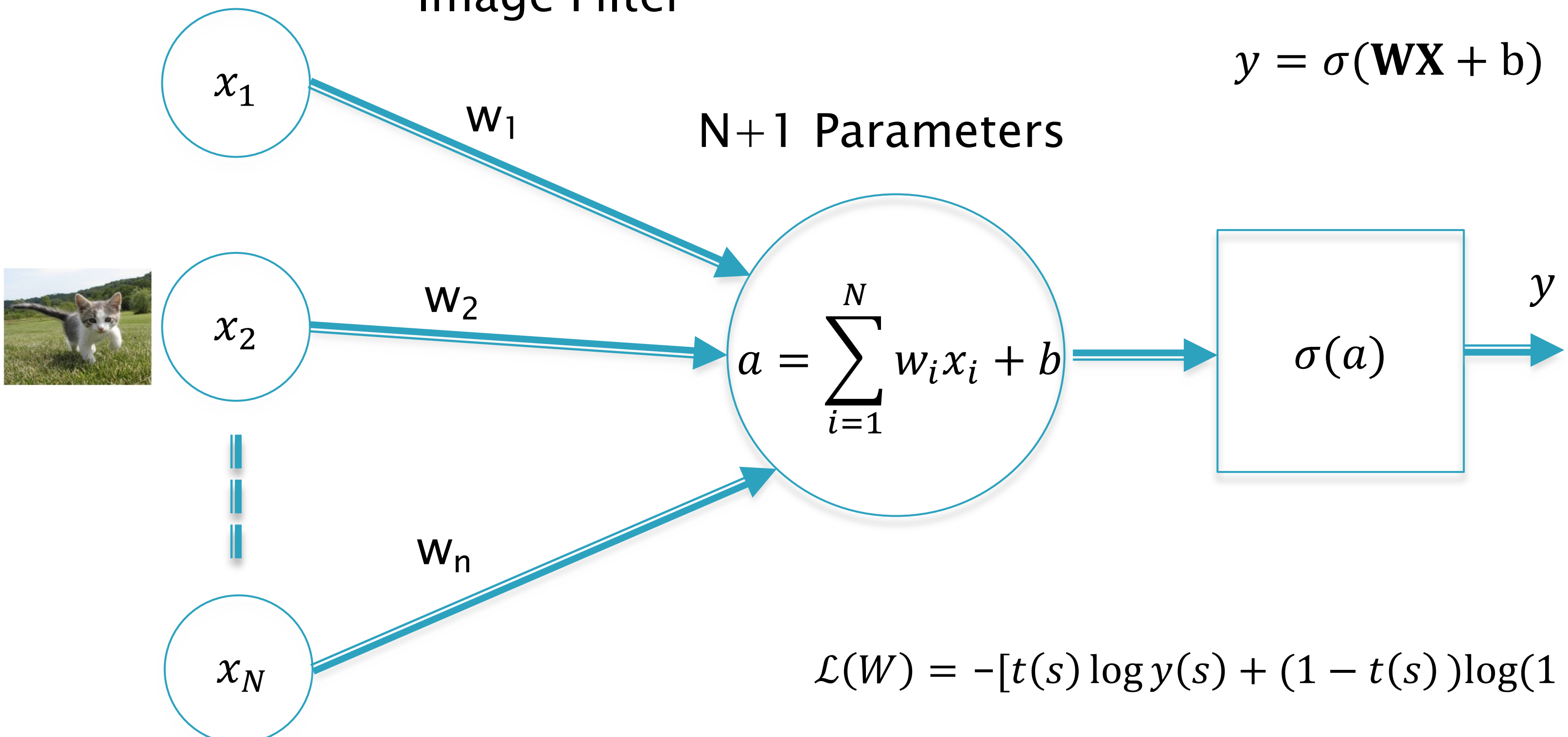
Dense Feed Forward Networks

Lecture 4

Subir Varma

Linear System with Two Classes

$$y = \sigma(\mathbf{WX} + \mathrm{b})$$



$$\mathcal{L}(W) = -[t(s)\log y(s) + (1 - t(s))\log(1 - y(s))]$$

$$\boxed{w \leftarrow w - \eta \frac{\partial L}{\partial w}}$$

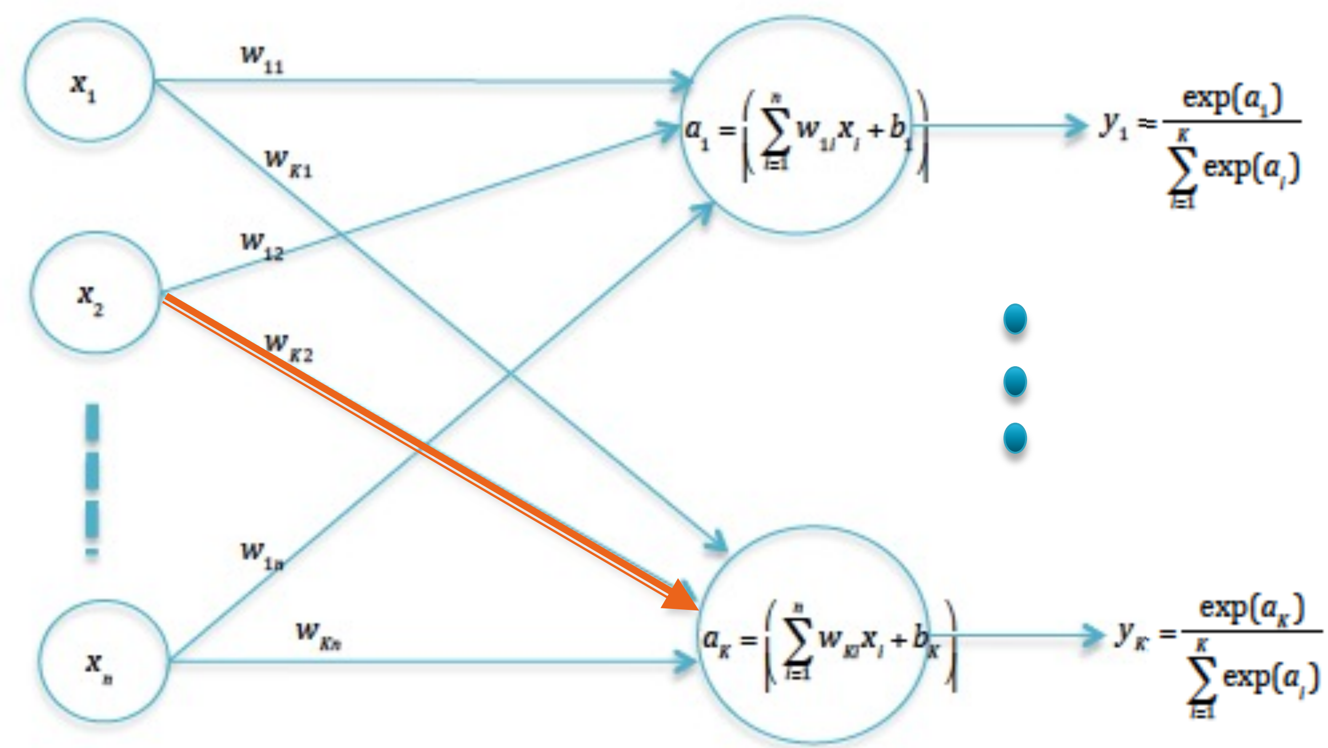
$$\frac{\partial L}{\partial w_i} = x_i(s)[y(s) - t(s)]$$

$$w_i \leftarrow w_i - \eta x_i(s)[y(s) - t(s)]$$

Learning Iteration

Linear Classification Models with K Classes

K-ary Classification



Input Logits Predictions

K Filters operating in parallel

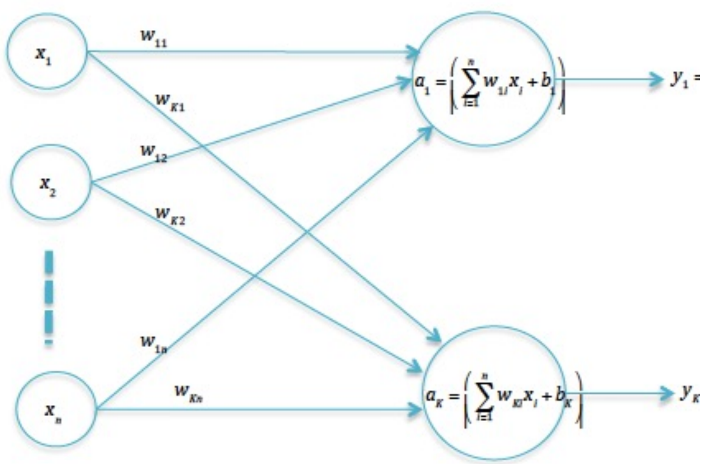
The SoftMax Classifier

$$y_k = h_k(a_1, \ldots, a_K) = \frac{e^{a_k}}{\sum_{j=1}^{K} e^{a_j}}$$

Sum of all K outputs is 1
Results in a probability distribution

- Appropriate for K-ary classification networks

K-ary Classification



$$a_k = \sum_{i=1}^{N} w_{ki}x_i + b_k$$

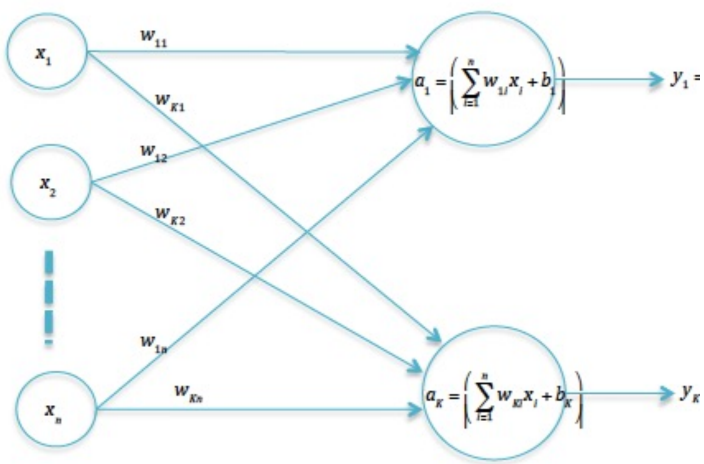
$$y_k = h_k(a_1, \ldots, a_K) = \frac{e^{a_k}}{\sum_{j=1}^{K} e^{a_j}}$$

NK+K Parameters

$$\begin{pmatrix} a_1 \\ \vdots \\ a_K \end{pmatrix} = \begin{pmatrix} w_{11} & \cdots & w_{1N} \\ \vdots & \ddots & \vdots \\ w_{K1} & \cdots & w_{KN} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix} + \begin{pmatrix} b_1 \\ \vdots \\ b_K \end{pmatrix}$$

$$\boxed{\begin{aligned} A &= WX + B \\ Y &= h(A) \end{aligned}}$$

K-ary Classification with Input in Batches



K Filters operating in parallel

$$a_k = \sum_{i=1}^{N} w_{ki} x_i + b_k$$

$$y_k = h_k(a_1, \ldots, a_K) = \frac{e^{a_k}}{\sum_{j=1}^{K} e^{a_j}}$$

Feed B inputs into the model together

$$\begin{pmatrix} a_1 & \ldots & a_{1B} \\ & \vdots & \\ a_K & \ldots & a_{NB} \end{pmatrix} = \begin{pmatrix} w_{11} & \cdots & w_{1N} \\ \vdots & \ddots & \vdots \\ w_{K1} & \cdots & w_{KN} \end{pmatrix} \begin{pmatrix} x_{11} & \ldots & x_{1B} \\ & \vdots & \\ x_{N1} & \ldots & x_{NB} \end{pmatrix} + \begin{pmatrix} b_1 \\ \vdots \\ b_K \end{pmatrix}$$

