

Day 106 電腦視覺實務延伸

史丹佛線上 ConvNetJS 簡介



出題教練

陳宇春

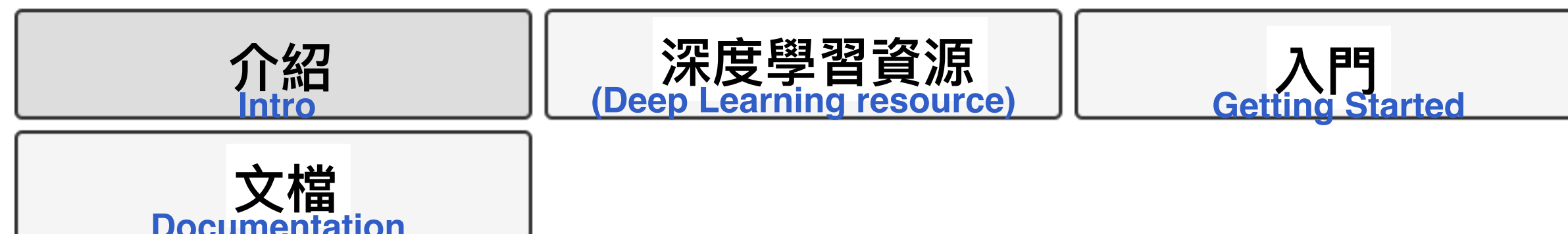


本日知識點目標

- 透過互動式網頁了解卷積網路
- 卷積網路超參數對於訓練與預測率的影響
- 特徵圖 (feature map) 視覺化的效果

何謂 ConvNetJS

- ConvNetJS 是一個 Javascript 庫，用於完全在您的瀏覽器中訓練深度學習模型（神經網路）
- 線上網址：
 - <https://cs.stanford.edu/people/karpathy/convnetjs/>
- 進入畫面：



