BT4222
Mining Business Insights with Web Data
Phase 3 Neural Network Methods

Deep Neural Networks

Dr. Qiuhong Wang
Term1 2022-23

Agenda

1. Why deep learning
2. The perceptron
3. Activation functions
4. Feed forward process
5. From perceptron to deep neural networks
6. Multi-class Neural Networks
7. Backpropagation
8. Regularization: dropout and early stopping

History

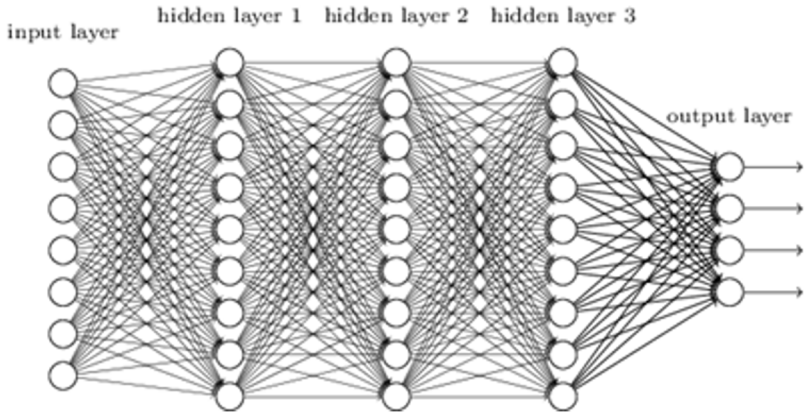
- The study of Artificial Neural Networks (ANNs) has been inspired by the observation of biological learning systems, and was first proposed in **1943** by the neurophysiologist **Warren McCulloch** and the mathematician **Walter Pitts.**
- The early successes until the **1960s** led to the widespread belief that we would soon be conversing with truly intelligent machines. As it became clear that this promise would go unfulfilled, funding flew elsewhere and ANNs entered a long dark era.
- By the **1990s**, powerful alternative Machine Learning techniques such as **Support Vector Machines** were favored by most researchers.

We are now witnessing another wave of interest in ANNs.

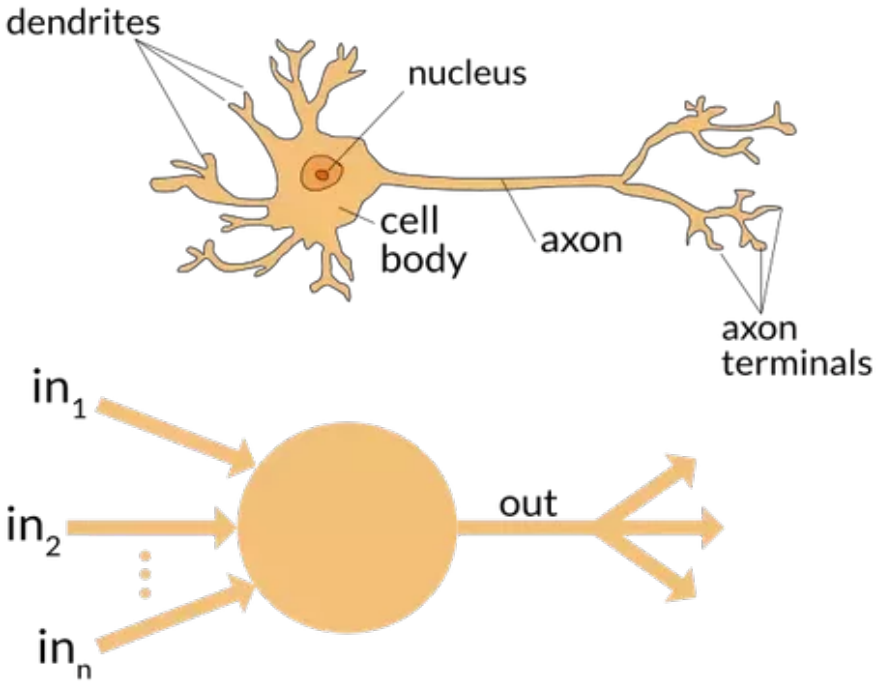
- Huge quantity of data
- Computing power
- Open-sourced platform, e.g. tensorflow
- Training algorithms

Neural Networks

- Human Brain vs. Neural Network
  - Many neurons or neuron-like switching units
  - Many weighted interconnections among neurons/units
  - Parallel computation

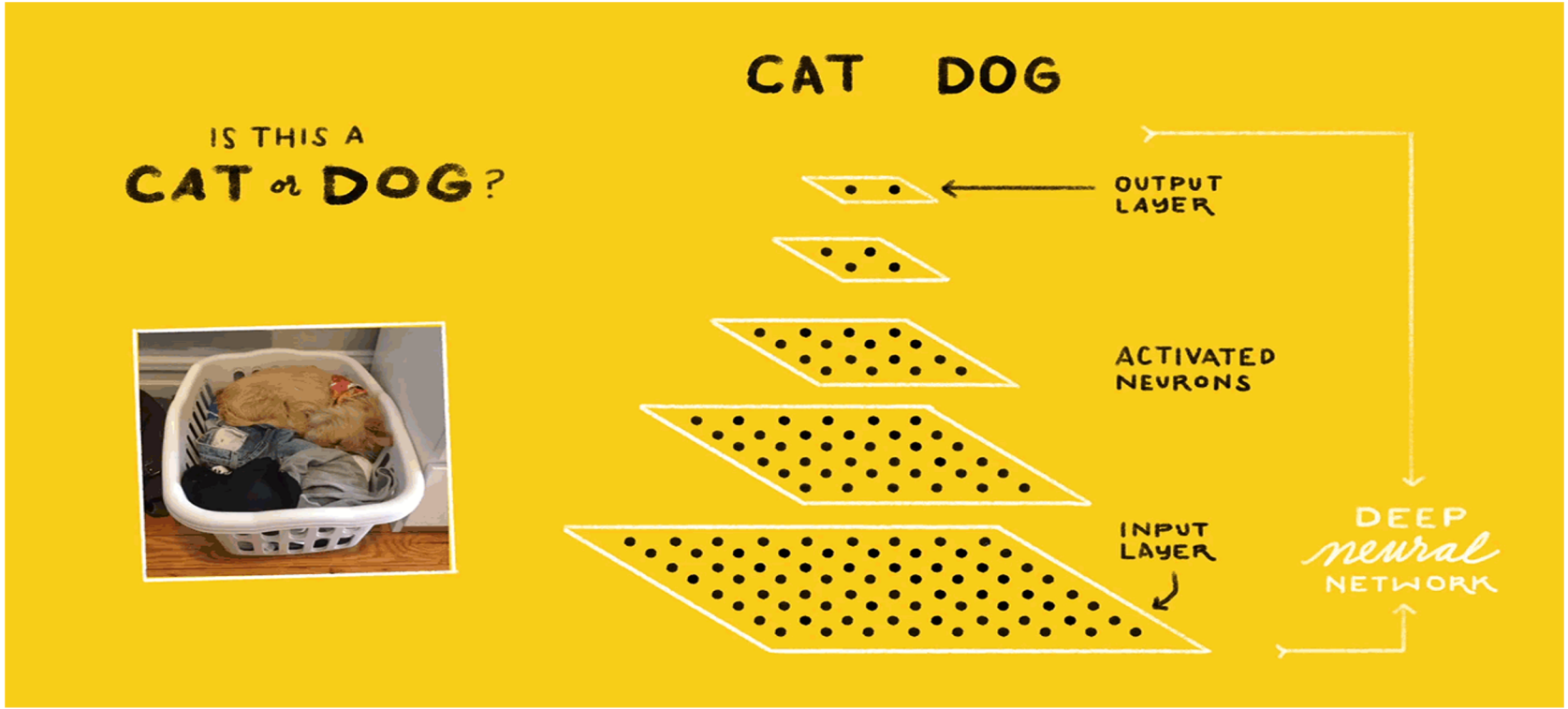


- From Wiki: NN is based on a collection of connected units of nodes called artificial **neurons** which loosely model the neurons in a biological brain.

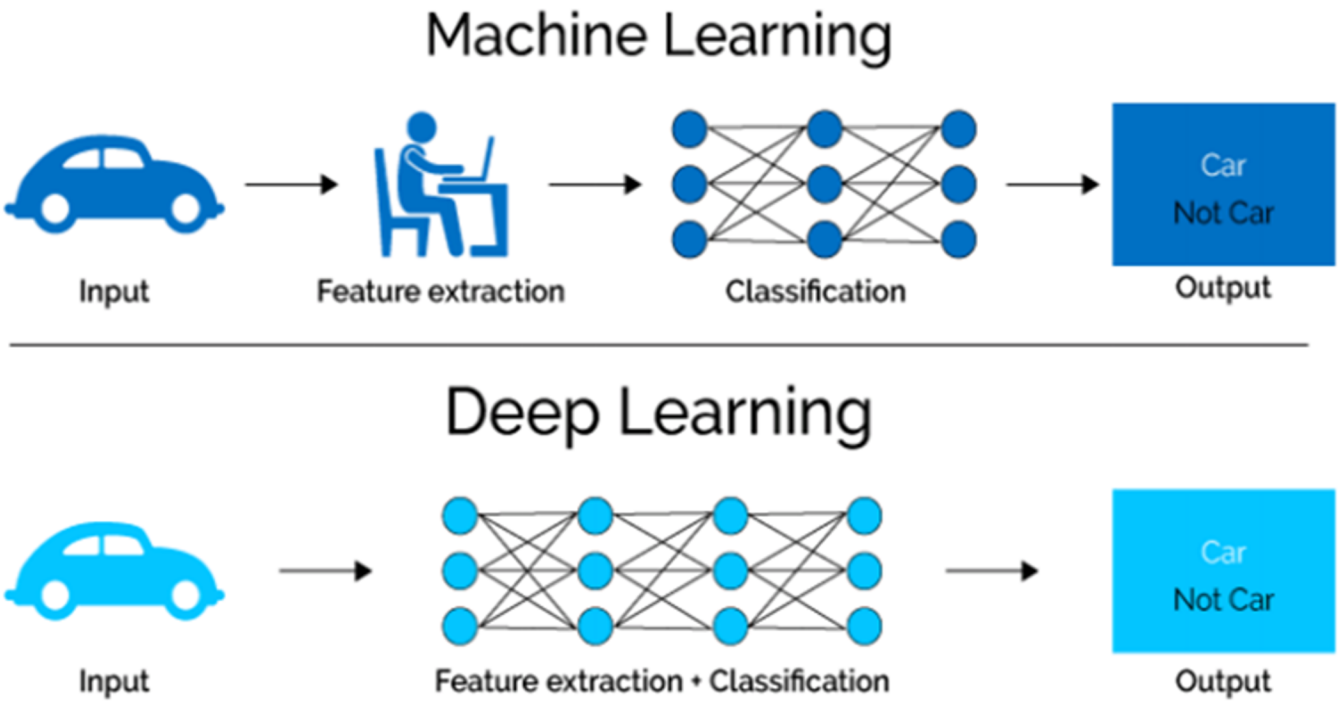


https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science-and-biological-neural-network

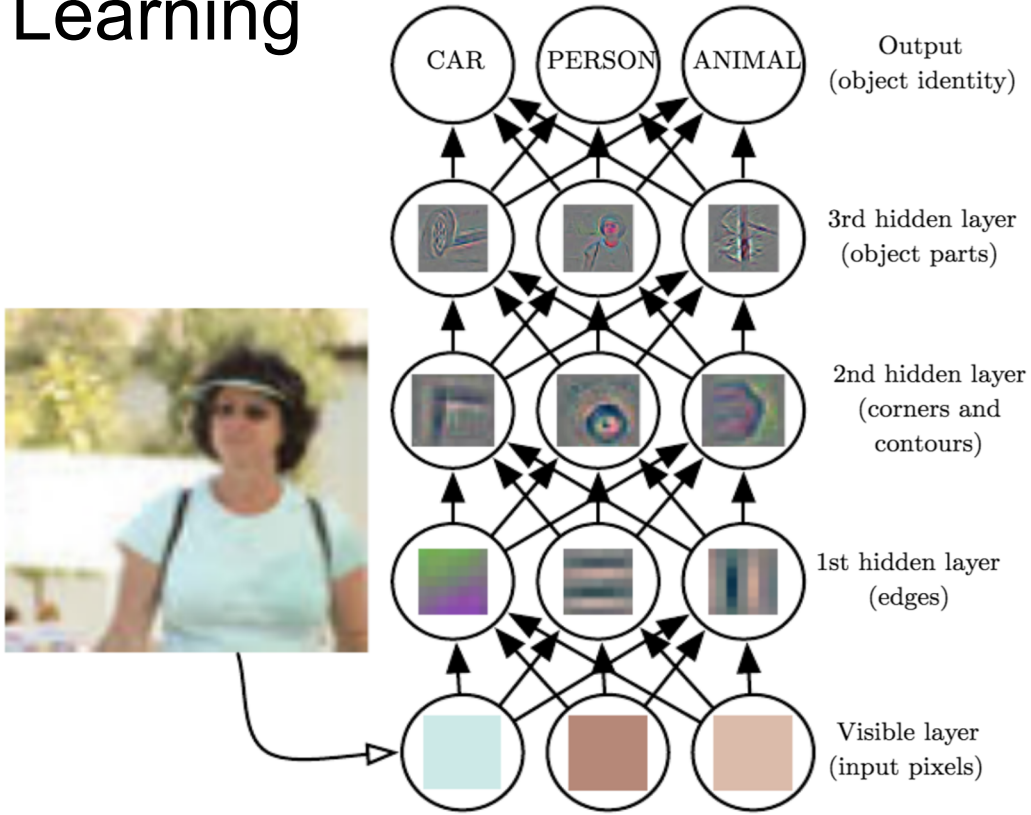
Neural network is a way of learning **representation**