$$\boxed{\begin{aligned} A &= WX + B \\ Y &= h(A) \end{aligned}}$$

Loss Function

Loss Function for the s^{th} Sample

$$\mathcal{L}(s) = -\sum_{k=1}^{K} t_k\,(\mathrm{s})\log y_k(s)$$

Loss Function for the Entire Training Set

$$L(W) = -\frac{1}{M}\sum_{s=1}^{M}\sum_{k=1}^{K} t_k\,(s)\log y_k(\mathrm{s})$$

Gradient Calculation

$$w_{kj} \leftarrow w_{kj} - \eta \frac{\partial \mathcal{L}}{\partial w_{kj}}$$

Evaluate $\frac{\partial \mathcal{L}}{\partial w_{kj}}$, *where*

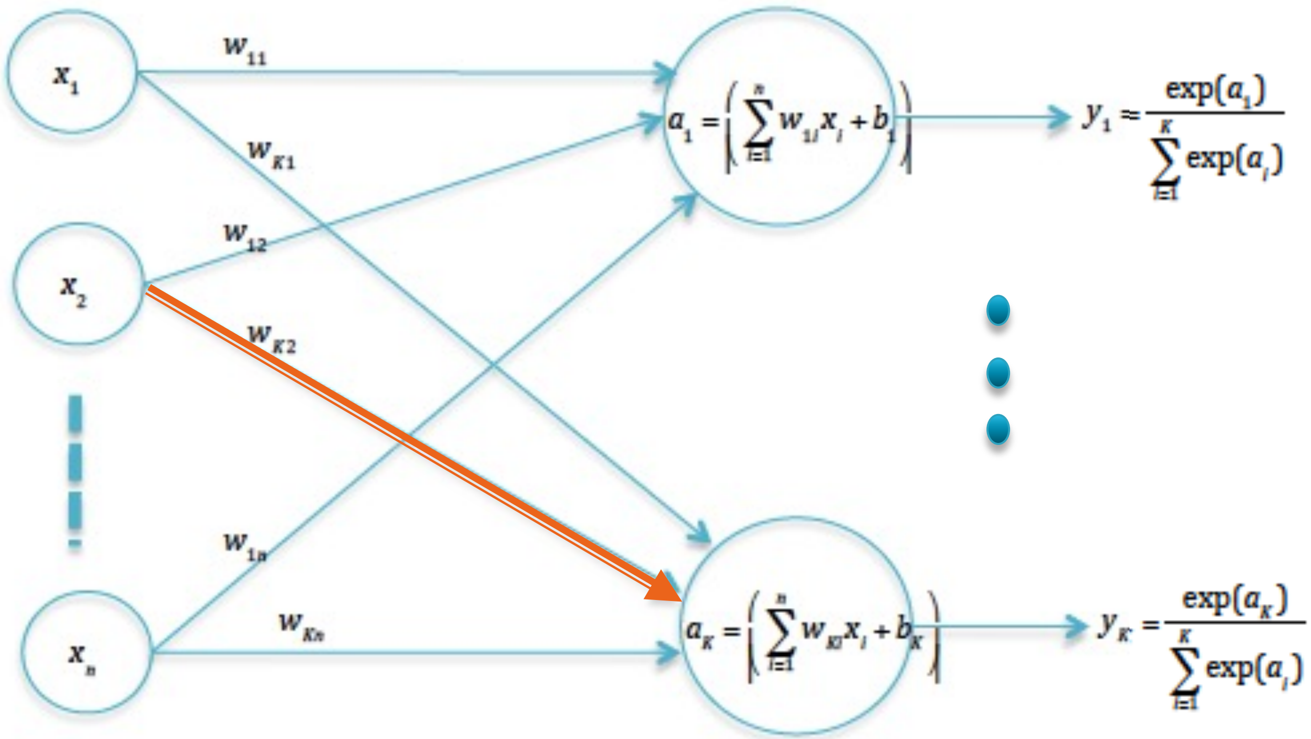
$\mathcal{L} = -\sum_{k=1}^{K} t_k \log y_k$, and

$$y_k = \frac{e^{a_k}}{\sum_{j=1}^{K} e^{a_j}}, \; a_k = \sum_{j=1}^{N} w_{kj}\, x_j + b_k$$

Answer:

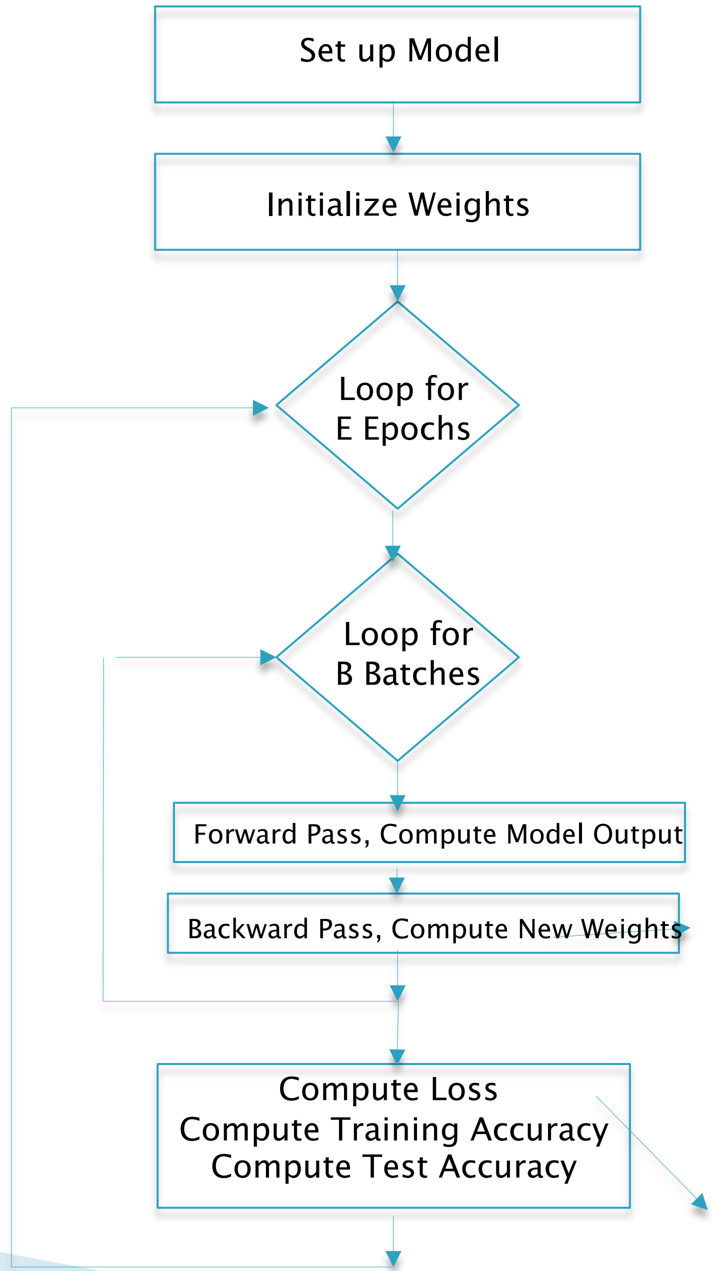
$$\frac{\partial \mathcal{L}}{\partial w_{kj}} = x_j\,(y_k - t_k)$$

K-ary Classification



If $\mathcal{L} = -\sum_{k=1}^{K} t_k \log y_k$
then
$$\frac{\partial \mathcal{L}}{\partial w_{kj}} = x_j \,(y_k - t_k)$$

Training Using Batch Gradient Descent For K-ary Classification



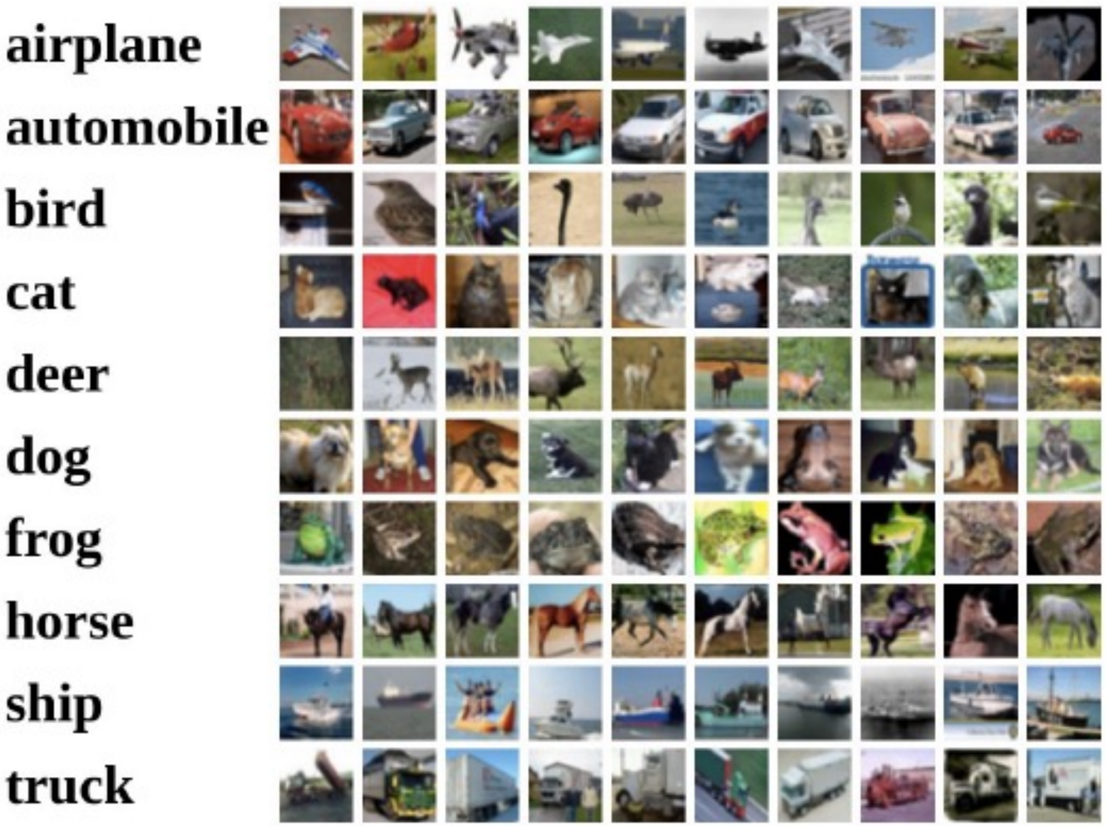
$$y_k = \frac{e^{a_k}}{\sum_{j=1}^{K} e^{a_j}}, \quad a_k = \sum_{j=1}^{N} w_{kj}\, x_j + b_k$$

$$w_{kj} \leftarrow w_{kj} - \frac{\eta}{B}\sum_{j=1}^{B} x_j(s)[y_k(s) - t_k(s)]$$

$$L = -\frac{1}{B}\sum_{s=1}^{B}\sum_{k=1}^{K} t_k(s)\log y_k(s)$$

Interpretation of the Linear Classifier

Interpretations of the Linear Classifier (with CIFAR-10)

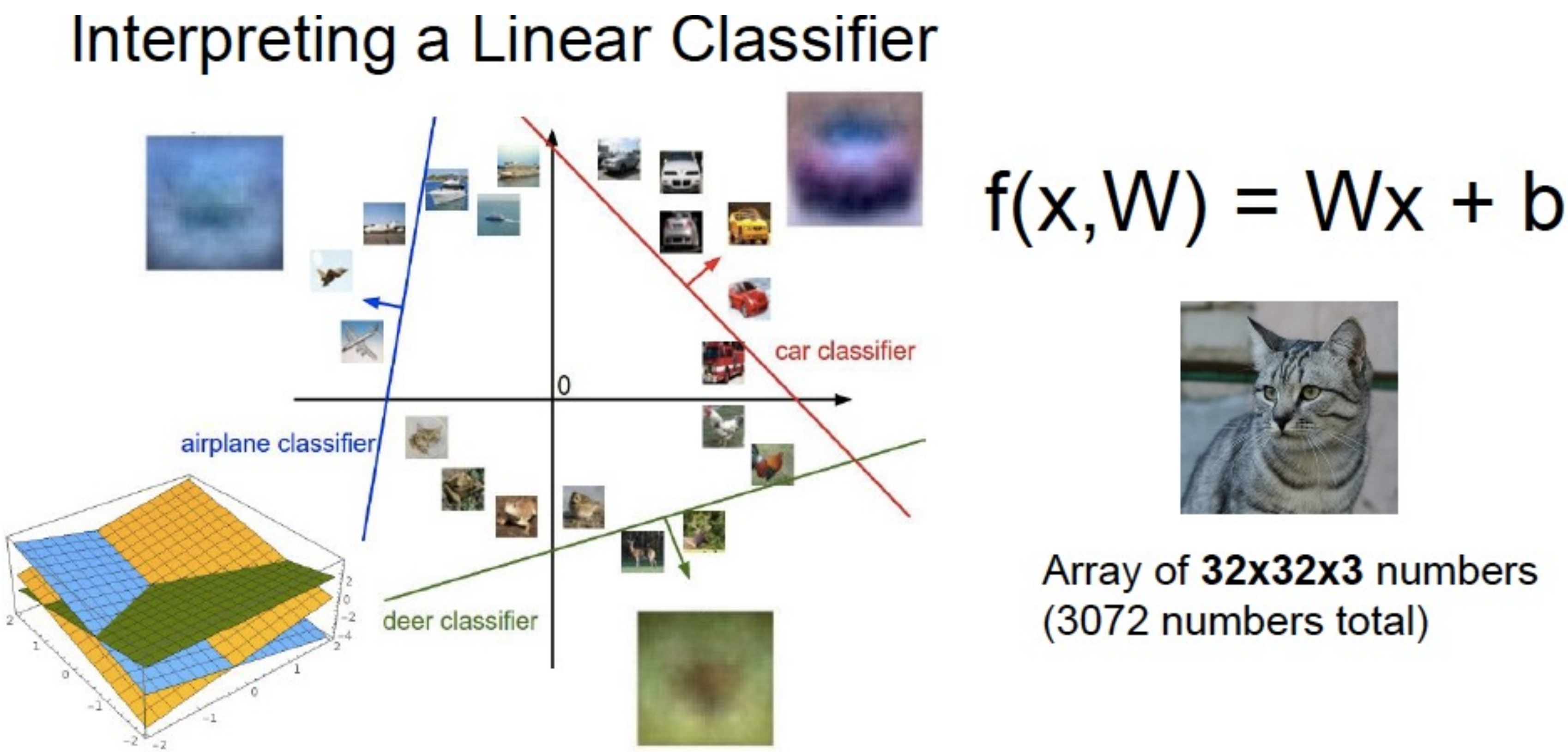


10 Categories

50,000 training images
each image is **32x32x3**

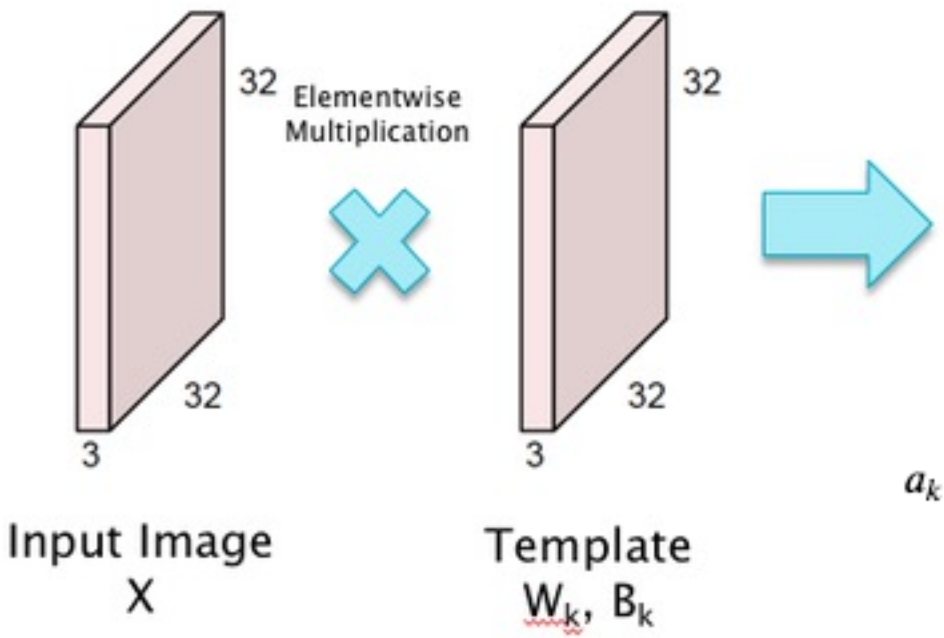
10,000 test images.

