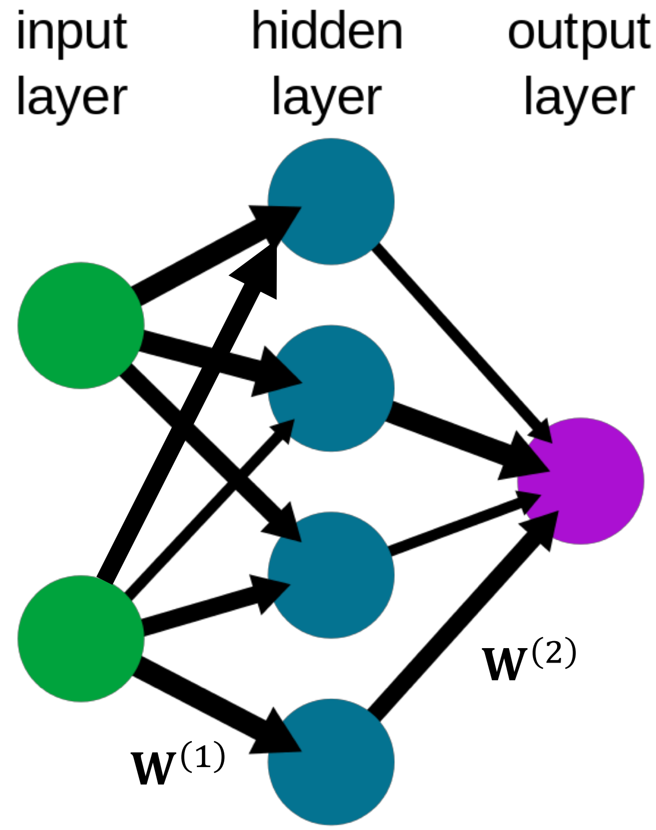
Layers of Neurons (Fully Connected Layers)

- **Single Hidden layer**

$$F(\boldsymbol{x}) = \sum_{i=1}^{k} w_i^{(2)} \phi\left(\sum_{j=1}^{m} w_{ij}^{(1)} x_j\right)$$

Compact representation:

$$F(\boldsymbol{x}) = \mathbf{W}^{(2)} \phi(\mathbf{W}^{(1)} \mathbf{x})$$

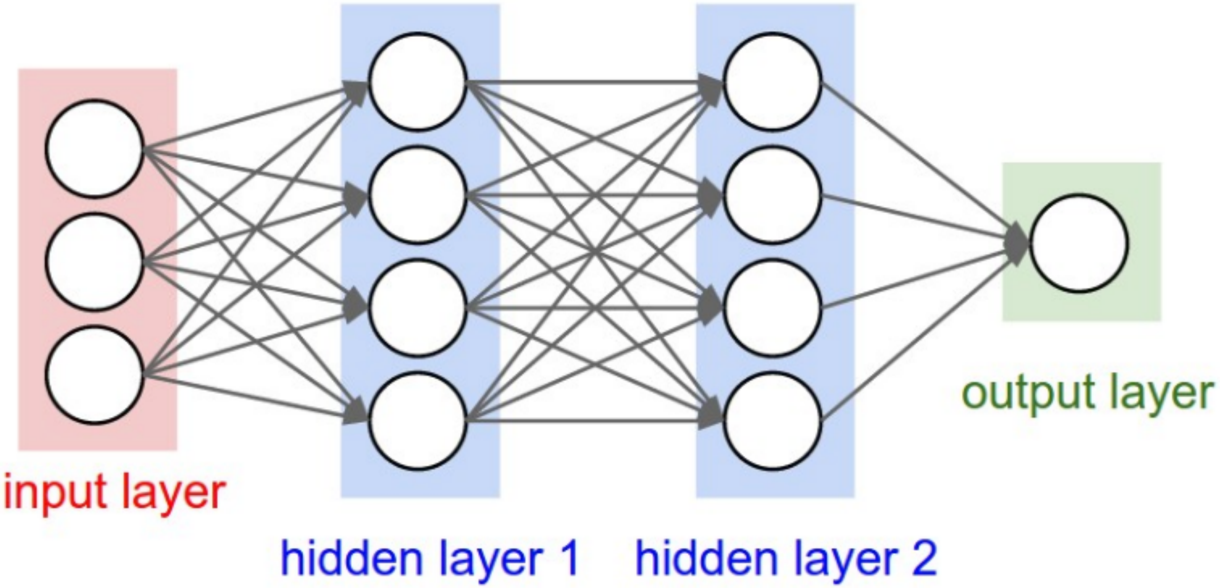


https://en.wikipedia.org/wiki/Neural_network

Layers of Neurons (Fully Connected Layers)

- **Deep Multi-Layer Networks (ANN, MLP):**

$$F(\boldsymbol{x}) = \phi^{(L)}(W^{(L)}\phi^{(L-1)}\left(W^{(L-1)} \ldots \phi^{(1)}\left(W^{(1)}x\right) \ldots\right))$$



