Language Models and Recurrent Neural Networks

COSC 7336: Advanced Natural Language Processing
Fall 2017

UNIVERSITY of HOUSTON

Some content on these slides was borrowed from J&M

UNIVERSIDAD NACIONAL DE COLOMBIA

Announcements

- ★ Next Week no class (Mid-Semester Bash)
- ★ Assignment 2 is out
- ★ Mid term exam Oct. 27th
- ★ Paper presentation sign up coming up
- ★ Reminder to submit report, code and notebook for Assignment 1 today!

Today's lecture

★ Short intro to language models
★ Recurrent Neural Networks
★ Demo: RNN for text

Language Models

- ★ They assign a probability to a sequence of words:
  - Machine Translation:
    - P(**high** winds tonite) > P(**large** winds tonite)
  - Spell Correction:
    - The office is about fifteen **minuets** from my house
    - P(about fifteen **minutes** from) > P(about fifteen **minuets** from)
  - Speech Recognition
    - P(I saw a van) >> P(eyes awe of an)
  - Summarization, question-answering, OCR correction and many more!

More Formally

- Given a sequence of words predict the next one:
  - $P(w_5|w_1,w_2,w_3,w_4)$
- Predict the likelihood of a sequence of words:
  - $P(W) = P(w_1,w_2,w_3,w_4,w_5\ldots w_n)$
- How do we compute these?

Chain Rule

- Recall the definition of conditional probabilities:
  - **$p(B|A) = P(A,B)/P(A)$** Rewriting: **$P(A,B) = P(A)P(B|A)$**
- More variables:
  - $P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$
- The Chain Rule in General
  - $P(x_1,x_2,x_3,\ldots,x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_1,\ldots,x_{n-1})$

Back to sequence of words

P("I am the fire that burns against the cold") = P(I) x P(am|I) x P(the|I am) x P(fire|I am the) x P(that| I am the fire) x P(burns| I am the fire that) x P(against| I am the fire that burns) x P(the| I am the fire that burns against) x P(cold| I am the fire that burns against the)

★ How do we estimate these probabilities?

count(I am the fire that burns against the cold)

count(I am the fire that burns against the)

We shorten the context (history)

Markov Assumption:

P(cold | I am the fire that burns against the) ≈ P (cold | burns against the)

This is: $P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$

When N = 1, this is a unigram language model:

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

When k =2, this is a bigram language model:

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$

Are all our problems solved?

Zeros

Training set:

… denied the allegations
… denied the reports
… denied the claims
… denied the request

P("offer" | denied the) = 0

Test set:

… denied the offer
… denied the loan

Laplace Smoothing

- Add one to all counts
- Unigram counts:

$$P(w_i) = \frac{c_i}{N}$$

- Laplace counts:

$$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

- Disadvantages:
  - Drastic change in probability mass
- But for some tasks Laplace smoothing is a reasonable choice

Leveraging Hierarchy of N-grams: Backoff

- Adding one to all counts is too drastic
- What about relying on shorter contexts?
  - Let's say we're tying to compute P(against | that burns), but count(that burns against) = 0
  - We backoff to a shorter context: P(against | that burns) ≈ P(against | burns)
- We backoff to a lower n-gram
- Katz Backoff (discounted backoff)

$$P_{\mathrm{BO}}(w_n|w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n|w_{n-N+1}^{n-1}), & \text{if } C(w_{n-N+1}^{n}) > 0 \\ \alpha(w_{n-N+1}^{n-1}) P_{\mathrm{BO}}(w_n|w_{n-N+2}^{n-1}), & \text{otherwise.} \end{cases}$$

Leveraging Hierarchy of N-grams: Interpolation

★ Better idea: why not always rely on lower order N-grams?

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\ &+\lambda_2 P(w_n|w_{n-1}) \\ &+\lambda_3 P(w_n)\end{aligned}$$

Subject to: $\sum_i \lambda_i = 1$

★ Even better, condition on context:

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1(w_{n-2}^{n-1}) P(w_n|w_{n-2}w_{n-1}) \\ &+\lambda_2(w_{n-2}^{n-1}) P(w_n|w_{n-1}) \\ &+\lambda_3(w_{n-2}^{n-1}) P(w_n)\end{aligned}$$

Absolute Discounting

Bigram count in training	Bigram count in held out set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

For each bigram count in training data, what's the count in held out set?

Approx. a 0.75 difference!

Absolute Discounting

$$P_{\text{AbsoluteDiscounting}}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - d}{\sum_v C(w_{i-1}v)} + \lambda(w_{i-1})P(w_i)$$

How much do we want to trust unigrams?

