Perceptrons and Structured Perceptrons

CS114B Lab 8

Kenneth Lai

March 26, 2021

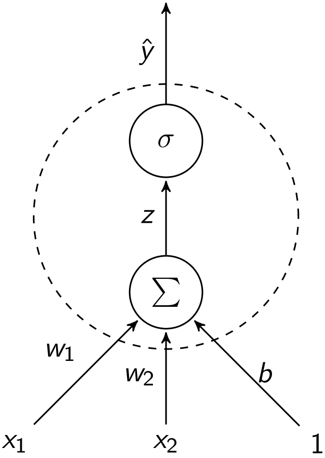
Perceptrons

- Documents are characterized by features
  - No independence assumptions
- For each feature j:
  - Value x_j
  - Weight w_j
- Bias term b
- "Score" $z = \left(\sum_{j=1}^{n} w_j x_j \right) + b = \mathbf{w} \cdot \mathbf{x} + b$

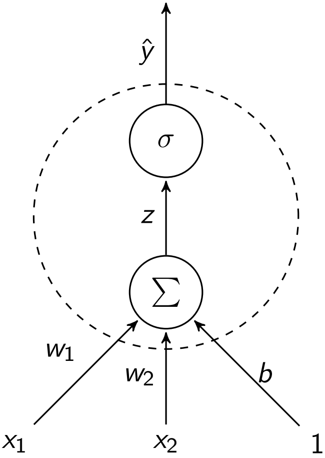
Perceptrons

- Documents are characterized by features
  - No independence assumptions
- For each feature j:
  - Value x_j
  - Weight w_j
- Bias term b
- "Score" $z = \left(\sum_{j=1}^{n} w_j x_j\right) + b = \mathbf{w} \cdot \mathbf{x} + b$
- Does this look familiar?

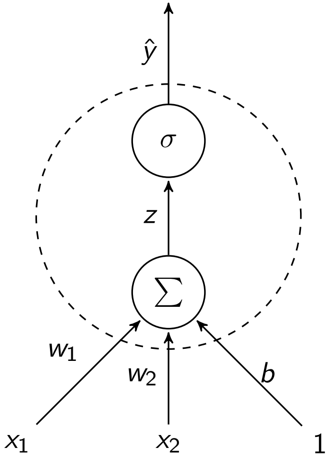
Graphical Representation of Logistic Regression



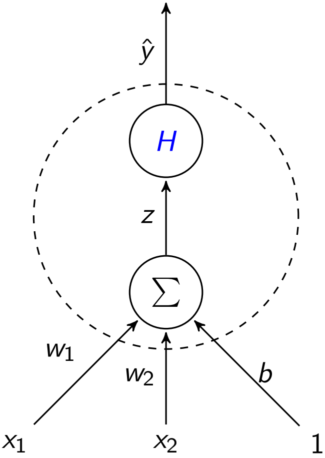
Graphical Representation of a Neuron



Graphical Representation of a Perceptron

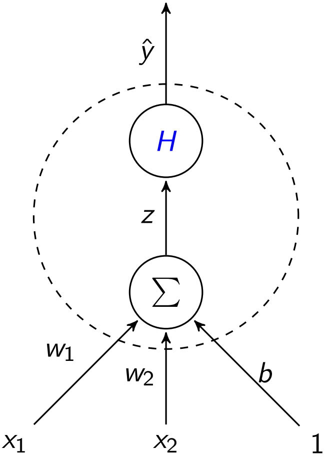


Graphical Representation of a Perceptron



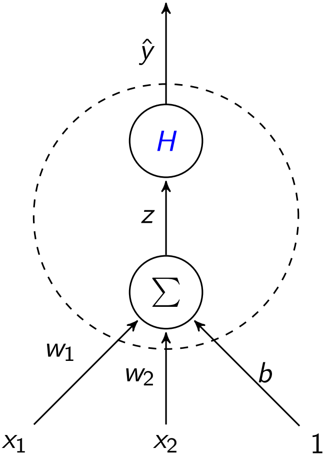
- (Heaviside) step function H

Graphical Representation of a Perceptron



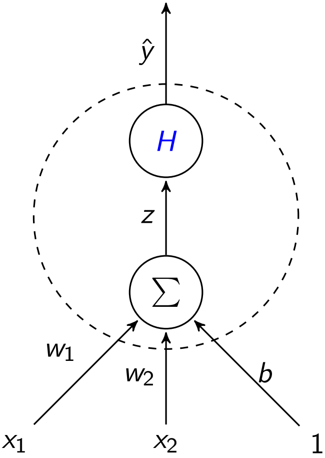
- (Heaviside) step function H
  - $H(z) = \begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{if } z < 0 \end{cases}$

Graphical Representation of a Perceptron



- (Heaviside) step function H
  - $H(z) = \begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{if } z < 0 \end{cases}$
  - What if $z = 0$?