何謂 ConvNetJS

瀏覽器演示

分類MINIST數字與卷積神經網路



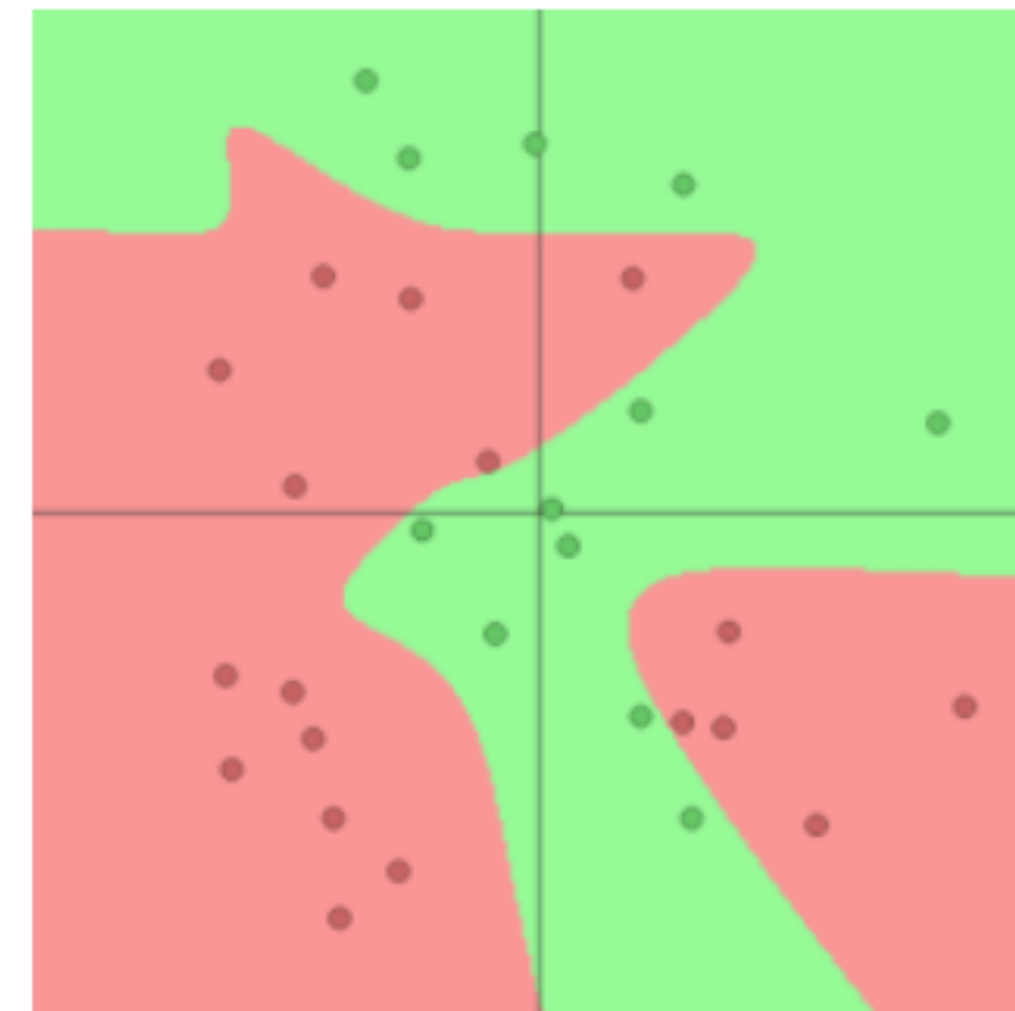
利用CNN 做手寫辨識

分類CIFR-10與卷積神經網路



利用CNN 做CIFAR10
影像辨識

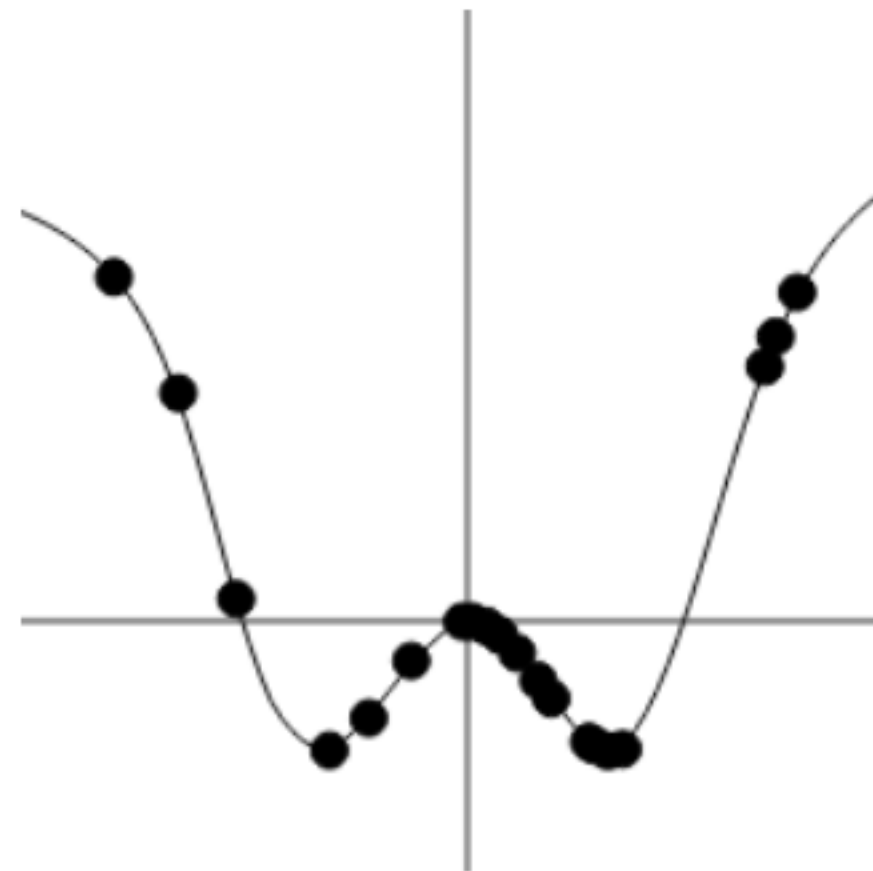
使用神經網路對玩具二維數據進行
交互式分類



利用NN 做二維分類演示

何謂 ConvNetJS

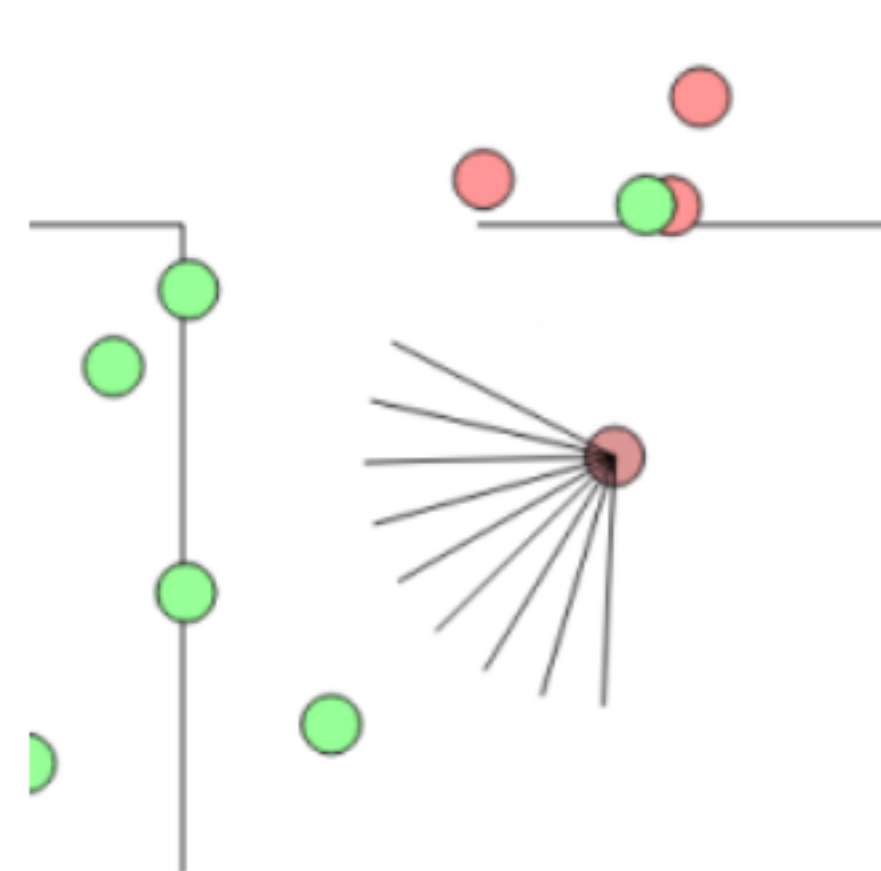
以交互方式回歸玩具1-D數據



訓練MNIST數字自動編碼器



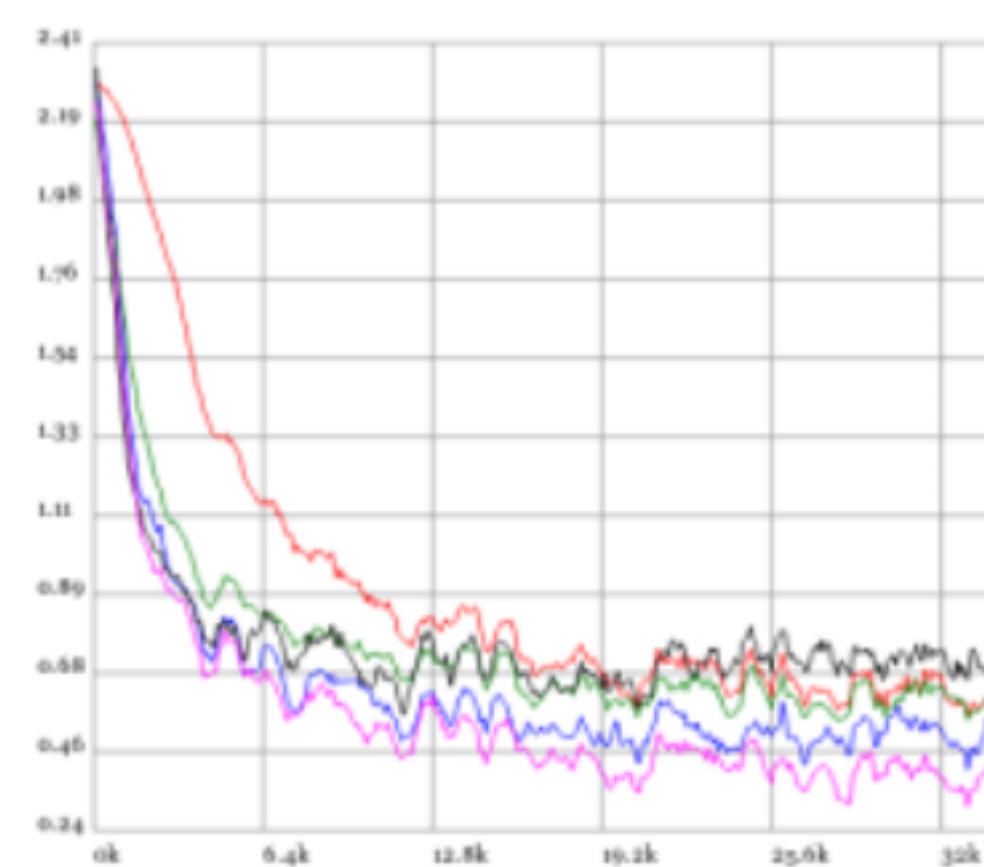
深度學習強化學習



神經網路「繪製」圖像



比較中SGD / Adadelata / Adagrad



利用 CNN 做 CIFAR10 影像辨識

這個演示在您的瀏覽器中訓練CIFAR-10數據集上的卷積神經網路，只有 Javascript。該數據集的最新技術水平約為 90%，人類表現約為 94%（不完美，因為數據集可能有點模稜兩可）。

這個數據集，數據增強包括隨機翻轉和水平和邏輯上最多 2px 的隨機圖像移位。

在本演示中，使用 Adadelata，它是每個參數自適應步長方法之一，因此我們不必擔心隨著時間的推移而改變學習率或動量。

分類CIFR-10與卷積神經網路



利用 CNN 做 CIFAR10 影像辨識 – 參數設定



pause

每個batch sample 前行時間
每個batch sample 後行時間
分類損失函數的值
採用L2 的學習率更新率
訓練精確度
驗證精確率
樣本數
學習率
學習動量
批次量大小
權重衰減率

Forward time per example: 8ms
Backprop time per example: 14ms
Classification loss: 1.93805
L2 Weight decay loss: 0.00115
Training accuracy: 0.25
