LA-UR- 08-7430

Approved for public release; distribution is unlimited.

Title:

LANL* V1.0: A Radiation Belt Drift Shell Model Suitable for Real-Time and Reanalysis Applications

Author(s):

Josef Koller
Geoffrey D. Reeves
Reiner H. W. Friedel

Intended for:

Journal of Geoscientific Model Development

Los Alamos
NATIONAL LABORATORY
EST. 1943

Manuscript prepared for Geosci. Model Dev.
with version 2.3 of the LaTeX class copernicus.cls.
Date: 13 November 2008

LANL* V1.0: A Radiation Belt Drift Shell Model Suitable for Real-Time and Reanalysis Applications

J. Koller[1], G. D. Reeves[1], and R. H. W. Friedel[1]

[1]Space Science and Applications, ISR-1, Los Alamos National Lab

Correspondence to: J. Koller
(jkoller@lanl.gov)

Abstract. Space weather modeling, forecasts, and predictions, especially for the radiation belts in the inner magnetosphere, require detailed information about the Earth's magnetic field. Results depend on the magnetic field model and the L^* (pron. L-star) values which are used to describe particle drift shells. Space weather models require integrating particle motions along trajectories that encircle the Earth. Numerical integration typically takes on the order of 10^5 calls to a magnetic field model which makes the L^* calculations very slow, in particular when using a dynamic and more accurate magnetic field model. Researchers currently tend to pick simplistic models over more accurate ones but also risking large inaccuracies and even wrong conclusions. For example, magnetic field models affect the calculation of electron phase space density by applying adiabatic invariants including the drift shell value L^*. We present here a new method using a surrogate model based on a neural network technique to replace the time consuming L^* calculations made with modern magnetic field models. The advantage of surrogate models (or meta-models) is that they can compute the same output in a fraction of the time while adding only a marginal error. Our drift shell model LANL* (Los Alamos National Lab L-star) is based on L^* calculation using the TSK03 model (Tsyganenko et al., 2003). The surrogate model has currently been tested and validated only for geosynchronous regions but the method is generally applicable to any satellite orbit. Computations with the new model are several million times faster compared to the standard integration method while adding less than 1% error. Currently, real-time applications for forecasting and even nowcasting inner magnetospheric space weather is limited partly due to the long computing time of accurate L^* values. Without them, real-time applications are limited in accuracy. Reanalysis application of past conditions in the inner magnetosphere are used to understand physical processes and their effect. Without sufficiently accurate L^* values, the interpretation of reanalysis results becomes difficult and uncertain. However, with a method that can calculate accurate L^* values orders of magnitude faster, analyzing whole solar

cycles worth of data suddenly becomes feasible.

1 Introduction

The Earth's magnetic field is an important ingredient in space weather modeling and forecasting. The field can be very dynamic and non-symmetric depending on the solar wind conditions outside of the Earth's magnetosphere. Results of space weather models and forecasts, especially for the inner magnetosphere, depend on the magnetic field model and the L^* values which are used to describe particle drift shells. L^* is a simple function of the magnetic flux, also known as the third adiabatic invariant Φ, the dipole moment k_0, and the Earth's radius R_E

$$L^* = -\frac{2\pi k_0}{\Phi R_E} \tag{1}$$

Physically, L^* is the radial distance from the Earth's center to the equatorial points of the symmetric particle drift shell, if all non-dipolar moments are turned off adiabatically.Particle motion, described by adiabatic invariants, depend on the magnetic field $\boldsymbol{B}$, the particle's energy E, and the pitch angle α. In a dipole field the invariants are analytical but a realistic field requires numerical integration

$$\Phi = \int \boldsymbol{B} \cdot d\boldsymbol{S} \tag{2}$$

where $\boldsymbol{B}$ is the magnetic field vector and $\boldsymbol{S}$ denotes the enclosed surface.

In an "adiabatic" system, phase space density is conserved as a function of L^* but not as a function of physical position. Therefore it is critical to pu spacecraft observations in the correct "magnetic coordinates" before realistic physical modeling can be done.

A number of empirical magnetic field models exist. However, it can take a long time to calculate L^* using more sophisticated models (McCollough et al., 2008) because full shell tracing in a complex magnetic field is computationally very expensive. Typical integration requires on the order of 10^5 calls to the magnetic field model for obtaining the magnetic field vector. The resulting long computation times cause researchers to trade more accurate model with faster, but simpler ones but also risking large inaccuracies and even wrong conclusions. Huang et al. (2008) recently quantified the effect of choosing a magnetic field model for radiation belt studies and concluded that global inaccuracies of magnetic field models could alter the results of the inferred radial profiles of phase space densities of radiation belt electrons. A field model affects the calculation of electron phase space density by applying adiabatic invariants (Green and Kivelson, 2004). Huang et al. (2008) found that during storm times L^* can vary by as much as 50% (Huang, priv. communications). As part of the DREAM project (Dynamic Radiation Environment Assimilation Model), Chen et al. (2007) studied the effect of choosing a magnetic field model on the phase space density calculation and found that an accurate magnetic field model is critical to radiation belt modeling.

