Recurrent Neural Networks (RNN)
Long Short-Term Memory (LSTM)
Recap: Second-Order Optimization

second-order Taylor expansion:

\[ f(x) = f(a) + (x - a)^T \nabla f + \frac{1}{2} (x - a)^T H (x - a) \]

Solving for the critical point we obtain the Newton parameter update:

\[ x^* = a - H^{-1} \nabla f \]

Hessian has \( O(N^2) \) elements
Inverting takes \( O(N^3) \)
\( N = \text{Millions} \)

Q: Why is this bad for deep learning?
Regularization: Dropout

In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common

Consider a single neuron.

Without dropout: \( E[a] = w_1x + w_2y \)

With dropout we have:

\[
E[a] = \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y)
\]

\[
= \frac{1}{2}(w_1x + w_2y)
\]

At test time, **multiply** by dropout probability.

Compute the expectation

\[
y = f(x) = E_z[f(x, z)] = \int p(z)f(x, z)dz
\]
Regularization: Data Augmentation

Load image and label

“cat”

Transform image

CNN

Compute loss
Gradient clipping: prevent large gradient step

Large gradient step will likely destabilize training (gradients are noisy!) Large gradient update can be caused by many issues, e.g., large weights, large input, bad loss function / activation function, ...
Should always first try to fix the root cause (normalization, better loss / activation function, etc.)

But if all things fail ... just clip the gradient

$$g_{new} = \min\left(1, \frac{\lambda}{\|g\|}\right) \times g$$

$g$: original gradient
$\lambda$: clipping threshold

```python
# Zero the gradients.
optimizer.zero_grad()

# Perform forward pass.
outputs = model(inputs)

# Compute the loss.
loss = loss_function(outputs, targets)

# Perform backward pass (compute gradients).
loss.backward()

# Clip the gradients.
torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)

# Update the model parameters.
optimizer.step()
```
Transfer Learning with CNNs

1. Train on Imagenet
   - FC-1000
   - FC-4096
   - FC-4096
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-128
   - Conv-128
   - MaxPool
   - Conv-64
   - Conv-64
   - Image

2. Small Dataset (C classes)
   - FC-C
   - FC-4096
   - FC-4096
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-128
   - Conv-128
   - MaxPool
   - Conv-64
   - Conv-64
   - Image

   Reinitialize this and train

3. Bigger dataset
   - FC-C
   - FC-4096
   - FC-4096
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-128
   - Conv-128
   - MaxPool
   - Conv-64
   - Conv-64
   - Image

   Train these

   With bigger dataset, train more layers

   Freeze these

   Lower learning rate when finetuning; 1/10 of original LR is good starting point

Razavian et al., “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014
Transfer learning is pervasive…
(it’s the norm, not an exception)

Pre-training
MLM on unlabelled data

Generic Language Model

word2vec
GloVe
skip-thought
InferSent
ELMo
ULMFiT
GPT
BERT

Fine-tuning
Cross-entropy on task labels
classification
sequence labeling
Q&A
....

Train with Task-specific Labels

https://ruder.io/recent-advances-lm-fine-tuning/
Diagnose your training
(without tons of GPUs)
Diagnose your training

Step 1: Check initial loss

Turn off weight decay, sanity check loss at initialization
e.g. $\log(C)$ for softmax with $C$ classes

Reminder: $L = -\log p = -\log(1/C) = \log(C)$
Diagnose your training

**Step 1**: Check initial loss

**Step 2**: Overfit a small sample

Try to train to 100% training accuracy on a small sample of training data (~5-10 minibatches); fiddle with architecture, learning rate, weight initialization

Loss not going down? LR too low, bad initialization, bug in code or errors in training labels

Loss explodes to Inf or NaN? LR too high, bad initialization, bug in code
Diagnose your training

Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Find LR that makes loss go down

Use the architecture from the previous step, use all training data, turn on small weight decay, find a learning rate that makes the loss drop significantly within ~100 iterations

Good learning rates to try: 1e-3, 3e-4, 1e-4
Diagnose your training

**Step 1:** Check initial loss  
**Step 2:** Overfit a small sample  
**Step 3:** Find LR that makes loss go down  
**Step 4:** Coarse grid, train for ~1-5 epochs

Choose a few values of learning rate and weight decay around what worked from Step 3, train a few models for ~1-5 epochs.