Validation accuracy: 0.16
Examples seen: 1300
Learning rate: 0.01
Momentum: 0.9
Batch size: 4
Weight decay: 0.0001

change

change

change

change

save network snapshot as JSON

init network from JSON snapshot

load a pretrained network (achieves ~80% accuracy)

Training Stats

Loss:

| Epochs | Loss |
|--------|------|
| 0.1k | 2.25 |
| 0.3k | 2.12 |
| 1.0k | 2.14 |
| 1.1k | 1.96 |

clear graph

圖片來源：cs.stanford

利用CNN 做 CIFAR10 影像辨識 - 開始執行網路訓練(1)



Instantiate a Network and Trainer

```
layer_defs = []; # 定義網路起始
# 建立DATA 輸入層, 維度: 32x32x3
layer_defs.push({type:'input', out_sx:32, out_sy:32, out_depth:3});
# 建立卷積層1,該層將使用16個內核執行卷積, 每個內核大小為5x5。
#移動步數為1,輸入將在所有邊上填充2個像素以使輸出Vol具有相同的大小, 激活函數為 ReLU
layer_defs.push({type:'conv', sx:5, filters:16, stride:1, pad:2, activation:'relu'});
# 建立池化層1,每個池化內核大小為2x2,移動步數為2
layer_defs.push({type:'pool', sx:2, stride:2});
# 建立卷積層2,該層將使用20個內核執行卷積, 每個內核大小為5x5。
#移動步數為1,輸入將在所有邊上填充2個像素以使輸出Vol具有相同的大小, 激活函數為 ReLU
