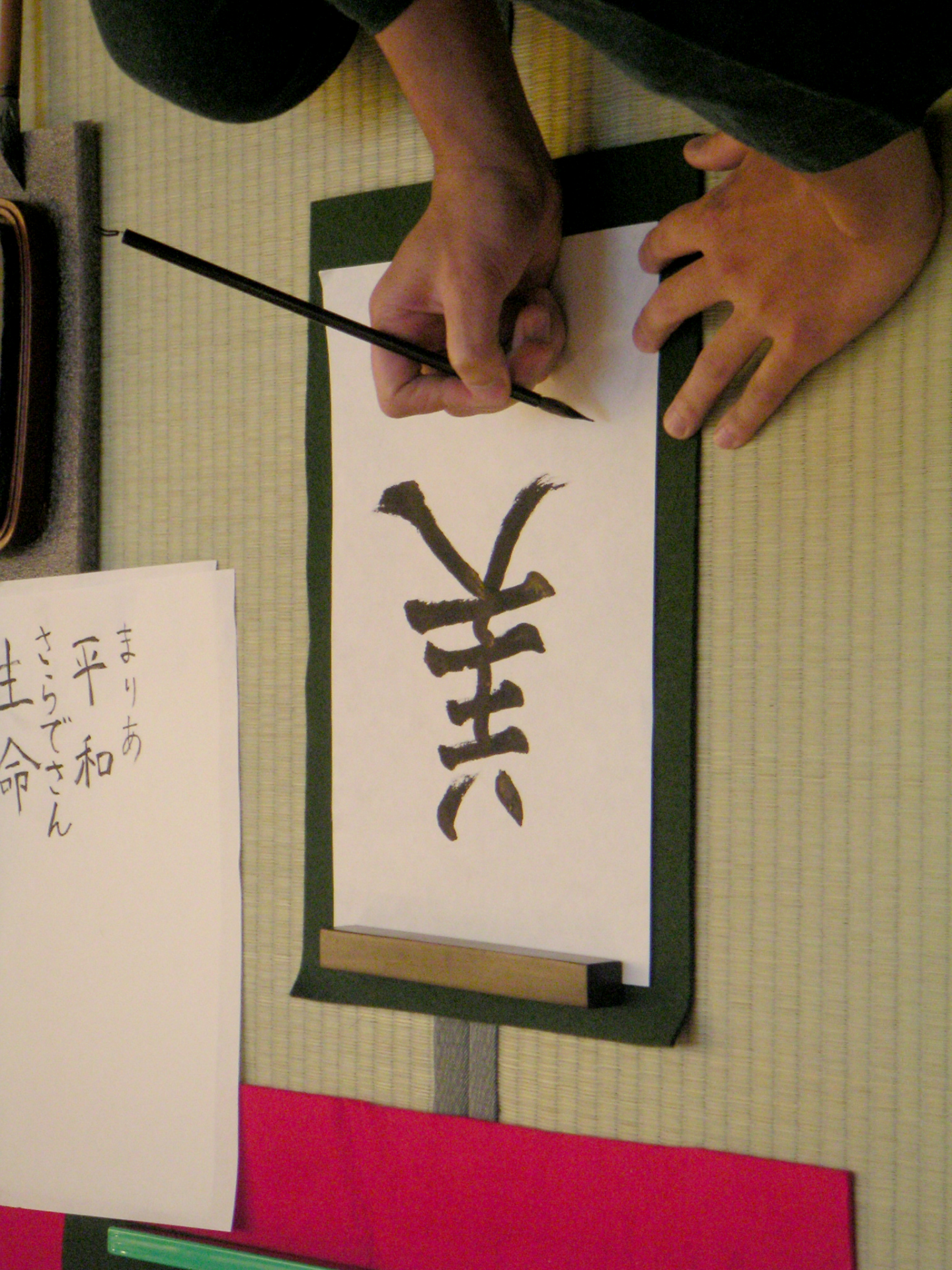
Machine Learning in q/kdb+
Teaching KDB to Read Japanese