Good weight decay to try: 1e-4, 1e-5, 0
Diagnose your training

**Step 1**: Check initial loss
**Step 2**: Overfit a small sample
**Step 3**: Find LR that makes loss go down
**Step 4**: Coarse grid, train for ~1-5 epochs
**Step 5**: Refine grid, train longer

Pick best models from Step 4, train them for longer (~10-20 epochs) without learning rate decay
Diagnose your training

**Step 1:** Check initial loss
**Step 2:** Overfit a small sample
**Step 3:** Find LR that makes loss go down
**Step 4:** Coarse grid, train for ~1-5 epochs
**Step 5:** Refine grid, train longer
**Step 6:** Look at loss and accuracy curves
Accuracy still going up, you need to train longer
Huge train / val gap means overfitting! Increase regularization, get more data.
No gap between train / val means underfitting: train longer, use a bigger model, reduce regularization
Losses may be noisy, use a scatter plot and also plot moving average to see trends better.

Look at learning curves!
Cross-validation

We develop "command centers" to visualize all our models training with different hyperparameters.

check out weights and biases
You can plot all your loss curves for different hyperparameters on a single plot.
Don't look at accuracy or loss curves for too long!
Choosing Hyperparameters

Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Find LR that makes loss go down
Step 4: Coarse grid, train for ~1-5 epochs
Step 5: Refine grid, train longer
Step 6: Look at loss and accuracy curves
Step 7: GOTO step 5
Hyperparameters to play with:
- network architecture
- learning rate, its decay schedule, update type
- regularization (L1/L2/Dropout strength)
Summary

- Improve your training error:
  - Optimizers
  - Learning rate schedules

- Improve your test error:
  - Regularization
  - Choosing Hyperparameters
Summary

**Training** Deep Neural Networks

- Details of the non-linear activation functions
- Data normalization
- Weight Initialization
- Batch Normalization
- Advanced Optimization
- Regularization
- Data Augmentation
- Transfer learning
- Hyperparameter Tuning
Today: Recurrent Neural Networks
“Vanilla” Neural Network

Vanilla Neural Networks
Recurrent Neural Networks: Process Sequences

e.g. Image Captioning
image -> sequence of words
Recurrent Neural Networks: Process Sequences

- e.g. sentiment analysis
  - sequence of words -> sentiment label
Recurrent Neural Networks: Process Sequences

- **One to one**: Direct mapping between two inputs.
- **One to many**: Multiple outputs from a single input.
- **Many to one**: Single output from multiple inputs.
- **Many to many**: Multiple outputs from multiple inputs.

Examples:
- Translation
- Q&A
- Conversation

Sequence of words -> sequence of words
Recurrent Neural Networks: Process Sequences

- One to one
- One to many
- Many to one
- Many to many

Example: Language entity recognition
Why are existing convnets insufficient?

Variable sequence length inputs and outputs!

**Example task:** video captioning

**Input** video can have variable number of frames

**Output** captions can be variable length.

Krishna, Hata, Ren, Fei-Fei, Niebles. Dense captioning Events in Videos. ICCV 2019
Let's start with a setting that takes a variable input and produces an output at every step.

Example: Video activity labeling

Input: video frame; Output: activity label at each frame

Recognizing an activity requires looking at more than one frame!

Want: a model that can make prediction for each frame based on the past frames.