- Instead of P(w): “How likely is w”
- $P_{continuation}(w)$: “How likely is w to appear as a novel continuation?
- For each word, count the number of bigram types it completes

$$P_{\text{CONTINUATION}}(w) = \frac{|\{v : C(vw) > 0\}|}{|\{(u', w') : C(u'w') > 0\}|}$$

Interpolated Kneser-Ney

★ Intuition: Use estimate of $P_{CONTINUATION}(w_i)$

$$P_{\mathrm{KN}}(w_i|w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^{i}) - d, 0)}{\sum_v c_{KN}(w_{i-n+1}^{i-1} v)} + \lambda(w_{i-n+1}^{i-1}) P_{KN}(w_i|w_{i-n+2}^{i-1})$$

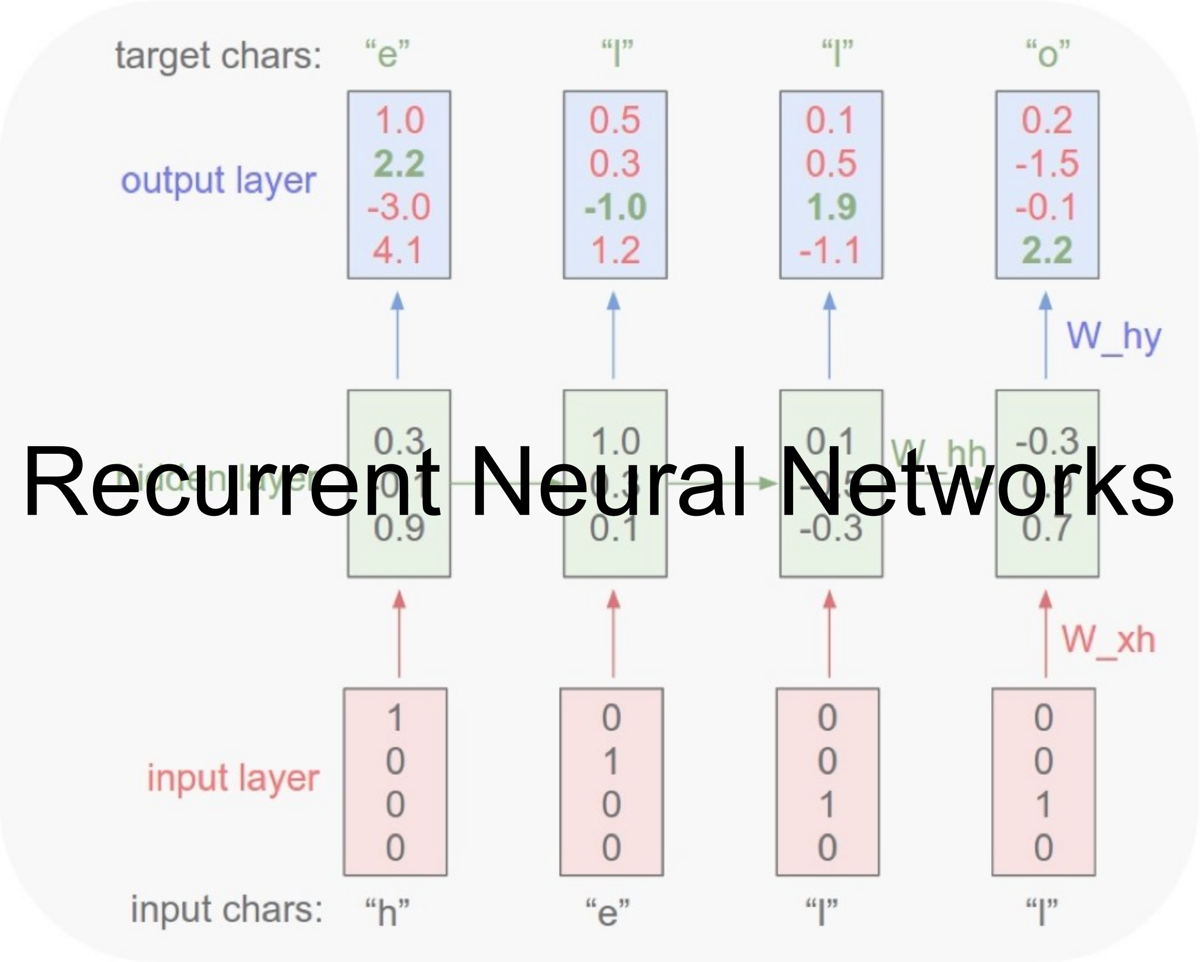
Evaluating Language Models

- ★ Ideal: Evaluate on end task (extrinsic)
- ★ Intrinsic evaluation: use perplexity:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$$