Interpretation: Hyperplane Separators



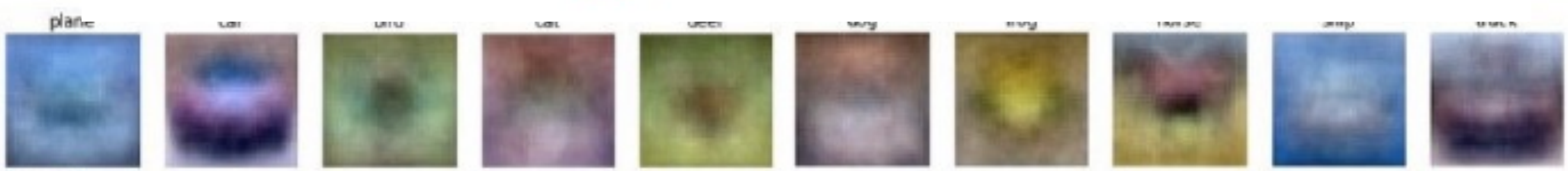
Interpretation of Weights as a Filter – Template Matching

10 Templates



Choose Category k for which $X*W_k + B_k$ is maximum

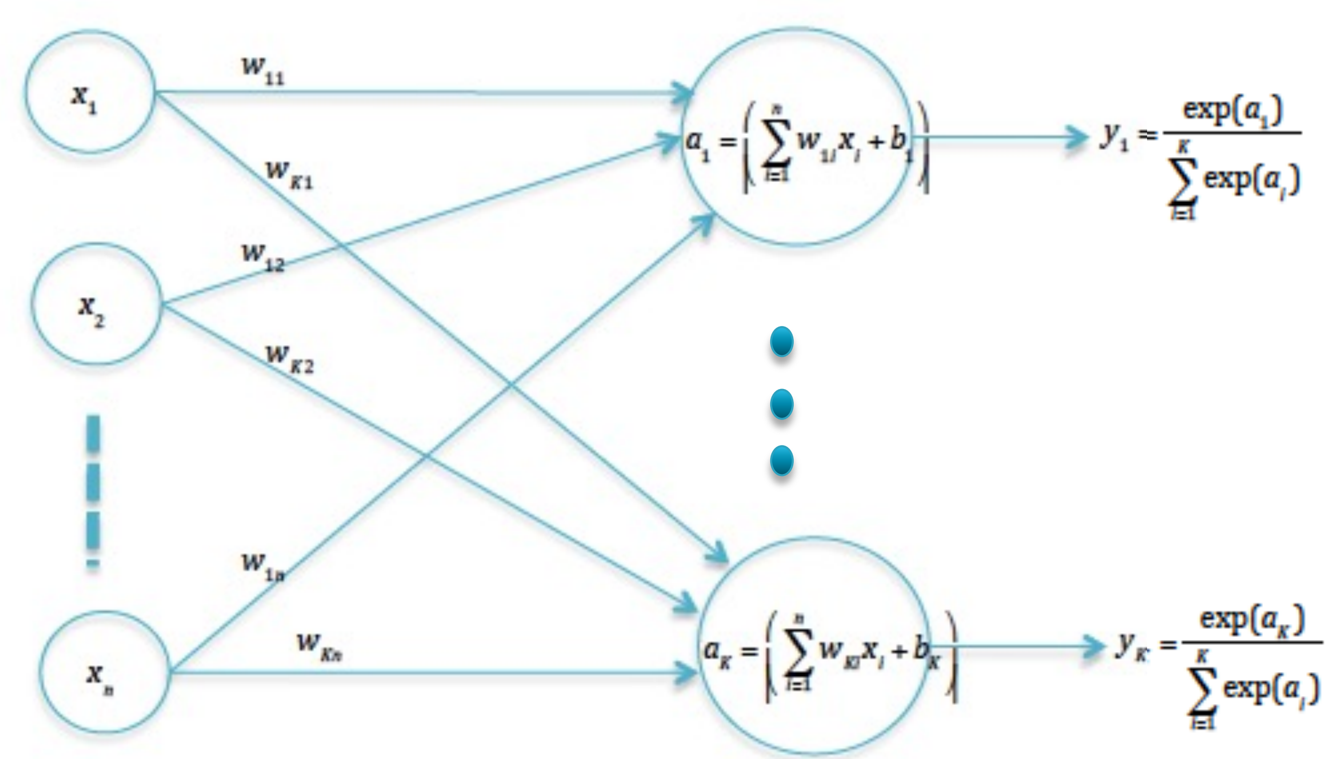
$$a_k = \sum_{i=1}^{3072} w_{ki} x_i + b_k, \;\; 1 \leq k \leq 10$$



The weight parameters are an image filter!

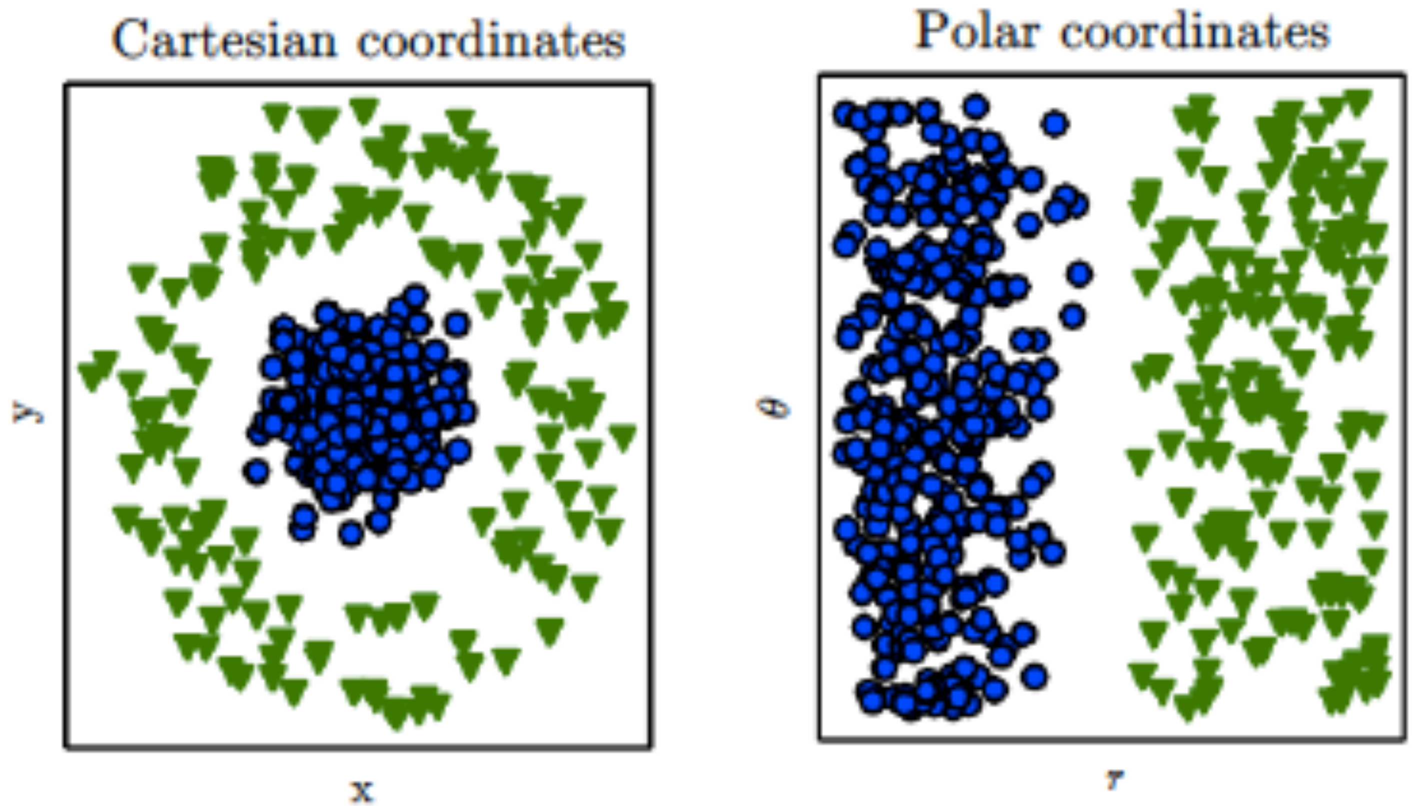
Dense Feed Forward Networks

Logistic Regression –Multiple Classes



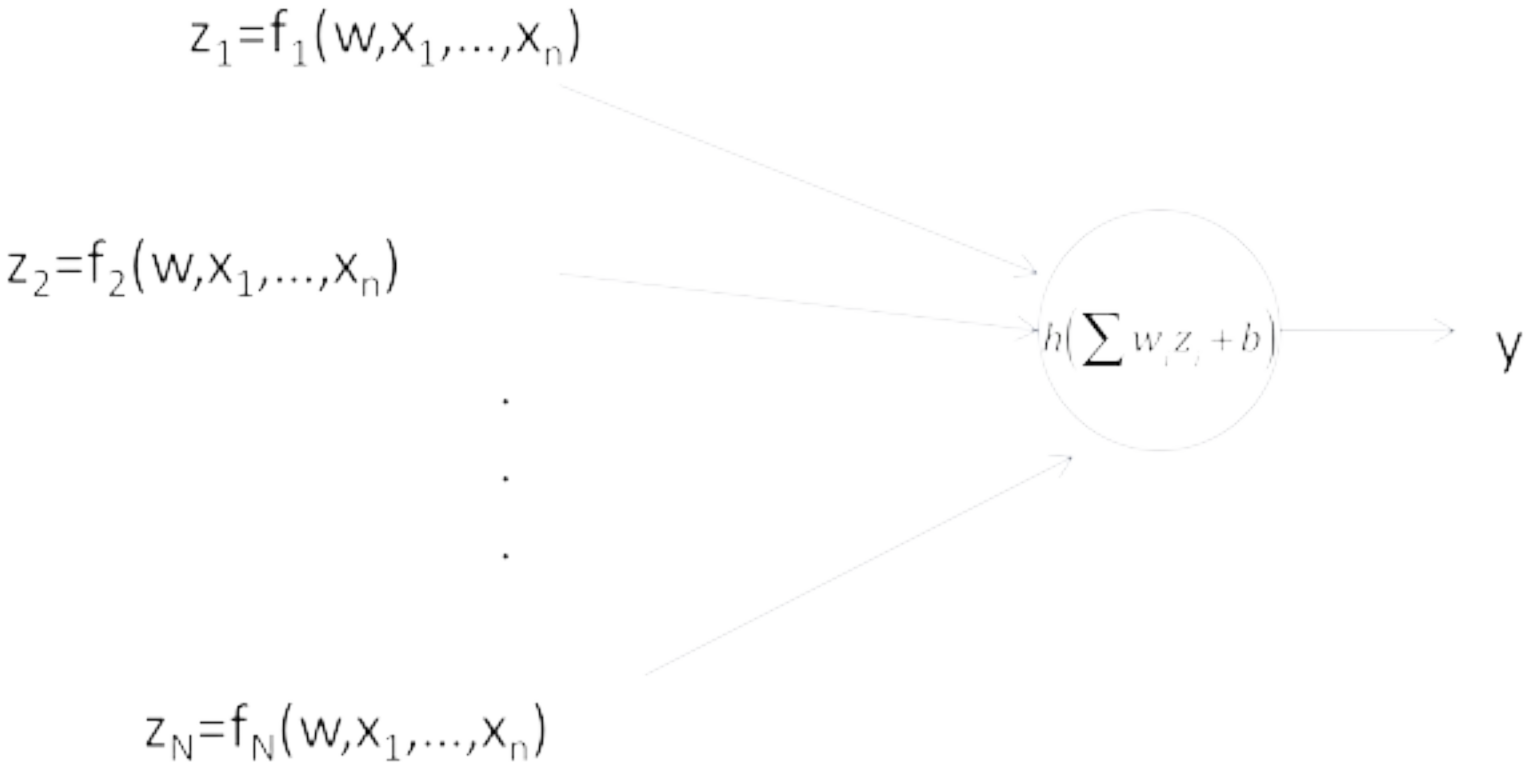
Works well only if the points ($x_1,\ldots,x_N$) are approximately linearly separable

