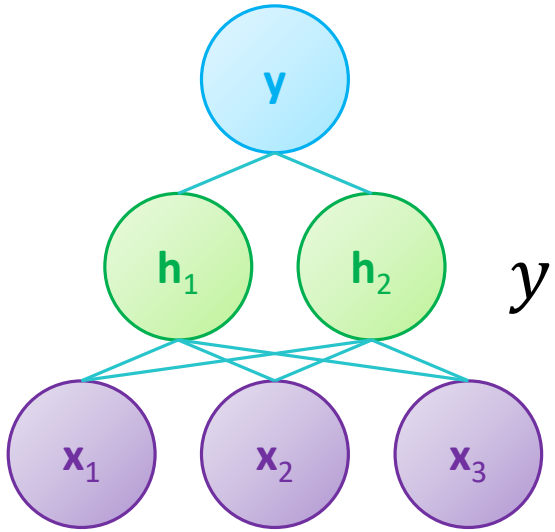
Machine Learning

PATRICK HALL AND LISA SONG

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

What is a neural network?

Historical Development

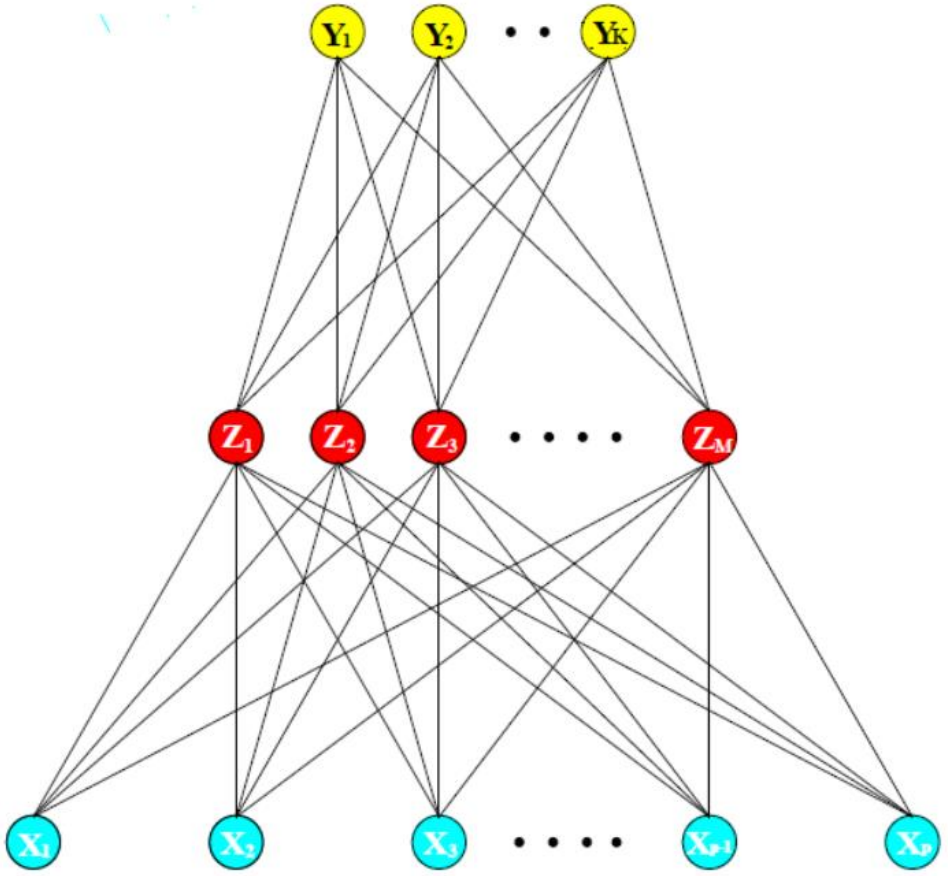
The name "neural network" originates from its initial conception to model neurotransmission (neural synaptic propagation) where each unit represents a neuron and the connections represent synapses with use of step function as an activation function for $\sigma(T)$ and $g_m(T)$. Later, step function was replaced by smoother threshold function (sigmoid function) for optimization.

Neural Network: Overview

Neural Networks - A class of learning method based on statistics and artificial intelligence

- Extract linear combination of the input parameters as derived features
- Model the target as a nonlinear function of the derived features

Neural Network: Illustration



Elements of Statistical Learning (pg.393)

Single Hidden Layer Feed-Forward Neural Network Schematic

Typical two-staged regression or classification model:

For regression, typical value of $K = 1$ with only one output unit Y_1 at the top. For K-classification model, K units at the top - *kth* unit modeling the probability of class and K target measurements $Y_k, K = 1, ..., K$ each being coded as 0-1 (dummy variable) for each of k classes.

FIGURE 11.2. *Schematic of a single hidden layer, feed-forward neural network.*

Model Conception

A neural network models a target Y_k as a function of a linear combination of derived features Z_m that are extracted from linear combination of the input variables X.

$$
\begin{aligned}
Z_m &= \sigma(\alpha_{0m} + \alpha_m^T X), \quad m = 1, ..., M \\
T_k &= \beta_{ok} + \beta_k^T Z, \quad k = 1, ..., K \\
f_k(X) &= g_k(T), \quad k = 1, ..., K
\end{aligned} \tag{1}
$$

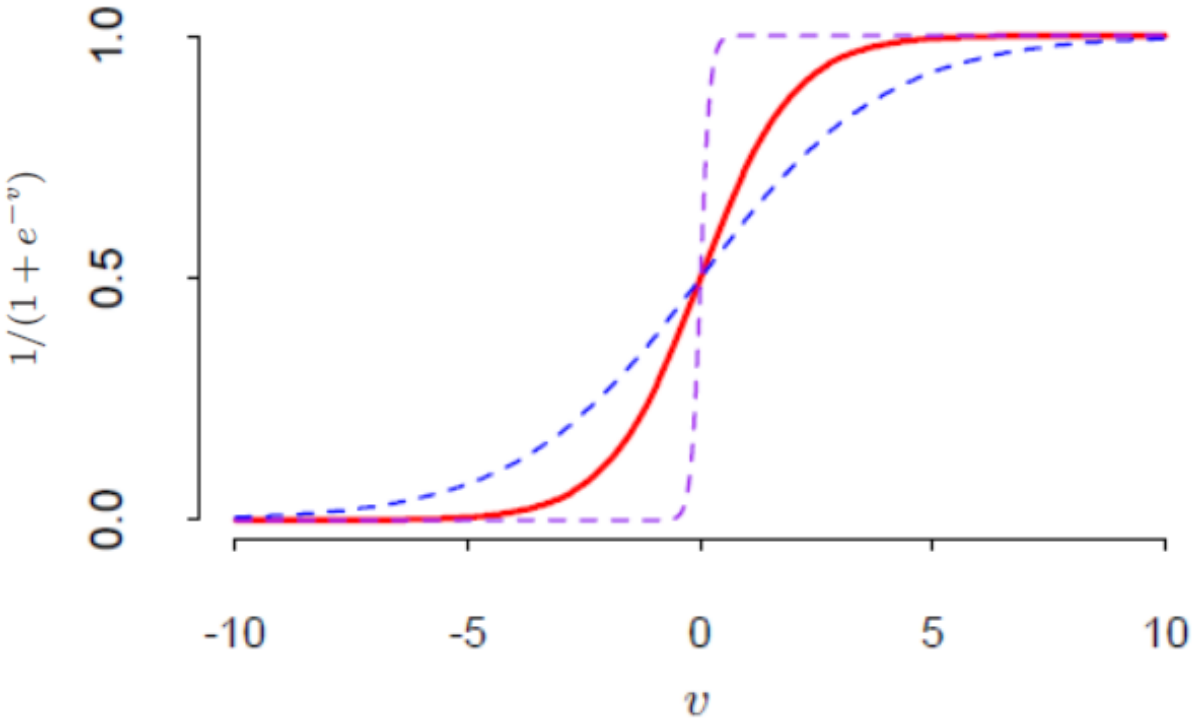
where $Z = (Z_1, Z_2, ..., Z_M)$, and output vectors as $T = (T_1, T_2, ..., T_K)$ and output functions denoted by $g_k(T)$.