We need a model that can memorize what it has seen so far!
Recurrent Neural Network
Recurrent Neural Network

Key idea: RNNs have an "internal state" that is updated as a sequence is processed. You can think of it as "memory".
Unrolled RNN
Unrolled RNN

The same model!
We can process a sequence of vectors $\mathbf{x}$ by applying a **recurrence formula** at every time step:

$$ h_t = f_W(h_{t-1}, \mathbf{x}_t) $$

new state (vector)  |  old state (vector)  |  input vector at some time step

some model with parameters $W$

Can set initial state $h_0$ to all 0’s
“Read out” the prediction by passing the hidden state through a network (e.g., a few FC layers)

\[
y_t = f_{W_{hy}}(h_t)
\]

output

new state

another model with parameters \( W_{hy} \)

The prediction network is often shared across timestep.
(Simple) Recurrent Neural Network

The state consists of a single “hidden” vector $h$:

$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Sometimes called a “Vanilla RNN” or an “Elman RNN” after Prof. Jeffrey Elman
RNN: Computational Graph

\[ f_W \]

\[ h_0 \rightarrow f_W \rightarrow h_1 \]

\[ x_1 \]
RNN: Computational Graph

\[ h_0 \xrightarrow{f_W} h_1 \xrightarrow{f_W} h_2 \]

\[ x_1 \xleftarrow{W} \quad x_2 \xleftarrow{W} \]
RNN: Computational Graph
RNN: Computational Graph

Re-use the same weight matrix at every time-step
RNN: Computational Graph: Many to Many
RNN: Computational Graph: Many to Many

\[ \begin{align*}
    h_0 & \rightarrow f_W \\
    x_1 & \rightarrow f_W \\
    y_1 & \rightarrow L_1 \\
    y_2 & \rightarrow L_2 \\
    y_3 & \rightarrow L_3 \\
    y_T & \rightarrow L_T \\
    h_1 & \rightarrow f_W \\
    h_2 & \rightarrow f_W \\
    h_3 & \rightarrow f_W \\
    \ldots \\
    h_T & \rightarrow \ldots
\end{align*} \]
RNN: Computational Graph: Many to Many
RNN: Computational Graph: Many to One
RNN: Computational Graph: Many to One

Example: sentence classification
RNN: Computational Graph: One to Many

Example: image captioning
RNN: Computational Graph: One to Many

Example: image captioning
RNN: Computational Graph: One to Many

Example: text generation
Sequence to Sequence: Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector

A vector that “memorizes” the entire sentence

Sutskever et al, “Sequence to Sequence Learning with Neural Networks”, NIPS 2014
Sequence to Sequence: Many-to-one + one-to-many

**Many to one**: Encode input sequence in a single vector

**One to many**: Produce output sequence from single input vector

A vector that “memorizes” the entire sentence

Sutskever et al, “Sequence to Sequence Learning with Neural Networks”, NIPS 2014
Example:
Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: “hello” with one-hot encoding
Example:
Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: “hello” with one-hot encoding

\[ h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \]
Example:
Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: "hello" with one-hot encoding
Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

At test-time sample characters one at a time, feed back to model.
Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

At test-time sample characters one at a time, feed back to model
Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

At test-time sample characters one at a time, feed back to model
Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

At test-time sample characters one at a time, feed back to model.
Backpropagation through time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient.
Backpropagation through time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

A single forward pass of the RNN model.
Backpropagation through time

Loss

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

A single forward pass of the RNN model. $T$ passes create a huge computation graph!
**Truncated Backpropagation through time**

Run forward and backward through chunks (length k) of the sequence instead of the whole sequence, do parameter update, clear gradient cache.
Truncated Backpropagation through time

Carry hidden states forward in time for k steps, backprop, update parameter, clear gradient …
Truncated Backpropagation through time
Multilayer RNNs
Multilayer RNNs

Each RNN layer takes as input (1) previous hidden state from the same layer and (2) the output of the previous layer at the same timestep (or the input).
From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buried thy content,
And tender churl mak'st waste in niggarding:
Pity the world, or else this glutten be,
To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserve thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
This were to be new made when thou art old,
And see thy blood warm when thou feel'st it cold.
at first:

"Tmont thithey" fomesserliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coanigencn Phe lism thond hon at. MeiDimorotion in ther thize."