- ★ Perplexity is inversely proportional to the probability of W

Recurrent Neural Networks

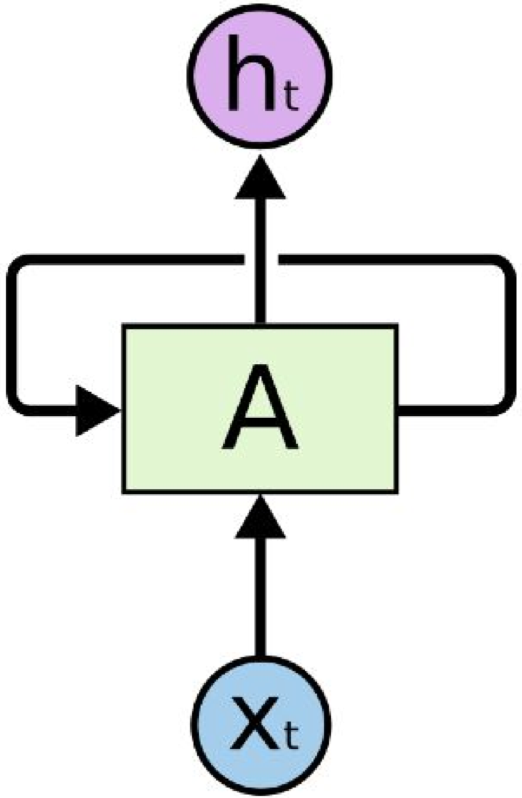


UNIVERSITY of HOUSTON

UNIVERSIDAD NACIONAL DE COLOMBIA

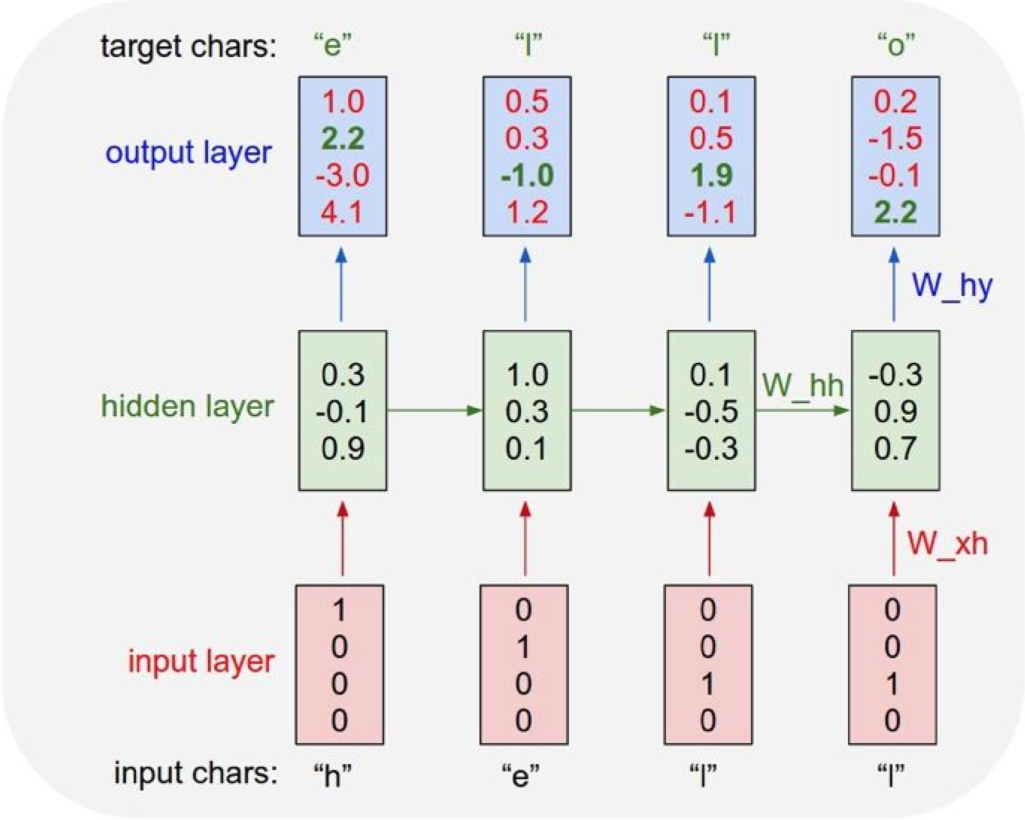
Recurrent NNs

- ★ Neural networks with memory
- ★ Feed-forward NN: output exclusively depends on the current input
- ★ Recurrent NN: output depends on current and previous states
- ★ This is accomplished through lateral/backward connections which carry information while processing a sequence of inputs



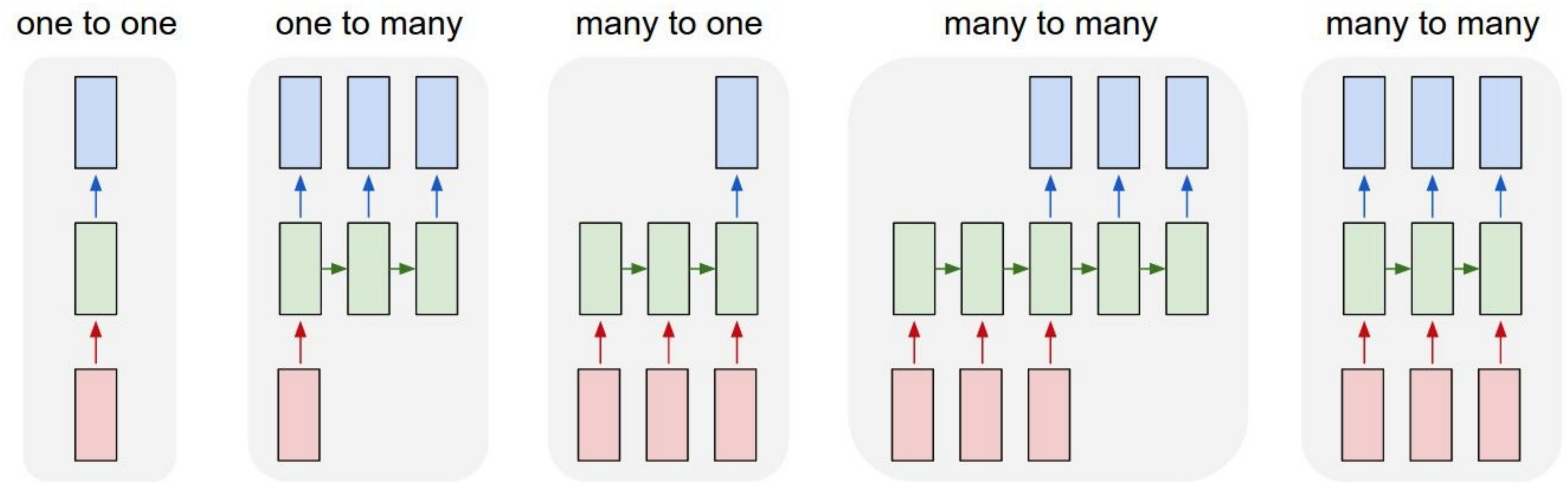
(source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Character-level language model