Model Conception

Activation Function:

- Activation Function: $\sigma(v)$
  - Sigmoid function: $\frac{1}{1+e^{-v}}$, most commonly used
  - hyperbolic tangent function: $tanhz = \frac{sinhz}{coshz} = \frac{e^{2z}-1}{e^{2z}+1}$
  - ReLU
  - Leaky ReLU
- Additional Bias Unit
  - Every unit in the hidden and output layers
  - Captures the intercepts a_{0m} and b_{0k} in the model equation (1).

Neural Network: Illustration



Elements of Statistical Learning (pg.394)

A plot of a sigmoid function - commonly used activation function. Note that if σ is an identity function, then the entire model is reduced to a linear model in the inputs. Hence, a neural network can be thought of as a nonlinear generalization of the linear model for both regression and classification.

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*.*

Model Conception

Output Function: $g_k(T)$

Allows a final transformation of the vector of outputs T

- For Regression
  - Typically choose the identify function $g_k(T) = T_k$
- For Classification
  - *softmax*

$$g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}} \tag{2}$$

Note, the *softmax* is the transformation that is used in the multilogit model.

Model Conception

Hidden Units:

- The units in the "middle" of the network
- Computes the derived features, Z_m
- Called hidden units - values of Z_m are not directly observed in the model
- Can be more than one hidden units and layers

We can think of the derived features, Z_m, as a basis expansion of the original input $\boldsymbol{X}$. Then we can think of a neural network as a standardized linear model or multilogit model using the transformation functions as inputs.

Neural Networks: Fitting

Fitting Neural Networks:

- Weights
- Measure of Fit
  - Regression: sum-of-squares
  - Classification: Squared error or Cross-Entropy
- Optimization: Gradient Descent (Back Propagation)
  - Learning rate - γ_r

Neural Networks: Weights

Weights: Unknown parameters that make the model fit the training data. A complete set of weights, $\boldsymbol{\theta}$, is defined as follows:

$$\begin{aligned} &\{\alpha_{om}, \alpha_m; m = 1, 2, ..., M\}\ M(p+1)\ \textit{weights}; \\ &\{\beta_{ok}, \beta_k; k = 1, 2, ..., K\}\ K(M+1)\ \textit{weights}. \end{aligned} \tag{3}$$

Neural Networks: Measure of Fit

Measure of Fit: error function and deviance measure

For regression: sum-of-squared error measure

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2 \tag{4}$$

For classification: either squared-error or cross-entropy measure

$$R(\theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log f_k(x_i) \tag{5}$$

where the corresponding classifier is $G(x) = argmax_k f_k(x)$.

Neural Networks

Note, in case of classification where softmax activation function and the cross-entropy error function are employed - this construction of a neural network model is exactly a logistic regression model in the hidden units, and all the parameters are estimated by the maximum likelihood.

Neural Networks: Regularization

Regularization: global minimizer of $R(\theta)$ is likely to be an overfit solution. Instead, regularization is needed and achieved through a penalty (direct) term or early stopping (indirect). The generic approach to minimize $R(\theta)$ by gradient descent is called *back-propagation*. Gradient descent can be constructed using the chain rule differentiation that is computed by a forward and backward sweep over the network. Here, gradient is the first row vector of the Jacobian matrix .

Neural Networks: Regularization

Back-propagation for squared error loss:

Let $z_{mi} = \sigma(\alpha_{om} + \alpha_m^T x_i)$ from equation (1) and $z_i = (z_{1i}, z_{2i}, ..., z_{Mi})$. Then we have

$$\begin{aligned} R(\theta) &= \sum_{i=1}^{N} R_i \\ &= \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2 \end{aligned} \tag{6}$$

Neural Networks: Gradient

With the derivatives:

$$\frac{\partial R_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)z_{mi},$$
$$\frac{\partial R_i}{\partial \alpha_{ml}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)\beta_{km}\sigma'(\alpha_m^T x_i)x_{il} \quad (7)$$

Neural Networks: Gradient Descent

Given previously defined derivatives, a gradient descent update at the $(r+1)th$ iteration has the form:

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}$$
$$\alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}} \tag{8}$$

where γ_r is the learning rate.

Neural Networks: Gradient Descent

Now, equation (7) is reduced to:

$$\begin{aligned} \frac{\partial R_i}{\partial \beta_{km}} &= \delta_{ki} z_{mi}, \\ \frac{\partial R_i}{\partial \alpha_{ml}} &= s_{mi} x_{il} \end{aligned} \tag{9}$$

Here, δ_{ki} and s_{mi} are "errors" at the output and hidden layer units, and these errors satisfy:

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki} \tag{10}$$

known as the back-propagation equation.

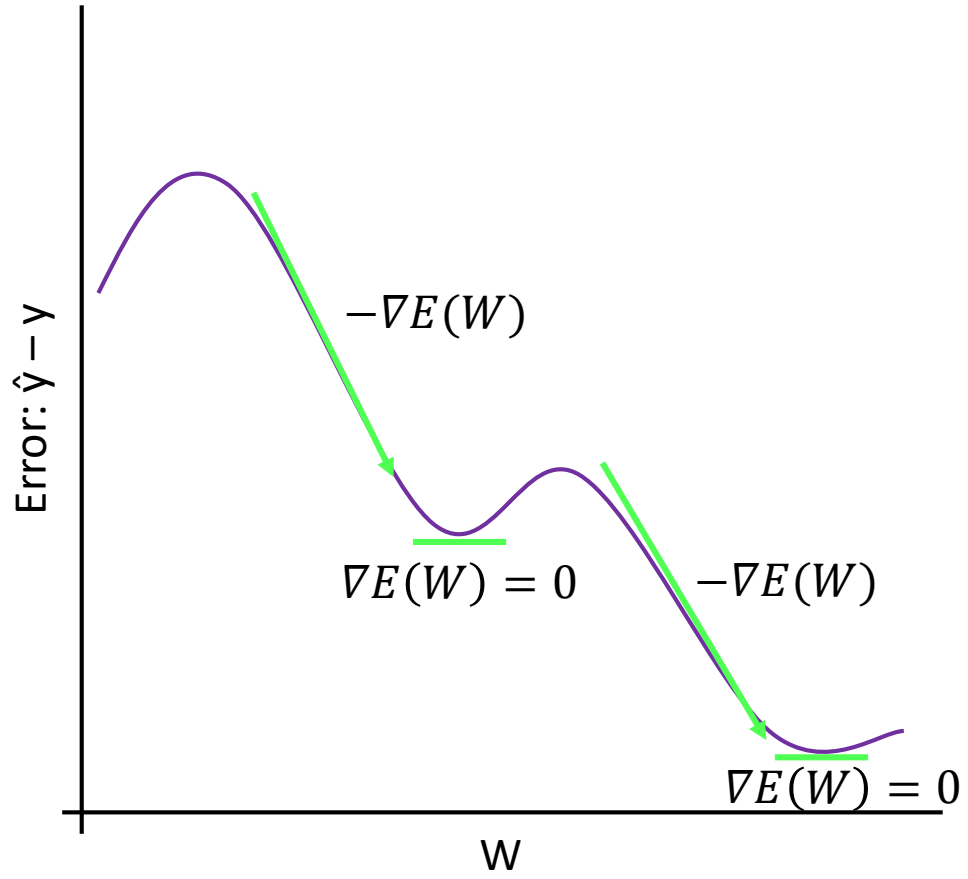
Neural Networks: Gradient Descent

Now, a 2-pass algorithm can be implemented using the updated gradient descent from equation (8).

- Forward Pass - the current weights are fixed and predicted values $\hat{f}_k(x_i)$ are computed from equation (10)
- Backward Pass - the error σ_{ki} are computed and then back-propagated via equation (10) to yield s_{mi}

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



Neural Networks: Training Issues

Issues in Training Neural Networks: Neural network models are are generally overparametrized and optimization is non-convex and unstable. Some important issues are:

- Starting Values
- Overfitting
- Scaling of the Inputs
- Number of Hidden Units and Layers
- Multiple Minimas

Neural Networks: Training Issues

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

Neural Networks: Training Issues

Overfitting: Neural Networks often have too many weights and will overfit the data at the global minimum of R. An early stopping rule is used to stop before the model reaches the global minimum R. A validation dataset is useful in determining when to stop since we expect the validation error to start increasing.

