Deep Learning for NLP

Instructor: Thien Huu Nguyen

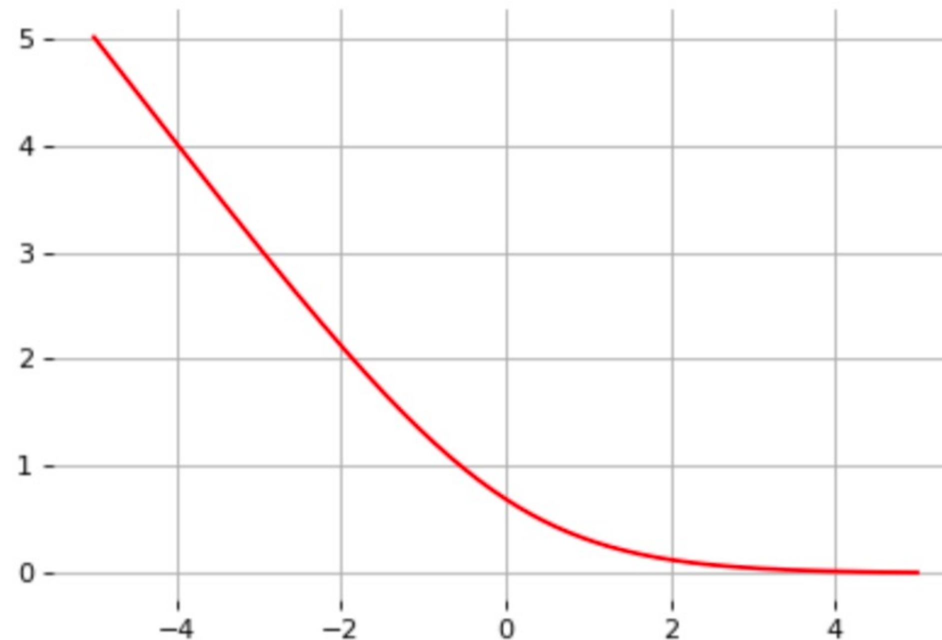
Based on slides from: Gilles Louppe, Fei-Fei Li, and Justin Johnson, and Gilles Louppe

Remember Logistic Regression?

$$P(Y = 1|\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x} + b\right)$$

The logit loss

$$\mathcal{L}(\mathbf{w}, b) = -\sum_{\mathbf{x}_i, y_i \in \mathbf{d}} \log \sigma\left(y_i(\mathbf{w}^T \mathbf{x}_i + b)\right).$$



Cross Entropy

We have,

$$\begin{aligned}
&\arg\max_{\mathbf{w},b} P(\mathbf{d}|\mathbf{w}, b) \\
&= \arg\max_{\mathbf{w},b} \prod_{\mathbf{x}_i, y_i \in \mathbf{d}} P(Y = y_i|\mathbf{x}_i, \mathbf{w}, b) \\
&= \arg\max_{\mathbf{w},b} \prod_{\mathbf{x}_i, y_i \in \mathbf{d}} \sigma(\mathbf{w}^T\mathbf{x}_i + b)^{y_i}(1 - \sigma(\mathbf{w}^T\mathbf{x}_i + b))^{1-y_i} \\
&= \arg\min_{\mathbf{w},b} \underbrace{\sum_{\mathbf{x}_i, y_i \in \mathbf{d}} -y_i \log \sigma(\mathbf{w}^T\mathbf{x}_i + b) - (1 - y_i) \log(1 - \sigma(\mathbf{w}^T\mathbf{x}_i + b))}_{\mathcal{L}(\mathbf{w},b) = \sum_i \ell(y_i, \hat{y}(\mathbf{x}_i; \mathbf{w}, b))}
\end{aligned}$$

This loss is an instance of the cross-entropy

$$H(p, q) = \mathbb{E}_p[-\log q]$$

for $p = Y|\mathbf{x}_i$ and $q = \hat{Y}|\mathbf{x}_i$.

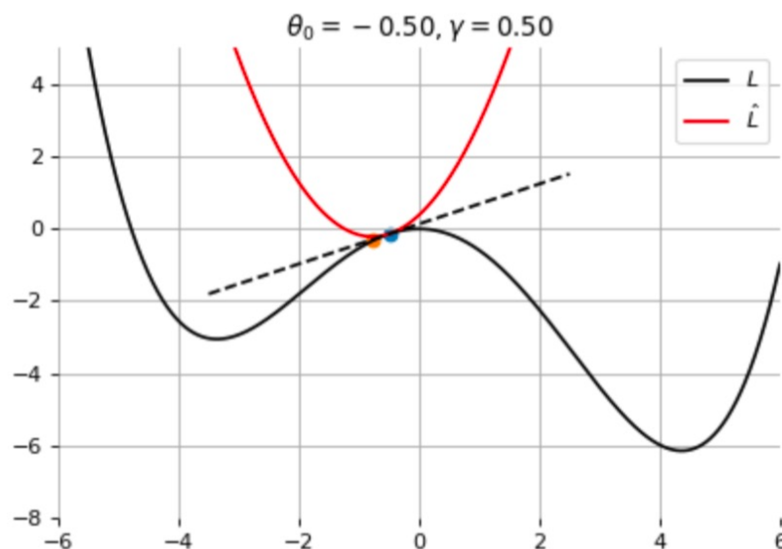
Gradient Descent

Let $\mathcal{L}(\theta)$ denote a loss function defined over model parameters θ (e.g., $\mathbf{w}$ and b).

To minimize $\mathcal{L}(\theta)$, gradient descent uses local linear information to iteratively move towards a (local) minimum.

