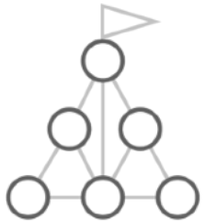


Deep Learning - Fully Connected Layer, Activation Function, and Loss Function

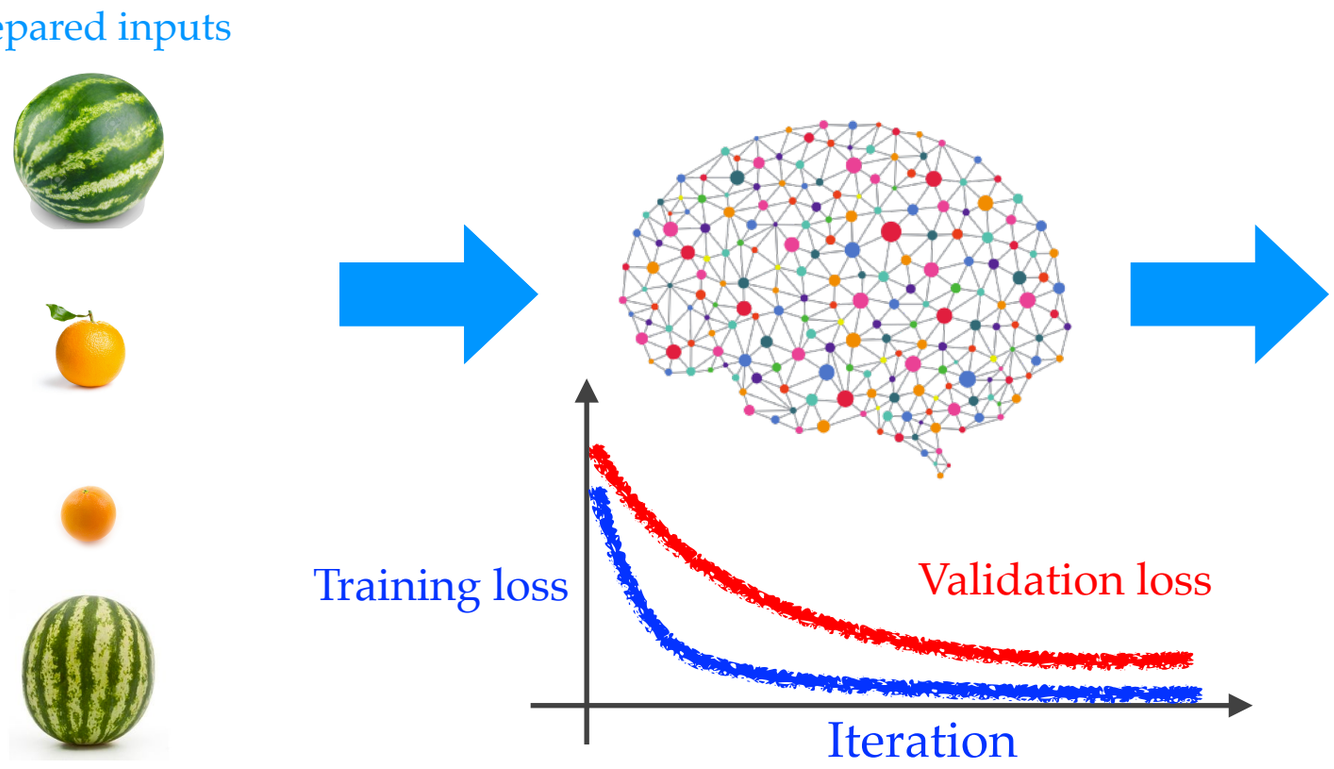
Itthi Chatnuntawech

Outline

- Deep Learning Components
  - Fully connected layer
  - Computation graph
  - Activation functions
  - Loss function
  - Model optimization
    - Gradient descent
    - Backpropagation
    - Stochastic gradient descent (SGD)
  - Regularization
    - ℓ_2 and ℓ_1 regularization
    - Dropout
    - Batch normalization
  - Convolutional layer
  - Pooling layer
- Convolutional Neural Network (CNN)

From Last Time: Image Classification Using Deep Learning

- Training phase: Optimize the weights of a deep neural network



- Test phase: Perform prediction using the trained neural network

Fully connected layer
(Dense)

Convolutional layer

Conv1D, 2D, 3D, …
separable Conv

Optimizer

SGD
Adam
RMSprop

Deep Learning

Pooling layer

max-pooling
average-pooling

Evaluation metric

accuracy
F1-score
AUC
confusion matrix

Activation function

sigmoid
softmax
ESP (swish)
ReLU

Loss function

categorical crossentropy
binary crossentropy
mean squared error
mean absolute error