Representation Learning

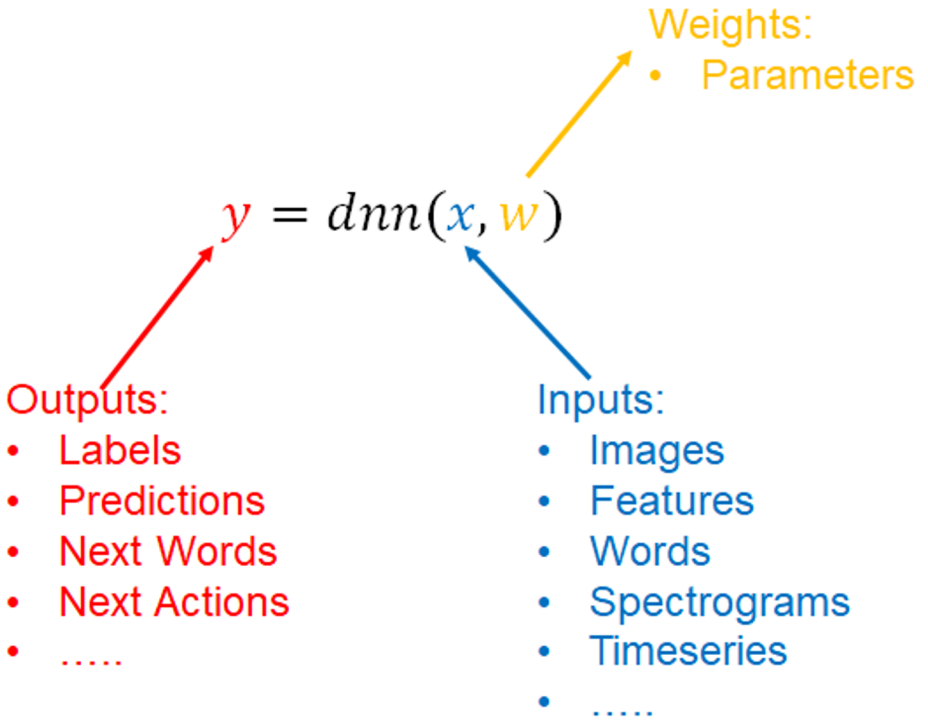


Representation Learning



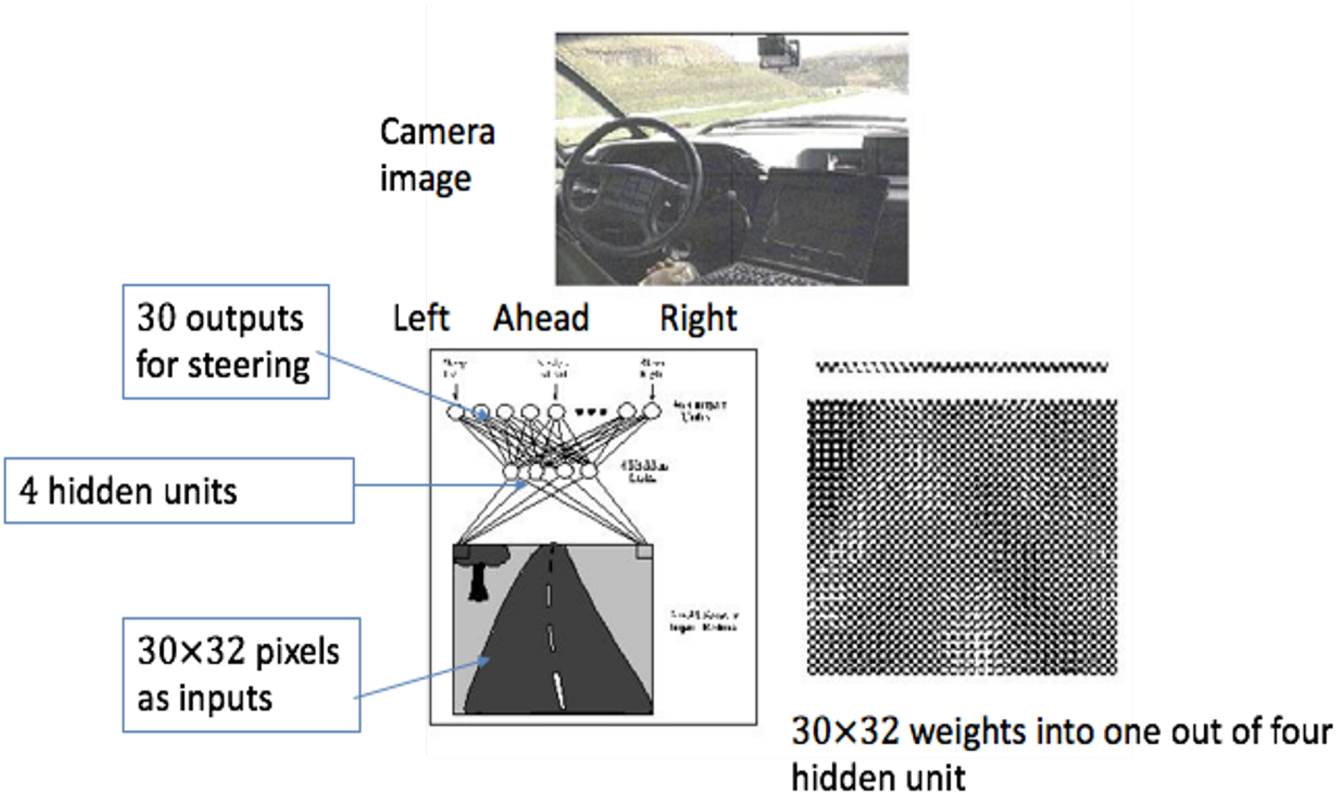
What can DNN do?

- Computer Vision
- Natural Language Processing
- Language Translation
- Reinforcement Learning
- …….



Autonomous Land Vehicle In a Neural Network

Drives 112 km/h on a public highway

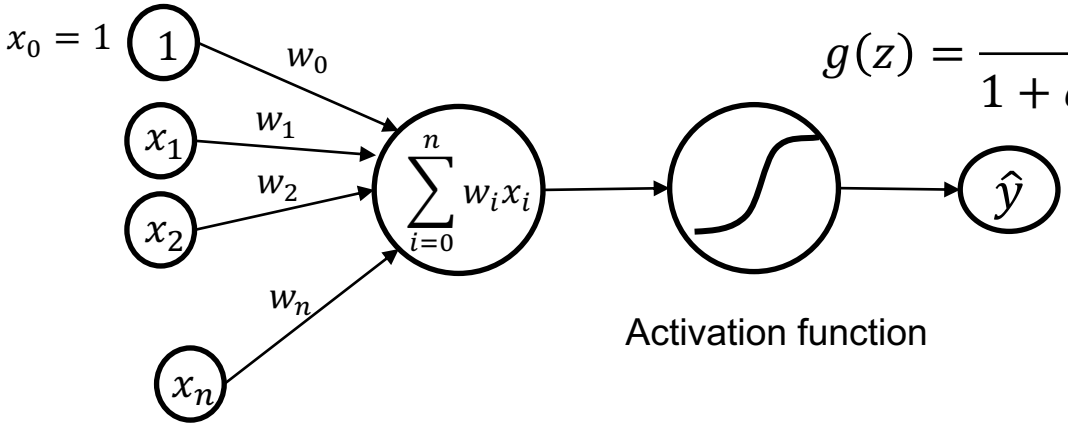


Agenda

1. Why deep learning
2. The perceptron
3. Activation functions
4. Feed forward process
5. From perceptron to deep neural networks
6. Multi-class Neural Networks
7. Backpropagation
8. Regularization: dropout and early stopping

Logistic Unit

- A standard neural network can be represented by network of logistic units.



$$g(z) = \frac{1}{1 + e^{-w^T x}}$$

A neuron is a logistic regression unit:

- performs a dot product with the input and its weights
- adds the bias and apply the non-linearity

$$\hat{y} = g(w_0 + X^T W)$$

$$\hat{y} = g\left(w_0 + \sum_{i=0}^{n} w_i x_i\right)$$

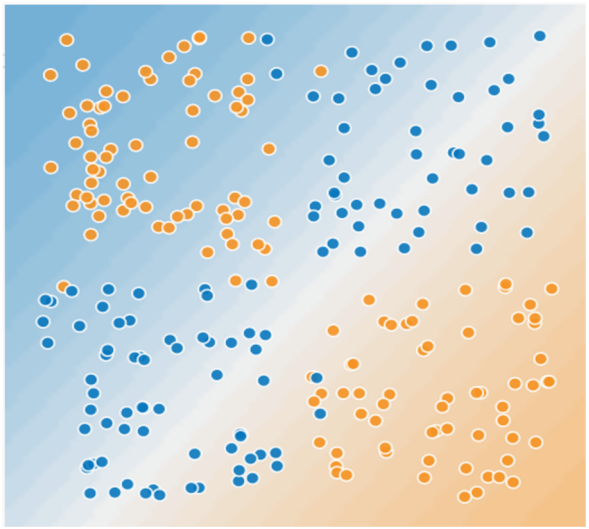
where: $X = \begin{bmatrix} x_1 \\ \cdots \\ x_n \end{bmatrix}$ and $W = \begin{bmatrix} w_1 \\ \cdots \\ w_n \end{bmatrix}$

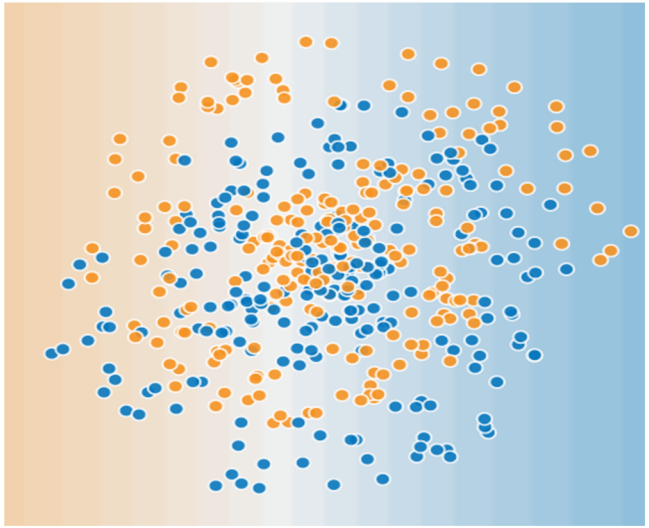
Agenda

1. Why deep learning
2. The perceptron
3. Activation functions
4. Feed forward process
5. From perceptron to deep neural networks
6. Multi-class Neural Networks
7. Backpropagation
8. Regularization: dropout and early stopping

Why to introduce Non-linearity?

Problem revisited





- The non-linearities activation function increases the capacity of model

Ways to learn non-linearities

- Feature crosses
  - Use powers and interactions as inputs
  - Inputs are still linearly combined
- Activation functions
  - Feed linearly combined input into non-linear functions
  - Decide what is to be fired to the next neuron.

