COMP9444
Neural Networks and Deep Learning

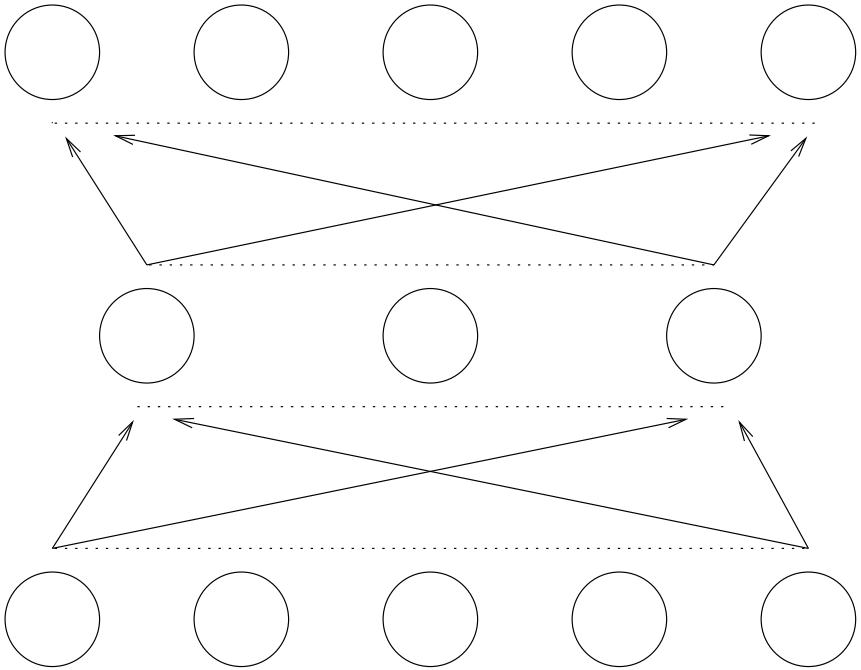
3a. Hidden Unit Dynamics

Textbook, Sections 5.2-5.3, 6.3, 7.11-7.12, 8.2

Outline

- geometry of hidden unit activations (8.2)
- limitations of 2-layer networks
- vanishing/exploding gradients
- alternative activation functions (6.3)
- ways to avoid overfitting in neural networks (5.2-5.3)
- dropout (7.11-7.12)

