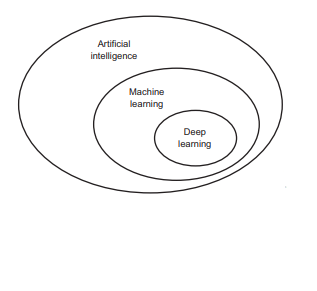
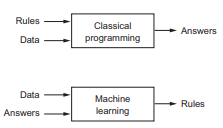
**Deep Learning**

**Artificial Intelligence:** The effort to automate intellectual tasks normally performed by humans.

**Machine Learning:** 

Training a machine or a model to mimic and perform rational calculations mirroring humans but more efficient.

* Input data points
* Examples of expected output
* Way to measure good algorithm output

**Deep** in DL refers to layered representations learning and hierarchical representations learning. Trained by ***neural networks*** models.

**Neural network:** i.e., *neurobiology.* (Myth: deep learning models are models of the brain)

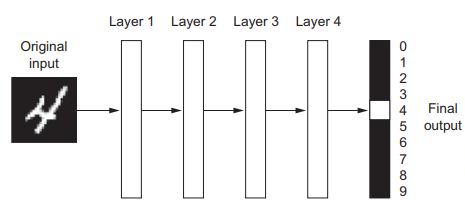
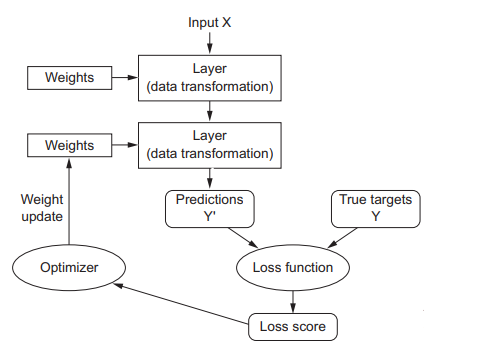


Fig: Deep neural network for digit classification

* A multistage way to learn data representations

**Working mechanisms**

Parameterized by weights (parameterization) Loss function to measure the quality of the networks output. Feedback signals to adjust weights.

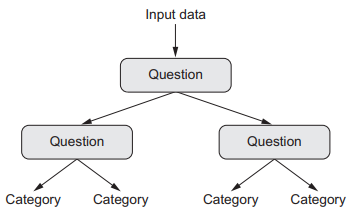


**Probabilistic modeling:** applying statistics to data analysis. Popular algorithms: Naïve bayes algorithm, logistic regression (logreg)

**Kernel methods:**  group of classification algorithms. E.g., support vector machine (SVM) developed by Vladmir vapnik in 1990s. Finds good decision boundaries by: -

1. Data hyperplane representation of decision boundary in a (high dimensional representation).
2. Maximizing the distance between hyperplane and closest data points from each class i.e., maximizing the margin for generalization

**Decision trees:** Flowchart-like structures



**Feature engineering:** manually engineering good layers for data representation. DL automates this step.

Characteristics of DL in learning data: *incremental, layer-by-layer. Simplicity, Scalability, versatility, and reusability.*

**Keras** makes deep learning as easy as manipulating LEGO bricks.

**MNIST** solving considered “Hello world” of DL.



*From keras.datasets import mnist*

*(train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data()*

* **Class:** category in a classification problem
* **Samples :** data points
* **Label:** Class associated with a specific sample

**DATA REPRESENTATION FOR NEURAL NETWORKS**

Data stored in multidimensional numpy arrays called tensors (basic data structure)

Tensor is a container for data, primarily numbers. They are a gernlization of matrices to an arbituary number of dimensions.

**Scalars (OD tensors):** tensor containing only one number, e.g float32 , float 64

*>>> Import numpy as np*

*>>> X= np.array(12)*

*>>> x.ndim*

*0*

**Vectors (1D tensor)**an array of numbers, one axis

*>>> x = np.array([12, 3, 6, 14])*

*>>> x.ndim*

*1*

**Matrices (2D tensors)** array of vectors, two axes (rows and columns).

*>>> x = np.array([[5, 78, 2, 34, 0],*

*[6, 79, 34, 5, 12],*

*[7, 2, 12, 5, 3]])*

*>>> x.ndim*

*2*

**3D tensors and higher-dimensional tensors:** Cube of numbers

*>>> x = np.array ( [ [ [ 6, 78, 2, 13 ],*

*[5, 34, 1, 61 ],*

*[1, 45, 9, 0 ] ],*

*[ [ 16, 8, 5, 3 ],*

*[5, 34, 1, 33 ],*

*[1, 45, 9, 0 ] ],*

*[ [ 6, 78, 2, 13 ],*

*[2, 34, 1, 81 ],*

*[1, 95, 9, 0 ] ] ] )*

*>>> x.ndim*

*3*

Tensor is defined by:-

* number of axes (rank)
* Shape (tuple of integers)
* Data type

*Q. Why char tensor is a rarity?*

Tensors live in preallocated, contiguous memory segments and strings being variable length would preclude the use of this implementation.