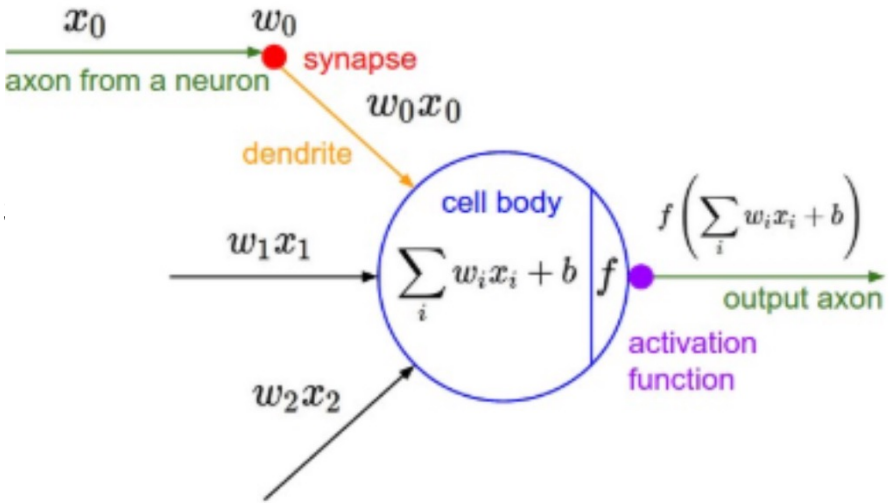
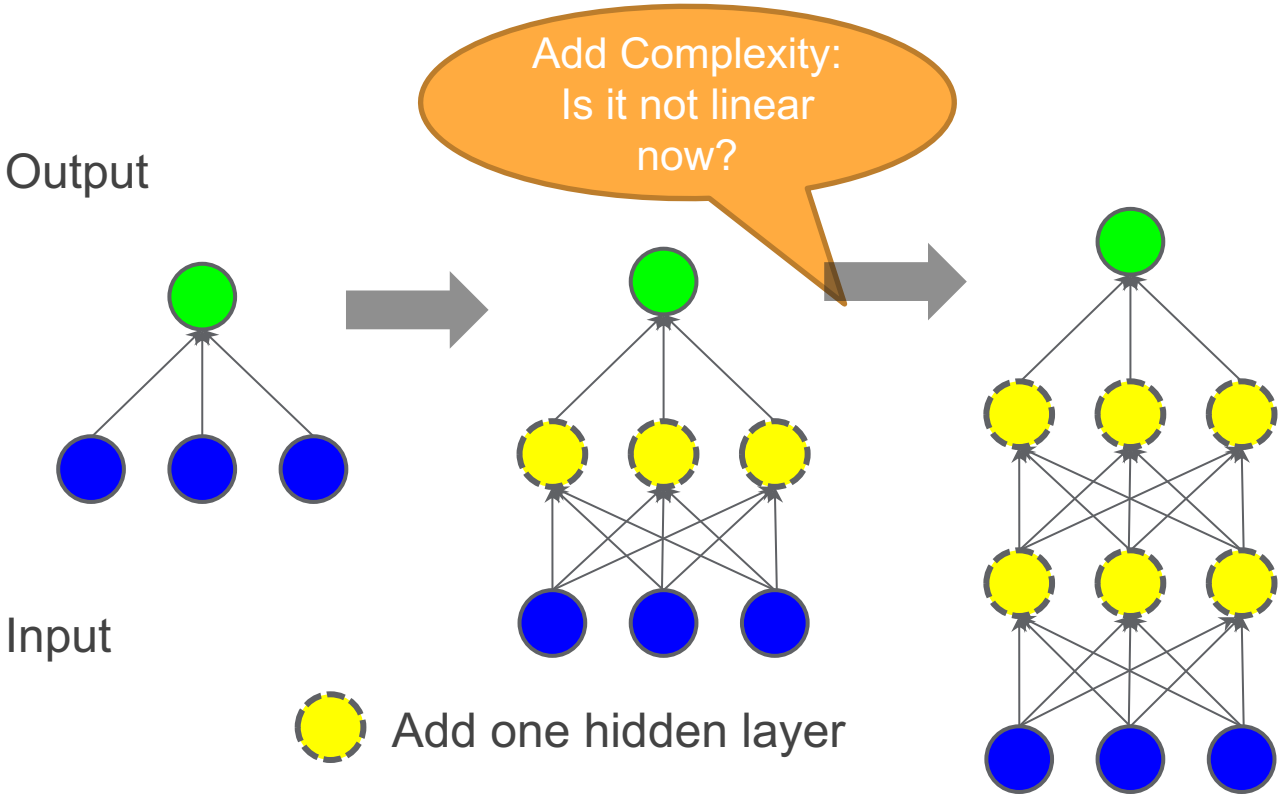


Figure source: cs231n by Stanford

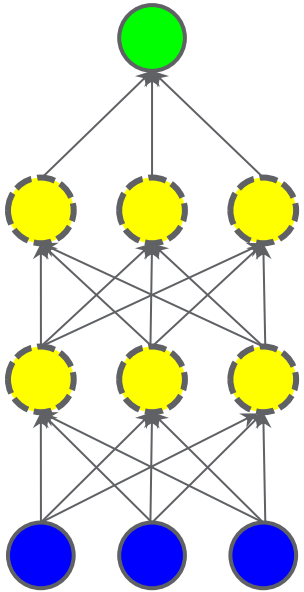
From A Linear Model to Another Linear Model

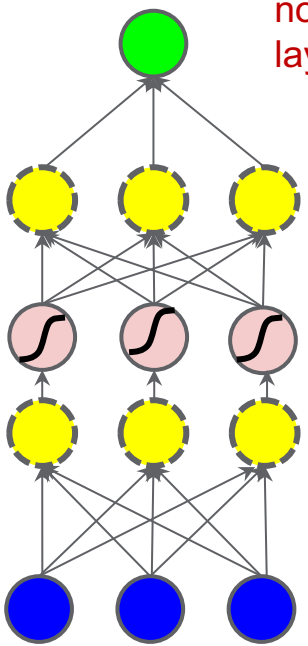


Without non-linearities, deep neural networks is meaningless: each extra layer is just one linear transform.

From A Linear Model to A Non-Linear Model

Output





We Usually don't draw non-linear transform layer!

Input

Add two hidden layers

Insert a non-linear transformation layer (a.k.a. activation function)

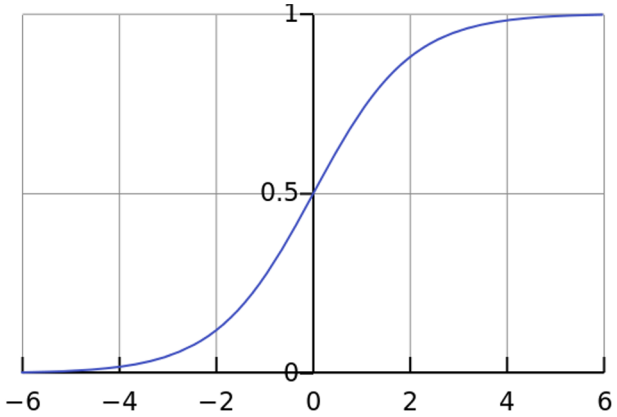
How to select activation functions?

- Low computational expense
- Zero-centered (preferred, not necessary)
  Output of the activation function should be symmetrical at zero so that the gradients do not shift to a particular direction.
- Differentiable (necessary)
  Neural networks are trained using the *gradient descent* process, hence the layers in the model need to differentiable or at least differentiable in parts.
- Alleviate vanishing gradient problem (pending for more explanation)

Sigmoid Function

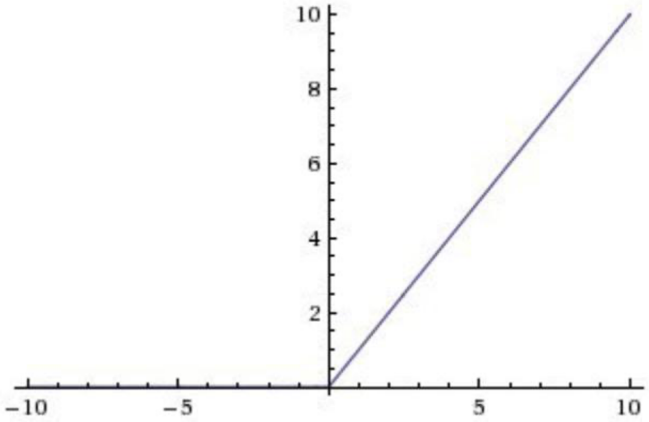
$$f(z) = \frac{1}{1+e^{-z}} \qquad g(z) = \frac{1}{1+e^{-w^T x}}$$

- Keep activation bound in range (0,1)
- Good for classification problem (clear distinctions)
- Smooth non-linear function
- Vanishing gradients (near −horizontal part of the curve)



Relu Function (Rectified Linear Unit)

$$f(z) = \max(z, 0)$$

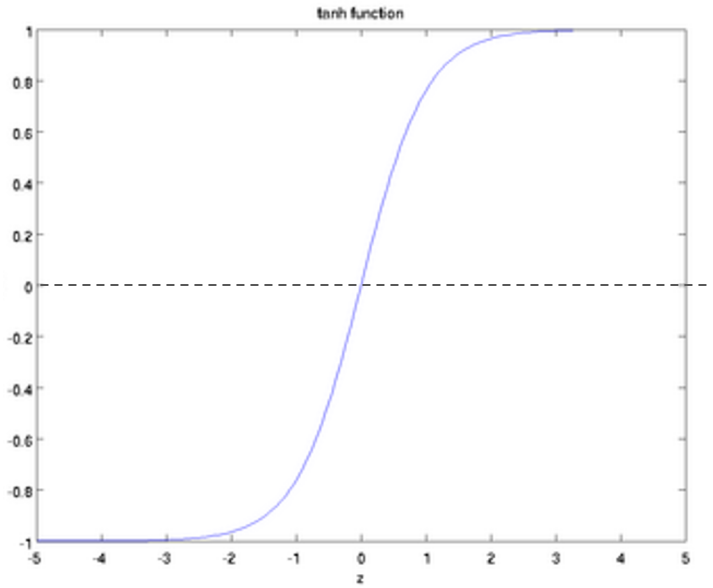


- Nonlinear Activation, but may blow up
- Sparsity/ Efficient
- Less computational expensive that is good for deep structure
- **Dying ReLu problem. (Several neurons will not respond)**

Tanh Function

$$f(z) = \frac{2}{1+e^{-2z}} - 1$$

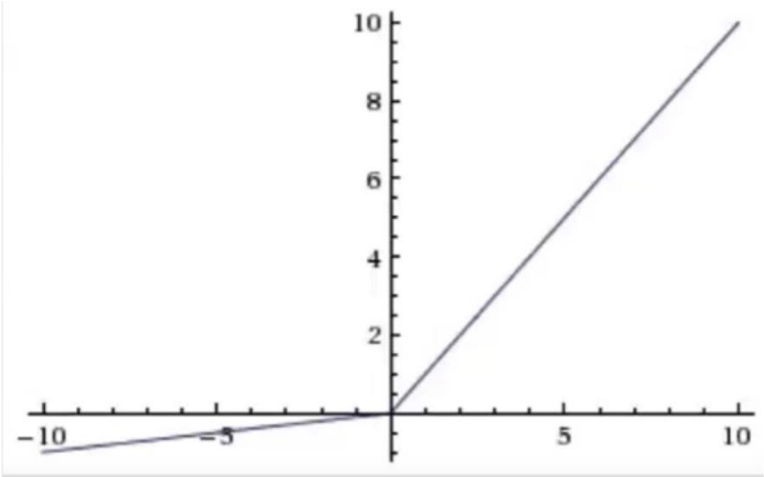
- Keep activation bound in range (-1,1)
- More steeper curve
- Smooth non-linear function
- Vanishing gradients (near–horizontal part of the curve)