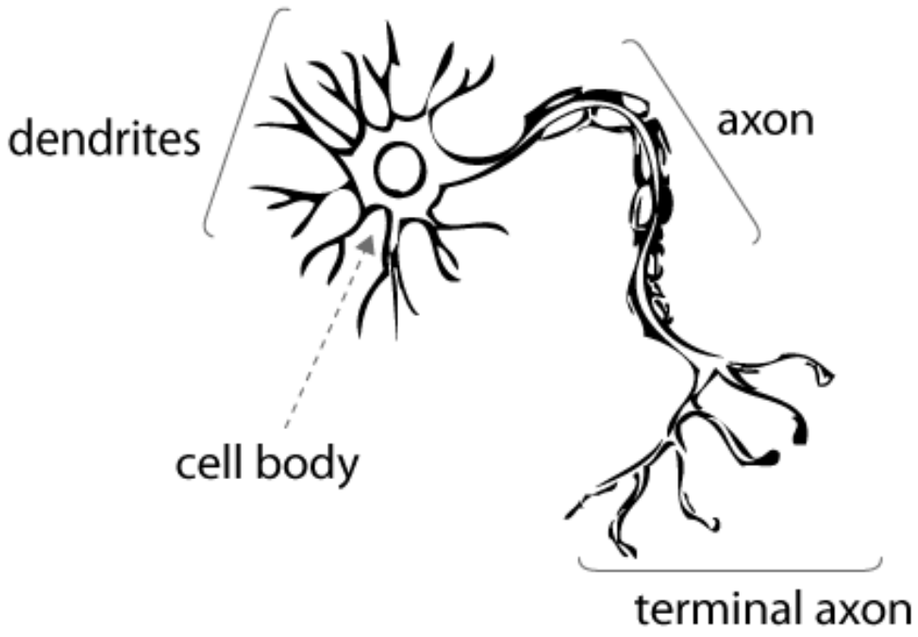
Regularization

Dropout
Data augmentation
l_1, l_2 regularizations

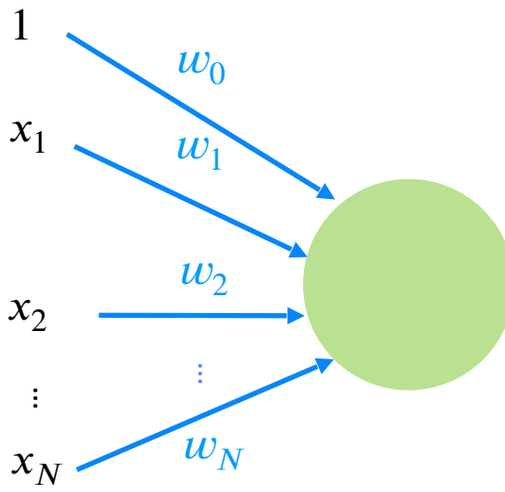
- Combine basic components to build a neural network
  - More components → "More" representative power

Artificial Neuron

Single neuron

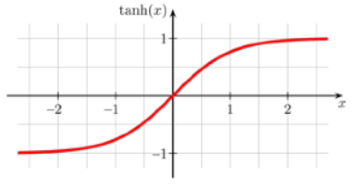