The Importance of Representations

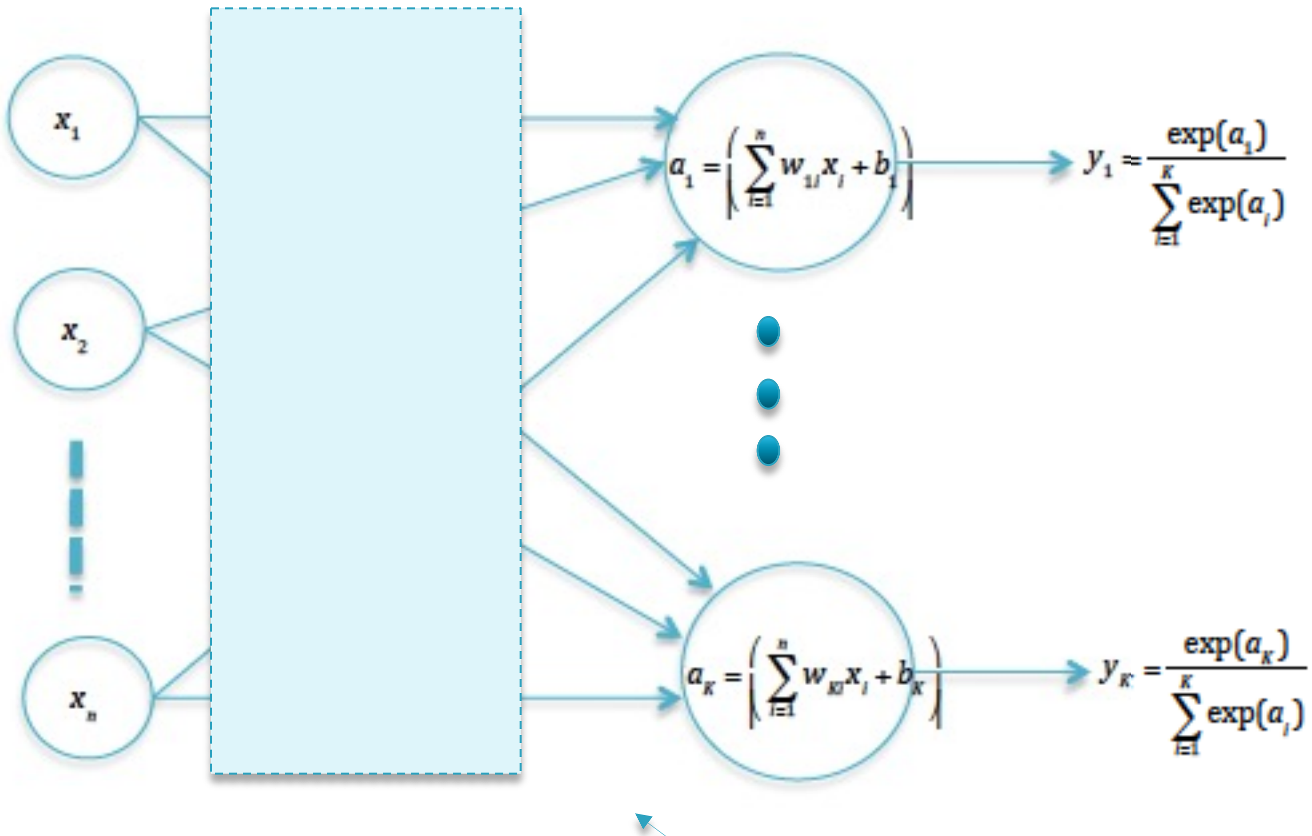


From "Deep Learning" by Goodfellow et.al.

Hand Tailored Representations

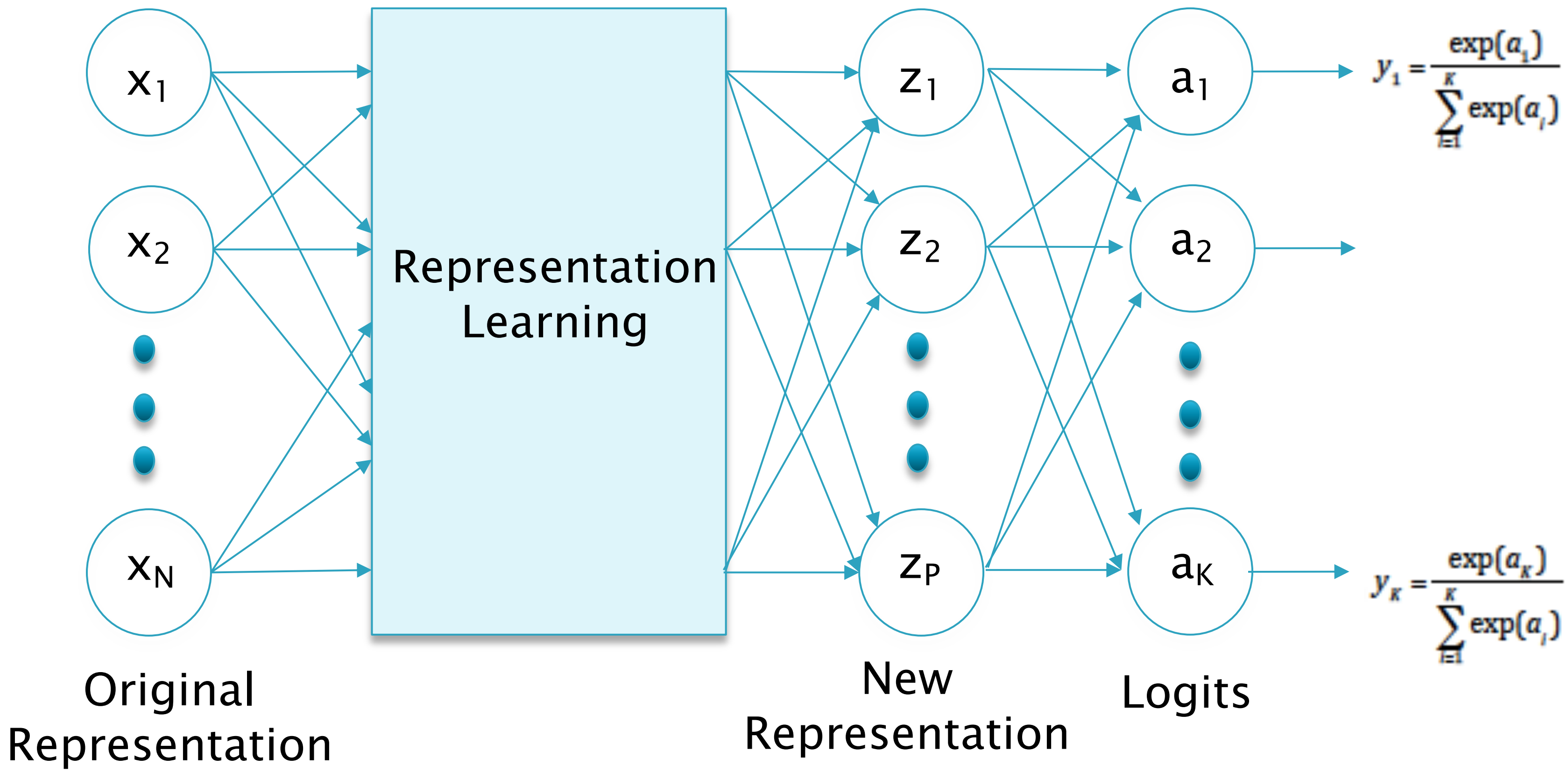


Transition to Deep Learning



Add a Module that can Learn Representations

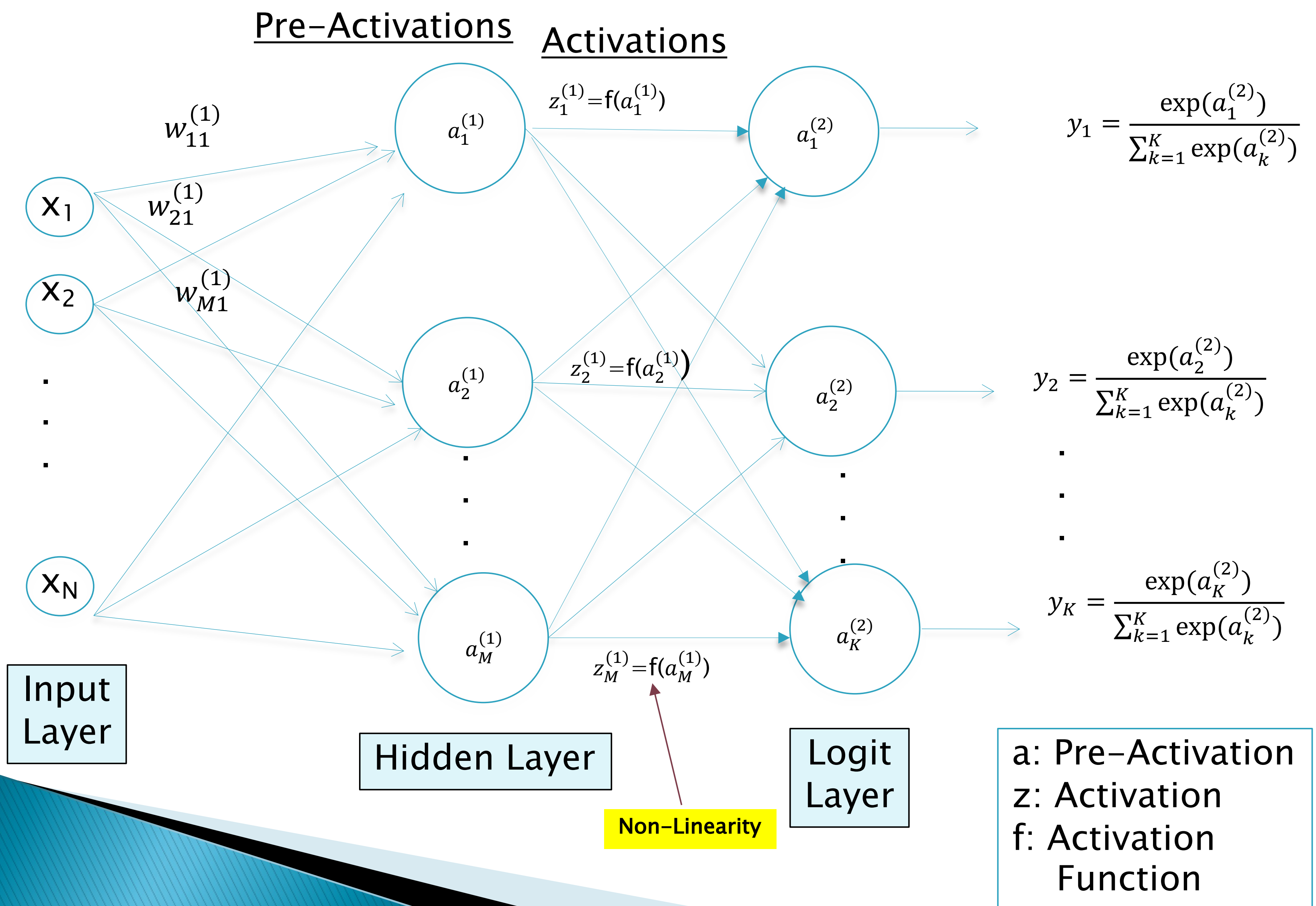
Transition to Deep Learning



$z_i = f_i(x_1, x_2, \ldots, x_n)$

What is a desirable property of a good representation?