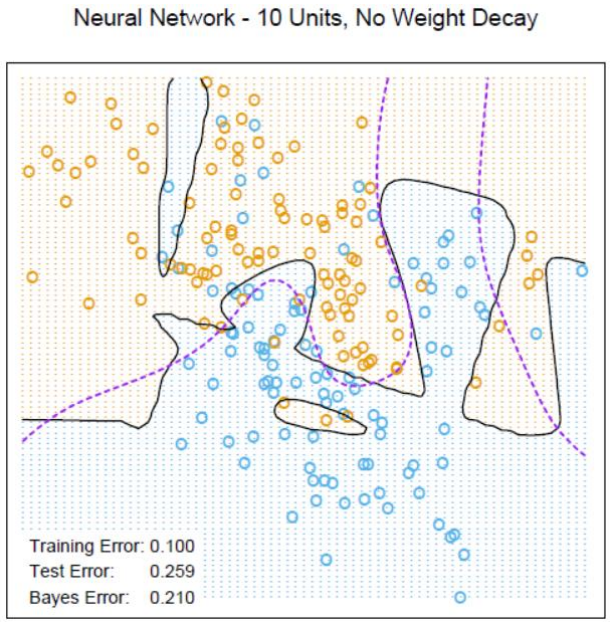
Neural Networks: Training Issues

Regularization: Weight Decay adds penalty term to the error function $R(\theta) + \lambda J(\theta)$ where,

$$J(\theta) = \sum_{km} \beta_{km}^2 + \sum_{ml} \alpha_{ml}^2 \tag{11}$$

and $\lambda \geq 0$ is a tuning parameter. Note, large values of λ will tend to shrink the weights toward zero; and cross-validation is used to estimate λ.

Neural Networks: Training Issues



Elements of Statistical Learning (pg.394)

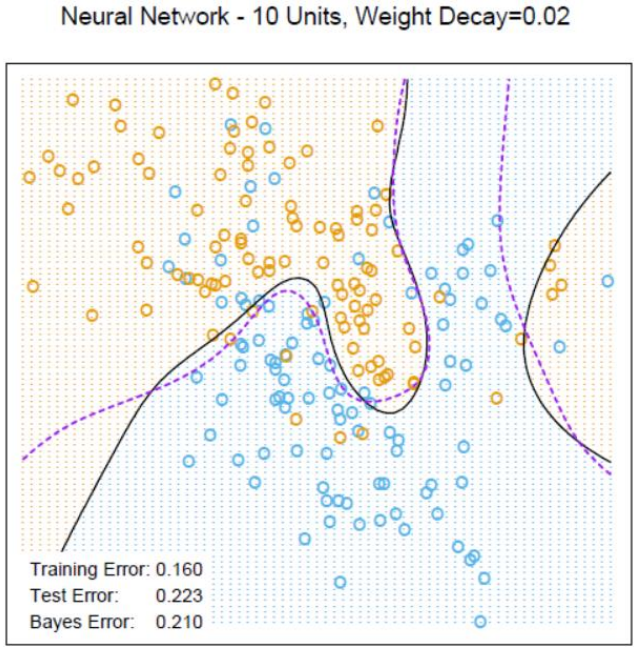


FIGURE 11.4. *A neural network on the mixture example of Chapter 2. The upper panel uses no weight decay, and overfits the training data. The lower panel uses weight decay, and achieves close to the Bayes error rate (broken purple boundary). Both use the softmax activation function and cross-entropy error.*

Neural Networks: Training Issues

Scaling of the Inputs: Determines the effective scaling of the weights in the bottom layer, hence, it can have large effect on the quality of the final solution.

- Standardize all input to have standard normal distribution, $X \in N(0,1)$
- Ensures all inputs are treated equally in regularization process
- Enables meaningful ranges for the random starting weights
- With standardization - typical random uniform weights are over the range of $[-0.7, 0.7]$

Neural Networks: Training Issues

Number of Hidden Units and Layers: In general, it is better to have too many hidden units than too few units.

- Too few - model might not have enough flexibility to capture the nonlinearity of the dataset
- Too many - extra weights can be shrunk towards zero if appropriate regularization is used
- Typical number of hidden units is 5-100
- Hidden layers - background knowledge and experimentation

Neural Networks: Training Issues

Multiple Minimas: Recall, the error function is non-convex, hence yields multiple local minima. Therefore, the final solution depends on the choice of starting weights.

- Try multiple random starting configuration and choose the solution that gives the lowest penalty error
- Use the average predictions over the collection of networks as the final prediction
- Bagging - averages the prediction of networks training from randomly perturbed versions of the training data.

Case: ZIP CODE Data

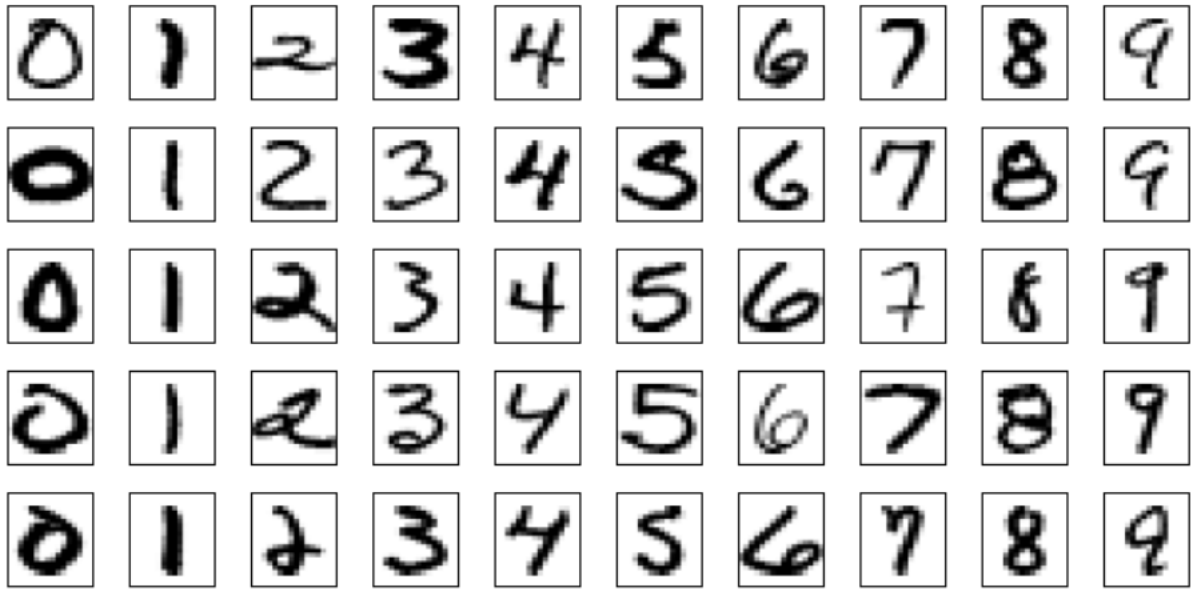


FIGURE 11.9. *Examples of training cases from ZIP code data. Each image is a 16 × 16 8-bit grayscale representation of a handwritten digit.*

Elements of Statistical Learning (pg.404)

Data Source and Treatment:

- Normalized handwritten digits scanned from envelopes from the US Postal Service
- The original scanned digits are binary and of different sizes and orientations
- Images have been deslanted and size normalized to yield 16x16 grayscale images
- 256 pixel values are used as inputs to the neural networks classifiers
- 320 digits in the training set and 160 in the test data

Case: ZIP CODE Data -new

Character Recognition Task: classification of handwritten numerals

Five different network architectures were used to fit the dataset:

- Net-1: No hidden layer, equivalent to multinominal logistic regression
- Net-2: One hidden layer, 12 hidden units fully connected
- Net-3: Two hidden layers locally connected
- Net-4: Two hidden layers, locally connected with weight sharing
- Net-5: Two hidden layers, locally connected, two levels of weight sharing