Multi-layer Network as Regressor

- **Output:**

  Real-valued output neuron(s), without activation function.
- **Loss function:**

  $\mathcal{L}_1$ or $\mathcal{L}_2$ distances between predicted and ground-truth output.

$$l(\boldsymbol{y}^*, \boldsymbol{y}) = \left|\left|\boldsymbol{y}^* - \boldsymbol{y}\right|\right|_1$$

Multi-layer Network as Classifier

Binary:

- **Output:** Single output neuron y, use **sigmoid** activation function as probability of class membership.

$$\sigma = \frac{1}{1 + e^{-y}}$$

- **Loss:** Use (**binary**) **cross-entropy loss.** ($y^* \in \{0, 1\}$)

$$l(y^*, y) = -y^* \log(\sigma) - (1 - y^*) \log(1 - \sigma)$$

Multi-layer Network as Classifier

Multi-class (C classes):

- **Output:** Multiple output neurons $\boldsymbol{y}$, use **SOFTMAX** function. Interpret output values as probability for corresponding class.

$$\sigma_i = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

- **Loss:** Use **cross-entropy loss.** ($\boldsymbol{y}^*$ is encoded as one-hot vector with one positive class (1) and $C - 1$ negative classes (0).)

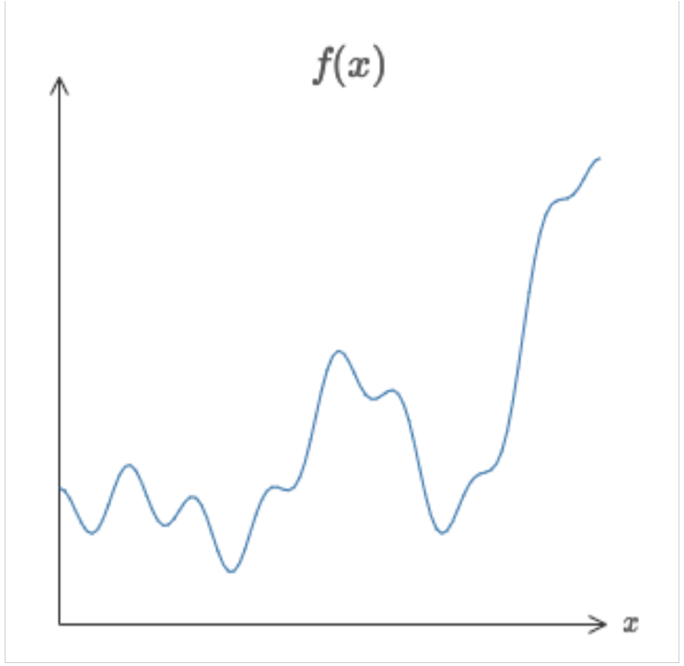
$$l(\boldsymbol{y}^*, \boldsymbol{y}) = -\sum_i y_i^* \log(\sigma_i)$$

Cross Entropy Loss Tutorial

Universal Approximation Theorem

- Given **ANY** continuous function $f(x)$ and some $\epsilon > 0$, there exists a Neural Network $g(x)$ with one hidden layer (with a reasonable choice of **non-linearity**, e.g. sigmoid) such that

$$\forall x, \;\; |f(x) - g(x)| < \epsilon.$$



Complete proof in [Approximation by Superpositions of Sigmoidal Functions (1989)].
[Intuitive explanation] from Michael Nielson.

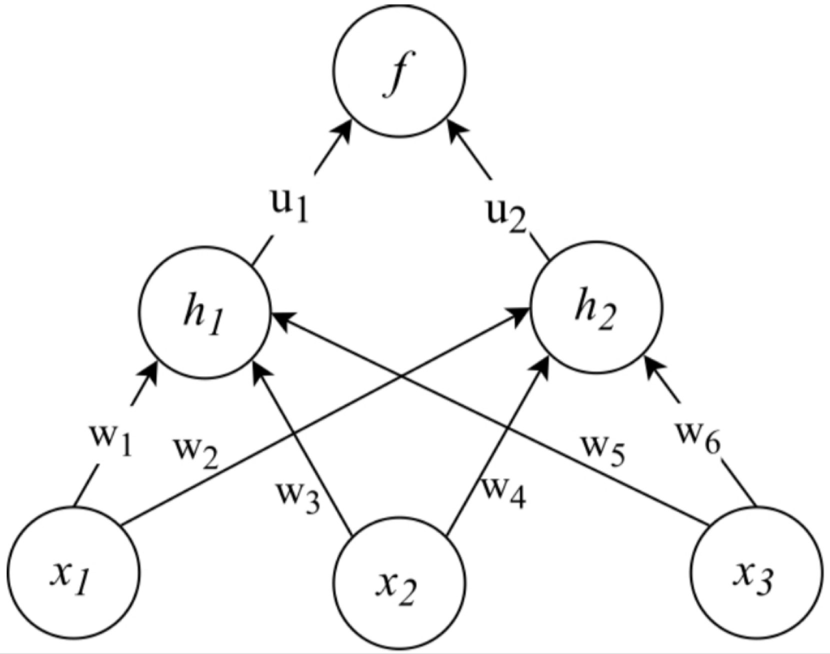
1. Recap of ANN architectures.

