

$$(X, O) = e^{-\frac{x^2}{2\sigma^2}}$$

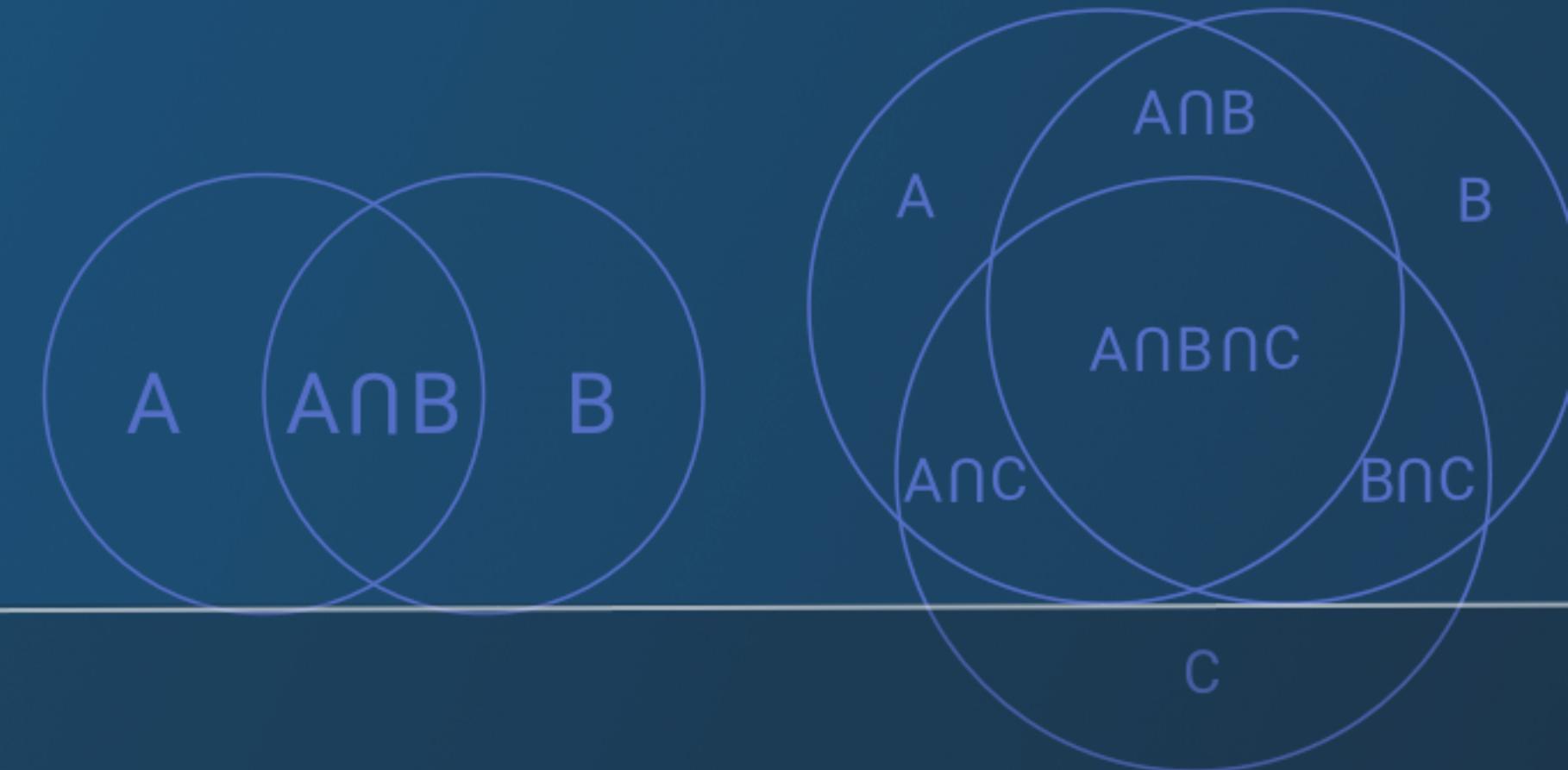
$$x(X, O) = -\frac{x}{\sigma^2} G(X, O) = -\frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$$