Case: ZIP CODE Data -new

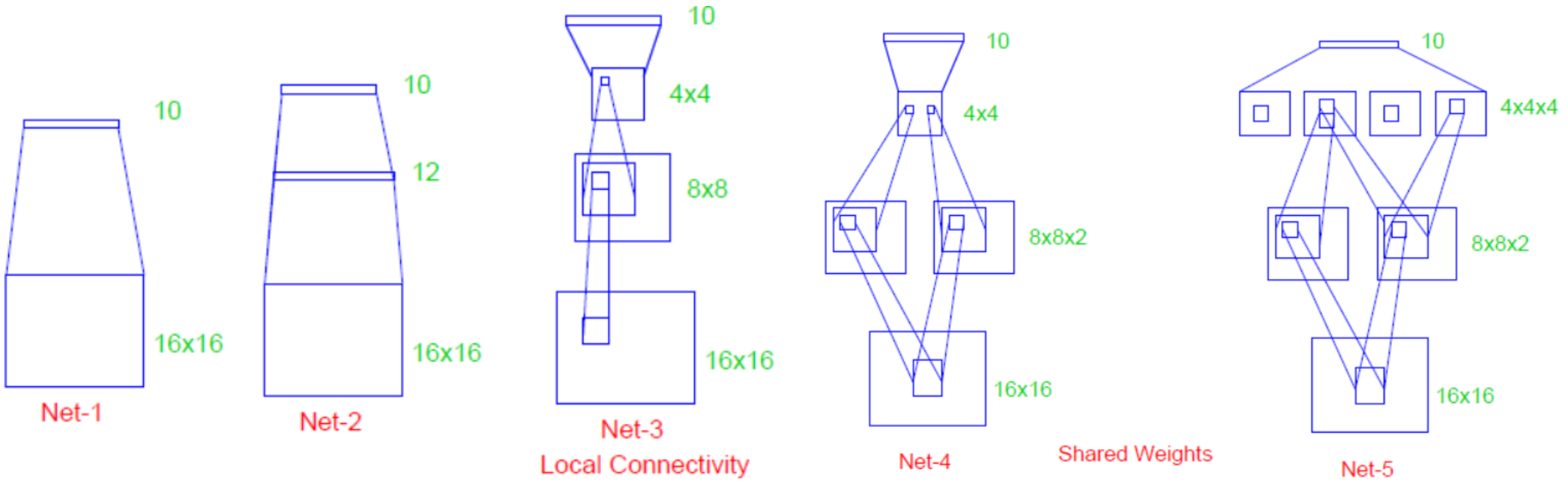
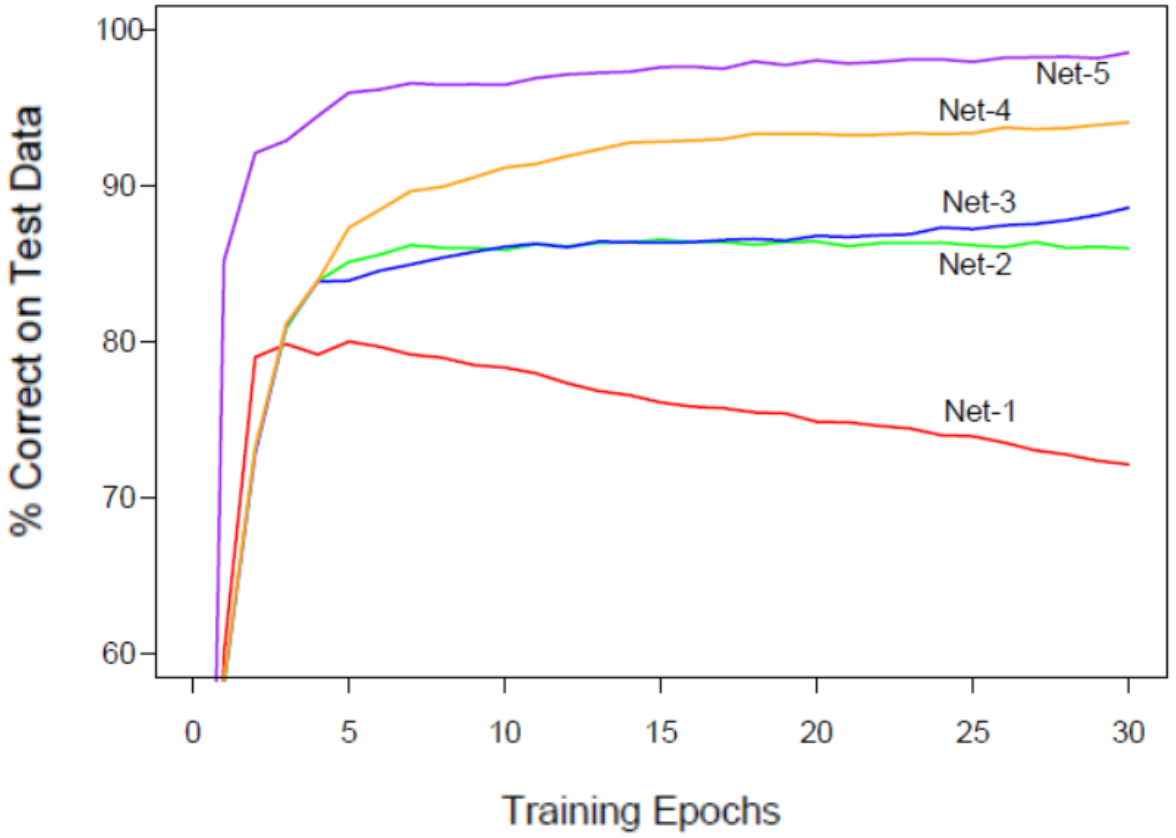


FIGURE 11.10. *Architecture of the five networks used in the ZIP code example.*

Elements of Statistical Learning (pg.405)

- Net-1 has 256 inputs and 10 outputs units for each of the digits 0-9
- The predicted values $\hat{f}_k(x)$ represent the estimated probability that an image x has digit class k for $k = 0, 1, ..., 9$

Case: ZIP CODE Data -new



Elements of Statistical Learning (pg.406)

- All networks have sigmoid output units
- All networks were fitted with sum-of-squared error function
- Net-1 overfits quickly
- Net-3,4, and 5 - demonstrates power and flexibility of neural network with additional features by introducing constraints on the network that allows for more complex connectivity but fewer parameters
- Net-4 and Net-5 have local connectivity with shared weights

FIGURE 11.11. *Test performance curves, as a function of the number of training epochs, for the five networks of Table 11.1 applied to the ZIP code data. (Le Cun, 1989)*

Case: ZIP CODE Data

Convolutional Network

- Net-4 and Net-5 have local connectivity with shared weights that allow all units in a local feature map to perform the same operation on different parts of the image
- The first layer of Net-4 has 8x8 arrays in which each of the units in a single 8x8 feature map share the same set of nine weights (with their own bias parameter)
- This architecture forces the extracted features in different parts of the image to be computed by the same linear function - known as **convolutional** network

Case: ZIP CODE Data -new

Elements of Statistical Learning (pg.407)

TABLE 11.1. *Test set performance of five different neural networks on a hand-written digit classification example (Le Cun, 1989).*

	Network Architecture	Links	Weights	% Correct
Net-1:	Single layer network	2570	2570	80.0%
Net-2:	Two layer network	3214	3214	87.0%
Net-3:	Locally connected	1226	1226	88.5%
Net-4:	Constrained network 1	2266	1132	94.0%
Net-5:	Constrained network 2	5194	1060	98.4%

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

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

Unsupervised Training

Why is unsupervised training of neural networks useful?

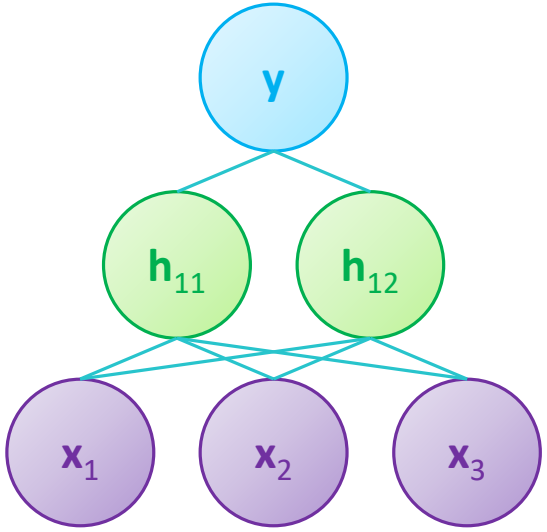
- Feature extraction and anomaly detection
- Initialization

How can a neural network be unsupervised?

