INTRODUCTION TO MACHINE LEARNING COMPSCI 4ML3

Lecture 20

Hassan Ashtiani

LOGISTIC REGRESSION

- $P(y = 1|x, W) = \sigma(W^T x)$
- $P(y = 0|x, W) = 1 - \sigma(W^T x)$
  - $\sigma(a) = \frac{1}{1+e^{-a}}$
- $P(x|W) = P(x)$
- MAXIMUM LIKELIHOOD SOLUTIONS FOR W:
- $\sum_{i=1}^{n}(y^i \log \sigma(< w, x^i >) + (1 - y^i) \log(1 - \sigma(< w, x^i)))$

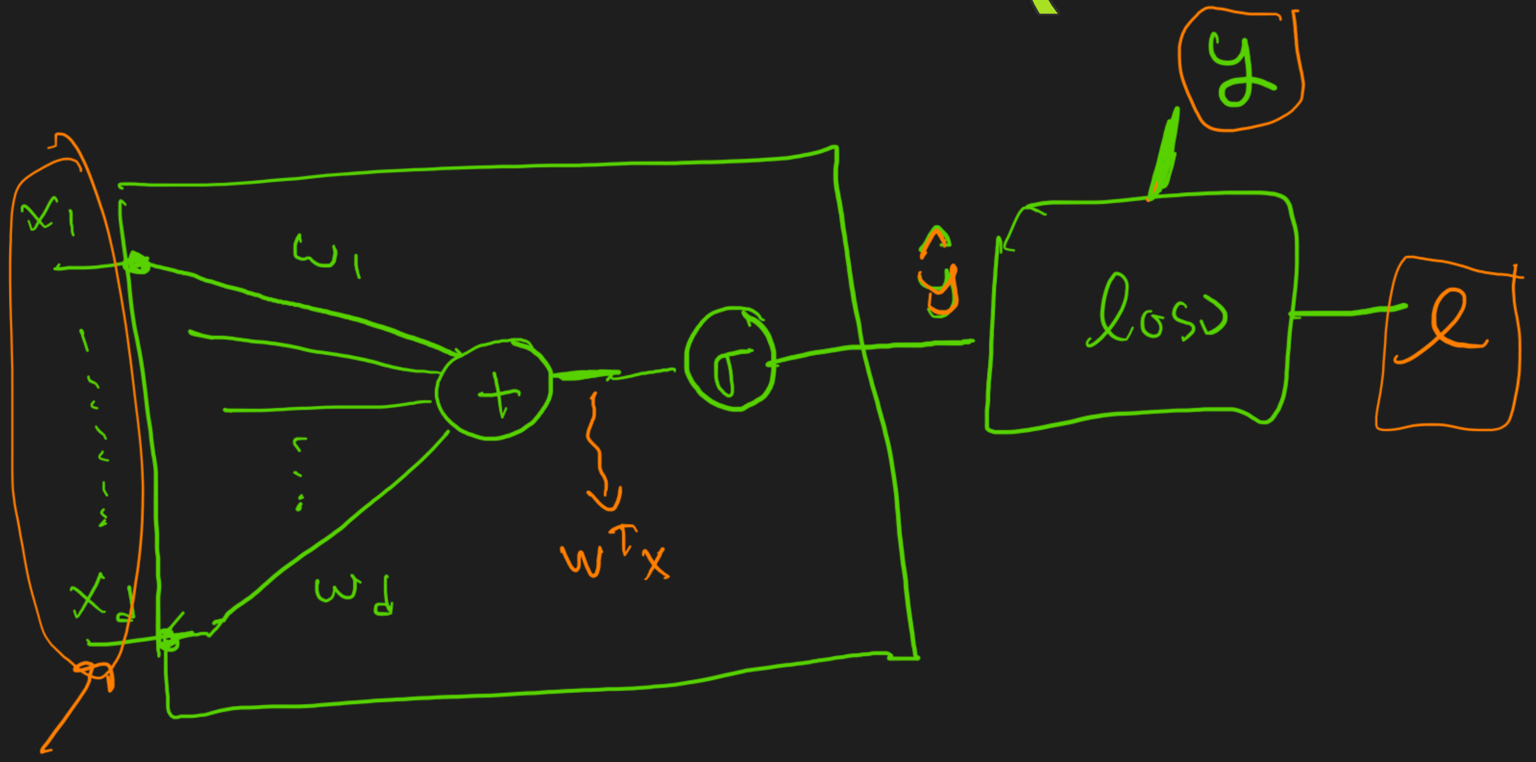
THE FORWARD MODEL

- $x \in R^d, w \in R^d, y \in \{0,1\}, f_W(x) \in [0,1]$

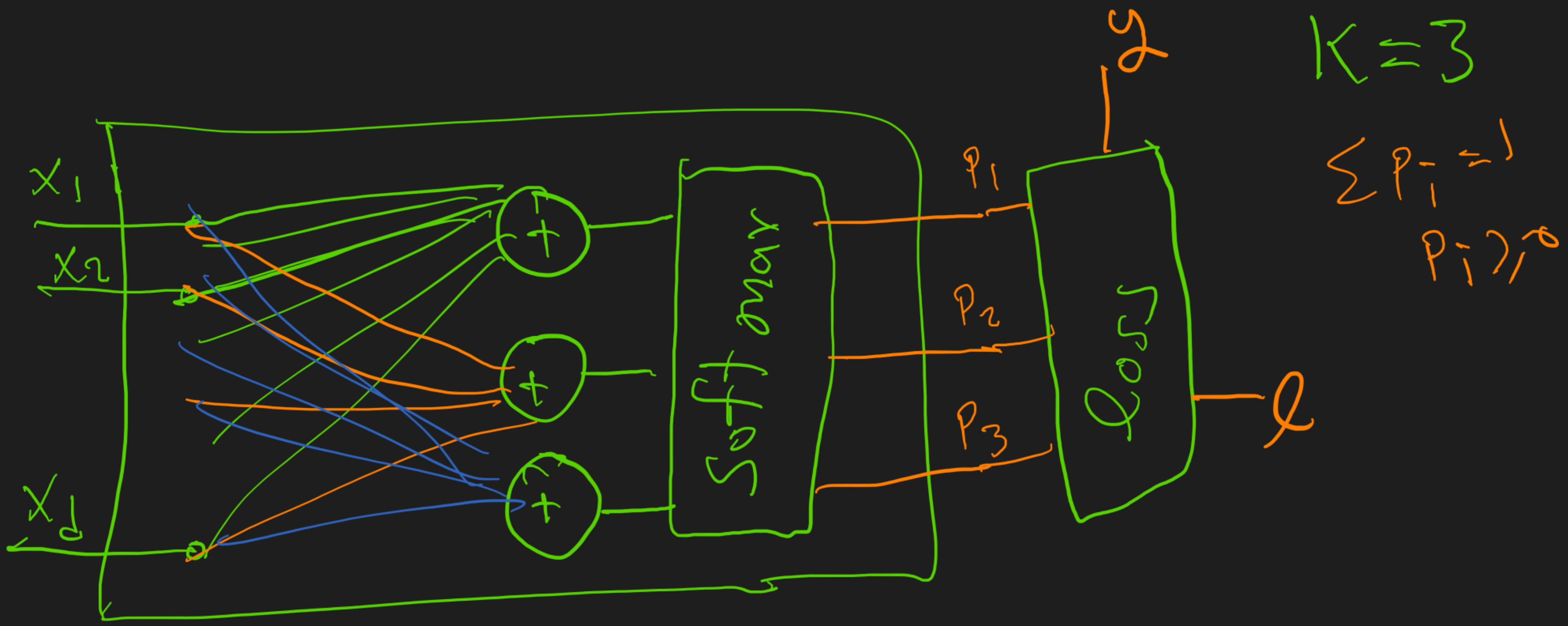
- $f_w(x) = \sigma(< w, x >)$
- $l(y, \hat{y}) = -y\text{LOG}\,\hat{y} - (1 - y)\text{LOG}(1 - \hat{y}))$

- $L = \sum_n^{i=1} l(y^i, f_w(x^i))$

LOGISTIC REGRESSION (VISUALIZED)