"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.
PANDARUS:
Alas, I think he shall be come approached and the day
When little strain 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.

VIOLA:
I'll drink it.

VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
0, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
The Stacks Project: open source algebraic geometry textbook

Latex source

http://stacks.math.columbia.edu/
The stacks project is licensed under the GNU Free Documentation License
For $\bigoplus_{i=1}^{m}$ where $L_{m} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $F$ on $X$, $U$ is a closed immersion of $S$, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \text{Spec}(R) = U \times X U \times X U$$

and the comparably in the fibre product covering we have to prove the lemma generated by $\prod Z U \to V$. Consider the maps $M$ along the set of points $\text{Sch}(U)$ and $U \to U$ is the fibre category of $S$ in $U$ in Section, ?? and the fact that any $U$ affine, see Morphisms, Lemma ??, Hence we obtain a scheme $S$ and any open subset $W \subset U$ in $\text{Sh}(G)$ such that $\text{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U _ i \times S ^ i U _ i$$

which has a nonzero morphism we may assume that $f_i$ is of finite presentation over $S$. We claim that $O_{X,S}$ is a scheme where $x \to x', x'' \to S'$ such that $O_{X,S} \to O_{X',S'}$ and separated. By Algebra, Lemma ?? we can define a map of complexes $GL_{S'}(x'/S')$ and we win.

To prove study we see that $F_{iU}$ is a covering of $X'$, and $T_i$ is an object of $F_{X/S}$ for $i > 0$ and $F_i$ exists and let $F_i$ be a presheaf of $O_{X}$-modules on $C$ as a $F$-module. In particular $F = U/F$ we have to show that

$$\overline{M} ^ * = T ^ * \otimes_{\text{Spec}(k)} O_{X,S} \otimes \mathfrak{m} JF$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = (\text{Sch}/S)^{op} \text{ppf}, (\text{Sch}/S)^{ppf}$$

and

$$V = \Gamma(S, O) \to (U, \text{Spec}(A))$$

is an open subset of $X$. Thus $U$ is affine. This is a continuous map of $X$ is the inverse, the groupoid scheme $S$.

Proof. See discussion of sheaves of sets.

The result for prove any open covering follows from the less of Example ??, It may replace $S$ by $X_{spaces, \text{etale}}$ which gives an open subspace of $X$ and $T$ equal to $S_{\text{etale}}$, see Descent, Lemma ??, Namely, by Lemma ?? we see that $R$ is geometrically regular over $S$.

Lemma 0.1. Assume (3) and (3) by the construction in the description. Suppose $X = \lim |X|$ by the formal open covering $X$ and a single map $\text{Proj}_X(A) = \text{Spec}(B)$ over $U$ compatible with the complex

$$\text{Sets}(A) = \Gamma(X, O_{X, A})$$

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ is stable under the following result in the second conditions of (1), and (3). This finishes the proof, By Definition ?? (without element is when the closed subschemes are catenary, if $T$ is surjective we may assume that $T$ is connected with residue fields of $S$. Moreover there exists a closed subspace $Z \subset X$ of $X$ where $U$ in $X'$ is proper (some defining as a closed subset the uniqueness it suffices to check the fact that the following theorem

1) $f$ is locally of finite type. Since $S = \text{Spec}(R)$ and $Y = \text{Spec}(R)$.

Proof. This is form all sheaves of sheaves on $X$. But given a scheme $U$ and a surjective étale morphism $U \to X$. Let $U \cap U = \bigcup_{i=1}^{m} U _ i$ be the scheme $X$ over $S$ at the schemes $X_i \to X$ and $U = \lim U_i$.