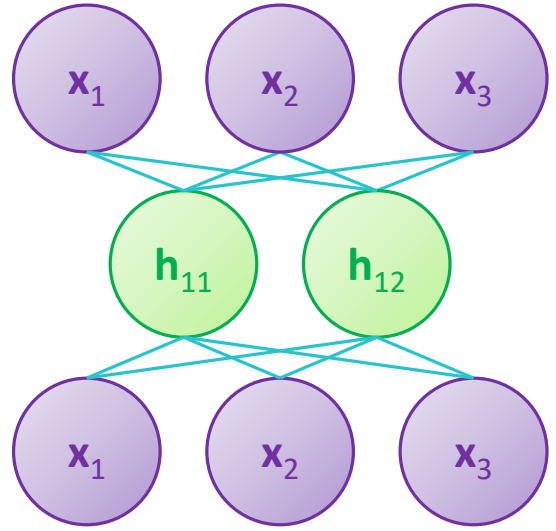
- Usually you try to predict a target vector **Y** from input vector **X**
- In unsupervised training, you try to predict **X** from **X**

Unsupervised Training

Supervised Neural Network



Unsupervised Neural Network



(Known as an autoencoder)

Unsupervised Training

- Is it trivial to learn **X** from **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

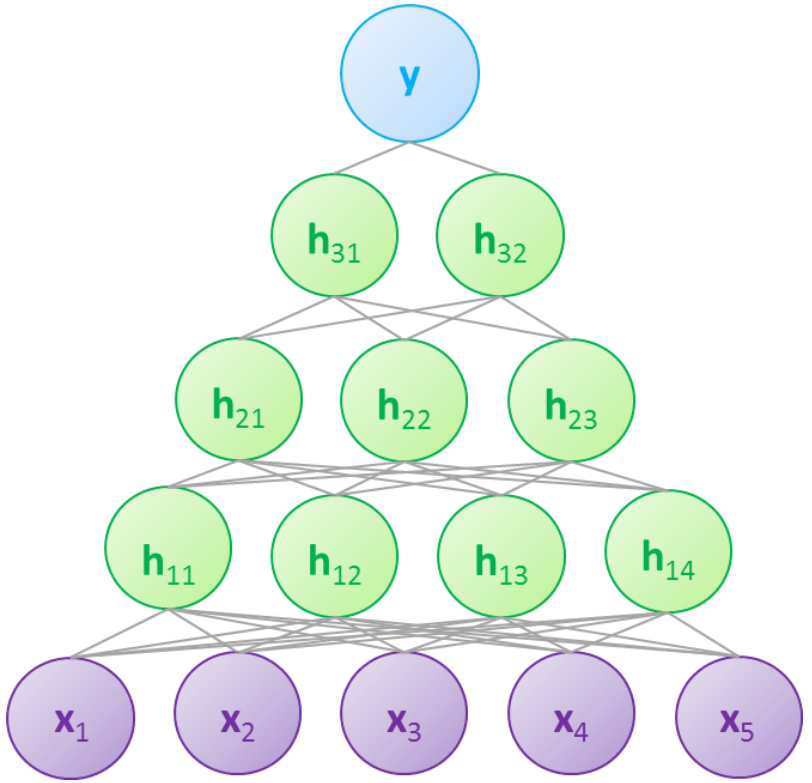
Stacked Layers

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

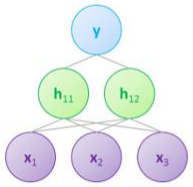
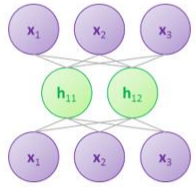
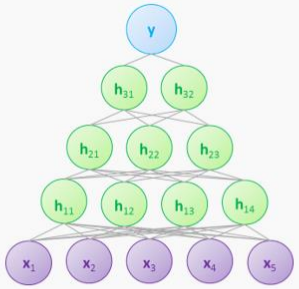
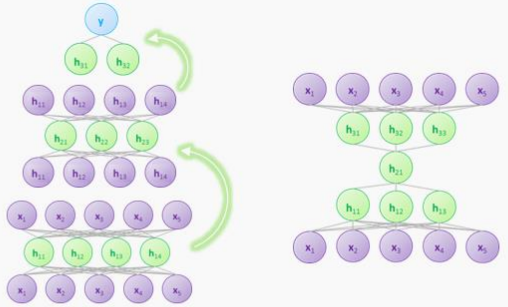
Conventional Neurons

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

Stacked layers: Convolutional Neural Networks

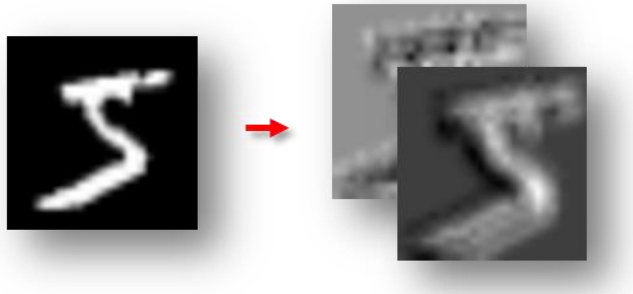
- Built by stacking layers composed of image convolution and down-sampling (pooling) followed by conventional hidden layers
- Typically used for supervised pattern recognition tasks
- Image convolution and down-sampling reduce the number of hidden units needed in upper layers of the network while also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

Supervised vs. Unsupervised Networks

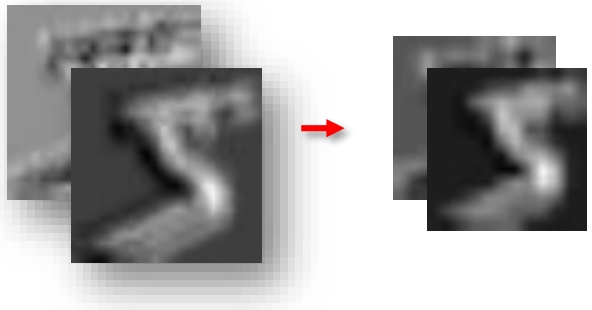
	Supervised	Unsupervised
Single-layer network	 Multilayer perceptron (MLP)	 Autoencoder (with noise injection: denoising autoencoder)
Deep Network	 Deep Neural Network	 Stacked autoencoder (with noise injection: stacked denoising autoencoder)

Convolution and Down-Sampling

Convolution applies a filter to a neighborhood of pixels



Down-sampling summarizes a group of pixels

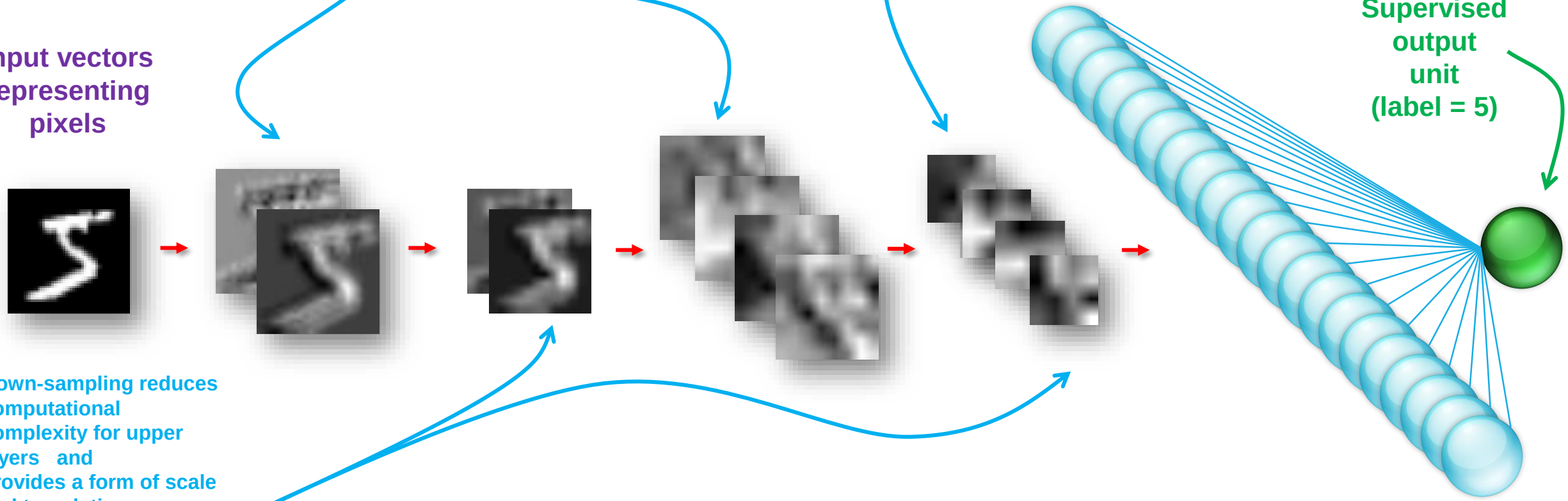


The number of convolutional filters at each layer is a hyperparameter of the network. Filter values are optimized along with hidden unit weights during supervised training.