MULTI CLASS LOGISTIC REGRESSION



MULTI-CLASS LOGISTIC REGRESSION

- $x \in R^d, W \in R^{d \times k}, f_W(x) \in [0,1]^k$
  - $y \in [0,1]^k$ (ONE-HOT ENCODING)
- SOFT-MAX FUNCTION:
  - $\text{SOFT}MAX(y)_i = \frac{e^{y_i}}{\sum_j e^{y_j}}$ WHERE $y \in R^k$
- $f_W(x) = \text{SOFT}MAX(W^T x)$ WHERE $W \in R^{d \times k}$
- $l(y, \hat{y}) = -\sum_{j=1}^{k} y_j \text{LOG}\, \hat{y}_j$

one-hot encoding

- $L = \sum_{n}^{i=1} l(y^i, f_W(x^i))$

e.g. $y=2$, $[0\ 1\ 0]^T$, $K=3$

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

$\sum (1+e^{-y_i})^{-1} = \cdots$

PYTORCH

BCELoss

CLASS torch.nn.BCELoss(*weight=None, size_average=None, reduce=None, reduction='mean'*) [SOURCE]

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_n \left[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n) \right],$$

BCEWithLogitsLoss

CLASS torch.nn.BCEWithLogitsLoss(*weight=None, size_average=None, reduce=None, reduction='mean', pos_weight=None*) [SOURCE]

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_n \left[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n)) \right],$$

PYTORCH

CrossEntropyLoss

CLASS torch.nn.CrossEntropyLoss(*weight=None*, *size_average=None*, *ignore_index=-100*, *reduce=None*, *reduction='mean'*, *label_smoothing=0.0*) [SOURCE]

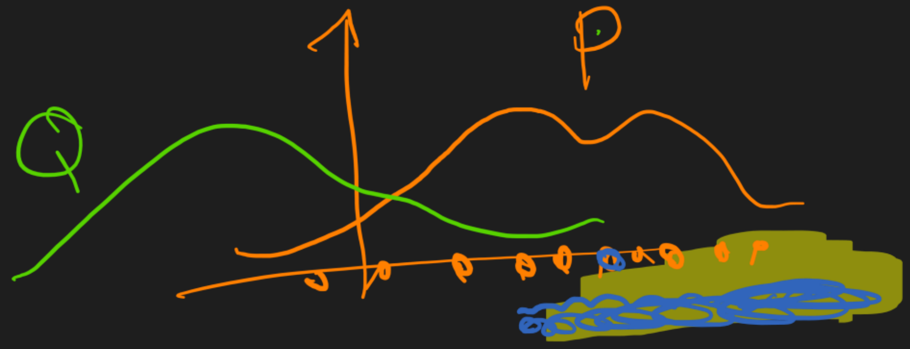
$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})} \cdot 1\{y_n \neq \text{ignore\_index}\}$$

already applies softmax

```python
class LogisticRegression(nn.Module):
    def __init__(self, input_size, num_classes):
        super(LogisticRegression, self).__init__()
        self.fc = nn.Linear(input_size, num_classes, bias=True)

    def forward(self, x):
        # Flatten the image
        x= x.view(-1, 28*28)
        return self.fc(x)

model = LogisticRegression(input_size, num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    for i, (images_batch, labels_batch) in enumerate(train_loader):
        optimizer.zero_grad() # Clear the gradients
        outputs = model(images_batch) # Forward pass
        loss = criterion(outputs, labels_batch) # Calculate loss
        loss.backward() # Backward pass
        optimizer.step() # Update weights
```

CROSS-ENTROPY – INFORMATION THEORY

- WE CAN VIEW THE OUTPUT OF THE SOFTMAX AS PROBABILITY DISTRIBUTION OVER k LABELS
- CROSS-ENTROPY
  - $H(P,Q) = E_{y\sim P} - \log(Q(y)) = -\sum P(y)\log(Q(y))$
  - UNLIKE PYTORCH, IT RECEIVES TWO PROBABILITY DISTRIBUTIONS SO THERE IS NO NEED FOR SOFTMAX



NON LINEAR MODELS?