Mark Lefevre

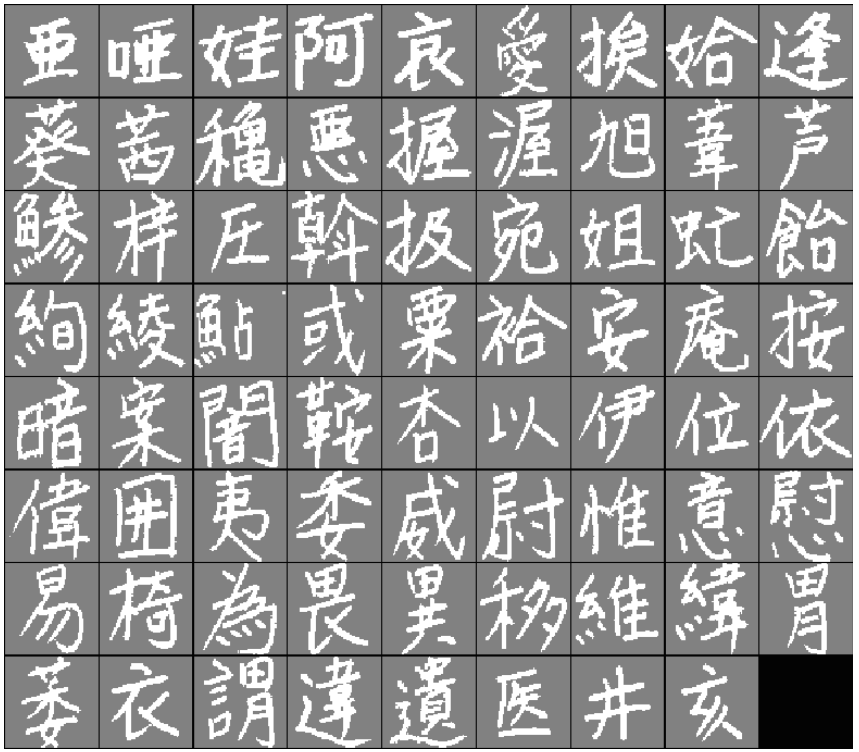
Algorithmic Quantitative Analyst


まりあ
平和
さらでさん
生命

Handwritten Japanese

Hiragana

Kanji



Machine Learning Introduction

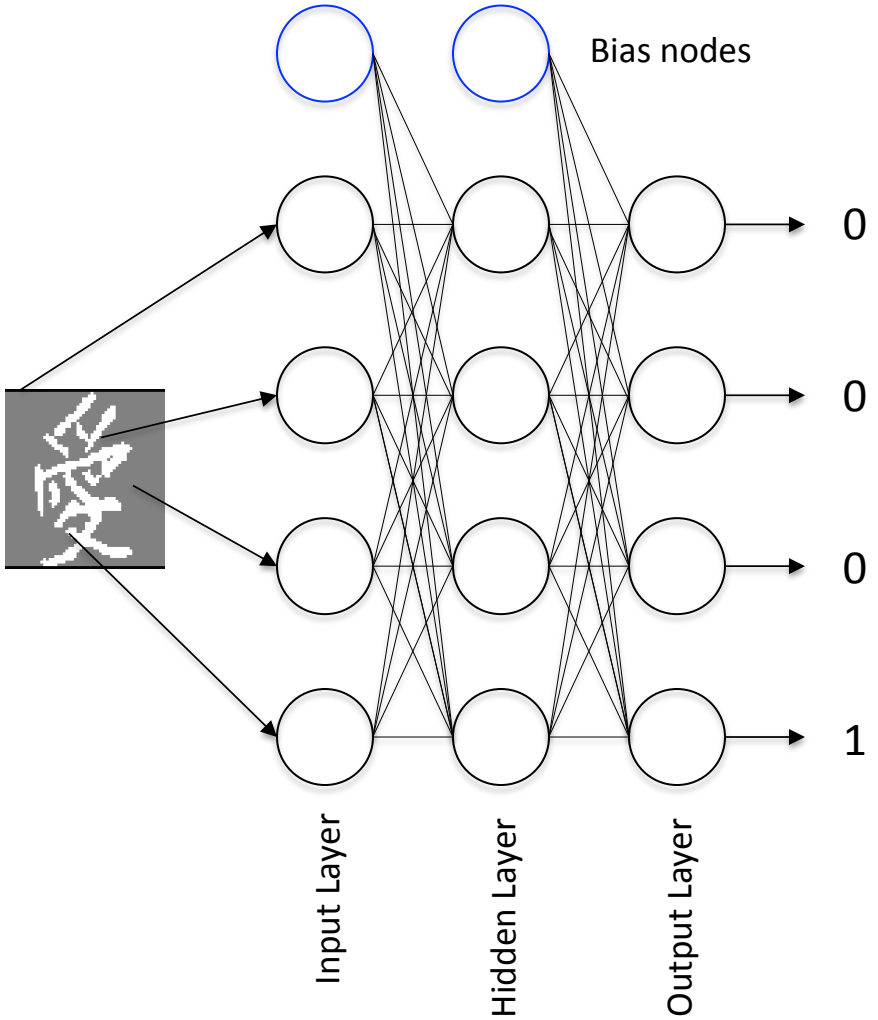
- ML algorithms can be grouped by learning style
  - Supervised Learning
  - Unsupervised Learning
  - Reinforcement Learning
- Or, alternatively, by similarity
  - Regression
  - Decision Trees
  - Neural Networks
  - Etc.

