Neural Attention Mechanisms

Guest Lecture: Deep Structured Prediction

Vlad Niculae

@vnfrombucharest

(Bahdanau et al., 2015)

Sequence-to-Sequence With Attention

```
words = [21, 79, 14]  # indices

embed = Embedding(vocab_sz, dim)
E = embed(words)  # (3 × dim)
enc = LSTM(dim, dim)
H = enc(E)  # (3 × dim)

dec = LSTM(2 * dim, dim)
initialize k=1, q[0], y[0]

while not done:
  s = H @ q[k - 1]  # attn scores
  # s = [-.3, -1.0, 1.8]

  p = softmax(s)  # attn proba
  # p = [.10, .05, .85]

  a = p @ H  # (1 × dim)

  q[k] = dec(a, y[k - 1], q[k - 1])
  y[k] = W_out @ q[k] + b
  k += 1
```

United Nations elections

(Bahdanau et al., 2015)

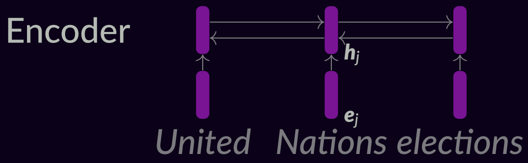
Sequence-to-Sequence With Attention

Encoder


```
words = [21, 79, 14]  # indices

embed = Embedding(vocab_sz, dim)
E = embed(words)  # (3 × dim)
enc = LSTM(dim, dim)
H = enc(E)  # (3 × dim)

dec = LSTM(2 * dim, dim)
initialize k=1, q[0], y[0]

while not done:
  s = H @ q[k - 1]  # attn scores
  # s = [-.3, -1.0, 1.8]

  p = softmax(s)  # attn proba
  # p = [.10, .05, .85]

  a = p @ H  # (1 × dim)

  q[k] = dec(a, y[k - 1], q[k - 1])
  y[k] = W_out @ q[k] + b
  k += 1
```

(Bahdanau et al., 2015)

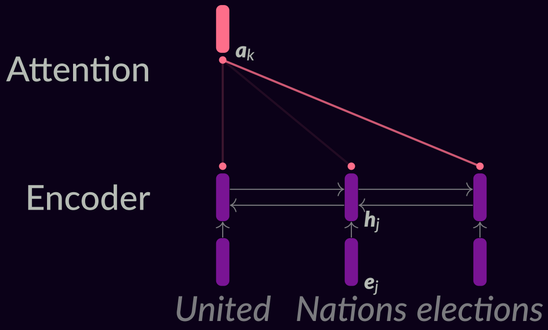
Sequence-to-Sequence With Attention




```
words = [21, 79, 14]  # indices

embed = Embedding(vocab_sz, dim)
E = embed(words)  # (3 × dim)
enc = LSTM(dim, dim)
H = enc(E)  # (3 × dim)

dec = LSTM(2 * dim, dim)
initialize k=1, q[0], y[0]

while not done:
  s = H @ q[k - 1]  # attn scores
  # s = [-.3, -1.0, 1.8]

  p = softmax(s)  # attn proba
  # p = [.10, .05, .85]

  a = p @ H  # (1 × dim)

  q[k] = dec(a, y[k - 1], q[k - 1])
  y[k] = W_out @ q[k] + b
  k += 1
```

(Bahdanau et al., 2015)

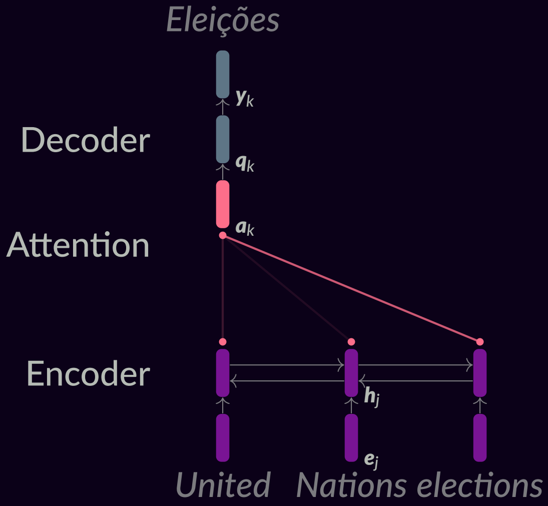
Sequence-to-Sequence With Attention




```
words = [21, 79, 14]  # indices

embed = Embedding(vocab_sz, dim)
E = embed(words)  # (3 × dim)
enc = LSTM(dim, dim)
H = enc(E)  # (3 × dim)

dec = LSTM(2 * dim, dim)
initialize k=1, q[0], y[0]

while not done:
  s = H @ q[k - 1]  # attn scores
  # s = [-.3, -1.0, 1.8]

  p = softmax(s)  # attn proba
  # p = [.10, .05, .85]

  a = p @ H  # (1 × dim)

  q[k] = dec(a, y[k - 1], q[k - 1])
  y[k] = W_out @ q[k] + b
  k += 1
```

(Bahdanau et al., 2015)

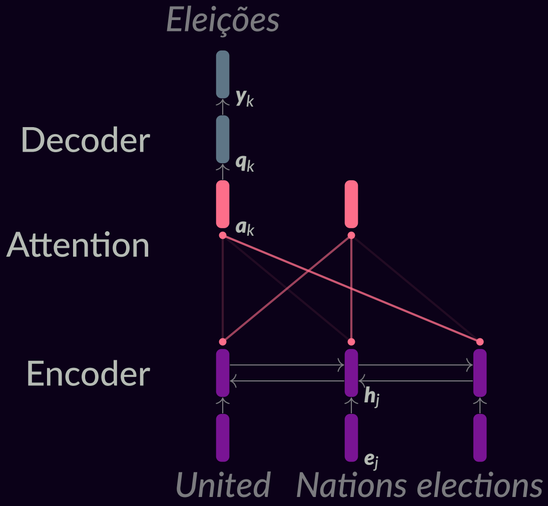
Sequence-to-Sequence With Attention




```
words = [21, 79, 14]  # indices

embed = Embedding(vocab_sz, dim)
E = embed(words)  # (3 × dim)
enc = LSTM(dim, dim)
H = enc(E)  # (3 × dim)

dec = LSTM(2 * dim, dim)
initialize k=1, q[0], y[0]

while not done:
  s = H @ q[k - 1]  # attn scores
  # s = [-.3, -1.0, 1.8]

  p = softmax(s)  # attn proba
  # p = [.10, .05, .85]

  a = p @ H  # (1 × dim)

  q[k] = dec(a, y[k - 1], q[k - 1])
  y[k] = W_out @ q[k] + b
  k += 1
```

(Bahdanau et al., 2015)

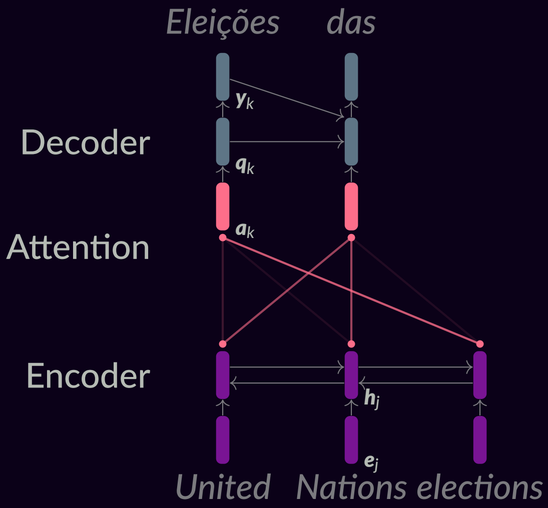
Sequence-to-Sequence With Attention




```
words = [21, 79, 14]  # indices

embed = Embedding(vocab_sz, dim)
E = embed(words)  # (3 × dim)
enc = LSTM(dim, dim)
H = enc(E)  # (3 × dim)

dec = LSTM(2 * dim, dim)
initialize k=1, q[0], y[0]

while not done:
  s = H @ q[k - 1]  # attn scores
  # s = [-.3, -1.0, 1.8]

  p = softmax(s)  # attn proba
  # p = [.10, .05, .85]

  a = p @ H  # (1 × dim)

  q[k] = dec(a, y[k - 1], q[k - 1])
  y[k] = W_out @ q[k] + b
  k += 1
```

(Bahdanau et al., 2015)

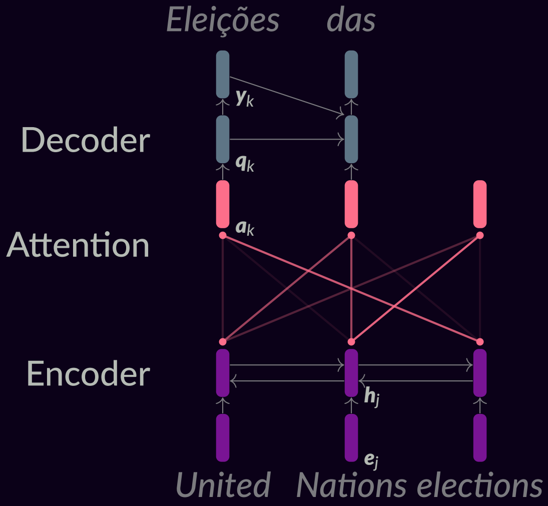
Sequence-to-Sequence With Attention




```
words = [21, 79, 14]  # indices

embed = Embedding(vocab_sz, dim)
E = embed(words)  # (3 × dim)
enc = LSTM(dim, dim)
H = enc(E)  # (3 × dim)

dec = LSTM(2 * dim, dim)
initialize k=1, q[0], y[0]

while not done:
  s = H @ q[k - 1]  # attn scores
  # s = [-.3, -1.0, 1.8]

  p = softmax(s)  # attn proba
  # p = [.10, .05, .85]

  a = p @ H  # (1 × dim)

  q[k] = dec(a, y[k - 1], q[k - 1])
  y[k] = W_out @ q[k] + b
  k += 1
```

