Part-of-Speech Tagging

+

Neural Networks 2

CS 287

Review: Bilinear Model

Bilinear model,

$$\hat{\mathbf{y}} = f((\mathbf{x}^0\mathbf{W}^0)\mathbf{W}^1 + \mathbf{b})$$

- $\mathbf{x}^0 \in \mathbb{R}^{1\times d_0}$ start with one-hot.
- $\mathbf{W}^0 \in \mathbb{R}^{d_0\times d_{\mathrm{in}}}$, $d_0 = |\mathcal{F}|$
- $\mathbf{W}^1 \in \mathbb{R}^{d_{\mathrm{in}}\times d_{\mathrm{out}}}, \mathbf{b} \in \mathbb{R}^{1\times d_{\mathrm{out}}}$; model parameters

Notes:

- Bilinear parameter interaction.
- $d_0 >> d_{\mathrm{in}}$, e.g. $d_0 = 10000, d_{\mathrm{in}} = 50$

Review: Bilinear Model: Intuition

$$(\mathbf{x}^0\mathbf{W}^0)\mathbf{W}^1 + \mathbf{b}$$

$$\begin{bmatrix} 0 & \dots & {\color{red}1} & \dots & 0 \end{bmatrix} \begin{bmatrix} w^0_{1,1} & \dots & w^0_{0,d_{\text{in}}} \\ & \vdots & \\ & \vdots & \\ & \vdots & \\ {\color{red}w^0_{k,1}} & {\color{red}\dots} & {\color{red}w^0_{k,d_{\text{in}}}} \\ & \vdots & \\ & \vdots & \\ & \vdots & \\ w^0_{d_0,1} & \dots & w^0_{d_0,d_{\text{in}}} \end{bmatrix} \begin{bmatrix} {\color{red}w^1_{1,1}} & \dots & \dots & {\color{red}w^1_{0,d_{\text{out}}}} \\ & \ddots & \ddots & \\ {\color{red}w^1_{d_{\text{in}},0}} & \dots & \dots & {\color{red}w^1_{d_{\text{in}},d_{\text{out}}}} \end{bmatrix}$$

Review: Window Model

Goal: predict t_5.

- Windowed word model.

$$w_1 \; w_2 \; [w_3 \; w_4 \; w_5 \; w_6 \; w_7] \; w_8$$

- w_3, w_4; left context
- w_5; Word of interest
- w_6, w_7; right context
- d_{win}; size of window ($d_{\mathrm{win}} = 5$)

Review: Dense Windowed BoW Features

- $f_1, \dots, f_{d_{\mathrm{win}}}$ are words in window
- Input representation is the concatenation of embeddings

$$\mathbf{x} = [v(f_1)\ v(f_2)\ \dots\ v(f_{d_{\mathrm{win}}})]$$

Example: Tagging

$$w_1\ w_2\ [w_3\ w_4\ w_5\ w_6\ w_7]\ w_8$$

$$\mathbf{x} = [v(w_3)\ v(w_4)\ v(w_5)\ v(w_6)\ v(w_7)]$$

		$\mathbf{x}$		
$d_{\mathrm{in}}/5$	$d_{\mathrm{in}}/5$	$d_{\mathrm{in}}/5$	$d_{\mathrm{in}}/5$	$d_{\mathrm{in}}/5$

Rows of $\mathbf{W}^1$ encode position specific weights.

Quiz

We are doing tagging with a windowed bilinear model with hinge-loss and no capitalization features. The model has $d_{\mathrm{win}} = 5$, $d_{\mathrm{in}} = 50$, $d_{\mathrm{out}} = 40$, and vocabulary size 10000.

We are given the input window:

`The dog walked to the`

Unfortunately we incorrectly classify `walked` as `NN` as opposed to `VP`, in a bilinear model with a hinge-loss .

What is the maximum number of parameters that receive a non-zero gradient?