In this paper we present a new method of calculating L^* based on a recent magnetic field model (Tsyganenko, 2002a,b; Tsyganenko et al., 2003), here TSK03 formerly also known as T01-storm.

The method is based on a forward feed neural network that has been trained on TSK03 calculations for all occuring solar wind conditions during the year 2002. Currently the neural network is trained only with geosynchronous regions but we will expand it in a future version to all regions of the inner magnetosphere. With a trained neural network, the TSK03 model can be replaced by the much faster surrogate model. This approach is also known as meta-modeling (see Kleijnen, 2008, for details). The method applies equally to any magnetic field model or arbitrary complexity - statistical, empirical, physics-based like magneto-hydrodynamic models, etc.

In the following section we will describe surrogate models in general and in Section 3 how neural networks can be used as such surrogate models. Section 4 describes the underlying magnetic field model used for creating the neural network and Section 5 how the network was trained. We validated and tested the neural network as explained in Section 6 and show how to use the provided code (supplemental material) in Section 7. We summarize and conclude with Section 8.

2 Surrogate Models

Surrogate models (or meta-models) can replace a complicated non-linear input-output relationship while adding only a minimal error. Other scientific fields use them frequently for studying the sensitivity of complex models on input parameters [references!]. Surrogate models are trained with input-output data from the original model. Once the training is successfully completed, the surrogate can replace the complex model and compute a sufficiently accurate output in a fraction of the time.

Surrogate models do not contain details of the physical processes or geometries but only focus on the input-output relationship. The results from such surrogate models are not exact but sufficiently close to the physics-based model. Different methods can be used to create surrogate models: The simplest surrogate models are based on polynomial regression. Others are based on Kriging, Gaussian process modeling, and neural networks (Kleijnen, 2008; Myers and Montgomery, 2002). We chose to use a forward feed neural network to create a surrogate model for TSK03.

Surrogate models are by definition fast to compute but do not necessarily represent the original model exactly. The goal of a surrogate model is to replace a very complex complex model with a fast computing model that can deliver results that are sufficiently close to the values calculated directly with the original model. The drift shell model presented here is able to calculate L^* with less than 1% error compared to the original model and over a million times faster. Figure 1 exemplifies a diagram of an artificial neural network use for our study.

3 Forward Feed Neural Networks

Artificial neural networks are loosely related to neurons in our nervous system in the sense that they represent a non-linear mapping from input to output signals (Bishop, 1995; Reed and Marks, 1999). In general, an artificial neural network consists of a number of non-linear processing units

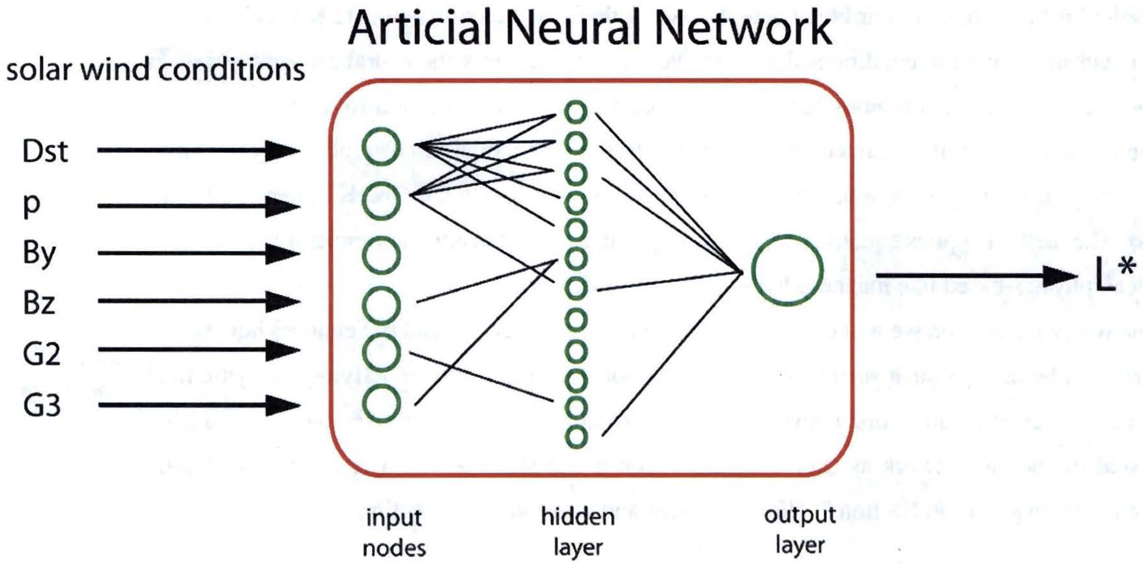

Fig. 1. Diagram for a layered forward feed neural network. Solar wind conditions are used as input for predicting L^* values. All nodes have a connection to every node from the previous layer but are not drawn here for simplicity. Also, not all possible parameters that can be used as input for the artificial neural network are shown. Specifically, our drift shell model includes additional values for Kp, solar wind density, velocity, and magnetic coordinates.

that are interconnected through weighted communication lines. The units are called neurons and receive input signals from a number of other nodes and produce a single scalar output which then can be used as input to other neurons via weighted connections. Feed-forward networks do not allow connections from the output back to the input either direct or indirect.