(Bahdanau et al., 2015)

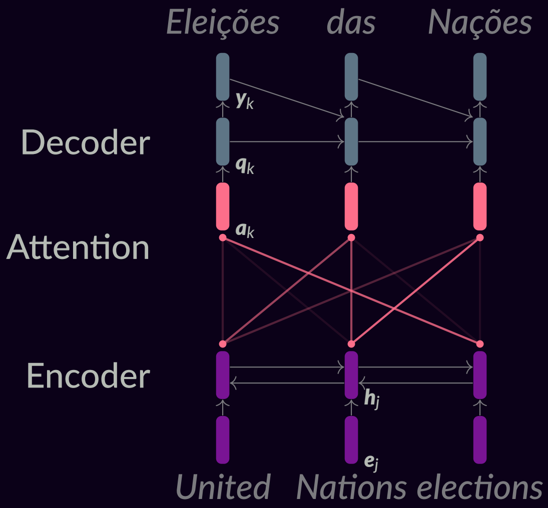
Sequence-to-Sequence With Attention




```
words = [21, 79, 14]  # indices

embed = Embedding(vocab_sz, dim)
E = embed(words)  # (3 × dim)
enc = LSTM(dim, dim)
H = enc(E)  # (3 × dim)

dec = LSTM(2 * dim, dim)
initialize k=1, q[0], y[0]

while not done:
  s = H @ q[k - 1]  # attn scores
  # s = [-.3, -1.0, 1.8]

  p = softmax(s)  # attn proba
  # p = [.10, .05, .85]

  a = p @ H  # (1 × dim)

  q[k] = dec(a, y[k - 1], q[k - 1])
  y[k] = W_out @ q[k] + b
  k += 1
```

(Bahdanau et al., 2015)

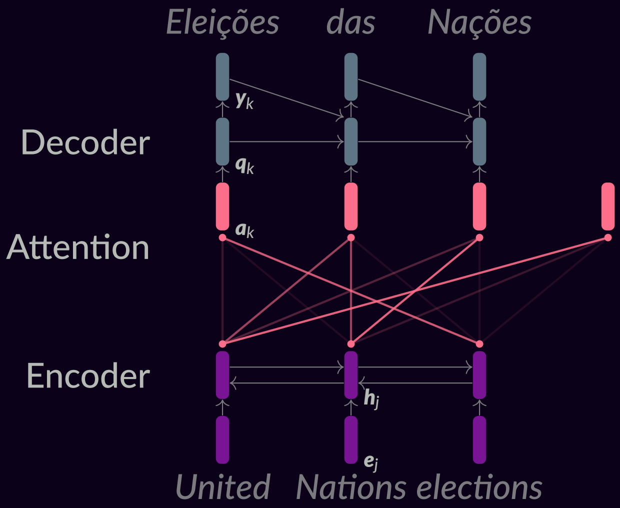
Sequence-to-Sequence With Attention




```
words = [21, 79, 14]  # indices

embed = Embedding(vocab_sz, dim)
E = embed(words)  # (3 × dim)
enc = LSTM(dim, dim)
H = enc(E)  # (3 × dim)

dec = LSTM(2 * dim, dim)
initialize k=1, q[0], y[0]

while not done:
  s = H @ q[k - 1]  # attn scores
  # s = [-.3, -1.0, 1.8]

  p = softmax(s)  # attn proba
  # p = [.10, .05, .85]

  a = p @ H  # (1 × dim)

  q[k] = dec(a, y[k - 1], q[k - 1])
  y[k] = W_out @ q[k] + b
  k += 1
```

(Bahdanau et al., 2015)

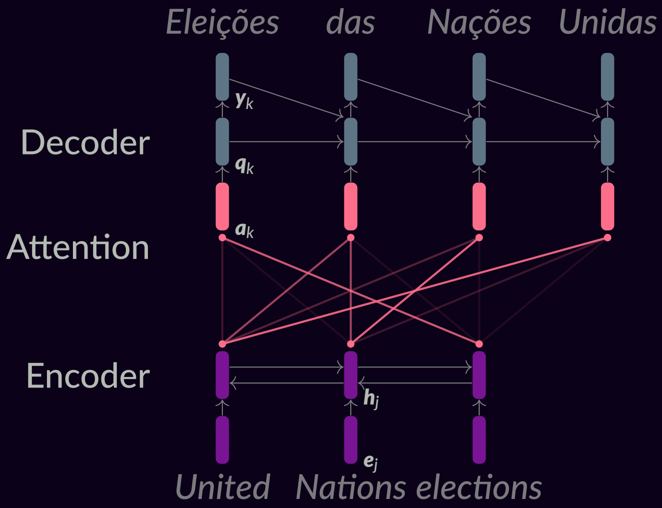
Sequence-to-Sequence With Attention




```
words = [21, 79, 14]  # indices

embed = Embedding(vocab_sz, dim)
E = embed(words)  # (3 × dim)
enc = LSTM(dim, dim)
H = enc(E)  # (3 × dim)

dec = LSTM(2 * dim, dim)
initialize k=1, q[0], y[0]

while not done:
  s = H @ q[k - 1]  # attn scores
  # s = [-.3, -1.0, 1.8]

  p = softmax(s)  # attn proba
  # p = [.10, .05, .85]

  a = p @ H  # (1 × dim)

  q[k] = dec(a, y[k - 1], q[k - 1])
  y[k] = W_out @ q[k] + b
  k += 1
```

(Bahdanau et al., 2015)

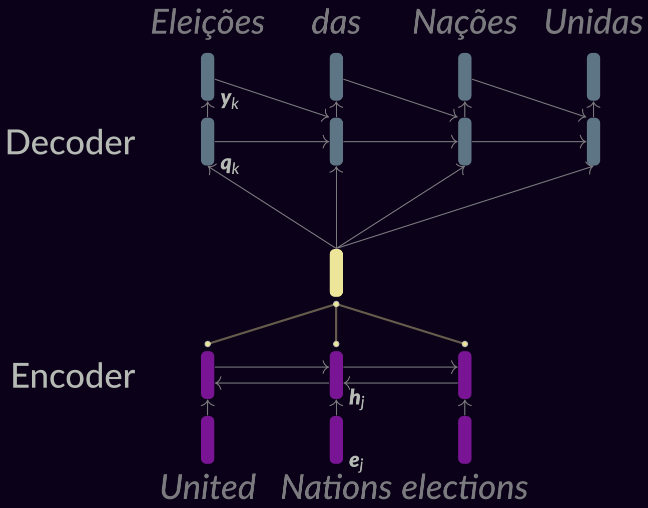
Sequence-to-Sequence With Attention




```
words = [21, 79, 14]  # indices

embed = Embedding(vocab_sz, dim)
E = embed(words)  # (3 × dim)
enc = LSTM(dim, dim)
H = enc(E)  # (3 × dim)

dec = LSTM(2 * dim, dim)
initialize k=1, q[0], y[0]

while not done:
  s = H @ q[k - 1]  # attn scores
  # s = [-.3, -1.0, 1.8]

  p = softmax(s)  # attn proba
  # p = [.10, .05, .85]

  a = p @ H  # (1 × dim)

  q[k] = dec(a, y[k - 1], q[k - 1])
  y[k] = W_out @ q[k] + b
  k += 1
```

(Bahdanau et al., 2015)

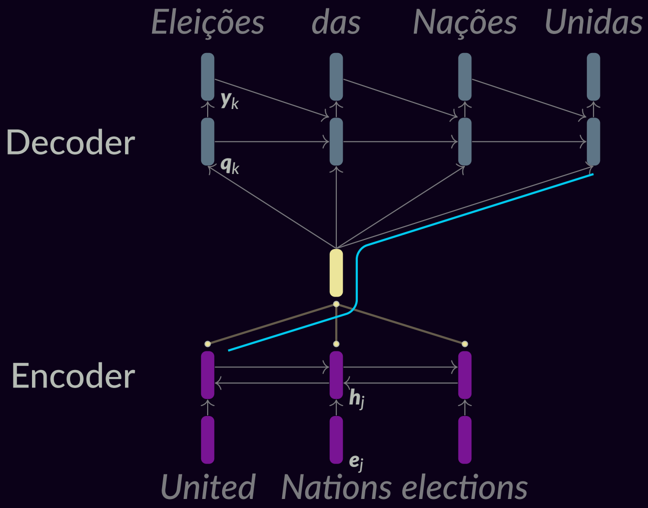
Sequence-to-Sequence With Attention




```
words = [21, 79, 14]  # indices

embed = Embedding(vocab_sz, dim)
E = embed(words)  # (3 × dim)
enc = LSTM(dim, dim)
H = enc(E)  # (3 × dim)

dec = LSTM(2 * dim, dim)
initialize k=1, q[0], y[0]

while not done:
  s = H @ q[k - 1]  # attn scores
  # s = [-.3, -1.0, 1.8]

  p = softmax(s)  # attn proba
  # p = [.10, .05, .85]

  a = p @ H  # (1 × dim)

  q[k] = dec(a, y[k - 1], q[k - 1])
  y[k] = W_out @ q[k] + b
  k += 1
```

(Bahdanau et al., 2015)

Sequence-to-Sequence With Attention




```
words = [21, 79, 14]  # indices

embed = Embedding(vocab_sz, dim)
E = embed(words)  # (3 × dim)
enc = LSTM(dim, dim)
H = enc(E)  # (3 × dim)

dec = LSTM(2 * dim, dim)
initialize k=1, q[0], y[0]

while not done:
  s = H @ q[k - 1]  # attn scores
  # s = [-.3, -1.0, 1.8]

  p = softmax(s)  # attn proba
  # p = [.10, .05, .85]

  a = p @ H  # (1 × dim)

  q[k] = dec(a, y[k - 1], q[k - 1])
  y[k] = W_out @ q[k] + b
  k += 1
```

Attention as a shortcut

Attention doesn't make models more expressive,
it makes it easier to express "better" functions.