Answer:

$$
\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix}
\begin{bmatrix}
w^0_{1,1} & \cdots & w^0_{0,d_{\text{in}}} \\
w^0_{the,1} & \cdots & w^0_{the,d_{\text{in}}} \\
\vdots & & \\
w^0_{dog,1} & \cdots & w^0_{dog,d_{\text{in}}} \\
\vdots & & \\
w^0_{walked,1} & \cdots & w^0_{walked,d_{\text{in}}} \\
\vdots & & \\
w^0_{to,1} & \cdots & w^0_{to,d_{\text{in}}} \\
\vdots & & \\
w^0_{the,1} & \cdots & w^0_{the,d_{\text{in}}} \\
\vdots & & \\
w^0_{d_0,1} & \cdots & w^0_{d_0,d_{\text{in}}}
\end{bmatrix}
\begin{bmatrix}
w^1_{1,1} & \cdots & w^1_{1,NN} & \cdots & w^1_{1,VP} & w^1_{0,d_{\text{out}}} \\
& \ddots & \ddots & & & \\
w^1_{d_{\text{in}},0} & \cdots & w^1_{d_{\text{in}},NN} & \cdots & w^1_{d_{\text{in}},VP} & w^1_{d_{\text{in}},d_{\text{out}}}
\end{bmatrix}
$$

$$\mathbf{W}^0 = 5 \times d_{\text{in}}$$

$$\mathbf{W}^1 = d_{\text{in}} \times 2$$

Part-of-Speech Tagging 1

Consider the following windowed model, and assume for now a linear model.

$$\left[w_1 \ \texttt{the} \ w_3 \ w_4 \ w_5\right]$$

- What information do we have about the tag of w_3?
- What weight should the features values associated with `the` in position w_2 take?

Part-of-Speech Tagging 2

Next Consider the following windowed model, and assume for now a linear model.

$$[w_1 \ w_2 \ w_3 \ \texttt{dog} \ w_5]$$

- What information do we have about the tag of w_3?
- What weight should the features values associated with `dog` in position w_4 take?

Part-of-Speech Tagging 3

Now finally consider the following windowed model, and assume for now a linear model.

$$[w_1 \; \texttt{the} \; w_3 \; \texttt{dog} \; w_5]$$

- What information do we have about the tag of w_3?
- What weight would we want if we combined both the features values?

Contents

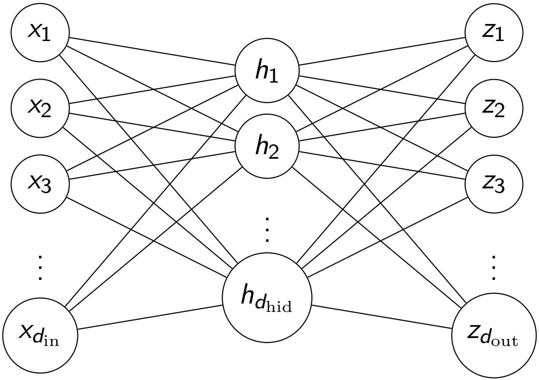
Neural Network

One-layer multi-layer perceptron architecture,

$$NN_{MLP1}(\mathbf{x}) = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1)W^2 + \mathbf{b}^2$$

- $\mathbf{x}\mathbf{W} + \mathbf{b}$; *perceptron*
- $\mathbf{x}$ is the dense representation in $\mathbb{R}^{1 \times d_{\text{in}}}$
- $\mathbf{W}^1 \in \mathbb{R}^{d_{\text{in}} \times d_{\text{hid}}}, \mathbf{b}^1 \in \mathbb{R}^{1 \times d_{\text{hid}}}$; first affine transformation
- $\mathbf{W}^2 \in \mathbb{R}^{d_{\text{hid}} \times d_{\text{out}}}, \mathbf{b}^2 \in \mathbb{R}^{1 \times d_{\text{out}}}$; second affine transformation
- $g : \mathbb{R}^{d_{\text{hid}} \times d_{\text{hid}}}$ is an *activation non-linearity* (often pointwise)
- $g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1)$ is the *hidden layer*

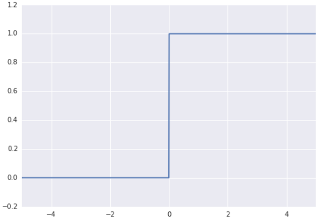
Schematic



Non-Linearities: 0/1

0/1 function:

$$0/1(t) = \mathbf{1}(t > 0)$$



- $01((\mathbf{xW}^1 + \mathbf{b}^1)_i)$
- Intuition: On, if above a threshold

Exercise

Input layer to NN_{MLP1} is the sparse indicator features of the word at each position.