Encoder Networks

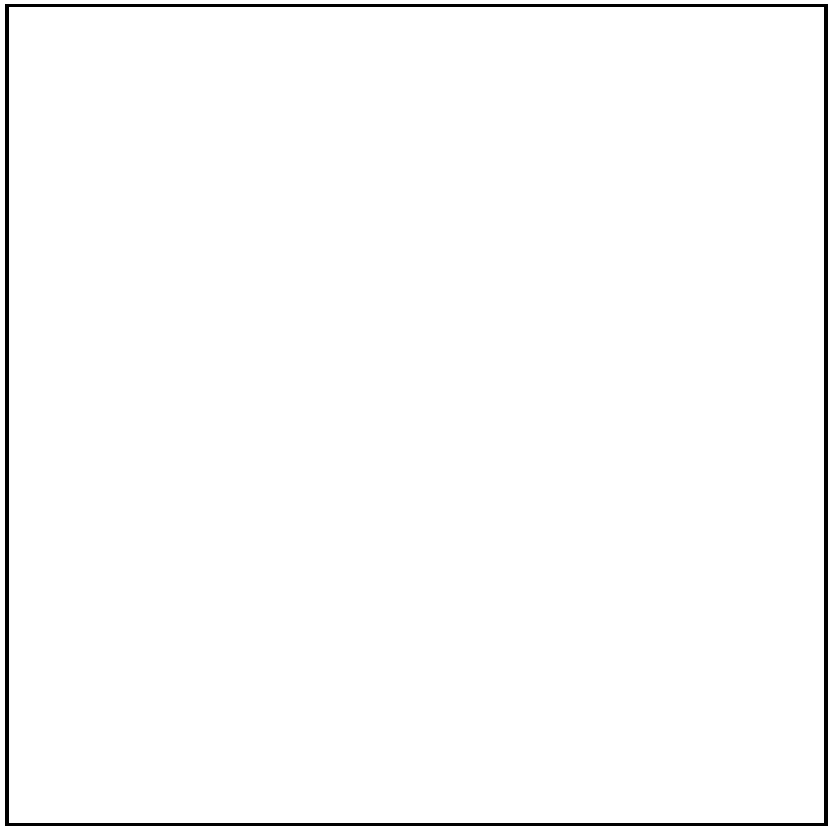


Inputs	Outputs
10000	10000
01000	01000
00100	00100
00010	00010
00001	00001

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

N–2–N Encoder

HU Space:



8–3–8 Encoder

Exercise:

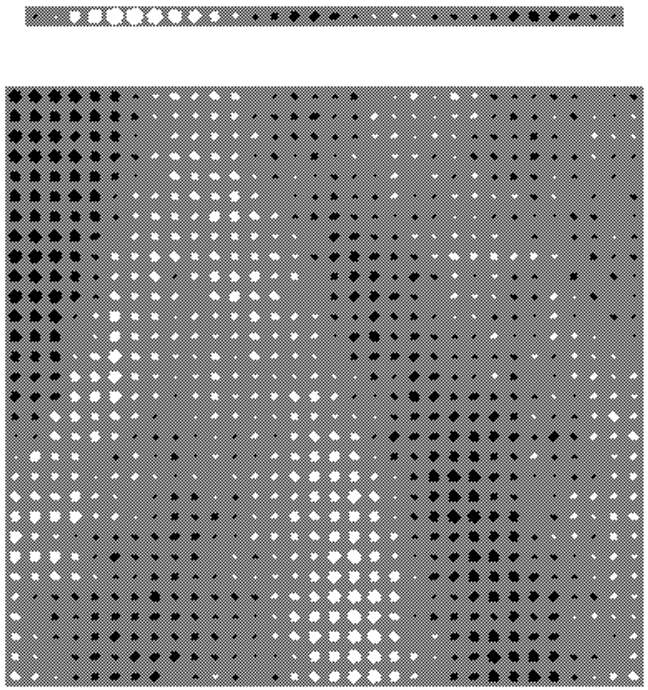
Draw the hidden unit space for 2-2-2, 3-2-3, 4-2-4 and 5-2-5 encoders.

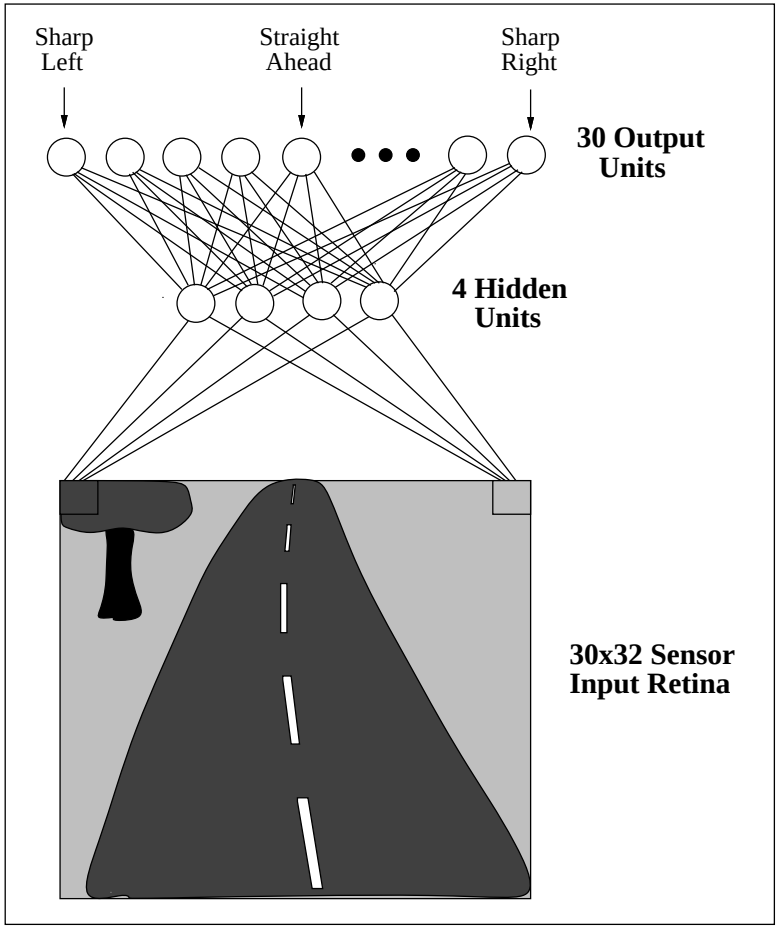
Represent the input-to-hidden weights for each input unit by a point, and the hidden-to-output weights for each output unit by a line.