For $\theta_0 \in \mathbb{R}^d$, a first-order approximation around θ_0 can be defined as

$$\hat{\mathcal{L}}(\theta_0 + \epsilon) = \mathcal{L}(\theta_0) + \epsilon^T \nabla_\theta \mathcal{L}(\theta_0) + \frac{1}{2\gamma}||\epsilon||^2.$$



Gradient Descent

A minimizer of the approximation $\hat{\mathcal{L}}(\theta_0 + \epsilon)$ is given for

$$\nabla_\epsilon \hat{\mathcal{L}}(\theta_0 + \epsilon) = 0$$
$$= \nabla_\theta \mathcal{L}(\theta_0) + \frac{1}{\gamma}\epsilon,$$

which results in the best improvement for the step $\epsilon = -\gamma \nabla_\theta \mathcal{L}(\theta_0)$.

Therefore, model parameters can be updated iteratively using the update rule

$$\theta_{t+1} = \theta_t - \gamma \nabla_\theta \mathcal{L}(\theta_t),$$

where

- θ_0 are the initial parameters of the model;
- γ is the learning rate;
- both are critical for the convergence of the update rule.

Stochastic Gradient Descent

In the empirical risk minimization setup, $\mathcal{L}(\theta)$ and its gradient decompose as

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{\mathbf{x}_i, y_i \in \mathbf{d}} \ell(y_i, f(\mathbf{x}_i; \theta))$$

$$\nabla \mathcal{L}(\theta) = \frac{1}{N} \sum_{\mathbf{x}_i, y_i \in \mathbf{d}} \nabla \ell(y_i, f(\mathbf{x}_i; \theta)).$$

Therefore, in batch gradient descent the complexity of an update grows linearly with the size N of the dataset.

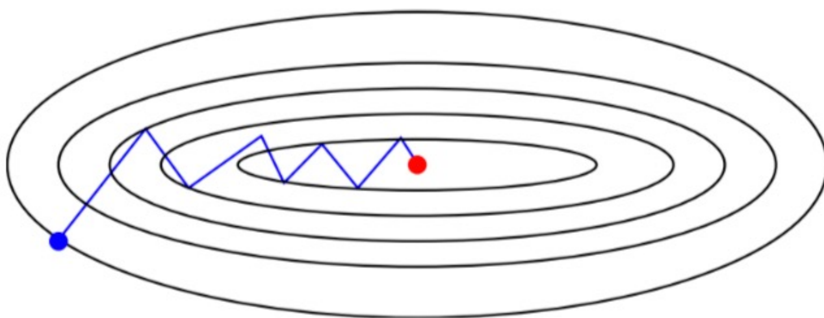
More importantly, since the empirical risk is already an approximation of the expected risk, it should not be necessary to carry out the minimization with great accuracy.

Stochastic Gradient Descent

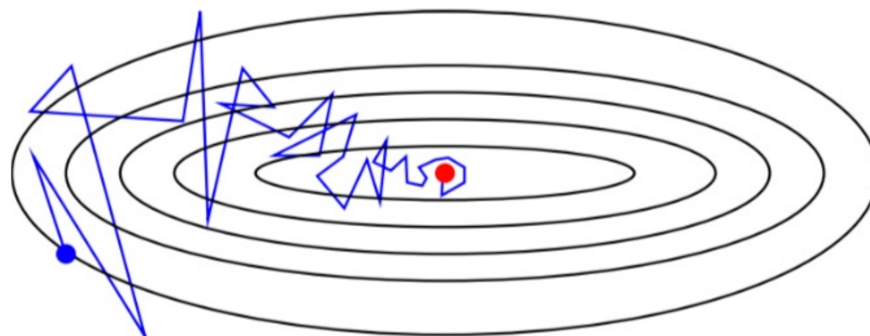
Instead, stochastic gradient descent uses as update rule:

$$\theta_{t+1} = \theta_t - \gamma \nabla \ell(y_{i(t+1)}, f(\mathbf{x}_{i(t+1)}; \theta_t))$$

- Iteration complexity is independent of N.
- The stochastic process $\{\theta_t | t = 1, \ldots\}$ depends on the examples $i(t)$ picked randomly at each iteration.



Batch gradient descent



Stochastic gradient descent

Stochastic Gradient Descent

Why is stochastic gradient descent still a good idea?

- Informally, averaging the update

$$\theta_{t+1} = \theta_t - \gamma \nabla \ell(y_{i(t+1)}, f(\mathbf{x}_{i(t+1)}; \theta_t))$$

  over all choices $i(t+1)$ restores batch gradient descent.

- Formally, if the gradient estimate is unbiased, e.g., if

$$\mathbb{E}_{i(t+1)}[\nabla \ell(y_{i(t+1)}, f(\mathbf{x}_{i(t+1)}; \theta_t))] = \frac{1}{N} \sum_{\mathbf{x}_i, y_i \in \mathbf{d}} \nabla \ell(y_i, f(\mathbf{x}_i; \theta_t))$$
$$= \nabla \mathcal{L}(\theta_t)$$

  then the formal convergence of SGD can be proved, under appropriate assumptions (see references).

- Interestingly, if training examples $\mathbf{x}_i, y_i \sim P_{X,Y}$ are received and used in an online fashion, then SGD directly minimizes the expected risk.