Neural networks are usually organized in several layers. Such a network is also called a multilayer perceptron. The first layer provides a node for each input element (see Figure 1). In our case the input layer consists of 16 nodes, one for each input parameter for the TSK03 model plus additional nodes for parameters that help to further specify the system (like geomagnetic coordinates). The hidden layer contains 20 neurons that are connected to each input node and one output node to produce L^* for a specified pitch angle.

The number of neurons in the hidden layer is somewhat arbitrary and usually has to be determined through testing. However, too many neurons in the hidden layer can cause the artificial neural network to simply memorize patterns. In such a case the network will not be able to perform with other data. Barron (1991, 1993, 1994) completed a study on how the error of a neural network output scales with the number of training samples and hidden nodes. He found that the error decreases like $O(1/\sqrt{N})$ as the number of training samples N increases. The error also decreases as a function of the number of hidden nodes M like $O(1/M)$. In general, it has been shown, by e.g. Cybenko (1989), that a sufficiently large network is able to approximate any function with arbitrary accuracy

Table 1. Input parameters for the neural network LANL*

Number	Parameter	Description	Input to TSK03
1	Year	Integer number representing the year	Yes
2	DOY	Day of the year	Yes
3	UT	Universal Time in units of hours	Yes
4	Kp	Kp index	No
5	Dst	Dynamic storm time index [nT]	Yes
6	n	Solar wind density [cm^{-3}]	No
7	v	Solar wind velocity [km/s]	No
8	p	Solar wind dynamic pressure [nPA]	Yes
9	B_y	Y component of the IMF field [nT]	Yes
10	B_z	Z component of the IMF field [nT]	Yes
11	G1	G1 value (Tsyganenko, 2002b)	No
12	G2	G2 value (Tsyganenko, 2002b)	Yes
13	G3	G3 value (Tsyganenko et al., 2003)	Yes
14	Lm	McIllwain value (Roederer, 1970)	No
15	MLT	magnetic local time	Yes
16	MLAT	magnetic latitude	No

(Bishop, 1995; Reed and Marks, 1999).

Similar to the real nervous system, artificial neural networks have to be trained by learning from examples. Given a set of input parameters and desired outputs, algorithms like the popular "back propagation" algorithm (Rumelhart et al., 1986) can automatically adjust the weights of the interconnections to produce the desired outputs. If the training is successful, then new input can be provided to the neural network and a correct (within a specified error) output is obtained.

Once the training of a neural network is completed, the output can be easily calculated given any set of input values. If $\boldsymbol{x}$ is the input vector, then the output vector $\boldsymbol{y}$ in a 1-hidden-layer architecture is

$$\boldsymbol{y} = f^1\left(\boldsymbol{W}^1 f^0\left(\boldsymbol{W}^0\boldsymbol{x} + \boldsymbol{b}^0\right) + \boldsymbol{b}^1\right) \tag{3}$$

where the matrices $\boldsymbol{W}^{0,1}$, $\boldsymbol{b}^{0,1}$ denote the weight matrices of the hidden and output layer and a bias vector $\boldsymbol{b}$. The bias vector is necessary to obtain a better classification but is, typically, absorbed into the weight vector assuming that one of the inputs is constant (bias node).

The function f is a non-linear squashing function applied to each component of a vector, for example

$$f(x_i) = \frac{1}{1+e^{-x_i}}. \tag{4}$$

Squashing functions are used to limit very large positive or negative values. Sigmoid and tanh functions are common choices.

The weight matrices are determined during training using an optimization algorithm which minimizes a chosen error function. For example, the mean-squared error is commonly used

$$E = \frac{1}{PN} \sum_{p} \sum_{i} (d_{pi} - y_{pi})^2 \tag{5}$$

where p indexes the patterns in the training set, i indexes the output nodes, and d_{pi} and y_{pi} are, respectively, the target and actual network output for the i-th output node on the p-th pattern. P and N are the number of training patterns and network outputs (Reed and Marks, 1999).