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.

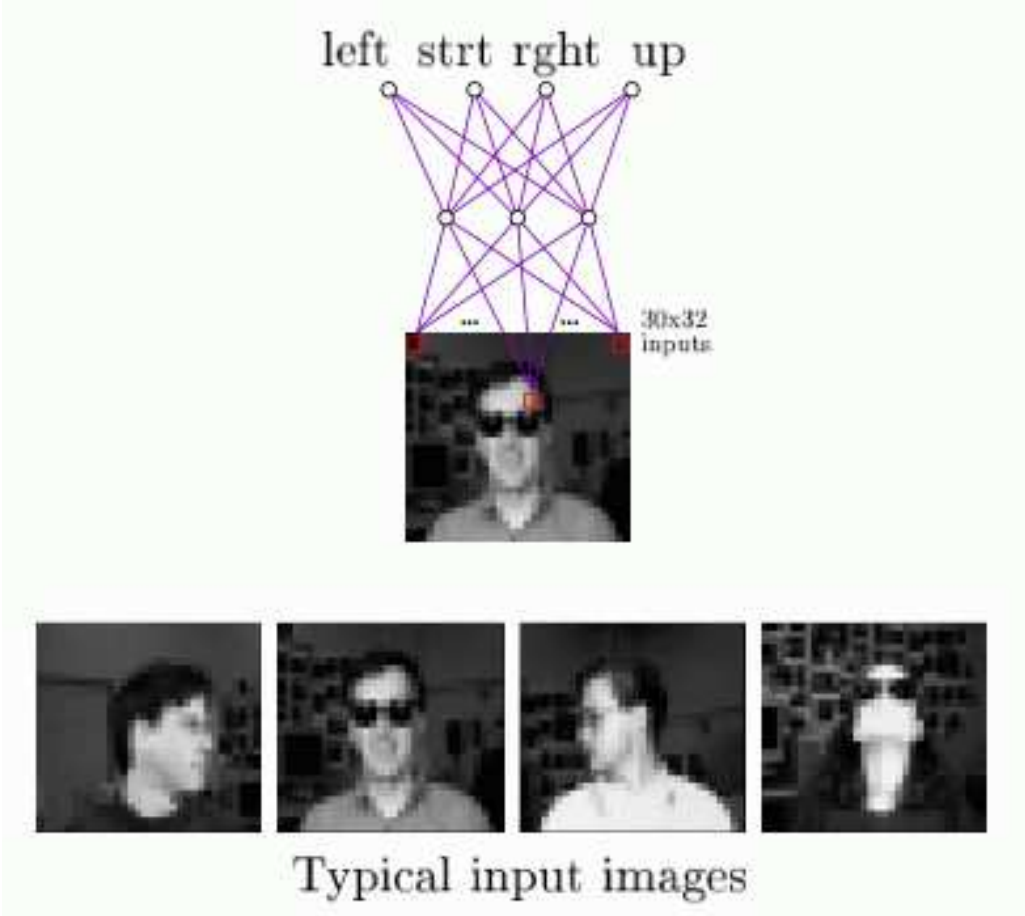
Hinton Diagrams



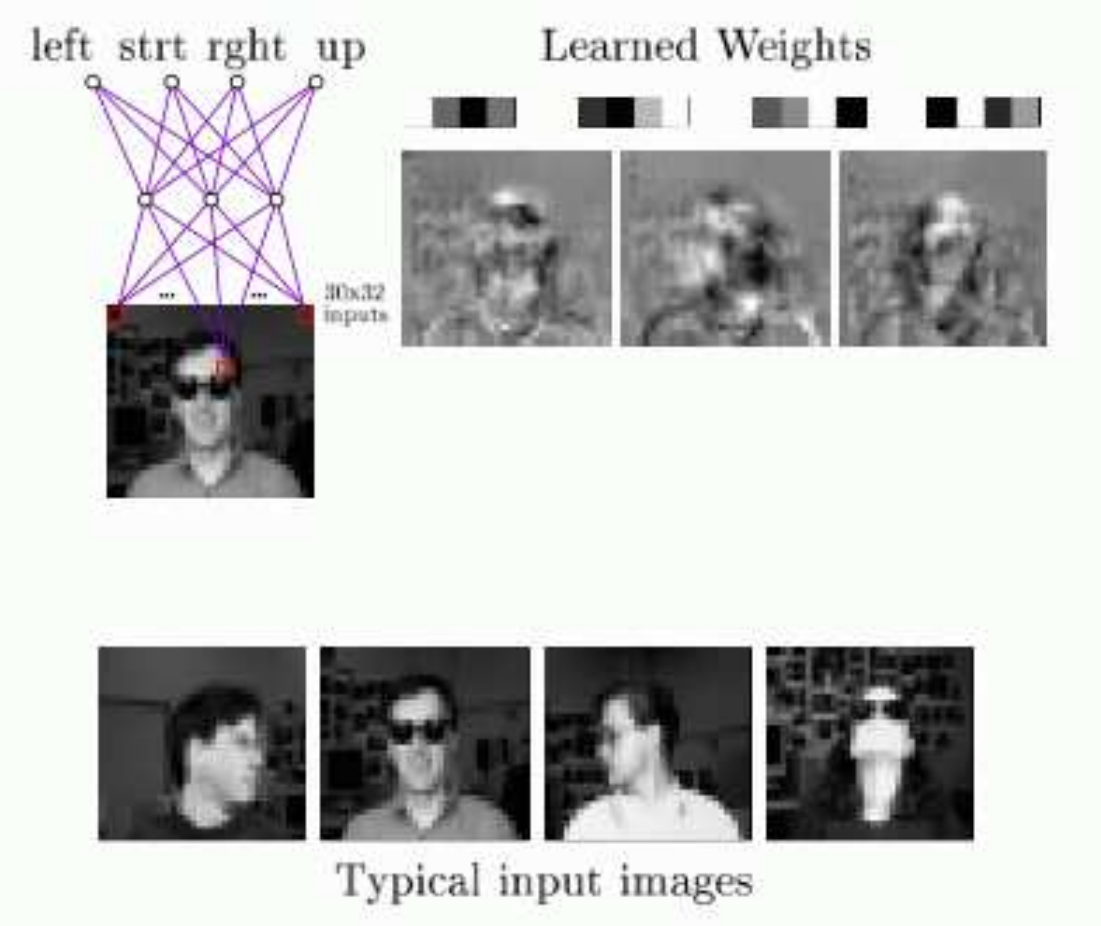


- used to visualize higher dimensions
- white = positive, black = negative

Learning Face Direction



Learning Face Direction



Weight Space Symmetry (8.2)

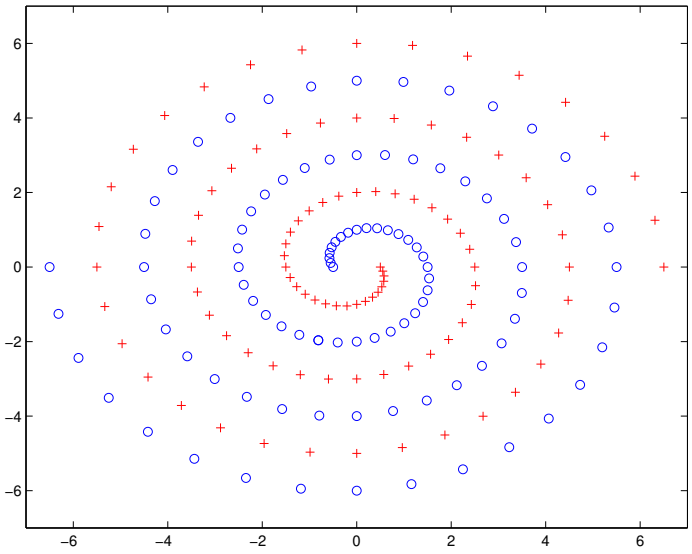
- swap any pair of hidden nodes, overall function will be the same
- on any hidden node, reverse the sign of all incoming and outgoing weights (assuming symmetric transfer function)
- hidden nodes with identical input-to-hidden weights in theory would never separate; so, they all have to begin with different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

