

# Machine Learning

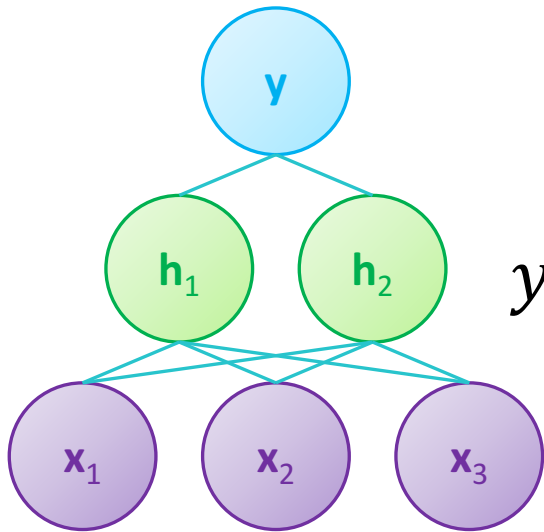
---

PATRICK HALL

DEPARTMENT OF DECISION SCIENCE

# What is a neural network?

Many neural networks are just sophisticated **regression models**



$$y = \tanh(w_{30} + w_{31} * \tanh(w_{10} + w_{11}x_1 + w_{12}x_2 + w_{13}x_3) + w_{32} * \tanh(w_{20} + w_{21}x_1 + w_{22}x_2 + w_{23}x_3))$$

Neural networks are similar to some biological mechanisms in the brain

# Neural Networks: Overview

---

- Conceptual Framework – a machine learning method that combines statistics and artificial intelligence
- Approach
  - Linear combination of inputs as derived features
  - Model the target as a nonlinear function of these features
- The name “neural networks” is derived from the fact that they were first developed as models for the human brain
  - Each hidden unit represents a neuron and the connections represents synapses
  - Recognized as a useful tool for nonlinear statistical modeling

# Neural Network

---

- A two-stage regression or classification model
  - Regression – typically  $K=1$  and only one  $Y_1$
  - $K$ -Classification –  $K$  units on top with  $K$  target measurements,  $Y_k$  for  $K=1$  to  $K$ , each being coded as a 0 or 1 variables for the  $k$ th class
- Derived Features:  $Z_m$ 
  - Hidden units or a basis expansion of the original input  $X$
  - Derived from linear combination of inputs and then the target,  $Y_k$ , is modeled as a function of linear combination of the  $Z_m$

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), \quad m = 1, \dots, M,$$

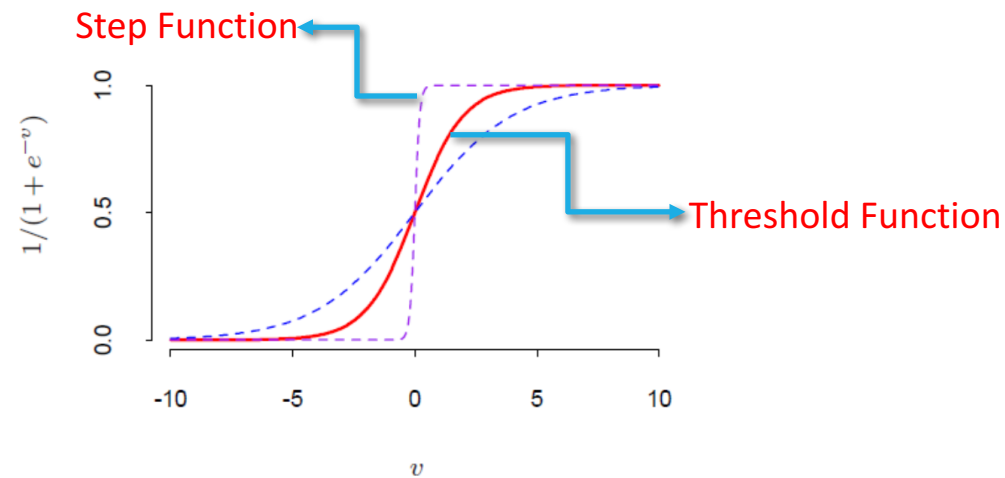
$$T_k = \beta_{0k} + \beta_k^T Z, \quad k = 1, \dots, K,$$

$$f_k(X) = g_k(T), \quad k = 1, \dots, K,$$

where  $Z = (Z_1, Z_2, \dots, Z_M)$ , and  $T = (T_1, T_2, \dots, T_K)$ .

# Neural Network

- Activation function
  - Typically chosen to be the sigmoid function:  $\sigma(v) = 1/(1 + e^{-v})$



**FIGURE 11.3.** Plot of the sigmoid function  $\sigma(v) = 1/(1 + \exp(-v))$  (red curve), commonly used in the hidden layer of a neural network. Included are  $\sigma(sv)$  for  $s = \frac{1}{2}$  (blue curve) and  $s = 10$  (purple curve). The scale parameter  $s$  controls the activation rate, and we can see that large  $s$  amounts to a hard activation at  $v = 0$ . Note that  $\sigma(s(v - v_0))$  shifts the activation threshold from 0 to  $v_0$ .

# Neural Networks: Pros & Cons

PROS	CONS
Few assumptions	Narrowly accepted
Can sometimes handle missing values	Narrowly understood
Can accurately model extremely nonlinear phenomena	Difficult to interpret
Tend to excel at image, sound, and other pattern recognition tasks	Tendency to overfit

# Modern Neural Networks: Multiple Layers

---

- Supervised neural networks with many layers achieve **state-of-the-art** results in **pattern recognition**
- Unsupervised neural networks with many layers are excellent **feature extractors** and **anomaly detectors**
- The bottom layer(s) of a deep network learn **optimal features**; top layers make predictions using these features
- More layers are usually **more efficient** than more neurons for attaining a given level of accuracy

# Contemporary Neural Networks

---

- Two important processes can differentiate training contemporary networks from training previous generations of neural networks
  - Unsupervised pre-training
  - Stacking many layers of hidden units



