Autoencoders

Deep Learning in Earth Science
Lecture 3
By Xiao Zhuowei

For researchers interested in studying Earth science with deep learning.

All resources in lectures are available at https://github.com/MrXiaoXiao/DLiES

OUTLINES

1 Autoencoder

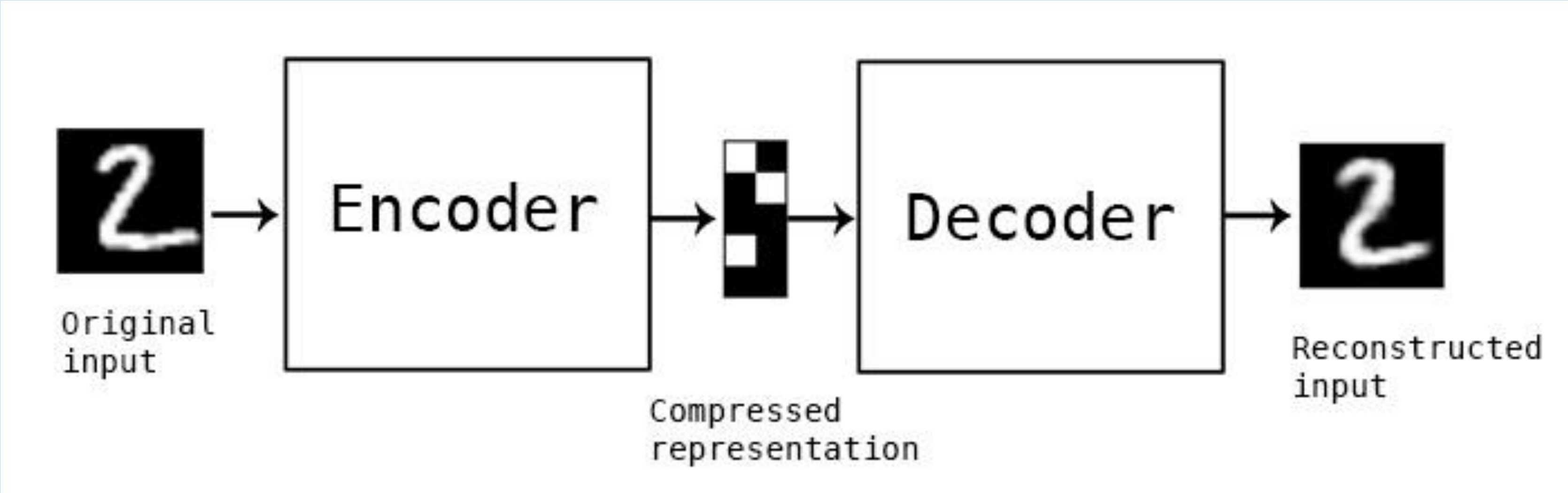
2 Variational autoencoder (VAE)