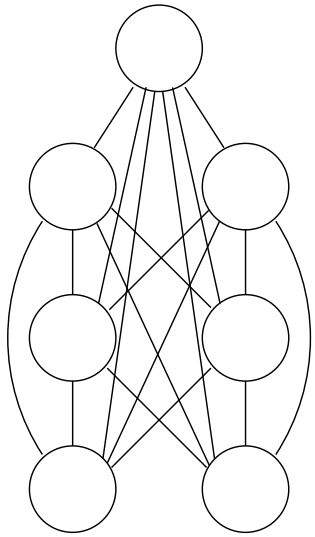
Controlled Nonlinearity

- 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 cannot be learned with a 2-layer sigmoidal network.





For example, this Twin Spirals problem cannot be learned with a 2-layer network, but it can be learned using a 3-layer network if we include shortcut connections between non-consecutive layers.

Adding Hidden Layers

- Twin Spirals can be learned by 3-layer network with shortcut connections
- first hidden layer learns linearly separable features
- second hidden layer learns “convex” features
- output layer combines these to produce “concave” features
- training the 3-layer network is delicate
- learning rate and initial weight values must be very small
- otherwise, the network will converge to a local optimum

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 will saturate to extreme values. As a result, the gradients at those layers will become very small, and will not be propagated to the earlier layers.

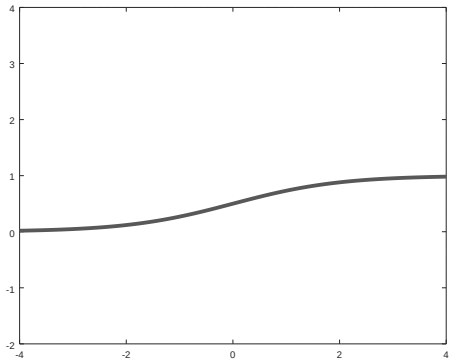
When the weights have intermediate values, the differentials will sometimes get multiplied many times is places where the transfer function is steep, causing them to blow up to large values.

Vanishing / Exploding Gradients

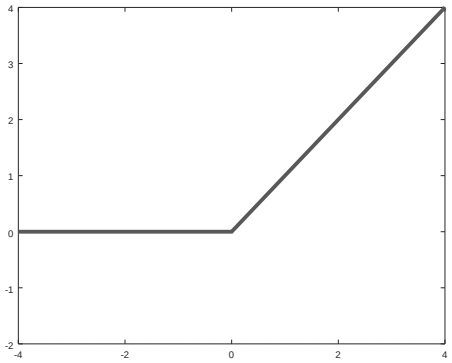
Ways to avoid vanishing/exploding gradients:

- new activations functions
- weight initialization (Week 5)
- batch normalization (Week 5)
- long short term memory (LSTM) (Week 6)
- layerwise unsupervised pre-training (Week 9)

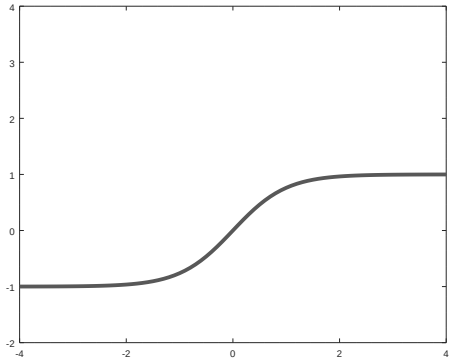
Activation Functions (6.3)