Because neural networks have such a redundant parallel structure, they exhibit a certain fault tolerance to some degree. Many nodes draw information from a number of other nodes to produce one overall output. This makes the system relatively insensitive to minor damage. The loss of some input degrades the system but does not necessarily lead to complete failure because the functions are distributed over several nodes instead of an isolated single location. This property has been called "graceful degradation" (Reed and Marks, 1999). Examples for magnetic field models include Kp, Dst, solar wind velocity v_{sw} and other input functions that are correlated among each other.

When neural networks are used as function approximators, they are typically used for interpolation and not extrapolation because the fit is usually good near the training data but poor elsewhere. This aspect of prediction accuracy is also called "generalization". The distribution of training data and network complexity play an important role in the overall performance of the neural network. A poor set of training data may contain misleading regularities (Bishop, 1995; Reed and Marks, 1999). The best choice is to randomly select training data following the same probability distribution that also governs future data.

Neural networks are not new to space physics and especially space weather modeling. They have been used before to predict the relativistic electron flux at geosynchronous orbit (Koons and Gorney, 1991), to forecast geomagnetic induced currents (Lundstedt, 1992), or to analyze solar wind data (Dolenko et al., 2001). To our knowledge, neural networks have not been used as surrogate models replacing complex space physics models. Our approach is different since we use a neural network for predicting integral values of a known statistical magnetic field model instead of an unknown combination of physical processes.

4 The Tsyganenko 2003 Model

The magnetic field model TSK03 (Tsyganenko et al., 2003) is just one out of series of models published by Tsyganenko and colleagues. It is one of the most accurate models currently available (Chen et al., 2007). It accounts for external contributions from the magnetotail current sheet, ring current, magnetopause current and Birkeland current (McCollough et al., 2008). It also includes partial ring current with field-aligned closure currents which allows it to account for local time asymmetries of the inner magnetospheric field. These currents are driven by separate variables calculated as a time integral for a combination of geoeffective parameters of solar wind density, speed, and the magnitude of the southward component of the interplanetary magnetic field (IMF).

The TSK03 model is occasionally referred to as the T01-storm model (Chen et al., 2007) with the same parameterization as Tsyganenko (2002a,b) but with a specific storm time data set. The performance of several different magnetic field models has been recently studied by Huang et al. (2008); McCollough et al. (2008); Chen et al. (2007). Based on these studies and the long computing time issue illustrated by McCollough et al. (2008), specifically when calculating L^* with more recent models e.g. (Tsyganenko and Sitnov, 2005), has lead us to choose TSK03 as an illustrativeexample to demonstrate the LANL* neural network. McCollough et al. (2008) reports a 2.5 hour computing time with TSK03 for 1440 L^* calculations.

We used the ONERA-DESP library V4.1 (Boscher et al., 2007) implementation of the magnetic field model TSK03 (option 10) which has no upper or lower limit on the input values. The model uses time, Dst, solar wind pressure, and the y and z components of the IMF magnetic field. It also includes two parameters G_2 and G_3 representing the time-integrated driving effect of the solar wind on the magnetosphere (McCollough et al., 2008).

5 Training the Network

In order to create the training data, we have constructed an optimized algorithm that can compute a large number of L^* in a short period of time on a high performance cluster (HPC) at Los Alamos National Lab. Our parallelized code can compute half a million L^* values typically within 45 hours compared to 900 hours on a recent single CPU desktop machine.

The generalization performance of the neural network, which is how efficiently it can predict in untrained domains, strongly depends on the training data. Best results are obtained by randomly distributing the input-output training patterns. This prevents the system from simply memorizing patterns in the input-output relations. In order to test the neural network methology we chose to train it for a coordinate torus with the following bounds: $r \in [6.6R_E, 6.7R_E]$, $\phi \in [-180°, +180°]$, $\theta \in [-6°, 6°]$ in spherical geographic coordinates. We randomly picked 10 locations inside this coordinate torus to calculate L^* for every hour in the year 2002 using full numerical integration of the TSK03 model (Fig. 2). This resulted in 87,600 input-output patterns that we used to train the neural network. The input data for Kp, Dst, solar wind density, pressure, velocity, y and z components of the IMF magnetic field were taken from the omni2 data set provided by NASA via omniweb (http://omniweb.gsfc.nasa.gov/).

Typically, the locations inside the coordinate training torus are on closed drift shells. L^* is only defined when the integral is closed. However, during storm conditions the magnetosphere can be

