Recurrent neural nets applications.

Victor Kitov

v.v.kitov@yandex.ru

Table of Contents

1. Acceptor RNN scheme
   - Long sequence problem
   - New words at test time problem
   - Accounting for other predictions
2. Generative models
3. Conditional generation

RNN as acceptor[1]

- Given input sequence, acceptor predicts scalar output.
- RNN as acceptor scheme:
  1. read input sequence, produce final output
  2. apply multi-layer perceptron to final output
  3. apply SoftMax to output of MLP.

[1] Gated RNNs (LSTM, GRU) consistently outperform simple RNNs on most tasks.

Example: sentiment classification

- Sentiment classification.
  - Positive example: It's not life-affirming—it's vulgar and mean, but I liked it.
  - Negative example: It's a disappointing that it only manages to be decent instead of brilliant.
  - Accounting only for presence of positive&negative words may not work!

Example: sentiment classification

- Sentiment classification.
  - Positive example: It's not life-affirming—it's vulgar and mean, but I liked it.
  - Negative example: It's a disappointing that it only manages to be decent instead of brilliant.
  - Accounting only for presence of positive&negative words may not work!
- More advanced use:
  - May predict **degree of positivity** (classes 1,2,...5).
  - Reviews may be positive, negative and **neutral** (classes -5,...-1,0,1,...5).

RNN acceptor scheme

Acceptor RNN scheme:

1. $\mathbf{x_{1:n}} = E[w_1], ...E[w_2]$ # get word embeddings
2. $\widehat{\mathbf{z}} = RNN(\mathbf{x_{1:n}})$ # get final output of RNN
3. $\widehat{\mathbf{y}} = softmax(MLP(\widehat{\mathbf{z}}))$ # get class probabilities
4. $p(label = k|w_{1:n}) = \widehat{\mathbf{y}}_k$ # use them

Word embeddings should be pre-trained from Word2Vec or Glove.

- may be fine-tuned for given task

Better to use bidirectional RNN.

Bidirectional RNN acceptor scheme

Bidirectional RNN acceptor scheme:

1. $\mathbf{x_{1:n}} = E[w_1], ...E[w_2]$ # get word embeddings
2. $\widehat{\mathbf{z}}^f = RNN^f(\mathbf{x_{1:n}})$ # get forward RNN output
3. $\widehat{\mathbf{z}}^b = RNN^b(\mathbf{x_{n:1}})$ # get backward RNN output
4. $\widehat{\mathbf{y}} = softmax(MLP([\widehat{\mathbf{z}}^f; \widehat{\mathbf{z}}^b]))$ # get class probabilities
5. $p(label = k|w_{1:n}) = \widehat{\mathbf{y}}_k$ # use them

1 Acceptor RNN scheme
- Long sequence problem
- New words at test time problem
- Accounting for other predictions

Hierarchical RNN acceptor scheme

For long sequences (documents=sequences of sentences, long sentences=sequences of comma separated short sentences):

1. split sequence $\mathbf{w_{1:n}}$ into m non-overlapping spans: $\mathbf{w^1_{1:l_1}}, ...\mathbf{w^m_{1:l_m}}$.
2. encode each span with RNN: $\mathbf{e_i} = RNN_1(\mathbf{x^i_{1:l_i}})$
3. apply RNN to encoded sequence $\widehat{\mathbf{z}} = RNN_2(\mathbf{e_{1:m}})$.
4. $\widehat{\mathbf{y}} = softmax(MLP(\widehat{\mathbf{z}}))$
5. $p(label = k|w_{1:n}) = \widehat{\mathbf{y}}_k$

Hierarchical RNN comments

Comments:

- May use biRNN for getting $\mathbf{e_i}$ and $\widehat{\mathbf{z}}$.
- Alternative architecture:
  - get whole sequence $\widehat{\mathbf{z}}_{\mathbf{1:m}} = RNN_2^*(\mathbf{e}_{\mathbf{1:m}})$
  - $\widehat{\mathbf{y}} = softmax(MLP(\frac{1}{m}\sum_{i=1}^{m}\widehat{\mathbf{z}}_\mathbf{i}))$

1 Acceptor RNN scheme
- Long sequence problem
- New words at test time problem
- Accounting for other predictions

Part-of-speech tagging application

- Use biRNN acceptor scheme.
- Unknown words problem
  - word is not present in the training set
  - word appears in the test set
- Solutions:
  - use **words dropout**
  - use words dropout & **character based embedding**

Words dropout

- Problem: unknown words at test time.
  - Replace them with token UNK
  - How to get embedding for UNK? At training time replace some words with UNK!
- Naive solution: replace least frequent words:
  - bad, because they are very informative!
- Good solution: replace random words with UNK
  - replacement probability is constant
  - replacement probability increases for rare words

$$p(w \to UNK) = \frac{\alpha}{\#(w) + \alpha}$$

$\alpha > 0$ - replacement agressiveness hyperparameter.

- Benefits:
  - model deals with new unknown words
  - model doesn't overfit to particular words (regularization)

Character based embedding

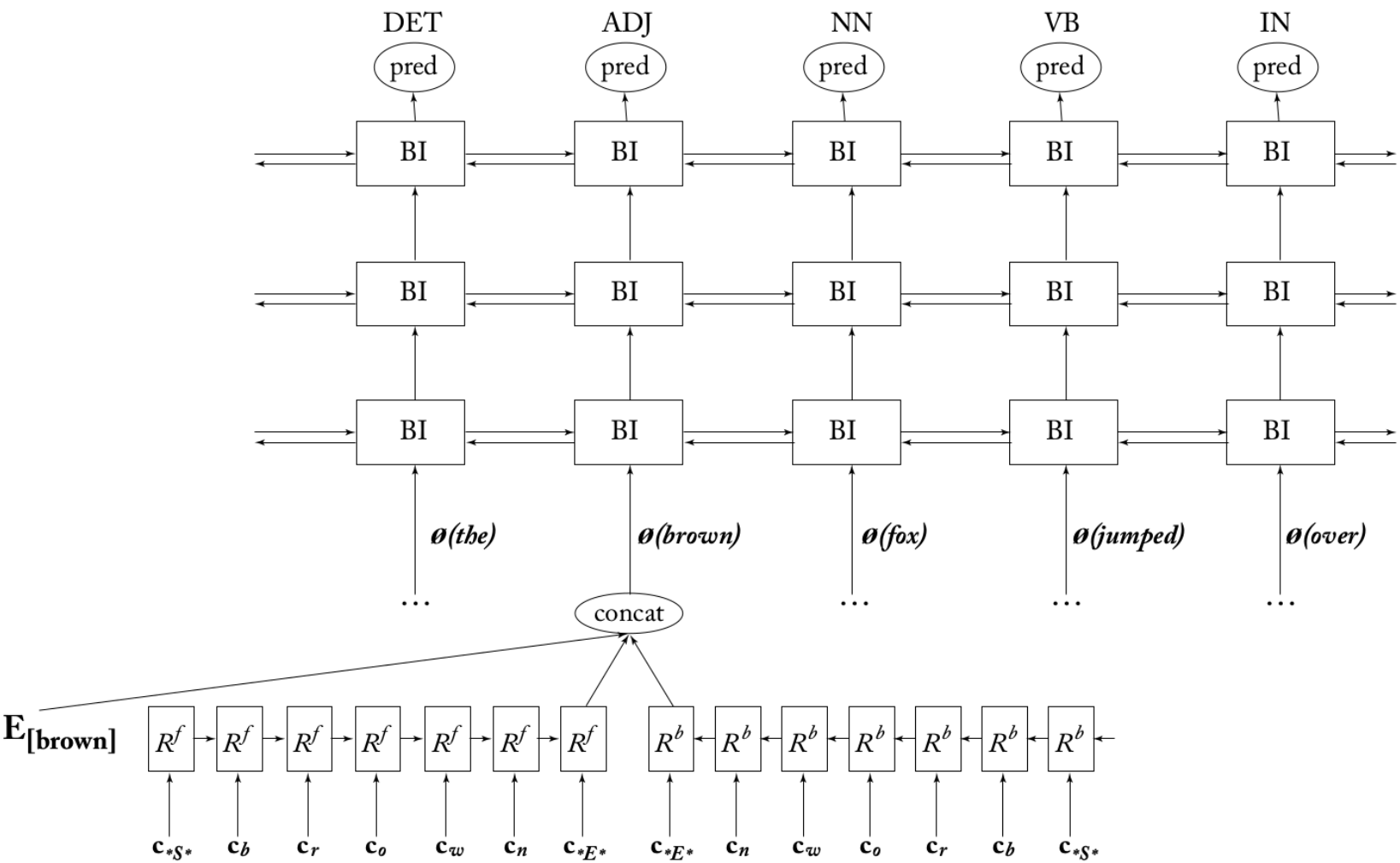
Consider word as a sequence of l characters: $w = c_{1:l}$

1. replace characters with their embeddings $c_i \rightarrow \mathbf{c_i}$
2. pass forward character RNN $z_f = RNN^f(\mathbf{c_{1:l}})$
3. pass backward character RNN $z_b = RNN^b(\mathbf{c_{l:1}})$
4. use concatenation of word embedding & 2 character embeddings:

$$w \rightarrow [E[w]; z_f; z_b]$$

- Benefit: for unknown word we still get reasonable unique representation!
- Comments:
  - better to replace character bigrams with bigram embeddings
  - may use convolution & pooling instead of RNN for characters representation.