- Design a network to recognize

$$[w_1 \; \texttt{the} \; w_3 \; \texttt{dog} \; w_5]$$

- Design a network to recognize where w_2 is not `the`

$$[w_1 \; w_2 \; w_3 \; \texttt{dog} \; w_5]$$

Feature Conjunctions

Many NLP tasks require conjunctive features, examples

- Sequence-based taggers look at last two-part of speech tags.
- Chinese part-of-speech taggers look at first character and last tag.
- Higher-level models (parses) look at tags of words and distances apart (example)

For some natural language tasks, conjunctions are painstakingly hard.

- NNs: Capacity to learn conjunctions and feature combinations.
- Also possible with other convex models such as SVMs

Feature Conjunctions

Many NLP tasks require conjunctive features, examples

- Sequence-based taggers look at last two-part of speech tags.
- Chinese part-of-speech taggers look at first character and last tag.
- Higher-level models (parses) look at tags of words and distances apart (example)

For some natural language tasks, conjunctions are painstakingly hard.

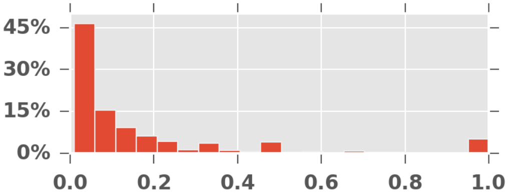
- NNs: Capacity to learn conjunctions and feature combinations.
- Also possible with other convex models such as SVMs

Simple Antecedent/Pairwise Features Not Discriminative

E.g., is [Lexus sales] the antecedent of [their sales]?

- Common pairwise features: String/Head Match, Sentences Between, Mention-Antecedent Numbers/Heads/Genders, etc.

$$\phi_{\mathrm{p}}(\text{[their sales]},\text{[Lexus sales]}) = \left\{ \begin{array}{c} \text{string-match=false} \\ \text{head-match=true} \\ \text{sentences-between=0} \\ \text{ment-ant-numbers=plur.,plur.} \\ \vdots \end{array} \right\}$$



Dealing with the Feature Problem

Finding discriminative features is a major challenge for coreference systems [Fernandes et al. 2012; Durrett and Klein 2013]

- Typical to define (or search for) feature conjunction-schemes to improve predictive performance [Fernandes et al. 2012; Durrett and Klein 2013; Björkelund and Kuhn 2014]. For instance:
  - string-match(x, y) $\wedge$ type(x) $\wedge$ type(y) [Durrett and Klein 2013], where

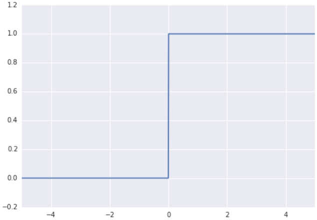
$$\text{type}(x) = \begin{cases} \text{Nom.} & \text{if } x \text{ is nominal} \\ \text{Prop.} & \text{if } x \text{ is proper} \\ \text{citation-form}(x) & \text{if } x \text{ is pronominal} \end{cases}$$

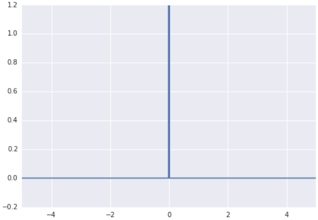
  - substring-match(head$(x), y)$ $\wedge$ substring-match$(x$, head$(y))$ $\wedge$ coarse-type(y) $\wedge$ coarse-type(x) [Björkelund and Kuhn 2014]
- Not just a problem for Mention Ranking systems!