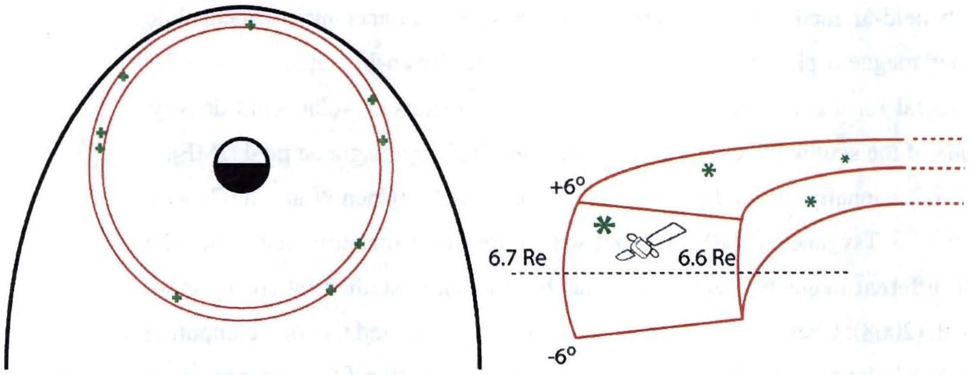

Fig. 2. (left) Coordinate range for creating the training data set (left: top view; right: side view)

compressed by the solar wind and the drift shells move outward due to adiabatic effects and end up as open drift shells. During the main phase of a storm the increase in ring current causes a decrease in the magnetic field strength in the inner magnetosphere and a reduction of the magnetic flux enclosed by an electron drift orbit (Kim and Chan, 1997; Roederer, 1970). This effect requires two separate neural networks, one that can tell us the maximum L^* value (NN-1) that is possible in a given magnetic field configuration and a second one (NN-2) that will actually provide us with the L^* value for the particle pitch angle and spacecraft location.

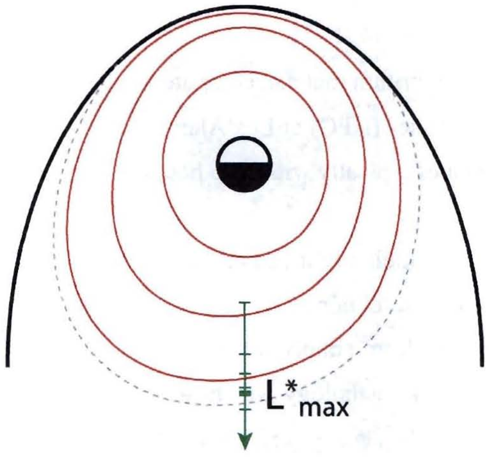

Fig. 3. Diagram of finding the last closed drift shell by using a leap-frog method along the radial direction at midnight local time. The dashed line represents the last closed drift shell with $L*_{\mathrm{max}}$.

We trained the first neural network NN-1 with L^*_{max} values calculated from the full integration of the TSK03 magnetic field model. We have devised a leap-frog method that can efficiently determine the last valid closed drift shell by calculating L^* along the radial coordinate at midnight local time (Fig. 3). Solar wind data including Dst and Kp were used as input and the obtained L^*_{max} values were used as target for training the network.

We trained the second neural network NN-2 with the L^* values provided by the magnetic field model. The input vector patterns are as described above but also include geomagnetic coordinate locations to better define the problem. Adding these coordinates drastically increased the performance of the neural network because they describe the location of the spacecraft as a direct function of the asymmetric magnetic field. In addition, we calculated L^* for several pitch angles between $\alpha \in [10°, 90°]$. Since the results from NN-1 and NN-2 depend on the pitch angle, it was necessary to create several neural networks for a range of pitch angles.

The setup of neural networks is displayed in Fig. 4. Each set consists of several neural networks for different pitch angles. One set is for calculating the last closed drift shell L^*_{max} and the second set is for calculating the actual L^* value. We have also added several more paramters than the ones actually required by TSK03 (see Table 1). We found that these additional values, including Kp, solar wind density, velocity, G1, and especially magnetic coordinates (MacIllwain L, magnetic longitude and latitude) dramatically increase the generalization properties of the neural network.