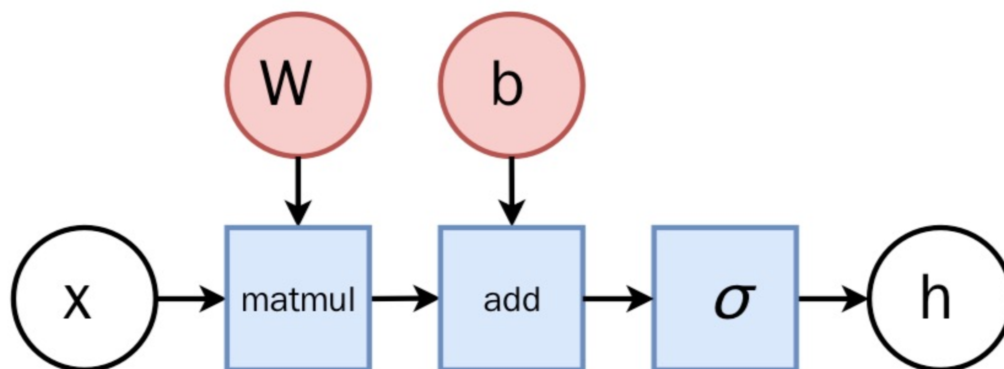
Layers

So far we considered the logistic unit $h = \sigma\left(\mathbf{w}^T\mathbf{x} + b\right)$, where $h \in \mathbb{R}$, $\mathbf{x} \in \mathbb{R}^p$, $\mathbf{w} \in \mathbb{R}^p$ and $b \in \mathbb{R}$.

These units can be composed in parallel to form a layer with q outputs:

$$\mathbf{h} = \sigma(\mathbf{W}^T\mathbf{x} + \mathbf{b})$$

where $\mathbf{h} \in \mathbb{R}^q$, $\mathbf{x} \in \mathbb{R}^p$, $\mathbf{W} \in \mathbb{R}^{p\times q}$, $b \in \mathbb{R}^q$ and where $\sigma(\cdot)$ is upgraded to the element-wise sigmoid function.



Multi-layer Perceptron/Neural Nets (MLPs)

Similarly, layers can be composed in series, such that:

$$\begin{aligned}
\mathbf{h}_0 &= \mathbf{x} \\
\mathbf{h}_1 &= \sigma(\mathbf{W}_1^T \mathbf{h}_0 + \mathbf{b}_1) \\
&\dots \\
\mathbf{h}_L &= \sigma(\mathbf{W}_L^T \mathbf{h}_{L-1} + \mathbf{b}_L) \\
f(\mathbf{x}; \theta) = \hat{y} &= \mathbf{h}_L
\end{aligned}$$

where θ denotes the model parameters $\{\mathbf{W}_k, \mathbf{b}_k, ... | k = 1, ..., L\}$.

This model is the multi-layer perceptron, also known as the fully connected feedforward network.

What if we don't have the non-linear functions?

Activation Functions

Also called “link functions”

$$\text{sign}(a)$$

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

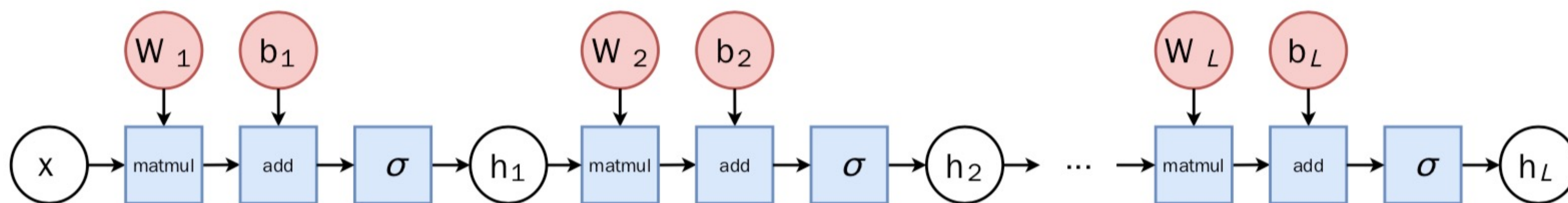
$$\tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}$$

$$\text{ReLU}(a) = \max(a, 0)$$

$$\text{SoftPlus}(a) = \log(1 + e^{a})$$

$$\text{ELU}(a) = \left\{ \begin{array}{ll} a, & \text{for } a \geq 0 \\ \alpha(e^{a} - 1), & \text{for } a < 0 \end{array} \right\}$$

Computational Graph



Classification

- For binary classification, the width q of the last layer L is set to 1, which results in a single output $h_L \in [0, 1]$ that models the probability $P(Y = 1|\mathbf{x})$.
- For multi-class classification, the sigmoid action σ in the last layer can be generalized to produce a (normalized) vector $\mathbf{h}_L \in [0, 1]^C$ of probability estimates $P(Y = i|\mathbf{x})$.

This activation is the Softmax function, where its i-th output is defined as

$$\text{Softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{C} \exp(z_j)},$$

for $i = 1, ..., C$.

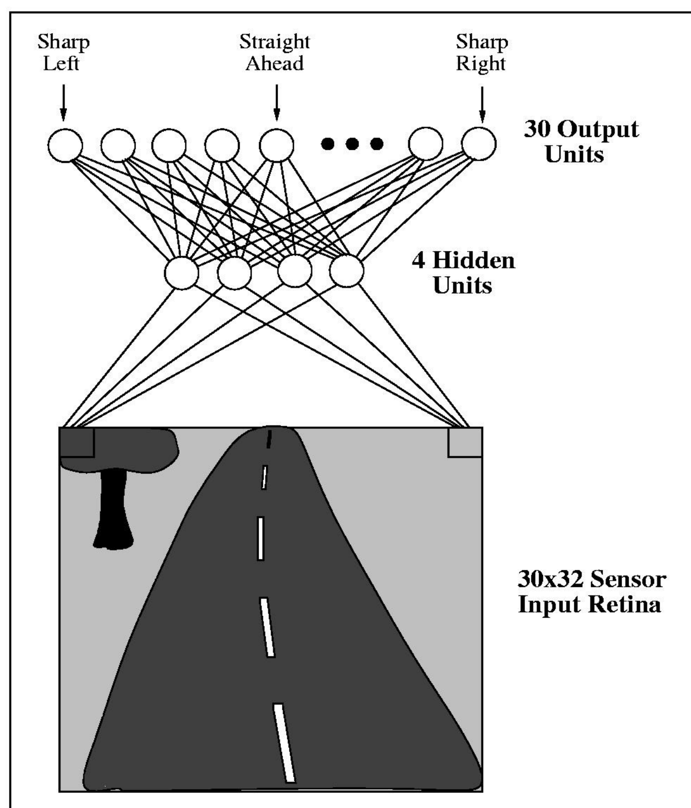
What is the loss function in this multi-class setting?