# Unsupervised Training

---

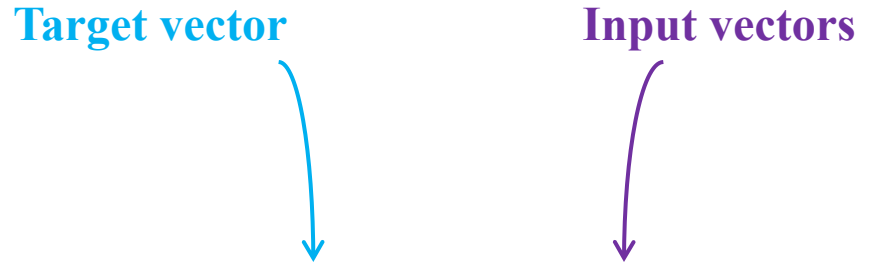
- Why is unsupervised training of neural networks useful?
  - Feature extraction and anomaly detection
  - Initialization
- How can a neural network be unsupervised?
  - Usually you try to predict a target vector  $\mathbf{Y}$  from input vector  $\mathbf{X}$
  - In unsupervised training, you try to predict  $\mathbf{X}$  from  $\mathbf{X}$

# Unsupervised Training

---

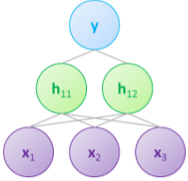
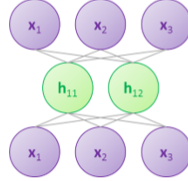
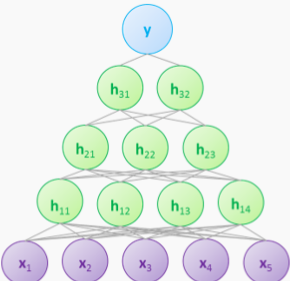
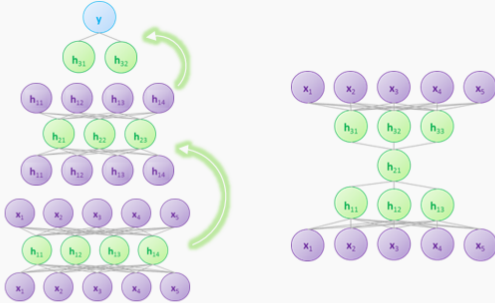
Target vector

Input vectors



$y$	$x_1$	$x_2$	$x_3$
1	2.54	1.65	0.02
0	1.14	0.70	0.82
1	0.99	0.51	2.11
$\vdots$	$\vdots$	$\vdots$	$\vdots$

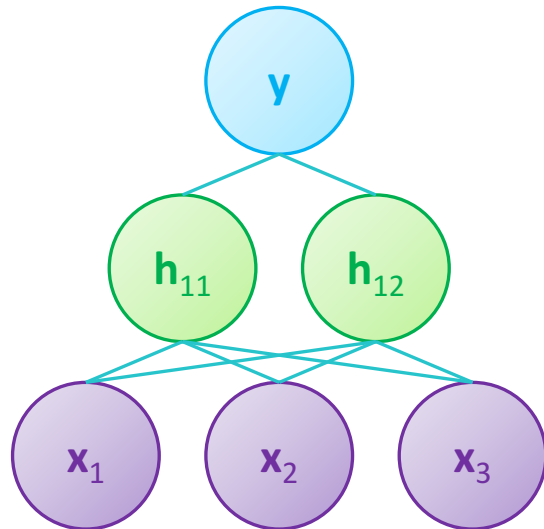
# Supervised vs. Unsupervised Networks

	Supervised	Unsupervised
Single-layer network	 <p>Multilayer perceptron (MLP)</p>	 <p><u>Autoencoder</u> (with noise injection: <u>denoising autoencoder</u>)</p>
Deep Network	 <p>Deep Neural Network</p>	 <p><u>Stacked autoencoder</u> (with noise injection: <u>stacked denoising autoencoder</u>)</p>

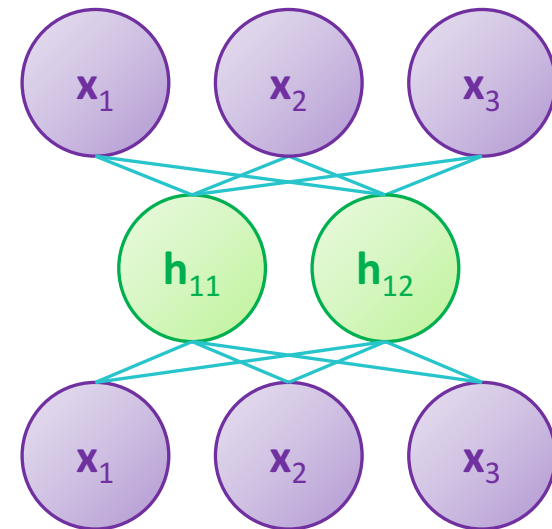
# Unsupervised Training

---

Supervised Neural Network



Unsupervised Neural Network



(Known as an autoencoder)

# Unsupervised Training

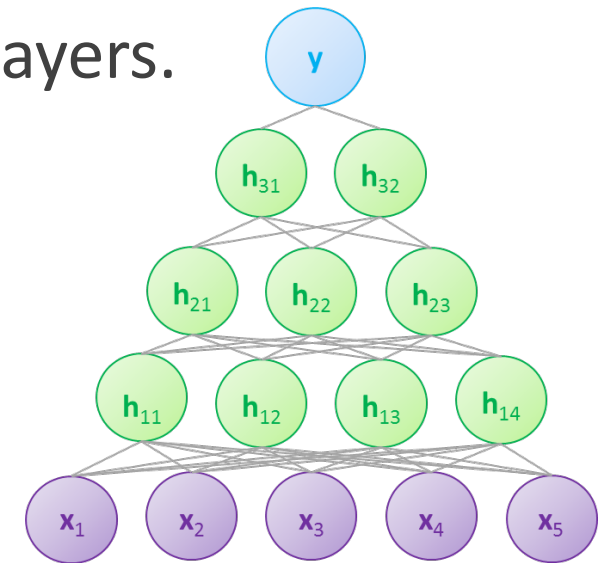
---

- Is it trivial to learn  $\mathbf{X}$  from  $\mathbf{X}$ ?
- Sometimes it is, but the neural network simply learns to duplicate the training data instead of learning **generalizable concepts** from the training data
- To avoid this problem, you can use **L2 regularization** which is equivalent to corrupting the training set by adding Gaussian noise
- A **denoising autoencoder** is a single-hidden layer unsupervised neural network with L2 regularization or Gaussian noise added to the training data

# Stacked Layers

---

- Deep networks are built by stacking layers of units
  - The output of one layer is the input to another layer
- Deep networks can be built from several types of layers.
  - Conventional hidden units
  - Convolutional filters



# Stacked layers: Conventional Hidden Units

---

- Conventional Neurons
  - Several types of networks can be built from layers composed of conventional neural network neurons