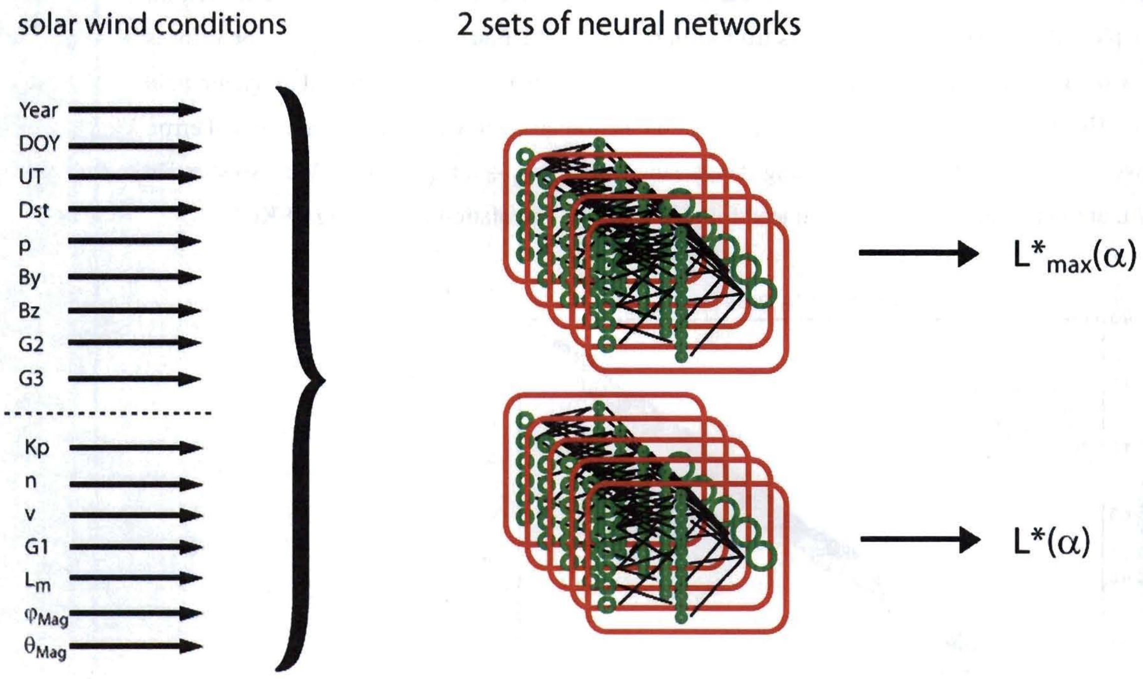

Fig. 4. Set of neural networks that can calculate L^* as a function of pitch angle. Each set consists of several neural networks for a range of pitch angles. One set calculates L^* and the other set computes the last closed drift shell L^*_{max}.

We used the python module ffnet (Wojciechowski, 2007) to train our neural networks with optimization algorithm provided in the ffnet package. The ffnet python module has a functionality that allows exporting the trained neural network into a FORTRAN subroutine which enables us to share the neural network efficiently.

In addition, the neural network can calculate L^* in a fraction of the time. Half a million calculations can be done in only a few seconds whereas running the magnetic field model in serial mode would have taken over 1700 hours. This translates into a speedup of over several million times.

6 Testing and Validating the Network

We validated our global neural network by comparing its results to the results from the actual magnetic field model. We chose a number of LANL geosynchronous satellites and calculated their L^* values in hourly resolution covering the years of 2002 and partially 2001 and 2005. The validation results of the neural network of this out-of-training-sample are shown in Fig. 5, 6. We have tested several different geosynchronous satellites and found similar performance with all of them. Figure 5 and 6 show one validation example with LANL-01A where the L^* values of the neural network L^* are plotted against the actual results from using TSK03. We find the standard deviation error is $\Delta L^* = 0.04$ or less than 1%.This is much lower than the intrinsic error of empirical magnetic field models (Huang et al., 2008) and shows that using the neural network will add only a marginal error. The overall error is calculated by adding the variances: $\sigma^2_{TSK03} + \sigma^2_{NN} = \sigma^2_{tot}$. We also show in Fig. 7 that the neural network L^* is indeed following the L^* calculation from using TSK03.

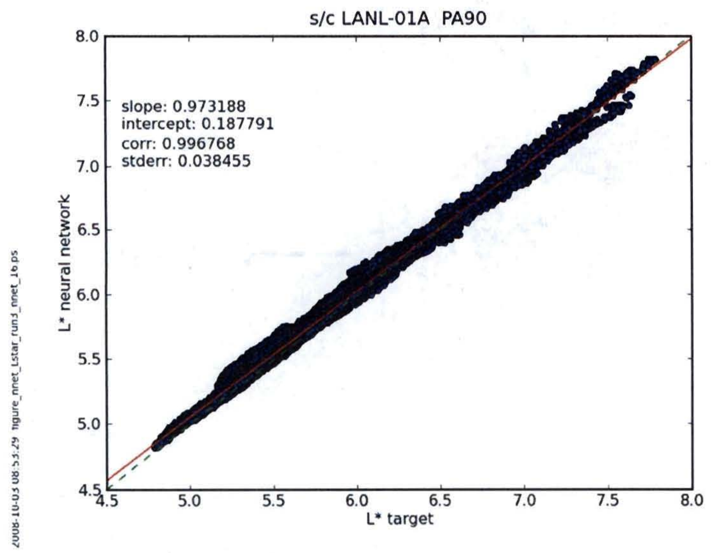

Fig. 5. Validation for the neural network using an out-of-sample data set from the positions of LANL-GEO spacecraft LANL-01A. Each point represents one L^* calculation by the Tsyganenko model versus the neural network L^* result. The dahsed green line would represent a perfect prediction by the neural network; the red line is a linear fit to the predictions. The standard deviation is $\Delta L* = 0.04$ or less than 1%.