3 Paper Reading

4 Discussions

Autoencoder

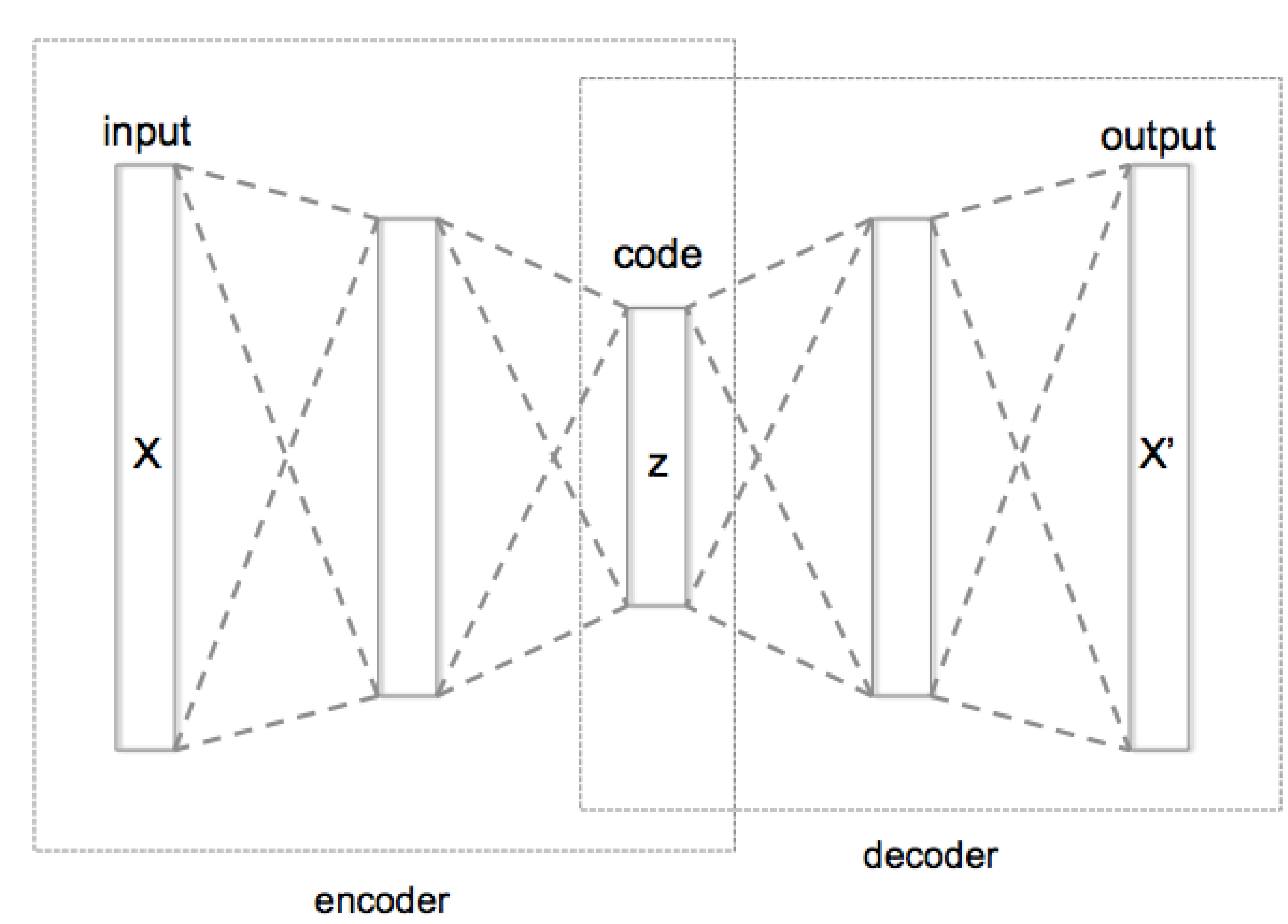
Encoding and Decoding



https://blog.keras.io/building-autoencoders-in-keras.html

Autoencoder

Autoencoder is a self-supervised learning method where the targets are generated from the input data.



https://en.wikipedia.org/wiki/Autoencoder#/media/File:Autoencoder_structure.png

Autoencoder

Fully connected autoenocder

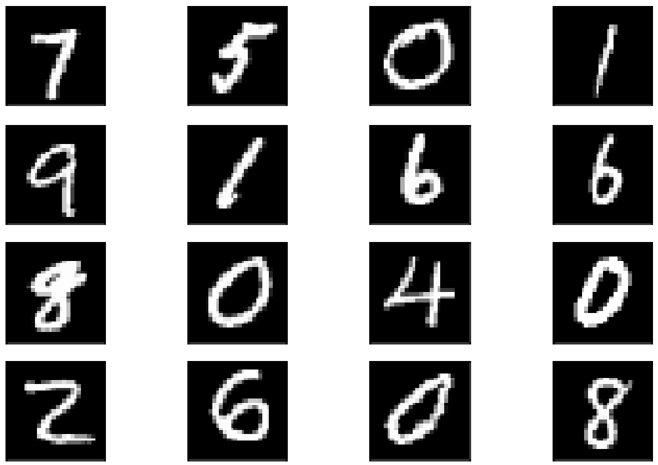
Prepare Data Set

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
import numpy as np
(x_train, _), (x_test, _) = mnist.load_data()
```

Check Data

```
import matplotlib.pyplot as plt
print('x_train shape: {}  x_test shape: {}'.format(np.shape(x_train),np.shape(x_test)))
for i in range(16):
    plt.subplot(4, 4, 1 + i, xticks=[], yticks=[])
    img_id = np.random.randint(np.shape(x_train)[0])
    im = x_train[img_id,::]
    plt.imshow(im)
    plt.gray()
```

```
x_train shape: (60000, 28, 28)  x_test shape: (10000, 28, 28)
```



Autoencoder

Fully connected autoenocder

Build Model