Artificial neuron



$$y = a\left(\sum_{n=1}^{N} x_n w_n + w_0\right)$$

non-linear function (activation)



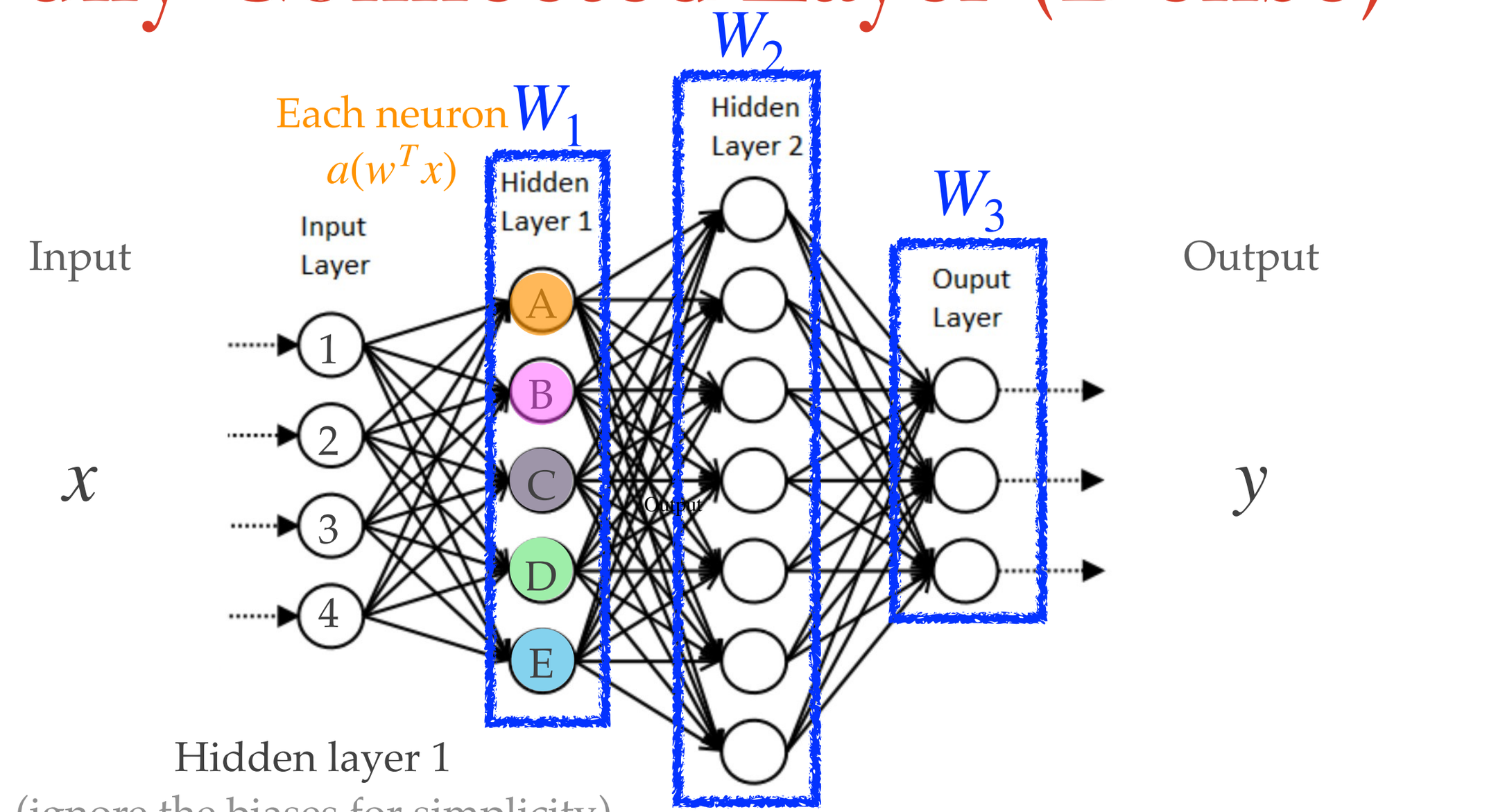
If we set all the weights to 0, then $y = 0$.
If we set $w_1 = 1$ and the rest to 0, then $y = a(x_1)$.
If we set $w_0 = 0$ and the rest to 1, then $y = a(x_1 + x_2 + \ldots + x_N)$.