Non-Linearities: 0/1

0/1 function:

$$0/1(t) = \mathbf{1}(t > 0)$$



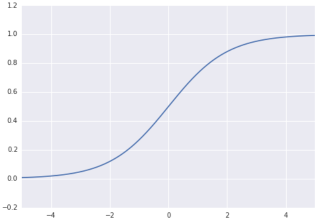


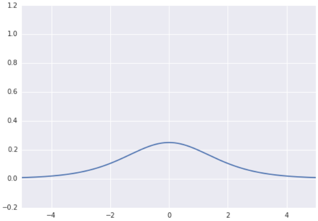
- Issue: No gradient anywhere

Non-Linear Functions: Sigmoid

Logistic sigmoid function:

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$



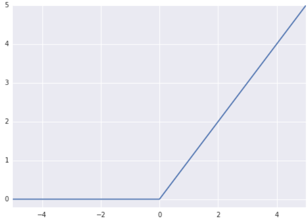


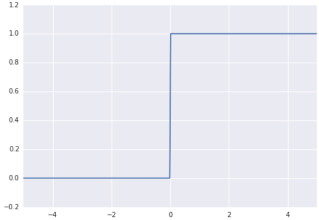
- $\sigma((\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1)_i)$
- Intuition: Each hidden dimension ("neuron") is result of logistic regression.

Other Non-Linearities: ReLU

Rectified Linear Unit:

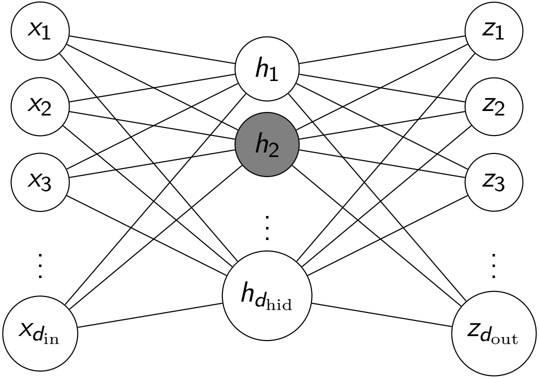
$$\mathrm{ReLU}(t) = \max\{0, t\}$$





- Intuition: Each hidden-unit gives activation margin
- No gradient (saturation) when below 0.

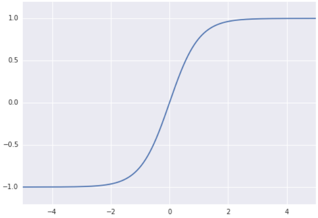
Saturation: Intuition

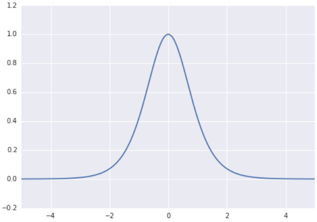


Other Non-Linearities: Tanh

Hyperbolic Tangeant:

$$\tanh(t) = \frac{\exp(t) - \exp(-t)}{\exp(t) + \exp(-t)}$$



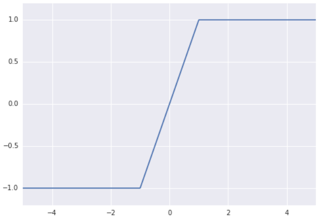


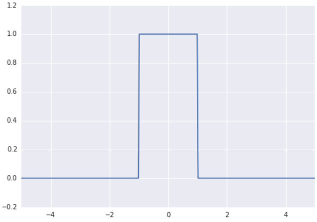
- Intuition: Similar to sigmoid, but range between 0 and -1.

Other Non-Linearities: Hard Tanh

Hyperbolic Tangeant:

$$\mathrm{hardtanh}(t) = \begin{cases} -1 & t < -1 \\ t & -1 \leq t \leq 1 \\ 1 & t > 1 \end{cases}$$



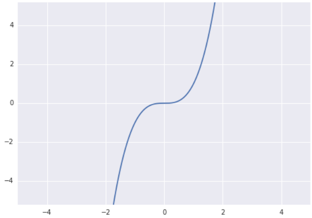


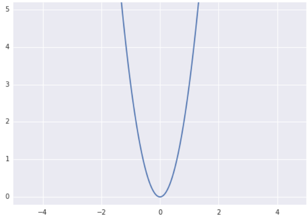
- Intuition: Similar to sigmoid, but range between 0 and -1.