The following lemma surjective rest decomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{X_{x_0}, U_0}$.

Lemma 0.2. Let $X$ be a locally Noetherian scheme over $S$, $E = F_{X/S}$. Set $T = J_1 \subset Z$. Since $\mathcal{T} \subset \mathcal{T}$ are nonzero over $i_0 \leq p$ is a subset of $J_{x_0} \circ \lambda_2$ works.

Lemma 0.3. In Situation ??, Hence we may assume $q' = 0$.

Proof. We will use the property we see that $p$ is the next functor (?). On the other hand, by Lemma ?? we see that

$$D(O_{X'}) = O_X(D)$$

where $K$ is an $F$-algebra where $\delta_{n+1}$ is a scheme over $S$. 

$\square$
Proof. Omitted.

Lemma 0.1. Let $C$ be a set of the construction.

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

\[ O_{\mathcal{O}_X} = O_X(L) \]

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

\[ O_X(\mathcal{F}) = \{ \text{morph}_{1} \times_{O_X} (\mathcal{G}, \mathcal{F}) \} \]

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

\[ Q \]

Lemma 0.2. This is an integer $Z$ is injective.

Proof. See Spaces, Lemma 22.

\[ Q \]

Lemma 0.3. Let $S$ be a scheme. Let $X$ be a scheme and $X$ is an affine open covering. Let $U \subset X$ 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 $O_X$-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 $O_X(U)$ which is locally of finite type.

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

\[ \begin{array}{c}
S \\
| \\
\xi \\
\downarrow \\
\mathcal{O}_{X'} \\
\downarrow \\
\mathcal{O}_X \\
\end{array} \]

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 $\mathcal{F}$. 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 seen that $X = \text{Spec}(R)$ 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 22.

A reduced above we conclude that $U$ is an open covering of $C$. The functor $\mathcal{F}$ is a "field"

\[ O_{X, x} \to \mathcal{F} \to \mathcal{O}_{\text{étale}} \to O_X, O_{X, x} \]

is an isomorphism of covering of $O_X$. 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 22 and we can filtered set of presentations of a scheme $O_X$-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 $O_{X, x}$ is a closed immersion, see Lemma 22. This is a sequence of $\mathcal{F}$ is a similar morphism.
static void do_command(struct seq_file *m, void *v)
{
    int column = 32 << (cmd[2] & 0x80);
    if (state)
        cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
    else
        seq = 1;
    for (i = 0; i < 16; i++) {
        if (k & (1 << i))
            if (pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000ffffffff) & 0x0000000f) << 8;
                if (count == 0)
                    sub(pid, ppc_md.kexec_handle, 0x20000000);
                    pipe_set_bytes(i, 0);
                }
            /* Free our user pages pointer to place camera if all dash */
                subsystem_info = &of_changes[PAGE_SIZE];
                rek_controls(offset, idx, &soffset);
                /* Now we want to deliberately put it to device */
                    control_check_polarity(&context, val, 0);
                    for (i = 0; i < COUNTER; i++)
                        seq_puts(s, "policy ");
        }
/*
 * Copyright (c) 2006-2010, Intel Mobile Communications. All rights reserved.
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 as published by
 * the Free Software Foundation.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 */

#include <linux/kexec.h>
#include <linux/errno.h>
#include <linux/io.h>
#include <linux/platform_device.h>
#include <linux/multi.h>
#include <linux/ckevent.h>

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>
#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

#define REG_PG vesa_slot_addr_pack
#define PFM_NOCOMP AFSR(0, load)
#define STACK_DDR(type) (func)

#define SWAP_ALLOCATE(nr) (e)
#define emulate_sigs() arch_get_unaligned_child()
#define access_rw(TST) asm volatile("movd %esp, %0, %3" : : "r" (0)); \n     if ((__type & DO_READ)

static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \n   pC>[1]);

static void
os_prefix(unsigned long sys)
{
    #ifdef CONFIG_PREEMPT
       PUT_PARAM_RAID(2, sel) = get_state_state();
       set_pid_sum((unsigned long)state, current_state_str(),
                (unsigned long)-1->lr_full; low;
    