Different weights give rise to different behavior

The Essense of Artificial Neural Networks by Ivan Liljeqvist (Medium)

keyword: Multi-Layer Perceptron (MLP)

Fully Connected Layer (Dense)



Input x

Output y

Hidden layer 1

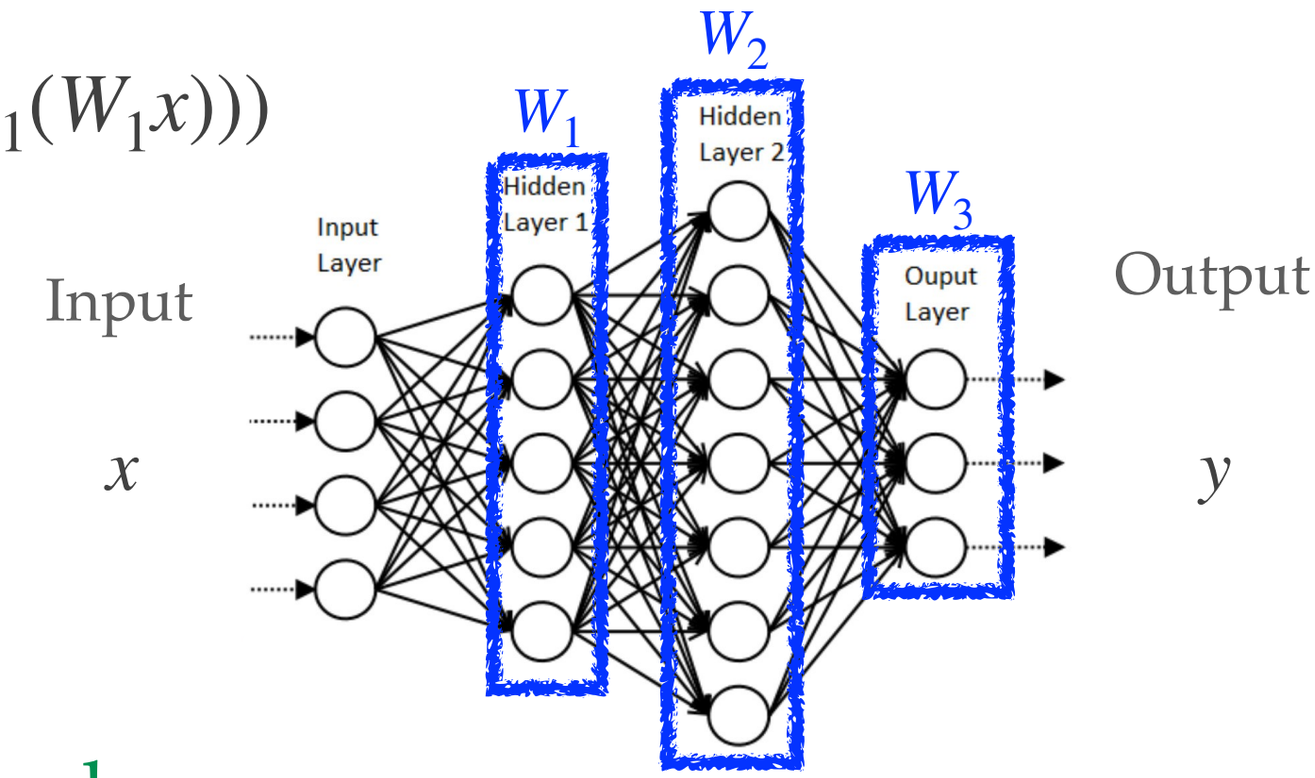
(ignore the biases for simplicity)

$$\begin{bmatrix} out_A \\ out_B \\ out_C \\ out_D \\ out_E \end{bmatrix} = a_1(W_1 x) = a_1\left(\begin{bmatrix} w_{A,1} & w_{A,2} & w_{A,3} & w_{A,4} \\ w_{B,1} & w_{B,2} & w_{B,3} & w_{B,4} \\ w_{C,1} & w_{C,2} & w_{C,3} & w_{C,4} \\ w_{D,1} & w_{D,2} & w_{D,3} & w_{D,4} \\ w_{E,1} & w_{E,2} & w_{E,3} & w_{E,4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \right)$$