Leaky Relu Function

$$f(z) = \mathbb{I}(x < 0)(ax) + \mathbb{I}(x > 0)(x)$$

- Proposed to solve the dying ReLu problem
- Still non-linear

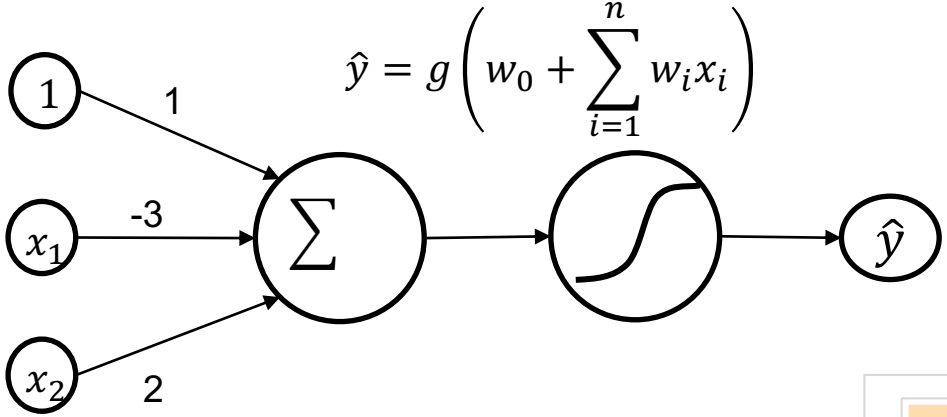


SoftMax

- Extend from binary case
- Model the distribution of p(y=i|x), i = 1, …, K, with softmax:

$$p(y = i|\mathbf{x}) = \frac{e^{w_i^T x}}{\sum_j e^{w_j^T x}}$$

The perceptron: Example

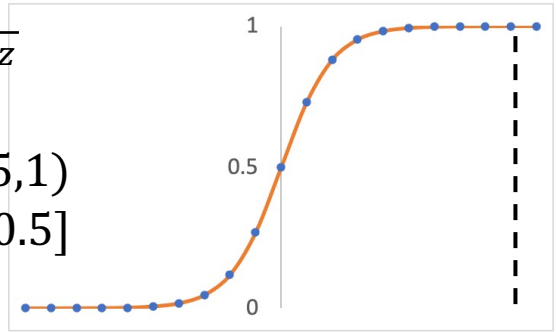


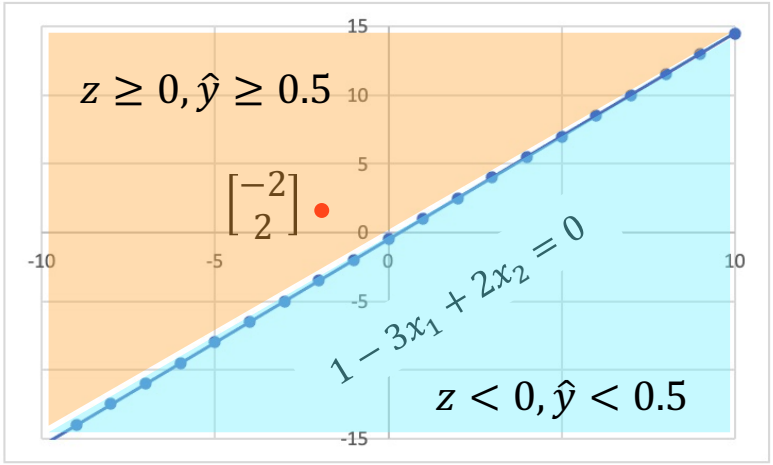
We have $w_0 = 1$ and $W = \begin{bmatrix} -3 \\ 2 \end{bmatrix}$

$$\hat{y} = g(w_0 + X^T W)$$
$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} -3 \\ 2 \end{bmatrix}\right)$$
$$= g(1 - 3x_1 + 2x_2)$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

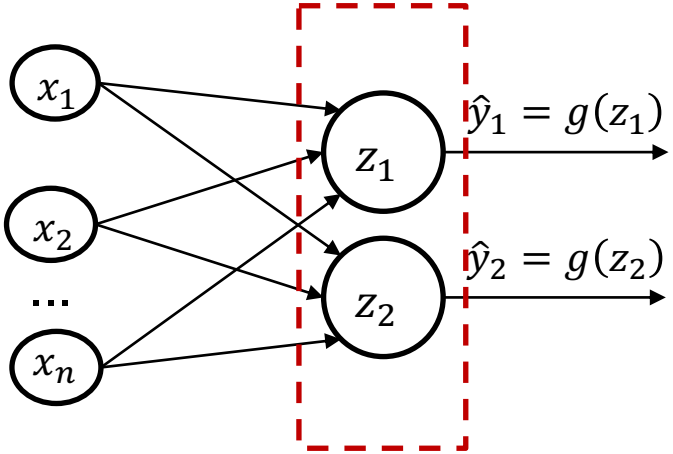
$z \geq 0, g(z) \in [0.5,1)$
$z < 0, g(z) \in (0,0.5]$





Multi Output perceptron

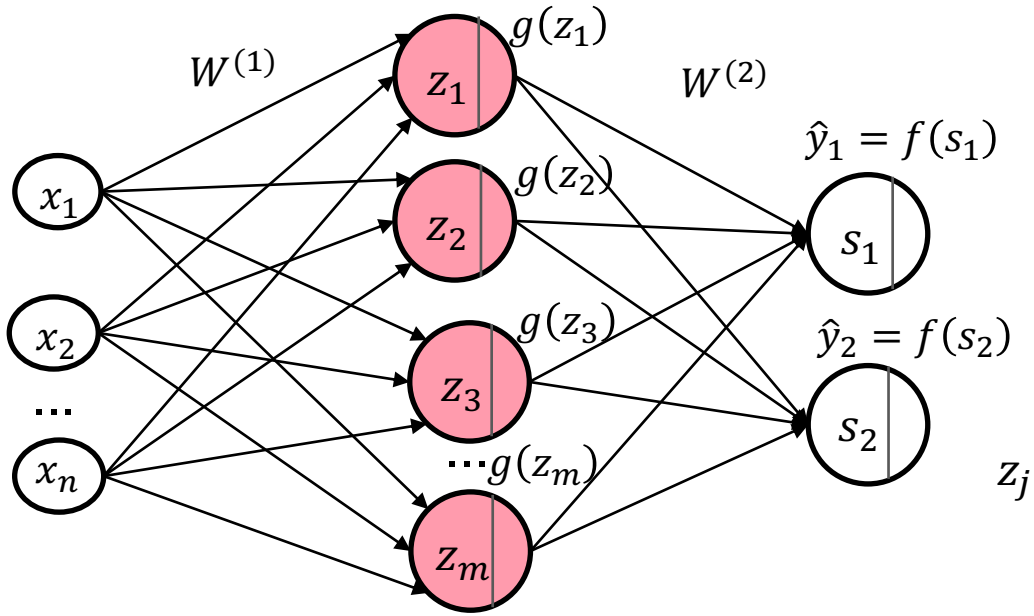
Because all inputs are densely connected to all outputs, these layers are called Dense layers.



$$\hat{y}_k = g(w_0 + \sum_{i=1}^{n} w_i x_i), k = 1,2$$

```
Import tensorflow as tf
layer = tf.keras.layers.Dense(units=2)
```

Single Layer Neural Network



Different weights results in different hidden nodes and different output

$g(z_j)$ *is the activation function for the hidden layer*

$f(s_k)$ *is the activation function for the output layer*

$$z_j = w_{0,j}^{(1)} + \sum_{i=1}^{n} w_{i,j}^{(1)} x_i \, , j = 1, .. , m$$

$$s_k = w_{0,k}^{(2)} + \sum_{j=1}^{m} w_{j,k}^{(2)} g(z_j)$$

$$\hat{y}_k = f(s_k), k = 1 \; or \; 2$$

Agenda

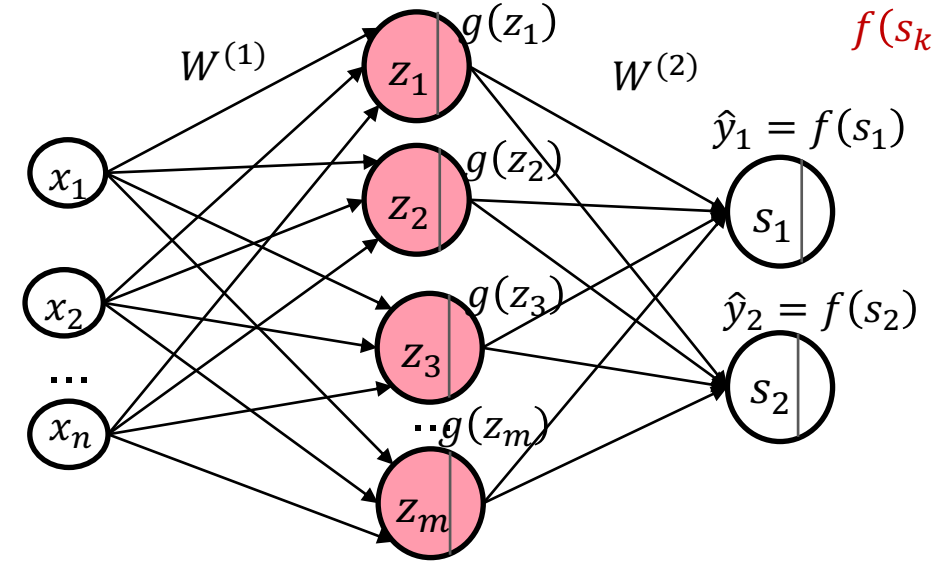
1. Why deep learning
2. The perceptron
3. Activation functions
4. Feed forward process
5. From perceptron to deep neural networks
6. Loss functions
7. Training and gradient descent
8. Backpropagation
9. Setting the learning rate
10. Batched gradient descent
11. Regularization: dropout and early stopping

Feed Forward Process

In sum, $\hat{y}_k = f\left(w_{0,k}^{(2)} + \sum_{j=1}^{m} w_{j,k}^{(2)} g\left(w_{0,j}^{(1)} + \sum_{i=1}^{n} w_{i,j}^{(1)} x_i\right)\right)$

$g(z_j)$ can be sigmoid or ReLU

$f(s_k)$is determined by the nature of data and problem



For hidden layer

$$z_j = w_{0,j}^{(1)} + \sum_{i=1}^{n} w_{i,j}^{(1)} x_i \,, j = 1, .., m$$

For output layer

$$s_k = w_{0,k}^{(2)} + \sum_{j=1}^{m} w_{j,k}^{(2)} g(z_j) \,, k = 1 \; or \; 2$$

Agenda

1. Why deep learning
2. The perceptron
3. Activation functions
4. Feed forward process
5. From perceptron to deep neural networks
6. Multi-class Neural Networks
7. Backpropagation
8. Regularization: dropout and early stopping

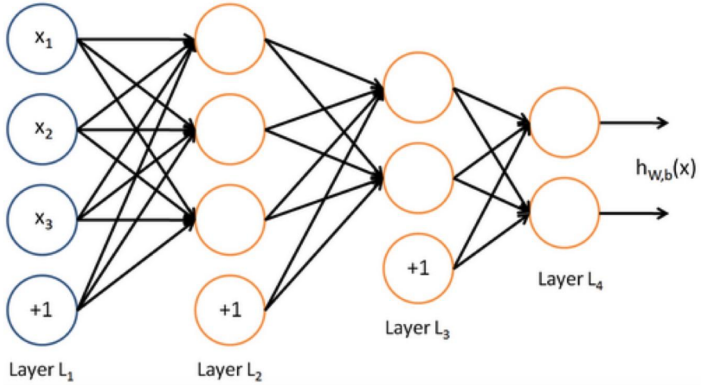
Stacking Layers

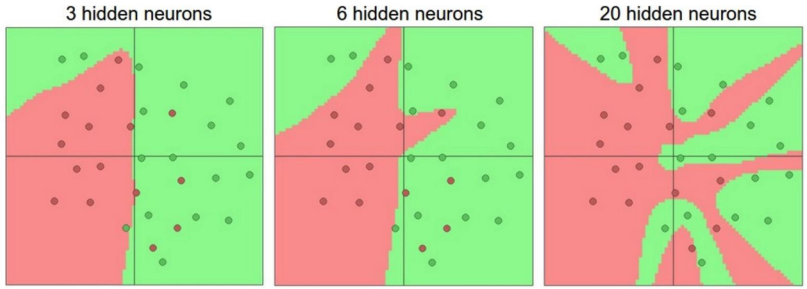
- We can feed the output of one layer into the next layer （a bunch of neuron computation)

- The output layer: (not require non-linear activation)
  - For classification: class scores
  - For regression: real-value target

- Measures to describe the size of neural networks:
  - Number of neurons: 3 + 2 + 2 = 7
  - Number of tunable parameters: (biases and weights) 3*4+2*3+2*2+3=25

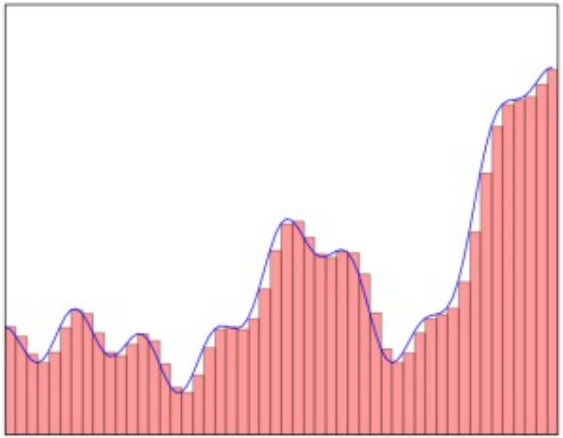
- Representation Power: neural networks with at least one hidden layer can approximate any continuous function.