layer_defs.push({type:'conv', sx:5, filters:20, stride:1, pad:2, activation:'relu'});
# 建立池化層2,每個池化內核大小為2x2,移動步數為2
```


利用CNN 做 CIFAR10 影像辨識 - 開始執行網路訓練(2)

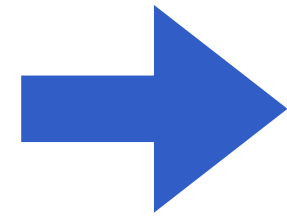


Instantiate a Network and Trainer

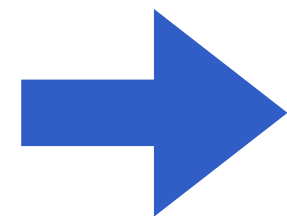
```
layer_defs.push({type:'pool', sx:2, stride:2});  
# 建立卷積層3,該層將使用20個內核執行卷積，每個內核大小為5x5。  
#移動步數為1,輸入將在所有邊上填充2個像素以使輸出Vol具有相同的大小，啟動函數為 ReLU  
layer_defs.push({type:'conv', sx:5, filters:20, stride:1, pad:2, activation:'relu'});  
# 建立池化層3,每個池化內核大小為2x2,移動步數為2  
layer_defs.push({type:'pool', sx:2, stride:2});  
#輸出Vol的大小為1x1x10  
layer_defs.push({type:'softmax', num_classes:10});  
#指定NET 為一個輸出網路  
net = new convnetjs.Net();  
#執行並建立網路  
net.makeLayers(layer_defs);  
#執行網路訓練，優化器採用adadelata, batch_size=4, l2_decay (l2,每次更新時學習率下降)=0.0001  
trainer = new convnetjs.SGDTrainer(net, {method:'adadelata', batch_size:4, l2_decay:0.0001});
```

