Learning a stationary covariance

Douglas Nychka

Colorado School of Mines

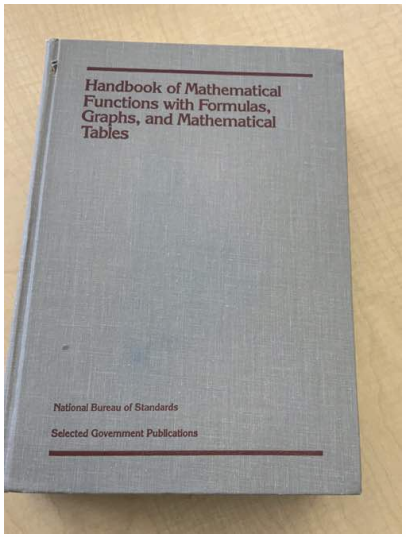
August 6, 2020

Outline

- Abramowitz and Stegun
- Climate model output
- Training Deep Nets on Gaussian Processes
- Comparison to Maximum likelihood
- Back to climate

Example of a statistical approximation

The normal cdf to 7 digits from *Abramowitz and Stegun*



$p = .47047 \quad a_1 = .34802\ 42 \quad a_2 = -.09587\ 98 \quad a_3 = .74785\ 56$

7.1.26

a numerical approximation — fifth degree polynomial

$$\operatorname{erf} x = 1 - (a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5) e^{-x^2} + \epsilon(x),$$

$$t = \frac{1}{1 + px}$$ — argument for poly

$$|\epsilon(x)| \leq 1.5 \times 10^{-7}$$

$p = .32759\ 11 \quad a_1 = .25482\ 9592$

$a_2 = -.28449\ 6736 \quad a_3 = 1.42141\ 3741$

$a_4 = -1.45315\ 2027 \quad a_5 = 1.06140\ 5429$

coefficients

7.1.27

Approximation works – but it is mysterious!

Example of a statistical approximation

Use convolutional neural networks (CNNs) to approximate maximum likelihood estimates.

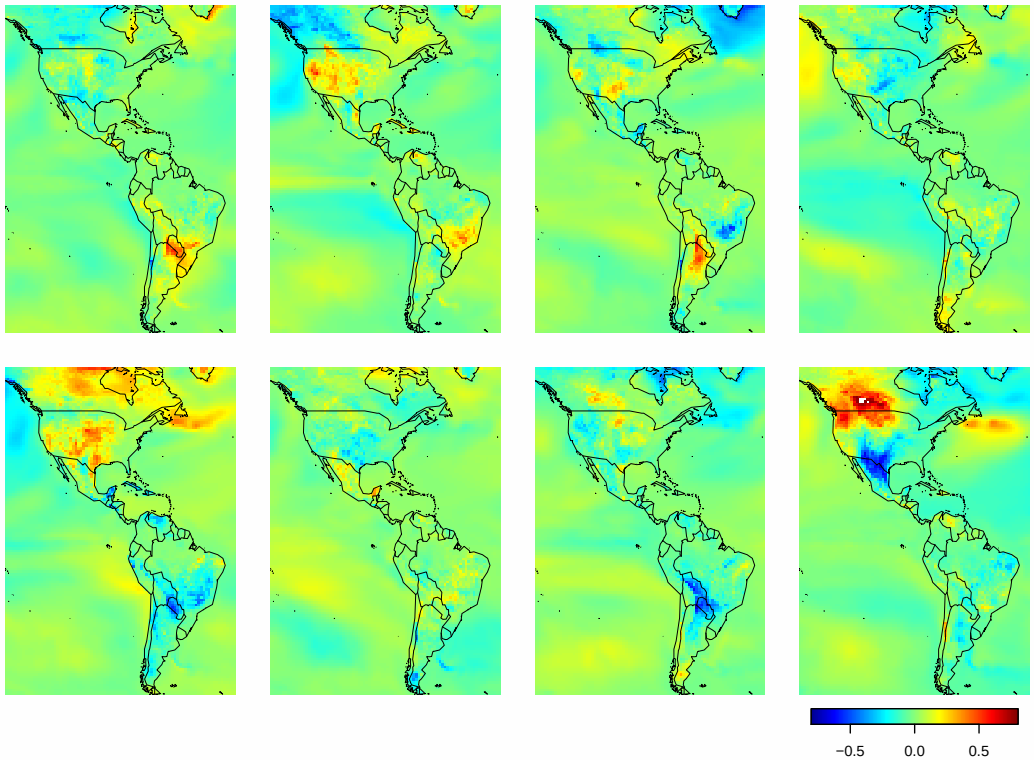
E.g. Covariance parameters for a Gaussian spatial process.