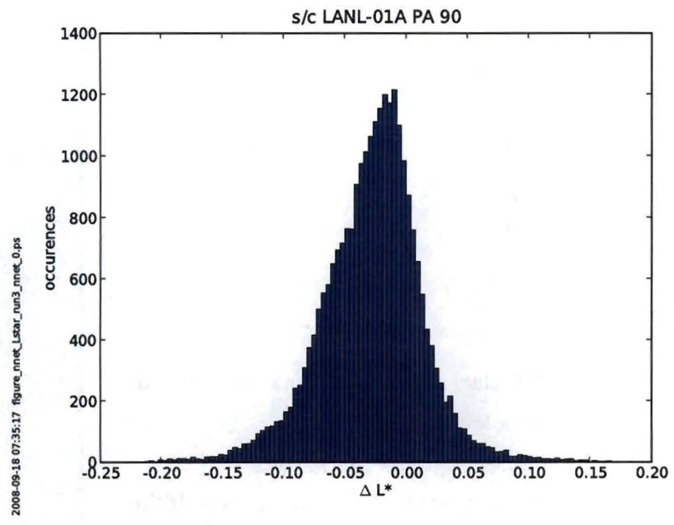

Fig. 6. Histogram plot of the error introduced by using the neural network.

7 How to Use the LANL* Neural Network

The complete library of neural networks plus examples are included as supplemental material to this publication. After extracting the files, read the "README" file and follow the instructions of using the Makefile and adopting your FORTRAN compiler.

After installation, use the following steps for calling the LANL* library.

1. Obtain the required Kp, Dst, and solar wind input parameters from an omni data base (from e.g. http://omniweb.gsfc.nasa.gov)

2. Obtain the coordinates of the spacecraft and convert to geomagnetic coordinates. This can be done with the onera-desp library by calling the coordinate transformation subroutine.

3. Decide on a pitch angle $\alpha \in [10°, 90°]$

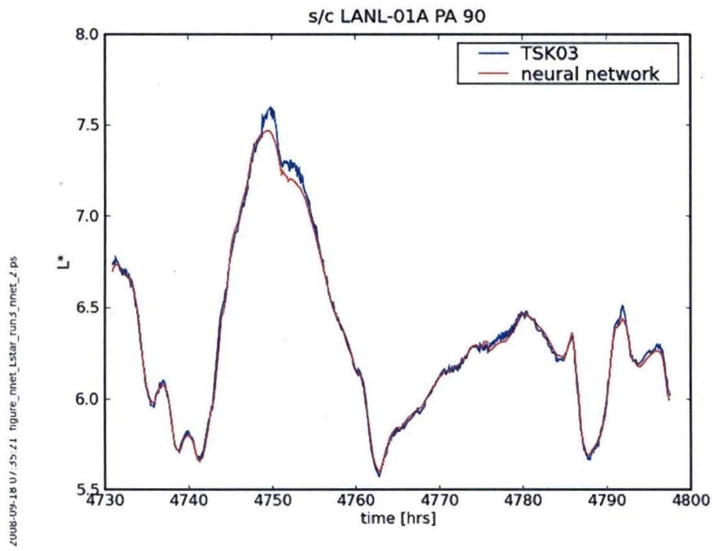

Fig. 7. Test case of calculating L^* with the Tsyganenko model TSK03 (blue) and with the neural network (red) for satellite LANL-01A. The standard deviation error is $\Delta L* = 0.04$ or less than 1%.

4. Call the LANL* subroutine to obtain the L^* value for the requested spacecraft location. If the spacecraft is located outside the last closed drift shell value and L^* is not defined, the number -99.0 is reported.

Since version 1.0 of the LANL* model was trained only with values inside the artificial geosynchronous coordinate torus, it should only be used for the geosynchronous region as well. If the satellite location is far away from this region, the results are expected to be bad because function approximating neural networks are known to have low extrapolation performance.

8 Conclusion and Summary

Space weather models for the inner magnetosphere use adiabatic invariants to convert observed quantities to phase space density coordinates in order to make a comparison with model results possible. In particular, radiation belt models use the coordinate L^* to study the dynamic and highly energetic environment which satellites are exposed to. L^* values are calculated by using current magentic field models and integrating along particle drift paths. However, such calculations require a long computational time which is sometimes not available.

We have presented an innovative method of calculating L^* values by creating a surrogate model using a forward-feed neural network method. Our method can replace complex and time-consuming magnetic field model integrations and provide a speed-up of several million times while adding only a marginal error of less than one percent.

The neural network is trained with hourly data from the year 2002 for a random number of locations. We found that 10 locations close to geosynchronous orbit was sufficient to create a well performing neural network. We also found that adding several more solar wind parameters increases the generalizability of the neural network. We have validated the neural network with geonsynchronous satellites and found less than one percent error with satellite LANL-01A.

The LANL* neural network for calculating L^* makes well-performing real-time and reanalysis applications feasible because our approach removes a major time-consuming task that previously prevented researchers from using accurate magnetic field models. Reanalysis of past geomagnetic events and even whole solar cycles can be used to better understand the physical processes controlling the environment in the inner magnetosphere. Real-time applications will become faster and better in assesing the current environment and forecasting future conditions.