Supervised Learning

- Uses a training dataset with known inputs and outputs
- Uses an a-priori defined model to describe the type of transformation
- Objective is to optimize the model parameters to give the “best” mapping between input and output on a training set
- Typical types of problems are classification and regression

Neural Networks

- Outperform all other approaches on most complex classification problems (speech, handwriting recognition, etc.)
- Can represent non-linear boundaries and arbitrarily complex functions
- Excellent at extracting the correct features



Perceptron (Artificial Neuron)

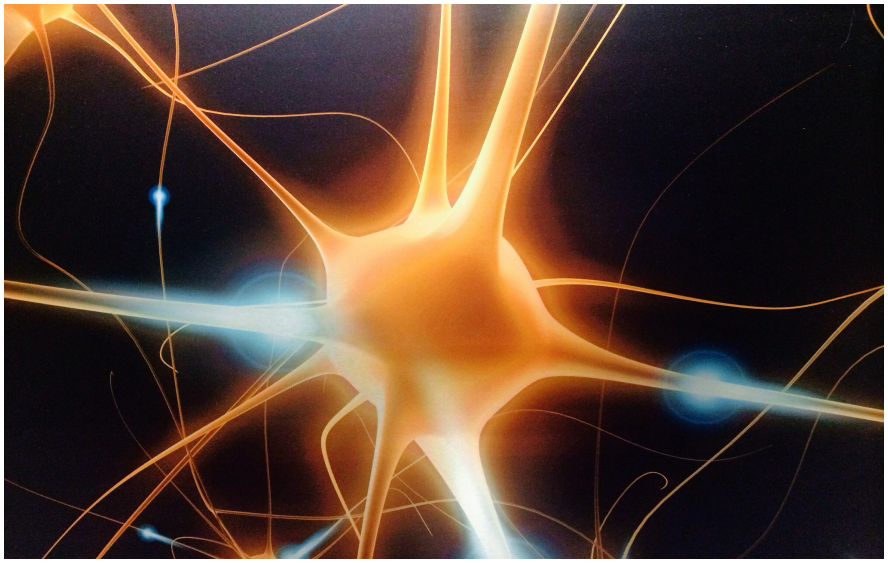
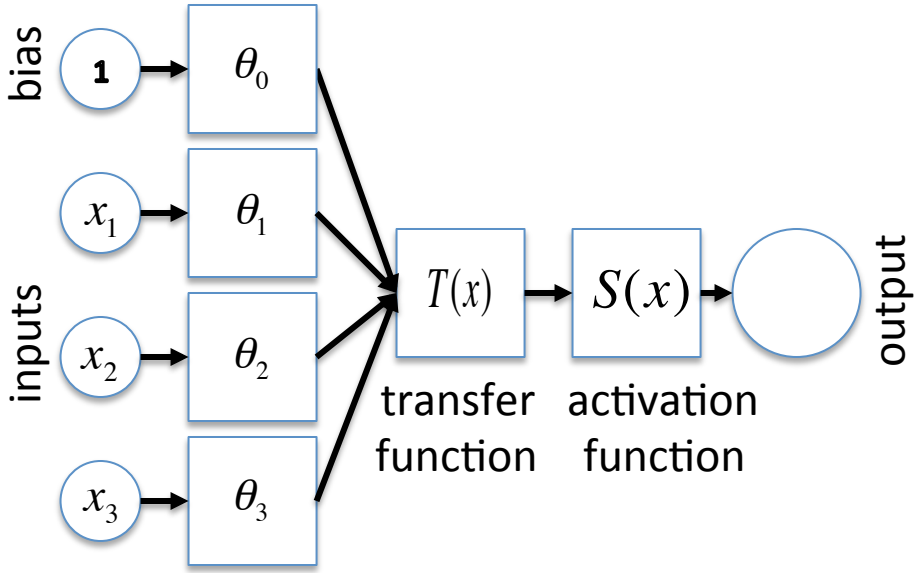