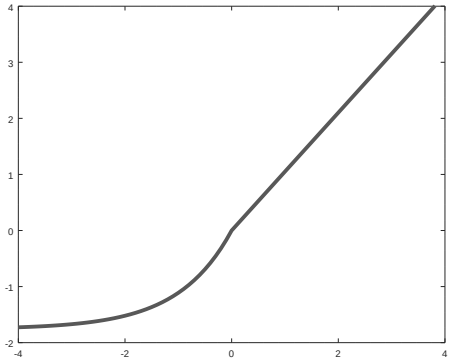
Sigmoid



Rectified Linear Unit (ReLU)



Hyperbolic Tangent



Scaled Exponential Linear Unit (SELU)

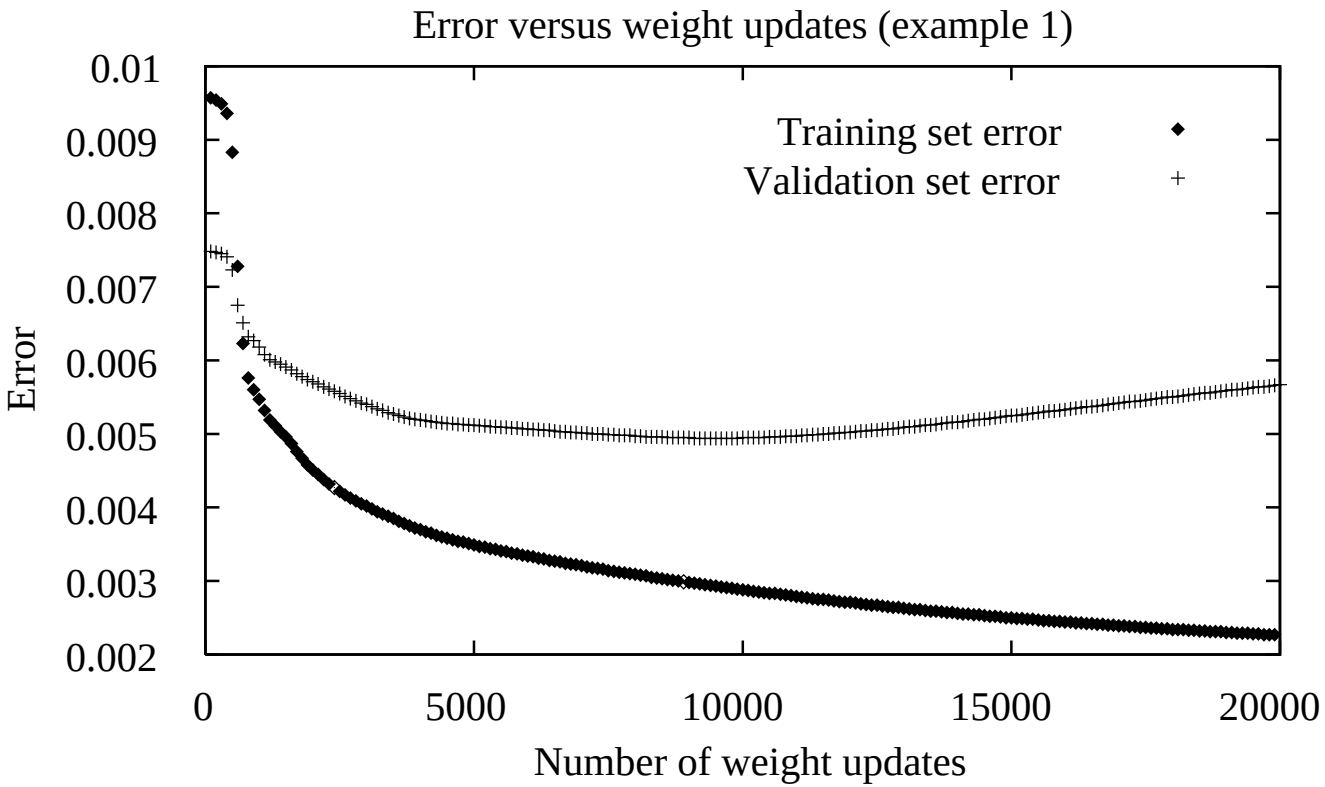
Activation Functions

- 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 don't vanish.
  But, their highly linear nature may cause other problems.
- Scaled Exponential Linear Units (SELUs) are a recent innovation which seems to work well for very deep networks.