$$xx(X, O) = \frac{x^2 - \sigma^2}{\sigma^4} G(X, O) = \frac{x^2 - \sigma^2}{\sigma^4} e^{-\frac{x^2}{2\sigma^2}}$$

$$xxx(X, O) = -\frac{x^3 - x\sigma^2}{\sigma^6} G(X, O) = -\frac{x^3 - x\sigma^2}{\sigma^6} e^{-\frac{x^2}{2\sigma^2}}$$

$$= x \ln(x + \sqrt{1 + x^2})$$

Julia 程式語言學習馬拉松



Day 32



cupay 陪跑專家 : Andy Tu

類神經網路模型簡介





重要知識點



- 認識 Flux.jl 深度學習框架
- 了解 Zygote 如何應用在深度學習框架中



Flux.jl



- Flux 是 Julia 中知名的深度學習框架，它是完全以 Julia 實作，運算效率上是依賴 Julia 語言本身。套件本身使用 Julia 語言的陣列，並與語法相容。
- Flux 的自動微分功能是由 Zygote 提供，其可以提供高效而優化的微分式。



Layer



- Flux 有 Keras 般以層為基礎的網路搭建方式。
- 撰寫全連接層 (fully-connected layer) 的方式非常簡單，只需要使用 Dense，第一個參數為輸入的維度，第二個參數為輸出的維度，第三個參數為 activation function，預設為 identity function， σ 代表 sigmoid function。
- 層本身是個函數。

```
Dense(768, 128, σ)
```



多層感知器

- 不同層之間可以藉由 Chain 來連結函數。
- 多層感知器可以用以下方式建構。
- Chain 也可以用在一般的函數或是其他函式。

```
model = Chain(  
    Dense(768, 256, σ),  
    Dense(256, 128, σ),  
    Dense(128, 10, σ),  
    softmax)
```



準備訓練資料集

- 我們使用 MLDatasets 套件中的 MNIST 資料集。

```
using MLDatasets
train_X, train_y =
MNIST.traindata(Float32)
test_X, test_y =
MNISTtestdata(Float32)
```



切分minibatch



- 這邊需要先將資料切成 minibatch。

```
using Flux.Data: DataLoader  
batchsize = 1024  
train = DataLoader(train_X, train_y,  
batchsize=batchsize, shuffle=true)  
test = DataLoader(test_X, test_y,  
batchsize=batchsize)
```



損失函數及callback



- 損失函數的定義可以由使用者自行定義，這邊我們使用分類器常用的 cross entropy 來構成。

```
loss(x, y) =  
logitcrossentropy(model(x), y)  
evalcb() = @show(loss(X, Y))
```



訓練多層感知器



- 使用 Flux.train 來訓練模型，相關參數如下。
- @epochs 提供了方便的方式來設定要跑多少個 epoch。

```
using Flux: @epochs
epochs = 20
@epochs epochs Flux.train!(loss,
params(model), train, ADAM(0.005),
cb=throttle(evalcb, 10))
```



模型評估

- 模型評估方面可以自己撰寫準確度 (accuracy) 或是其他評估方式。
- 相關的評估方式並不由 Flux 套件提供，但可以由其他第三方套件提供。

```
using Flux: onecold
accuracy(x, y) = mean(onecold(m(x)) .==
onecold(y))
accuracy(X, Y)
```



Zygote 作為 Flux 的梯度計算核心



- Flux 套件的核心為 backpropagation 演算法，而 backpropagation 演算法需要計算梯度，這部份由 Zygote 提供自動微分的功能來計算梯度。
- 接下來會展示 Flux 是如何使用 Zygote 來得到梯度值。



使用 Zygote 計算梯度



```
julia> using Flux, Zygote
julia> m = Chain(Dense(10, 5, relu),
Dense(5, 2))
Chain(Dense(10, 5, NNlib.relu),
Dense(5, 2))
julia> x = rand(10);
julia> gs = gradient(() -> sum(m(x))),
params(m))
julia> gs[m[1].W]
```

知識點 回顧

- 認識 Flux.jl 深度學習框架
- 了解 Zygote 如何應用在深度學習框架中



推薦閱讀

- [Flux.jl 官方網頁](#)





解題時間

請跳出 PDF 至官網 Sample Code
& 作業開始解題