Other Non-Linearities: Cube

Cube non-linearity (directly encourage parameter interaction):

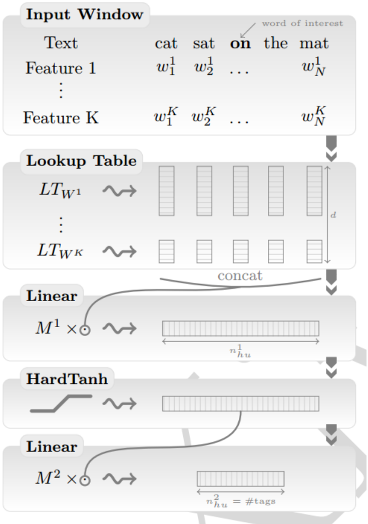
$$\text{cube}(t) = t^3$$





- Intuition: Directly encourage higher-order interactions.

Tagging from Scratch



Function Approximator

MLP1 is a universal approximator

Can approximate with any desired non-zero amount of error a family of functions that include all continuous functions on a closed and bounded subset of $\mathbb{R}^n$*, and any function mapping from any finite dimensional discrete space to another (YG)*

Caveats:

- Does not give size of hidden layer.
- Does not specify how hard this is to learn.

Deep Neural Networks (DNNs)

Can stack MLPs, create deep fully connected networks,

$$NN_{MLP1}(\mathbf{x}) = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1)W^2 + \mathbf{b}^2$$

$$NN_{MLP2}(\mathbf{x}) = g(NN_{MLP1}(\mathbf{x})\mathbf{W}^1 + \mathbf{b}^1)W^2 + \mathbf{b}^2$$

- Can have multiple hidden layers, etc.
- Benefit: may be able to find better function
- Known to be harder to train (although other approaches)

Other Layers

We will discuss many other neural network layers,

- convolutional
- attention-based
- gated layers
- ...

Highway Network

$\mathbf{y}$: output from CharCNN

Multilayer Perceptron

$$\mathbf{z} = g(\mathbf{W}\mathbf{y} + \mathbf{b})$$

Highway Network

(Srivastava, Greff, and Schmidhuber 2015)

$$\mathbf{z} = \mathbf{t} \odot g(\mathbf{W}_H\mathbf{y} + \mathbf{b}_H) + (\mathbf{1} - \mathbf{t}) \odot \mathbf{y}$$

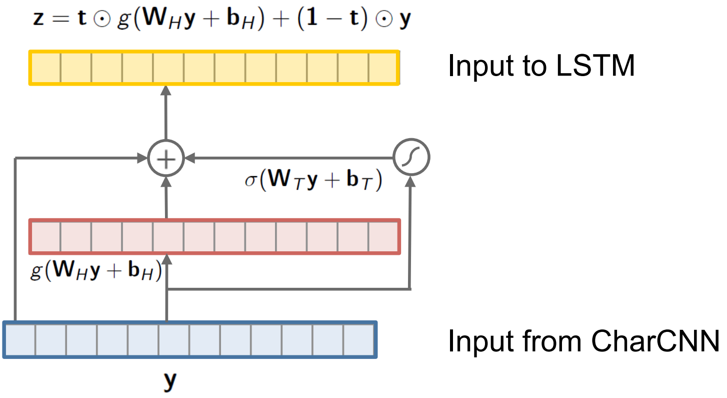
$$\mathbf{W}_H, \mathbf{b}_H : \text{Affine transformation}$$

$$\mathbf{t} = \sigma(\mathbf{W}_T\mathbf{y} + \mathbf{b}_T) : \textit{transform} \text{ gate}$$

$$\mathbf{1} - \mathbf{t} : \textit{carry} \text{ gate}$$

Hierarchical, adaptive composition of character *n*-grams.

