COMP9444: Neural Networks and Deep Learning

Week 3b. Hidden Unit Dynamics

Alan Blair

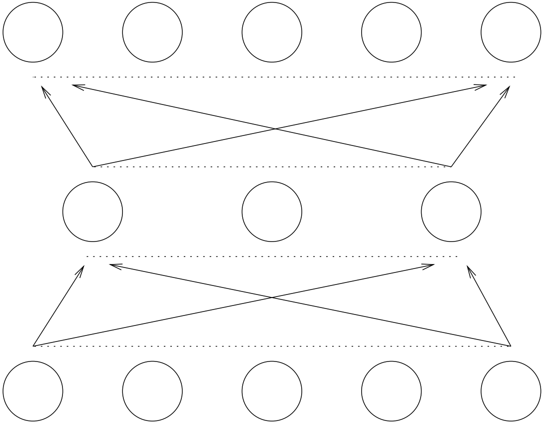
School of Computer Science and Engineering

September 23, 2024

Outline

- geometry of hidden unit activations
- limitations of 2-layer networks
- vanishing / exploding gradients
- alternative activation functions
- ways to avoid overfitting in neural networks

Encoder Networks



Inputs	Outputs
10000	10000
01000	01000
00100	00100
00010	00010
00001	00001

- identity mapping through a bottleneck
- also called N–M–N task
- used to investigate hidden unit representations

N–2–N Encoder

Hidden Unit Space:

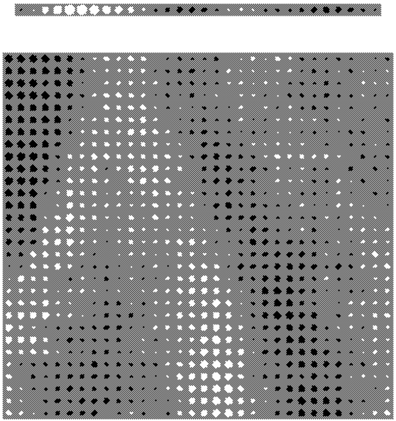
8–3–8 Encoder

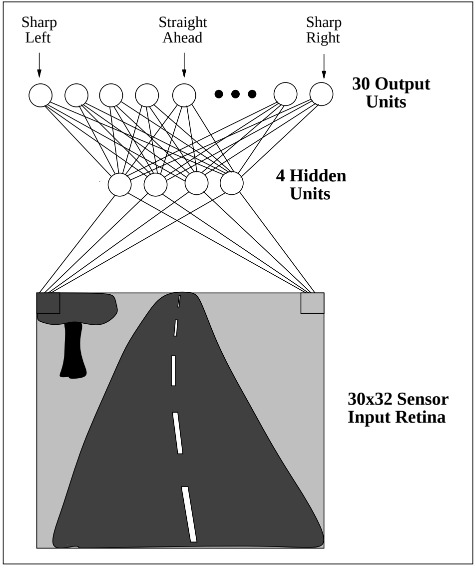
Exercise:

- Draw the hidden unit space for 2-2-2, 3-2-3, 4-2-4 and 5-2-5 encoders.
- Represent the input-to-hidden weights for each input unit by a point, and the hidden-to-output weights for each output unit by a line.
- Now consider the 8-3-8 encoder with its 3-dimensional hidden unit space.
  - what shape would be formed by the 8 points representing the input-to-hidden weights for the 8 input units?
  - what shape would be formed by the planes representing the hidden-to-output weights for each output unit?

Hint: think of two platonic solids, which are “dual” to each other.