"You May Not Need Attention" for NMT,
but reordering is needed for good results.

(Press and Smith, 2018)

```
# attention scores:
s = H @ W_attn @ state
# s = [-.3, -1.0, 1.8]

p = softmax(s)
# p = [.10, .05, .85]
```

record scratch

freeze frame

1. How to select an item from a set?

How to select an item from a set?

United

Nations

Elections

How to select an item from a set?

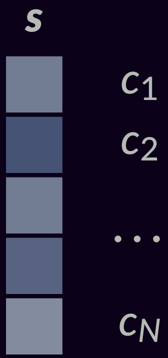
$$c_1$$

$$c_2$$

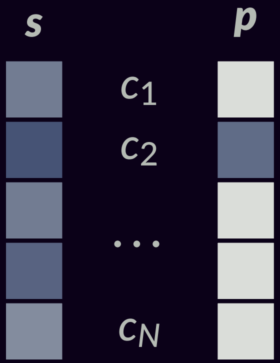
$$\cdots$$

$$c_N$$

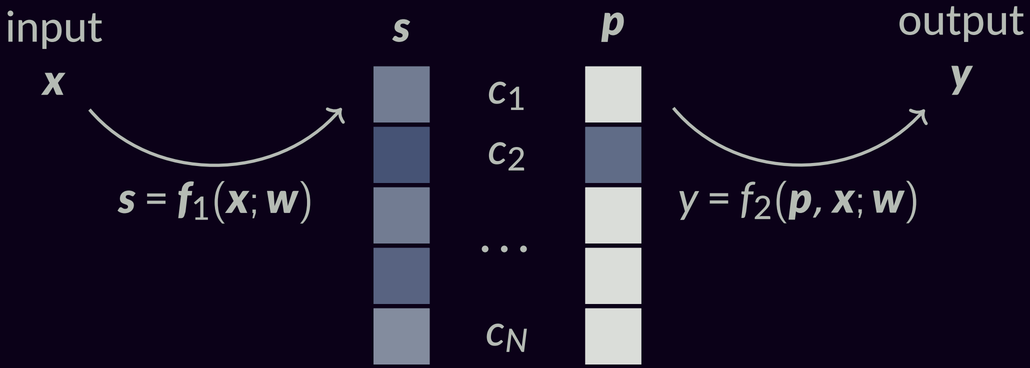
How to select an item from a set?



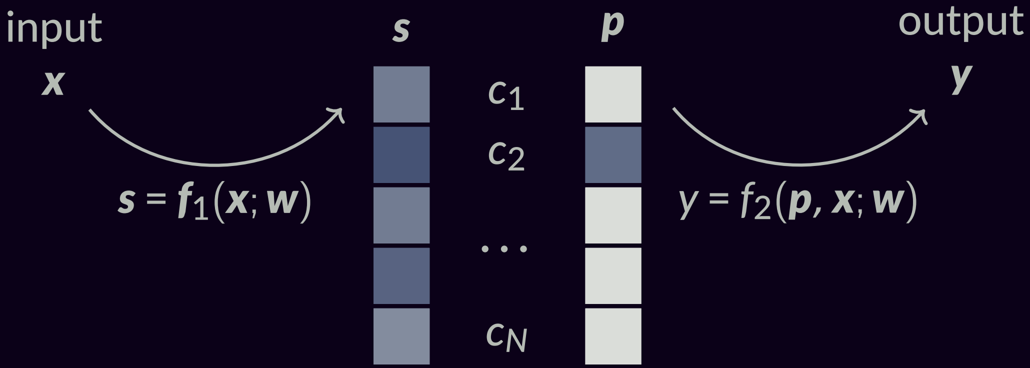
How to select an item from a set?



How to select an item from a set?

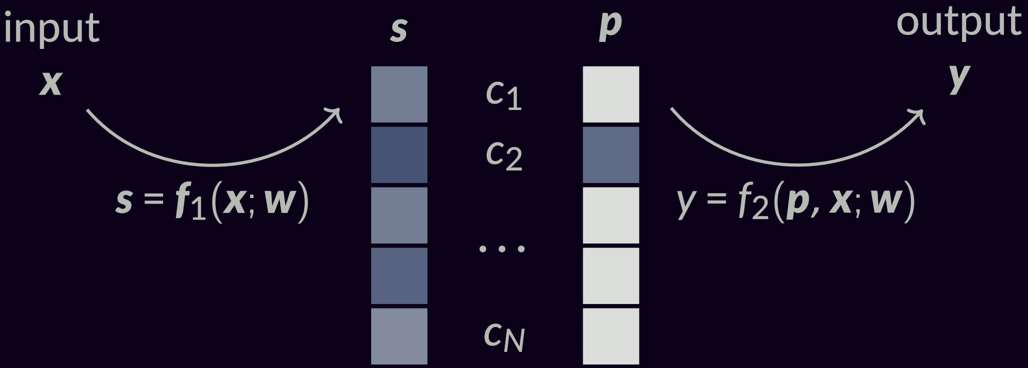


How to select an item from a set?



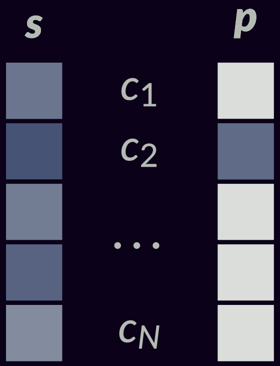
$$\frac{\partial y}{\partial \boldsymbol{w}} = ?$$

How to select an item from a set?



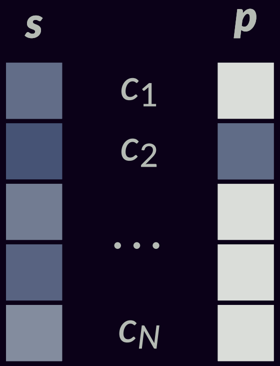
$\frac{\partial y}{\partial \boldsymbol{w}} = ?$ or, essentially, $\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} = ?$

Winner Takes It All



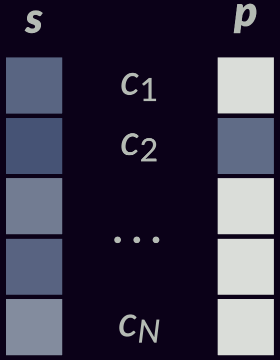
$$\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} = ?$$

Winner Takes It All



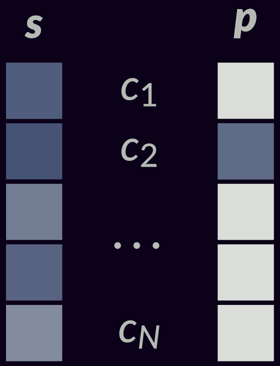
$$\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} = ?$$

Winner Takes It All



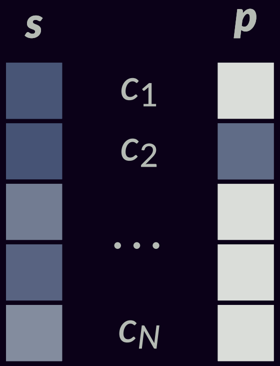
$$\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} = ?$$

Winner Takes It All



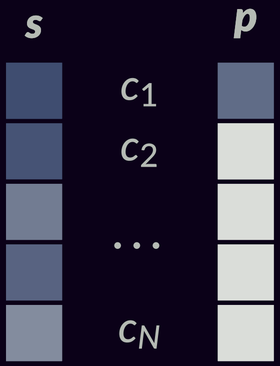
$$\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} = ?$$

Winner Takes It All



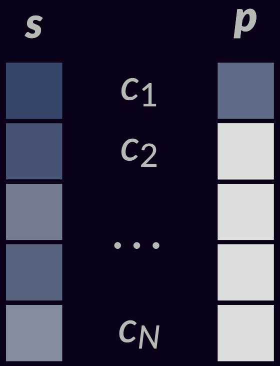
$$\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} = ?$$

Winner Takes It All



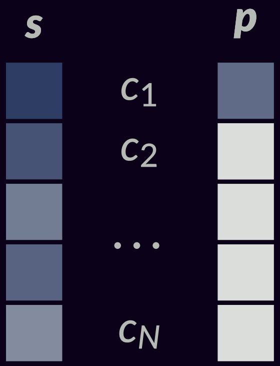
$$\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} = ?$$