# Stacked layers: Convolutional Neural Networks

---

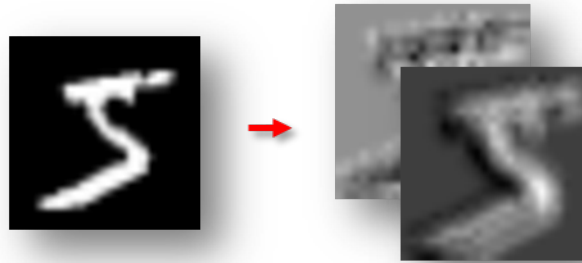
- Built by stacking layers composed of image convolution and down-sampling (pooling) followed by conventional hidden layers
- Typically used for supervised pattern recognition tasks
- Image convolution and down-sampling reduce the number of hidden units needed in upper layers of the network while also extracting robust features from the training data



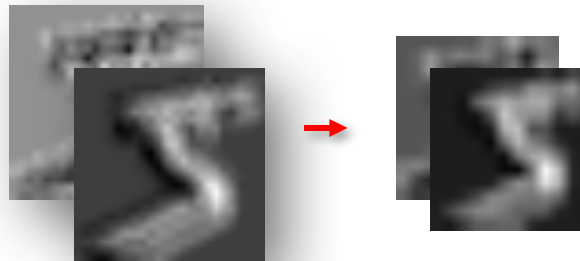
# Convolution and Down-Sampling

---

Convolution applies a filter to a neighborhood of pixels



Down-sampling summarizes a group of pixels

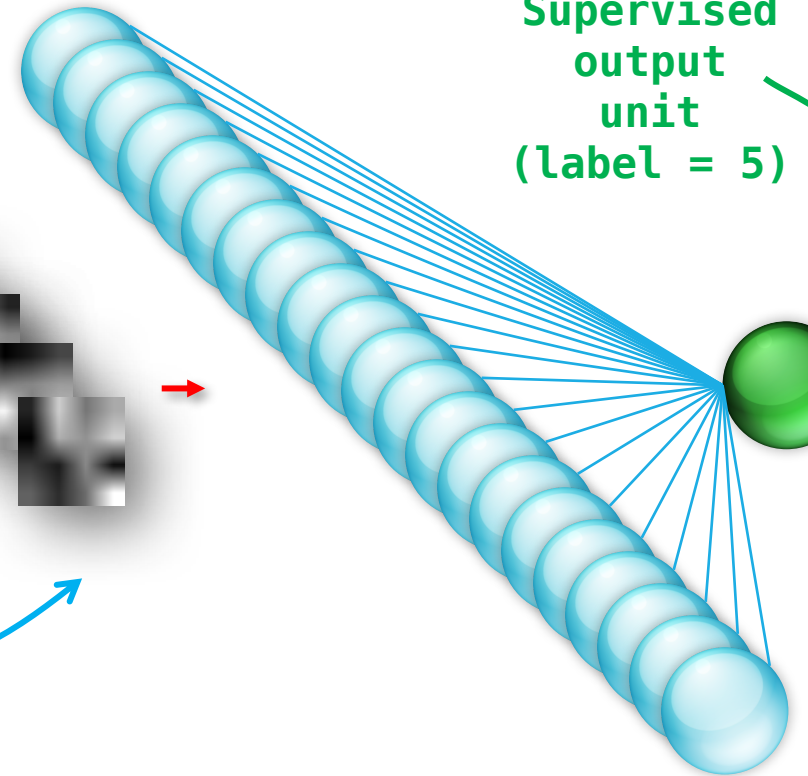
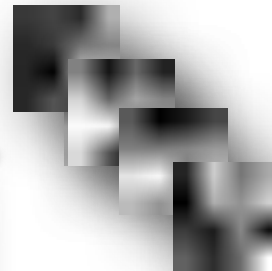
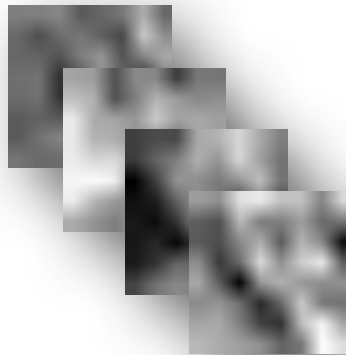


The number of convolutional filters at each layer is a hyperparameter of the network. Filter values are optimized along with hidden unit weights during supervised training.

All images of each input example are flattened into a single vector and fed to an Multi-layer perceptron classifier.

Supervised  
output  
unit  
(label = 5)

Input  
vectors  
representing  
pixels



Down-sampling  
reduces  
computational  
complexity for  
upper layers and  
provides a form of  
scale and  
translation  
invariance.

1<sup>st</sup>  
convolutional  
layer

1<sup>st</sup> down-sampling  
layer

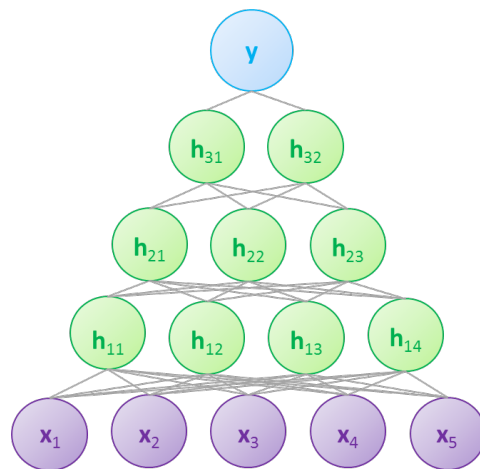
2<sup>nd</sup> convolutional  
layer

2<sup>nd</sup> down-sampling  
layer

Fully connected MLP  
classifier

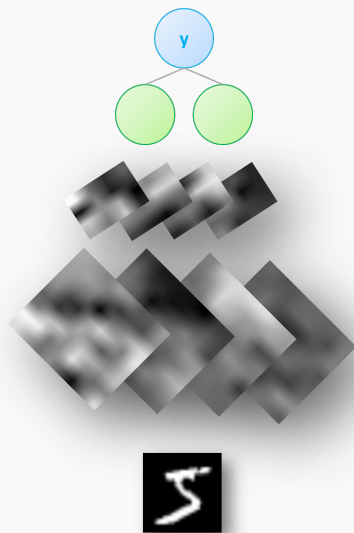
# Deep Supervised Network

Conventional Hidden  
Layers



Deep Neural Network

Convolution, Down-  
sampling, conventional  
Hidden layers



Convolutional neural network

# Fitting Neural Networks

---

- Objective – seek to optimize unknown weight parameters to fit the training data
  - Set of weights,  $\theta$ , consist of

$$\begin{aligned} \{\alpha_{0m}, \alpha_m; m = 1, 2, \dots, M\} & \quad M(p + 1) \text{ weights,} \\ \{\beta_{0k}, \beta_k; k = 1, 2, \dots, K\} & \quad K(M + 1) \text{ weights.} \end{aligned}$$