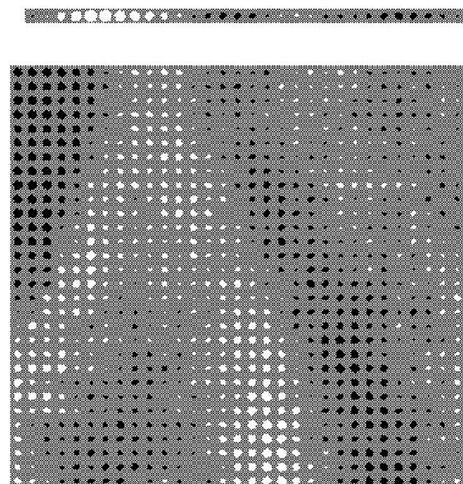
Regression

The last activation σ can be skipped to produce unbounded output values $h_L \in \mathbb{R}$.

Self-driving cars

Self-driving cars





ALVINN: Autonomous Land Vehicle In a Neural Network (1989)

Automatic Differentiation

To minimize $\mathcal{L}(\theta)$ with stochastic gradient descent, we need the gradient $\nabla_\theta \ell(\theta_t)$.

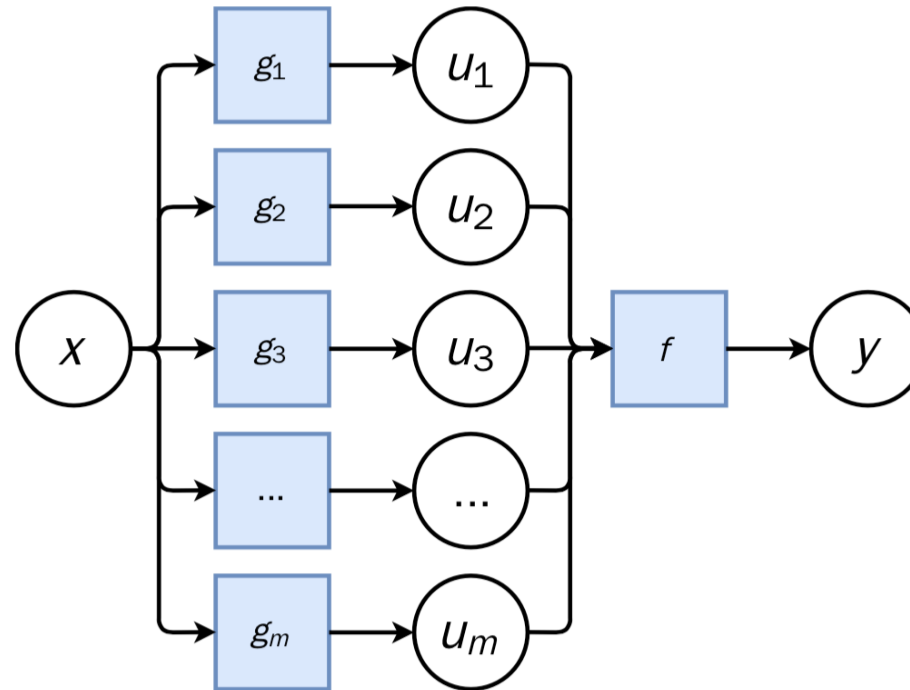
Therefore, we require the evaluation of the (total) derivatives

$$\frac{\mathrm{d}\ell}{\mathrm{d}\mathbf{W}_k}, \frac{\mathrm{d}\ell}{\mathrm{d}\mathbf{b}_k}$$

of the loss ℓ with respect to all model parameters $\mathbf{W}_k, \mathbf{b}_k$, for $k = 1, ..., L$.

These derivatives can be evaluated automatically from the computational graph of ℓ using automatic differentiation.

Chain Rule



Let us consider a 1-dimensional output composition $f \circ g$, such that

$$y = f(\mathbf{u})$$
$$\mathbf{u} = g(x) = (g_1(x), ..., g_m(x)).$$

Chain Rule

The chain rule states that $(f \circ g)' = (f' \circ g)g'$.

For the total derivative, the chain rule generalizes to

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \sum_{k=1}^{m} \frac{\partial y}{\partial u_k} \underbrace{\frac{\mathrm{d}u_k}{\mathrm{d}x}}_{\text{recursive case}}$$

Reserve Automatic Differentiation

- Since a neural network is a composition of differential functions, the total derivatives of the loss can be evaluated backward, by applying the chain rule recursively over its computational graph.
- The implementation of this procedure is called reserve automatic differentiation.

Example

Let us consider a simplified 2-layer MLP and the following loss function:

$$f(\mathbf{x}; \mathbf{W}_1, \mathbf{W}_2) = \sigma\left(\mathbf{W}_2^T \sigma\left(\mathbf{W}_1^T \mathbf{x}\right)\right)$$
$$\ell(y, \hat{y}; \mathbf{W}_1, \mathbf{W}_2) = \text{cross_ent}(y, \hat{y}) + \lambda\left(||\mathbf{W}_1||_2 + ||\mathbf{W}_2||_2\right)$$

for $\mathbf{x} \in \mathbb{R}^p, y \in \mathbb{R}, \mathbf{W}_1 \in \mathbb{R}^{p \times q}$ and $\mathbf{W}_2 \in \mathbb{R}^q$.