The current version (V 1.0) of the LANL* libarary is only valid for geosynchronous regions. However, we are working on extending the neural network training to include the whole inner magnetosphere to be published in a future version of the LANL* library.

Acknowledgements. This work has been supported by Los Alamos National Laboratory's LDRD Directed Research Program (DREAM) and the National Science Foundation (grant 0718710).

References

Barron, A.: Approximation Bounds For Superpositions Of A Sigmoidal Function, Information Theory, IEEE Transactions on, pp. 85–85, 1991.

Barron, A.: Universal approximation bounds for superpositions of a sigmoidalfunction, Information Theory, IEEE Transactions on, 39, 930–945, 1993.

Barron, A. R.: Approximation and estimation bounds for artificial neural networks, Machine Learning, 14, 115, 1994.

Bishop, C. M.: Neural networks for pattern recognition, Clarendon Press ; Oxford University Press, Oxford; New York, 1995.

Boscher, D., Bourdarie, S., O'Brien, P., and Guild, T.: ONERA-DESP library V4.1, http://craterre.onecert.fr/support/user_guide.html, 2007.

Chen, Y., Friedel, R. H. W., Reeves, G. D., Cayton, T. E., and Christensen, R.: Multisatellite determination of the relativistic electron phase space density at geosynchronous orbit: An integrated investigation during geomagnetic storm times, J. Geophys. Res., 112, 1–16, a11214, 2007.

Cybenko, G.: Approximation by superpositions of a sigmoidal function, Mathematics of control, signals, and systems, 2, 303, 1989.

Dolenko, S., Orlov, Y., Persiantsev, I., and Shugai, Y.: Neural Network Analysis of Solar Wind Data, Applied Problems in systems of Patttern Recognition and Image Analysis, 11, 296–299, 2001.

Green, J. C. and Kivelson, M. G.: Relativistic electrons in the outer radiation belt: Differentiating between acceleration mechanisms, Journal of Geophysical Research (Space Physics), 109, 03 213, 2004.

Huang, C.-L., Spence, H. E., Singer, H. J., and Tsyganenko, N. A.: A quantitative assessment of empirical magnetic field models at geosynchronous orbit during magnetic storms, Journal of Geophysical Research (Space Physics), 113, 04 208, 2008.

Kim, H.-J. and Chan, A. A.: Fully adiabatic changes in storm time relativistic electron fluxes, Journal of Geophysical Research, 102, 22 107–22 116, dOI: 10.1029/97JA01814, 1997.

Kleijnen, J. P. C.: Design and analysis of simulation experiments, International series in operations research and management science, 111, Springer, New York, 2008.

Koons, H. C. and Gorney, D. J.: A neural network model of the relativistic electron flux at geosynchronous orbit, Journal of Geophysical Research, 96, 5549–5556, 1991.

Lundstedt, H.: Neural networks and predictions of solar-terrestrial effects, Planetary and Space Science, 40, 457–464, 1992.

McCollough, J. P., Gannon, J. L., Baker, D. N., and Gehmeyr, M.: A Statistical Comparison of Commonly Used External Magnetic Field Models, Space Weather, in press, 2008.

Myers, R. H. and Montgomery, D. C.: Response surface methodology process and product optimization using designed experiments, Wiley, New York; Chichester, 2002.

Reed, R. D. and Marks, R. J.: Neural smithing : supervised learning in feedforward artificial neural networks, The MIT Press, Cambridge, Mass., 1999.

Roederer, J. G.: Dynamics of geomagnetically trapped radiation, Physics and chemistry in space, v. 2, Springer-Verlag, Berlin; New York, 1970.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J.: Learning representations by back-propagating errors, Nature, 323, 533–536, 10.1038/323533a0, 1986.

Tsyganenko, N. A.: A model of the near magnetosphere with a dawn-dusk asymmetry 1. Mathematical structure, Journal of Geophysical Research (Space Physics), 107, 1179, 2002a.

Tsyganenko, N. A.: A model of the near magnetosphere with a dawn-dusk asymmetry 2. Parameterization and fitting to observations, Journal of Geophysical Research (Space Physics), 107, 1176, 2002b.

Tsyganenko, N. A. and Sitnov, M. I.: Modeling the dynamics of the inner magnetosphere during strong geomagnetic storms, Journal of Geophysical Research (Space Physics), 110, 03 208, 2005.

Tsyganenko, N. A., Singer, H. J., and Kasper, J. C.: Storm-time distortion of the inner magnetosphere: How severe can it get?, Journal of Geophysical Research (Space Physics), 108, 1209, 2003.

Wojciechowski, M.: ffnet: Feed-forward neural network for python, http://ffnet.sourceforge.net/, 2007.