A Dense Feed Forward Network

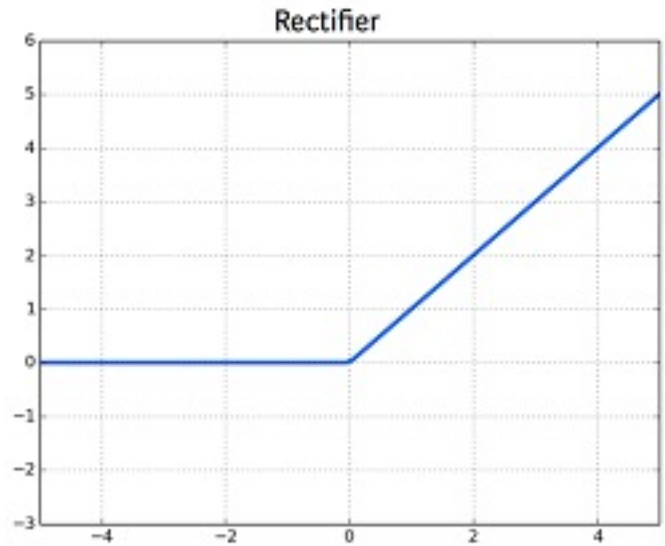


Benefits of Adding Hidden Layers

- Representations can be learnt as part of the training process, from the data itself
- The Classification problem is broken up into smaller parts, with each node in the Logit Layer responding to sub-parts of an image
- Provides a way to create more powerful models, by adding additional layers and/or additional hidden nodes per layer

Activation Functions: ReLU

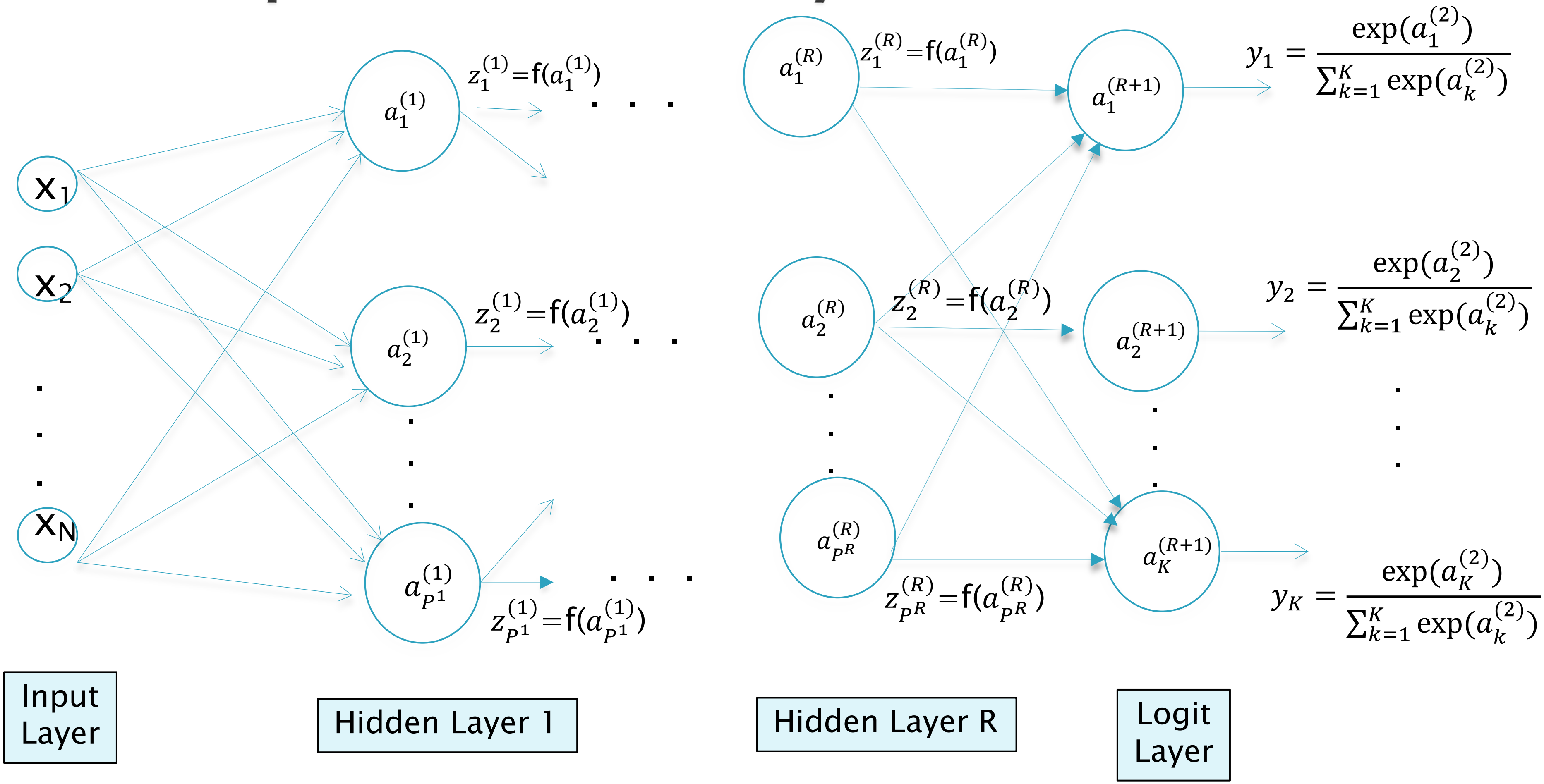
ReLU (Rectified Linear Activation)



$$RELU(x) = \begin{cases} 0 \; if \; x < 0 \\ x \; if \; x >= 0 \end{cases}$$

Default Choice
For Activation Function

Deep Feed Forward Network with Multiple Hidden Layers

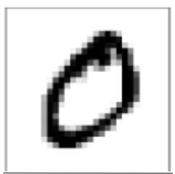


More layers increase the power of the representation learning
But: Training becomes harder

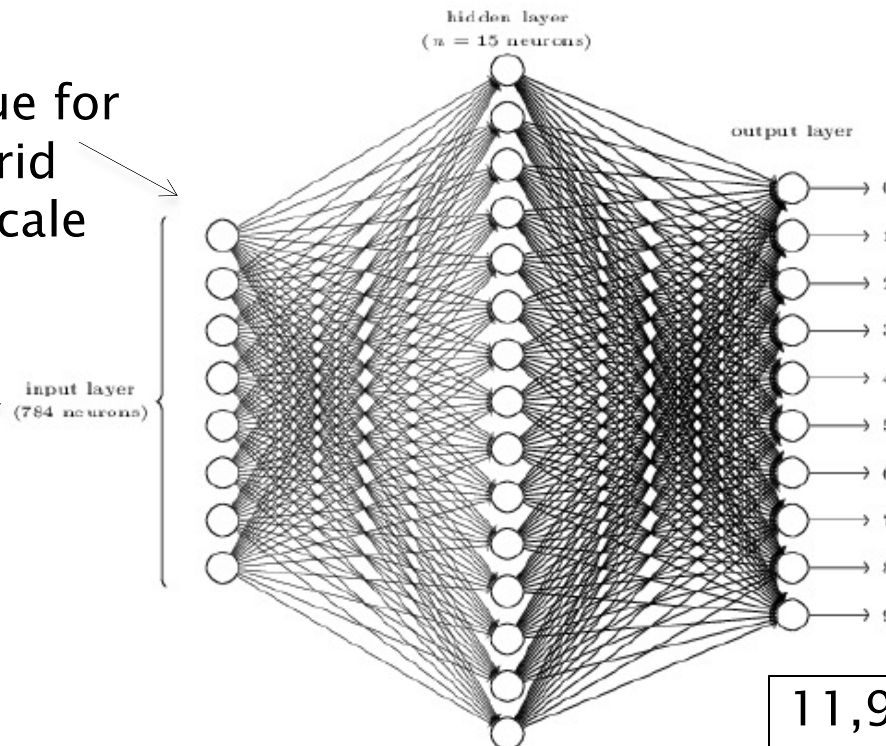
$Z^{(1)} = f(W^{(1)}X)$ $Z^{(r)} = f(W^{(r)}Z^{(r-1)})$ $Y = h(W^{(R+1)}Z^{(R)})$

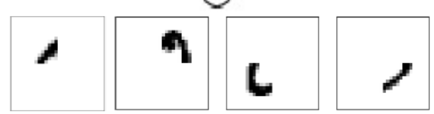
MNIST Classification: By Composition

Input value for
each grid
In grayscale



28 X 28
Grid





11,935
Parameters

Instead of trying to detect an entire object, detect smaller shapes

Put together small shapes to create entire object

Add Layers or Nodes?

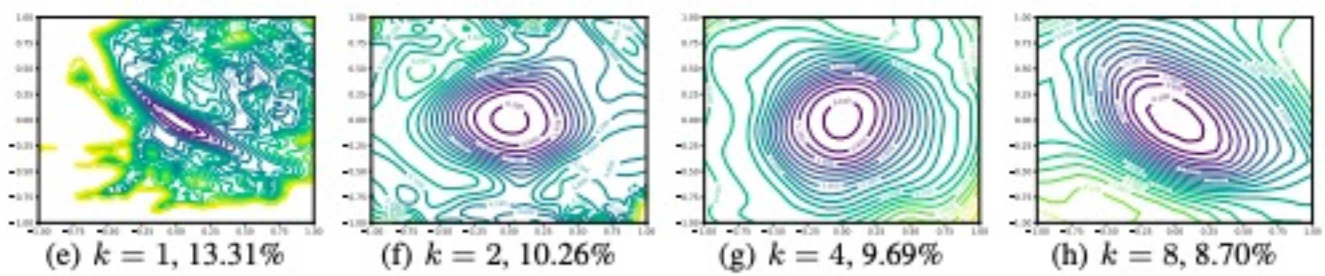
- It is better add additional layers to make the network more powerful (as opposed to increasing the nodes per layer)
  - Results in a network with a smaller number of nodes
  - Increases the network non-linearity
  - Allows the network to develop better hierarchical representations

How Deep can the Network Be?

- Stochastic Gradient Descent runs into computational problems, which were only solved in the last 10 years.
  - Vanishing Gradient Problem

In practice the number of hidden layers is limited to 20 or less

How Wide Should the Network Be?



- The width of the network has a critical effect on the smoothness of its Loss Function
- Loss Function landscape becomes progressively smoother as we make network wider. This makes the optimization task much easier
- Effect more pronounced in networks with hundreds of layers

https://arxiv.org/pdf/1712.09913.pdf

Keras Model for CIFAR-10

```
import keras
keras.__version__
from keras import models
from keras import layers

from keras.datasets import cifar10

(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

train_images = train_images.reshape((50000, 32 * 32 * 3))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 32 * 32 * 3))
test_images = test_images.astype('float32') / 255

from tensorflow.keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

network = models.Sequential()
network.add(layers.Dense(20, activation='relu', input_shape=(32 * 32 * 3,)))
network.add(layers.Dense(15, activation='relu'))
network.add(layers.Dense(10, activation='softmax'))

network.compile(optimizer='sgd',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

history = network.fit(train_images, train_labels, epochs=100, batch_size=128, validation_split=0.2)
```