**Real-world examples of data tensors**

* + Vector data
  + Timeseries data or sequence data
  + Images \_ 4D tensors of shape
  + Video \_ 5D tensors of shape

**Tensor slicing:**  selecting specific elements in a tensor

*>>> my\_slice = train\_images[10:100]*

*>>>print(my\_slice.shape)*

*(90, 28, 28)*

*\*negative indices indicate a position relative to the end of the current axis*

**Tensor operations:**  addition, multiplication

**ELEMENT WISE OPERATIONS:**

RELU operation, Addition

*Import numpy as np*

*Z = x+y # element wise addition*

*Z = np.maximum(z, 0.) # element wise relu*

**Broadcasting**

1. Axes are added to the smaller tensors to match the ndim of the larger tensor
2. The smaller the tensor is repeated alongside these new aces to match the full shape of the larger tensor

With broadcasting we can generally apply to two tensor element wise operations if one tensor has shape (a, b, … n, n+1, … m) and the other has shape (n, n+1, … m). Broadcasting will automatically happen for axes a through n-1.

*Import numpy as np*

*X = np.random.random((64, 3, 32, 10))*

*Y = np.random.random((32, 10))*

*Z = np.maximum(x, y)*

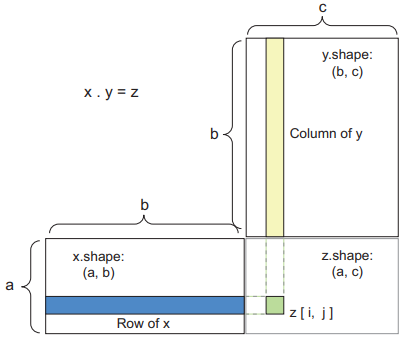
*Output will have shape (64, 3, 32, 10) like x*

**Tensor dot:** tensor product. Combines entries in the input tensors.

*Import numpy as np*

*Z = np.dot(x, y)*

**Matrix dot-product box diagram**



(a, b, c, d) . (d,) -> (a, b, c)

(a, b, c, d) . (d, e) -> (a, b, c, e)

**Transposition:** special reshaping case, exchanging rows and columns.

*>>> x = np.zeroes((300, 20))*

*>> x = np.transpose(x)*

*>>> print(x.shape)*

*(20, 300)*

**Gradient based Optimization Steps:**

1. Draw a batch of training sasmples x and corresponding targets y
2. Run networks on x (forward pass) to obtain predictions y\_pred
3. Compute loss of the network on the batch, measure of mismatch between y\_pred and y
4. Update all weights of the network in a way that slightly reduces the loss on batch

**Gradient:** derivative of tensor flow operation.

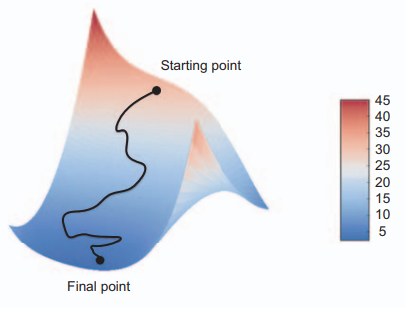
*Y\_pred = dot(W, x)*

*Loss\_value = loss(y\_pred, y)*

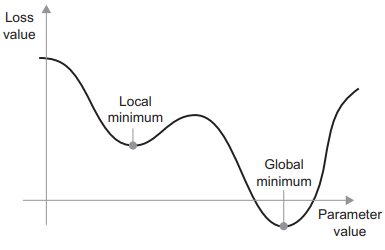
**Stochastic gradient descent:**

1. Draw batch of training samples x corresponding target y
2. Run network on x to obtain predictions y\_pred
3. Compute loss of network on batch
4. Computer gradient of loss with regards to the network parameter (backward pass)
5. Move parameters little in opposite direction from gradient.

*Stochastic refers to the fact that each batch of data is drawn at random*



**Gradient descent down a 2D loss surface (two learnable parameters)**



**Backpropagation algorithm:** suppose network f contains three stensor operations a, b, c with weight W1, W2, W3

F(w1, w2, w3) = a(w1, b(w2, c(w3)))

With the laws of calculus, we can use chain rule

*F(g(x)) = f’(g(x) \* g’(x)*