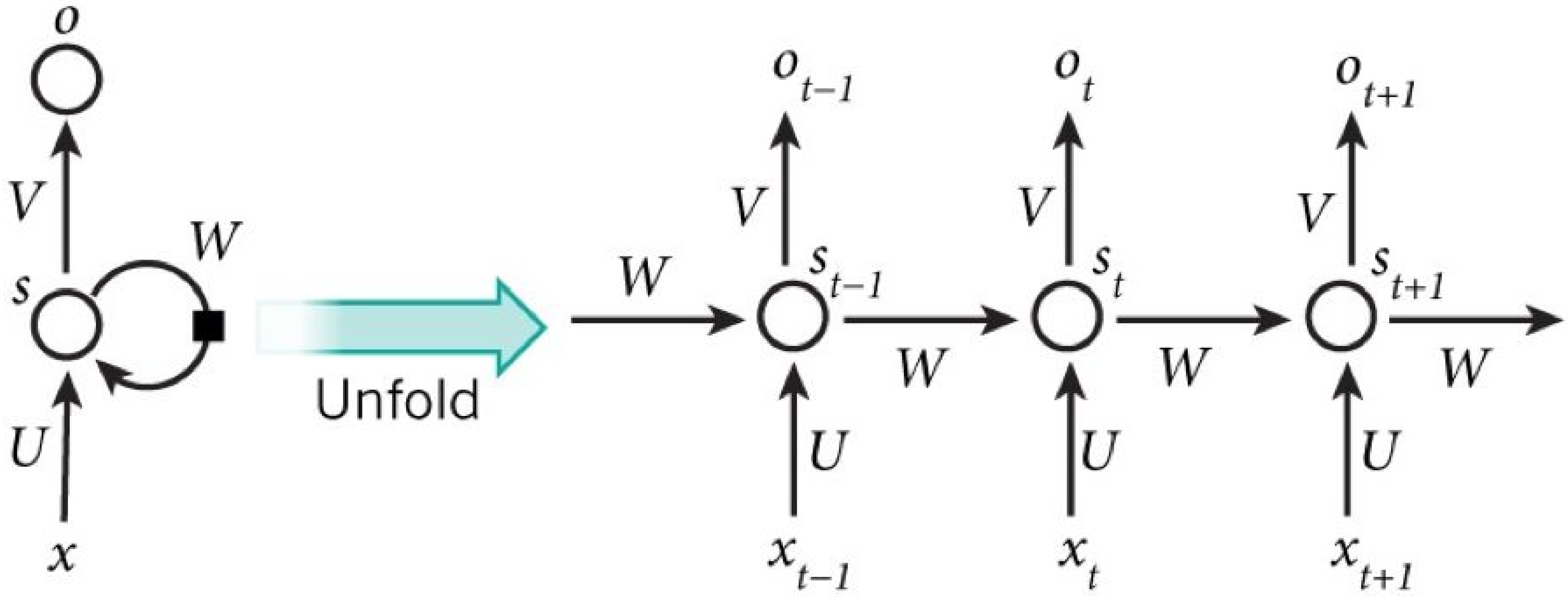
(source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