Graphical Representation of a Perceptron



- (Heaviside) step function H
  - $H(z) = \begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{if } z < 0 \end{cases}$
  - What if $z = 0$?
    - Set by convention (1, 0, or 1/2)

Gradients in Perceptrons

- Hinge loss $L(\hat{y}, y) = (\hat{y} - y)z$

Gradients in Perceptrons

- Hinge loss $L(\hat{y}, y) = (\hat{y} - y)z$
- We want to compute $\frac{\partial L}{\partial w_j}$
- Chain Rule of calculus: $\frac{dy}{dx} = \frac{dy}{dz}\frac{dz}{dx}$
- Looking at the graph: $\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial w_j}$

Gradients in Perceptrons

- Hinge loss $L(\hat{y}, y) = (\hat{y} - y)z$
- We want to compute $\dfrac{\partial L}{\partial w_j}$
- Chain Rule of calculus: $\dfrac{dy}{dx} = \dfrac{dy}{dz}\dfrac{dz}{dx}$
- Looking at the graph: $\dfrac{\partial L}{\partial w_j} = \dfrac{\partial L}{\partial \hat{y}}\dfrac{\partial \hat{y}}{\partial z}\dfrac{\partial z}{\partial w_j}$
  - Problem: $\dfrac{\partial \hat{y}}{\partial z} = 0$ almost everywhere

Gradients in Perceptrons

- Hinge loss $L(\hat{y}, y) = (\hat{y} - y)z$
- We want to compute $\dfrac{\partial L}{\partial w_j}$
- Chain Rule of calculus: $\dfrac{dy}{dx} = \dfrac{dy}{dz}\dfrac{dz}{dx}$
- Looking at the graph: $\dfrac{\partial L}{\partial w_j} = \dfrac{\partial L}{\partial \hat{y}}\dfrac{\partial \hat{y}}{\partial z}\dfrac{\partial z}{\partial w_j}$
  - Problem: $\dfrac{\partial \hat{y}}{\partial z} = 0$ almost everywhere
    - A change in z does not result in a change in $\hat{y}$ unless you are at the threshold

Gradients in Perceptrons

- Hinge loss $L(\hat{y}, y) = (\hat{y} - y)z$
- We want to compute $\frac{\partial L}{\partial w_j}$
- Chain Rule of calculus: $\frac{dy}{dx} = \frac{dy}{dz}\frac{dz}{dx}$
- Looking at the graph: $\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial w_j}$
  - Problem: $\frac{\partial \hat{y}}{\partial z} = 0$ almost everywhere
    - A change in z does not result in a change in $\hat{y}$ unless you are at the threshold
  - Solution: consider $\frac{\partial L}{\partial z}$ directly

Gradients in Perceptrons

- Hinge loss $L(\hat{y}, y) = (\hat{y} - y)z$
- We want to compute $\frac{\partial L}{\partial w_j}$
- Chain Rule of calculus: $\frac{dy}{dx} = \frac{dy}{dz}\frac{dz}{dx}$
- Looking at the graph: $\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial w_j}$

Gradients in Perceptrons

- $\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial w_j}$

Gradients in Perceptrons

- $\dfrac{\partial L}{\partial w_j} = \dfrac{\partial L}{\partial z} x_j$
  - $\dfrac{\partial z}{\partial w_j} = x_j$

Gradients in Perceptrons

- $\frac{\partial L}{\partial w_j} = (\hat{y} - y)x_j$
  - $\frac{\partial z}{\partial w_j} = x_j$
  - For the hinge loss: $\frac{\partial L}{\partial z} = \hat{y} - y$

Gradients in Perceptrons

- $\frac{\partial L}{\partial w_j} = (\hat{y} - y)x_j$
  - $\frac{\partial z}{\partial w_j} = x_j$
  - For the hinge loss: $\frac{\partial L}{\partial z} = \hat{y} - y$
- $\frac{\partial L}{\partial b} = \hat{y} - y$
  - $\frac{\partial z}{\partial b} = 1$
  - ...