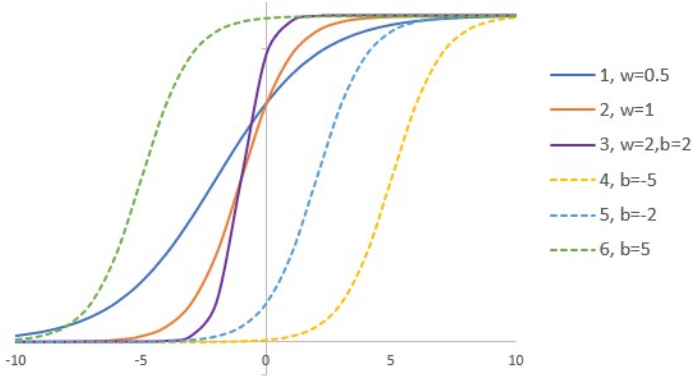
Universal Approximation Theorem

A multilayer network of neurons with a single hidden layer can be used to approximate any continuous function to any desired precision.

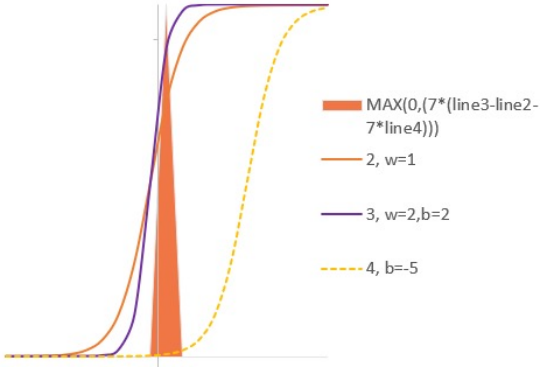


A function can be approximated
with superimposing several towers functions.

$$g(x) = \frac{1}{1 + e^{-(wx+b)}}$$



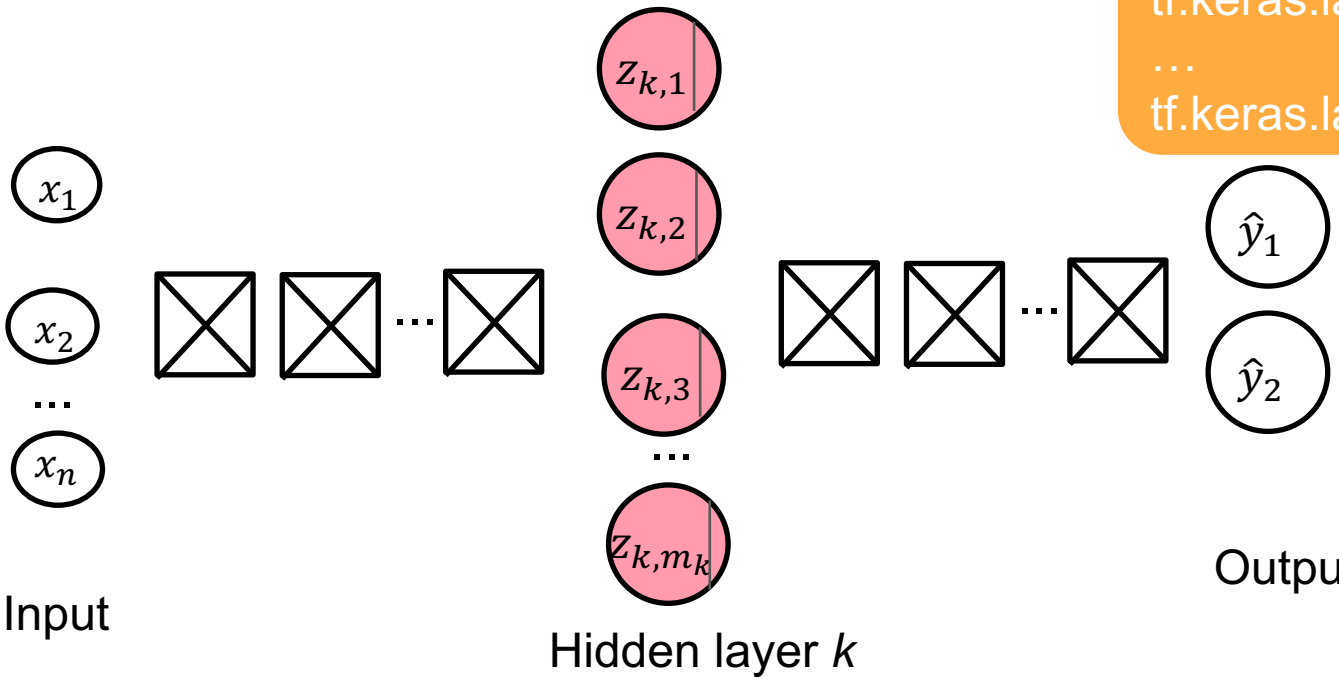
Each tower function is the linear combination of sigmoid functions with different parameters



Deep Neural Network

$$z_{k,j} = w_{0,j}^{(k)} + \sum_{i=1}^{m_{k-1}} w_{i,j}^{(k)} g\left(z_{k-1,i}\right), j = 1, .., m_k$$

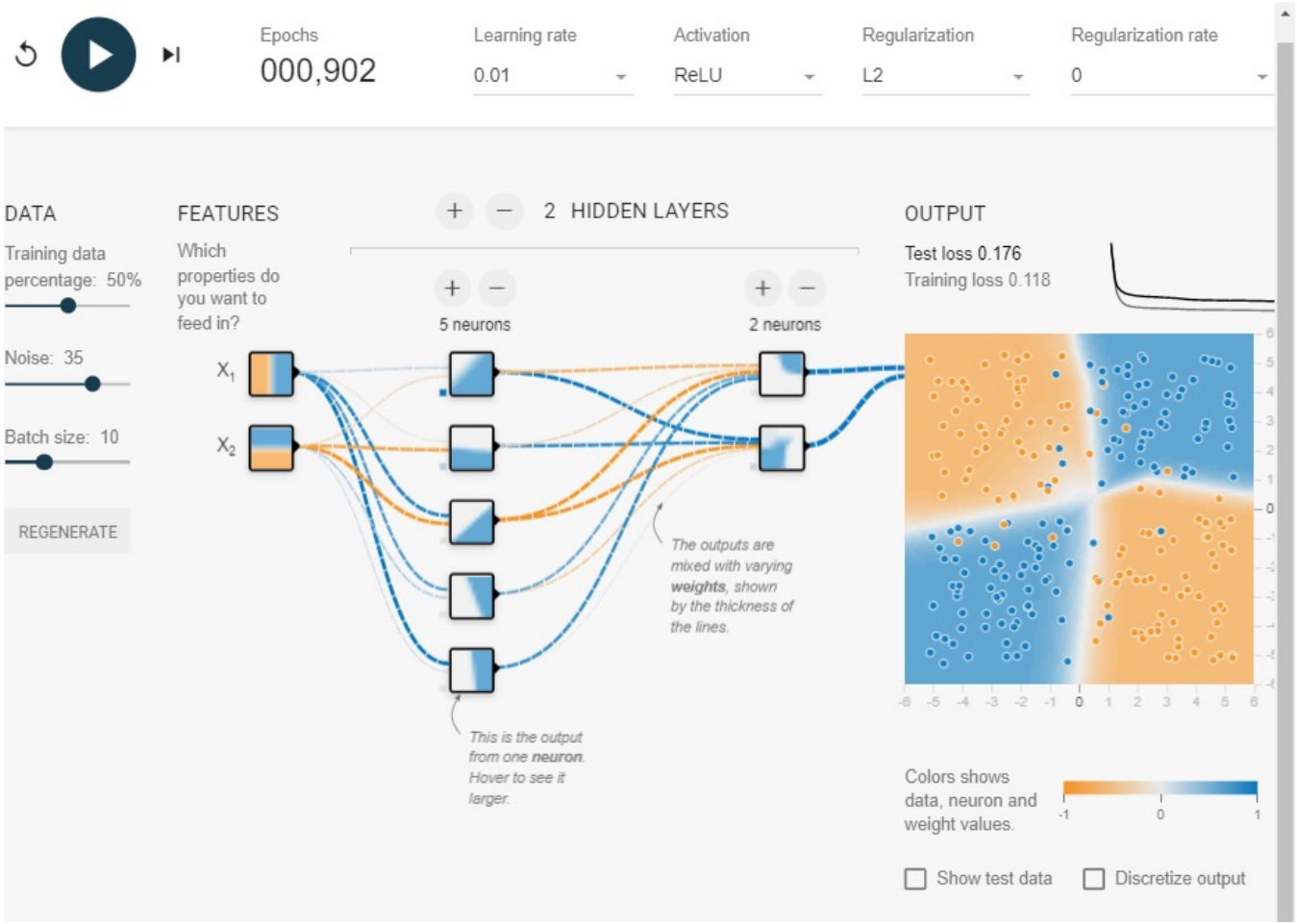
```
Import tensorflow as tf
model = tf.keras.Sequential([
 tf.keras.layers.Dense (n1),
tf.keras.layers.Dense (n2),
…
tf.keras.layers.Dense (nk),
…
tf.keras.layers.Dense (2) ])
```



Training for Neural Networks

1. Randomly initialize Model Parameters
2. FeedForward Computation
3. Backpropagation for gradient computation
4. Update the model parameters.
5. Repeat steps 2-4 until convergence

Playground


Epochs
000,902
Learning rate
0.01
Activation
ReLU
Regularization
L2
Regularization rate
0
DATA
Training data percentage: 50%
Noise: 35
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
2 HIDDEN LAYERS
5 neurons
2 neurons
X1
X2
The outputs are mixed with varying weights, shown by the thickness of the lines.
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.176
Training loss 0.118
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output

Agenda

1. Why deep learning
2. The perceptron
3. Activation functions
4. Feed forward process
5. From perceptron to deep neural networks
6. Multi-class Neural Networks
7. Backpropagation
8. Regularization: dropout and early stopping

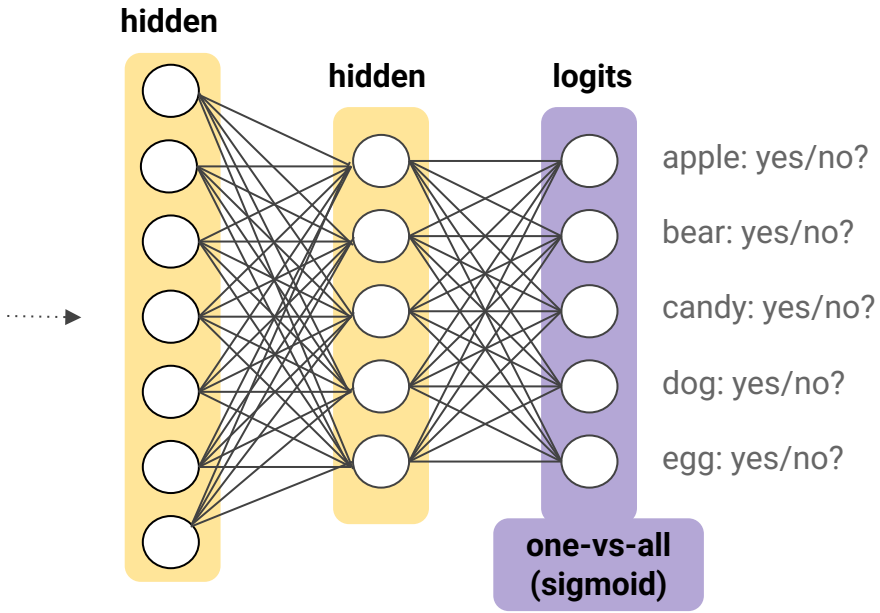
Multi-class Neural Networks

- Logistic regression gives useful probabilities for binary-class problems.
  - spam / not-spam
  - click / not-click

- What about multi-class problems?
  - animal, vegetable, mineral
  - red, orange, yellow, green, blue, indigo, violet
  - apple, banana, car, cardiologist, ..., walk sign, zebra, zoo