```
from tensorflow.keras.layers import Dense
autoencoder = tf.keras.Sequential()
#Encoder
autoencoder.add(Dense(128,activation='relu',input_shape=(784,)))
autoencoder.add(Dense(64,activation='relu'))

#Compressed representation
autoencoder.add(Dense(32,activation='relu'))

#Decoder
autoencoder.add(Dense(64,activation='relu'))
autoencoder.add(Dense(128,activation='relu'))
autoencoder.add(Dense(784,activation='sigmoid'))

autoencoder.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_3 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_4 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_5 (Dense)              (None, 784)               101136
=================================================================
Total params: 222,384
Trainable params: 222,384
Non-trainable params: 0
_________________________________________________________________
```

Autoencoder

Fully connected autoenocder

Train Model

```
autoencoder.compile(optimizer='adadelta', loss='mse')

autoencoder.fit(x_train, x_train,
                epochs=100,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 92/100
60000/60000 [==============================] - 3s 48us/step - loss: 0.0174 - val_loss: 0.0176
Epoch 93/100
60000/60000 [==============================] - 3s 48us/step - loss: 0.0173 - val_loss: 0.0169
Epoch 94/100
60000/60000 [==============================] - 3s 49us/step - loss: 0.0173 - val_loss: 0.0166
Epoch 95/100
60000/60000 [==============================] - 3s 49us/step - loss: 0.0172 - val_loss: 0.0168
Epoch 96/100
60000/60000 [==============================] - 3s 49us/step - loss: 0.0171 - val_loss: 0.0166
Epoch 97/100
60000/60000 [==============================] - 3s 48us/step - loss: 0.0171 - val_loss: 0.0167-
Epoch 98/100
60000/60000 [==============================] - 3s 48us/step - loss: 0.0169 - val_loss: 0.0165
Epoch 99/100
60000/60000 [==============================] - 3s 48us/step - loss: 0.0169 - val_loss: 0.0167
Epoch 100/100
60000/60000 [==============================] - 3s 49us/step - loss: 0.0168 - val_loss: 0.0165

<tensorflow.python.keras.callbacks.History at 0x212577447f0>
```

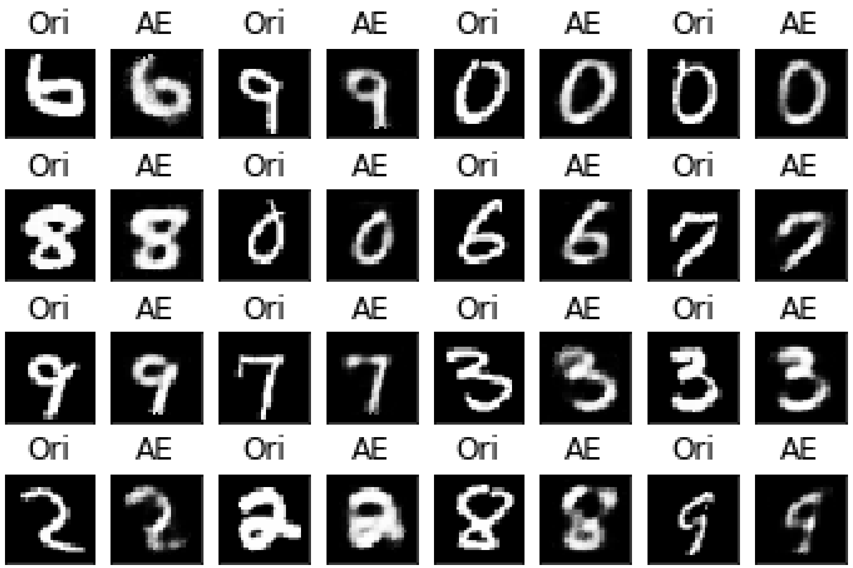
Autoencoder

Fully connected autoenocder

Evaluate Model