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- At each time step t:
  - Compute gradient ∇L
  - Move in direction of negative gradient
- $\theta_{t+1} = \theta_t - \eta \nabla L$

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- At each time step t:
  - Compute gradient ∇L
  - Move in direction of negative gradient
- $\theta_{t+1} = \theta_t - \eta \nabla L$
- Does this look familiar?

Perceptron Learning Algorithm

- If $\hat{y} = y$, then $\hat{y} - y = 0$
  - Do nothing

Perceptron Learning Algorithm

- If $\hat{y} = y$, then $\hat{y} - y = 0$
  - Do nothing
- If $\hat{y} = 0$ and $y = 1$, then $\hat{y} - y = -1$
  - $\nabla L = -\mathbf{x}$
  - $\theta_{t+1} = \theta_t + \eta \mathbf{x}$
  - Increment weights

Perceptron Learning Algorithm

- If $\hat{y} = y$, then $\hat{y} - y = 0$
  - Do nothing
- If $\hat{y} = 0$ and $y = 1$, then $\hat{y} - y = -1$
  - $\nabla L = -\mathbf{x}$
  - $\theta_{t+1} = \theta_t + \eta \mathbf{x}$
  - Increment weights
- If $\hat{y} = 1$ and $y = 0$, then $\hat{y} - y = 1$
  - $\nabla L = \mathbf{x}$
  - $\theta_{t+1} = \theta_t - \eta \mathbf{x}$
  - Decrement weights

Multi-class Perceptrons

- Separate weights and bias terms for each class c

$$z_c = \left(\sum_{j=1}^{n} w_{jc} x_j \right) + b_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

Multi-class Perceptrons

- Separate weights and bias terms for each class c

$$z_c = \left(\sum_{j=1}^{n} w_{jc} x_j \right) + b_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

- $\hat{y} = \operatorname*{argmax}_{k=1}^{K} z_k$

Multi-class Perceptrons

- Separate weights and bias terms for each class c

$$z_c = \left(\sum_{j=1}^{n} w_{jc} x_j \right) + b_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

- $\hat{y} = \operatorname*{argmax}_{k=1}^{K} z_k$
  - Unlike in multinomial logistic regression, $\hat{y}$ does not have to be a vector

Multi-class Perceptrons

- Separate weights and bias terms for each class c

$$z_c = \left(\sum_{j=1}^{n} w_{jc} x_j \right) + b_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

- $\hat{y} = \operatorname*{argmax}_{k=1}^{K} z_k$
    - Unlike in multinomial logistic regression, $\hat{y}$ does not have to be a vector
        - If we wanted to, we could define one-hot vectors $\hat{\mathbf{y}}$ and $\mathbf{y}$

Multi-class Perceptrons

- Separate weights and bias terms for each class c

$$z_c = \left(\sum_{j=1}^{n} w_{jc} x_j \right) + b_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

- $\hat{y} = \operatorname*{argmax}_{k=1}^{K} z_k$
  - Unlike in multinomial logistic regression, $\hat{y}$ does not have to be a vector
    - If we wanted to, we could define one-hot vectors $\hat{\mathbf{y}}$ and $\mathbf{y}$
  - What if there is a tie?

Multi-class Perceptrons

- Separate weights and bias terms for each class c

$$z_c = \left(\sum_{j=1}^{n} w_{jc} x_j \right) + b_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

- $\hat{y} = \operatorname*{argmax}_{k=1}^{K} z_k$
  - Unlike in multinomial logistic regression, $\hat{y}$ does not have to be a vector
    - If we wanted to, we could define one-hot vectors $\hat{\mathbf{y}}$ and $\mathbf{y}$
  - What if there is a tie?
    - Do whatever `numpy.argmax` does

Multi-class Perceptrons

- Separate weights and bias terms for each class c

$$z_c = \left(\sum_{j=1}^{n} w_{jc} x_j \right) + b_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