One-Vs-All Multi-Class

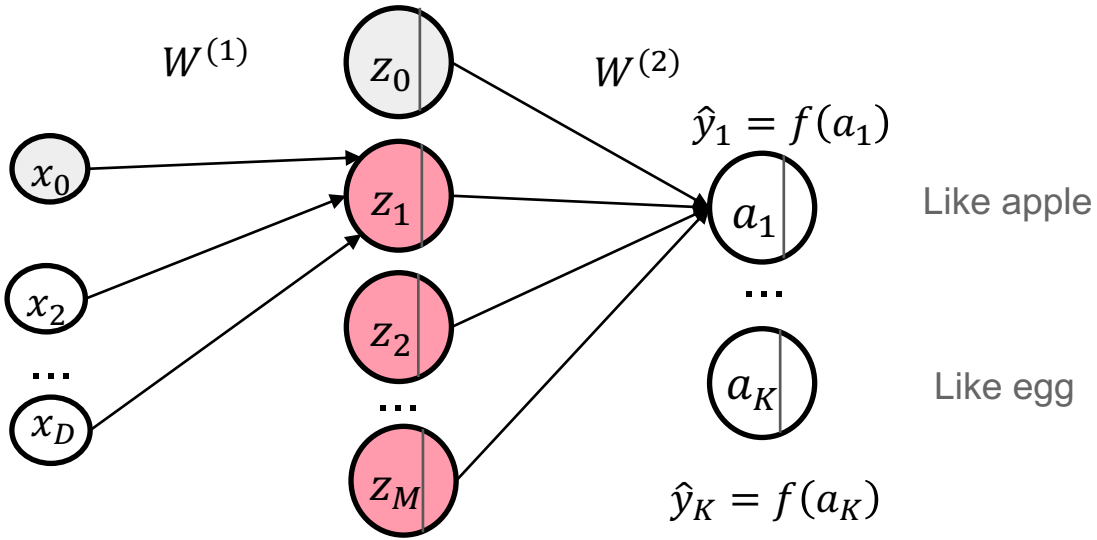
- Create a unique output for each possible class
- Train that on a signal of “my class” vs “all other classes”
- Can do in a deep network, or with separate models
- Typically optimize logistic loss



Output Activation and Error Functions

Multiple independent binary classification

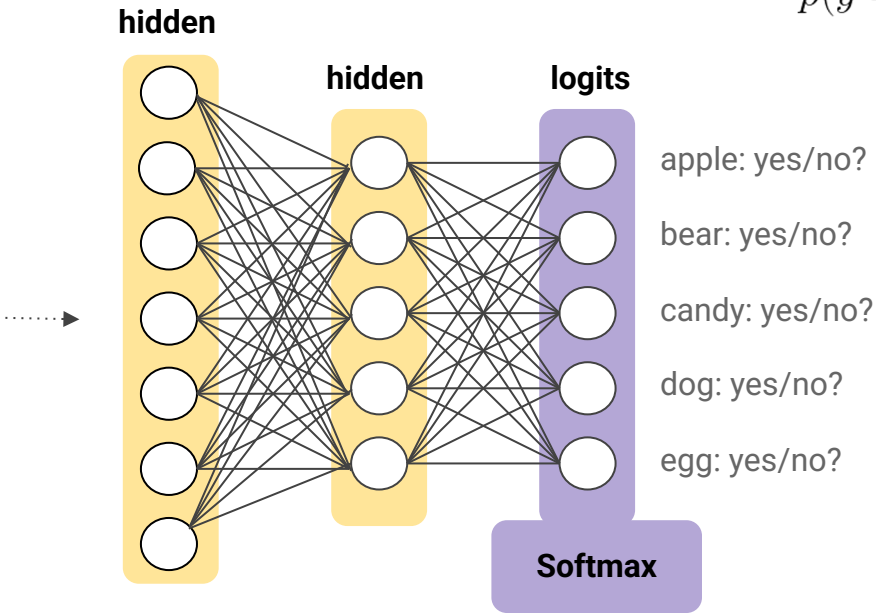
- Output: sigmoid function $\hat{y}_k(x_n, w) = \frac{1}{1+e^{-a_k}}$
- Negative log of likelihood:
- $E(w) = -\sum_{n=1}^{N}\sum_{k=1}^{K}\{t_{nk} \ln \hat{y}_k(x_n, w) + (1 - t_{nk}) \ln(1 - \hat{y}_k(x_n, w))\}$



Softmax Multi-Class

- Add an additional constraint:
  - Require output of all one-vs-all nodes to sum to 1.0
  - Allows outputs to be interpreted as probabilities
- Typically optimize cross-entropy loss

$$p(y = j|\mathbf{x}) = \frac{exp(\mathbf{w}_j^T\mathbf{x} + b_j)}{\sum_{k \in K} exp(\mathbf{w}_k^T\mathbf{x} + b_k)}$$



Output Activation and Error Functions

Multiple-class classification

- Output: Soft-max function $\hat{y}_k(x_n, w) = \frac{exp(a_k)}{\sum_{k=1}^{K} exp(a_k)}$
- Cross-entropy:
- $E(w) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln \hat{y}_k(x_n, w)$

One-vs-All Multi-Class vs. Softmax Multi-Class

- Example, to calculate negative log of likelihood loss (NLL) and cross-entropy loss for one observation in a three-class classification problem.
- Actual classification {1,0,0}; predicted output $a_k\{1.5,1,0.8\}$

Class	a_k	t_{nk}	$L = \frac{1}{1+e^{-a_k}}$	$S = \frac{exp(a_k)}{\sum_{k=1}^{K} exp(a_k)}$	$t_{nk} \ln L$	$(1-t_{nk}) \times \ln(1-L)$	$t_{nk} \ln S$
1	1.5	1	0.82	0.48	0.82	0.00	0.48
2	1	0	0.73	0.29	0.00	0.27	0.00
3	0.8	0	0.69	0.24	0.00	0.31	0.00

NLL: $E(w) = -\sum_{n=1}^{N}\sum_{k=1}^{K}\{t_{nk} \ln \hat{y}_k(x_n, w) + (1-t_{nk}) \ln(1-\hat{y}_k(x_n, w))\} = -(0.82 + 0.27 + 0.31) = -1.40$

Cross-Entropy: $E(w) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln \hat{y}_k(x_n, w) = -0.48$

NLL Loss < Cross-Entropy Loss as the latter adds additional constraint to the prediction model

Single-Label vs. Multi-Label

Multi-Class, Single-Label Classification

- An example may be a member of only one class.
- Constraint that classes are mutually exclusive is valuable if you want to treat predictions as probabilities.

Multi-Class, Multi-Label Classification:

- An example may be a member of more than one class.
- No additional constraints on class membership to exploit.
- One logistic regression loss for each possible class.

Summary: Output Activation and Error Functions

- Regression:
  - Identity function: $y_k = a_k$
  - Sum of square error: $E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\|\boldsymbol{y}(\boldsymbol{x}_n, \boldsymbol{w}) - \boldsymbol{t}_n\|^2$

- Multiple independent binary classification:
  - Logistic function: $y_k = \sigma(a_k)$
  - Negative log of likelihood:

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K}\{t_{nk}\ln y_k(\boldsymbol{x}_n, \boldsymbol{w}) + (1 - t_{nk})\ln(1 - y_k(\boldsymbol{x}_n, \boldsymbol{w}))\}$$

- Multi-class classification:
  - Soft-max function: $y_k = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$
  - Cross-entropy: $E(\boldsymbol{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\ln y_k(\boldsymbol{x}_n, \boldsymbol{w})$

- In sum, there is a nature choice of both output unit activation function and matching error function

Agenda

1. Why deep learning
2. The perceptron
3. Activation functions
4. Feed forward process
5. From perceptron to deep neural networks
6. Multi-class Neural Networks
7. Backpropagation
8. Regularization: dropout and early stopping

Loss Minimization towards Model Optimization

- We want to find the network weights that achieve the lowest loss

$$W^* = \underset{W}{argmin} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; W\right), y^{(i)}\right)$$

$$W^* = argmin\, J(W), \text{where } W = \left\{W^{(0)}, W^{(1)}, \ldots\right\}$$

- Gradient, $\nabla E_n(W) = \frac{\partial J(W)}{\partial W}$

Gradient Descent Algorithm

Step 1. Initialize weights randomly $\sim N(0, \sigma^2)$

Step 2. Loop steps 3 and 4 until convergence:

Stochastic:

Step 3. Compute gradient for each training example *i*

$$\nabla E_n(W) = \frac{\partial J(W)}{\partial W}$$

Step 4. Update weights,

$$W \leftarrow W - \eta \nabla E_n(W)$$

Batch (mini-batch):

Step 3. Compute the gradient for each training example in a batch

$$\nabla E_N(W) = \sum_N \nabla E_n(W)$$

Step 4. Update weights,

$$W \leftarrow W - \eta \nabla E_N(W)$$

Step 5. Return weights.

Compute Gradients: Chain Rule

- How does a small change in one weight w affects the final $\text{loss}\, J(W)$?

Chain rule: to find the derivative of a composite function. If $f(u)$ is a differential function of u and $u=g(w)$ is a differentiable function of x, then $y=f(g(w))$ is a differentiable function of w

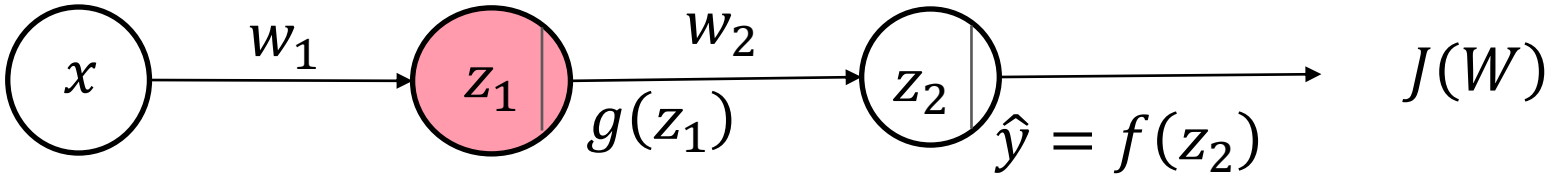
$f\big(g(w)\big)$ Inner function: $\mathrm{u} = g(w)$
Outer function: $h = f(u)$

$$\frac{\partial f\big(g(w)\big)}{\partial w} = \frac{\partial f}{\partial g} \times \frac{\partial g}{\partial w}$$

Compute Gradients: Backpropagation in Neural Network

- **Gradient** – The partial derivative of loss function *J*(W) with aspect to weight *W*
- **Backpropagation** -- Find the partial derivatives backwards

  from output layer → hidden layers till the layer where *W* first applied



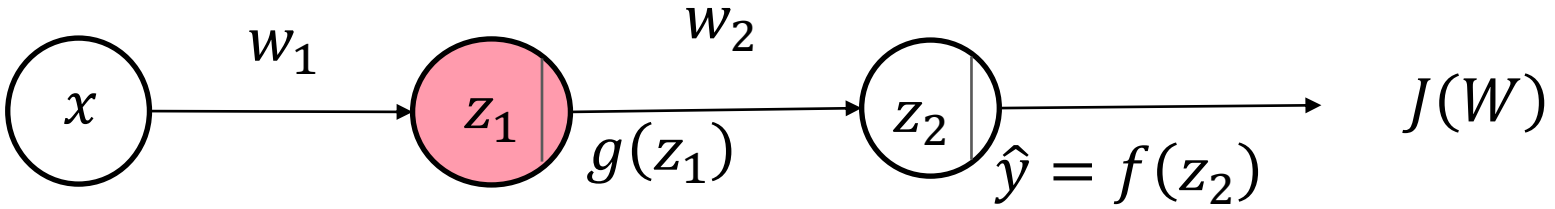
$$\frac{\partial J(W)}{\partial w_2} = \frac{\partial J(W)}{\partial z_2} \times \frac{\partial z_2}{\partial w_2}$$

$$z_1 = w_1 x$$

$$z_2 = w_2 g(z_1)$$

If $\hat{y} = f(z_2)$, $\frac{\partial J(W)}{\partial z_2} = \frac{\partial J(W)}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_2}$

Compute Gradients: Backpropagation in Neural Network



$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial z_2} \times \frac{\partial z_2}{\partial w_1}$$

$$= \frac{\partial J(W)}{\partial z_2} \times \frac{\partial z_2}{\partial g(z_1)} \times \frac{\partial g(z_1)}{w_1}$$