```
x_recon = autoencoder.predict(x_test)
for i in range(16):
    plt.subplot(4, 8, 1 + i*2, xticks=[], yticks=[])
    img_id = np.random.randint(np.shape(x_test)[0])
    im = x_test[img_id,::].reshape([28,28])
    plt.imshow(im)
    plt.gray()
    plt.title('Ori')
    plt.subplot(4, 8, 2 + i*2, xticks=[], yticks=[])

    im = x_recon[img_id,::].reshape([28,28])
    plt.imshow(im)
    plt.gray()
    plt.title('AE')
```



Autoencoder

Fully connected autoenocder

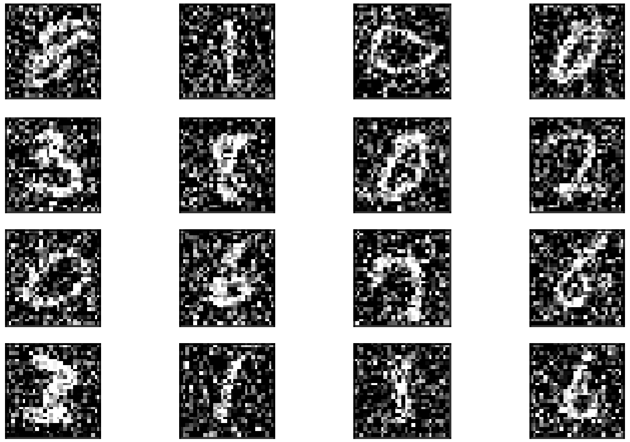
Denosing with Autoencoder

Add noise to data ¶

In [23]:
```
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```

Check Data

In [24]:
```
for i in range(16):
    plt.subplot(4, 4, 1 + i, xticks=[], yticks=[])
    img_id = np.random.randint(np.shape(x_train)[0])
    im = x_train_noisy[img_id,::].reshape([28,28])
    plt.imshow(im)
    plt.gray()
```



Autoencoder

Fully connected autoenocder

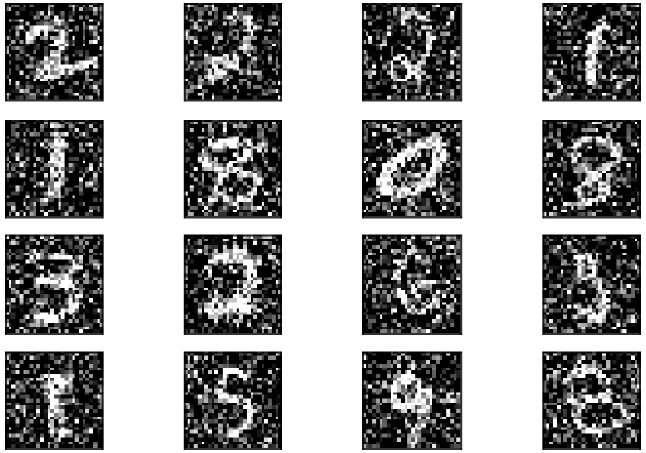
Denosing with Autoencoder

Add noise to data

```
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```

Check Data ¶

```
for i in range(16):
    plt.subplot(4, 4, 1 + i, xticks=[], yticks=[])
    img_id = np.random.randint(np.shape(x_train)[0])
    im = x_train_noisy[img_id,::].reshape([28,28])
    plt.imshow(im)
    plt.gray()
```



Autoencoder

Fully connected autoenocder

Create a New Model for Denosing

```
tf.reset_default_graph()
tf.keras.backend.clear_session()
autoencoder = tf.keras.Sequential()
#Encoder
autoencoder.add(Dense(128,activation='relu',input_shape=(784,)))
autoencoder.add(Dense(64,activation='relu'))

#Compressed representation
autoencoder.add(Dense(32,activation='relu'))

#Decoder
autoencoder.add(Dense(64,activation='relu'))
autoencoder.add(Dense(128,activation='relu'))
autoencoder.add(Dense(784,activation='sigmoid'))

#Train
autoencoder.compile(optimizer='adadelta', loss='mse')

autoencoder.fit(x_train, x_train,
                epochs=100,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 92/100
60000/60000 [==============================] - 3s 47us/step - loss: 0.0178 - val_loss: 0.0173
Epoch 93/100
60000/60000 [==============================] - 3s 50us/step - loss: 0.0177 - val_loss: 0.0172A: 1s - - ETA: 0s - loss: 0. - ETA: 0s - los
Epoch 94/100
60000/60000 [==============================] - 3s 48us/step - loss: 0.0176 - val_loss: 0.0172
Epoch 95/100
60000/60000 [==============================] - 3s 47us/step - loss: 0.0175 - val_loss: 0.0171
Epoch 96/100
60000/60000 [==============================] - 3s 47us/step - loss: 0.0175 - val_loss: 0.0170
Epoch 97/100
60000/60000 [==============================] - 3s 47us/step - loss: 0.0174 - val_loss: 0.0168
Epoch 98/100
60000/60000 [==============================] - 3s 49us/step - loss: 0.0173 - val_loss: 0.0168
Epoch 99/100
60000/60000 [==============================] - 3s 49us/step - loss: 0.0173 - val_loss: 0.0167
Epoch 100/100
60000/60000 [==============================] - 3s 48us/step - loss: 0.0172 - val_loss: 0.0170
```