Illustration



1. Acceptor RNN scheme
   - Long sequence problem
   - New words at test time problem
   - Accounting for other predictions

POS tagging accounting for nearby tags

- Tag of w_i depends on tags of $w_{i-1}, w_{i-1}, ...$
- Solution:
  1. embed tags: $y \to E[y]$
  2. at time t for current tagging y_t use embeddings of previous tags $y_{t-1}, ..., y_{t-k}$:

$$p(y_t = j | w_{1:n}, y_{t-1}, ...y_{t-k})$$
$$= \text{softmax}\left(MLP\left(biRNN\left(w_{1:n}, t\right)\right); E[y_{t-1}], ...E[y_{t-k}]\right)[j]$$

Comments

Comments:

- May account for all previous tags with RNN:

$$p(y_t = j | w_{1:n}, y_{t-1}, \dots y_{t-k})$$
$$= \text{softmax}\left(MLP\left(biRNN\left(w_{1:n}, t\right)\right); RNN^{tag}\left(\mathbf{y_{1:t-1}}\right)\right)[j]$$

- Prediction making strategies:
  - greedy
  - beam search (better)
  - using dynamic programming (optimal, requires Viterbi algorithm)

Table of Contents

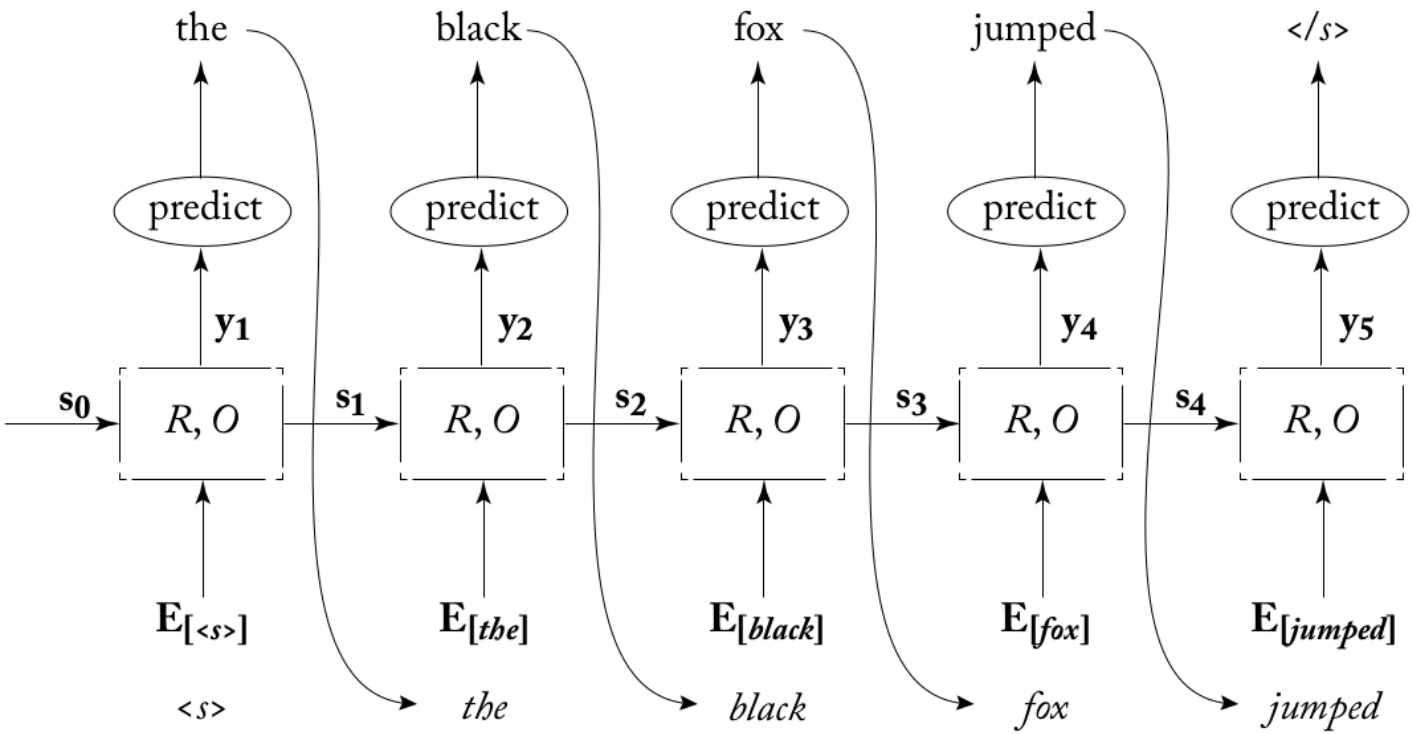
1 Acceptor RNN scheme

2 Generative models
- Beam search
- Demos

3 Conditional generation

Generative models

Output is passed to input until we get <end-of-sequence> word.



Teaching generative RNNs

For sentence: "the black fox jumped" we form training set:

1. (<s>)→the
2. (<s>,the)→black
3. (<s>,the,black)→fox
4. (<s>,the,black,fox)→jumped
5. (<s>,the,black,fox,jumped)→</s>

This works, but this training assumes network predicts based on correct previous decisions.

- but it can make error in the middle of the sentence!

Basic schemes

Most probable generation (least diverse, minimal inconsistencies):

repeat
$p(y_t) = RNN(\mathbf{y_{1:t-1}})$
sample $y_t \sim Categorical(p(y_t))$
$\mathbf{y_t} = E[y_t]$
until y_t=<end-of-sequence>

Random generation (more diverse, more inconsistencies):

repeat
$p(y_t) = RNN(\mathbf{y_{1:t-1}})$
$y_t = \arg\max_y p(y_t)$
$\mathbf{y_t} = E[y_t]$
until y_t=<end-of-sequence>

More advanced schemes

- Denote $s_i = score(y_t = i)$.
- Since final layer is softmax:

$$p(y_t = i) = \frac{e^{s_i}}{\sum_k e^{s_k}}$$

we may additionally control sharpness/diversity of output with $\alpha > 0$:

$$p(y_t = i|\alpha) = \frac{e^{\alpha s_i}}{\sum_k e^{\alpha s_k}}$$

  - $\alpha = 0$: uniform generation
  - $\alpha = 1$: random generation
  - $\alpha \to \infty$: most probable generation.

- May use **beam search** for higher probability output.

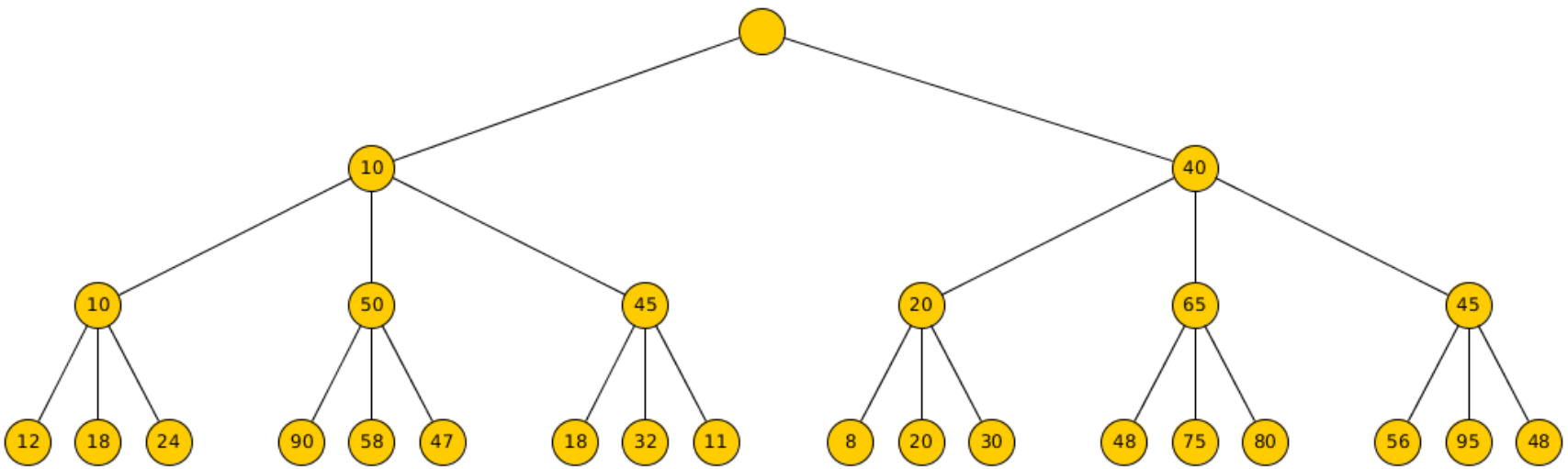
2 Generative models
- Beam search
- Demos