Winner Takes It All



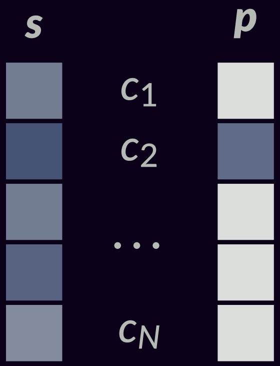
$$\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} = ?$$

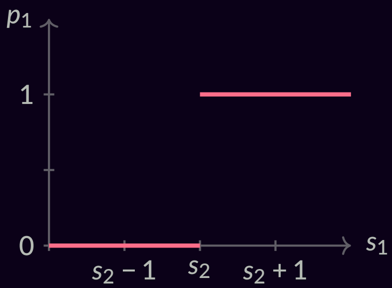
Winner Takes It All



$$\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} = ?$$

Argmax

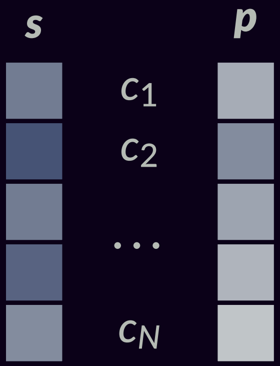


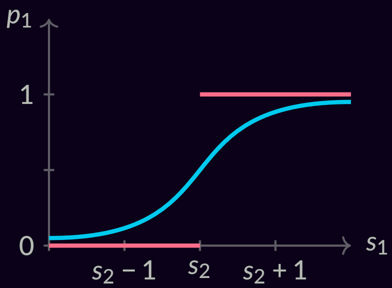


$$\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} = \mathbf{0}$$

Argmax vs. Softmax

$$p_j = \exp(s_j)/Z$$





$$\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} = \mathrm{diag}(\boldsymbol{p}) - \boldsymbol{p}\boldsymbol{p}^\top$$

Background: Optimization

$$f : \mathbb{R}^d \to \mathbb{R} \cup \{\infty\}$$

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) := v \text{ s.t. } (a)\ \exists \boldsymbol{x}^\star \in \mathbb{R}^d, f(\boldsymbol{x}^\star) = v$$

$$(b)\ \forall \boldsymbol{x}' \in \mathbb{R}^d, f(\boldsymbol{x}') \geq v$$

$$\arg\min_{\boldsymbol{x}} f(\boldsymbol{x}) := \{\boldsymbol{x}^\star \in \mathbb{R}^d : f(\boldsymbol{x}^\star) = \min_{\boldsymbol{x}} f(\boldsymbol{x})\}$$

f convex: optimization algos available

f **strictly** convex: $\arg\min_{\boldsymbol{x}} f(\boldsymbol{x}) = \{\boldsymbol{x}^\star\}$

Background: Constrained Optimization

$$\min_{\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^d} f(\boldsymbol{x})$$

The indicator function: $\mathrm{Id}_{\mathcal{X}}(\boldsymbol{x}) = \begin{cases} 0, & \boldsymbol{x} \in \mathcal{X}, \\ \infty, & \boldsymbol{x} \notin \mathcal{X}. \end{cases}$

$$\arg\min_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x}) = \arg\min_{\boldsymbol{x} \in \mathbb{R}^d} f(\boldsymbol{x}) + \mathrm{Id}_{\mathcal{X}}(\boldsymbol{x}).$$

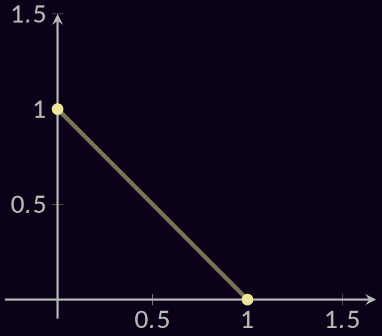
$\mathrm{Id}_{\mathcal{X}}$ is a convex function when $\mathcal{X}$ a convex set.

The Simplex

$$\triangle = \{\boldsymbol{p} \in \mathbb{R}^d : \ \boldsymbol{p} \geq \boldsymbol{0}, \ \boldsymbol{1}^\top \boldsymbol{p} = 1\}$$

The Simplex

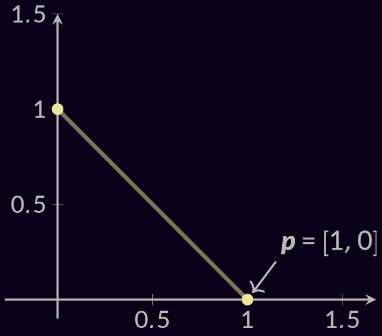
$$\Delta = \{\boldsymbol{p} \in \mathbb{R}^d : \ \boldsymbol{p} \geq \boldsymbol{0}, \ \boldsymbol{1}^\top \boldsymbol{p} = 1\}$$



$d = 2$

The Simplex

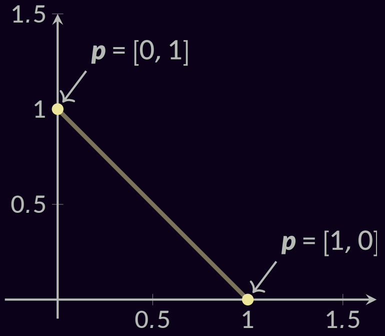
$$\Delta = \{\boldsymbol{p} \in \mathbb{R}^d : \ \boldsymbol{p} \geq \boldsymbol{0}, \ \boldsymbol{1}^\top \boldsymbol{p} = 1\}$$



$d = 2$

The Simplex

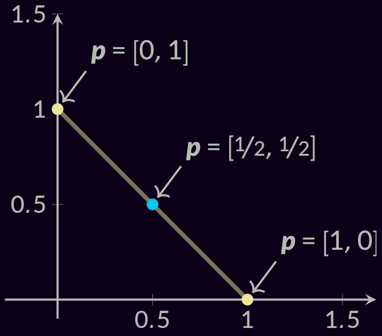
$$\Delta = \{\boldsymbol{p} \in \mathbb{R}^d : \ \boldsymbol{p} \geq \boldsymbol{0}, \ \boldsymbol{1}^\top \boldsymbol{p} = 1\}$$



$d = 2$

The Simplex

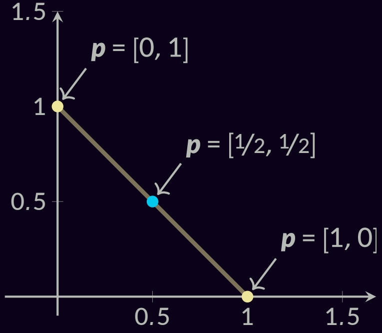
$$\Delta = \{\boldsymbol{p} \in \mathbb{R}^d : \ \boldsymbol{p} \geq \mathbf{0}, \ \mathbf{1}^\top \boldsymbol{p} = 1\}$$



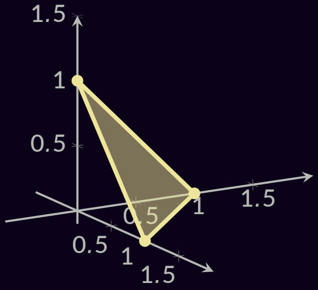
$d = 2$

The Simplex

$$\Delta = \{\boldsymbol{p} \in \mathbb{R}^d : \ \boldsymbol{p} \geq \mathbf{0}, \ \mathbf{1}^\top \boldsymbol{p} = 1\}$$



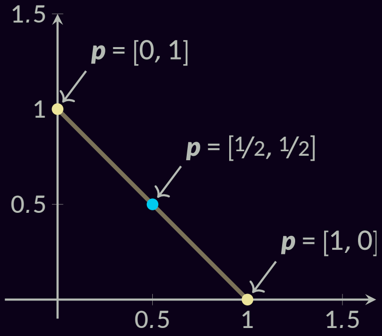
$d = 2$



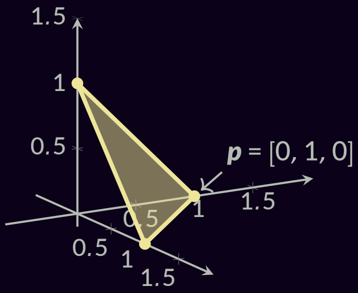
$n = 3$

The Simplex

$$\Delta = \{\boldsymbol{p} \in \mathbb{R}^d : \boldsymbol{p} \geq \mathbf{0},\ \mathbf{1}^\top \boldsymbol{p} = 1\}$$



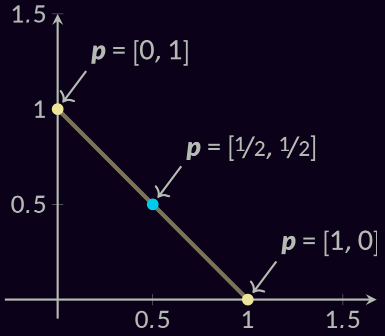
$d = 2$



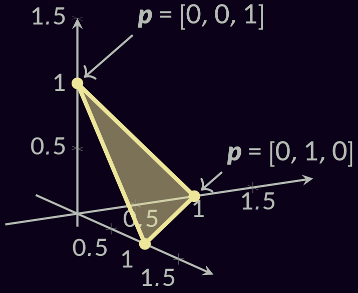
$n = 3$

The Simplex

$$\Delta = \{\boldsymbol{p} \in \mathbb{R}^d : \ \boldsymbol{p} \geq \mathbf{0}, \ \mathbf{1}^\top \boldsymbol{p} = 1\}$$