Example

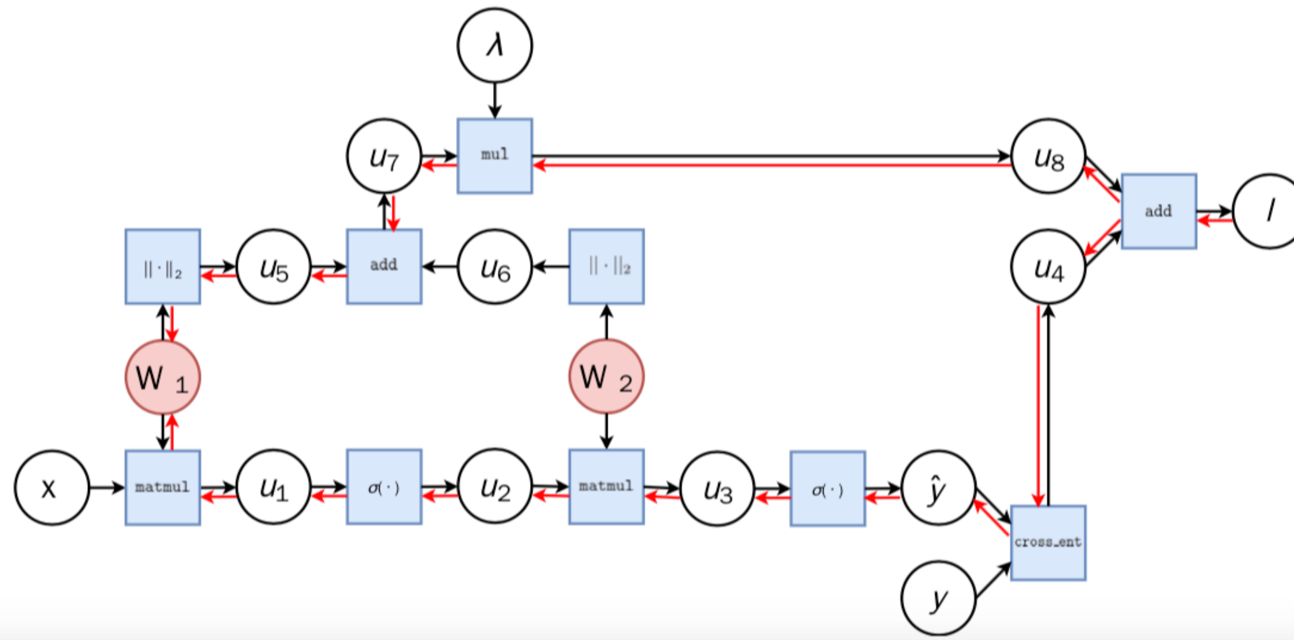
In the forward pass, intermediate values are all computed from inputs to outputs, which results in the annotated computational graph below:

Example

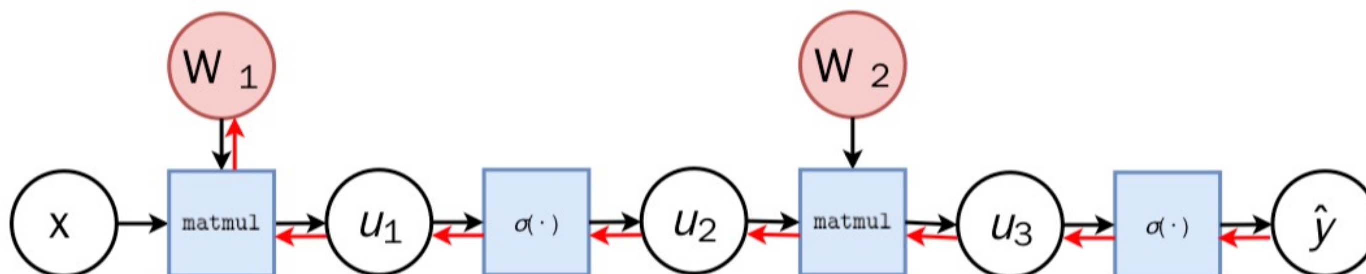
The total derivative can be computed through a backward pass, by walking through all paths from outputs to parameters in the computational graph and accumulating the terms. For example, for $\frac{d\ell}{d\mathbf{W}_1}$ we have:

$$\frac{d\ell}{d\mathbf{W}_1} = \frac{\partial \ell}{\partial u_8}\frac{du_8}{d\mathbf{W}_1} + \frac{\partial \ell}{\partial u_4}\frac{du_4}{d\mathbf{W}_1}$$

$$\frac{du_8}{d\mathbf{W}_1} = \ldots$$



Example



Let us zoom in on the computation of the network output $\hat{y}$ and of its derivative with respect to $\mathbf{W}_1$.

- Forward pass: values u_1, u_2, u_3 and $\hat{y}$ are computed by traversing the graph from inputs to outputs given $\mathbf{x}$, $\mathbf{W}_1$ and $\mathbf{W}_2$.
- Backward pass: by the chain rule we have

$$\frac{\mathrm{d}\hat{y}}{\mathrm{d}\mathbf{W}_1} = \frac{\partial \hat{y}}{\partial u_3} \frac{\partial u_3}{\partial u_2} \frac{\partial u_2}{\partial u_1} \frac{\partial u_1}{\partial \mathbf{W}_1}$$
$$= \frac{\partial \sigma(u_3)}{\partial u_3} \frac{\partial \mathbf{W}_2^T u_2}{\partial u_2} \frac{\partial \sigma(u_1)}{\partial u_1} \frac{\partial \mathbf{W}_1^T u_1}{\partial \mathbf{W}_1}$$

Note how evaluating the partial derivatives requires the intermediate values computed forward.