Flatten 3D Tensor into a Vector

Convert Labels from Integers to 1-Hot Vectors

Define Model

Compile

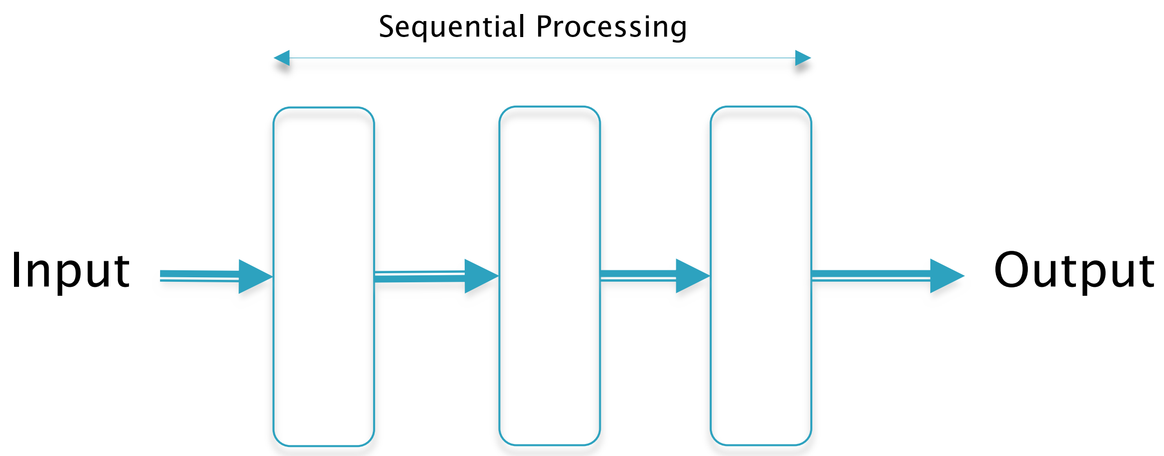
Execute Model

Feed Data in Batches

Validation Data

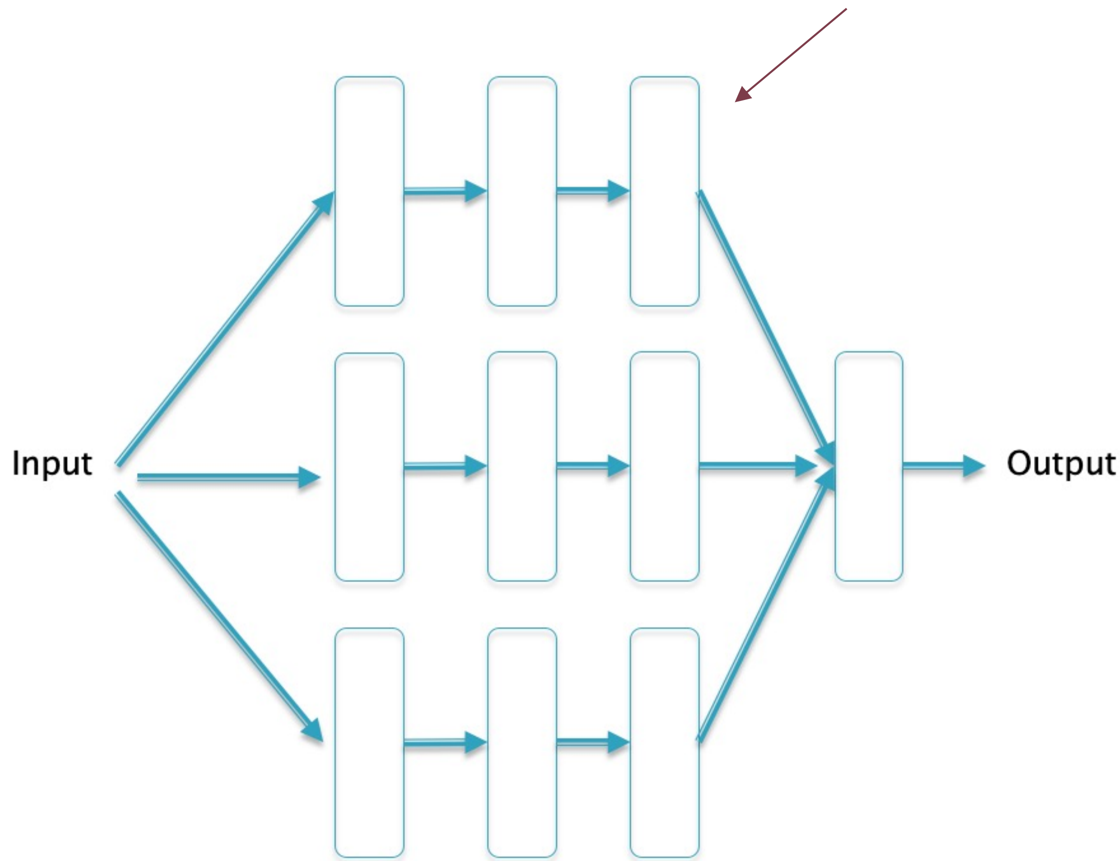
Network Topologies for Deep Networks

Sequential Processing



Parallel Processing: Model Ensembles

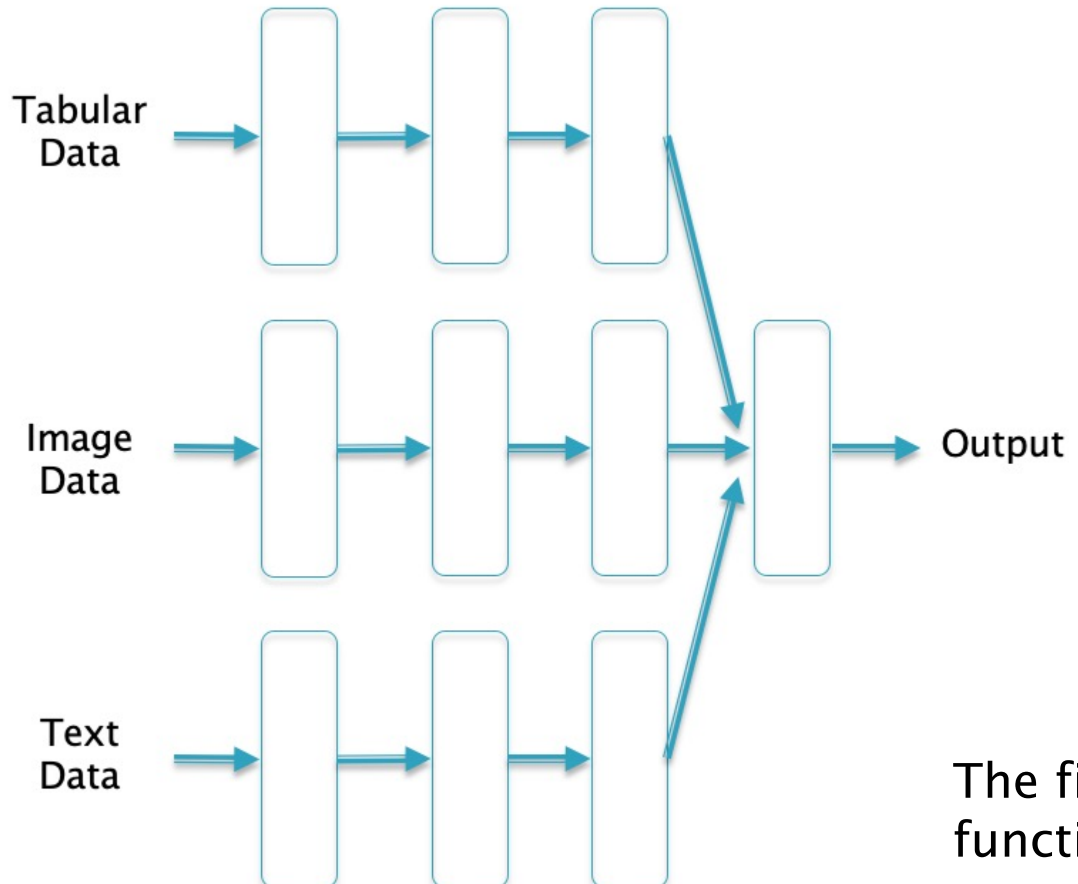
Multiple Predictions



Example: Majority Vote models

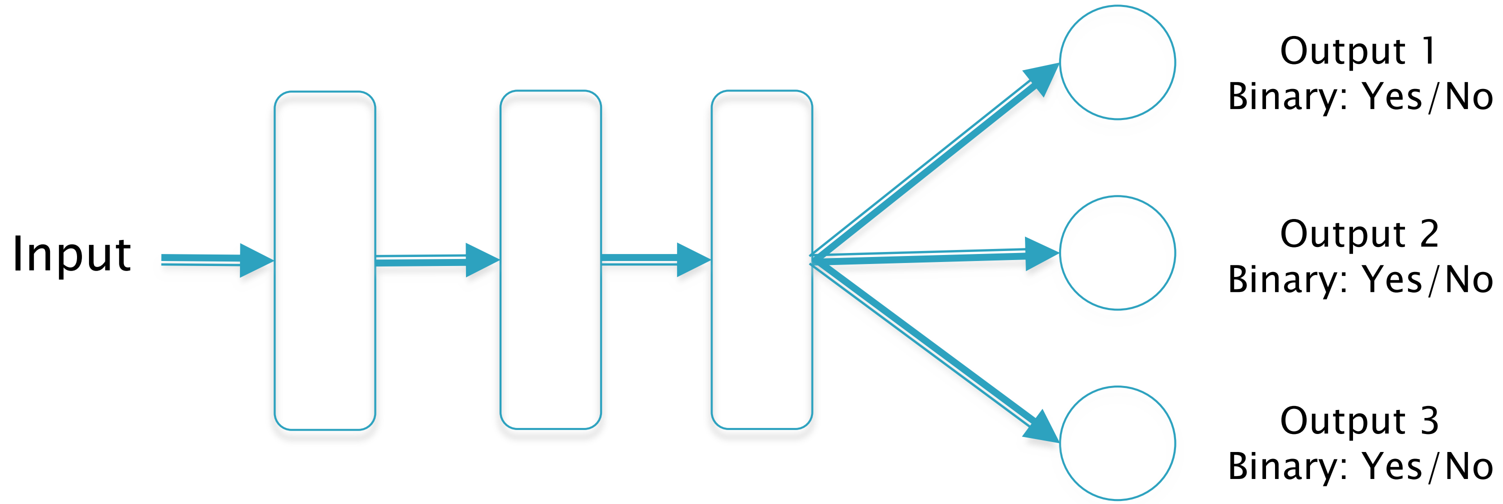
Increases prediction accuracy

Multi-Input Models



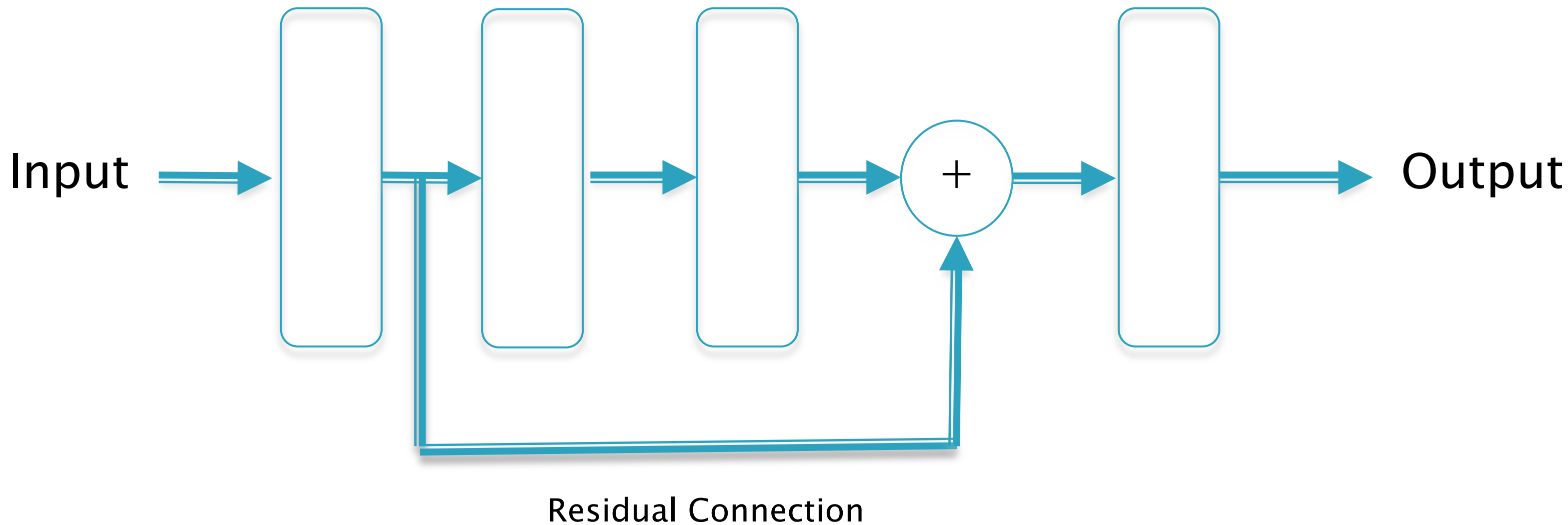
The final decision is a function of more than one type of input data

Multi-Label Classification



For classifying more than one object per input

Residual Connections



Enables the training of models with hundreds of hidden layers

Keras Sequential vs Functional API

All these different topologies can be easily coded using the Keras Functional API

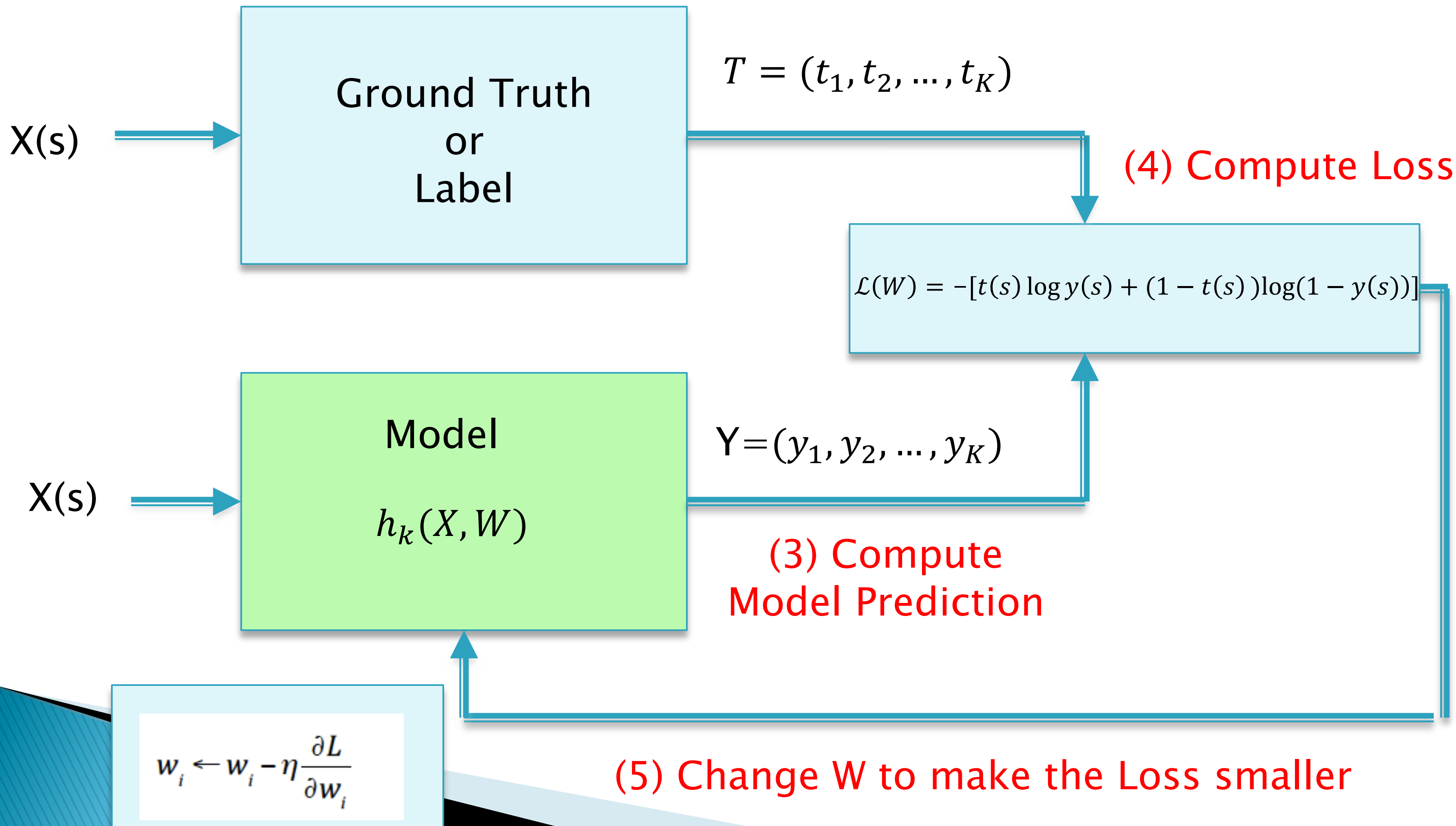
```
import keras
keras.__version__
from keras import Sequential, Model
from keras import layers
from keras import Input

from keras.datasets import cifar10

(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

train_images = train_images.reshape((50000, 32 * 32 * 3))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 32 * 32 * 3))
test_images = test_images.astype('float32') / 255

from tensorflow.keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

input_tensor = Input(shape=(32 * 32 * 3,))
x = layers.Dense(20, activation='relu')(input_tensor)
y = layers.Dense(15, activation='relu')(x)
output_tensor = layers.Dense(10, activation='softmax')(y)

model = Model(input_tensor, output_tensor)

model.compile(optimizer='sgd',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10, batch_size=128, validation_split=0.2)
```