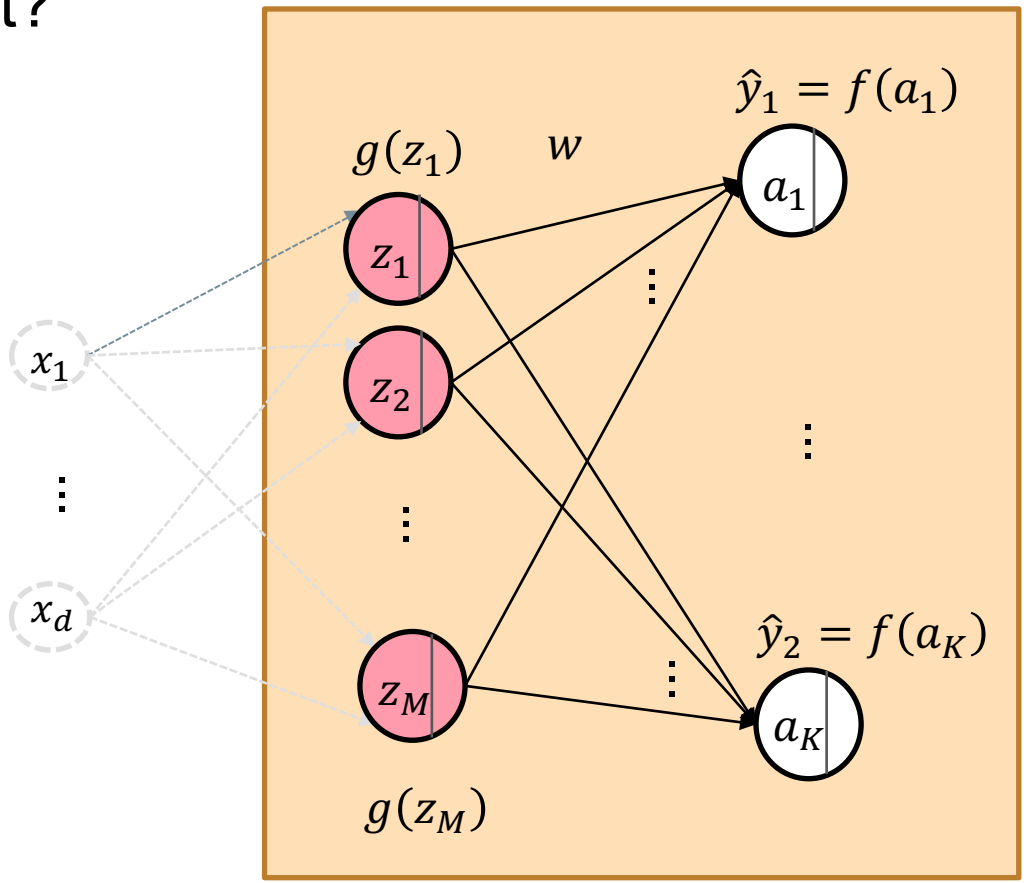
$$= \frac{\partial J(W)}{\partial z_2} \times \frac{\partial z_2}{\partial g(z_1)} \times \frac{\partial g(z_1)}{\partial z_1} \times \frac{\partial z_1}{\partial w_1}$$

$$z_1 = w_1 x$$

$$z_2 = w_2 g(z_1)$$

If $\hat{y} = f(z_2)$, $\frac{\partial J(W)}{\partial z_2} = \frac{\partial J(W)}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_2}$

Error function and its gradient in neural networks with different output?



Errors for Output Unit for Regression

- Identification function: $y_k = a_k$

$a_k = \sum_i w_i z_i$, *where* z_i could be activation of a hidden unit or input; w_i is the weight of z_i at hidden layer j; y_k is one of the K output.

Sum of square error: $E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N} \|\boldsymbol{y}(\boldsymbol{x}_n, \boldsymbol{w}) - \boldsymbol{t}_n\|^2$

$$\frac{dE_n}{dw} = \frac{dE_n}{da_k} \times \frac{da_k}{dw} = \delta_k \times z$$

Omit n for brevity

$$\delta_k = \frac{dE_n}{da_k} = \frac{dE_n}{dy_k}\frac{dy_k}{da_k} = y_k - t_k$$

Note: $E_n = \frac{1}{2}(y_k - t_k)^2$

$$\frac{dE_n}{dw} = (y_k - t_k)z$$

Errors for Output Unit for Multiple Independent Binary Classification

- Logistic function: $y_k = \sigma(a_k)$

$a_k = \sum_i w_i z_i$, *where* z_i could be activation of a hidden unit or input; w_i is the weight of z_i at hidden layer j; y_k is one of the K output.

- Negative log of likelihood

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K}\{t_{nk}\ln y_k(\boldsymbol{x}_n, \boldsymbol{w}) + (1-t_{nk})\ln(1-y_k(\boldsymbol{x}_n, \boldsymbol{w}))\}$$

$$\frac{dE_n}{dw} = (y_k - t_k)z$$

$$\frac{dE_n}{dw} = \frac{dE_n}{da_k} \times \frac{da_k}{dw} = \delta_k \times z$$

$$\delta_k = \frac{dE_n}{da_k} = \frac{dE_n}{dy_k}\frac{dy_k}{da_k} = \left(\frac{1-t_k}{1-y_k} - \frac{t_k}{y_k}\right) y_k(1-y_k) = y_k - t_k$$

Note: $E_n = -[t_k \ln y_k + (1-t_k)\ln(1-y_k)]$

- Omit *n* for brevity

Quotient rule: if $f(x) = \frac{g(x)}{h(x)}$, then $f'(x) = \frac{g'(x)h(x) - g(x)h'(x)}{[h(x)]^2}$

Errors for Output Unit for Multi-Class Classification

Softmax function: $y_k = \frac{exp(a_k)}{\sum_{k=1}^{K} exp(a_k)}$,

$a_k = \sum_i w_i z_i$, *where* z_i could be activation of a hidden unit or input; w_i is the weight of z_i at hidden layer j; y_k is one of the K output.

$$\frac{\partial f}{\partial w_i} = \sum_{j=1}^{n} \frac{\partial f}{\partial g^{(j)}} \times \frac{\partial g^{(j)}}{\partial w_i}$$

- Cross Entropy $E(w) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln y_k(x_n, w)$

$$\frac{dE_n}{dw} = \frac{dE_n}{da_k} \times \frac{da_k}{dw} = \delta_k \times z$$

$$\frac{dE_n}{dw} = (y_k - t_k)z$$

- Omit n for brevity

$$\delta_k = \frac{dE_n}{da_k} = \sum_m \frac{dE_n}{dy_m} \frac{dy_m}{da_k} = \sum_m -\frac{t_m}{y_m} \frac{dy_m}{da_k} = -t_k(1 - y_k) + \sum_{m \neq k} t_m y_k = y_k - t_k$$

Note: $E_n = -\sum_{k=1}^{K} t_k y_k$ and $\sum_{k=1}^{K} t_k = 1$

$$\frac{dy_m}{da_k} = \begin{cases} -y_k y_m & if\ m \neq k \\ y_k(1 - y_k) & if\ m = k \end{cases}$$

a_k is included in each of the K outputs, $y_m, m = 1, \ldots, K$.

Agenda

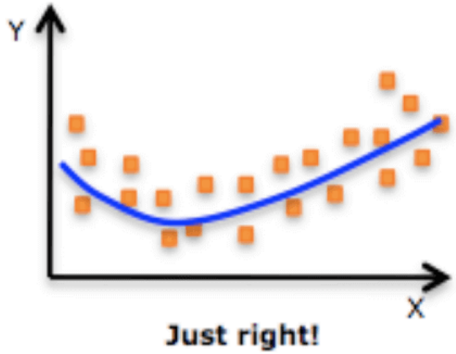
1. Why deep learning
2. The perceptron
3. Activation functions
4. Feed forward process
5. From perceptron to deep neural networks
6. Multi-class Neural Networks
7. Backpropagation
8. Regularization: dropout and early stopping

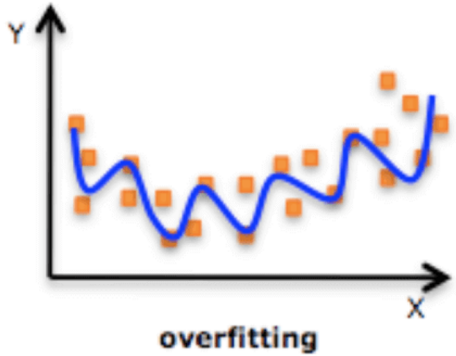
Normalizing Feature Values

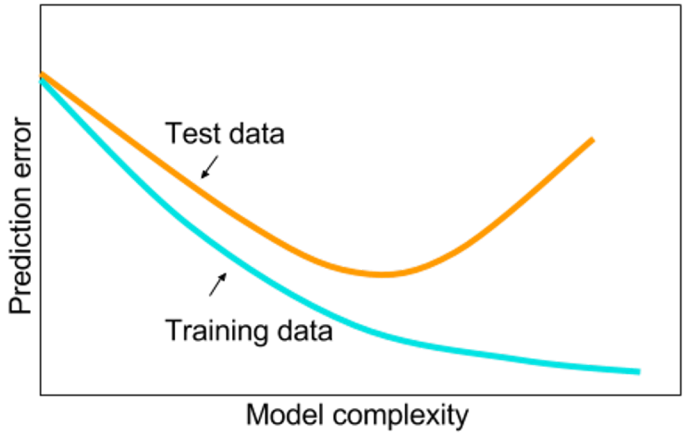
- Features with reasonable scales
  - Roughly zero-centered, [-1, 1]
  - Avoid NaN trap
  - Avoiding outlier values
- Standard methods:
  - Linear scaling
  - Hard cap (clipping) to max, min
  - Log scaling

Overfitting

Overfitting refers to a model that models the training data too well, which negatively impact the models ability to generalize







Neural Network with a deep structure easily get overfitted

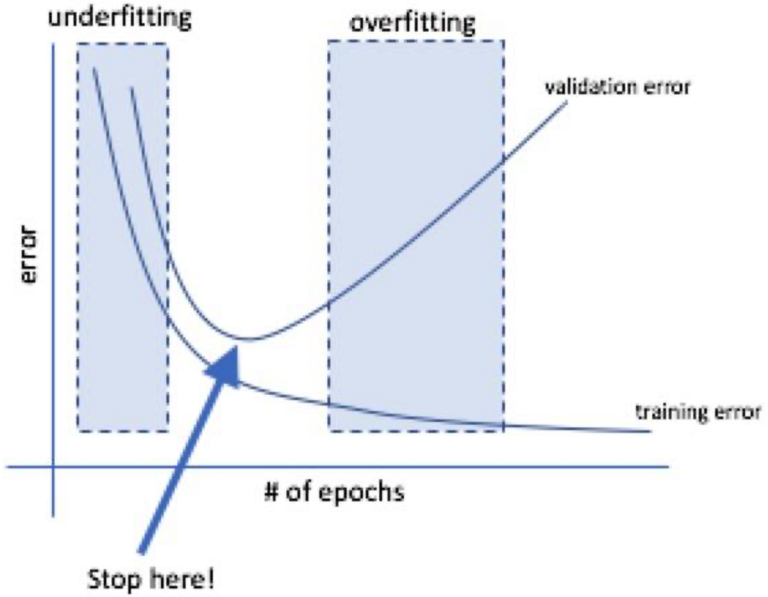
Solutions of Overfitting NN

- Early Stopping
- Parameters Regularization
- Dropout
- Train with more data or remove features

Early Stopping

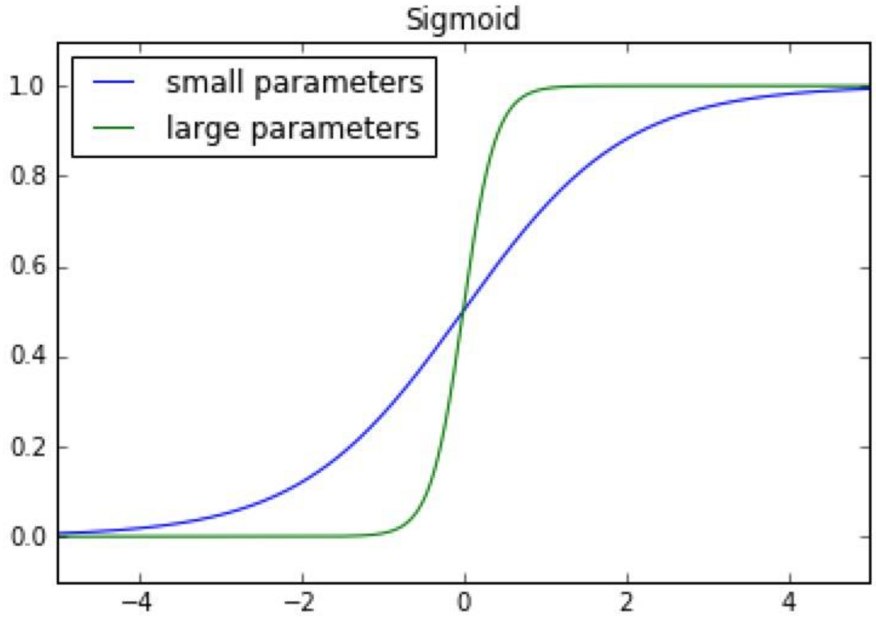
1. Watch the validation curve

1. Stop updating the weights once validation error starts increasing



Regularization

- In NN, inputs are linearly combined with parameters. Therefore, large parameters can amplify small changes in the input.
- Large parameters may arbitrarily increases the confidence in our predictions.
- In cost function, add regularization term to penalize parameters
- The core idea is to prevent parameters from becoming excessively large during training