```
<tensorflow.python.keras.callbacks.History at 0x21203d9d160>
```

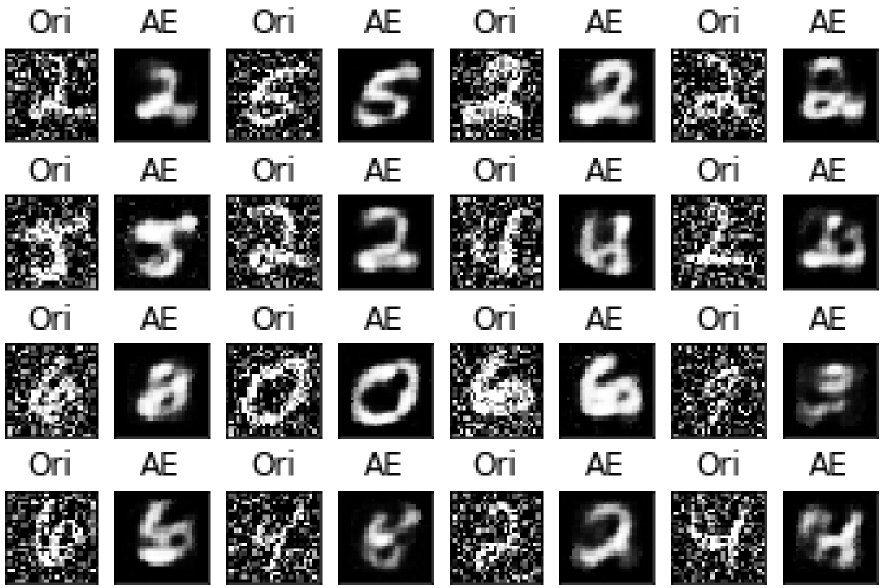
Autoencoder

Fully connected autoenocder

Evaluate Model

```
x_denoise = autoencoder.predict(x_test_noisy)
for i in range(16):
    plt.subplot(4, 8, 1 + i*2, xticks=[], yticks=[])
    img_id = np.random.randint(np.shape(x_test_noisy)[0])
    im = x_test_noisy[img_id,::].reshape([28,28])
    plt.imshow(im)
    plt.gray()
    plt.title('Ori')
    plt.subplot(4, 8, 2 + i*2, xticks=[], yticks=[])

    im = x_denoise[img_id,::].reshape([28,28])
    plt.imshow(im)
    plt.gray()
    plt.title('AE')
```



Autoencoder

Convolutional connected autoencoder

Build Model

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, UpSampling2D

autoencoder = tf.keras.Sequential()
#Encoder
autoencoder.add(Conv2D(32, (3,3),activation='relu',padding='same',input_shape=x_train.shape[1:]))
autoencoder.add(MaxPooling2D(pool_size=(2, 2)))
autoencoder.add(Conv2D(32, (3,3),activation='relu',padding='same'))

#Compressed representation
autoencoder.add(MaxPooling2D(pool_size=(2, 2)))

#Decoder
autoencoder.add(Conv2D(32, (3,3),activation='relu',padding='same'))
autoencoder.add(UpSampling2D((2, 2)))
autoencoder.add(Conv2D(32, (3,3),activation='relu',padding='same'))
autoencoder.add(UpSampling2D((2, 2)))
autoencoder.add(Conv2D(1, (3,3),activation='sigmoid',padding='same'))

autoencoder.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 32)          9248
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 32)        9248
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 28, 28, 1)         289
```