The Backprop Algorithm

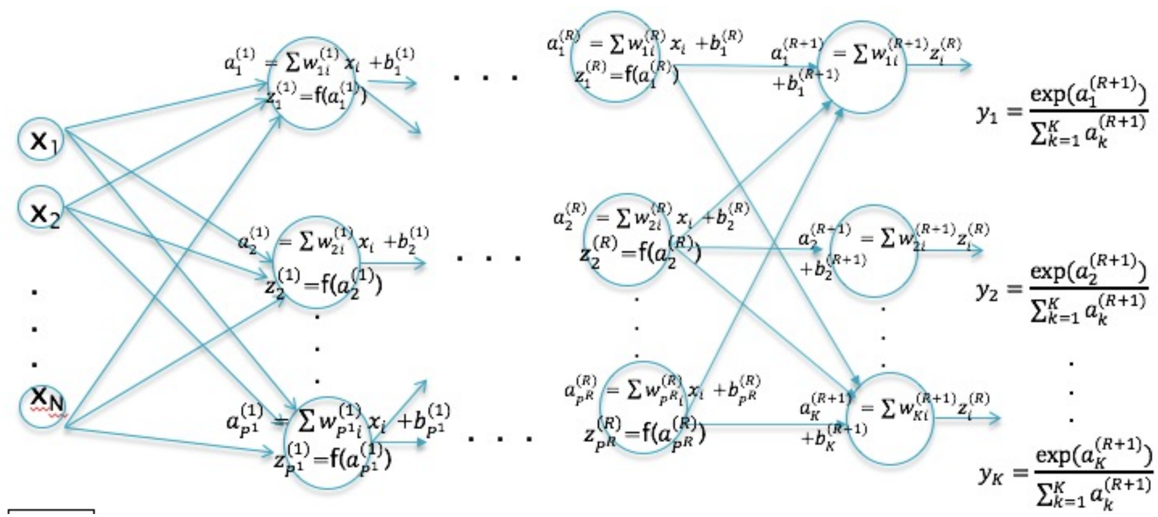
Framework for Supervised Learning

(1) Collect Labeled Data

(2) Choose Model $h_k(X,W)$



What Problem are we Solving?



The training algorithm stays the same:

$$\mathcal{L} = -\sum_{k=1}^{K} t_k \log y_k$$

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial \mathcal{L}}{\partial w_{ij}}$$

Need a way of Efficiently Computing $\frac{\partial \mathcal{L}}{\partial w_{ij}}$ for EVERY Weight!!

Historical Context

- By the late 1960s, people realized that hidden layers were needed to increase the modeling power of Neural Networks.
- There was little progress in this area until the mid-1980s, since there was no efficient algorithm for computing $\frac{\partial \mathcal{L}}{\partial w}$
- The Backprop algorithm (1986) met this need, and today remains a key part of the training scheme for all kinds of new deep architectures that have been discovered since then.

Numerical Differentiation

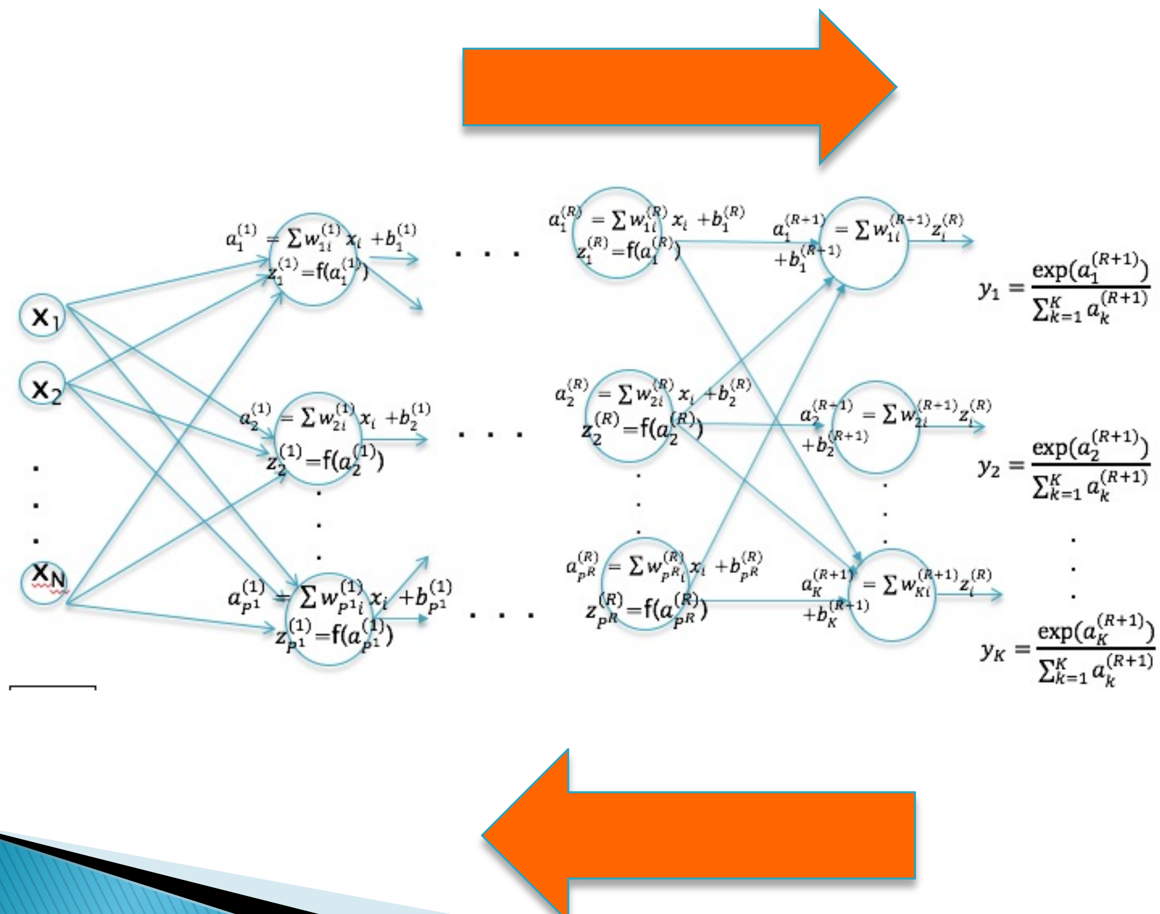
$$\frac{\partial L(w_1, w_2, \cdots, w_i, \cdots, w_n)}{\partial w_i} \approx \frac{L(w_1, w_2, \cdots, w_i + \Delta w_i, \cdots, w_n) - L(w_1, w_2, \cdots, w_i, \cdots, w_n)}{\Delta w_i}$$

What is wrong with this??

With a million weights, need million and one passes through the network to compute all the derivatives!!!

Using Backprop

- Backprop requires only TWO passes to compute ALL the derivatives, irrespective of the size of the network!



FORWARD PASS
Compute the Node Activations z

$$\frac{\partial \mathcal{L}}{\partial w} = z\delta$$

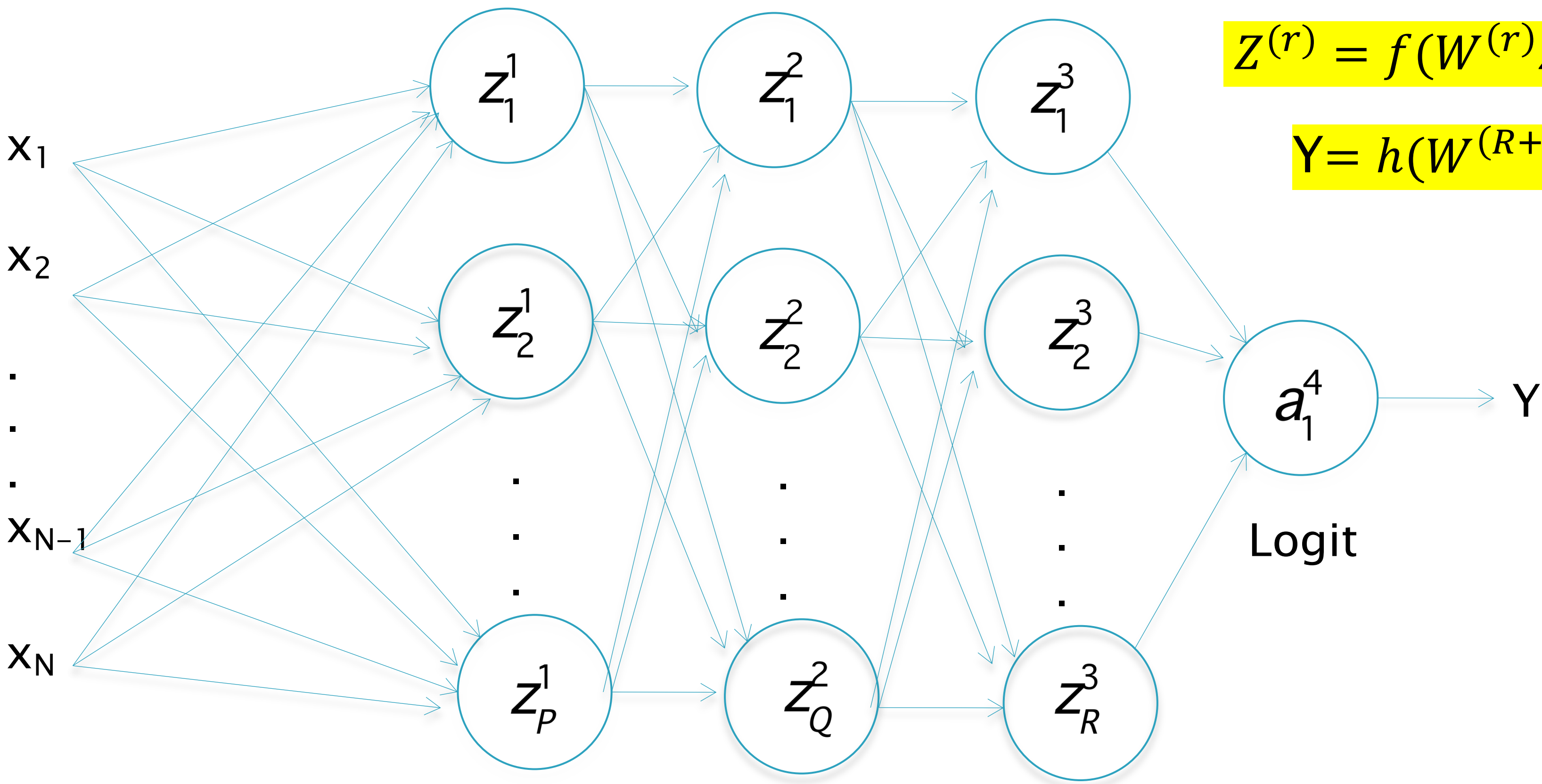
BACKWARD PASS
Compute the Node Gradients δ

Backprop – Forward Pass

$$Z^{(1)} = f(W^{(1)}X)$$

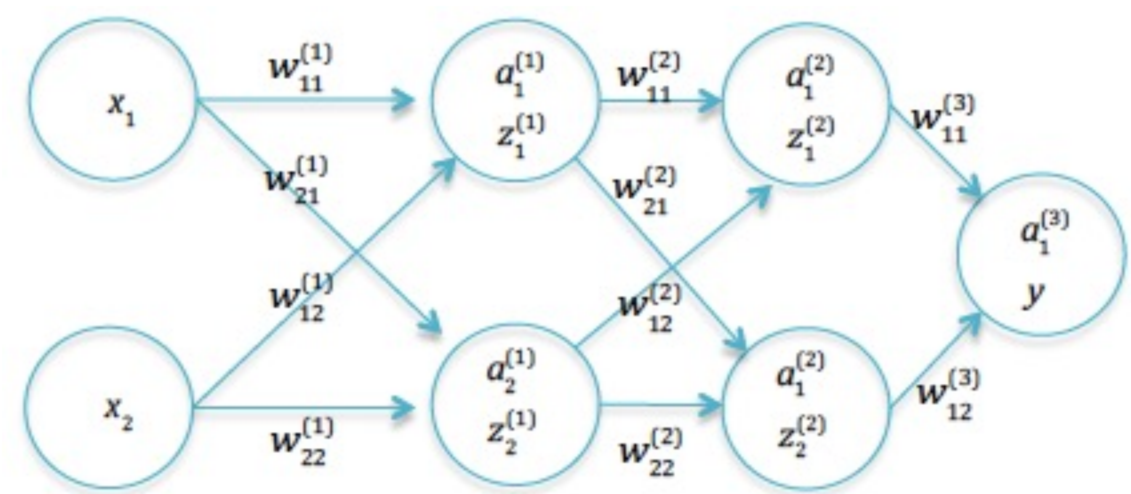
$$Z^{(r)} = f(W^{(r)}Z^{(r-1)})$$

$$Y = h(W^{(R+1)}Z^{(R)})$$



Given an input vector X, compute the activations z for each neuron in the network