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

Input vectors representing pixels

Supervised output unit (label = 5)



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

1st convolutional layer

1st down-sampling layer

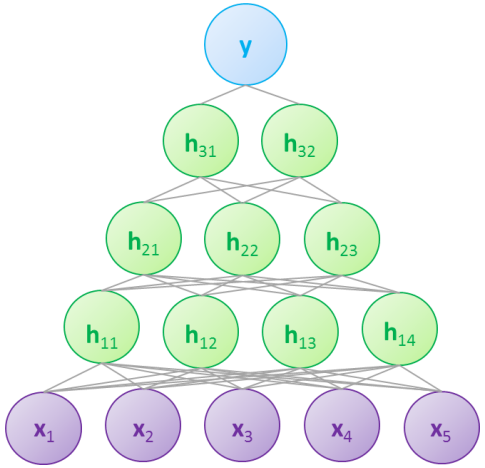
2nd convolutional layer

2nd down-sampling layer

Fully connected MLP classifier

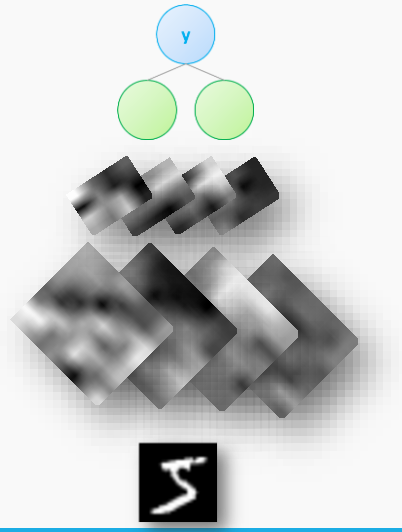
Deep Supervised Network

Conventional Hidden Layers



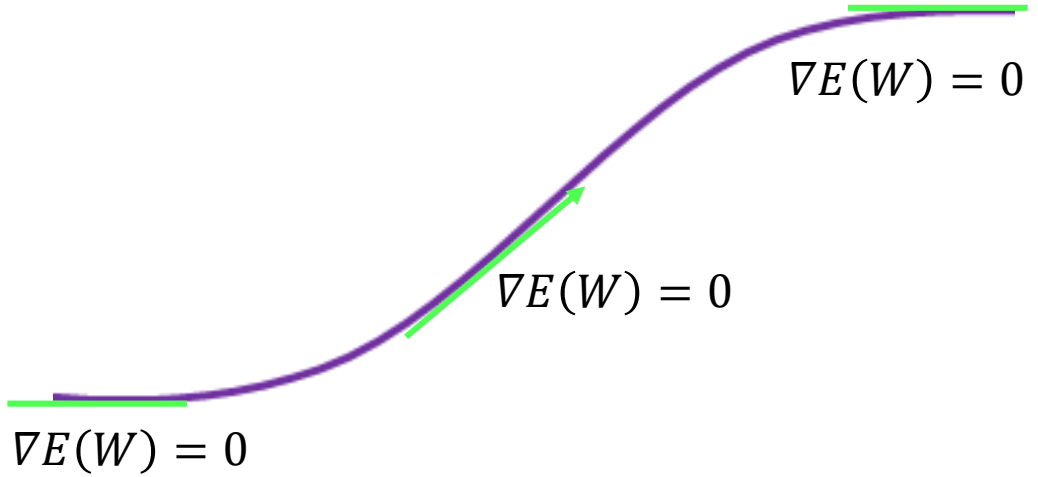
Deep Neural Network

Convolution, Down-sampling, conventional Hidden layers



Convolutional neural network

Vanishing Gradients



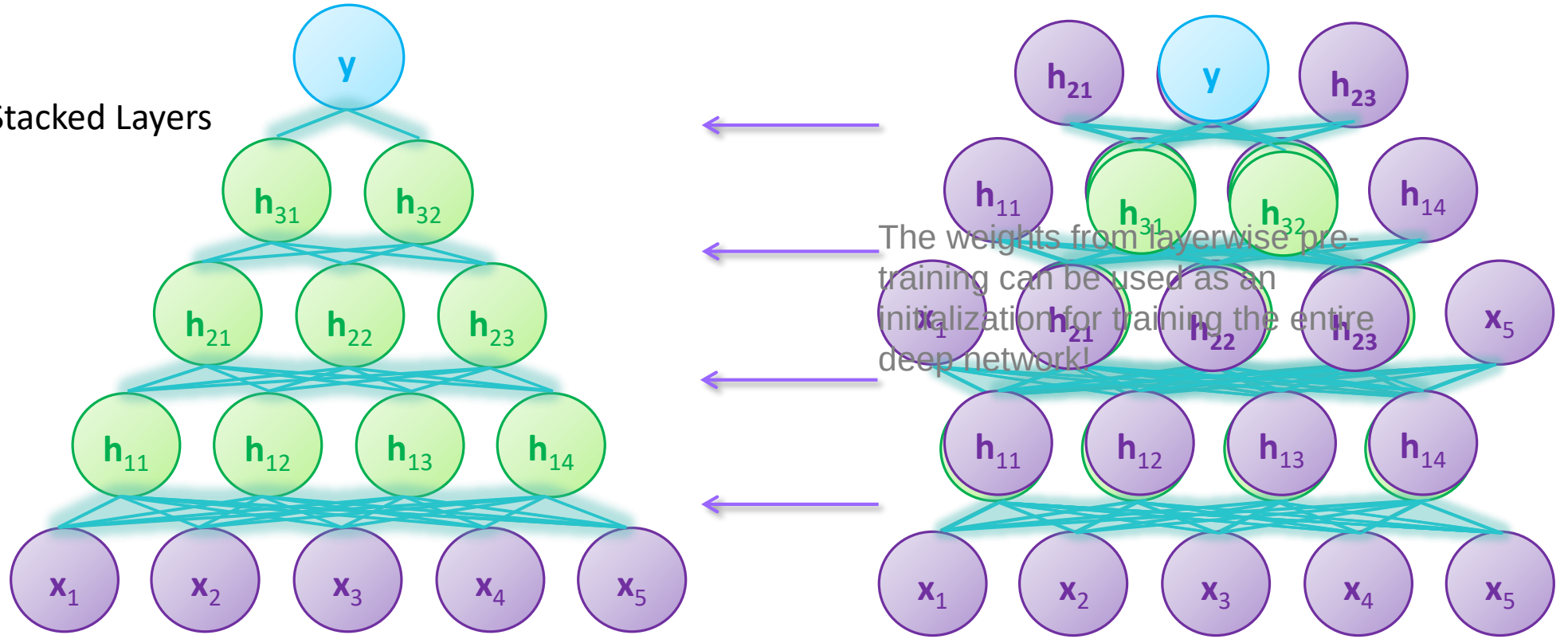
- To calculate the gradient for training a deep neural network, the weights and their layer-wise gradients must be multiplied together many times
- Initialization with large random numbers often causes neurons to become saturated, leading to extremely small gradients
- Initialization with small random 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

Training & Stacked Layers



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:

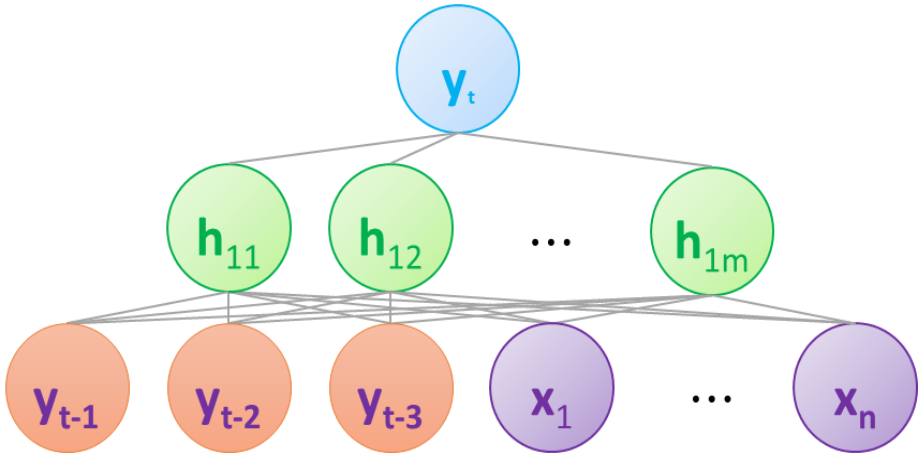
- Benchmarking
- Grid search
- Design of experiments (DOE)
- Genetic Algorithms (GA)
- Bayesian Approaches

Neural Networks for Time Series & Sequences

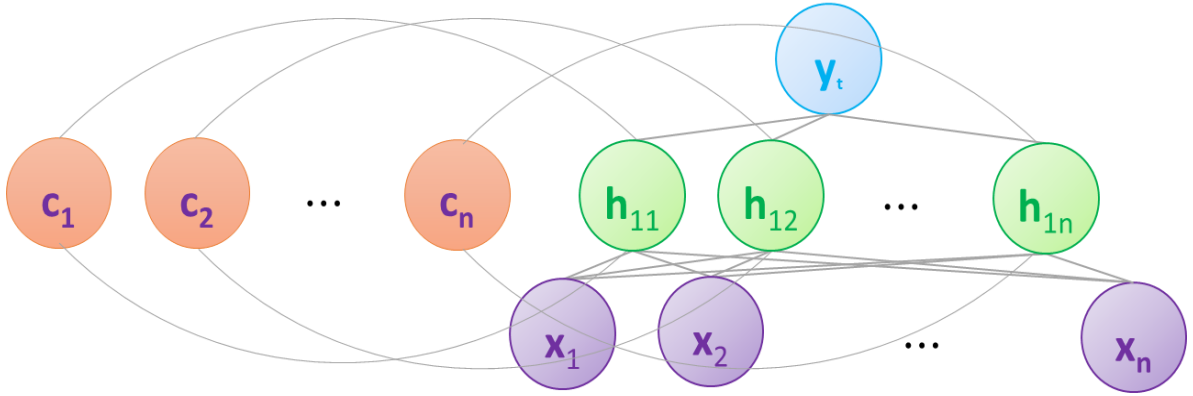
- Neural networks and other machine learning algorithms are occasionally more accurate than traditional time series methods on a **difficult series**
  - No distributional assumptions
  - Less interpretable
- Traditional time series approaches do not model **sequences of categorical targets**

Applications to Time Series & Sequences

NARX and RNN

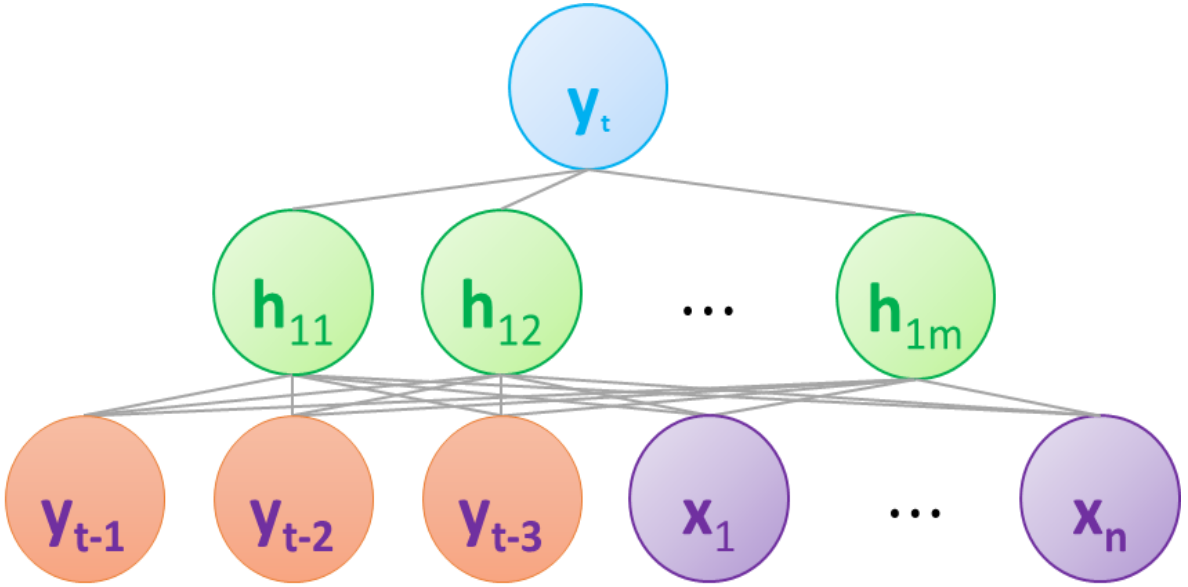


Nonlinear AutoRegressive network with eXogenous inputs (NARX)



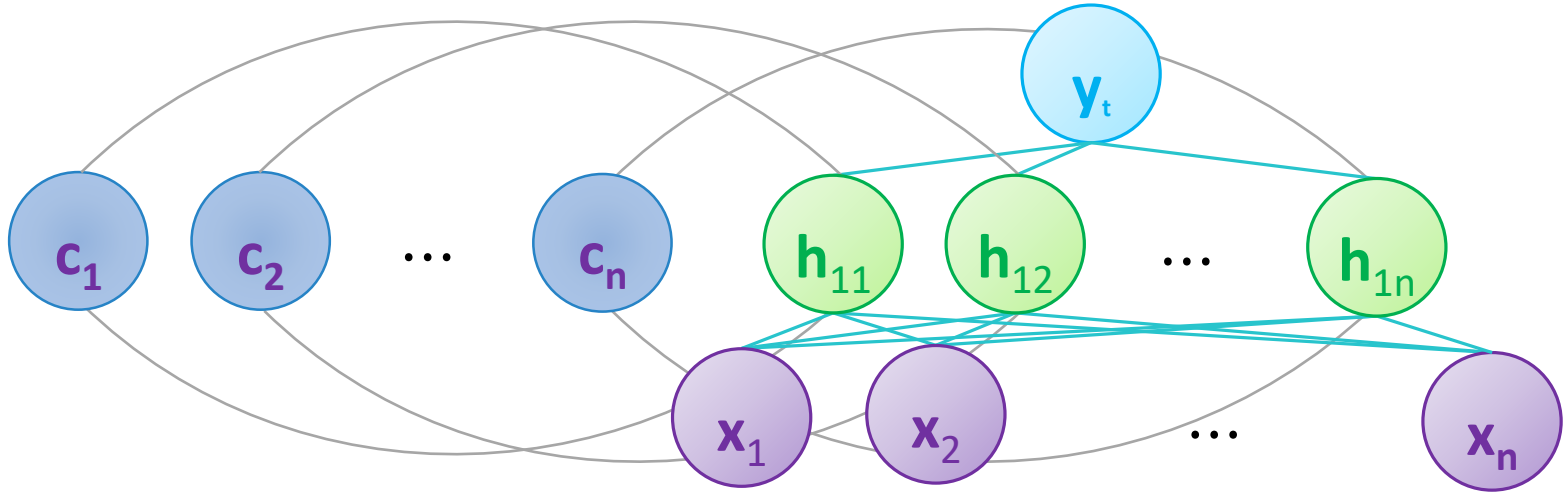
Recurrent Neural Network (RNN)

Nonlinear AutoRegressive Network with eXogenous Inputs (NARX)