- $\hat{y} = \underset{k=1}{\overset{K}{\operatorname{argmax}}}\, z_k$
  - Unlike in multinomial logistic regression, $\hat{y}$ does not have to be a vector
    - If we wanted to, we could define one-hot vectors $\hat{\mathbf{y}}$ and $\mathbf{y}$
  - What if there is a tie?
    - Do whatever `numpy.argmax` does
- Hinge loss $L(\hat{y}, y) = z_{\hat{y}} - z_y$

Multi-class Perceptron Learning Algorithm

- If $\hat{y} = y$, then do nothing
- Else:

Multi-class Perceptron Learning Algorithm

- If $\hat{y} = y$, then do nothing
- Else:
  - For the correct class y, $\dfrac{\partial L}{\partial z_y} = -1$
    - $(\nabla L)_y = -\mathbf{x}$
    - $(\theta_y)_{t+1} = (\theta_y)_t + \eta\mathbf{x}$
    - Increment weights

Multi-class Perceptron Learning Algorithm

- If $\hat{y} = y$, then do nothing
- Else:
  - For the correct class y, $\dfrac{\partial L}{\partial z_y} = -1$
    - $(\nabla L)_y = -\mathbf{x}$
    - $(\theta_y)_{t+1} = (\theta_y)_t + \eta \mathbf{x}$
    - Increment weights
  - For the predicted class $\hat{y}$, $\dfrac{\partial L}{\partial z_{\hat{y}}} = 1$
    - $(\nabla L)_{\hat{y}} = \mathbf{x}$
    - $(\theta_{\hat{y}})_{t+1} = (\theta_{\hat{y}})_t - \eta \mathbf{x}$
    - Decrement weights

Multi-class Perceptron Learning Algorithm

- If $\hat{y} = y$, then do nothing
- Else:
  - For the correct class y, $\dfrac{\partial L}{\partial z_y} = -1$
    - $(\nabla L)_y = -\mathbf{x}$
    - $(\theta_y)_{t+1} = (\theta_y)_t + \eta \mathbf{x}$
    - Increment weights
  - For the predicted class $\hat{y}$, $\dfrac{\partial L}{\partial z_{\hat{y}}} = 1$
    - $(\nabla L)_{\hat{y}} = \mathbf{x}$
    - $(\theta_{\hat{y}})_{t+1} = (\theta_{\hat{y}})_t - \eta \mathbf{x}$
    - Decrement weights
  - For other classes, do nothing

Structured Perceptrons

- Perceptrons for sequence labeling

Structured Perceptrons

- Perceptrons for sequence labeling
  - Given a sequence $X = [x_1, \ldots, x_T]$, predict labels $Y = [y_1, \ldots, y_T]$

Structured Perceptrons

- Perceptrons for sequence labeling
  - Given a sequence $X = [x_1, \ldots, x_T]$, predict labels $Y = [y_1, \ldots, y_T]$
  - We do not care about the probability $P(Y|X)$, just which Y has the highest score Z

Structured Perceptrons

- Perceptrons for sequence labeling
  - Given a sequence $X = [x_1, \ldots, x_T]$, predict labels $Y = [y_1, \ldots, y_T]$
  - We do not care about the probability $P(Y|X)$, just which Y has the highest score Z
- $\hat{Y} = \underset{k \in K^T}{\operatorname{argmax}} Z_k$

Structured Perceptrons

- Score Z decomposes into a sum of local parts

Structured Perceptrons

- Score Z decomposes into a sum of local parts
  - At each time step i, for each possible combination of current tag y_i and previous tag y_{i-1}, compute a local score $z(y_i, y_{i-1})$

Structured Perceptrons

- Score Z decomposes into a sum of local parts
  - At each time step i, for each possible combination of current tag y_i and previous tag y_{i-1}, compute a local score $z(y_i, y_{i-1})$
  - Use the Viterbi algorithm to combine the local scores across the sequence, and find the argmax

Structured Perceptrons

- Suppose that at each time step i, we want to predict the current tag y_i using the following features:

Structured Perceptrons

- Suppose that at each time step i, we want to predict the current tag y_i using the following features:
  - Previous tag y_{i-1}