- Measure of fit – Error Function
  - Regression – SSE

$$R(\theta) = \sum_{k=1}^K \sum_{i=1}^N (y_{ik} - f_k(x_i))^2.$$

- Classification – squared error or cross-entropy (deviance)

$$R(\theta) = - \sum_{i=1}^N \sum_{k=1}^K y_{ik} \log f_k(x_i).$$

# Fitting Neural Networks

---

- Optimizing Error Function
  - Do not want global minimizer of  $R(\theta)$  – likely to yield over-fitted solution
  - Hence, need some regularization – i.e. penalty term or early stopping
- Gradient Descent – generic approach to minimize  $R(\theta)$ 
  - AKA back-propagation
  - Easily be derived using differentiation (chain-rule)
- Let  $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$  and  $z_m = (z_{1i}, z_{2i}, \dots, z_{Mi})$
- Then we have

$$\begin{aligned} R(\theta) &\equiv \sum_{i=1}^N R_i \\ &= \sum_{i=1}^N \sum_{k=1}^K (y_{ik} - f_k(x_i))^2 \end{aligned}$$

# Fitting Neural Networks

---

with derivatives

$$\begin{aligned}\frac{\partial R_i}{\partial \beta_{km}} &= -2(y_{ik} - f_k(x_i))g'_k(\beta_k^T z_i)z_{mi}, \\ \frac{\partial R_i}{\partial \alpha_{m\ell}} &= -\sum_{k=1}^K 2(y_{ik} - f_k(x_i))g'_k(\beta_k^T z_i)\beta_{km}\sigma'(\alpha_m^T x_i)x_{i\ell}.\end{aligned}\tag{11.12}$$

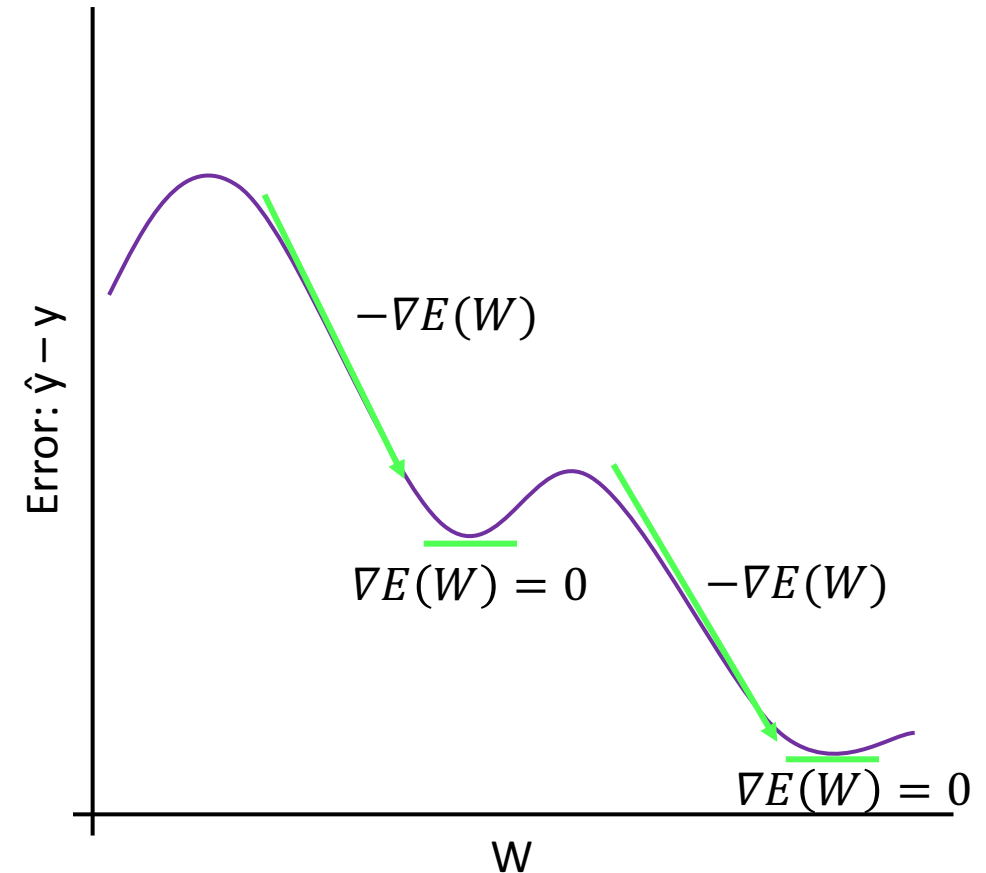
Given these derivatives, a gradient descent update at the  $(r + 1)$ st iteration has the form

$$\begin{aligned}\beta_{km}^{(r+1)} &= \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^N \frac{\partial R_i}{\partial \beta_{km}^{(r)}}, \\ \alpha_{m\ell}^{(r+1)} &= \alpha_{m\ell}^{(r)} - \gamma_r \sum_{i=1}^N \frac{\partial R_i}{\partial \alpha_{m\ell}^{(r)}},\end{aligned}\tag{11.13}$$

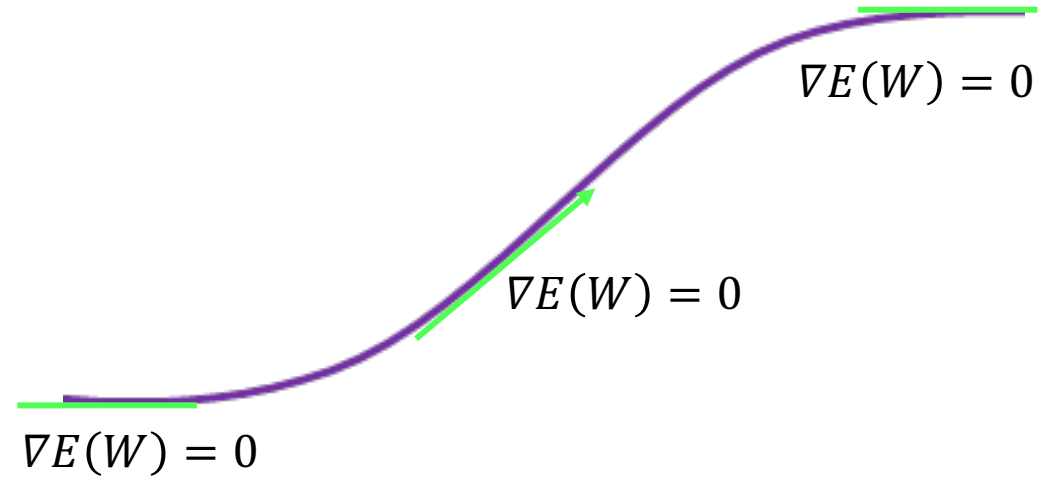
where  $\gamma_r$  is the *learning rate*, discussed below.