Introduction

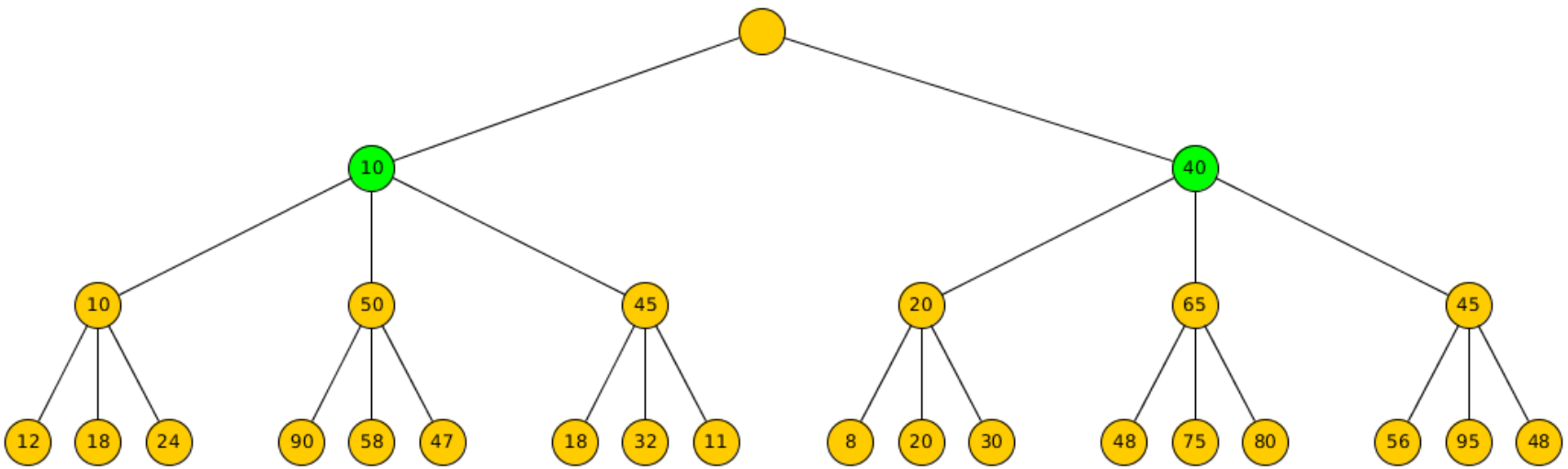
- Need to make sequential decisions with multiplicative/additive cost.
- E.g. POS tagging, machine translation:
  1. y_1, score $p(y_1|x_{1:n})$
  2. y_2, score $p(y_1 y_2|x_{1:n}) = p(y_1|x_{1:n})p(y_2|y_1, x_{1:n})$
  3. y_3, score $p(y_1|x_{1:n})p(y_2|y_1, x_{1:n})p(y_3|y_1, y_2, x_{1:n})$

  ...
- This can be reformulated as finding leaf node of the tree with maximal score.
- Simplest solution: greedy (top-1) search
  - complexity $O(BD)$, where B-branching factor, D-tree depth.
- Beam search: greedy (top-K) search