Approximation works – but it is mysterious!

Climate model output

Local temperature sensitivity to global temperature

First 8 out of 30 *centered* ensemble members



Goal: Simulate additional fields efficiently that match the spatial dependence in this 30 member ensemble.

A Statistical Approach

- $\approx$ 13K grid boxes over N and S America
- Estimate a spatially varying covariance function by fitting stationary covariances to small windows. (16×16)
- Range and variance parameter for each window.
- Encode local estimates into a global model to simulate Gaussian random fields.

Train a convolution neural net (CNN) to estimate covariance parameters.

I found a speedup by a factor of 50!

Matérn covariance function

Covariance function:

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sigma^2 \text{Matern function }_{\nu}(d)$$

with $d = ||\boldsymbol{x}_1 - \boldsymbol{x}_2||/\theta$

- Matérn function is $\mathcal{K}_\nu$ a modified Bessel function.
- Smoothness ν measures number of mean square derivatives and is equivalent to the polynomial tail behavior of the spectral density.
- θ is the range parameter.

Observational model

$$Y(\boldsymbol{x}) = g(\boldsymbol{x}) + \boldsymbol{e}(\boldsymbol{x})$$

with g following a Gaussian process with Matérn covariance and $\boldsymbol{e}$ white noise, variance τ^2.

We are interested in θ (range) and $\log(\tau^2)$ (log Variance).

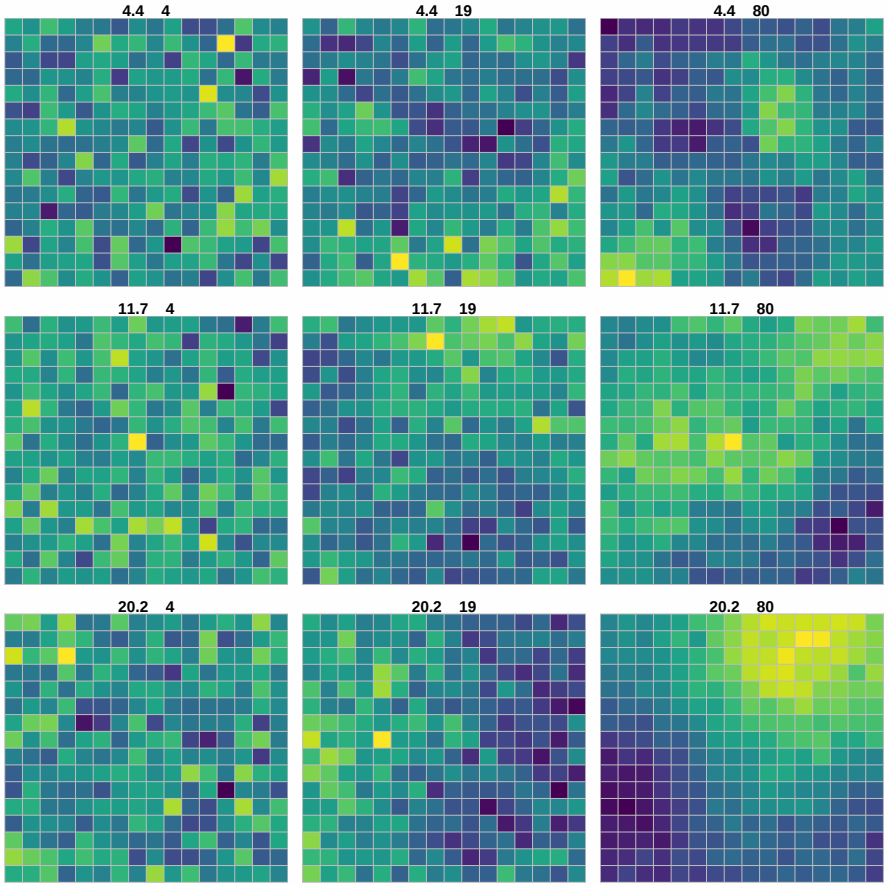
Examples of training fields

$$Y(\boldsymbol{x}) = g(\boldsymbol{x}) + \boldsymbol{e}$$

$\boldsymbol{x}$ on a 16×16 grid, $Var(Y) = 1$,

$\rightarrow$ decreasing noise (tau^2)

increasing range (θ) $\downarrow$



Neural net setup

Training / testing sets

- Input are 100K 16×16 Gaussian fields
- Grid of $20 \times 20 = 400$ range and log Variance levels
- Tested on $5 \times 6 = 30$ range and log Variance levels 4500 total
- MLEs for range and log variance found for each test image.