Photo Bryan Jones©, flickr



$$x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \Theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

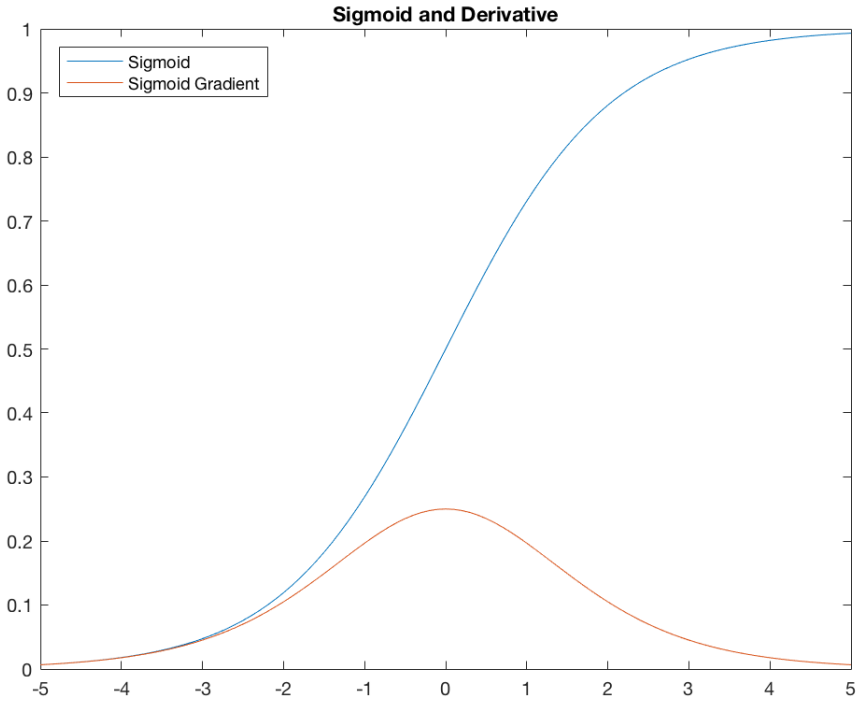
$$T(x) = \theta^T x$$

```
(flip theta)$x
```

Activation Functions

- Model neuron's action potential firing
- Desirable properties
  - Non-linear
  - Continuously differentiable
  - Monotonic
  - Fast
- Alternatives
  - Tanh
  - Rectified Linear Unit (ReLU)
  - Soft plus

- Sigmoid Function (aka Standard Logistic Function)