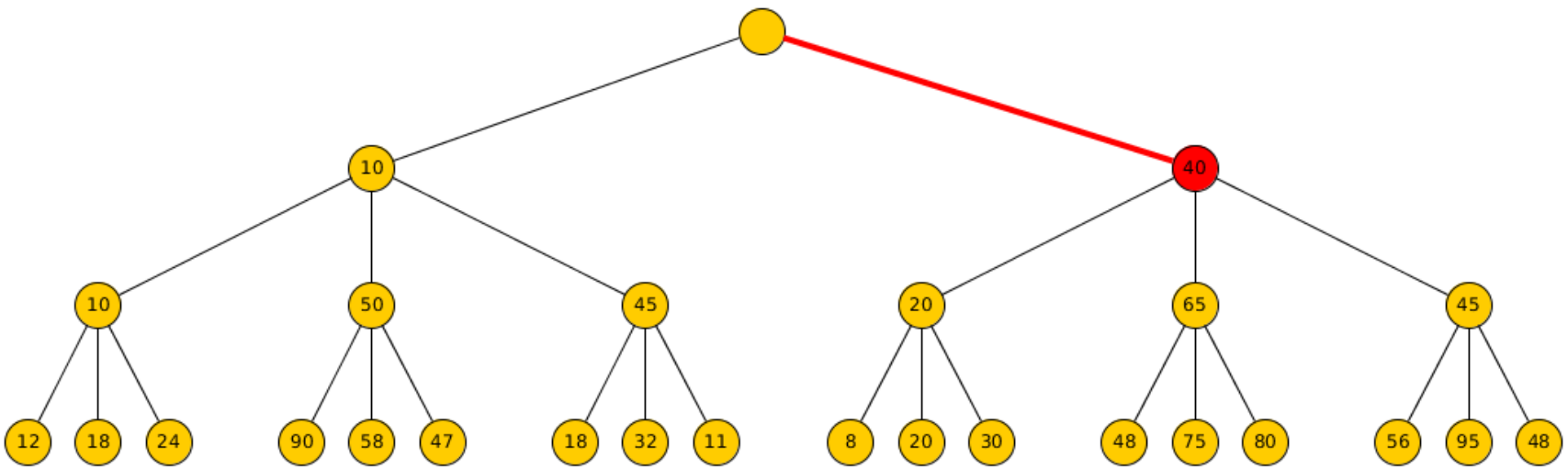
Greedy (top-1) search



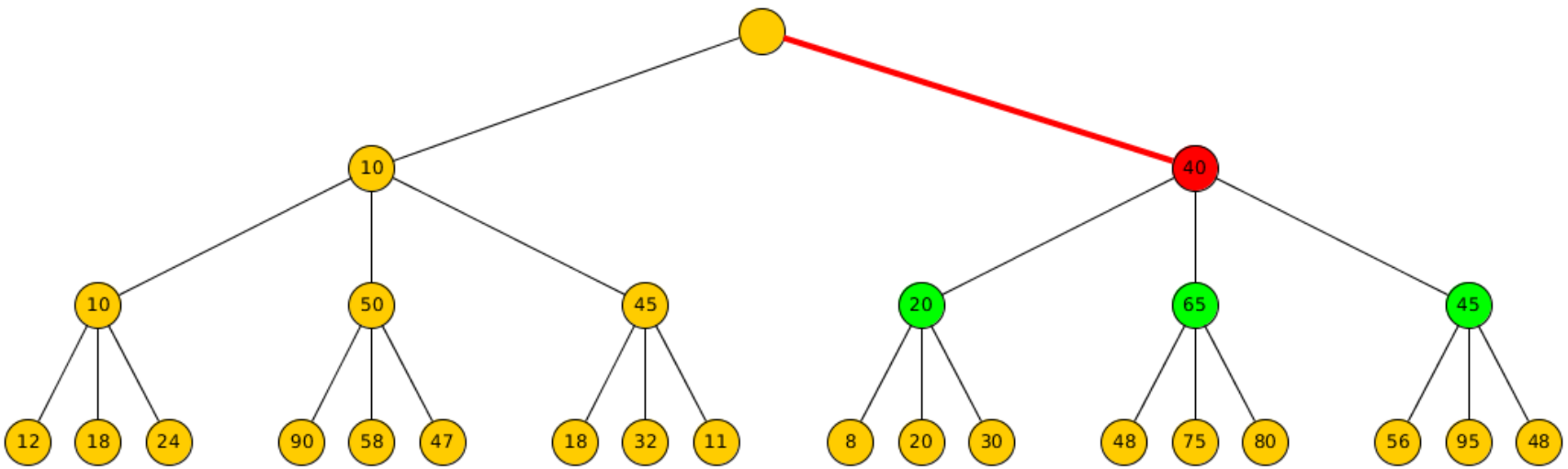
Greedy (top-1) search



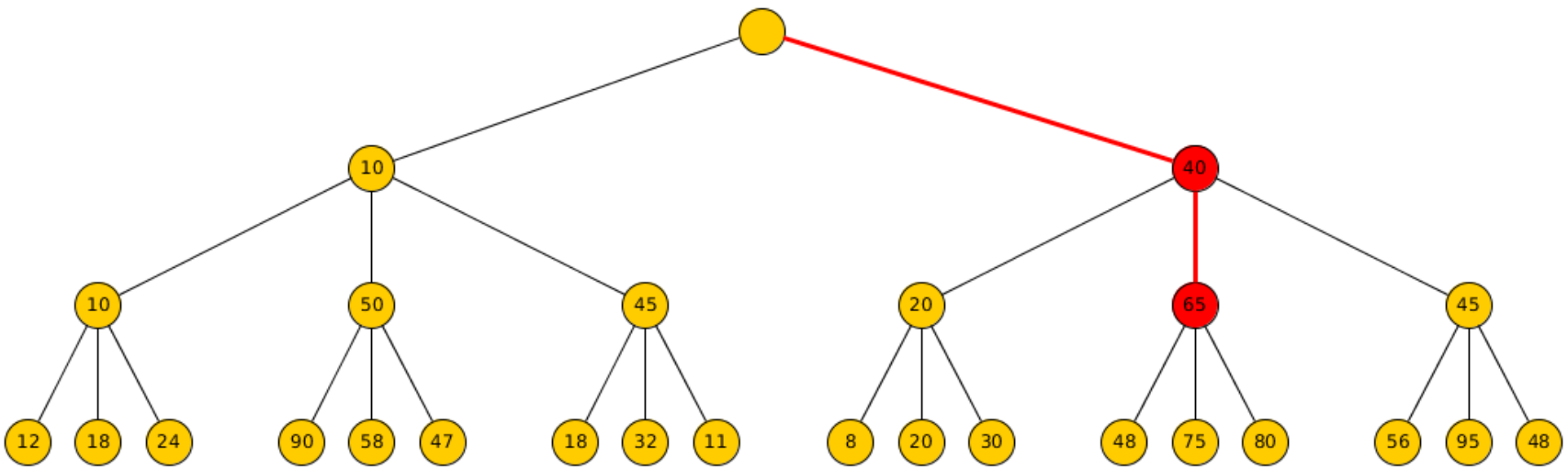
Greedy (top-1) search



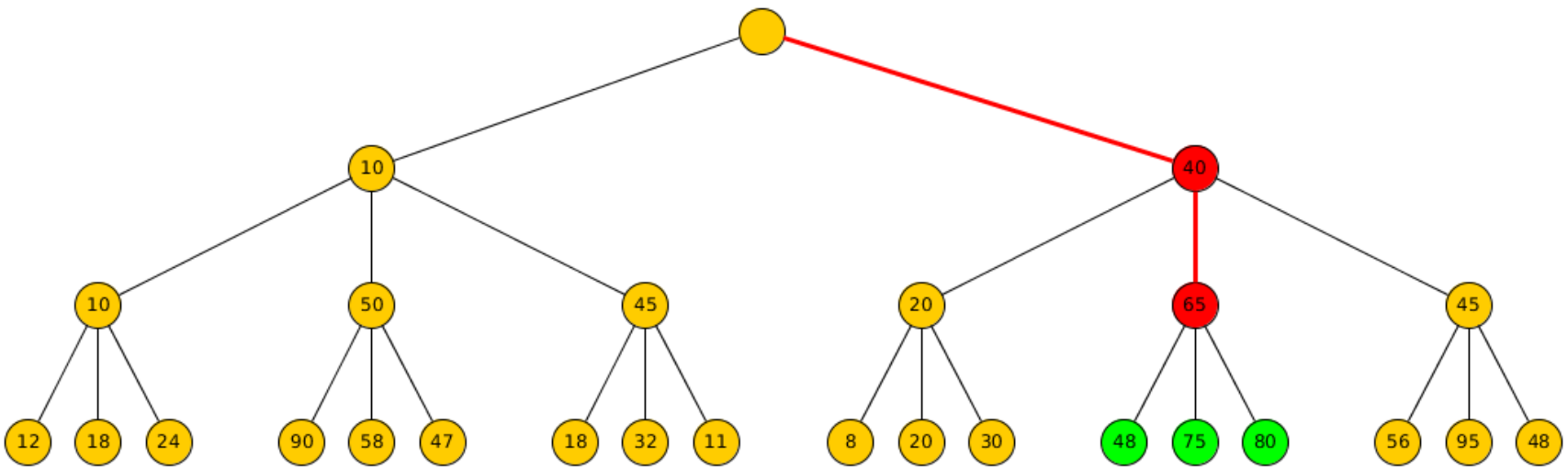
Greedy (top-1) search



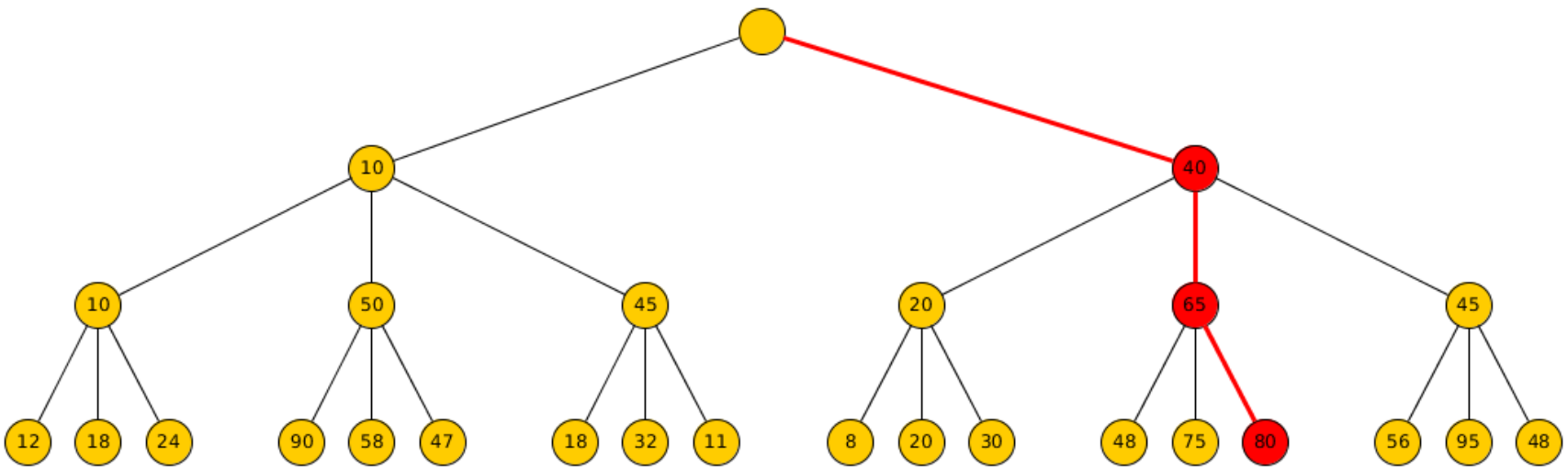
Greedy (top-1) search



Greedy (top-1) search