利用 CNN 做 CIFAR10 影像辨識 – 網路視覺化

第一層啟動後的
視覺呈現



第一層啟動後的特徵圖
視覺呈現

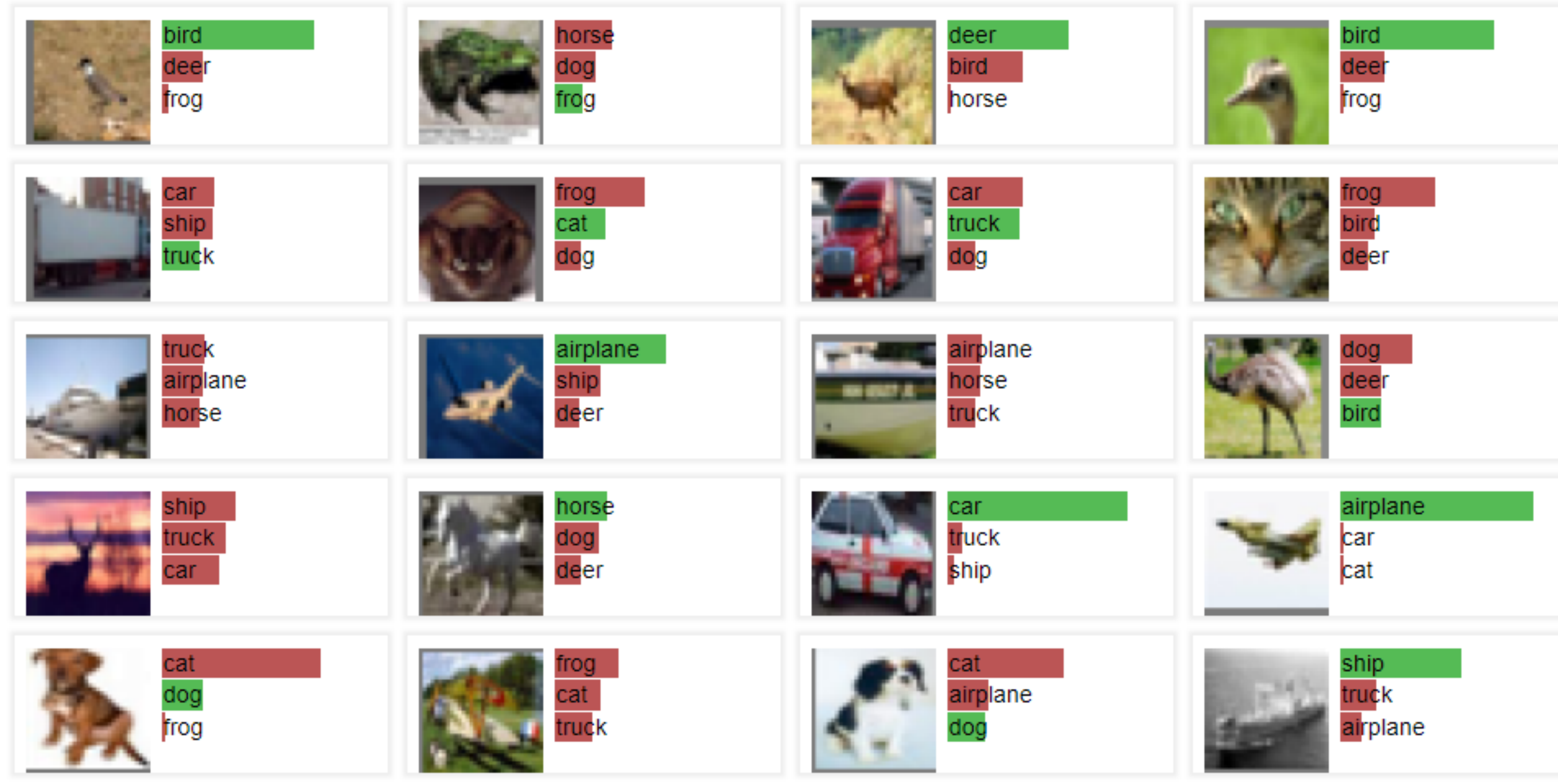


利用 CNN 做 CIFAR10 影像辨識 – 結果預測輸出

測試200張圖之後的精確率

Example predictions on Test set

test accuracy based on last 200 test images: 0.47



重要知識點複習：相關參數與優化器

- L2_weight decay：在設置上，Weight Decay 是一個 L2 penalty，是對參數取值平方和的懲罰
 - (1)取值可以量子化，即存在大量可壓縮空間
 - (2)因為 Relu 的存在使得其有界。
- weight decay（權值衰減）的使用最終目的是防止過擬合。所以 weight decay 的作用是調節模型複雜度對損失函數的影響，若 weight decay 很大，則復雜的模型損失函數的值也就大。
- Momentum 是梯度下降法中一種常用的加速技術

延伸閱讀：SGD, Adadelta, Adagrad

- 如果你是新手，我建議使用 **Adadelta** 或 **Adagrad**。他們會自動調整學習率並相對合理地進行調整。
- 如果你使用 **SGD**，你幾乎總是想要使用非零動量。0.9 通常用於動量。您需要稍微學習一下學習速度：如果它太高，您的網路將永遠不會收斂，並且在最壞的情況下將會發生災難性的災難，特別是如果您使用 ReLU 啟動。但如果它太低，網路將需要很長時間來訓練。你基本上總是想要使用非零 **l2_decay**。如果它太高，網路將非常強烈地正規化。如果你的訓練數據很少，這可能是一個好主意。如果您的訓練錯誤也非常低（因此您的網路正在完美地破壞訓練集），您可能需要增加一點以獲得更好的概括。如果您的訓練錯誤非常高（因此網路難以學習您的數據），您可能希望嘗試減少它。

延伸閱讀：SGD, Adadelta, Adagrad

- 如果您希望您的網路在末尾具有稀疏權重，則使用 **l1_decay** 而不是 **l2_decay**，因為權重的 l1 範數是稀疏性令人鼓舞的。如果您不知道我在說什麼，請不要觸摸 **l1_decay** 並將其保留為 0（默認值）。
- 通常您希望使用 1 的 **batch_size**。這基本上控制了網路的梯度步驟的準確度。如果您讓網路批量查看 100 個示例，它將能夠在實際執行該步驟之前估計更好的漸變值。然而，在實踐中，值 1（並且具有適當小的學習率）可能是最好的方式。

延伸閱讀：使用 JSON 保存和加載網路

- 使用 JSON 保存和加載網路

簡單地說，使用 toJSON () 和fromJSON () 函數。例如，保存和加載網：
執行最後一行後，net2 應該與原始網路完全相同

```
# network outputs all of its parameters into json object
```

```
# 網路輸出所有相關參數, 放入 json 物件
```