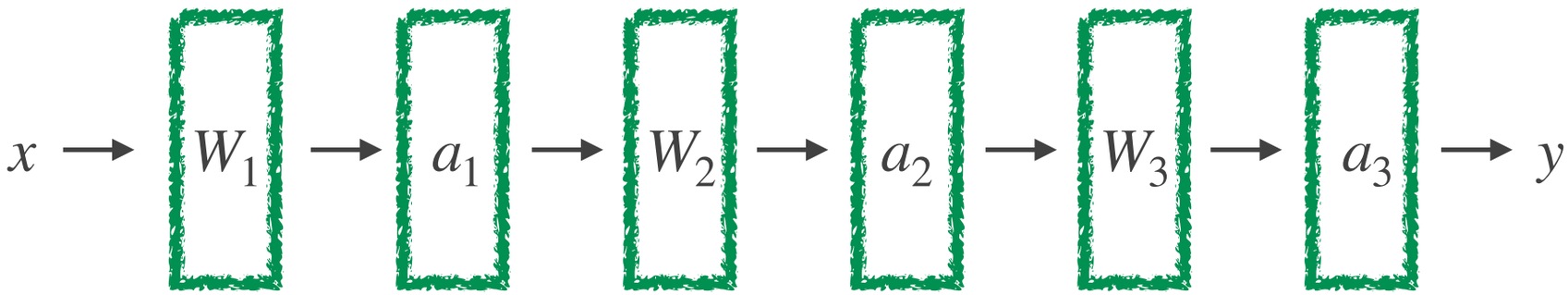
$$y = a_3(W_3 a_2(W_2 a_1(W_1 x)))$$

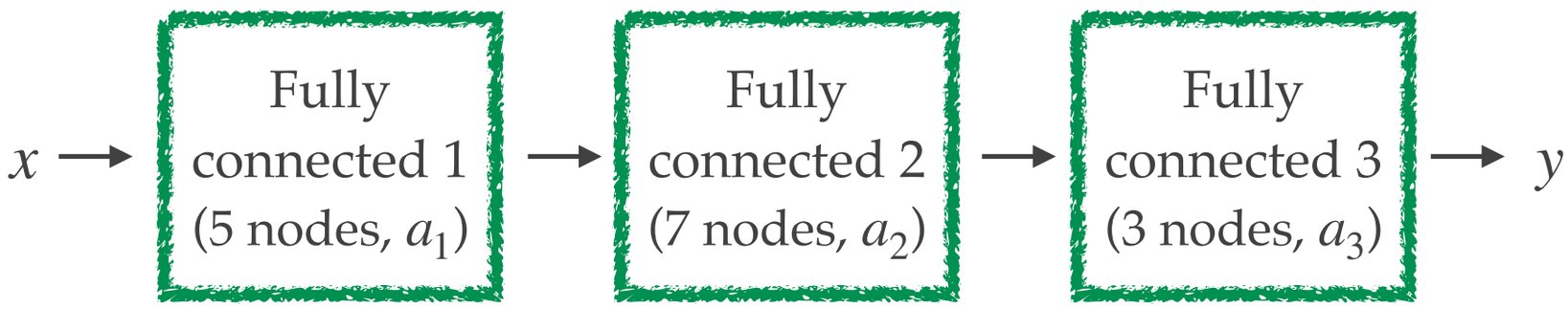
Representing Neural Network

$$y = a_3(W_3 a_2(W_2 a_1(W_1 x)))$$



Computation graph

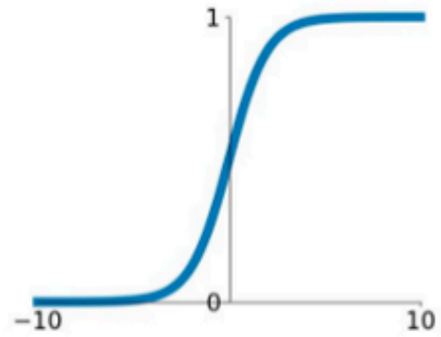




Activation Functions

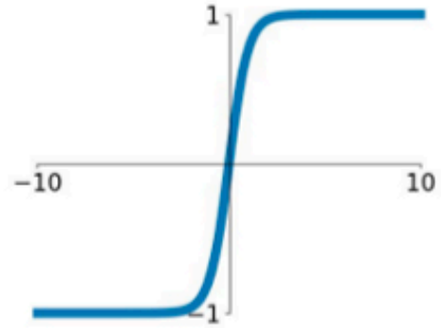
Sigmoid

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



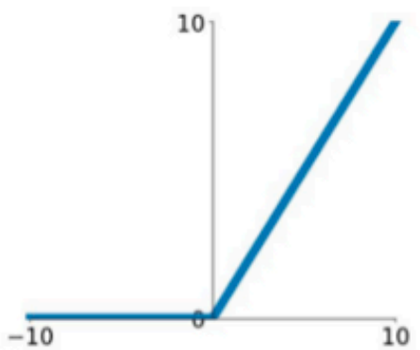
tanh

$$\tanh(x)$$



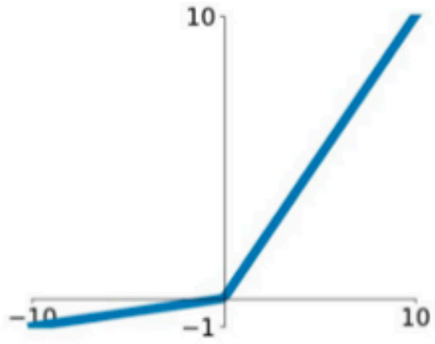
ReLU

$$\max(0, x)$$



Leaky ReLU

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

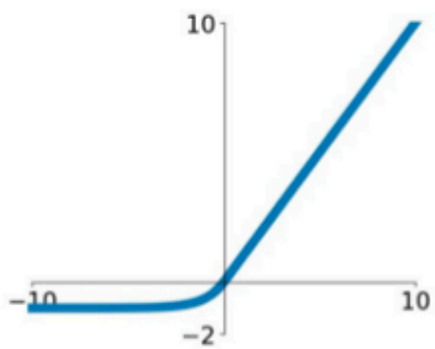


Maxout

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

ELU

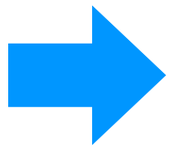
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