# Training Deep Networks: Gradient Descent

- The gradient of the training objective function is used to train the network and minimize training error
- When the gradient vector becomes zero, training stops because zero gradient vector represents minimum in the training objective function



# Vanishing Gradients



- To calculate the gradient for training a deep neural network, the weights and their layer-wise gradients must be multiplied together many times
- Initialization with large random numbers often causes neurons to become saturated, leading to extremely small gradients
- Initialization with small random numbers often causes the gradient to become meaninglessly small due to finite precision problems
- **So deep networks must be trained or initialized some other way**

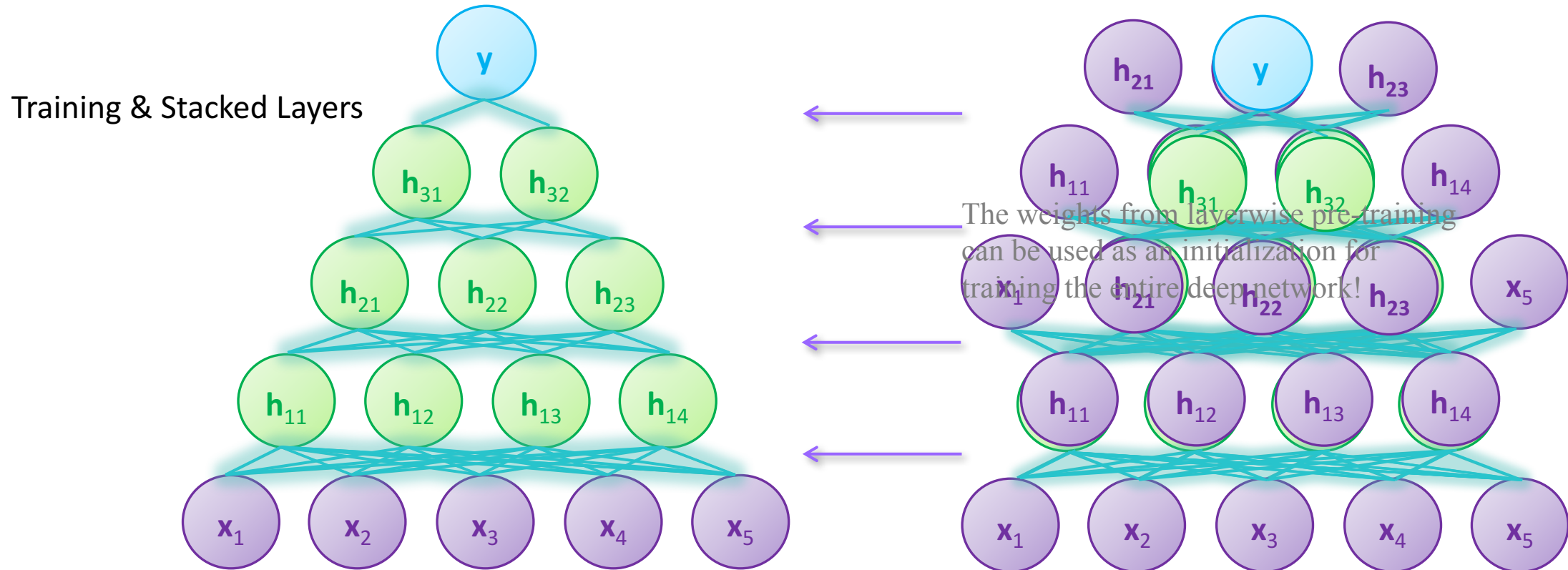


## Initialize Deep Networks by Unsupervised Training

---

- Originally done to avoid vanishing gradients
- Separately initialize each layer of a deep network as a single-layer autoencoder
- Use trained weights from each single-layer network as the initialization for training the entire deep network

# Unsupervised pretraining



Many separate, unsupervised, single hidden-layer networks are used to initialize a larger unsupervised network in a layerwise fashion

# In Practice...

---

Many researchers question the need for layerwise pre-training. They say that all the layers of a deep network can be trained together using the right initialization and optimization routine

Architectures and optimization routines for deep networks still require a lot of tuning for any specific problem