Highway Network



Contents

Sequential Neural Network

Sequential neural networks consist of a series of composed functions,
Consider a vector-valued parameterized functions $f_1, \dots, f_k$ where

- $f_i(\mathbf{x}; \boldsymbol{\theta}_i) : \mathbb{R}^{n_{i-1}} \mapsto \mathbb{R}^{n_i}$; function
- $\boldsymbol{\theta} \in \mathbb{R}^{d_i}$; function parameters

Consider a scalar-valued loss function $L(\mathbf{y}, \hat{\mathbf{y}})$ where

- $L(\mathbf{y}, *) : \mathbb{R}^{n_k} \mapsto \mathbb{R}$; loss for input

Backpropagation

- Forward Step (f-prop):

  Compute

  $$L(f_k(\dots f_1(\mathbf{x}^0)))$$

  Saving intermediary values

  $$f_i(\dots f_1(\mathbf{x}^0)))$$

- Backward Step (b-prop):

$$\frac{\partial L}{\partial f_i(\dots f_1(\mathbf{x}^0))} = \sum_{j=1}^{n_i} \frac{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}{\partial f_i(\dots f_1(\mathbf{x}^0))} \frac{\partial L}{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}$$

$$\frac{\partial L}{\partial \theta_i} = \sum_{j=1}^{n_i} \frac{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}{\partial \theta_i} \frac{\partial L}{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}$$

Backpropagation

- Forward Step (f-prop):

  Compute
  $$L(f_k(\dots f_1(\mathbf{x}^0)))$$
  Saving intermediary values
  $$f_i(\dots f_1(\mathbf{x}^0)))$$

- Backward Step (b-prop):
  $$\frac{\partial L}{\partial f_i(\dots f_1(\mathbf{x}^0))} = \sum_{j=1}^{n_i} \frac{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}{\partial f_i(\dots f_1(\mathbf{x}^0))} \frac{\partial L}{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}$$

$$\frac{\partial L}{\partial \theta_i} = \sum_{j=1}^{n_i} \frac{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}{\partial \theta_i} \frac{\partial L}{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}$$

Backpropagation

- Forward Step (f-prop):

  Compute

$$L(f_k(\dots f_1(\mathbf{x}^0)))$$

  Saving intermediary values

$$f_i(\dots f_1(\mathbf{x}^0)))$$

- Backward Step (b-prop):

$$\frac{\partial L}{\partial f_i(\dots f_1(\mathbf{x}^0))} = \sum_{j=1}^{n_i} \frac{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}{\partial f_i(\dots f_1(\mathbf{x}^0))} \frac{\partial L}{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_i} = \sum_{j=1}^{n_i} \frac{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}{\partial \boldsymbol{\theta}_i} \frac{\partial L}{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}$$

Backpropagation: Data flow

$$
\begin{tikzcd}
f_i(\ldots f_1(\mathbf{x}^0)) \arrow[dr] & & f_{i+1}(f_i(\ldots f_1(\mathbf{x}^0))) \\
& \boxed{f_{i+1}(*;\boldsymbol{\theta}_{i+1})} \arrow[ur] \arrow[dl] & \\
\dfrac{\partial L}{\partial f_i(\ldots f_1(\mathbf{x}^0))} & \dfrac{\partial L}{\partial \boldsymbol{\theta}_{i+1}} & \dfrac{\partial L}{\partial f_{i+1}(\ldots f_1(\mathbf{x}^0))} \arrow[ul]
\end{tikzcd}
$$

Torch Implementation

Torch uses declarative unit-based specification of NN

- Every function is a represented as a unit.
- Responsibilities:
  1. Expose any parameters $\boldsymbol{\theta}_{i+1}$ as tensors
  2. Compute $f_{i+1}(\mathbf{x}, \boldsymbol{\theta}_{i+1})$ (fprop)
  3. Compute any necessary state needed for bprop
  4. Compute chain-rule given $\frac{\partial L}{\partial f_{i+1}(\ldots f_1(\mathbf{x}^0))}$ and $f_i(\ldots f_1(\mathbf{x}^0))$
  5. Compute parameter gradient $\frac{\partial L}{\partial \boldsymbol{\theta}_{i+1}}$
- Contract: forward will always be called before backward.