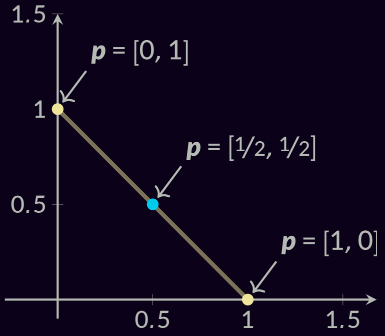
$d = 2$



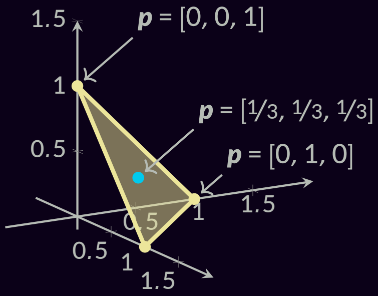
$n = 3$

The Simplex

$$\Delta = \{\boldsymbol{p} \in \mathbb{R}^d : \boldsymbol{p} \geq \mathbf{0},\ \mathbf{1}^\top \boldsymbol{p} = 1\}$$



$d = 2$

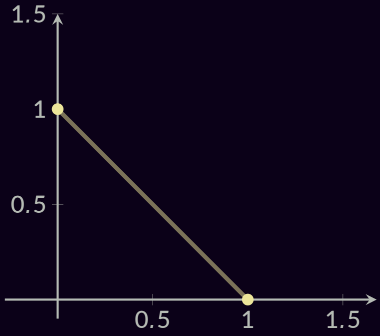


$n = 3$

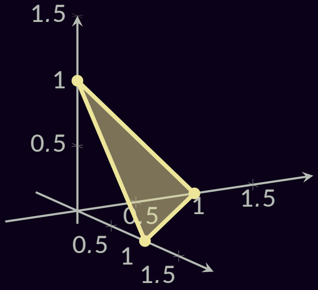
Highest Element of a Vector

$$\max_j s_j = \max_{\boldsymbol{p} \in \Delta} \boldsymbol{p}^\top \boldsymbol{s}$$

Fundamental Thm. Lin. Prog.
(Dantzig et al., 1955)



$d = 2$

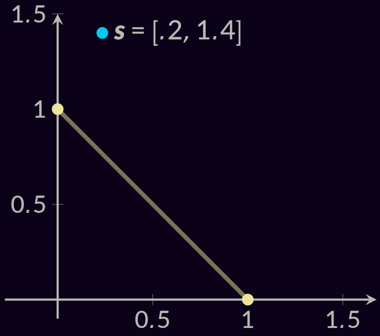


$n = 3$

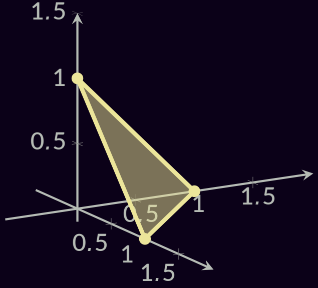
Highest Element of a Vector

$$\max_j s_j = \max_{\boldsymbol{p} \in \Delta} \boldsymbol{p}^\top \boldsymbol{s}$$

Fundamental Thm. Lin. Prog.
(Dantzig et al., 1955)



$d = 2$

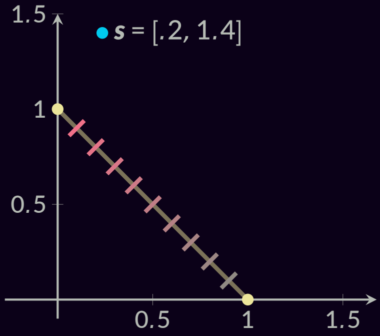


$n = 3$

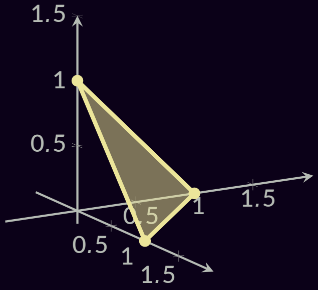
Highest Element of a Vector

$$\max_j s_j = \max_{\boldsymbol{p} \in \Delta} \boldsymbol{p}^\top \boldsymbol{s}$$

Fundamental Thm. Lin. Prog.
(Dantzig et al., 1955)



$d = 2$

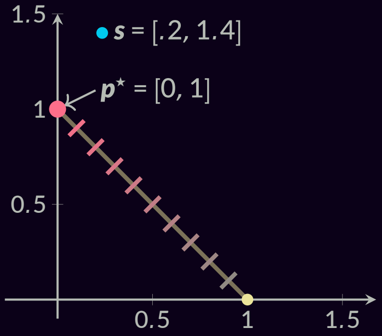


$n = 3$

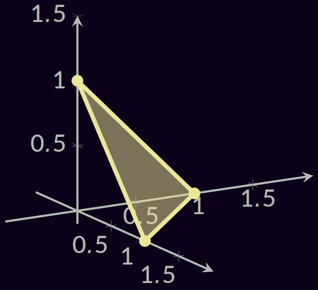
Highest Element of a Vector

$$\max_j s_j = \max_{\boldsymbol{p} \in \triangle} \boldsymbol{p}^\top \boldsymbol{s}$$

Fundamental Thm. Lin. Prog.
(Dantzig et al., 1955)



$d = 2$

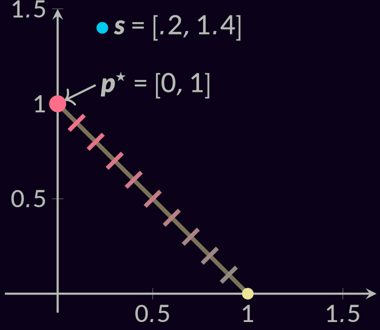


$n = 3$

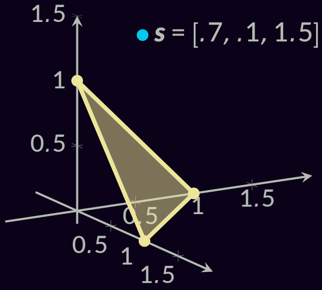
Highest Element of a Vector

$$\max_j s_j = \max_{\boldsymbol{p}\in\Delta} \boldsymbol{p}^\top \boldsymbol{s}$$

Fundamental Thm. Lin. Prog.
(Dantzig et al., 1955)



$d = 2$

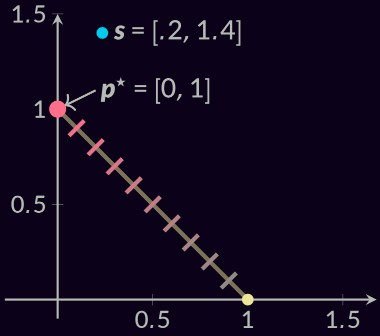


$n = 3$

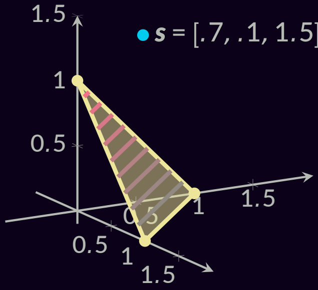
Highest Element of a Vector

$$\max_j s_j = \max_{\boldsymbol{p} \in \Delta} \boldsymbol{p}^\top \boldsymbol{s}$$

Fundamental Thm. Lin. Prog.
(Dantzig et al., 1955)



$d = 2$

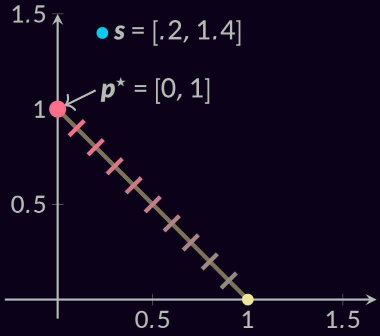


$n = 3$

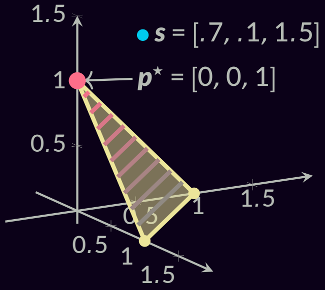
Highest Element of a Vector

$$\max_j s_j = \max_{\boldsymbol{p}\in\Delta} \boldsymbol{p}^\top \boldsymbol{s}$$

Fundamental Thm. Lin. Prog.
(Dantzig et al., 1955)



$d = 2$

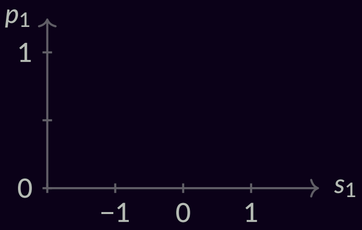


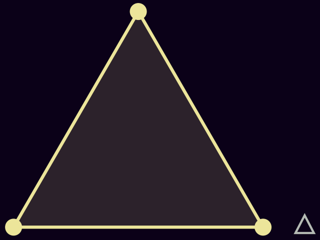
$n = 3$

(Niculae and Blondel, 2017)

Smoothed Max Operators

$$\max_\Omega(\boldsymbol{s}) = \max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} - \Omega(\boldsymbol{p})$$