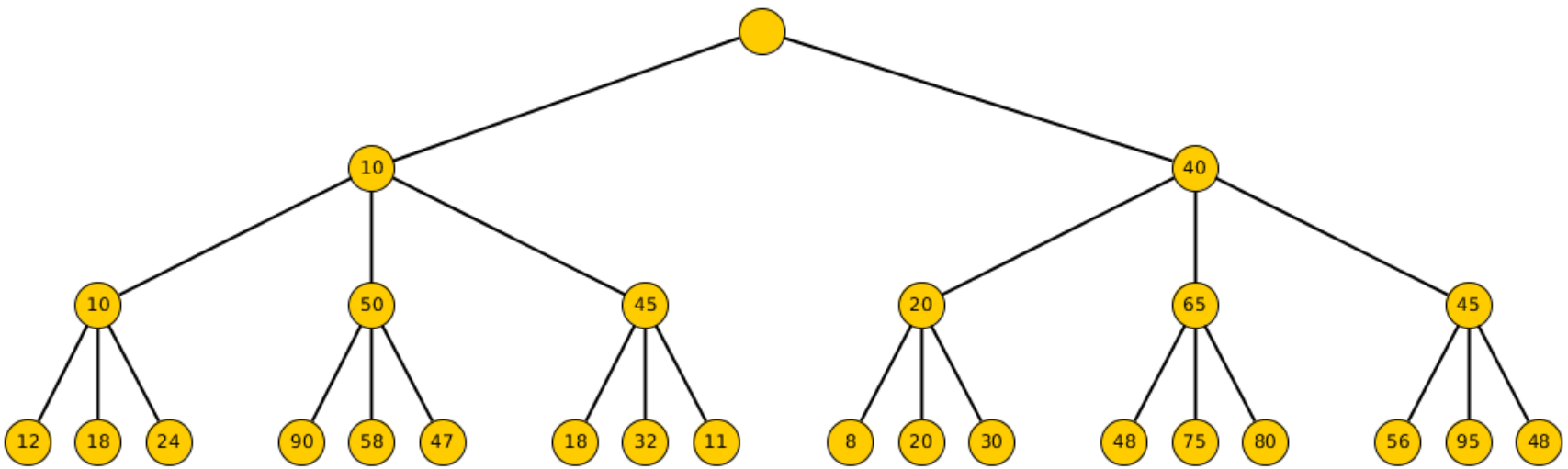
Greedy (top-1) search



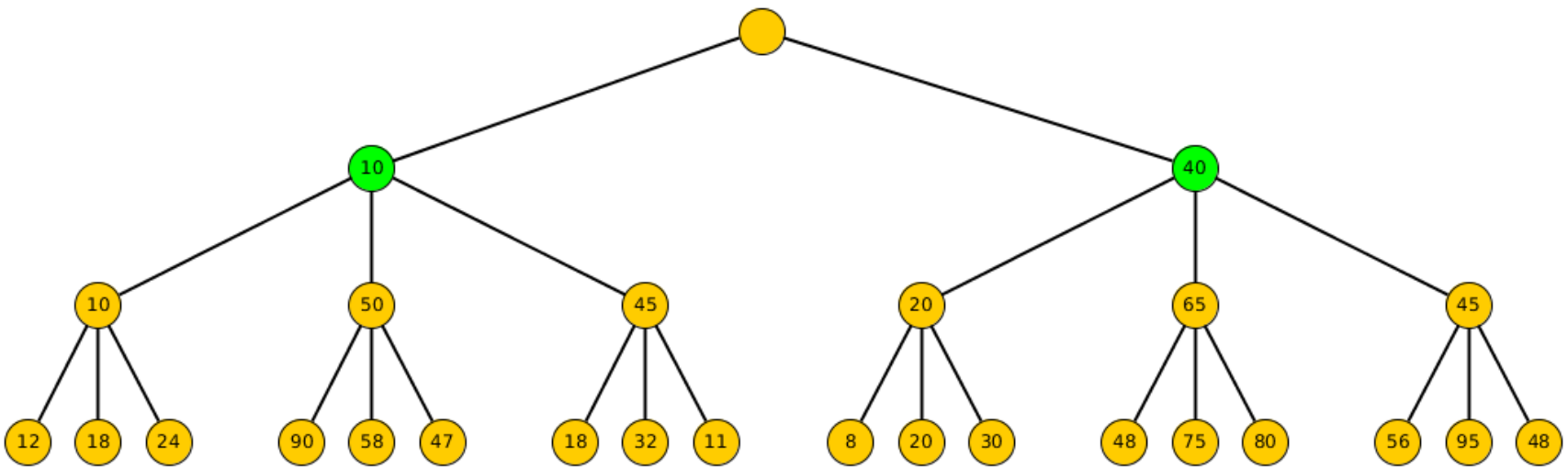
Beam search

- Beam search: greedy breadth first search
- Considers K best alternatives instead of one.
  - complexity $O(KBD)$, where B-branching factor, D-tree depth.
- For K large enough ($K \geq B^{D-1}$) reduces to exhaustive search.