Ways to Avoid Overfitting in Neural Netorks

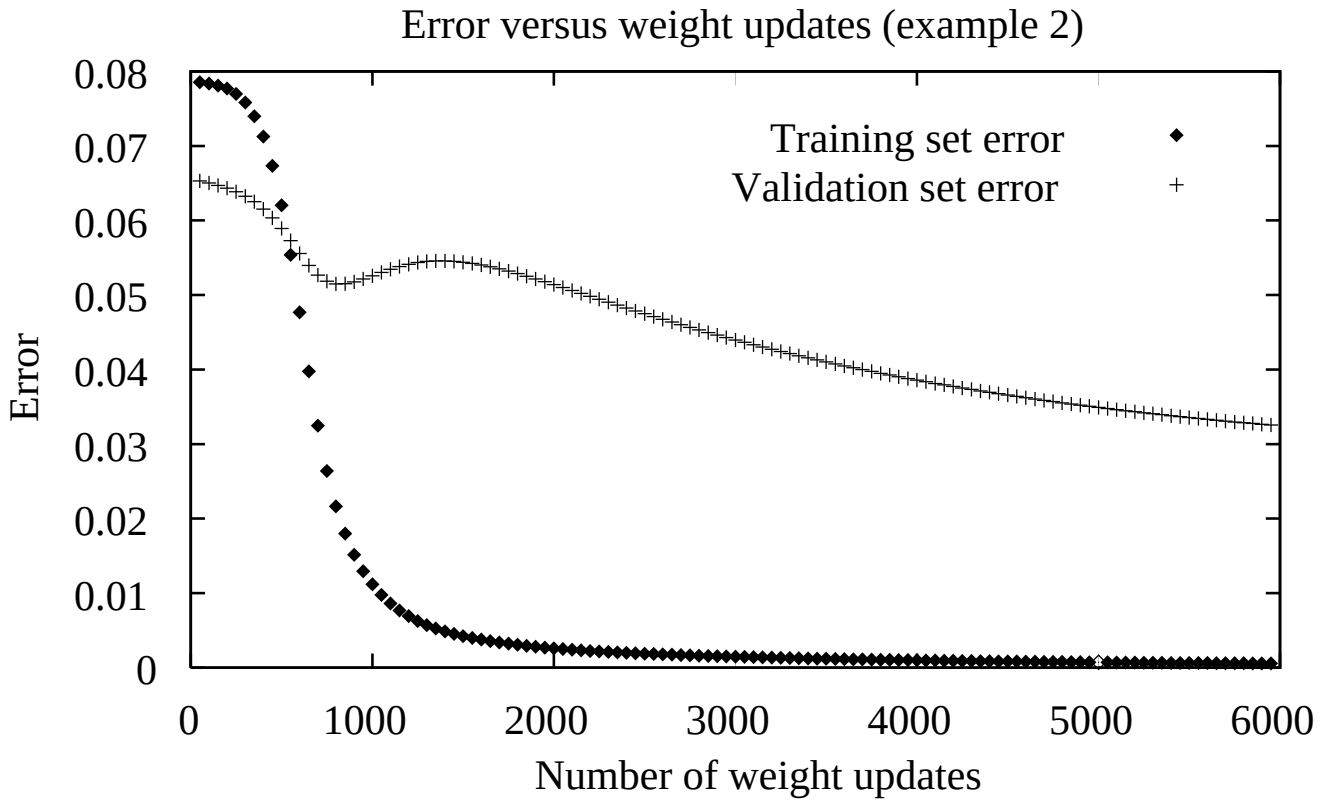
- Limit the number of hidden nodes or connections
- Limit the number of training epochs (weight updates)
- Weight Decay
- Dropout