}
Image Captioning

Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
Show and Tell: A Neural Image Caption Generator, Vinyals et al.
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick
Recurrent Neural Network

Convolutional Neural Network
test image
before:
\[ h = \tanh(W_{xh} \ast x + W_{hh} \ast h) \]

now:
\[ h = \tanh(W_{xh} \ast x + W_{hh} \ast h + W_{ih} \ast v) \]
test image
test image

sample

<END> token

=> finish.
A cat sitting on a suitcase on the floor
A cat is sitting on a tree branch
A dog is running in the grass with a frisbee
A white teddy bear sitting in the grass

Two people walking on the beach with surfboards
A tennis player in action on the court
Two giraffes standing in a grassy field
A man riding a dirt bike on a dirt track
Image Captioning: Failure Cases

A woman is holding a cat in her hand

A woman standing on a beach holding a surfboard

A person holding a computer mouse on a desk

A bird is perched on a tree branch

A man in a baseball uniform throwing a ball
Visual Question Answering (VQA)

Figure from Zhu et al, copyright IEEE 2016. Reproduced for educational purposes.
Visual Dialog: Conversations about images

Das et al., "Visual Dialog", CVPR 2017
Figures from Das et al, copyright IEEE 2017. Reproduced with permission.
Agent encodes instructions in language and uses an RNN to generate a series of movements as the visual input changes after each move.

**Instruction**

Turn right and head towards the *kitchen*. Then turn left, pass a *table* and enter the *hallway*. Walk down the hallway and turn into the *entry way* to your right *without doors*. Stop in front of the *toilet*.

---


Figures from Wang et al, copyright IEEE 2017. Reproduced with permission.
RNN tradeoffs

RNN Advantages:
- Can process any length input
- Computation for step $t$ can (in theory) use information from many steps back
- Model size doesn’t increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

RNN Disadvantages:
- Recurrent computation is slow
- In practice, difficult to access information from many steps back
- **Vanishing gradient / gradient explosion**
Vanilla RNN Gradient Flow

\[ h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t) \]

\[ = \tanh \left( \begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \]

\[ = \tanh \left( W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \]

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Vanilla RNN Gradient Flow

Backpropagation from $h_t$ to $h_{t-1}$ multiplies by $W$ (actually $W_{hh}^T$)

$$
\begin{align*}
    h_t &= \tanh(W_{hh} h_{t-1} + W_{xh} x_t) \\
    &= \tanh \left( (W_{hh} \quad W_{hx}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\
    &= \tanh \left( W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)
\end{align*}
$$

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Vanilla RNN Gradient Flow

Backpropagation from $h_t$ to $h_{t-1}$ multiplies by $W$ (actually $W_{hh}^T$)

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$
$$= \tanh \left( (W_{hh} \ W_{hx}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$
$$= \tanh \left( W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh} h_{t-1} + W_{xh} x_t) W_{hh}$$

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Vanilla RNN Gradient Flow

\[ \frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W} \]

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Vanilla RNN Gradient Flow

Gradients over multiple time steps:

\[ \frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W} \]

\[ \frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} \]

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Vanilla RNN Gradient Flow

Gradients over multiple time steps:

\[ \frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W} \]

\[ \frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_T}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left( \prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_1}{\partial W} \]
Vanilla RNN Gradient Flow

Gradients over multiple time steps:

\[
\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}
\]

\[
\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_T}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left( \prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_1}{\partial W}
\]

\[
\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh} h_{t-1} + W_{xh} x_t) W_{hh}
\]

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Vanilla RNN Gradient Flow

Gradients over multiple time steps:

\[ \frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W} \]

\[ \frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \left( \prod_{t=2}^{T} \tanh' \left( W_{hh} h_{t-1} + W_{xh} x_t \right) \right) W_h^{T-1} \frac{\partial h_1}{\partial W} \]

Always < 1

Vanishing gradients

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Vanilla RNN Gradient Flow

Gradients over multiple time steps:

\[
\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}
\]

What if we assumed no non-linearity?