Sequence learning alternatives



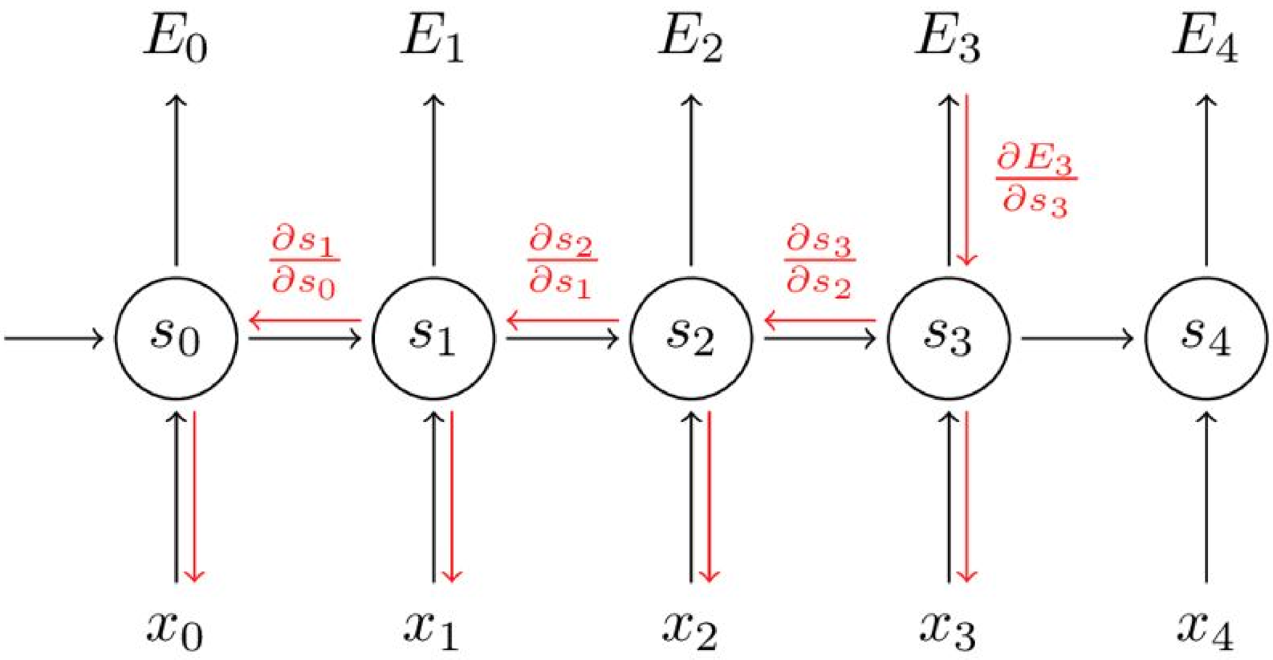
(source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

Network unrolling



(source: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/)

Backpropagation through time (BPTT)



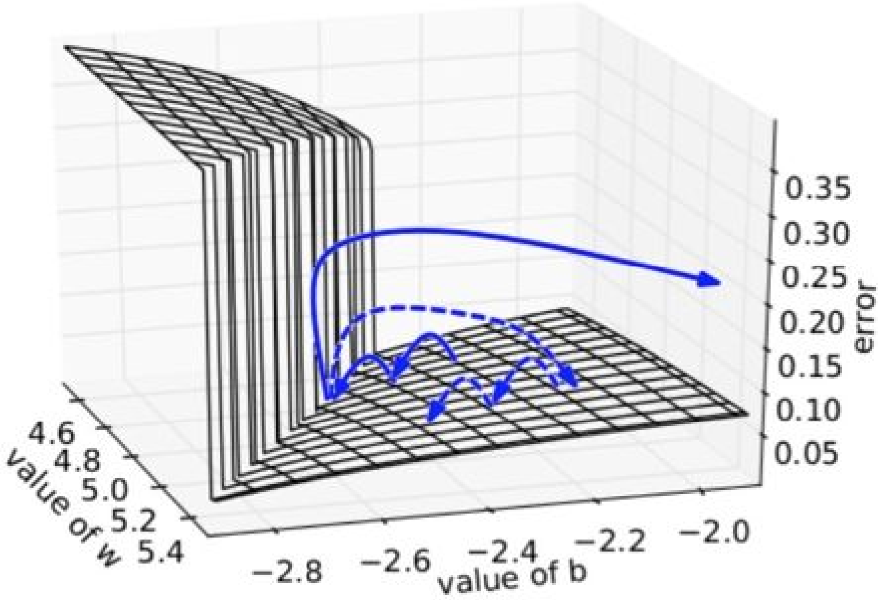
(source: http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/)

BPTT is hard

★ The vanishing and the exploding gradient problem

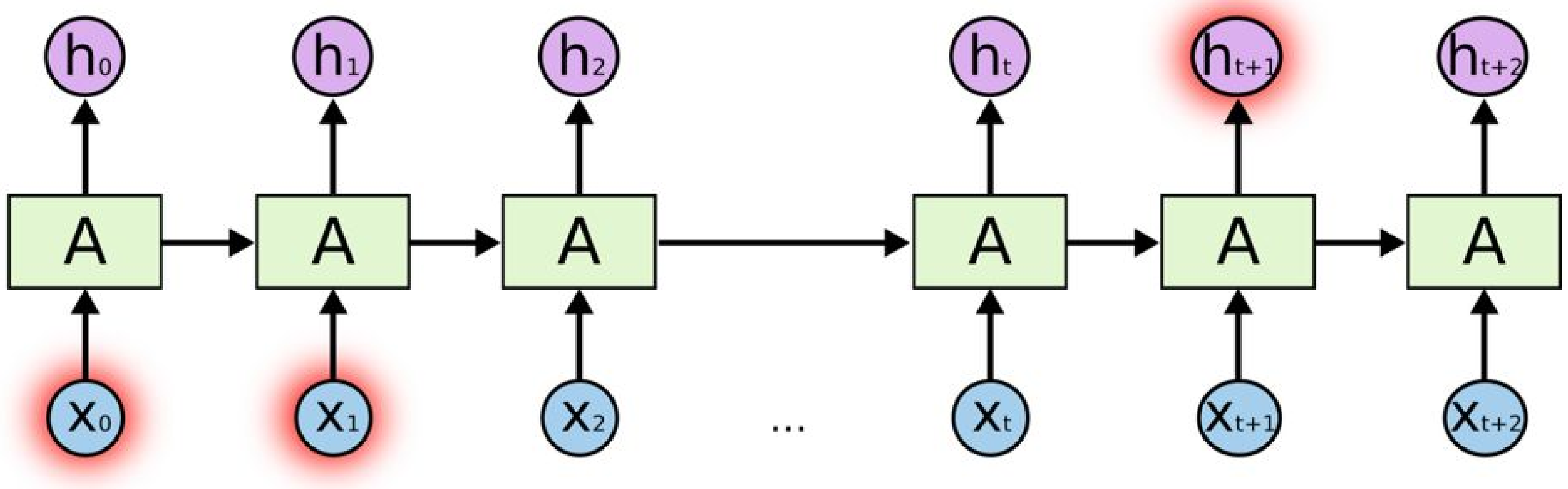
★ Gradients could vanish (or explode) when propagated several steps back