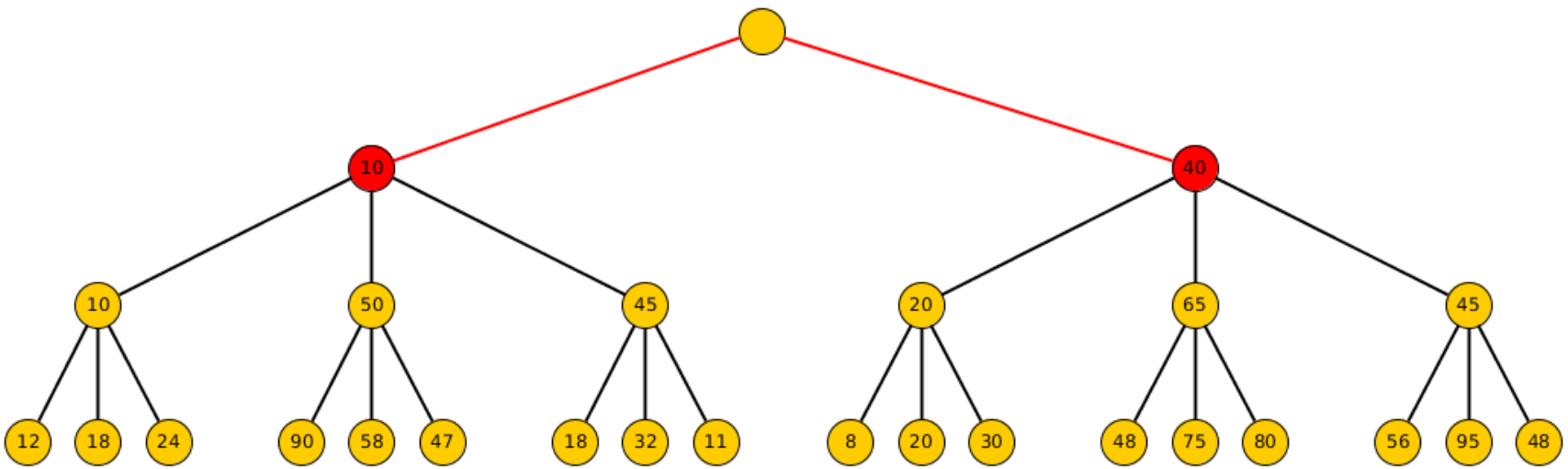
Beam (top-K) search, K=2



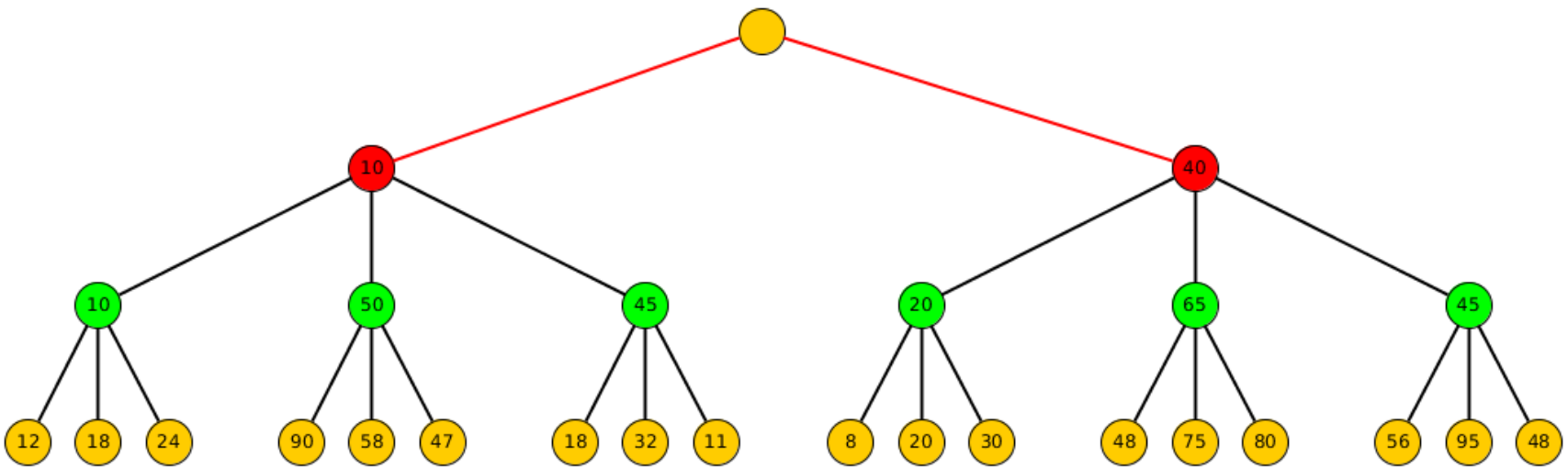
Beam (top-K) search, K=2



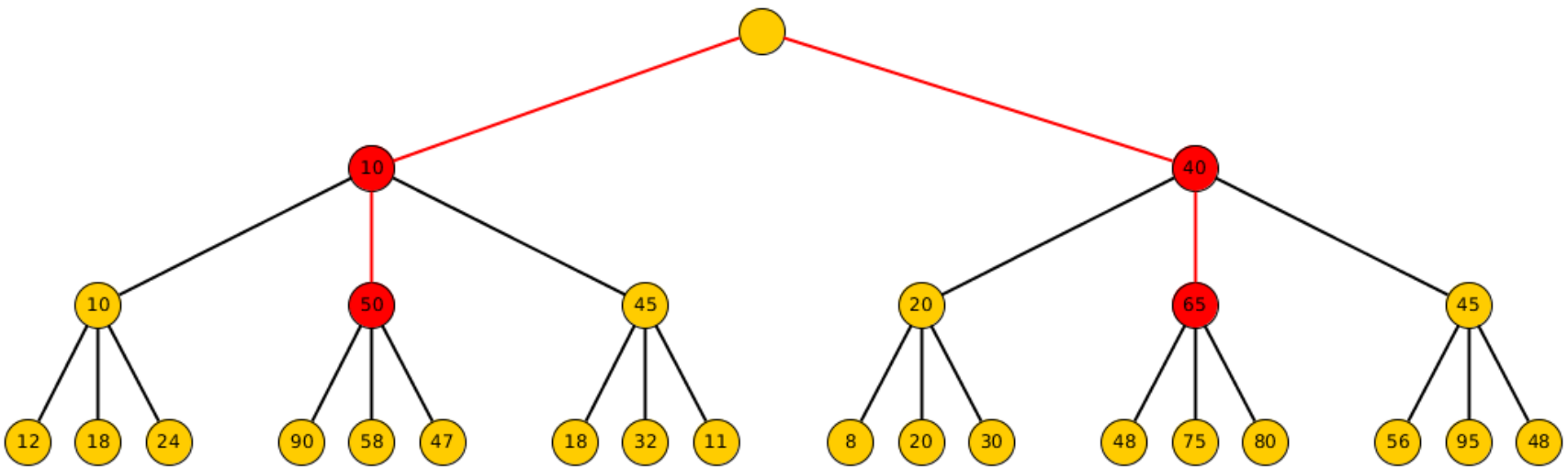
Beam (top-K) search, K=2



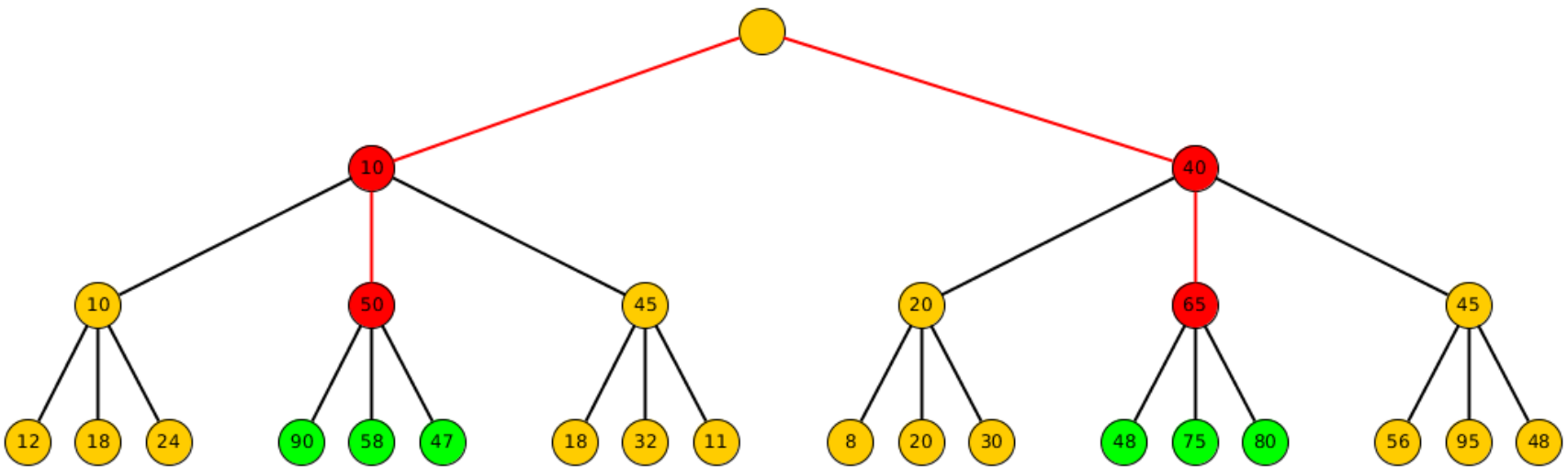
Beam (top-K) search, K=2



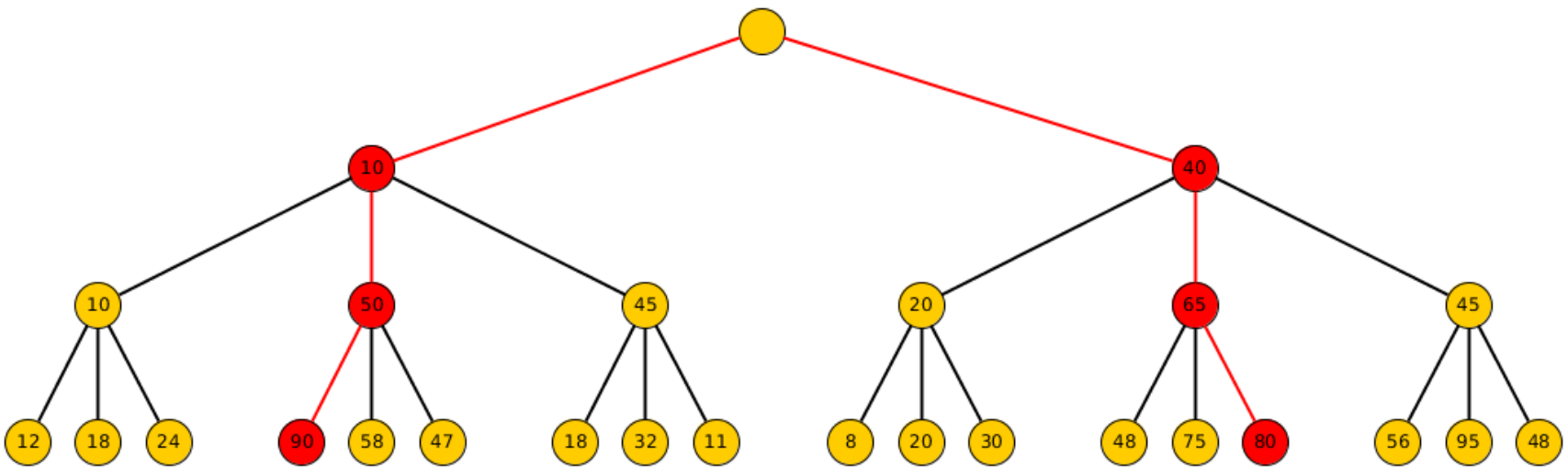
Beam (top-K) search, K=2



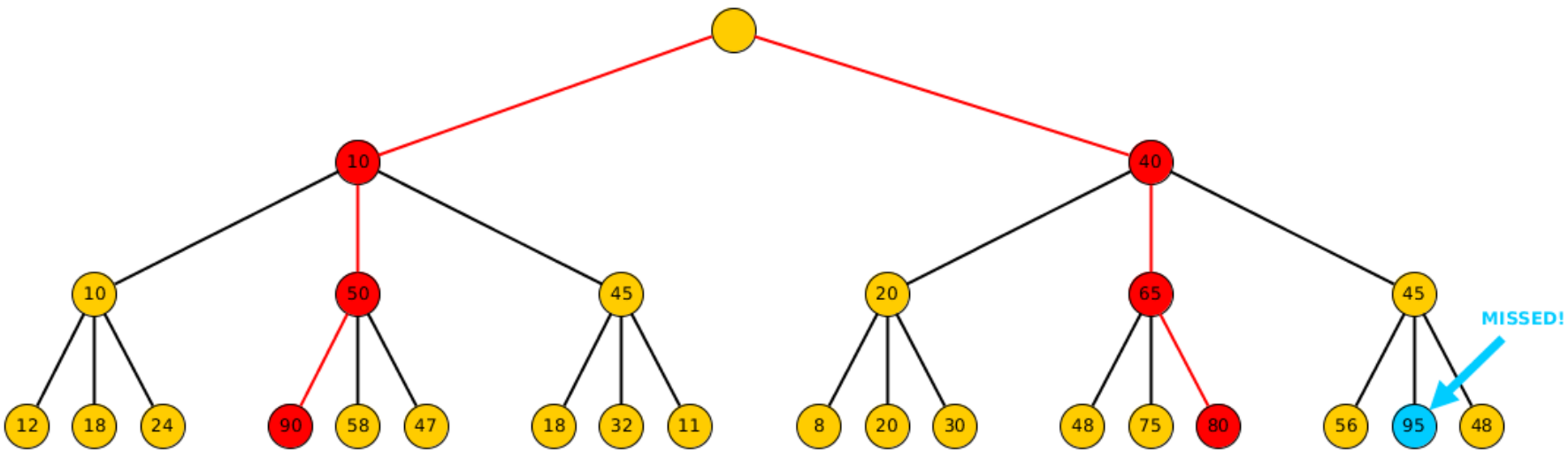
Beam (top-K) search, K=2



Beam (top-K) search, K=2



Beam (top-K) search, K=2



Beam search

- Uses PriorityQueue - priority queue class with methods
  - push(elements,scores): pushes elements with corresponding scores
  - getKbest(K): retrieves K elements with highest scores.