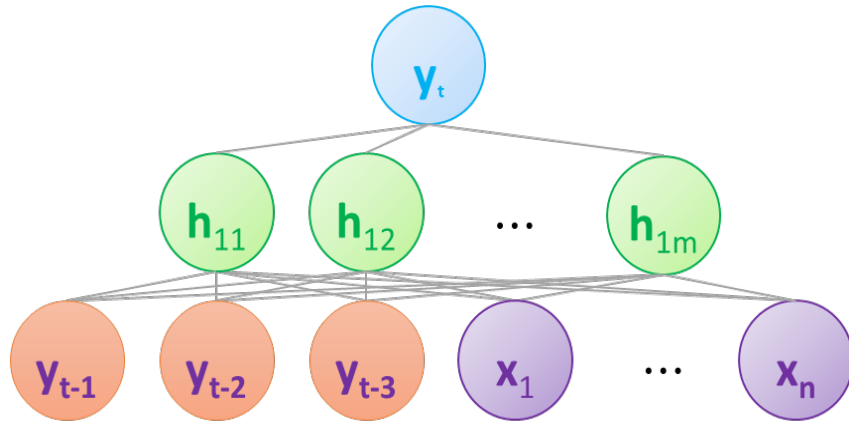
# Parameter Tuning

---

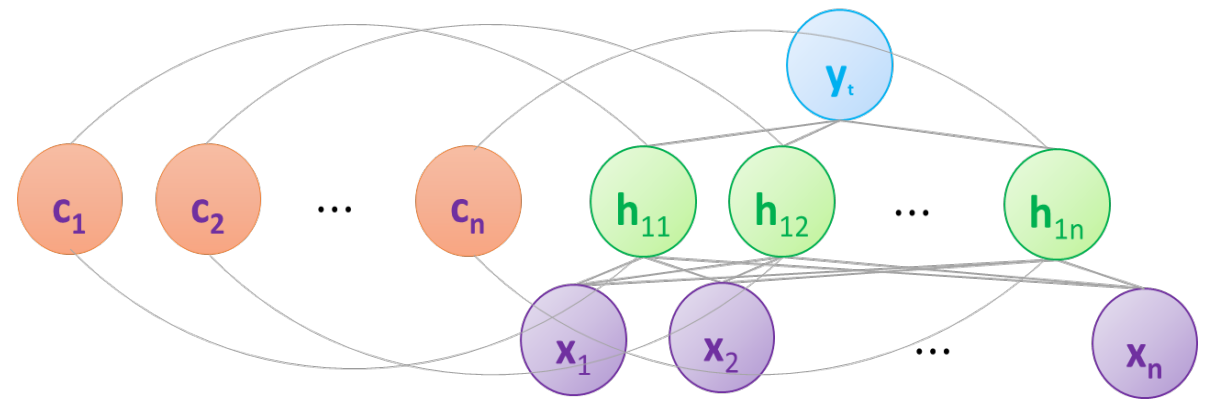
- Model performance can vary widely under different parameter settings
- Serious attempts at machine learning must approach parameter tuning scientifically:
  - Benchmarking
  - Grid search
  - Design of experiments (DOE)
  - Local search optimization (LSO)
  - Genetic Algorithms (GA)

# Applications to Time Series & Sequences

We will discuss:



Nonlinear AutoRegressive network with  
eXogenous inputs (NARX)



Recurrent Neural Network (RNN)

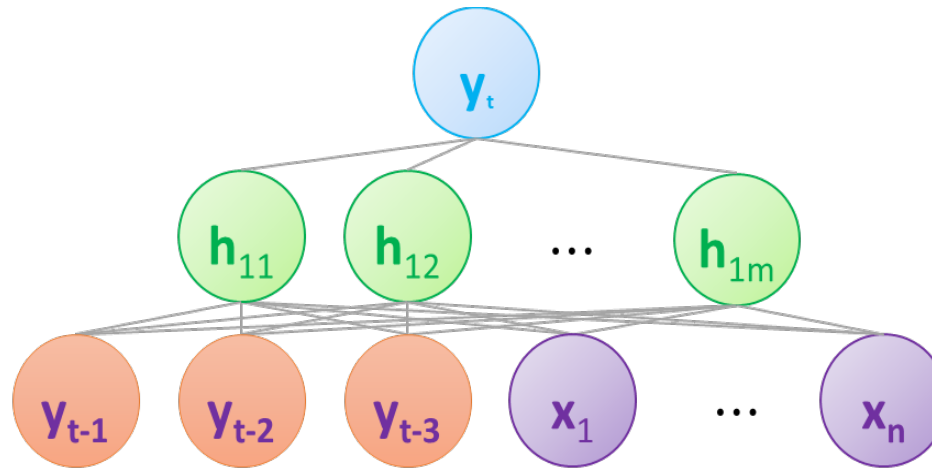
# Why Use Neural Networks for Time Series & Sequences?

---

- Neural networks and other machine learning algorithms are occasionally more accurate than traditional time series methods on **difficult series**
  - No distributional assumptions
  - Less interpretable
- Traditional time series approaches do not model **sequences of categorical targets**

# Nonlinear AutoRegressive Network with eXogenous Inputs (NARX)

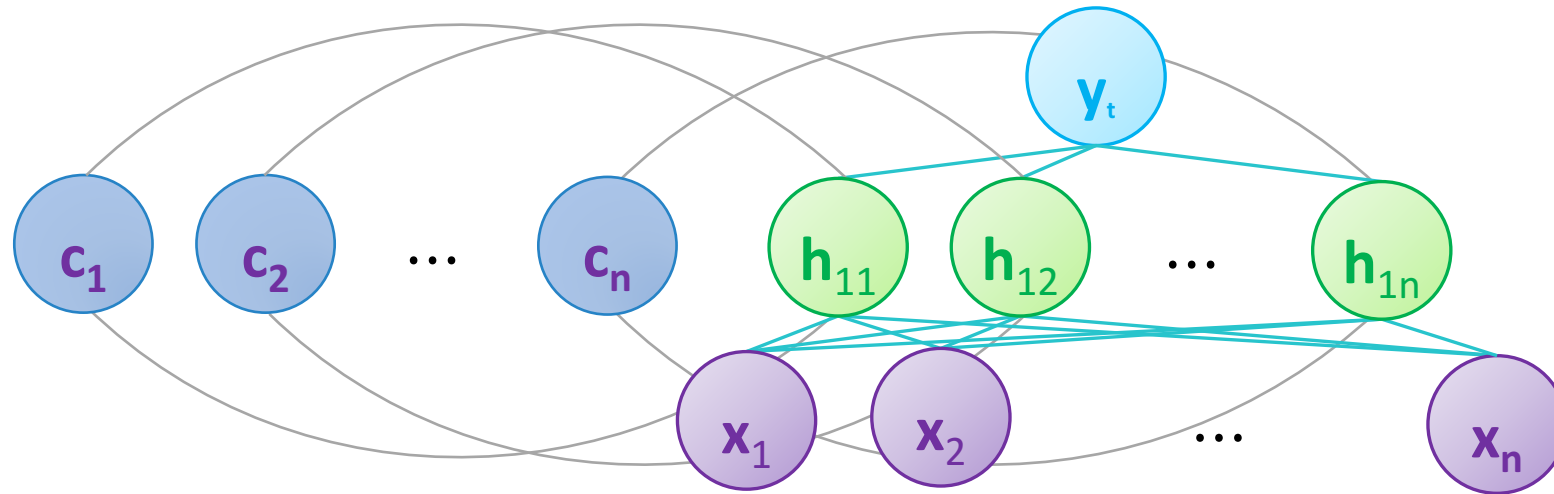
---



A NARX model uses time lags of the target variable and other input variables to predict the next value in a series of interval targets or a sequence of categorical targets