Sigmoid Function and Derivative

- Sigmoid

$$S(x) = \frac{1}{1+e^{-x}}$$

```
sigmoid:{1.0%1.0+exp neg x};
```

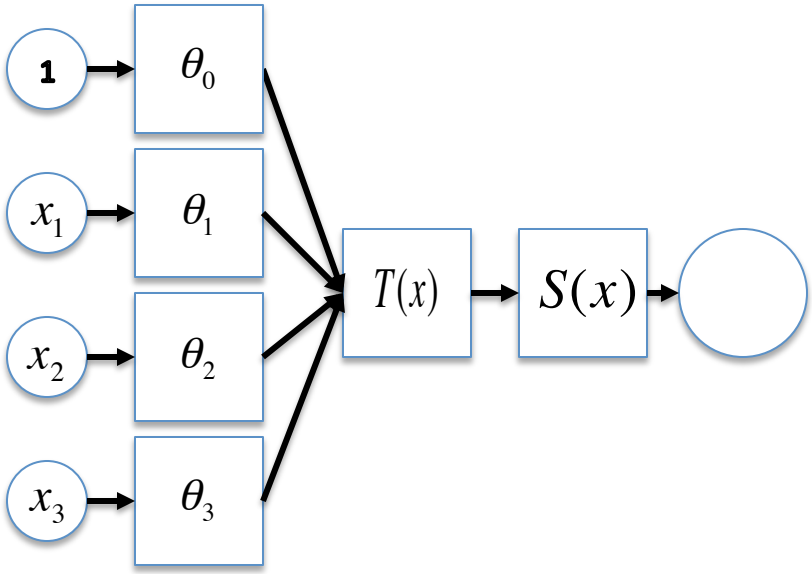
- Gradient

$$S'(x) = \frac{e^{-x}}{\left(1+e^{-x}\right)^2} = S(x)\left(1-S(x)\right)$$

```
sigmoidGradient:{g:1.0%1.0+exp neg x; g*1-g};
sigmoidGradient:{{x*1-x}sigmoid[x]};
```

Perceptron Mathematical Representation

$$x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \Theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$



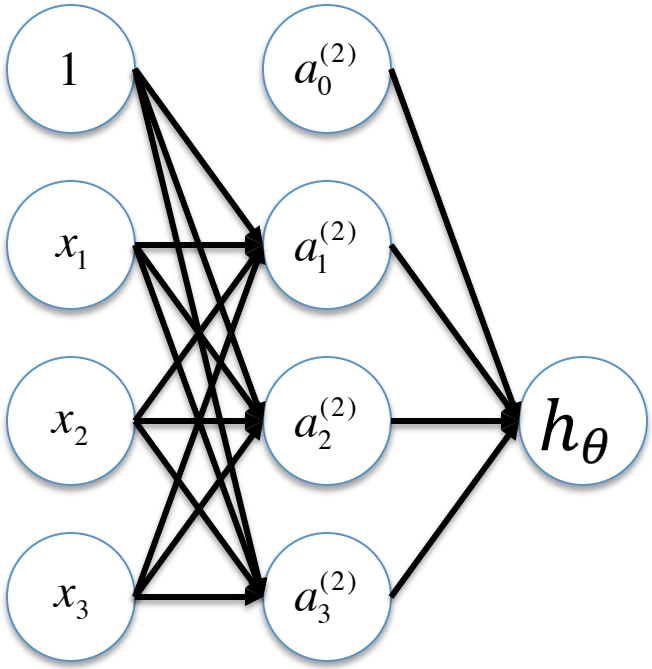
$$T(x) = \theta^T x \qquad S(x) = \frac{1}{1+e^{-x}}$$

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

```
perceptron:{[x;theta]{1.0%1.0+exp neg x}(flip theta)$x};
```

Feedforward Neural Network

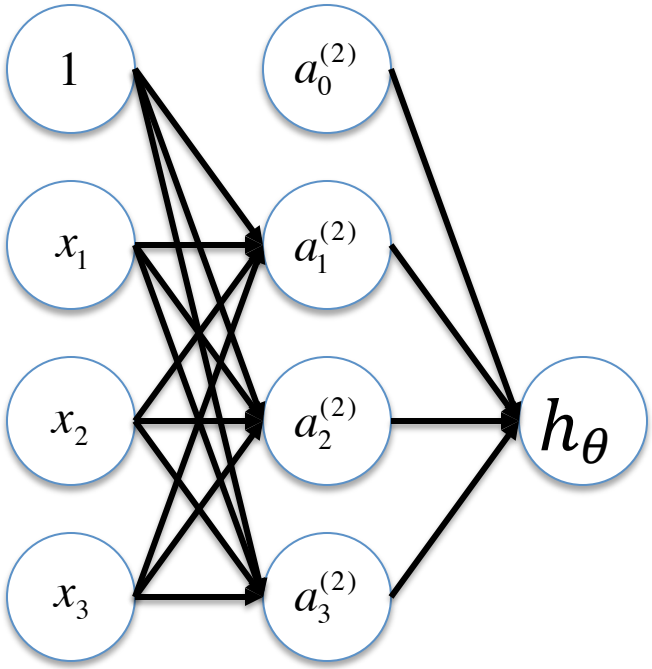
- Example NN Consists of
  - Input layer (1 bias)
  - Hidden layer (1 bias)
  - Output layer
- Directed acyclical path