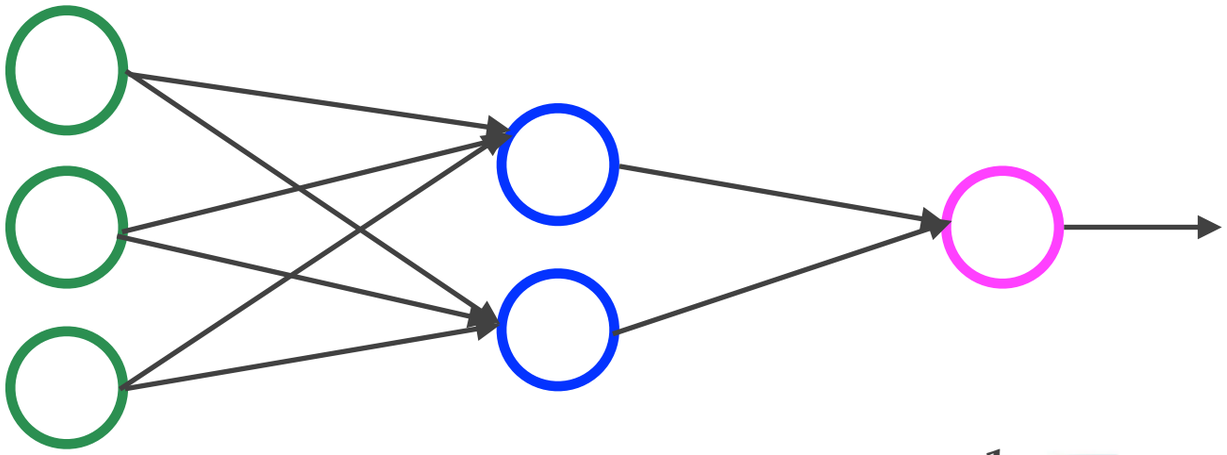


Introduction to Exponential Linear Unit

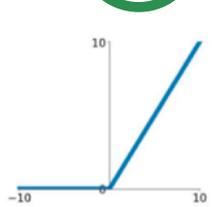
class 1 class 2

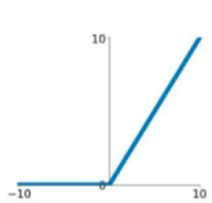
Two-class Classification

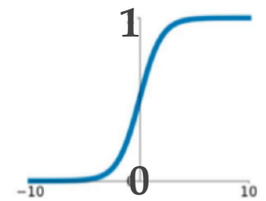


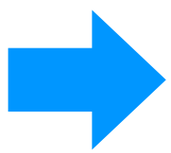


activation functions









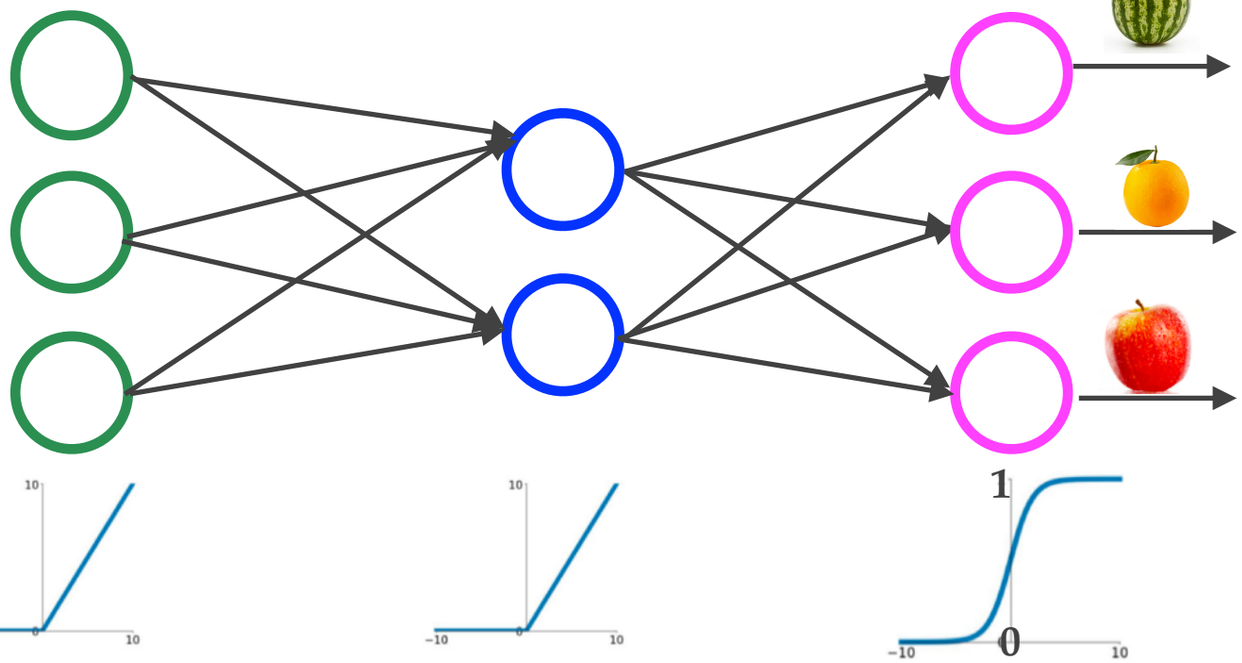
Fully connected 1 (3 nodes, ReLU) → Fully connected 2 (2 nodes, ReLU) → Fully connected 3 (1 nodes, sigmoid) → 0.99

```
model.add(tf.keras.layers.Dense(3, activation='relu'))
model.add(tf.keras.layers.Dense(2, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

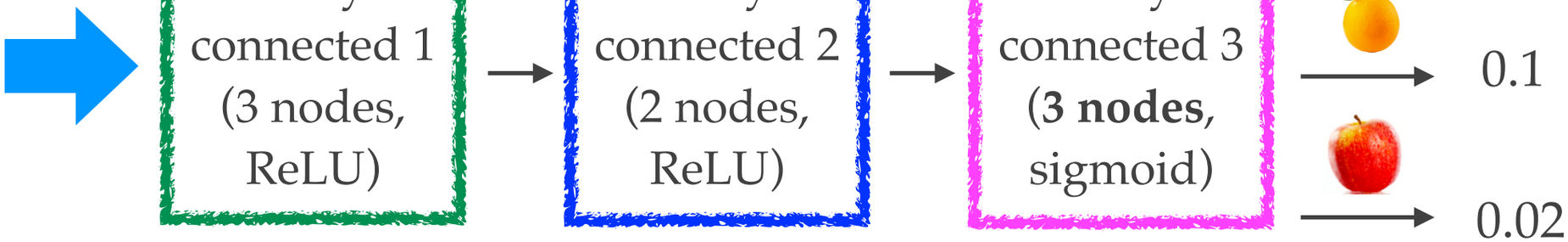
class 1 class 2 class 3



Three-class Classification



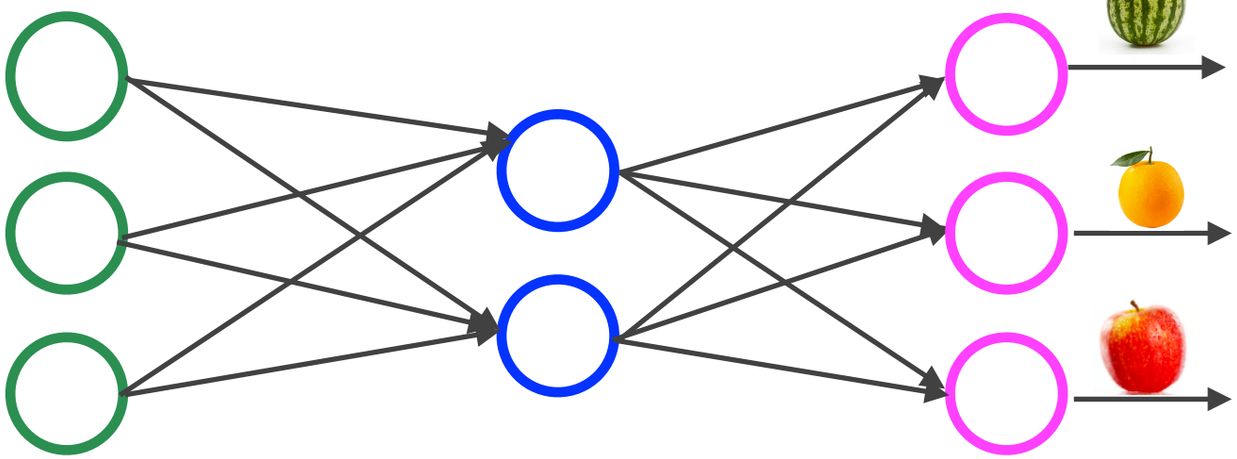
activation functions