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Vanilla RNN Gradient Flow

Gradients over multiple time steps:

What if we assumed no non-linearity?

\[
\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}
\]

Largest eigen value > 1: Exploding gradients

\[
\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} W^{T-1} \frac{\partial h_1}{\partial W}
\]

Largest eigen value < 1: Vanishing gradients

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Vanilla RNN Gradient Flow

Gradients over multiple time steps:

What if we assumed no non-linearity?

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

Largest eigen value $> 1$: Exploding gradients

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} W^{T-1} \frac{\partial h_1}{\partial W}$$

Largest eigen value $< 1$: Vanishing gradients

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994

Gradient clipping:
Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
    grad *= (threshold / grad_norm)
```
Vanilla RNN Gradient Flow

Gradients over multiple time steps:

What if we assumed no non-linearity?

$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$

$Largest$ $eigen$ $value > 1$: Exploding gradients

$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} W_{T-1}^{T} \frac{\partial h_1}{\partial W}$

$Largest$ $eigen$ $value < 1$: Vanishing gradients

We need a new RNN architecture!

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Long Short Term Memory (LSTM)

Vanilla RNN

\[ h_t = \tanh \left( W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \]

LSTM

\[
\begin{pmatrix}
    i \\
    f \\
    o \\
    g
\end{pmatrix} =
\begin{pmatrix}
    \sigma \\
    \sigma \\
    \sigma \\
    \tanh
\end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}
\]

\[
c_t = f \odot c_{t-1} + i \odot g
\]

\[
h_t = o \odot \tanh(c_t)
\]

Learn to control information flow from previous state to the next state

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997
Long Short Term Memory (LSTM)

Vanilla RNN

\[ h_t = \tanh \left( W \left( h_{t-1}, x_t \right) \right) \]

Long-term memory \( c \) determines how much information should go into the hidden state \( h \) (short-term memory)

LSTM

\[
\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}
\]

\[ c_t = f \odot c_{t-1} + i \odot g \]

\[ h_t = o \odot \tanh(c_t) \]

Two “memory vectors”

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997
Long Short Term Memory (LSTM)  
[Hochreiter et al., 1997]
Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

- **Input gate (i)**: whether to write to cell memory from before ($h_{t-1}$)
- **Forget gate (f)**: whether to erase cell memory from below ($x$)
- **Output gate (o)**: how much to reveal cell memory from before

$$
\begin{align*}
(i) & = \sigma(x) \\
(f) & = \sigma(x) \\
(o) & = \sigma(x) \\
(g) & = \tanh(x)
\end{align*}
$$

$$
\begin{align*}
W & = \begin{pmatrix}
\sigma & \sigma \\
\sigma & \tanh
\end{pmatrix}
\end{align*}
$$

$$
\begin{align*}
c_t &= f \odot c_{t-1} + i \odot g \\
h_t &= o \odot \tanh(c_t)
\end{align*}
$$
Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]

- **i**: Input gate, whether to write to cell
- **f**: Forget gate, whether to erase cell
- **o**: Output gate, how much to reveal cell
- **g**: Gate gate (?), what to write to cell

\[
\begin{bmatrix}
i \\
f \\
o \\
g
\end{bmatrix} = \begin{bmatrix}
sigmoid \\
sigmoid \\
sigmoid \\
tanh
\end{bmatrix} W \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}
\]

\[
c_t = f \odot c_{t-1} + i \odot g
\]

\[
h_t = o \odot \tanh(c_t)
\]
Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]

- **i:** Input gate, whether to write to cell
- **f:** Forget gate, whether to erase cell
- **g:** Gate gate (.), what to write to cell

4h x 2h

**Input from below (x)**

**Memory from before (h)**