Beam search

- $y \in \{\omega_1, ...\omega_I\}$ - possible class values
- $\mathbf{s} = [i_1, ...i_{t-1}]$ - list of $t-1$ indices
- $\mathbf{y_s} = \omega_{i_1}...\omega_{i_{t-1}}$ - sequence of outputs with corresponding indices.
- $S = [\mathbf{s_1}, ...\mathbf{s_K}]$ - list K best index sequences.

$t=1$, $S=\{\}$, q=PriorityQueue()

```
while StopCondition==False:
    for s ∈ S:
        for i ∈ {1,2,...I}:
            q.push([s;i], p(y_t = i|x_{1:n}; y_s)p(y_s|x_{1:n}))
        q=PriorityQueue(q.getKbest(K))
    S = q.getKbest()
    t=t+1
```

Possible stop conditions

Possible stop conditions:

- $t > n$
  - intuition: K best tag sequences $y_{1:n}$ for $x_{1:n}$ were obtained.
  - useful for fixed length output
  - e.g. in POS tagging.
- every $\mathbf{s} \in S$ ends with index, corresponding to <end-of-sequence> tag.
  - intuition: best K sequences ended.
  - useful for variable length output
  - e.g. in language generation, machine translation.

2 Generative models
- Beam search
- Demos

RNN trained on Shakespeare[2]

3 layer RNN with 512 neurons each trained on Shakespeare works:

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.
```

[2]All demos of this section taken from Karpathy [2015]. All RNNs operate on characters.

RNN trained on Latex sources

- Multilayer LSTM trained on latex sources of 1 math book.
- Output compiled after few fixes:

Proof. Omitted. □

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 $X_{étale}$ 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. □

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

Proof. See Spaces, Lemma ??. □

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. □

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

$$\begin{array}{ccc} \mathcal{S} & \longrightarrow & \\ \downarrow & & \\ \xi & \longrightarrow & \mathcal{O}_{X'} \\ & & \uparrow \\ \mathrm{gor}_s & & \end{array} \qquad \begin{array}{ccc} & = \alpha' & \longrightarrow \\ & \updownarrow & \\ & = \alpha' & \longrightarrow \alpha \end{array} \qquad \begin{array}{c} X \\ \downarrow \\ \mathrm{d}(\mathcal{O}_{X_{X/k}}, \mathcal{G}) \end{array}$$

$$\mathrm{Spec}(K_\psi) \qquad \mathrm{Mor}_{Sets}$$

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.

□

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. □

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

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{étale}}) \longrightarrow \mathcal{O}_{X_t}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\mathbb{V}})$$

is an isomorphism of covering of $\mathcal{O}_{X_t}$. 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. □

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.

RNN trained on Linux sources

3-layer LSTM trained on concatenated Linux source code.

```
/*
 * If this error is set, we will need anything right after that BSD.
 */
static void action_new_function(struct s_stat_info *wb)
{
  unsigned long flags;
  int lel_idx_bit = e->edd, *sys & ~((unsigned long) *FIRST_COMPAT);
  buf[0] = 0xFFFFFFFF & (bit << 4);
  min(inc, slist->bytes);
  printk(KERN_WARNING "Memory allocated %02x/%02x, "
    "original MLL instead\n"),
    min(min(multi_run - s->len, max) * num_data_in),
    frame_pos, sz + first_seg);
  div_u64_w(val, inb_p);
  spin_unlock(&disk->queue_lock);
  mutex_unlock(&s->sock->mutex);
  mutex_unlock(&func->mutex);
  return disassemble(info->pending_bh);
}

static void num_serial_settings(struct tty_struct *tty)
{
  if (tty == tty)
    disable_single_st_p(dev);
  pci_disable_spool(port);
  return 0;
}
```

Comments on generating code

- uses properly strings, pointer notation
- opens and closes brackets {[correctly
- learns to indent its code
- adds license and comments
- can't keep track of variable names:
  - uses undefined variables
  - declares variables it never uses
  - returns non-existing variables

Keeping track of variables requires too long memory!

What model learns

- RNN trained on Leo Tolstoy's "War and Peace"
- Generated samples every 100 iterations of training.
- Model learns incrementally:
  1. to separate words with spaces
  2. quotes and periods
  3. shortest and most common words such as “we”, “he”, “and”, etc.
  4. quotations and question/exclamation marks
  5. long words
- Output after long training:

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Particular neuron activations visualization

Cell sensitive to position in line:

```
The sole importance of the crossing of the Berezina lies in the fact
that it plainly and indubitably proved the fallacy of all the plans for
cutting off the enemy's retreat and the soundness of the only possible
line of action--the one Kutuzov and the general mass of the army
demanded--namely, simply to follow the enemy up. The French crowd fled
at a continually increasing speed and all its energy was directed to
reaching its goal. It fled like a wounded animal and it was impossible
to block its path. This was shown not so much by the arrangements it
made for crossing as by what took place at the bridges. When the bridges
broke down, unarmed soldiers, people from Moscow and women with children
who were with the French transport, all--carried on by vis inertiae--
pressed forward into boats and into the ice-covered water and did not,
surrender.
```

Cell that turns on inside quotes:

```
"You mean to imply that I have nothing to eat out of.... On the
contrary, I can supply you with everything even if you want to give
dinner parties," warmly replied Chichagov, who tried by every word he
spoke to prove his own rectitude and therefore imagined Kutuzov to be
animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating
smile: "I meant merely to say what I said."
```

Cell that robustly activates inside if statements:

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
        siginfo_t *info)
{
  int sig = next_signal(pending, mask);
  if (sig) {
    if (current->notifier) {
      if (sigismember(current->notifier_mask, sig)) {
        if (!(current->notifier)(current->notifier_data)) {
          clear_thread_flag(TIF_SIGPENDING);
          return 0;
        }
      }
    }
    collect_signal(sig, pending, info);
  }
  return sig;
}
```

Particular neuron activations visualization

Cell that turns on inside comments and quotes:

```
/* Duplicate LSM field information.  The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
           struct audit_field *sf)
{
  int ret = 0;
  char *lsm_str;
  /* our own copy of lsm_str */
  lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
  if (unlikely(!lsm_str))
    return -ENOMEM;
  df->lsm_str = lsm_str;
  /* our own (refreshed) copy of lsm_rule */
  ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
                 (void **)&df->lsm_rule);
  /* Keep currently invalid fields around in case they
   * become valid after a policy reload. */
  if (ret == -EINVAL) {
    pr_warn("audit rule for LSM \'%s\' is invalid\n",
      df->lsm_str);
    ret = 0;
  }
  return ret;
}
```

Cell that is sensitive to the depth of an expression:

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
  int i;
  if (classes[class]) {
    for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
      if (mask[i] & classes[class][i])
        return 0;
  }
  return 1;
}
```

Table of Contents

1 Acceptor RNN scheme

2 Generative models

3 Conditional generation

Conditional generation

Unconditional generation:

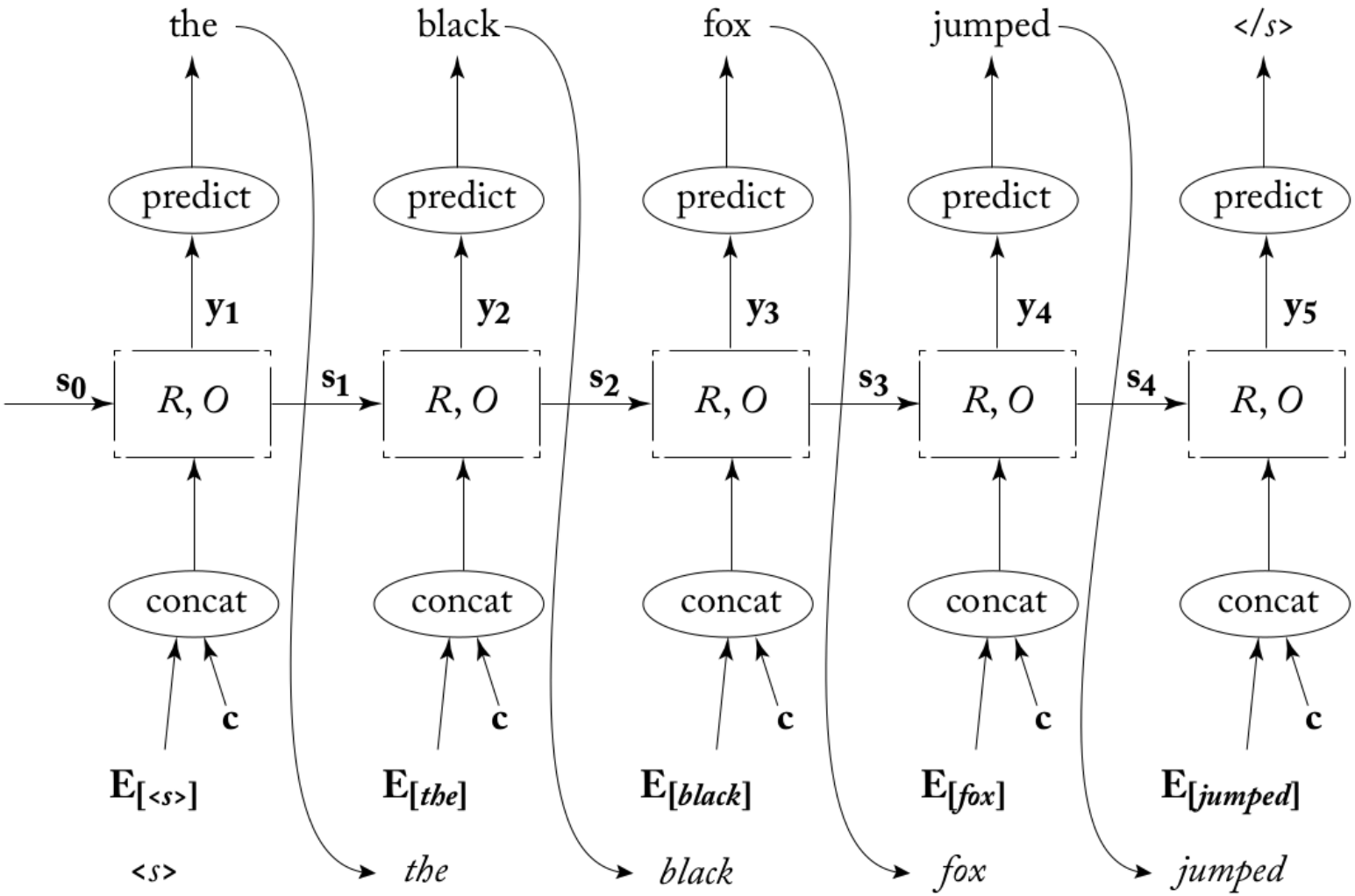
$$p(y_{t+1} = k|\widehat{\mathbf{y}}_{\mathbf{1:t}}) = \text{softmax}(RNN(\widehat{\mathbf{y}}_{\mathbf{1:t}}))$$
$$\widehat{y}_{t+1} \sim p(y_{t+1}|\widehat{\mathbf{y}}_{\mathbf{1:t}})$$
$$\widehat{\mathbf{y}}_{\mathbf{t+1}} = E[\widehat{y}_{t+1}]$$

Conditional generation:

$$p(y_{t+1} = k|\widehat{\mathbf{v}}_{\mathbf{1:t}}) = \text{softmax}(RNN(\widehat{\mathbf{v}}_{\mathbf{1:t}}))$$
$$\widehat{y}_{t+1} \sim p(y_{t+1}|\widehat{\mathbf{y}}_{\mathbf{1:t}})$$
$$\widehat{\mathbf{v}}_{\mathbf{t+1}} = [E[\widehat{y}_{t+1}]; \mathbf{c}]$$

Condition is encoded with $\mathbf{c} \in \mathbb{R}^m$

Illustration



Examples of conditions

Examples of conditions:

- topic for generated text
- polarity of review
- geographic region of the author
- level of vocabulary used
- first person or 3rd person story style

Sequence to sequence models

1. Use encoder to obtain condition $\mathbf{c}$:

$$\mathbf{c} = RNN^{enc}(\mathbf{x_{1:n}})$$

2. Then use standard conditional generation:

$$p(y_{t+1} = k|\widehat{\mathbf{v}}_{\mathbf{1:t}}) = \text{softmax}(RNN(\widehat{\mathbf{v}}_{\mathbf{1:t}}))$$
$$\widehat{y}_{t+1} \sim p(y_{t+1}|\widehat{\mathbf{y}}_{\mathbf{1:t}})$$
$$\widehat{\mathbf{v}}_{\mathbf{t+1}} = [E[\widehat{y}_{t+1}]; \mathbf{c}]$$

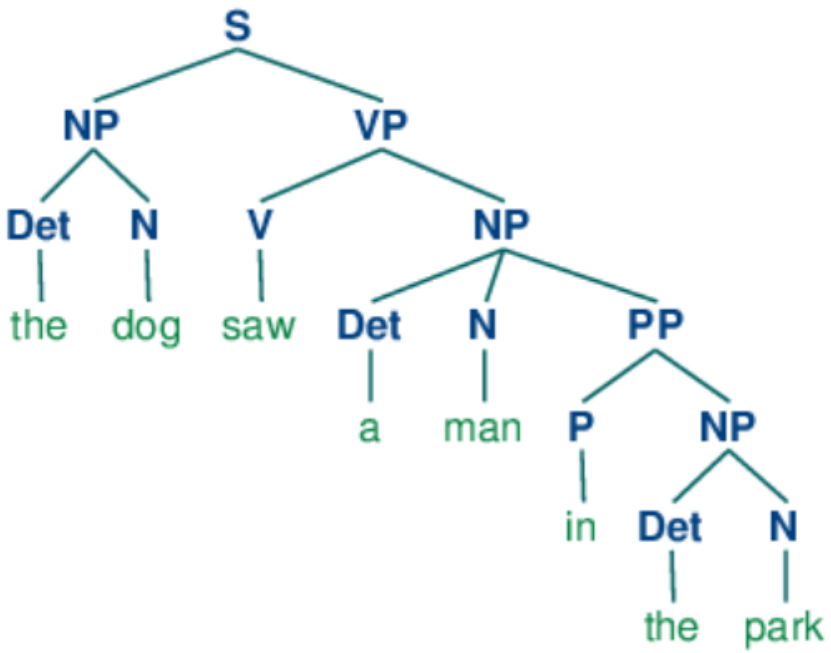
Applications

Some applications of sequence to sequence models:

- **machine translation**
  - models with attention work better
- **email auto-response**
  - trained on email-response pairs
- **sentence compression**
  - compressed sentence should: be grammatical, be shorter and retain the important information from the original sentence.
  - e.g. "I would really like to have a delicious tasty cup of coffee"->"I want coffee"

Applications

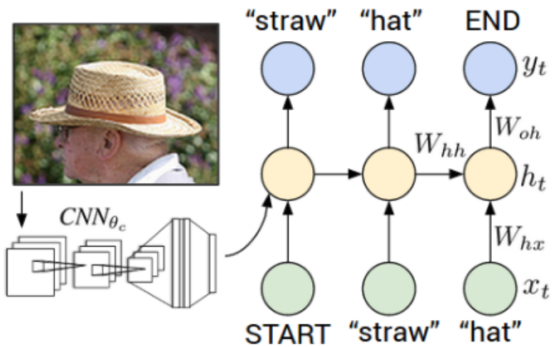
- syntactic parsing



- The dog saw a man in the park->[S [NP [Det] [N] NP] [VP [V] [NP [Det] [N] [PP [P [NP [Det] [N] NP]]PP] NP] VP] S]

Complex conditions

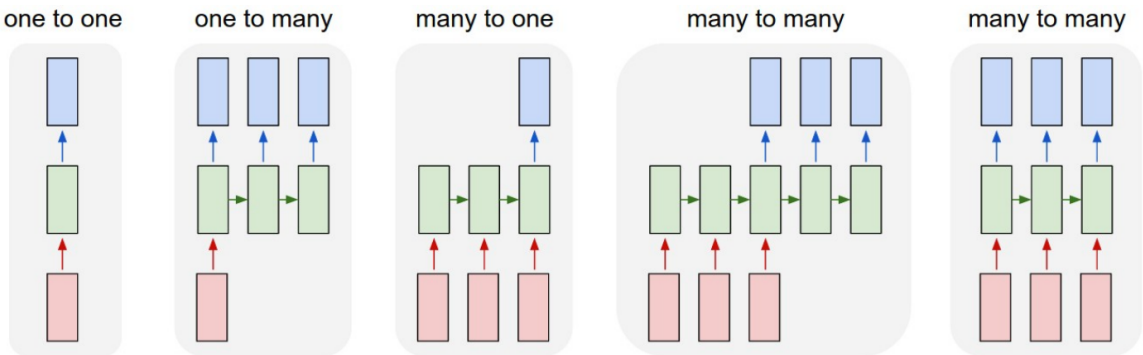
- **Image captioning:** image->textual description of the image



  - Image->context with CNN. Context may sent to decoder:
    - as context vector c appended to each input
    - or as initial state of unconditional RNN (see figure)
- **Visual story telling:** series of images->describe progression of images.
- **Personalized communication:** context describes user in dialog system.
  - his age, gender, interests.

Conclusion

NN & RNN architectures:



Examples, where these architectures arise:

- **one to one:** classical classification, image classification.
- **one to many:** image captioning, story generation based on topic.
- **many to one:** text classification, sentiment analysis.
- **many to many:** machine translation, summarization.
- **synced many to many:** POS tagging, activity detection on video.