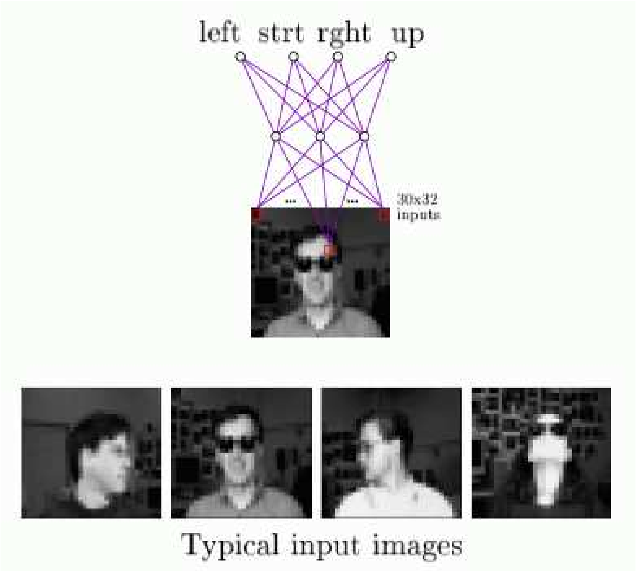
Hinton Diagrams



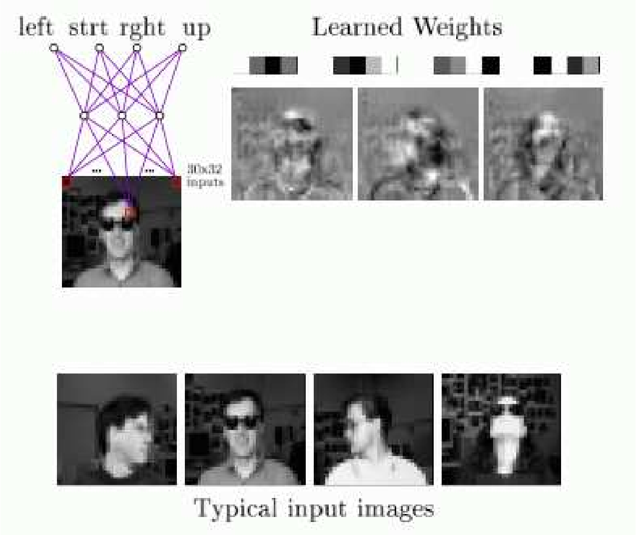


- used to visualize higher dimensions
- white = positive, black = negative

Learning Face Direction



Learning Face Direction



Weight Space Symmetry

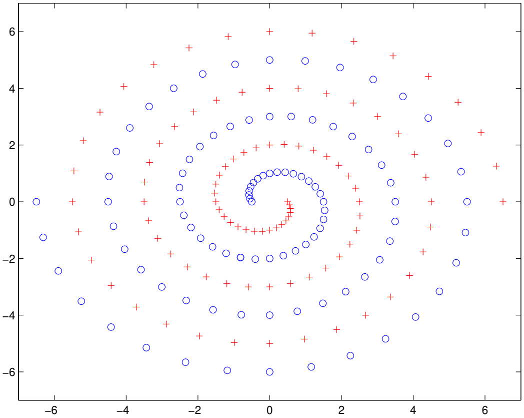
- swap any pair of hidden nodes, overall function will be the same
- on any hidden node, reverse the sign of all incoming and outgoing weights (assuming symmetric transfer function)
- hidden nodes with identical input-to-hidden weights in theory would never separate; so, they all have to begin with different random weights
- in practice, all hidden nodes may try to do similar job at first, then gradually specialize.

Controlled Nonlinearity

- for small weights, each layer implements an approximately linear function, so multiple layers also implement an approximately linear function.
- for large weights, transfer function approximates a step function, so computation becomes digital and learning becomes very slow.
- with typical weight values, two-layer neural network implements a function which is close to linear, but takes advantage of a limited degree of nonlinearity.

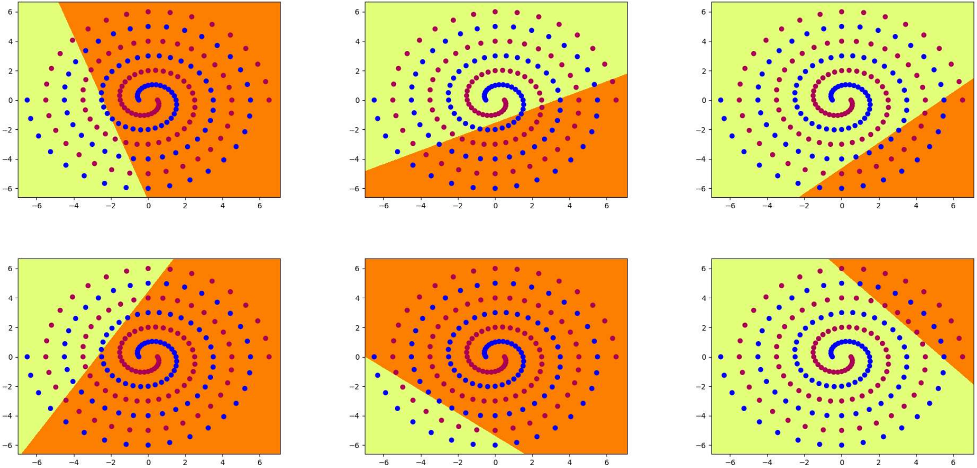
Limitations of Two-Layer Neural Networks

Some functions are difficult for a 2-layer network to learn.