```
(i) = \begin{pmatrix}
\sigma \\
\sigma \\
\sigma \\
tanh
\end{pmatrix}
W \left( h_{t-1} \right)
```

- \( c_t = f \odot c_{t-1} + i \odot g \)
- \( h_t = o \odot \tanh(c_t) \)
Long Short Term Memory (LSTM)  
[Hochreiter et al., 1997]

i: Input gate, whether to write to cell  
f: Forget gate, whether to erase cell  
o: Output gate, how much to reveal cell  
g: Gate gate (?), what to write to cell

$$c_t = f \odot c_{t-1} + i \odot g$$  
$$h_t = o \odot \tanh(c_t)$$
Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]

\[
\begin{align*}
\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} &= \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \\
ct &= f \odot ct_{-1} + i \odot g \\
h_t &= o \odot \tanh(ct)
\end{align*}
\]
Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

Backpropagation from $c_t$ to $c_{t-1}$ only elementwise multiplication by $f$ (forget gate), no matrix multiply by $W$

\[
\begin{pmatrix}
i \\ f \\ g \\ o
\end{pmatrix} = \begin{pmatrix}
\sigma & \\ \sigma & \\ \sigma & \\ \tanh
\end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}
\]

\[
c_t = f \odot c_{t-1} + i \odot g
\]

\[
h_t = o \odot \tanh(c_t)
\]
Long Short Term Memory (LSTM): Gradient Flow
[Hochreiter et al., 1997]

Uninterrupted gradient flow!

Notice that the gradient contains the $f$ gate’s vector of activations
- allows better control of gradients values, using suitable parameter updates of the forget gate.
The hidden state is emitted from $c$ with an output gate ($o$), instead of recurrent multiplication with a weight vector.
Do LSTMs solve the vanishing gradient problem?

The LSTM architecture makes it easier for the RNN to preserve information over many timesteps

- e.g. **if the** $f = 1$ **and the** $i = 0$, then the information of that cell is preserved indefinitely.
- By contrast, it’s harder for vanilla RNN to learn a recurrent weight matrix $W_h$ that preserves info in hidden state

LSTM **doesn’t guarantee** that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies.

It is possible to mitigate vanishing / exploding gradient by learning the correct $i$ and $f$. 
Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

Uninterrupted gradient flow!

Similar to ResNet!
Neural Architecture Search for RNN architectures

Zoph et Le, "Neural Architecture Search with Reinforcement Learning", ICLR 2017
Other RNN Variants

**GRU** [Learning phrase representations using rnn encoder-decoder for statistical machine translation, Cho et al. 2014]

\[
\begin{align*}
    r_t &= \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \\
    z_t &= \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \\
    \tilde{h}_t &= \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \\
    h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t
\end{align*}
\]

Simpler than LSTM, control information flow without cell state.

[An Empirical Exploration of Recurrent Network Architectures, Jozefowicz et al., 2015]

\[
\begin{align*}
    z &= \text{sigm}(W_{xx}x_t + b_x) \\
    r &= \text{sigm}(W_{xr}x_t + W_{hr}h_{t-1} + b_t) \\
    h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + \tanh(x_t) + b_h) \odot z + h_t \odot (1 - z) \\
    h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t
\end{align*}
\]

[LSTM: A Search Space Odyssey, Greff et al., 2015]
Recommendations

- If you want to use RNN-like models, try LSTM
- Use variants like GRU if you want faster compute and less parameters
- Try transformers (next lecture) as they are dominating sequencing modeling
- New variants of RNNs are still active research topic. Example: RWKV (“Transformer-level performance but with RNN”)
Summary

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don’t work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research, as well as new paradigms for reasoning over sequences.