Structured Perceptrons

- Suppose that at each time step i, we want to predict the current tag y_i using the following features:
  - Previous tag y_{i-1}
    - At the beginning of the sentence, let the previous tag y_{i-1} be the start symbol `<S>`

Structured Perceptrons

- Suppose that at each time step i, we want to predict the current tag y_i using the following features:
  - Previous tag y_{i-1}
    - At the beginning of the sentence, let the previous tag y_{i-1} be the start symbol `<S>`
  - Current word x_i

Structured Perceptrons

- Suppose that at each time step i, we want to predict the current tag y_i using the following features:
  - Previous tag y_{i-1}
    - At the beginning of the sentence, let the previous tag y_{i-1} be the start symbol `<S>`
  - Current word x_i
- For simplicity, we will assume that these are the only features, and we will ignore the bias term

Structured Perceptrons

- Suppose that at each time step i, we want to predict the current tag y_i using the following features:
  - Previous tag y_{i-1}
    - At the beginning of the sentence, let the previous tag y_{i-1} be the start symbol `<S>`
  - Current word x_i
- For simplicity, we will assume that these are the only features, and we will ignore the bias term
- Let $\mathbf{f}(X, y_i, y_{i-1}, i)$ be the feature vector at time step i

Structured Perceptrons

- Suppose that at each time step i, we want to predict the current tag y_i using the following features:
  - Previous tag y_{i-1}
    - At the beginning of the sentence, let the previous tag y_{i-1} be the start symbol `<S>`
  - Current word x_i
- For simplicity, we will assume that these are the only features, and we will ignore the bias term
- Let $\mathbf{f}(X, y_i, y_{i-1}, i)$ be the feature vector at time step i
  - Using $\mathbf{f}$ instead of $\mathbf{x}$, because features can include more than just the input

Structured Perceptrons

- We can arrange our weight matrix $\mathbf{\Theta}$ as follows:

$$\begin{bmatrix} \rule{2cm}{0.4pt} \\ \\ \\ \rule{2cm}{0.4pt} \\ \\ \\ \end{bmatrix}$$

Structured Perceptrons

- We can arrange our weight matrix $\boldsymbol{\Theta}$ as follows:

$$\left[\begin{array}{c} \pi \\ \hline \\ \\ \\ \hline \\ \\ \\ \end{array}\right]$$

- Initial features
  - $y_{i-1} = \text{<S>}, y_i = \ldots$

Structured Perceptrons

- We can arrange our weight matrix $\mathbf{\Theta}$ as follows:

$$\begin{bmatrix} \pi \\ \hline \\ \mathbf{A} \\ \\ \hline \\ \\ \end{bmatrix}$$

- Initial features
  - $y_{i-1} = \text{<S>}, y_i = \ldots$
- Transition features
  - $y_{i-1} = \ldots, y_i = \ldots$

Structured Perceptrons

- We can arrange our weight matrix $\mathbf{\Theta}$ as follows:

$$\left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array}\right]$$

- Initial features
  - $y_{i-1} = \texttt{<S>}, y_i = \ldots$
- Transition features
  - $y_{i-1} = \ldots, y_i = \ldots$
- Emission features
  - $x_i = \ldots, y_i = \ldots$

Initialization Step

- We want to compute local scores $z(y_1, \texttt{<S>})$ for each possible y_1

Initialization Step

- We want to compute local scores $z(y_1, \texttt{<S>})$ for each possible y_1
  - These are the elements of $\mathbf{z}_1 = \mathbf{f}(X, y_1, \texttt{<S>}, 1) \cdot \boldsymbol{\Theta}$

Initialization Step

$$\mathbf{z}_1 = \mathbf{f}(X, y_1, \texttt{<S>}, 1) \cdot \mathbf{\Theta}$$

$$= \left[\quad \middle| \qquad \middle| \qquad\qquad \right] \cdot \begin{bmatrix} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{bmatrix}$$

Initialization Step

$$\mathbf{z}_1 = \mathbf{f}(X, y_1, \texttt{<S>}, 1) \cdot \boldsymbol{\Theta}$$

$$= \left[\ 1 \;\middle|\; \qquad \;\middle|\; \qquad \right] \cdot \begin{bmatrix} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{bmatrix}$$