# Recurrent Neural Network (RNN)

---

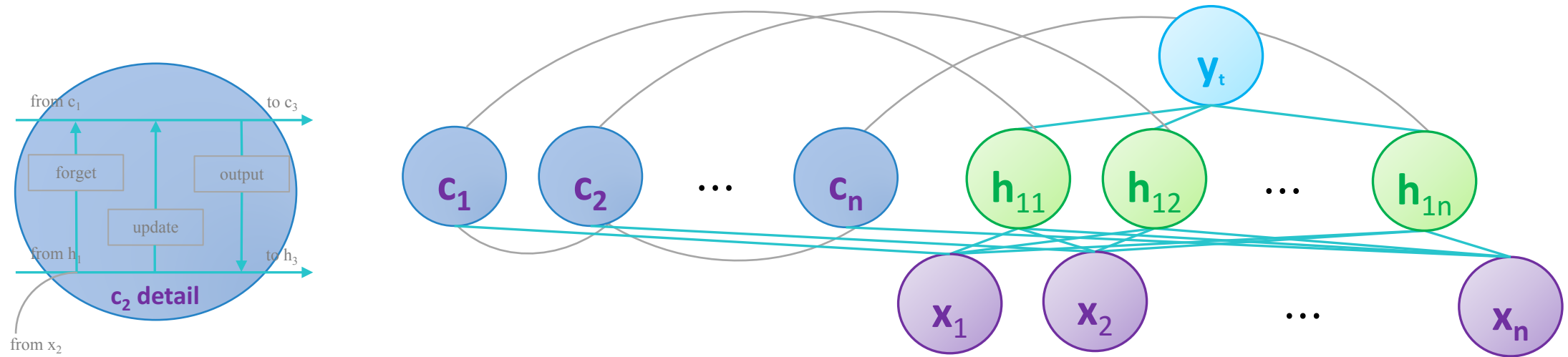


A RNN model attempts to optimally generate features from past events (remember past events) and use these features along with conventional model inputs to predict a series of interval targets or a sequence of categorical targets

The basic RNN architecture has been widely discredited



# Long Short Term Memory Recurrent Neural Network (LSTM RNN)



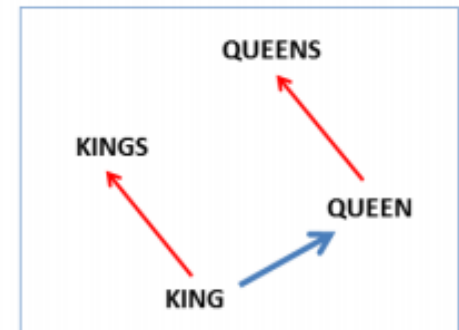
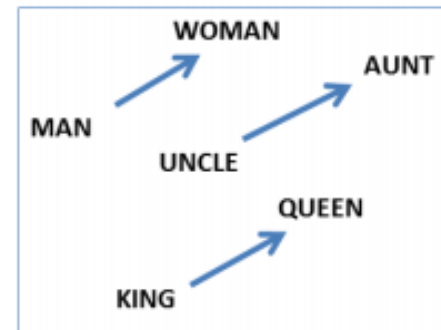
An LSTM RNN uses a more sophisticated network structure in which past information can be remembered or forgotten

Many of the recent advances in deep learning have been applied to LSTM RNN models enabling significant breakthroughs in sequence learning

# Applications to Text Mining

- Bag of words is a global, **matrix factorization** model capable representing prominent ideas in a group of documents; it assumes independence between terms
- Term embeddings are local, contextual models capable of representing relationships between words that can be generated by neural networks