★ This makes difficult to learn long-term dependencies.



Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training Recurrent Neural Networks. Proc. of ICML, abs/1211.5063.

Long term dependencies

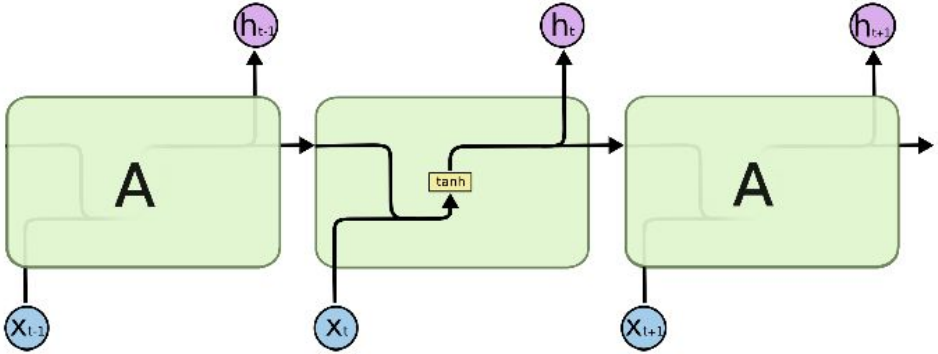


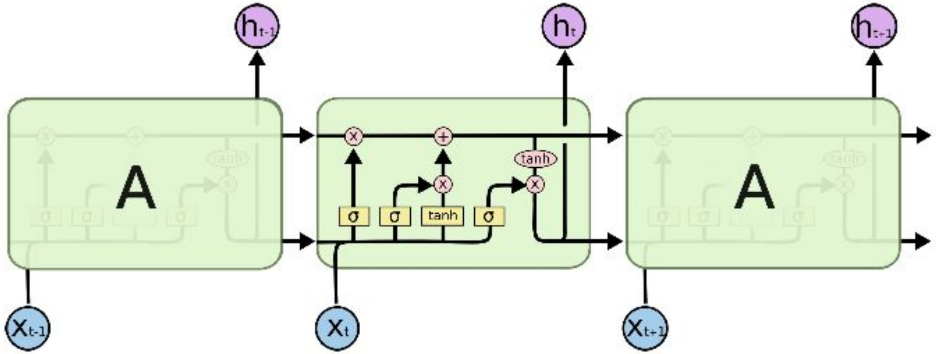
(source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Long short-term memory (LSTM)

★ LSTM networks solve the problem of long-term dependency problem.
★ They use gates that allow to keep memory through long sequences and be updated only when required.
★ Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9, no. 8 (1997): 1735-1780.

Conventional RNN vs LSTM





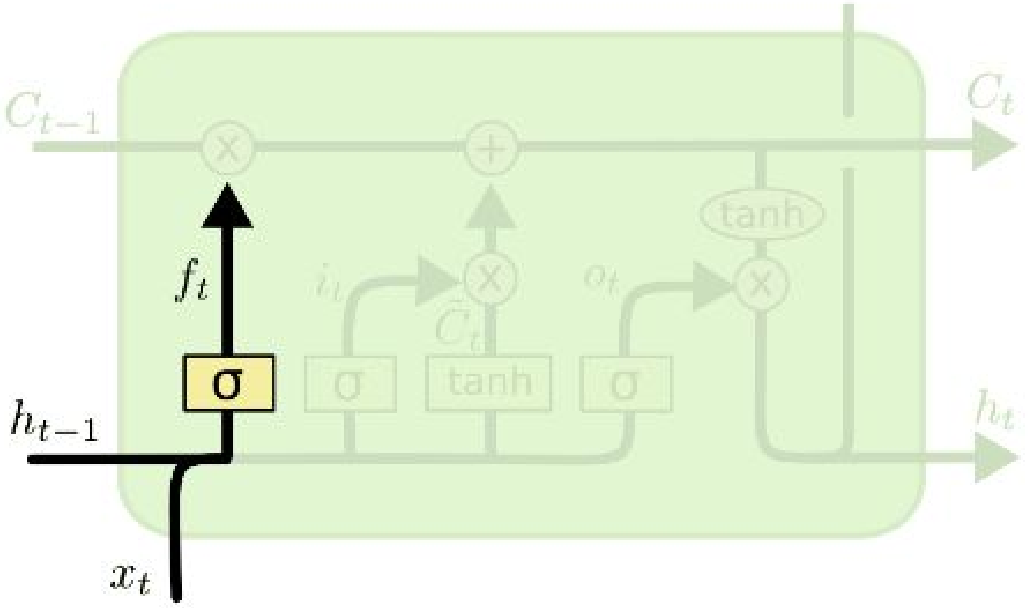
(source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Forget Gate

