Recurrent Neural Networks for Energy Forecasting

Nikil Pancha
Neural Networks

Universal function approximators
Learnable weights
Non-convex
Trainable by stochastic gradient descent and backpropagation
Regularization

Weight norm regularization: penalty on norm of weights (usually $L_{1,1}$ or $L_{2,2}$)

Batch normalization: after each layer, normalize the outputs (subtract batch mean and divide by standard deviation). Learn scale and offset to be applied after normalization

Dropout: Randomly set a proportion of nodes to output 0 for a particular batch.
Dropout

Usually drop 20% to 50% of nodes

Acts as a method of model averaging
Convolutional NNs

Learn weights of filters to apply to inputs

Widely used in image processing

Require fewer learnable parameters (local connectivity)

Inspired by how cats and monkeys see
Recurrent Neural Networks

Used on sequential data

Trained by backpropagation through time (BPTT)

RNN uses:
- Audio
- Text
- Images
- Time series
http://colah.github.io/posts/2015-08-Understanding-LSTMs/
**LSTM**

Allows long term dependencies

Adds several gates to control information flow:
- Forget
- Input
- Output
Unrolling RNNs
Encoder

Decoder

\[ \sum \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]

\[ y \]

\[ \hat{y} \]
Tools

Tensorflow

Keras

SciPy/NumPy/Pandas/Matplotlib
/*
 * 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.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software Foundation,
 * Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#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>

#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)); \
     if (__type & DO_READ)

static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
     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;
    }

http://karpathy.github.io/2015/05/21/rnn-effectiveness
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 $\mathcal{O}$-modules. We have to show that

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

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

$$\mathcal{O}_X(\mathcal{F}) = \{\text{morph}_{1} \times \mathcal{O}_X(\mathcal{G}, \mathcal{F})\}$$

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

**Lemma 0.2.** This is an integer $Z$ 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 $U \subseteq X$ 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

$$b: X \rightarrow Y' \rightarrow Y \rightarrow Y \rightarrow X' \times_X Y \rightarrow X$$

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

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 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 ??.

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

$$\mathcal{O}_{X,t} \rightarrow F_t = \mathcal{O}_{X,t} \left(\mathcal{O}_{X,t}^{\text{flat}}\right) \rightarrow \mathcal{O}_{X,t}^\text{flat} \mathcal{O}_{X,t}(\mathcal{O}_{X,t}^{\text{flat}})$$

is an isomorphism of covering of $\mathcal{O}_{X,t}$. If $\mathcal{F}$ is the unique element of $F$ 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 $\mathcal{O}_{X,t}$-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 $\mathcal{O}_{X,t}$ is a closed immersion, see Lemma ?? This is a sequence of $\mathcal{F}$ is a similar morphism.
Input Variables

Electric: cos(hour), cos(dayofweek), cos(month), temperature, wind speed, cloud cover, is_holiday

Gas: cos(dayofweek), cos(month), cos(dayofmonth), temperature, wind speed, is_holiday
Conclusion / Future work

Autoregressive networks work better when there is less training data available

Try new methods:
- Attention is All You Need (Google Brain)
- Convolutional NNs
- Echo state networks
Backpropagation

\[ h_{11} = \sigma(v_{11}), \text{ where } v_{11} = w_1 x_1 + w_3 x_2. \]

\[ \sigma'(u) = \sigma(u)(1-\sigma(u)). \]
References