What if our model outputs something like 0.99 0.99 0.99 ? Not good

class 1 class 2 class 3

Three-class Classification

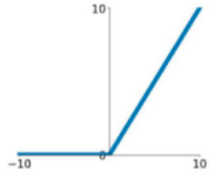


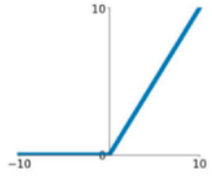
probability of being class 1

probability of being class 2

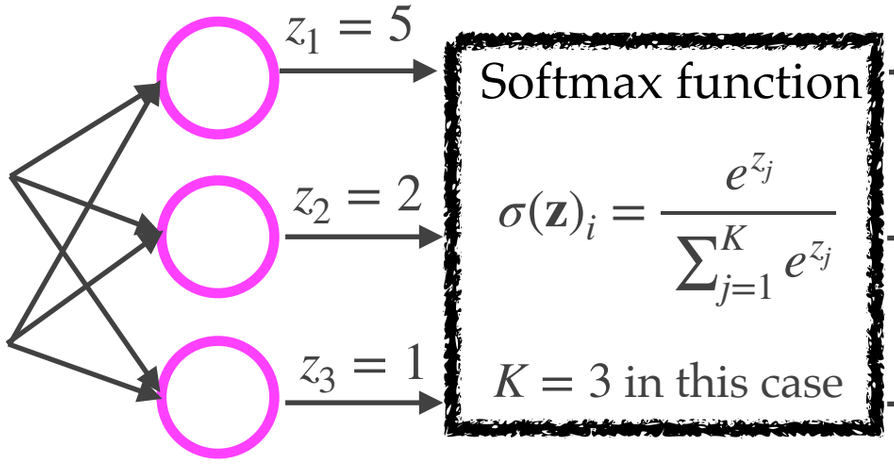
probability of being class 3

activation functions





Softmax function



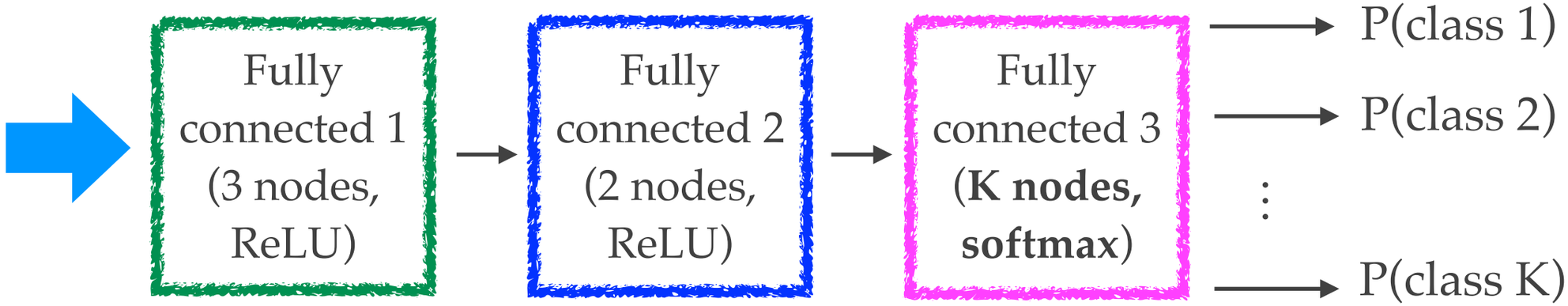
$$\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}} = \frac{e^{5}}{e^{5}+e^{2}+e^{1}} = 0.936$$

$$\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}} = \frac{e^{2}}{e^{5}+e^{2}+e^{1}} = 0.047$$

$$\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}} = \frac{e^{1}}{e^{5}+e^{2}+e^{1}} = 0.017$$

$$0.936 + 0.047 + 0.017 = 1$$

Extend the Model to K-class Classification