Training, Validation and Test Error



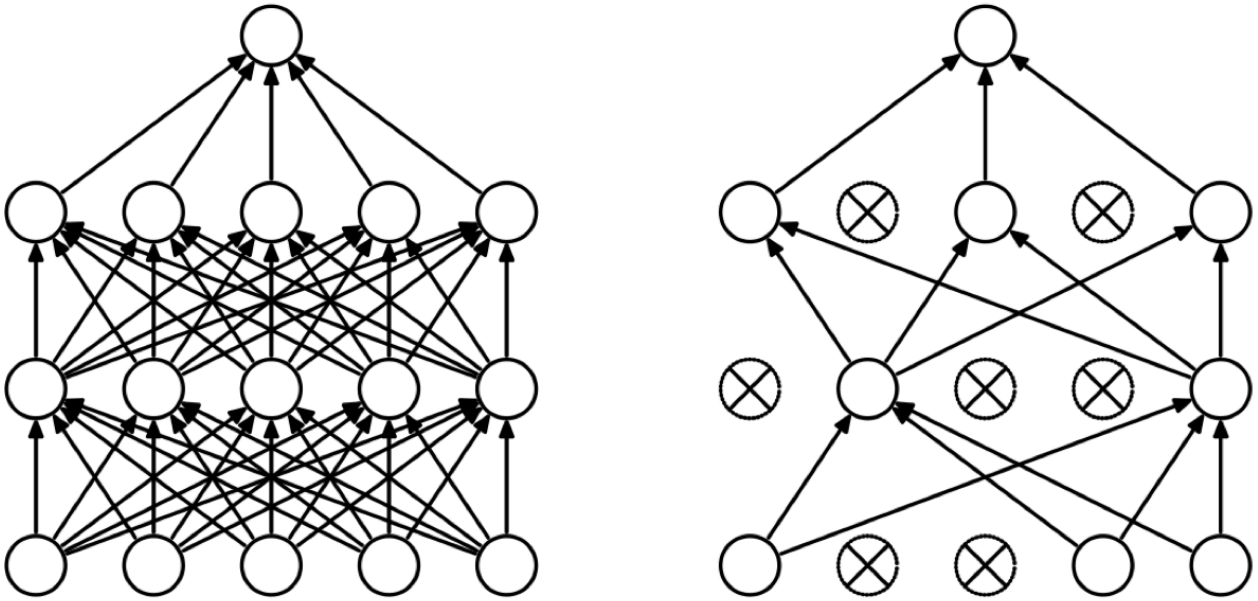
Choose number of hidden nodes, or number of weight updates, to minimize validation set error, and it will likely also perform well on the test set.

Overfitting in Neural Networks



In this example, the validation set error is much larger than the training set error; but, it is still decreasing at epoch 6000.

Dropout (7.12)



(a) Standard Neural Net (b) After applying dropout.

For each minibatch, randomly choose a subset of nodes to not be used.
Each node is chosen with some fixed probability (usually, one half).

Dropout (7.12)

When training is finished and the network is deployed, all nodes are used, but their activations are multiplied by the same probability that was used in the dropout.

Thus, the activation received by each unit is the average value of what it would have received during training.

Dropout forces the network to achieve redundancy because it must deal with situations where some features are missing.

Another way to view dropout is that it implicitly (and efficiently) simulates an ensemble of different architectures.

Ensembling (7.11)

Ensembling is a method where a number of different classifiers are trained on the same task, and the final class is decided by "voting" among them.

In order to benefit from ensembling, we need to have diversity in the different classifiers.

For example, we could train three neural networks with different architectures, three Support Vector Machines with different dimensions and kernels, as well as two other classifiers, and ensemble all of them to produce a final result.
(Kaggle Competition entries are often done in this way).

Bagging

Diversity can also be achieved by training on different subsets of data.

Suppose we are given N training items.

Each time we train a new classifier, we choose N items from the training set with replacement. This means that some items will not be chosen, while others are chosen two or three times.

There will be diversity among the resulting classifiers because they have each been trained on a different subset of data. They can be ensembled to produce a more accurate result than a single classifier.

Dropout as an Implicit Ensemble

In the case of dropout, the same data are used each time but a different architecture is created by removing the nodes that are dropped.

The trick of multiplying the output of each node by the probability of dropout implicitly averages the output over all of these different models.