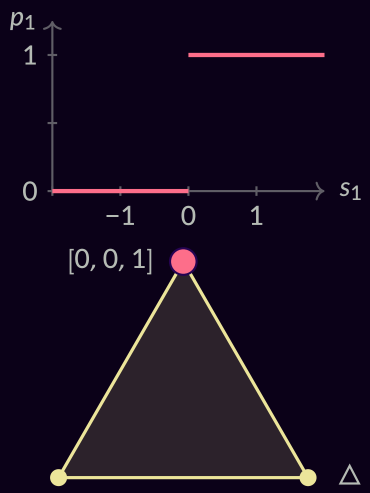


(Niculae and Blondel, 2017)

Smoothed Max Operators

$$\max_\Omega(\boldsymbol{s}) = \max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} - \Omega(\boldsymbol{p})$$

- argmax: $\Omega(\boldsymbol{p}) = 0$

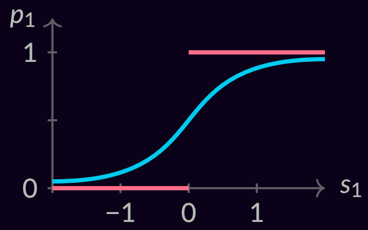


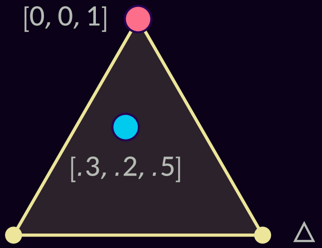
(Niculae and Blondel, 2017)

Smoothed Max Operators

$$\max_{\Omega}(\boldsymbol{s}) = \max_{\boldsymbol{p}\in\Delta} \boldsymbol{p}^\top \boldsymbol{s} - \Omega(\boldsymbol{p})$$

- argmax: $\Omega(\boldsymbol{p}) = 0$
- softmax: $\Omega(\boldsymbol{p}) = \sum_j p_j \log p_j$



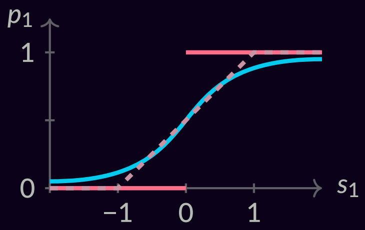


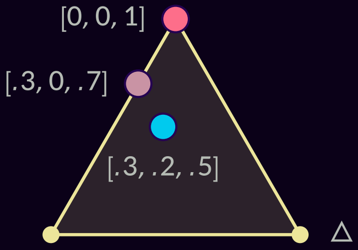
(Niculae and Blondel, 2017)

Smoothed Max Operators

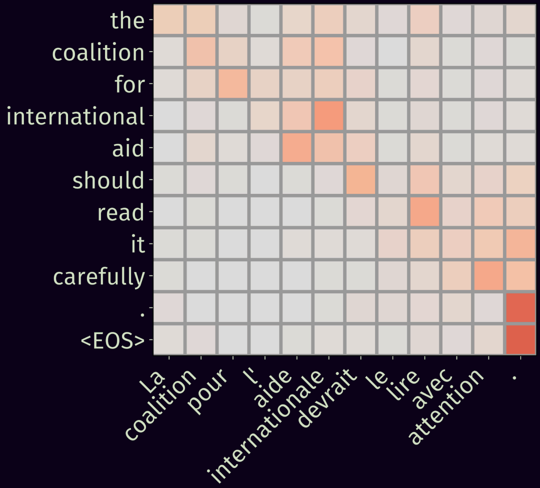
$$\max_{\Omega}(\boldsymbol{s}) = \max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^{\top}\boldsymbol{s} - \Omega(\boldsymbol{p})$$

- argmax: $\Omega(\boldsymbol{p}) = 0$
- softmax: $\Omega(\boldsymbol{p}) = \sum_j p_j \log p_j$
- sparsemax: $\Omega(\boldsymbol{p}) = 1/2\|\boldsymbol{p}\|_2^2$

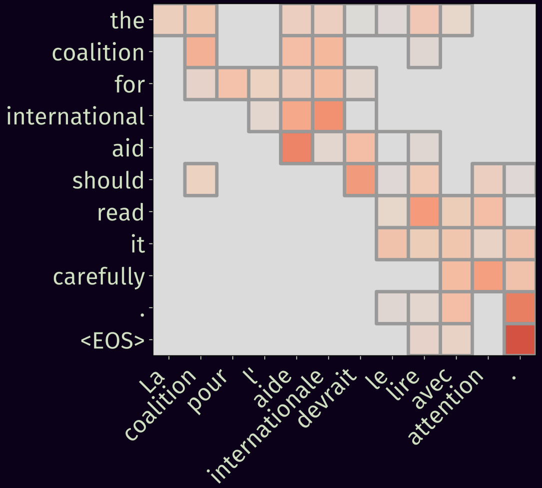




(Martins and Astudillo, 2016)


the
coalition
for
international
aid
should
read
it
carefully
.
<EOS>
La
coalition
pour
l'
aide
internationale
devrait
le
lire
avec
attention
.

softmax


the
coalition
for
international
aid
should
read
it
carefully
.
<EOS>
La
coalition
pour
l'
aide
internationale
devrait
le
lire
avec
attention
.

sparsemax

Sparsemax

$$\operatorname{sparsemax}(\boldsymbol{s}) = \arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} - 1/2\|\boldsymbol{p}\|_2^2$$

$$= \arg\min_{\boldsymbol{p}\in\triangle} \|\boldsymbol{p} - \boldsymbol{s}\|_2^2$$

Computation:

$$\boldsymbol{p}^\star = [\boldsymbol{s} - \tau\mathbf{1}]_+$$

$$s_i > s_j \Rightarrow p_i \geq p_j$$

$\mathcal{O}(d)$ via partial sort

(Held et al., 1974; Brucker, 1984; Condat, 2016)

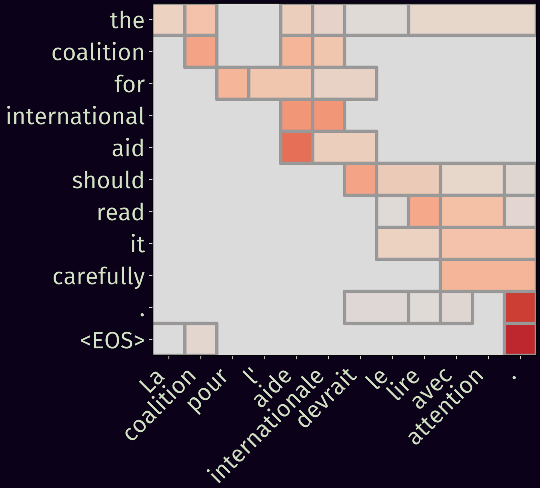
Backward pass:

$$\boldsymbol{J}_{\text{sparsemax}} = \operatorname{diag}(\boldsymbol{s}) - \frac{1}{|\mathcal{S}|}\boldsymbol{s}\boldsymbol{s}^\top$$

where $\mathcal{S} = \{j : p_j^\star > 0\}$,

$$s_j = [\![j \in \mathcal{S}]\!]$$

(Martins and Astudillo, 2016)


the
coalition
for
international
aid
should
read
it
carefully
.
<EOS>
La
coalition
pour
l'
aide
internationale
devrait
le
lire
avec
attention
.

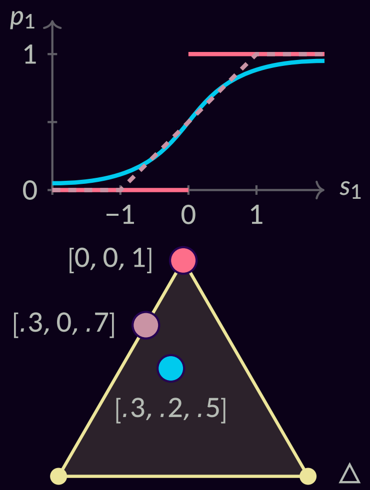
fusedmax

(Niculae and Blondel, 2017)

Smoothed Max Operators

$$\max_\Omega(\boldsymbol{s}) = \max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} - \Omega(\boldsymbol{p})$$

- argmax: $\Omega(\boldsymbol{p}) = 0$
- softmax: $\Omega(\boldsymbol{p}) = \sum_j p_j \log p_j$
- sparsemax: $\Omega(\boldsymbol{p}) = 1/2 \|\boldsymbol{p}\|_2^2$



Fusedmax

$$\text{fusedmax}(\boldsymbol{s}) = \underset{\boldsymbol{p}\in\Delta}{\arg\max}\, \boldsymbol{p}^\top \boldsymbol{s} - 1/2\|\boldsymbol{p}\|_2^2 - \sum_{2\le j\le d} |p_j - p_{j-1}|$$

$$= \underset{\boldsymbol{p}\in\Delta}{\arg\min}\, \|\boldsymbol{p} - \boldsymbol{s}\|_2^2 + \sum_{2\le j\le d} |p_j - p_{j-1}|$$

$$\text{prox}_{\text{fused}}(\boldsymbol{s}) = \underset{\boldsymbol{p}\in\mathbb{R}^d}{\arg\min}\, \|\boldsymbol{p} - \boldsymbol{s}\|_2^2 + \sum_{2\le j\le d} |p_j - p_{j-1}|$$