```
model.add(tf.keras.layers.Dense(3, activation='relu'))
model.add(tf.keras.layers.Dense(2, activation='relu'))
model.add(tf.keras.layers.Dense(K, activation='softmax'))
```

Fully connected layer
(Dense)

Optimizer

SGD
Adam
RMSprop

Convolutional layer

Conv1D, 2D, 3D, …
separable Conv

Pooling layer

max-pooling
average-pooling

Evaluation metric

accuracy
F1-score
AUC
confusion matrix

Activation function

sigmoid
softmax
ESP (swish)
ReLU

Loss function

categorical crossentropy
binary crossentropy
mean squared error
mean absolute error

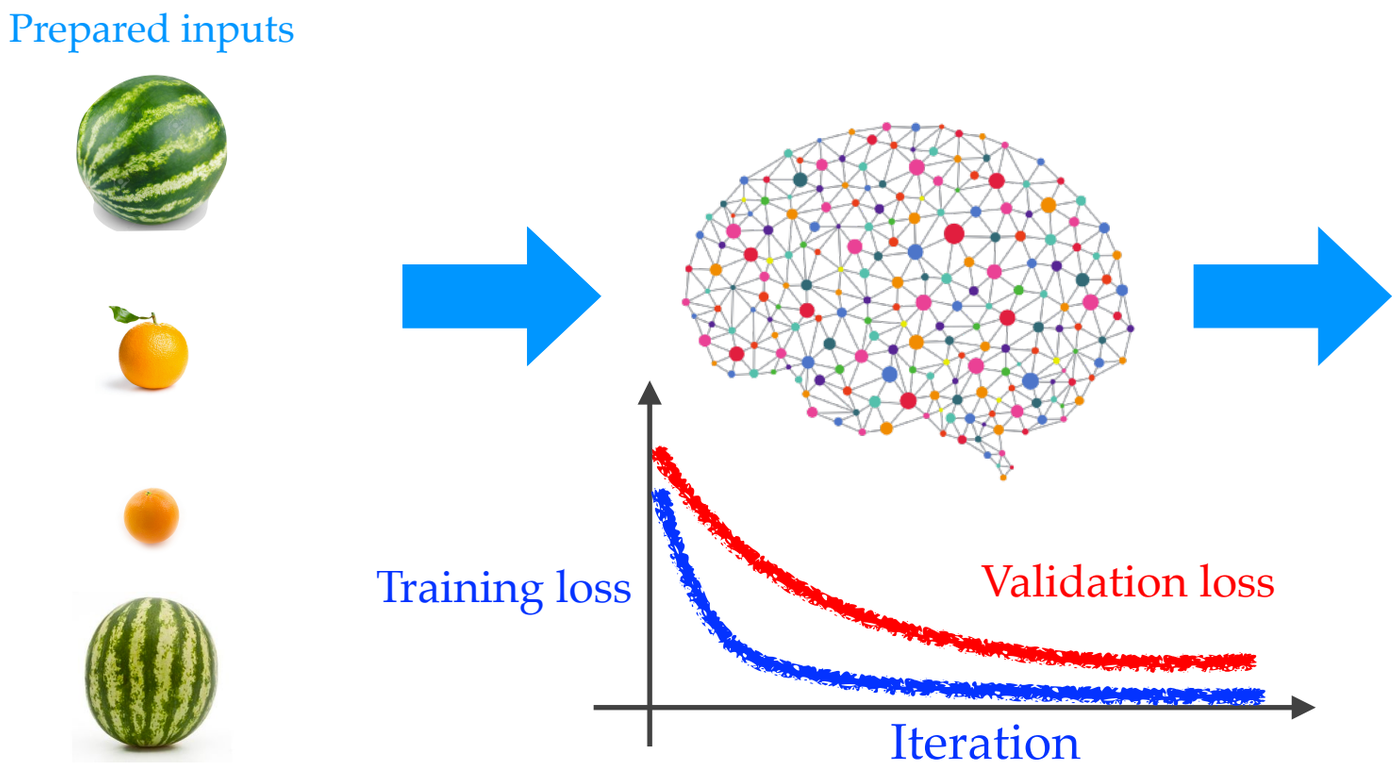
Regularization

Dropout
Data augmentation
l_1, l_2 regularizations

- Combine basic components to build a neural network
  - More components → “More” representative power

Image Classification Using Deep Learning

- Training phase: Optimize the weights of a deep neural network



- Test phase: Perform prediction using the trained neural network

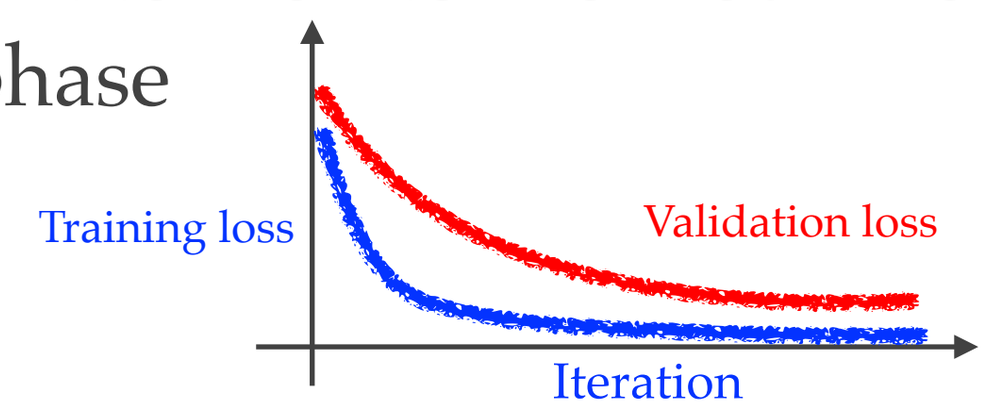
Loss Function

(cost function and error function)

- A function L that maps values of variable(s) onto a real number

  For example, $L(y, \hat{y})$ gives us a number that tells us how "different" y and $\hat{y}$ are

- We have used it to monitor how our model performs during the training phase



$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|$$

- Mean squared error (MSE) and mean absolute error (MAE) are two popular choices for regression

- Cross-entropy is the most common choice for a classification task

Cross-entropy

- Well suited to comparing probability distributions

3-class classification

deep learning layers → Fully connected (**3 nodes, softmax**)

	predicted prob. dist. $\hat{y}$	true prob. dist. y
	0.7	1
	0.1	0
	0.2	0

One-hot encoding

$$L(y,\hat{y}) = -\sum_{k=1}^{K} y_k log\hat{y}_k \quad \text{by definition } (K=3)$$

$$= -(1 \times log(0.7) + 0 \times log(0.1) + 0 \times log(0.2)) = -1 \times log(0.7) = 0.15$$

Assume that we have updated the weights of our model and got $\hat{y} = [0.9, 0.1, 0]$, then $L(y,\hat{y}) = -1 \times log(0.9) + 0 + 0 = 0.046$

Lower loss. $\hat{y}$ has become more similar to y than the previous case.
Our model performs better!