```
var json = net.toJSON();
```

```
# the entire object is now simply string. You can save this somewhere
```

```
# 整個物件現在只是字符串。你可以把它保存在某個地方
```

```
var str = JSON.stringify(json);
```

```
# 重新建立一個神經網路:
```

```
//從字符串中創建json對象( creates json object out of a string)
```

```
var json = JSON.parse(str);
```

```
// 創建一個新的網路 (create an empty network)
```

```
var net2 = new convnetjs.Net();
```

```
// 載入所有參數(load all parameters from JSON)
```

```
net2.fromJSON(json);
```

延伸閱讀：使用 JSON 保存和加載網路 - Keras



使用Keras 儲存與加載 網路模型架構

- 保存網路模型但是不包括參數

```
# save as JSON
```

```
json_string = model.to_json()
```

```
# 使用JSON 建構模型 (model reconstruction from JSON):
```

```
from keras.models import model_from_json
```

```
model = model_from_json(json_string)
```


延伸閱讀：使用 JSON – 關於優化器

ConvNetJS 網路一樣可以使用優化器：

 SGD+Momentum

Adadelta



adagrad.

```
// example SGD+Momentum trainer. Performs a weight update every 10 examples
```

```
var trainer = new convnetjs.Trainer(net, {method: 'sgd', learning_rate: 0.01,  
    l2_decay: 0.001, momentum: 0.9, batch_size:  
    10, l1_decay: 0.001});
```

// example that uses adadelta. Reasonable for beginners.

```
var trainer = new convnetjs.Trainer(net, {method: 'adadelta', l2_decay: 0.001,
                                         batch_size: 10});
```

```
// example adagrad.
```

[illegible]