- We know that $y_{i-1} = \texttt{<S>}$

Initialization Step

$$\mathbf{z}_1 = \mathbf{f}(X, y_1, \texttt{<S>}, 1) \cdot \mathbf{\Theta}$$

$$= \left[\begin{array}{c|c|c} 1 & \mathbf{0} & \quad\quad\quad\quad \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array}\right]$$

- We know that y_{i-1} cannot be any other tag

Initialization Step

$$\mathbf{z}_1 = \mathbf{f}(X, y_1, \texttt{<S>}, 1) \cdot \boldsymbol{\Theta}$$

$$= \left[\begin{array}{c|c|c} 1 & \mathbf{0} & \mathbf{1}\{x_1 = o_1\} \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array}\right]$$

- One-hot vector of the first word

Initialization Step

$$
\begin{aligned}
\mathbf{z}_1 &= \mathbf{f}(X, y_1, \texttt{<S>}, 1) \cdot \boldsymbol{\Theta} \\
&= \left[\begin{array}{c|c|c} 1 & \mathbf{0} & \mathbf{1}\{x_1 = o_1\} \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array}\right] \\
&= 1 \cdot \pi + \mathbf{0} \cdot \mathbf{A} + \mathbf{1}\{x_1 = o_1\} \cdot \mathbf{B}
\end{aligned}
$$

Initialization Step

$$
\begin{aligned}
\mathbf{z}_1 &= \mathbf{f}(X, y_1, \texttt{<S>}, 1) \cdot \boldsymbol{\Theta} \\
&= \left[\begin{array}{c|c|c} 1 & \quad \mathbf{0} \quad & \mathbf{1}\{x_1 = o_1\} \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \\ \mathbf{A} \\ \\ \hline \\ \mathbf{B} \\ \\ \end{array}\right] \\
&= 1 \cdot \pi + \mathbf{0} \cdot \mathbf{A} + \mathbf{1}\{x_1 = o_1\} \cdot \mathbf{B} \\
&= \pi + \mathbf{b}(o_1)
\end{aligned}
$$

Initialization Step

$$
\begin{aligned}
\mathbf{z}_1 &= \mathbf{f}(X, y_1, \texttt{<S>}, 1) \cdot \boldsymbol{\Theta} \\
&= \left[\begin{array}{c|c|c} 1 & \quad \mathbf{0} \quad & \mathbf{1}\{x_1 = o_1\} \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \\ \mathbf{A} \\ \\ \hline \\ \mathbf{B} \\ \\ \end{array}\right] \\
&= 1 \cdot \pi + \mathbf{0} \cdot \mathbf{A} + \mathbf{1}\{x_1 = o_1\} \cdot \mathbf{B} \\
&= \pi + \mathbf{b}(o_1)
\end{aligned}
$$

- These local scores go into the first column of the Viterbi trellis

Recursion Step

- We want to compute local scores $z(y_i, y_{i-1})$ for each possible combination of y_i and y_{i-1}

Recursion Step

- We want to compute local scores $z(y_i, y_{i-1})$ for each possible combination of y_i and y_{i-1}
  - We can stack the feature vectors for each possible y_{i-1}, $\mathbf{f}(X, y_i, y_{i-1}, i)$, on top of each other, in order

Recursion Step

- We want to compute local scores $z(y_i, y_{i-1})$ for each possible combination of y_i and y_{i-1}
  - We can stack the feature vectors for each possible y_{i-1}, $\mathbf{f}(X, y_i, y_{i-1}, i)$, on top of each other, in order
    - Form a feature matrix $\mathbf{F}_i$

Recursion Step

- We want to compute local scores $z(y_i, y_{i-1})$ for each possible combination of y_i and y_{i-1}
  - We can stack the feature vectors for each possible y_{i-1}, $\mathbf{f}(X, y_i, y_{i-1}, i)$, on top of each other, in order
    - Form a feature matrix $\mathbf{F}_i$
  - Compute $\mathbf{Z}_i = \mathbf{F}_i \cdot \mathbf{\Theta}$

Recursion Step

$$\mathbf{Z}_i = \mathbf{F}_i \cdot \boldsymbol{\Theta}$$

$$= \left[\begin{array}{c|c|c} \quad & \quad\quad & \quad\quad\quad\quad \end{array} \right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array} \right]$$