Proposition: $\text{fusedmax}(\boldsymbol{s}) = \text{sparsemax}\big(\text{prox}_{\text{fused}}(\boldsymbol{s})\big)$

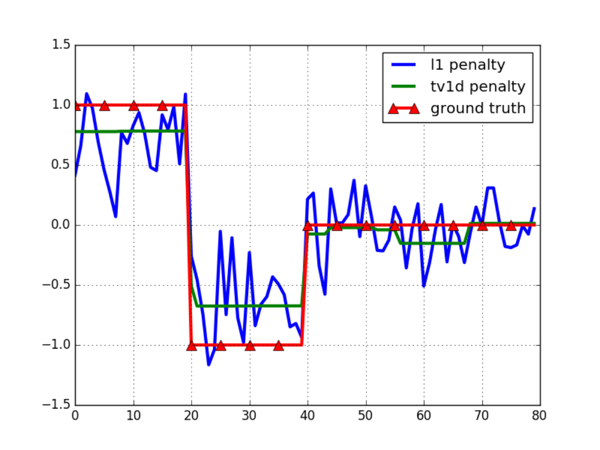
(Niculae and Blondel, 2017)

fusedmax $\ldots p_j - p_{j-1}|$

$\ldots{-1}|$

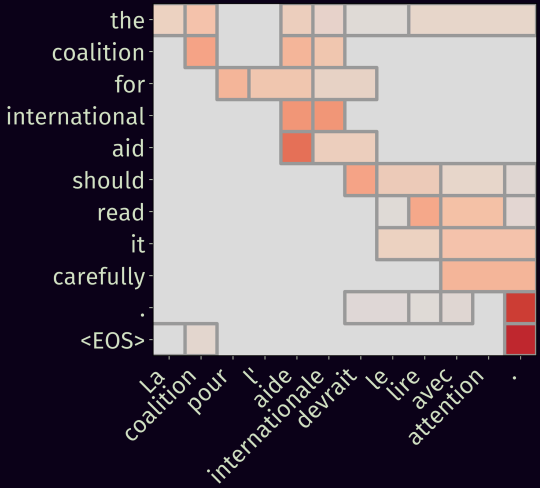
$\text{prox}_{\text{fused}} \ldots{-1}|$

Propos $\ldots\text{fused}(\boldsymbol{s}))$



"Fused Lasso" a.k.a. 1-d Total Variation

(Tibshirani et al., 2005)



the
coalition
for
international
aid
should
read
it
carefully
.
<EOS>
La
coalition
pour
l'
aide
internationale
devrait
le
lire
avec
attention
.

fusedmax

Constrained Attention

$$\begin{aligned} &\underset{\boldsymbol{p}\in\triangle\cup\mathcal{X}}{\arg\max}\ \boldsymbol{p}^\top \boldsymbol{s} - \Omega(\boldsymbol{p}) \\ = &\underset{\boldsymbol{p}\in\triangle}{\arg\max}\ \boldsymbol{p}^\top \boldsymbol{s} - \underbrace{\Omega_{\mathcal{X}}(\boldsymbol{p})}_{\Omega + \mathrm{Id}_{\mathcal{X}}} \end{aligned}$$

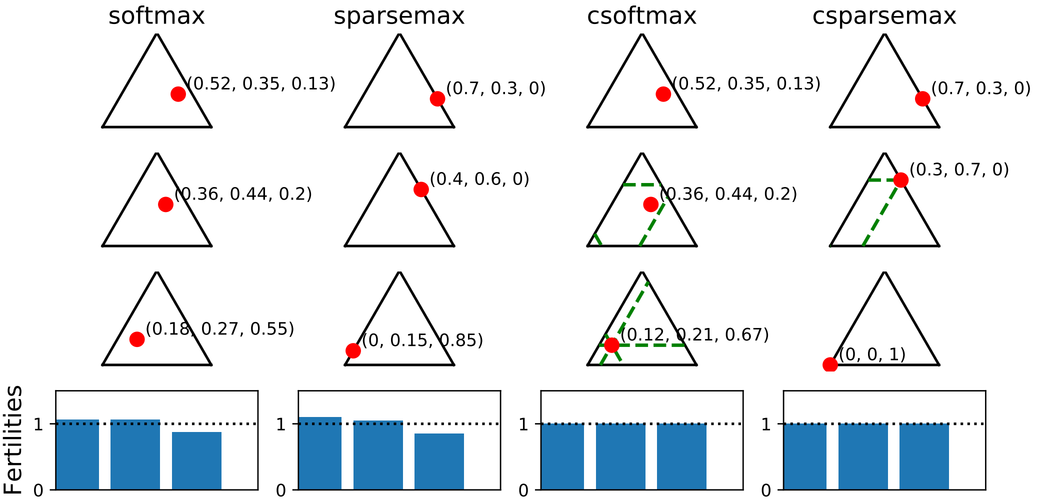
Constrained Attention

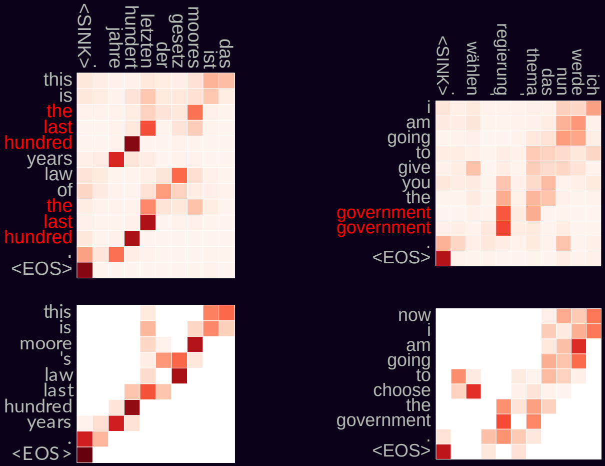
$$\arg\max_{\boldsymbol{p} \in \triangle \cup \mathcal{X}} \boldsymbol{p}^\top \boldsymbol{s} - \Omega(\boldsymbol{p})$$

$$= \arg\max_{\boldsymbol{p} \in \triangle} \boldsymbol{p}^\top \boldsymbol{s} - \underbrace{\Omega_{\mathcal{X}}(\boldsymbol{p})}_{\Omega + \mathrm{Id}_{\mathcal{X}}}$$

Example: upper bounds $\mathcal{X} = \{\boldsymbol{p} \in \mathbb{R}^d : p_j \leq b_j\}$
constrained softmax (Martins and Kreutzer, 2017) and sparsemax (Malaviya et al., 2018)
Application: incorporating fertility in Neural MT

Example: Source Sentence with Three Words




<SINK>
.
jahre
hundert
letzten
der
gesetz
moores
ist
das
this
is
the
last
hundred
years
law
of
the
last
hundred
.
<EOS>
<SINK>
.
wählen
regierung
,
.
thema
das
nun
werde
ich
i
am
going
to
give
you
the
government
government
.
<EOS>
this
is
moore
's
law
last
hundred
years
.
<EOS>
now
i
am
going
to
choose
the
government
.
<EOS>

(Niculae and Blondel, 2017)

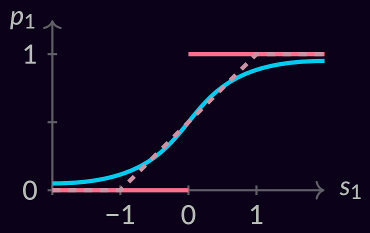
Smoothed Max Operators

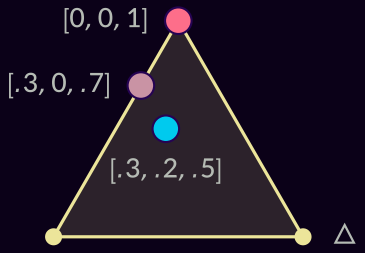
$$\max_\Omega(\boldsymbol{s}) = \max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} - \Omega(\boldsymbol{p})$$

- argmax: $\Omega(\boldsymbol{p}) = 0$
- softmax: $\Omega(\boldsymbol{p}) = \sum_j p_j \log p_j$
- sparsemax: $\Omega(\boldsymbol{p}) = 1/2\|\boldsymbol{p}\|_2^2$

fusedmax: $\Omega(\boldsymbol{p}) = 1/2\|\boldsymbol{p}\|_2^2 + \sum_j |p_j - p_{j-1}|$

csparsemax: $\Omega(\boldsymbol{p}) = 1/2\|\boldsymbol{p}\|_2^2 + \mathrm{Id}_{\boldsymbol{p}\leq\boldsymbol{b}}$

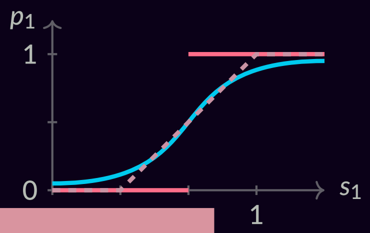




(Niculae and Blondel, 2017)

Smoothed Max Operators

$$\max_\Omega(\boldsymbol{s}) = \max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} - \Omega(\boldsymbol{p})$$

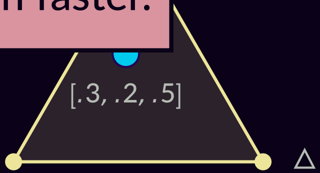


- argmax: $\Omega(\boldsymbol{p}) = 0$
- softma
- sparsema

fusedmax: $\Omega(\boldsymbol{p}) = 1/2\|\boldsymbol{p}\|_2^2 + \sum_j |p_j - p_{j-1}|$

csparsemax: $\Omega(\boldsymbol{p}) = 1/2\|\boldsymbol{p}\|_2^2 + \mathrm{Id}_{\boldsymbol{p}\le\boldsymbol{b}}$

Black-box solvers available (e.g. FISTA), specialized solvers can be much faster.