Example: Forward Pass



$$a_1^{(1)} = w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2$$
$$z_1^{(1)} = f(a_1^{(1)})$$

$$a_2^{(1)} = w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2$$
$$z_2^{(1)} = f(a_2^{(1)})$$

$\rightarrow$

$$a_1^{(2)} = w_{11}^{(2)} z_1^{(1)} + w_{12}^{(2)} z_2^{(1)}$$
$$z_1^{(2)} = f(a_1^{(2)})$$

$$a_2^{(2)} = w_{21}^{(2)} z_1^{(1)} + w_{22}^{(2)} z_2^{(1)}$$
$$z_2^{(2)} = f(a_2^{(2)})$$

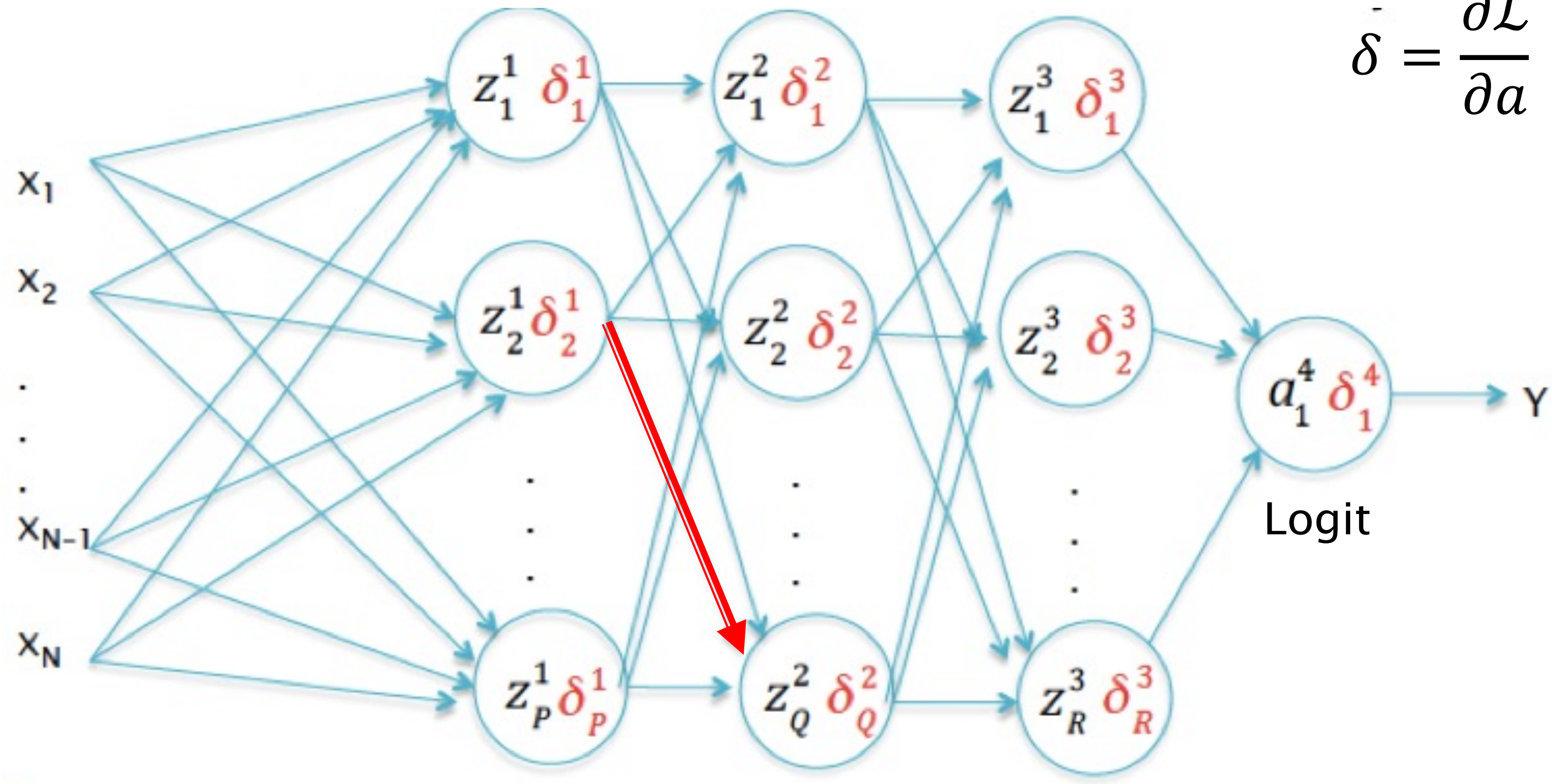
$\rightarrow$

$$a_1^{(3)} = w_{11}^{(3)} z_1^{(2)} + w_{12}^{(3)} z_2^{(2)}$$
$$y = h(a_1^{(3)})$$

f: ReLu Function
h: Softmax Function

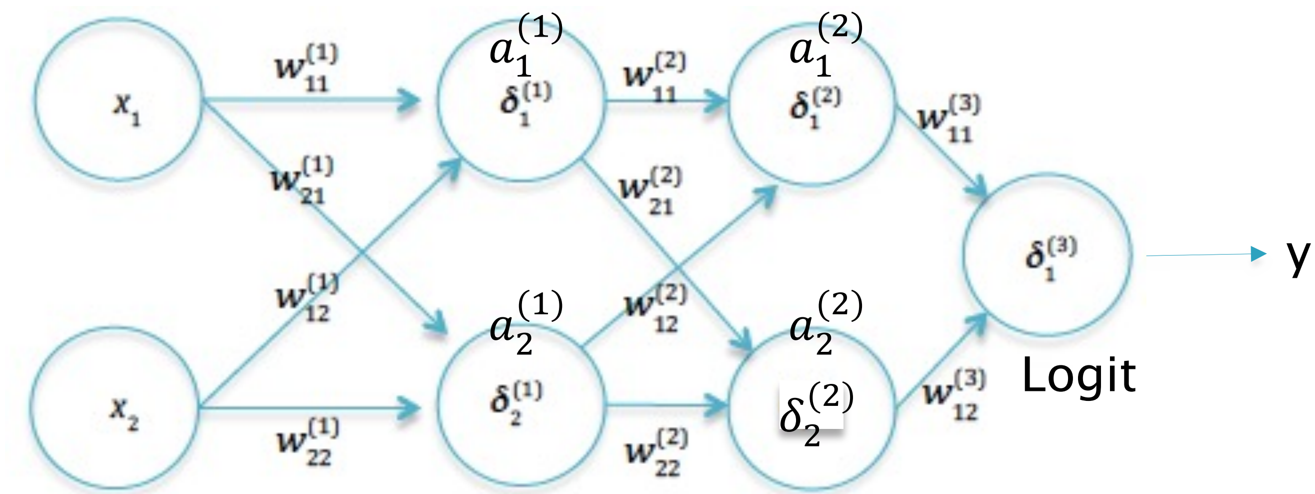
Backprop – Backward Pass

$$\delta = \frac{\partial \mathcal{L}}{\partial a}$$



$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(r)}} = z_j^{(r)}\delta_i^{(r+1)}$$

Example: Backward Pass



$$\delta_1^{(1)} = [w_{11}^{(2)}\delta_1^{(2)} + w_{21}^{(2)}\delta_2^{(2)}]f'(a_1^{(1)})$$

$$\delta_2^{(1)} = [w_{12}^{(2)}\delta_1^{(2)} + w_{22}^{(2)}\delta_2^{(2)}]f'(a_2^{(1)})$$

$$\delta_1^{(2)} = w_{11}^{(3)}\delta_1^{(3)}f'(a_1^{(2)})$$

$$\delta_2^{(2)} = w_{12}^{(3)}\delta_1^{(3)}f'(a_2^{(2)})$$

$$\delta_1^{(3)} = y - t$$

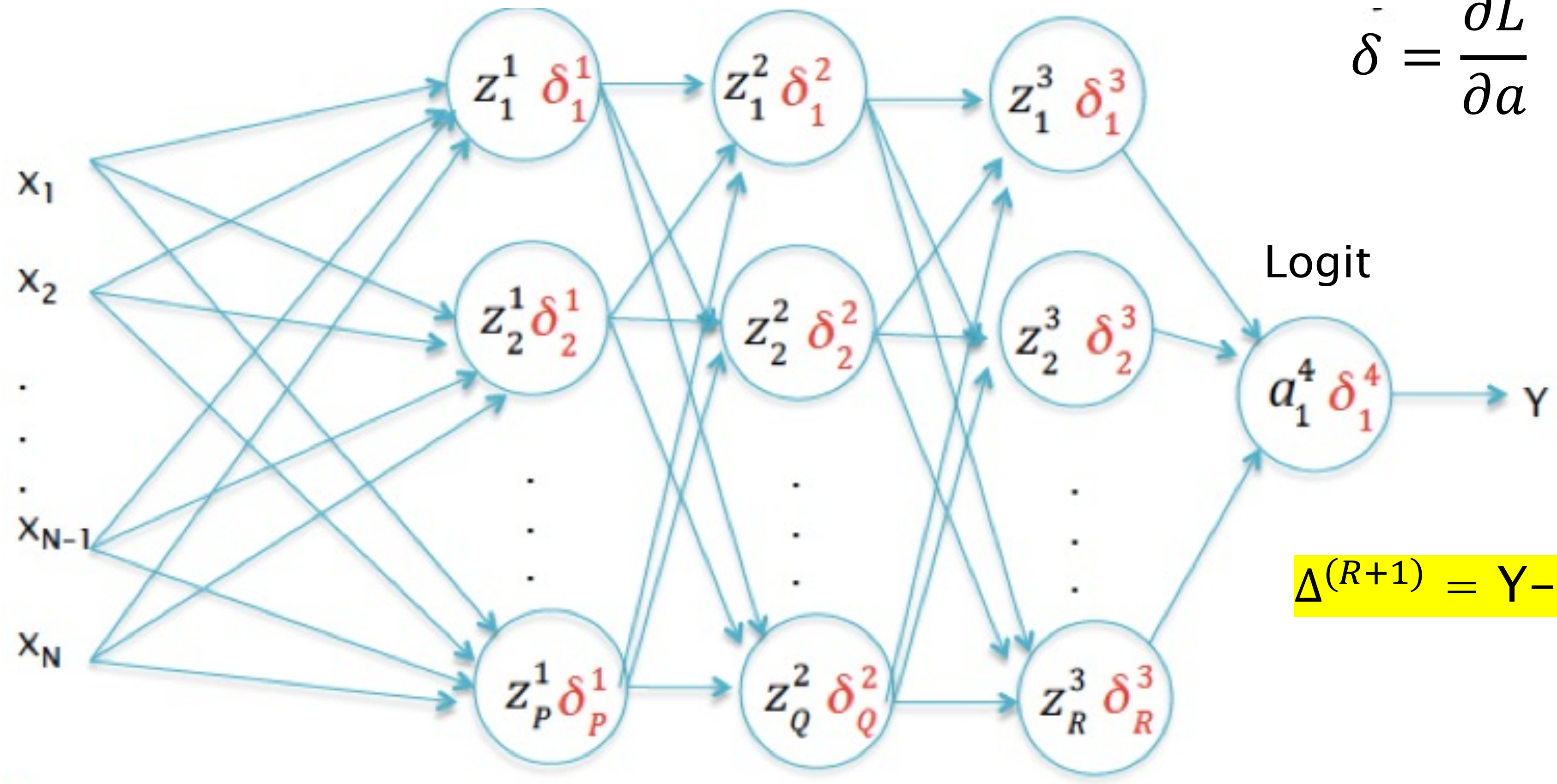
$$\Delta^{(3)} = \mathrm{Y}-\mathrm{T}$$

$$\Delta^{(1)} = f'\left(A^{(1)}\right)\odot\left(W^{(2)}\right)^T\Delta^{(2)}$$

$$\Delta^{(2)} = f'\left(A^{(2)}\right)\odot\left(W^{(3)}\right)^T\Delta^{(3)}$$

Backprop – Backward Pass

$$\delta = \frac{\partial L}{\partial a}$$



$\Delta^{(R+1)} = \text{Y–T}$

$$\Delta^{(r)} = f'\left(A^{(r)}\right) \odot \left(W^{(r+1)}\right)^T \Delta^{(r+1)}$$

Supplementary Reading

- Chapters 5: Linear Learning Models
- Chapter 6: NNDeep Learning https://srdas.github.io/DLBook2/
- First few Sections of Chapter 7: TrainingNNsBackprop