The CNNs

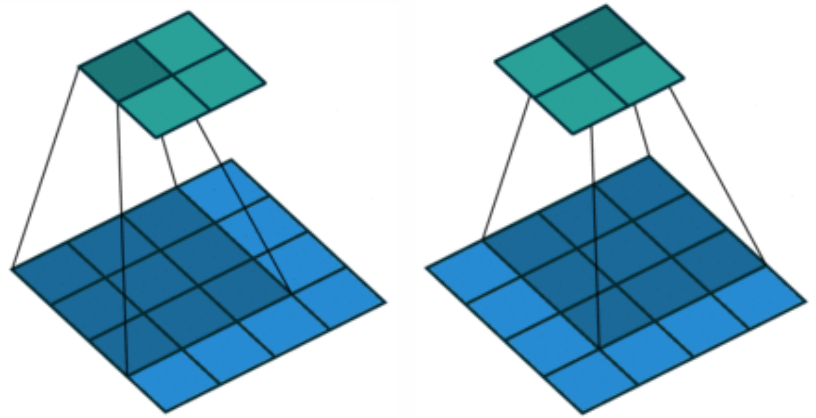
Simple example of net architecture and training

- Two convolution layers with 32, 3×3 filters
- max pooling layers inbetween.
- Flattened layer with fully connected neural network 128 inputs and giving two output values (range and log variance).
- A total of about 9800 CNN parameters to train.
- Training on batches of 128 and 30 epochs – takes several minutes.

Used the Keras interface in R for Tensor Flow – Easy!

A convolution step

A linear filter is applied to every 3×3 block of the input field followed by a nonlinear transformation



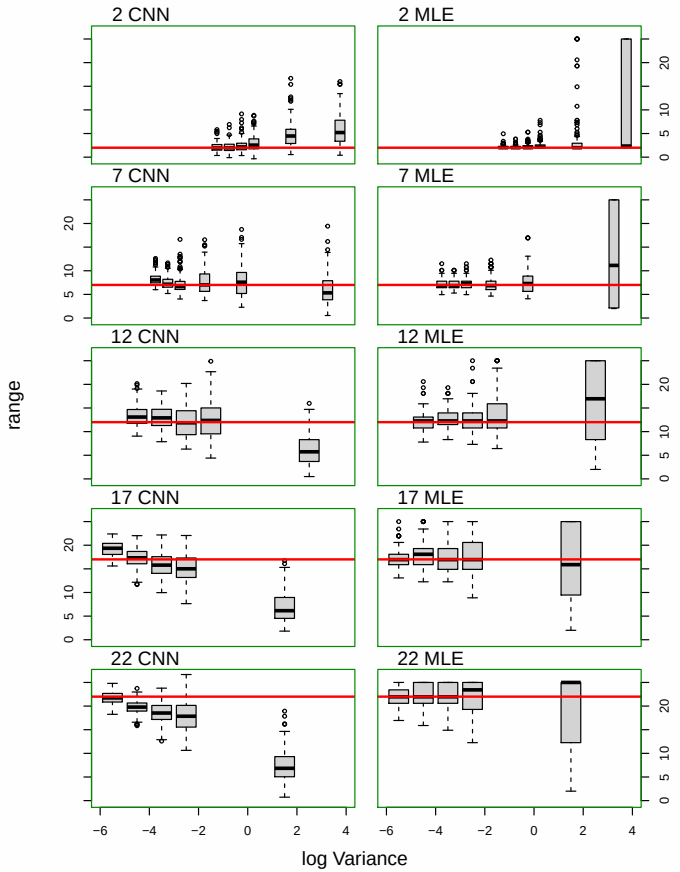
- These filter results are then filtered again . . . and again !
- Many filters (32) are considered.
- Filter weights found by training (of course!).