2. Attention architectures.

Computing the scores

$$s_j = \sigma(\boldsymbol{h}_j, \boldsymbol{q})$$

name	$\sigma(\boldsymbol{h}, \boldsymbol{q})$	
additive	$\boldsymbol{v}^\top \tanh(\boldsymbol{W}_1\boldsymbol{h} + \boldsymbol{W}_2\boldsymbol{q})$	(Bahdanau et al., 2015)
dot-product	$\boldsymbol{h}^\top \boldsymbol{q}$	(Luong et al., 2015)
bilinear	$\boldsymbol{h}^\top \boldsymbol{W}\boldsymbol{q}$	(Luong et al., 2015)
scaled	$(1/\sqrt{d})\, \boldsymbol{h}^\top \boldsymbol{W}\boldsymbol{q}$	(Vaswani et al., 2017)

Beyond seq2seq

The spirit of *attention mechanisms* reaches far:

- Key-Value Attention
- Multi-head Attention
- Self-Attention and the Transformer
- Hierarchical Attention
- Memory Networks, Pointer Networks, Neural Turing Machines...

(Graves et al., 2014; Sukhbaatar et al., 2015)

Key-Value Attention

idea: the objects we average *(values)*
and the objects used to compute scores *(keys)*
don't need to be identical!

$$s_j = \boldsymbol{h}_j^\top \boldsymbol{q}$$

$$\boldsymbol{u} = \mathrm{softmax}(\boldsymbol{s})^\top \boldsymbol{H}$$

$$s_j = \boldsymbol{k}_j^\top \boldsymbol{q}$$

$$\boldsymbol{u} = \mathrm{softmax}(\boldsymbol{s})^\top \boldsymbol{V}$$

(Vaswani et al., 2017)

Multi-head Attention

idea: compute k different attention averages,
& concatenate the outputs.

$$s_j = \boldsymbol{k}_j^\top \boldsymbol{q}$$

$$\boldsymbol{u} = \mathrm{softmax}(\boldsymbol{s})^\top \boldsymbol{V}$$

$$s_j^{(i)} = (\boldsymbol{W}_k^{(i)} \boldsymbol{k}_j)^\top (\boldsymbol{W}_q^{(i)} \boldsymbol{q})$$

$$\boldsymbol{u}^{(i)} = \mathrm{softmax}(\boldsymbol{s}^{(i)})^\top (\boldsymbol{V}\boldsymbol{W}_v^{(i)})$$

$$\boldsymbol{u} = [\boldsymbol{u}^{(1)}; \cdots; \boldsymbol{u}^{(k)}]$$

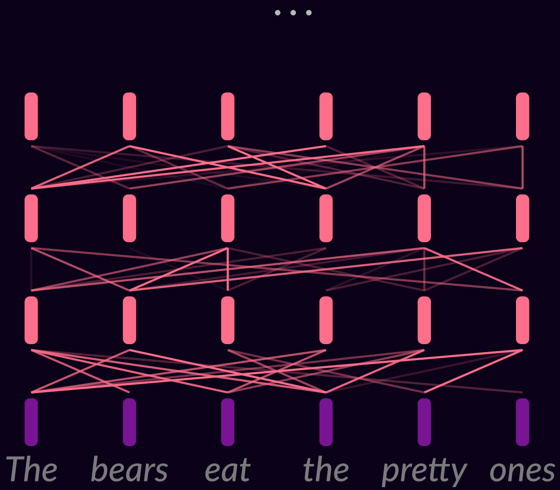
```
u = softmax(K @ q) @ V
```

```
for i in range(num_heads):
  Ki = K @ Wk[i].t()
  Vi = V @ Wv[i].t()
  qi = q @ Wq[i].t()
  ui = softmax(Ki @ qi) @ Vi
u = concat(ui)
```

(Cheng et al., 2016)

Self-attention

Attention as an *encoder layer*

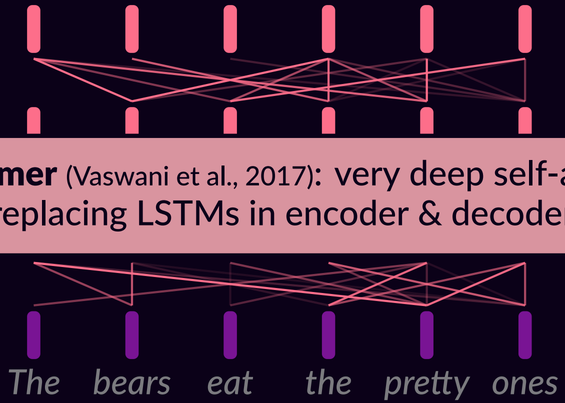


(Cheng et al., 2016)

Self-attention

Attention as an *encoder layer*

. . .

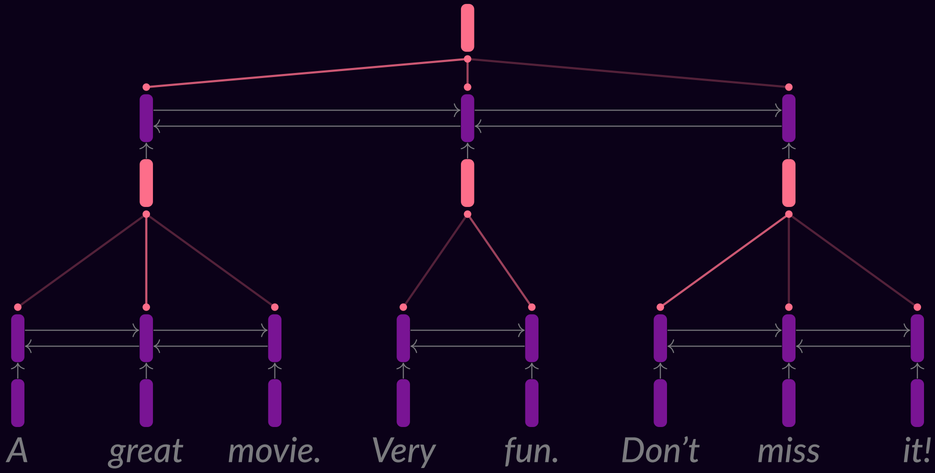


Transformer (Vaswani et al., 2017): very deep self-attention replacing LSTMs in encoder & decoder

(Yang et al., 2016)

Hierarchical Attention

Encode document by first encoding its sentences.



s

References I

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). "Neural machine translation by jointly learning to align and translate". In: *Proc. of ICLR*.

Brucker, Peter (1984). "An $O(n)$ algorithm for quadratic knapsack problems". In: *Operations Research Letters* 3.3, pp. 163–166.

Cheng, Jianpeng, Li Dong, and Mirella Lapata (2016). "Long Short-Term Memory-Networks for Machine Reading". In: *Proc. of EMNLP*.

Condat, Laurent (2016). "Fast projection onto the simplex and the ℓ_1 ball". In: *Mathematical Programming* 158.1-2, pp. 575–585.

Dantzig, George B, Alex Orden, and Philip Wolfe (1955). "The generalized simplex method for minimizing a linear form under linear inequality restraints". In: *Pacific Journal of Mathematics* 5.2, pp. 183–195.

Graves, Alex, Greg Wayne, and Ivo Danihelka (2014). "Neural Turing Machines". In: *arXiv preprint arXiv:1410.5401*.

Held, Michael, Philip Wolfe, and Harlan P Crowder (1974). "Validation of subgradient optimization". In: *Mathematical Programming* 6.1, pp. 62–88.

Luong, Minh-Thang, Hieu Pham, and Christopher D Manning (2015). "Effective approaches to attention-based neural machine translation". In: *Proc. of EMNLP*.

Malaviya, Chaitanya, Pedro Ferreira, and André F. T. Martins (2018). "Sparse and constrained attention for neural machine translation". In: *Proc. of ACL*.

Martins, André FT and Ramón Fernandez Astudillo (2016). "From softmax to sparsemax: A sparse model of attention and multi-label classification". In: *Proc. of ICML*.

References II

Martins, André FT and Julia Kreutzer (2017). “Learning What’s Easy: Fully Differentiable Neural Easy-First Taggers”. In: *Proc. of EMNLP*, pp. 349–362.

Niculae, Vlad and Mathieu Blondel (2017). “A regularized framework for sparse and structured neural attention”. In: *Proc. of NeurIPS*.

Press, Ofir and Noah A Smith (2018). “You May Not Need Attention”. In: *arXiv preprint arXiv:1810.13409*.

Sukhbaatar, Sainbayar, Jason Weston, and Rob Fergus (2015). “End-to-end memory networks”. In: *Proc. of NeurIPS*.

Tibshirani, Robert, Michael Saunders, Saharon Rosset, Ji Zhu, and Keith Knight (2005). “Sparsity and smoothness via the fused lasso”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67.1, pp. 91–108.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin (2017). “Attention Is All You Need”. In: *Proc. of NeurIPS*.

Yang, Zichao, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy (2016). “Hierarchical attention networks for document classification”. In: *Proc. of NAACL-HLT*.