Back-propagation

- This algorithm is also known as **back-propagation**
- An equivalent procedure can be defined to evaluate the derivatives in forward mode, from inputs to outputs.
- Since differentiation is a linear operator, automatic differentiation can be implemented efficiently in terms of tensor operations.

Convolutions

Input

1	4	-1	0	2	-2	1	3	3	1

W

1	2	0	-1

w

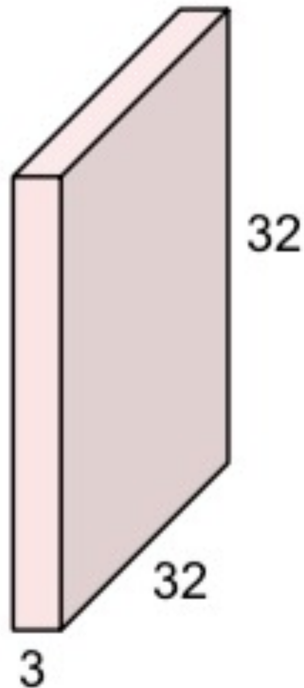
Output

9	

$W - w + 1$

Convolutions

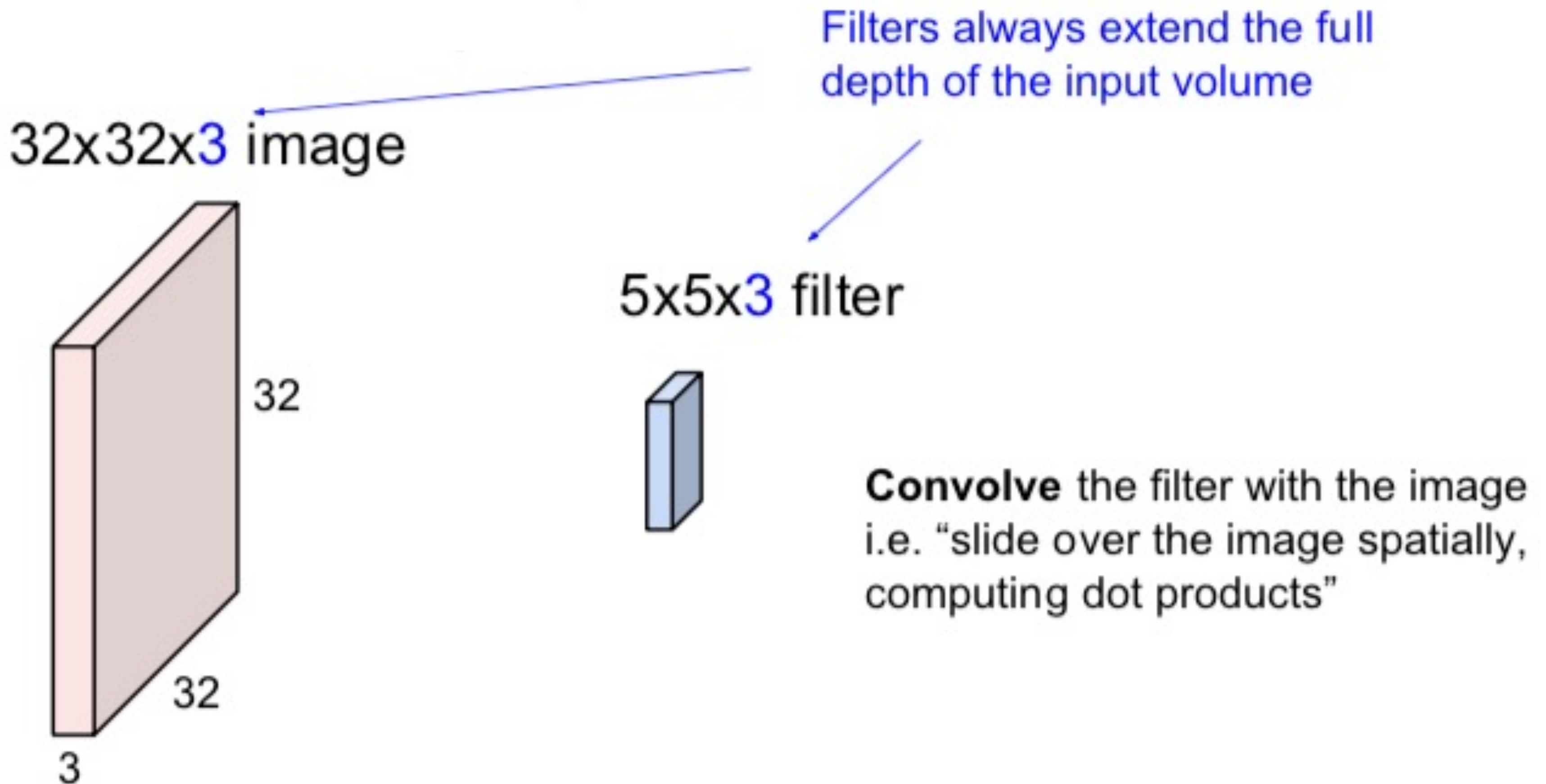
32x32x3 image



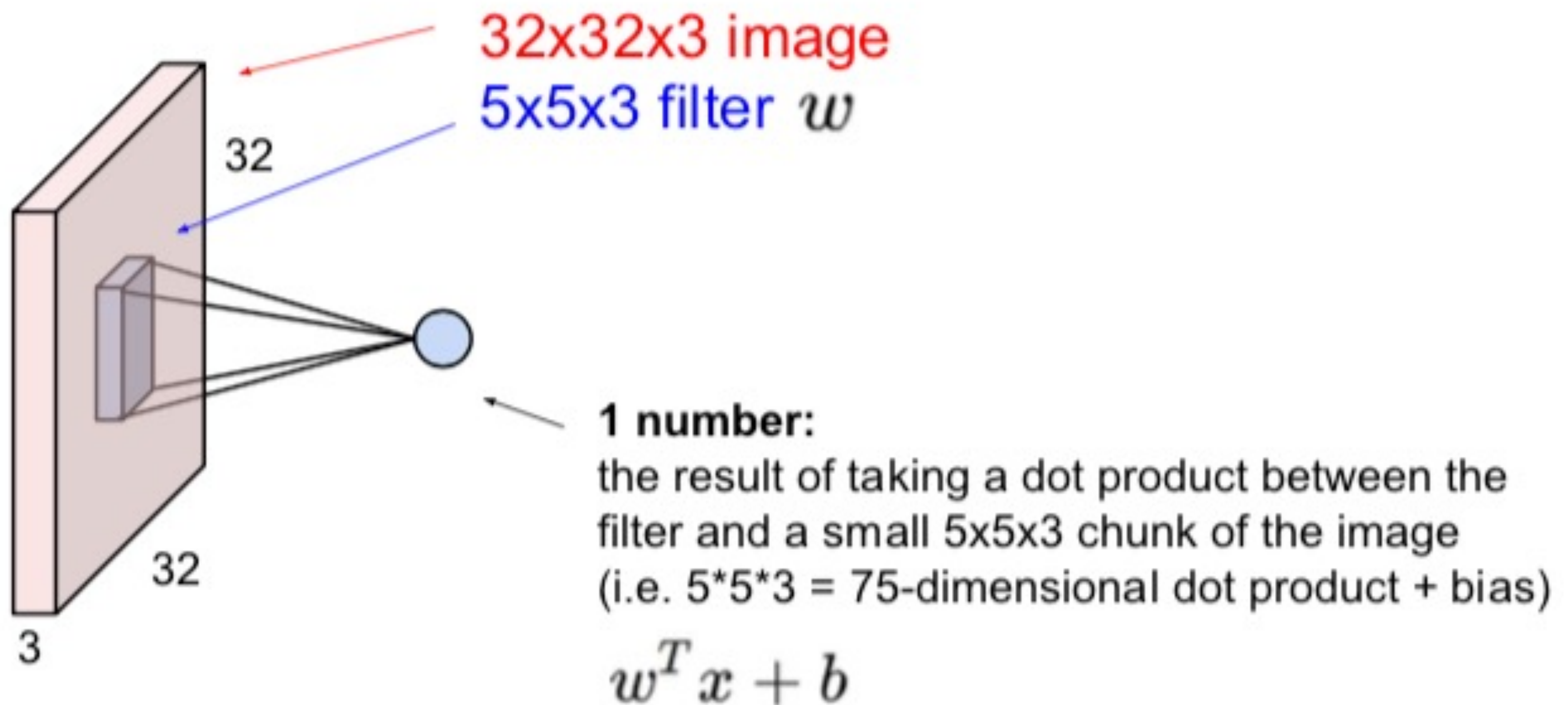
5x5x3 filter

Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products"