Recursion Step

$$\mathbf{Z}_i = \mathbf{F}_i \cdot \boldsymbol{\Theta}$$

$$= \left[\begin{array}{c|c|c} \mathbf{0} & \quad & \qquad \end{array} \right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array} \right]$$

- We know that $y_{i-1} \neq$ <S>

Recursion Step

$$\mathbf{Z}_i = \mathbf{F}_i \cdot \boldsymbol{\Theta}$$

$$= \left[\begin{array}{c|c|c} \mathbf{0} & \mathbf{I} & \quad\quad\quad\quad\quad \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array}\right]$$

- Identity matrix!

Recursion Step

$$\mathbf{Z}_i = \mathbf{F}_i \cdot \boldsymbol{\Theta}$$

$$= \left[\begin{array}{c|c|c} \mathbf{0} & \mathbf{I} & \mathbf{1}\{x_i = o_i\} \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array}\right]$$

- Stack of one-hot vectors

Recursion Step

$$\mathbf{Z}_i = \mathbf{F}_i \cdot \mathbf{\Theta}$$

$$= \left[\begin{array}{c|c|c} \mathbf{0} & \mathbf{I} & \mathbf{1}\{x_i = o_i\} \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array}\right]$$

$$= 0 \cdot \pi + \mathbf{I} \cdot \mathbf{A} + \mathbf{1}\{x_i = o_i\} \cdot \mathbf{B}$$

Recursion Step

$$
\begin{aligned}
\mathbf{Z}_i &= \mathbf{F}_i \cdot \boldsymbol{\Theta} \\
&= \left[\begin{array}{c|c|c} \mathbf{0} & \mathbf{I} & \mathbf{1}\{x_i = o_i\} \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array}\right] \\
&= 0 \cdot \pi + \mathbf{I} \cdot \mathbf{A} + \mathbf{1}\{x_i = o_i\} \cdot \mathbf{B} \\
&= \mathbf{A} + \mathbf{b}(o_i)
\end{aligned}
$$

Recursion Step

$$
\begin{aligned}
\mathbf{Z}_i &= \mathbf{F}_i \cdot \boldsymbol{\Theta} \\
&= \left[\begin{array}{c|c|c} \mathbf{0} & \mathbf{I} & \mathbf{1}\{x_i = o_i\} \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array}\right] \\
&= 0 \cdot \pi + \mathbf{I} \cdot \mathbf{A} + \mathbf{1}\{x_i = o_i\} \cdot \mathbf{B} \\
&= \mathbf{A} + \mathbf{b}(o_i)
\end{aligned}
$$

- Use the Viterbi algorithm to combine these local scores with scores from the rest of the sequence

Structured Perceptron Learning Algorithm

- Use the Viterbi algorithm to compute the best tag sequence

Structured Perceptron Learning Algorithm

- Use the Viterbi algorithm to compute the best tag sequence
- If $\hat{Y} = Y$, then do nothing

Structured Perceptron Learning Algorithm

- Use the Viterbi algorithm to compute the best tag sequence
- If $\hat{Y} = Y$, then do nothing
- Else:
  - Increment weights for features in Y, decrement weights for features in $\hat{Y}$

Structured Perceptron Learning Algorithm

- Use the Viterbi algorithm to compute the best tag sequence
- If $\hat{Y} = Y$, then do nothing
- Else:
  - Increment weights for features in Y, decrement weights for features in $\hat{Y}$
  - In other words, for each time step i:

Structured Perceptron Learning Algorithm

- Use the Viterbi algorithm to compute the best tag sequence
- If $\hat{Y} = Y$, then do nothing
- Else:
  - Increment weights for features in Y, decrement weights for features in $\hat{Y}$
  - In other words, for each time step i:
    - Increment weights for features in y_i, decrement weights for features in $\hat{y}_i$

Structured Perceptron Learning Algorithm

- Use the Viterbi algorithm to compute the best tag sequence
- If $\hat{Y} = Y$, then do nothing
- Else:
  - Increment weights for features in Y, decrement weights for features in $\hat{Y}$
  - In other words, for each time step i:
    - Increment weights for features in y_i, decrement weights for features in $\hat{y}_i$
  - Nothing fancy; no Numpy tricks needed