★ Controls the flow of the previous internal state Ct-1

★ f_t=1 ⇒ keep previous state

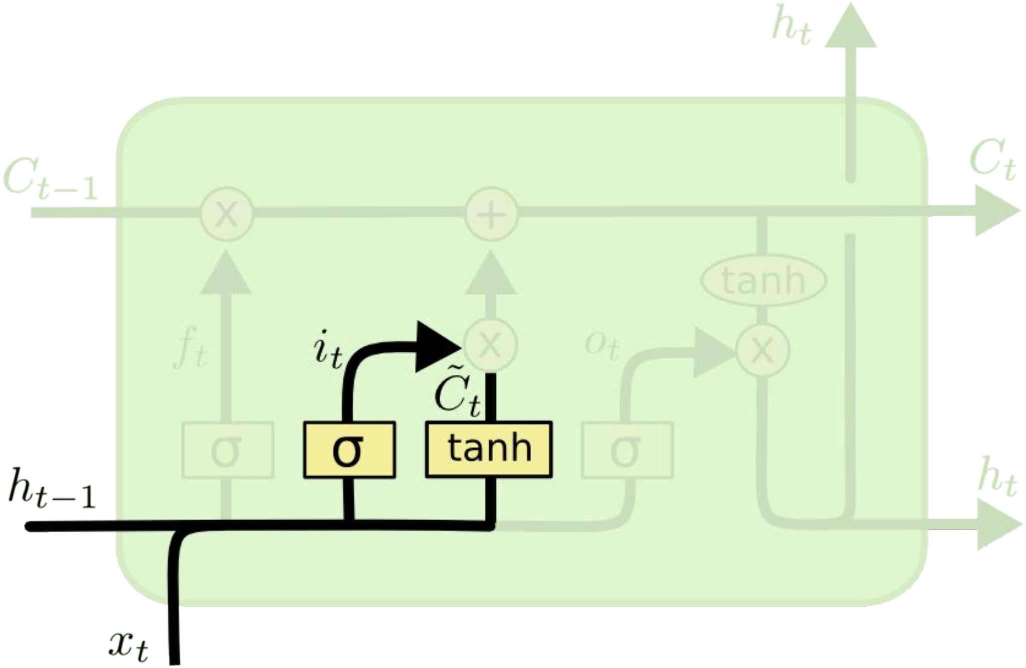
★ f_t=0 ⇒ forget previous state



(source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

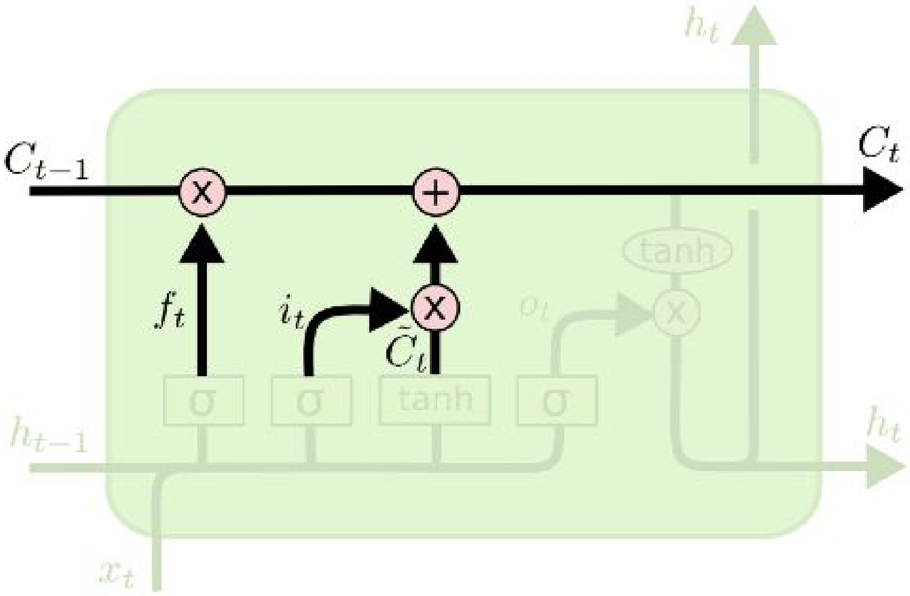
Input Gate

- ★ Controls the flow of the input state xt
- ★ i_t=1 ⇒ take input into account
- ★ i_t=0 ⇒ ignore input