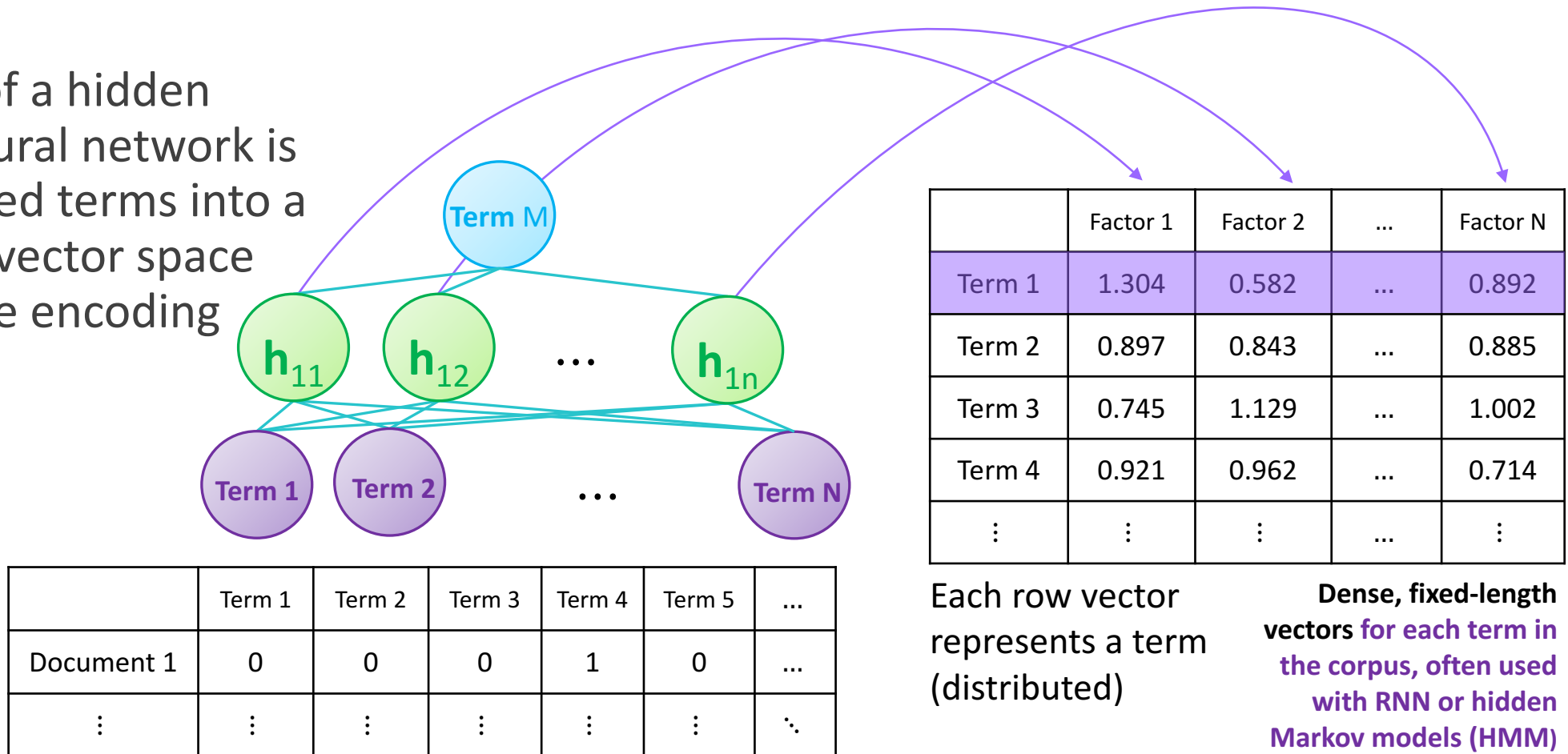
*king – man + women = queen*



Mikolov, Tomas, Wen-tau Yih, and Geoffrey Zweig. "Linguistic Regularities in Continuous Space Word Representations." HLT-NAACL. 2013.

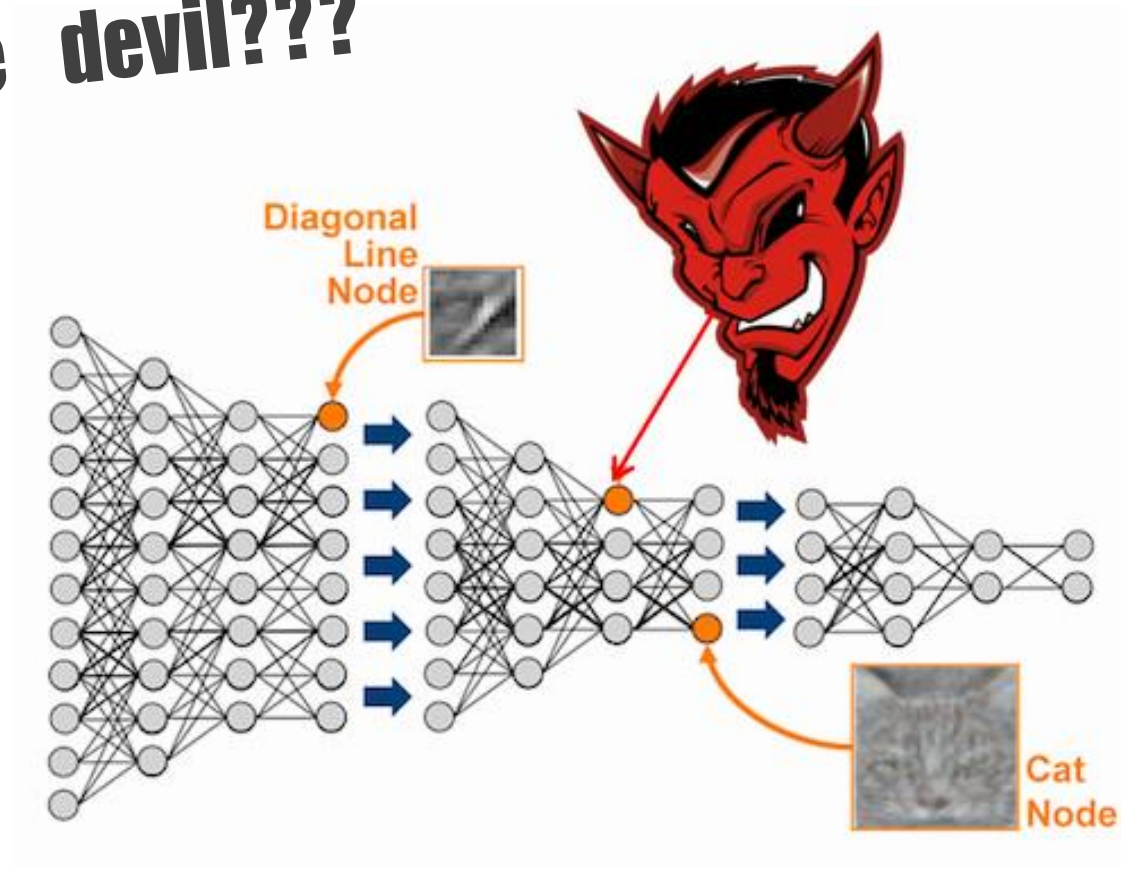
# Neural Network Term Embeddings (like Google's word2vec)

The output of a hidden layer of a neural network is used to embed terms into a fixed-length vector space from a simple encoding



# The shocking truth revealed!

**Is deep learning from the devil???**



# Who knows! Regardless ...

---

- Issues and Problems:
  - Empirical, brute force approach
  - Models are very difficult to interpret
  - Known failures for simple use cases
  - Somewhat unproven outside of pattern recognition

# Issues in Training Neural Networks

---

- Model is generally over-parametrized
- Optimization – non-convex and unstable
- Starting Values
  - Typical starting values for weights are chosen to be random values near zero, hence the model starts out nearly linear
  - If exact zero is used for weights, then derivative is zero
  - Hence, the optimization algorithm never moves
  - Whereas, starting with large weights often leads to poor solutions
- Overfitting
  - Often, neural networks have too many weights and will overfit the data at the global minimum of  $R$ . Hence, the data is trained with an early stopping rule to avoid overfitting (stop before reaching the global minimum)
  - Since the weights start at a highly regularized solution (random values near zero), this has the effect of shrinking the final model toward a linear model
  - Hence, validation dataset is used for determining when to stop since we expect the validation error to start increasing

# Issues in Training Neural Networks

---

- Scaling of the Inputs
  - Large effect on the quality of the final solution
  - Best to scale the inputs as  $N(0,1)$  – standard normal with mean 0 and standard deviation of 1
- Choosing Number of Hidden Units (HUs) and Hidden Layers (HLs)
  - Too few – model might not capture the nonlinearity of the data
  - Too many – extra weight can be shrunk toward zero, if appropriate regularization is used
  - Typical HUs: 5-100
  - Start with reasonably large numbers of units and train the data set with regularization
- Multiple Minima
  - The error function is nonconvex – more than one local minima in the solution of the gradient function
  - As a result, final solution depends on the choice of starting weights
  - Hence, should try a number of random starting configurations and choose the solution that gives the lowest (penalized) error
  - Another approach – bagging (averages the predictions of the networks training from randomly perturbed versions of the training data)