Convolutions



Convolutions



Convolutions

activation map

32x32x3 image

5x5x3 filter

32

32

3

convolve (slide) over all spatial locations

28

28

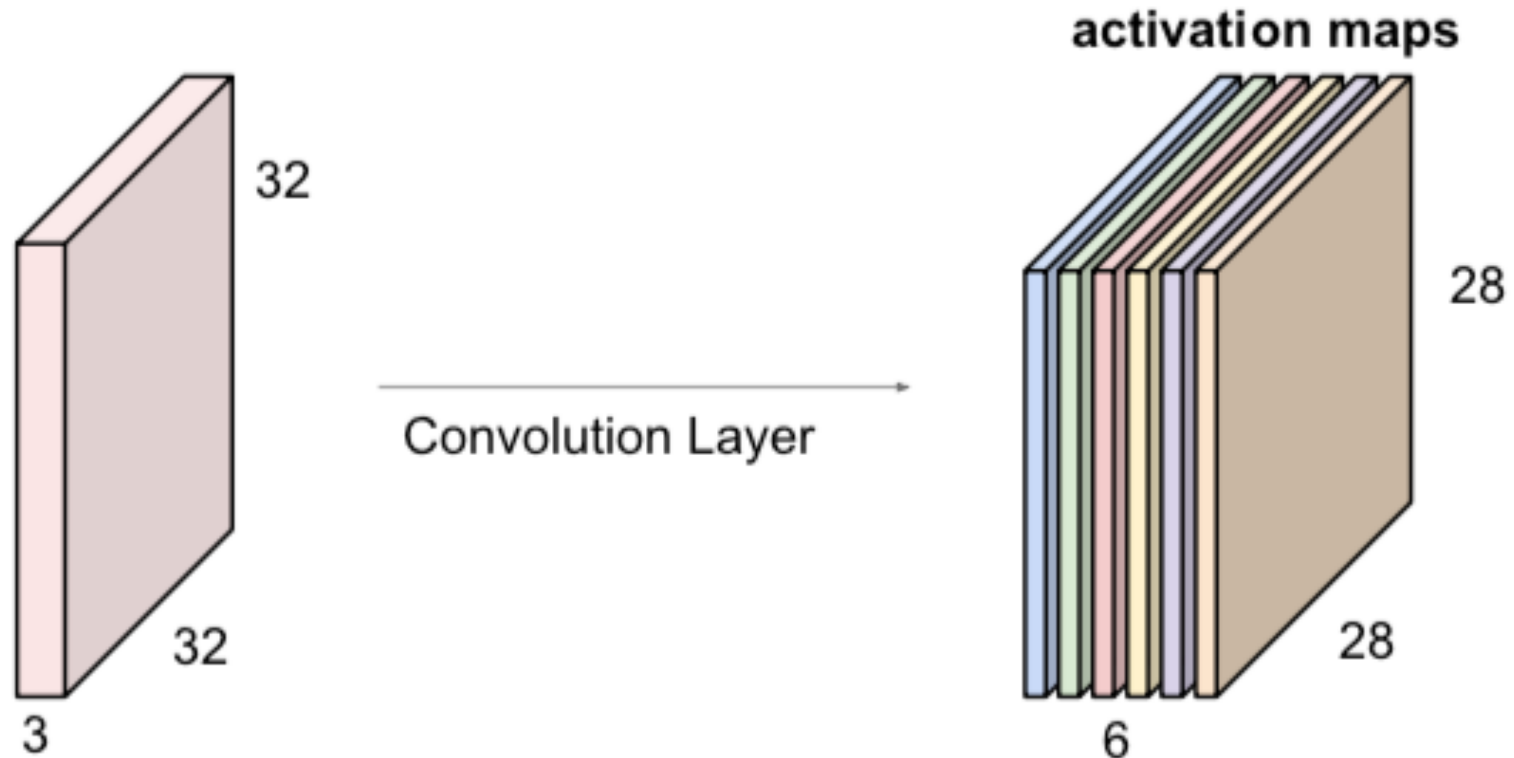
1

Convolutions

let't add one more filter, the green one

Convolutions

with 6 separate filters, we'll get 6 separate activation maps



Convolutional Neural Networks (CNN)

LeNet-5 (LeCun et al, 1998)

- First convolutional network to use backpropagation.
- Applied to character recognition.

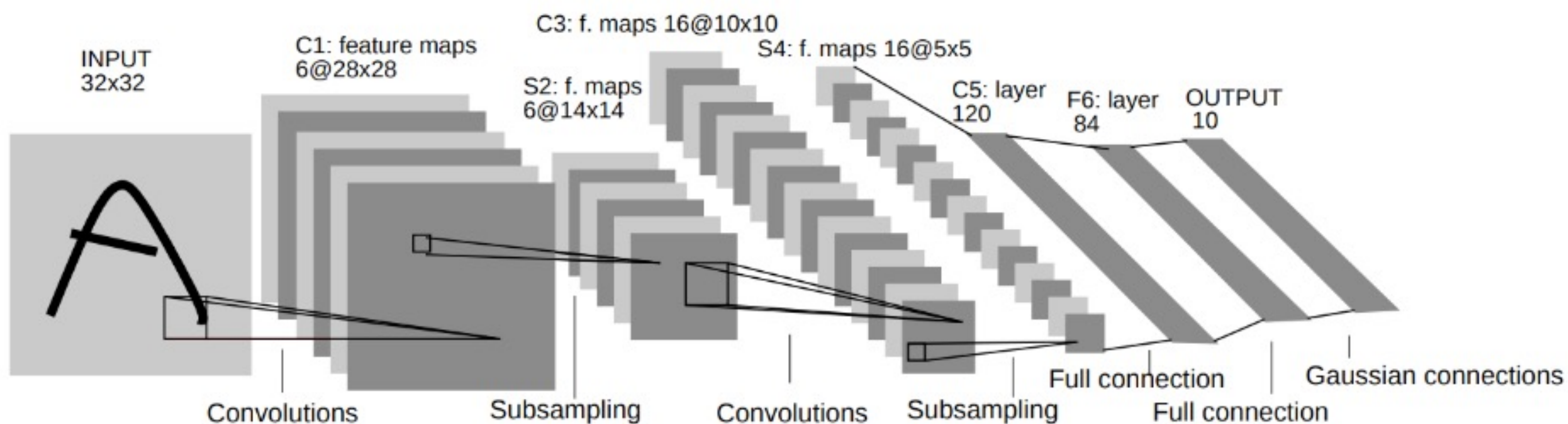


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

CNN for Sentence Classification

Temporal convolution/1D convolution

wait
for
the
video
and
do
n't
rent
it

n x k representation of sentence with static and non-static channels

Convolutional layer with multiple filter widths and feature maps

Max-over-time pooling

Fully connected layer with dropout and softmax output

Convolutional Neural Networks for Sentence Classification, Kim et al., 2014 (EMNLP)

CNN for Sentence Classification

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	–	–	–	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–	–	–
RNTN (Socher et al., 2013)	–	45.7	85.4	–	–	–	–
DCNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0	–	–
Paragraph-Vec (Le and Mikolov, 2014)	–	**48.7**	87.8	–	–	–	–
CCAE (Hermann and Blunsom, 2013)	77.8	–	–	–	–	–	87.2
Sent-Parser (Dong et al., 2014)	79.5	–	–	–	–	–	86.3
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	–	–	**93.6**	–	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	–	–	93.4	–	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	–	–	**93.6**	–	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	–	–	–	–	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	–	–	–	–	–	82.7	–
SVM_S (Silva et al., 2011)	–	–	–	–	**95.0**	–	–

Convolutional Neural Networks for Sentence Classification, Kim et al., 2014 (EMNLP)

CNN for Relation Extraction

Relation Extraction: Perspective from Convolutional Neural Networks, Nguyen and Grishman, 2015