$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right)$$
$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right)$$
$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right)$$
$$h_{\theta(x)} = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

Vectorized Feedforward

$$x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \Theta = \begin{bmatrix} \theta_{10}^{(1)} & \theta_{11}^{(1)} & \theta_{12}^{(1)} & \theta_{13}^{(1)} \\ \theta_{20}^{(1)} & \theta_{21}^{(1)} & \theta_{22}^{(1)} & \theta_{23}^{(1)} \\ \theta_{30}^{(1)} & \theta_{31}^{(1)} & \theta_{32}^{(1)} & \theta_{33}^{(1)} \end{bmatrix}$$



$$z^{(2)} = \Theta^{(1)} x$$

$$a^{(2)} = g(z^2)$$

$$a_0^{(2)} = 1$$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$

$$h_\theta = a^{(3)} = g(z^3)$$

Back-propagation

- How do we train the neural network?
- Cost function

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log\, h_\theta\left(x^{(i)}\right)_k + \left(1 - y_k^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)_k\right)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}\left(\Theta_j^{(l)}\right)^2$$

- Need to calculate

$$J(\Theta)$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$$

Practical KDB ML Application

- 41 sets of 3036 handwritten characters
  - 71 hiragana
  - 2965 kanji
- Generated by binarization of a greyscale image
- 64 x 63 = 4032 pixels
- Used 40 sets as training data
  - Only a few samples for each character
- Held out 1 set for test data

- Building a classifer with 3036 outputs nodes seemed excessive
  - Started with hiragana only
  - Then used same configuration on first 71 kanji
  - Same neural network can be repurposed by simply changing weights
  - Configuration is 4032-252-71
  - Used regularization term with lambda of 1

Demo and Hiragana Results

- After 50 iterations in optimization
  - 89.04% Training
  - 92.96% Test
  - 5 misclassifications
  - (6, 46, 52, 55, 56)
- After 150 iterations in optimization
  - 100.00% Training
  - 98.59% Test
  - 1 misclassification
  - 55

Demo and Kanji Results

- After 50 iterations in optimization
  - 93.41% Training
  - 81.69% Test
  - 13 misclassifications
- After 150 iterations in optimization
  - 100.00% Training
  - 84.51% Test
  - 11 misclassifications

Conclusion

- Briefly introduced machine learning, the concept of supervised learning and neural networks
- Demonstrated the implementation of several neural network algorithms in q/kdb+ for recognizing handwritten Japanese characters
- Appropriate utilization of vectorized operations on lists and matrix operations result in much simpler and efficient code
- Hopefully, this has provide an interesting look at the opportunities to utilize q/kdb+ in new ways

About Me

- Mark is currently consulting at one of the largest banks in Tokyo as an algorithmic quantitative analyst developing high-performance algorithmic trading systems on the e-FX desk. Prior to moving to Japan, he worked in London for Unicredit on the Equity-Linked Origination desk creating convertible bonds for European corporates, consulted in the US on e-commerce analytics and worked for several high-tech software companies.
- Earlier in his career, he worked for Mitsubishi Semiconductor America designing semiconductors and a startup developing a DSP. He then moved into applications engineering for an Electronic Design Automation (EDA) company and, subsequently, internet software companies in CA and Europe.
- Mark has a bachelors degree in Electrical Engineering and Computer Science from Duke University, a masters degree in Computer Engineering from North Carolina State University and an MBA in Quantitative Finance from the Wharton School of Business. He recently completed a Certificate in Quantitative Finance (CQF).
- He dreams of the day when he can create software without encountering a single type error