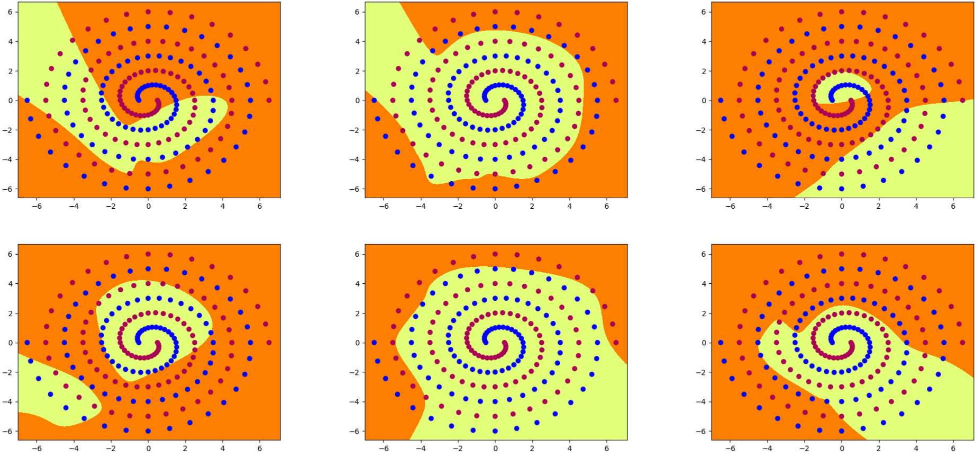


For example, this Twin Spirals problem is difficult to learn with a 2-layer network, but it can be learned using a 3-layer network.

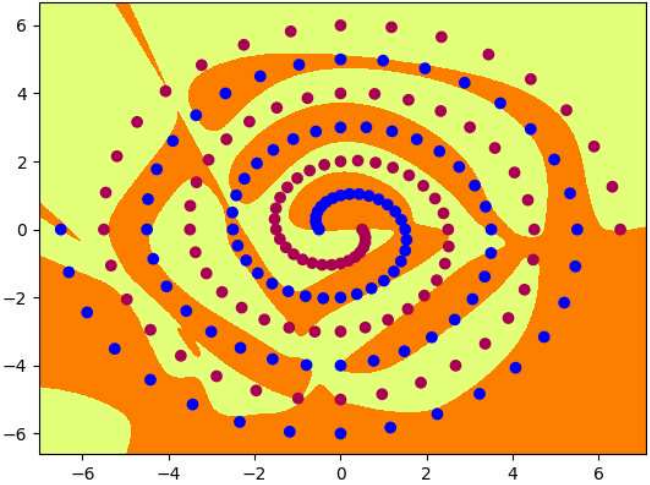
First Hidden Layer



Second Hidden Layer



Network Output



Adding Hidden Layers

- ➢ twin spirals can be learned by 3-layer network
- ➢ first hidden layer learns linearly separable features
- ➢ second hidden layer combines these to produce more complex features
- ➢ learning rate and initial weight values must be small
- ➢ learning can be improved using the Adam optimizer

Vanishing / Exploding Gradients

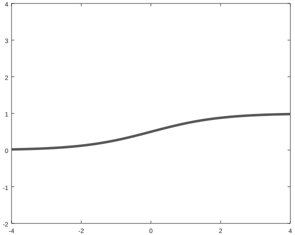
- training by backpropagation in networks with many layers is difficult
- when the weights are small, the differentials become smaller and smaller as we backpropagate through the layers, and end up having no effect
- when the weights are large, the activations in the higher layers may saturate to extreme values
- when the weights are large, the differentials may sometimes get multiplied twice in succession in places where the transfer function is steep, causing them to blow up to large values

Vanishing / Exploding Gradients

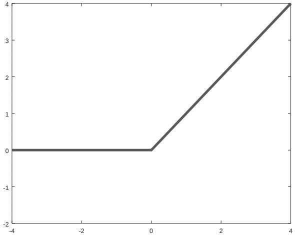
Ways to avoid vanishing / exploding gradients:

- new activations functions
- weight initialization (Week 4)
- batch normalization (Week 4)
- skip connections (Week 4)
- long short term memory (LSTM) (Week 5)

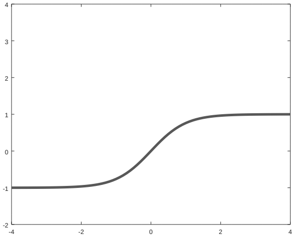
Activation Functions



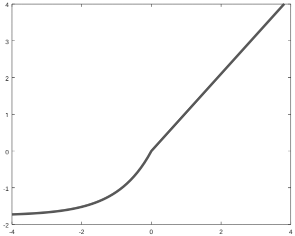
Sigmoid



Rectified Linear Unit (ReLU)



Hyperbolic Tangent



Scaled Exponential Linear Unit (SELU)

Activation Functions

- sigmoid and hyperbolic tangent traditionally used for 2-layer networks, but suffer from vanishing gradient problem in deeper networks.
- rectified linear units (ReLUs) are popular for deep networks (including convolutional networks); gradients will not vanish because derivative is either 0 or 1.