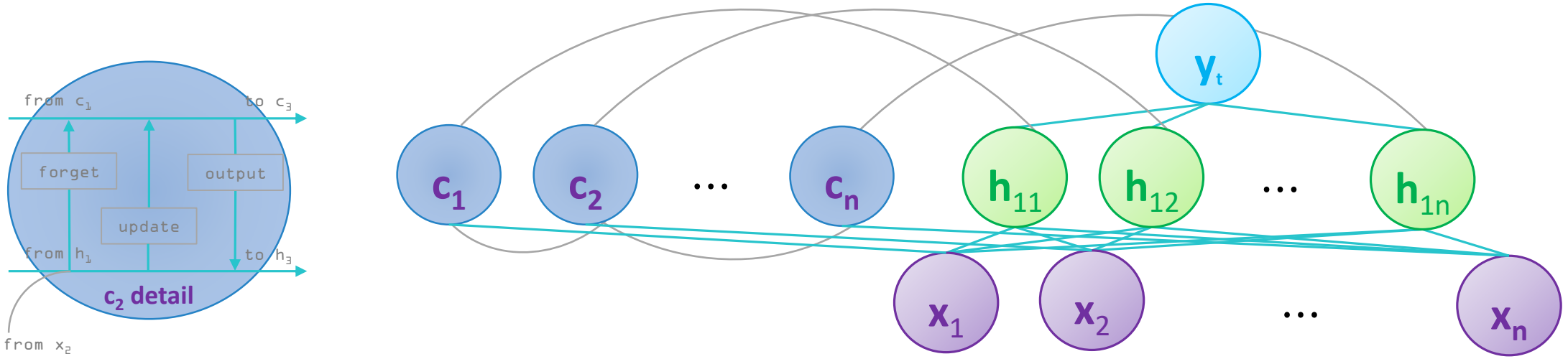
A NARX model uses time lags of the target variable and other input variables to predict the next value in a series of interval targets or a sequence of categorical targets

Recurrent Neural Network (RNN)



- A RNN model attempts to optimally generate features from the past events (remember past events) and use these features along with conventional model inputs to predict series of interval targets or a sequence of categorical targets
- The Basic RNN structure 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 model enabling significant breakthroughs in sequence learning

Applications to Text Mining

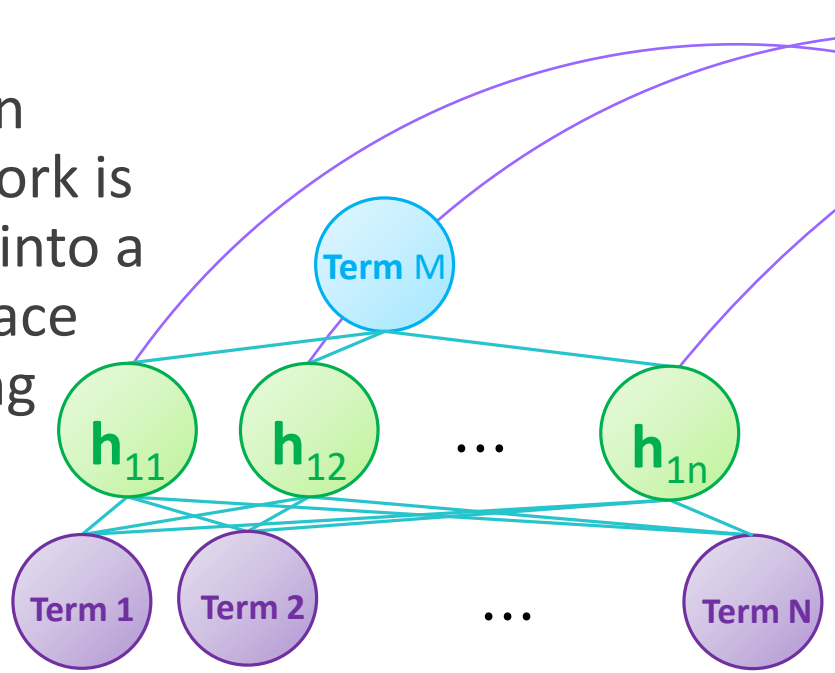
- Bag of words is a global, **matrix factorization** model capable representing prominent ideas in a group of documents; it assumes independence between terms
- Term embeddings are local, contextual models capable of representing relationships between words that can be generated by neural networks

king − man + women = queen

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

Embeddings: Analogous to 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



	Term 1	Term 2	Term 3	Term 4	Term 5	...
Document 1	0	0	0	1	0	...
⋮	⋮	⋮	⋮	⋮	⋮	⋱

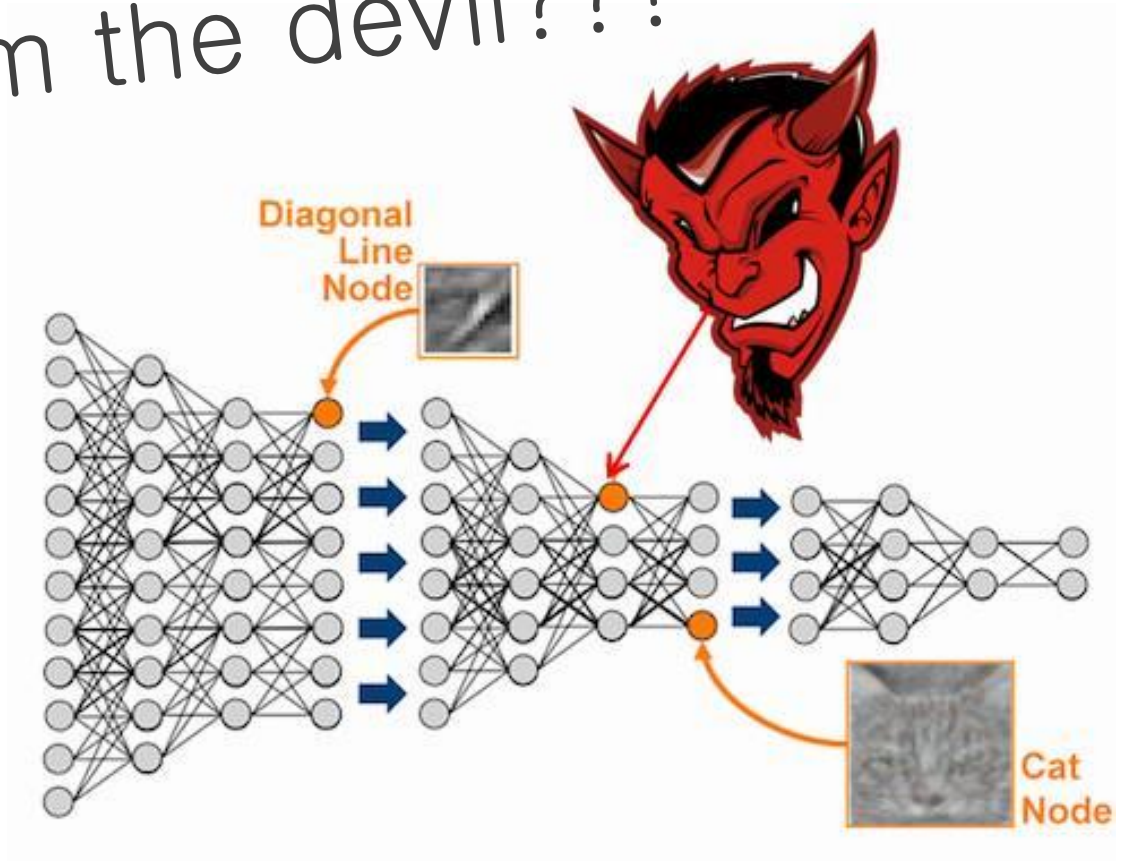
	Factor 1	Factor 2	...	Factor N
Term 1	1.304	0.582	...	0.892
Term 2	0.897	0.843	...	0.885
Term 3	0.745	1.129	...	1.002
Term 4	0.921	0.962	...	0.714
⋮	⋮	⋮	...	⋮

Each row vector represents a term (distributed)

Dense, fixed-length vectors for each term in the corpus, often used with RNN or hidden Markov models (HMM)

The shocking truth revealed!

Is deep learning from the devil???



http://www.kdnuggets.com/2015/10/deep-learning-vapnik-einstein-devil-yandex-conference.html

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