2. **Solving One Exam Question.**

3. PyTorch Demos.

Exam 2016
Question 5

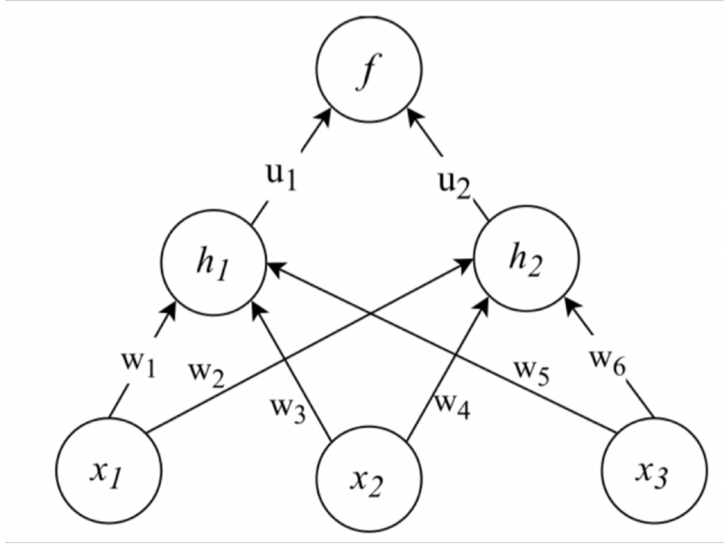
Consider the following neural network with two logistic hidden units h_1, h_2, and three inputs x_1, x_2, x_3. The output neuron f is a linear unit, and we are using the squared error cost function $E = (y - f)^2$. The logistic function is defined as $\rho(x) = 1/\left(1 + e^{-x}\right)$.



(i) Consider a single training example $\boldsymbol{x} = [x_1, x_2, x_3]$ with target output (label) y. Write down the sequence of calculations required to compute the squared error cost (called forward propagation).

(ii) A way to reduce the number of parameters to avoid overfitting is to tie certain weights together, so that they share a parameter. Suppose we decide to tie the weights w_1 and w_4, so that $w_1 = w_4 = w_{\text{tied}}$. What is the derivative of the error E with respect to w_{tied}, i.e. $\nabla_{w_{\text{tied}}} E$?

(i)



$$h_1 = \phi(w_1 x_1 + w_3 x_2 + w_5 x_3)$$
$$h_2 = \phi(w_2 x_1 + w_4 x_2 + w_6 x_3)$$
$$f = u_1 h_1 + u_2 h_2$$
$$E = (y - f)^2$$

(ii)

$$\frac{\partial E}{\partial w_{tied}} = \frac{\partial E}{\partial f}\left(\frac{\partial f}{\partial h_1}\frac{\partial h_1}{\partial w_{tied}} + \frac{\partial f}{\partial h_2}\frac{\partial h_2}{\partial w_{tied}}\right)$$

$$\frac{\partial E}{\partial w_{tied}} = -2(y - f)(u_1 x_1 \phi'(w_1 x_1 + w_3 x_2 + w_5 x_3)$$
$$+ u_2 x_2 \phi'(w_2 x_1 + w_4 x_2 + w_6 x_3))$$

$$\frac{\partial E}{\partial w_{tied}} = -2(y - f)(u_1 x_1 h_1(1 - h_1) + u_2 x_2 h_2(1 - h_2))$$

(iii) For a data set $D = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \cdots, (\boldsymbol{x}^{(n)}, y^{(n)})\}$ consisting of n labeled examples, augment the pseudocode of the stochastic gradient descent algorithm below with learning rate η_t for optimizing the weight w_{tied} (assume all the other parameters are fixed).

begin

$w_{\text{tied}} \leftarrow 0, \eta_t = 1/t$;

for $t = 1$ *to* T **do**

// Fill in code to implement SGD

Select $(x^{(i)}, y^{(i)})$ from D uniformly at random.

Forward pass like in (i) to compute h_1, h_2, f, E.

Backward pass like in (ii) to compute $\frac{\partial E}{\partial w_{tied}}$.

Update parameter $w_{tied} = w_{tied} - \eta_t \frac{\partial E}{\partial w_{tied}}$.

end

end

1. Recap of ANN architectures.

2. Solving One Exam Question.

3. **PyTorch Demos.**

References

- [Standford CS231n]
- [Coursera Deep Learning Specialization]