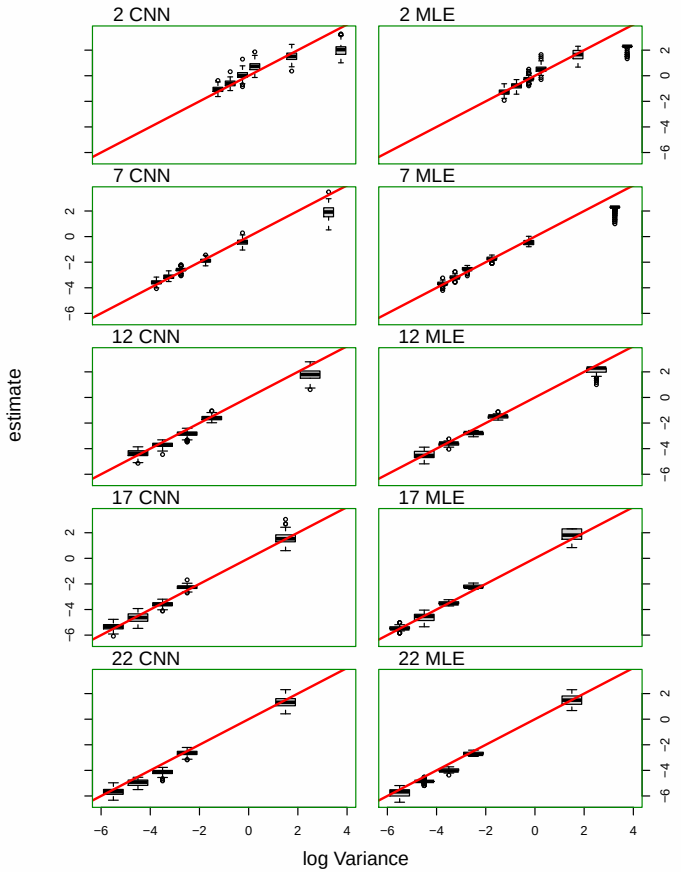
Parameters from the CNN verses MLEs

Red lines indicate true parameters

Range parameters (2,7,12,17,22)

log Variance (various)

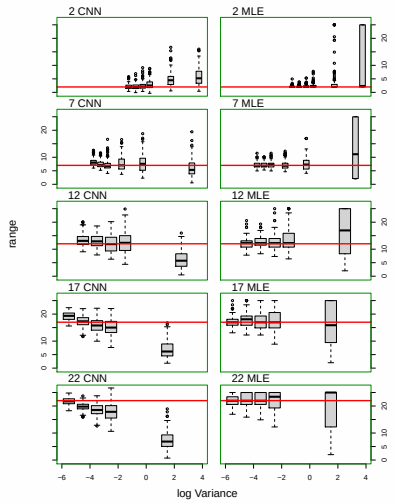




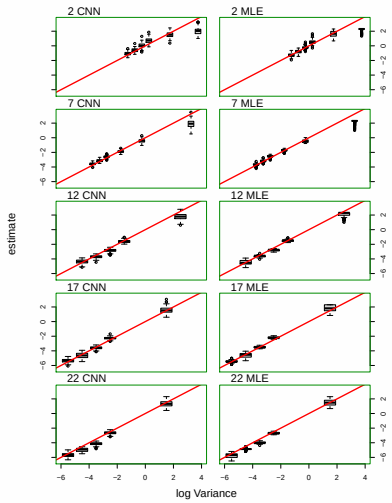
Results continued

- CNN and MLE estimates tend to track the red lines (truth)
- CNN overall has comparable accuracy to the MLEs
- CNN does not do as well for high noise and large ranges
- Tradeoff between bais in CNN estimates and variance in MLEs