(source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Current state calculation

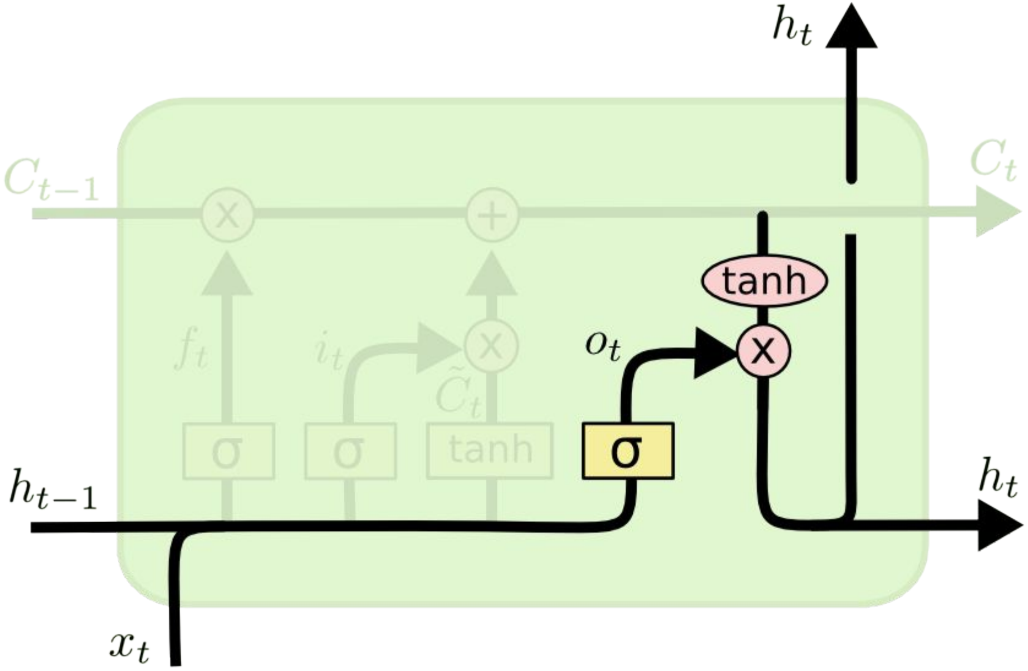


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

(source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

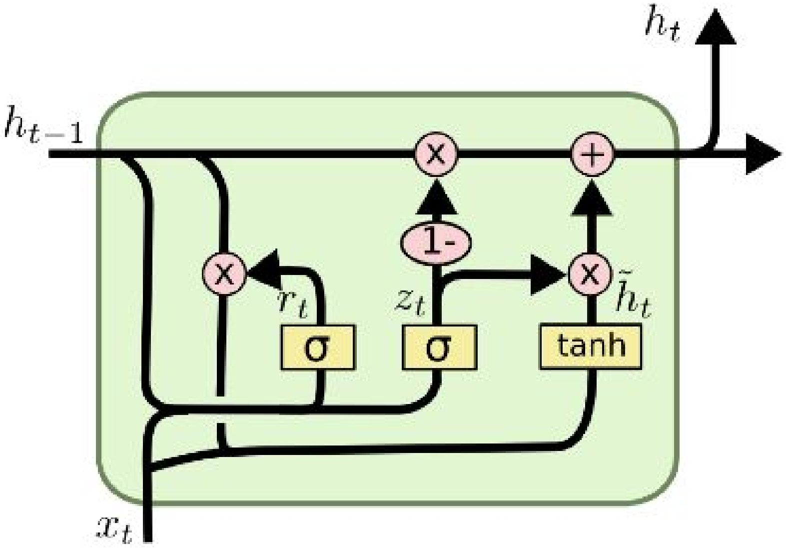
Output Gate

- ★ Controls the flow of information from the internal state x_t to the outside h_t
- ★ $o_t=1 \Rightarrow$ allows internal state out
- ★ $o_t=0 \Rightarrow$ doesn't allow internal state out



(source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Gated Recurrent Unit (GRU)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.

(source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

The Unreasonable Effectiveness of Recurrent Neural Networks

★ Famous blog entry from Andrej Karpathy (UofS)
★ Character-level language models based on multi-layer LSTMs.
★ Data:
  - Shakspare plays
  - Wikipedia
  - LaTeX
  - Linux source code

Algebraic geometry book in LaTeX

Proof. Omitted. $\square$

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. $\square$

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. $\square$

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

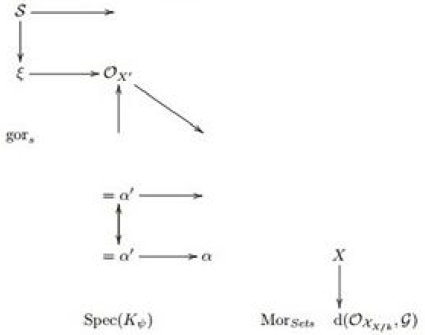
be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. $\square$

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram



is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

$\square$

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. $\square$

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. $\square$

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

Linux source code

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
```

LSTM language model demo