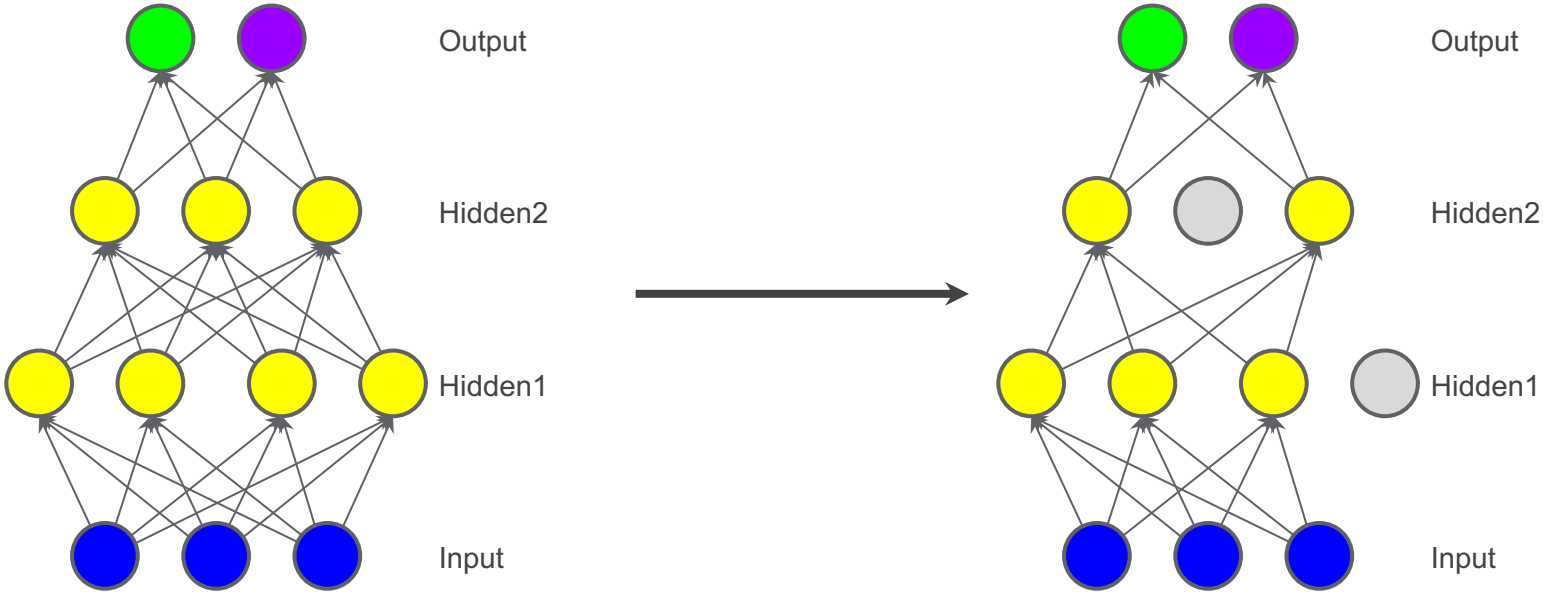
$$J = \frac{1}{N}\sum_{i=1}^{N} \ell(f_\theta(x_i), y_i) + \lambda g(\theta)$$

Control the degree to which we select to penalize large parameters

Dropout

- Works by randomly “dropping out” units in a network for a single gradient step
  - They come back with unchanged weights at the next step
- The more you drop out, the stronger the regularization
  - 0.0 = no dropout regularization
  - 1.0 = drop everything out! Learns nothing
  - Intermediate values more useful

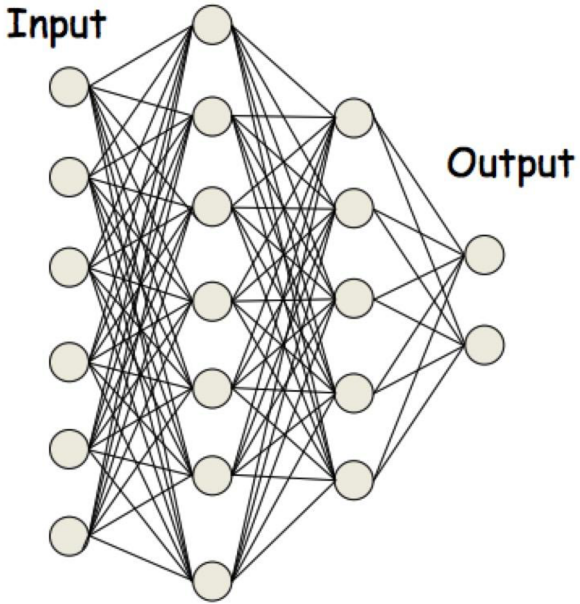
Dropout Regularization Illustrated

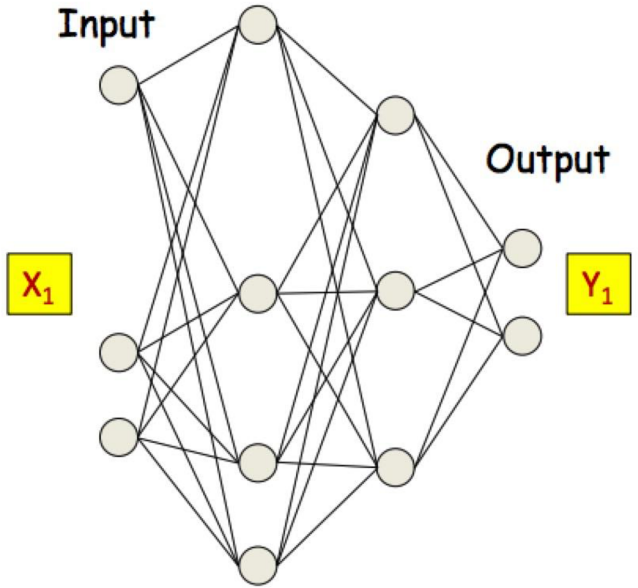


Fully Connected NN

NN with Dropout Regularization

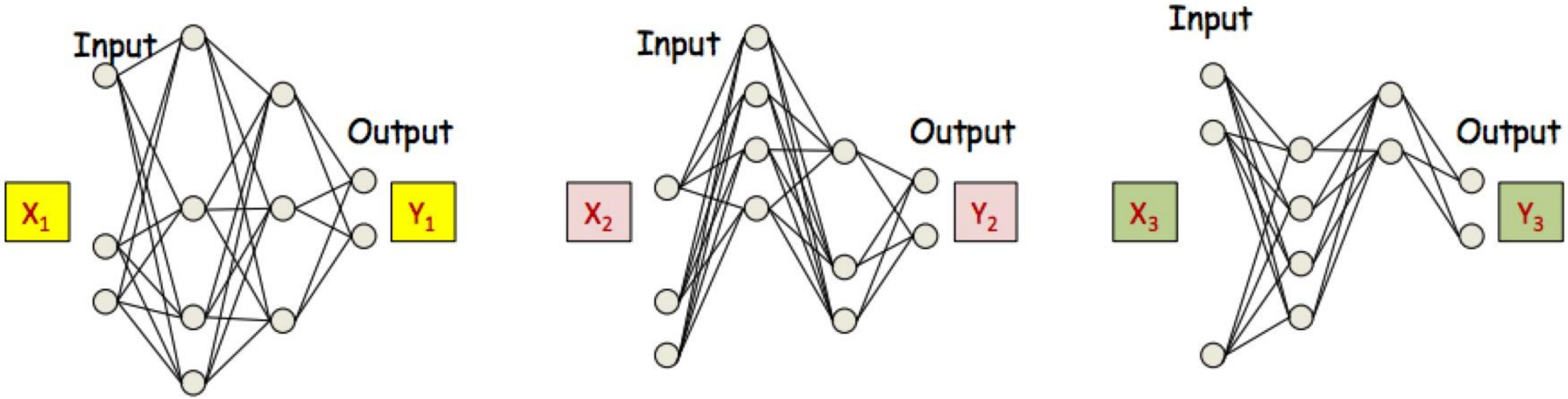
Dropout Regularization Illustrated





- During training: for each input, at each iteration, “turn off” each neuron with a probability 1-α

Dropout Regularization Illustrated

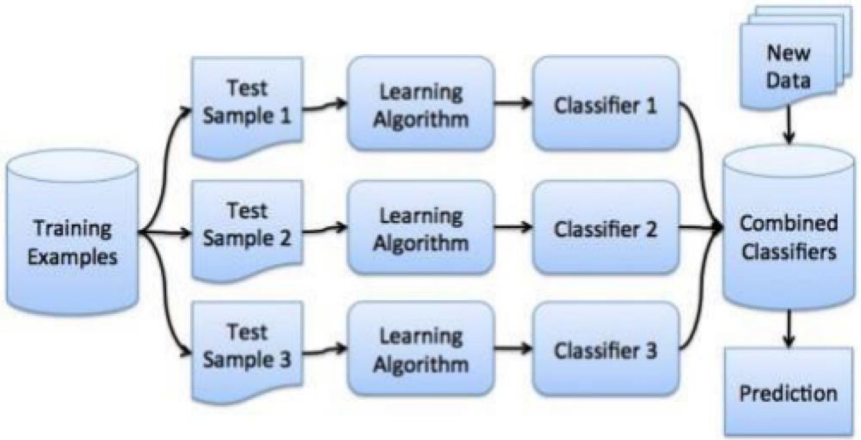


- Due to randomness, the drop nodes changes for each input
- BP is effectively performed only over the remaining network
- For closed nodes, the gradients are zero

Understanding Dropout from another angle

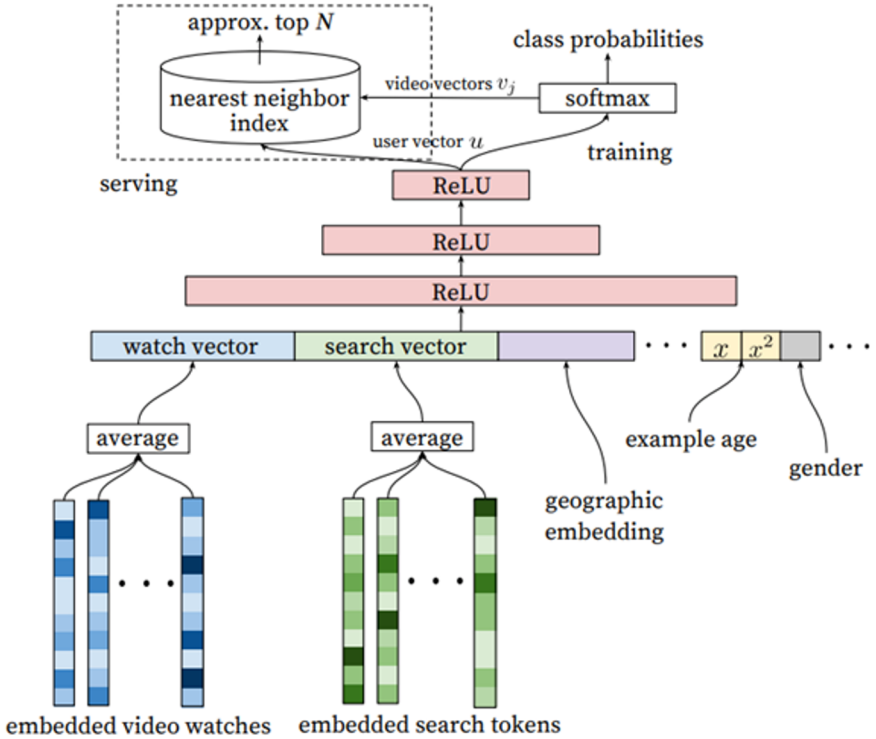
Similar to **Bagging**

- Sample training data and train several different classifiers
- Classify test instance with entire ensemble of classifiers
- Vote across classifiers for final decision



Real Life Applications

Youtube Recommendation



Source: P. Covington, et al., "Deep Neural Networks for YouTube Recommendations", 2016

Word to Vector