Range parameter

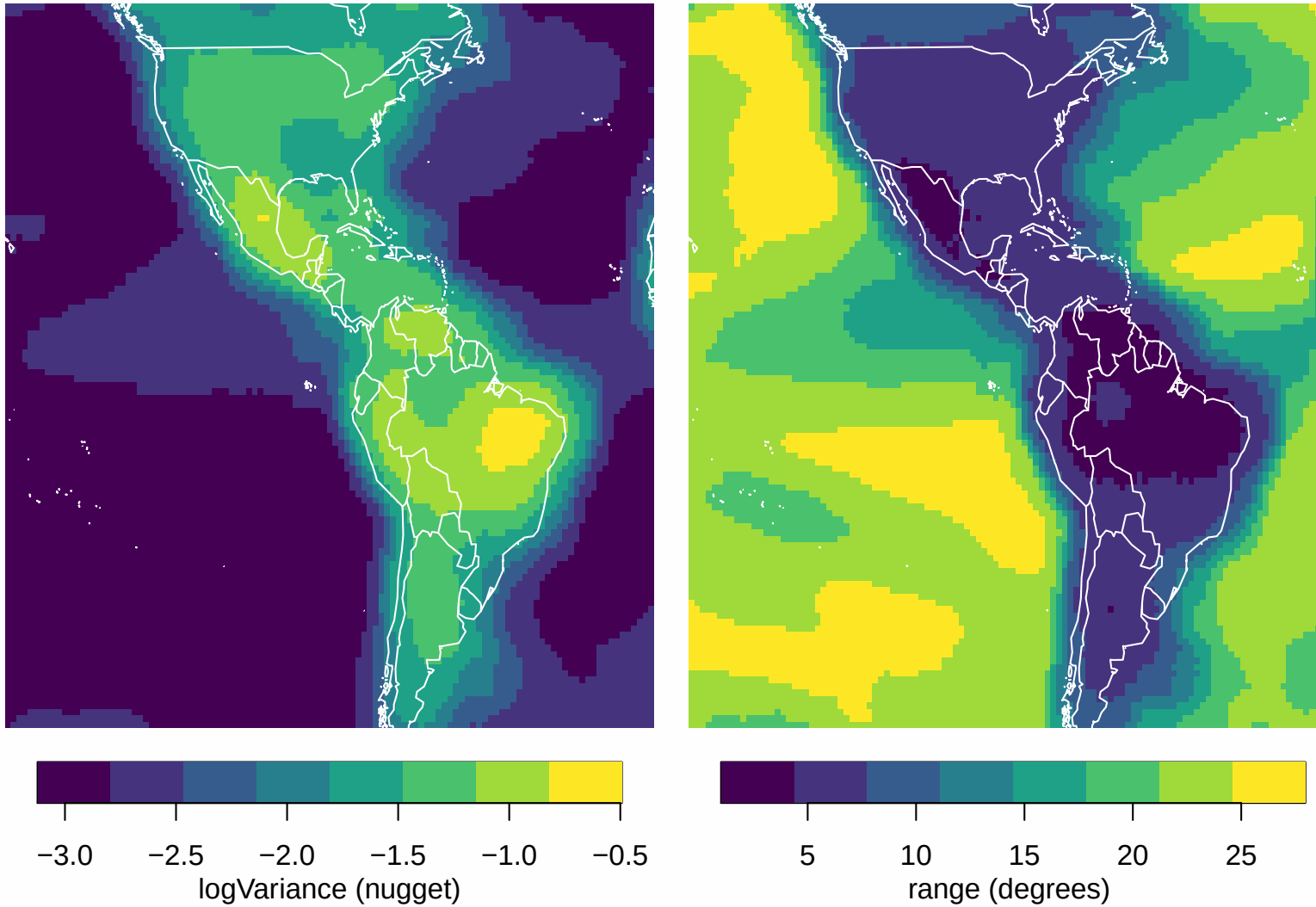


log Variance



Climate model emulation

Estimated log Variance and range using the CNN



Timing

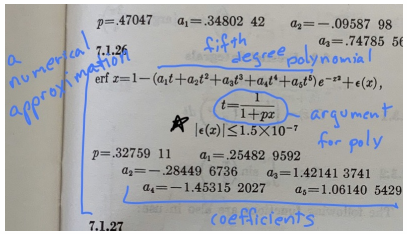
On my **MacBook Pro** 2.3 Ghz i5 core, 8GB memory,
and in R . . .

- Reshaping data for processing is surprisingly fast.
- 13K $\times$ 30 CNN estimates tic/ toc $\rightarrow \approx$ 14 seconds
- 13K $\times$ 30 MLEs tic/ toc $\rightarrow \approx$ 860 seconds

optimized as a tricksy coarse grid search will be much longer with more parameters

Loose ends

- How to train using replicate spatial fields?
- Train on a SAR model (LatticeKrig, SPDE) directly instead of Matérn .
- Train for the likelihood *surface* instead of just the estimates.
- Understand why this works!



Thank you

Keras/R specification

```
modelMatern11 %>%
  layer_conv_2d( 32, kernel_size=3, activation='relu',
                 input_shape=c(N,N,1) ) %>%
  layer_max_pooling_2d() %>%
  layer_conv_2d( 32, kernel_size=3, activation='relu') %>%
  layer_max_pooling_2d() %>%
  layer_flatten() %>% layer_dense(2)
```

Keras model summary

```
  > modelMatern11
Model
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 14, 14, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 5, 5, 32)          9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2D) (None, 2, 2, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 2)                 258
=================================================================
Total params: 9,826
Trainable params: 9,826
Non-trainable params: 0
_________________________________________________________________
```