Torch Units

input

$f_i(\ldots f_1(\mathbf{x}^0))$

self.output

$f_{i+1}(f_i(\ldots f_1(\mathbf{x}^0)))$

weights

$f_{i+1}(*; \boldsymbol{\theta}_{i+1})$

self.gradInput

$$\frac{\partial L}{\partial f_i(\ldots f_1(\mathbf{x}^0))}$$

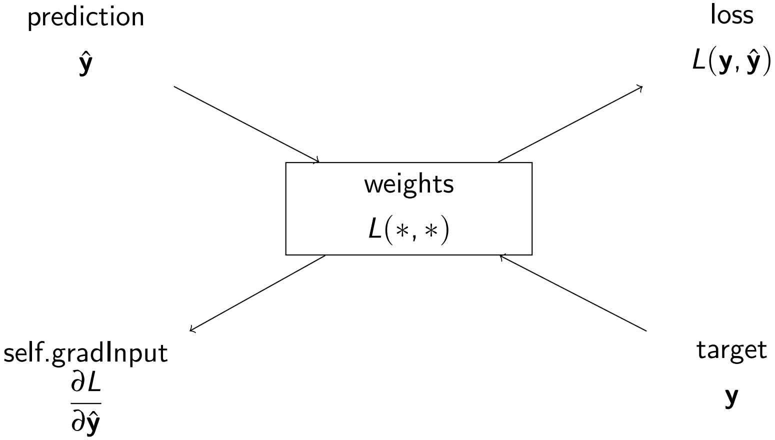
self.gradWeight

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{i+1}}$$

gradOutput

$$\frac{\partial L}{\partial f_{i+1}(\ldots f_1(\mathbf{x}^0))}$$

Loss Criterions



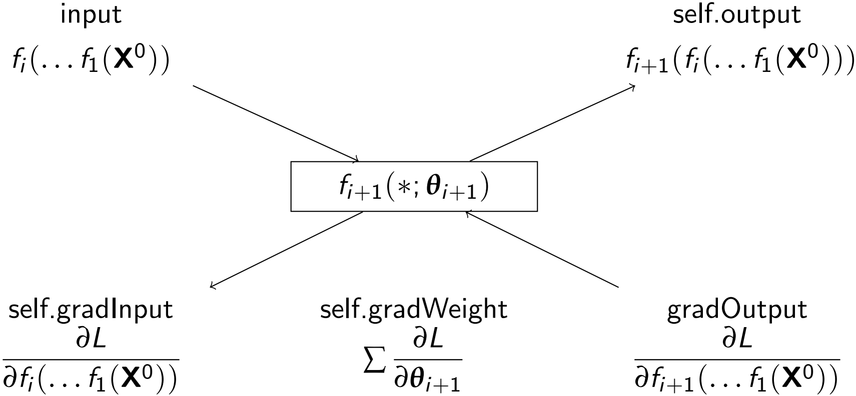
Torch Internals

- Fprop step: `self:updateOutput`
- Bprop step: `self:updateGradInput`

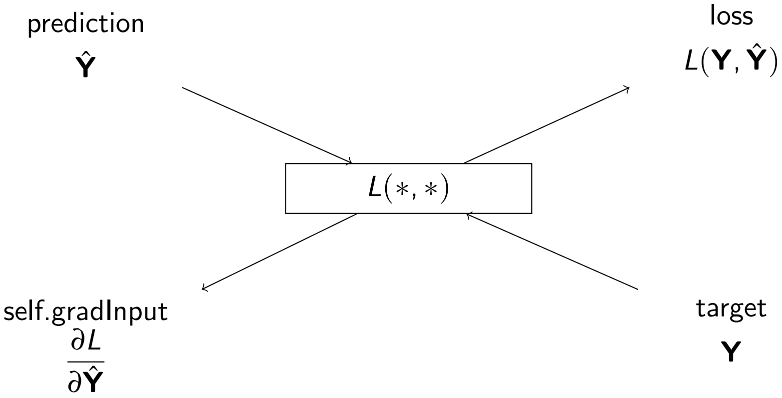
Torch Internals

- Fprop step: `self:updateOutput`
- Bprop step: `self:updateGradInput`

Torch Units: Batch



Loss Criterions



Today

- Benefits of neural networks
- Training neural networks

Next time: Pretraining and word embeddings