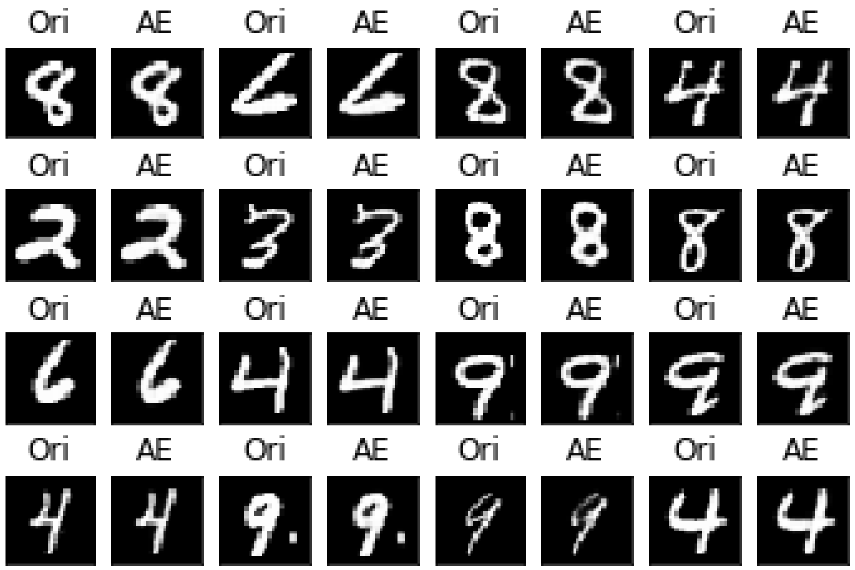
Autoencoder

Convolutional connected autoenocder

Evaluate Model

```
x_recon = autoencoder.predict(x_test)
for i in range(16):
    plt.subplot(4, 8, 1 + i*2, xticks=[], yticks=[])
    img_id = np.random.randint(np.shape(x_test)[0])
    im = x_test[img_id,::].reshape([28,28])
    plt.imshow(im)
    plt.gray()
    plt.title('Ori')
    plt.subplot(4, 8, 2 + i*2, xticks=[], yticks=[])

    im = x_recon[img_id,::].reshape([28,28])
    plt.imshow(im)
    plt.gray()
    plt.title('AE')
```



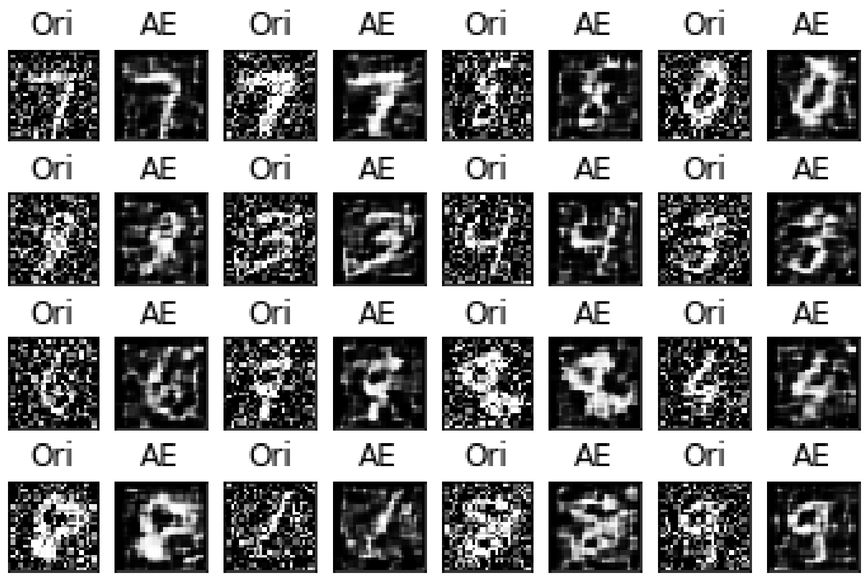
Autoencoder

Evaluate Model

Convolutional connected autoenocder

```
x_denoise = autoencoder.predict(x_test_noisy)
for i in range(16):
    plt.subplot(4, 8, 1 + i*2, xticks=[], yticks=[])
    img_id = np.random.randint(np.shape(x_test_noisy)[0])
    im = x_test_noisy[img_id,::].reshape([28,28])
    plt.imshow(im)
    plt.gray()
    plt.title('Ori')
    plt.subplot(4, 8, 2 + i*2, xticks=[], yticks=[])

    im = x_denoise[img_id,::].reshape([28,28])
    plt.imshow(im)
    plt.gray()
    plt.title('AE')
```



OUTLINES

1 Autoencoder

2 Variational autoencoder (VAE)

3 Paper Reading

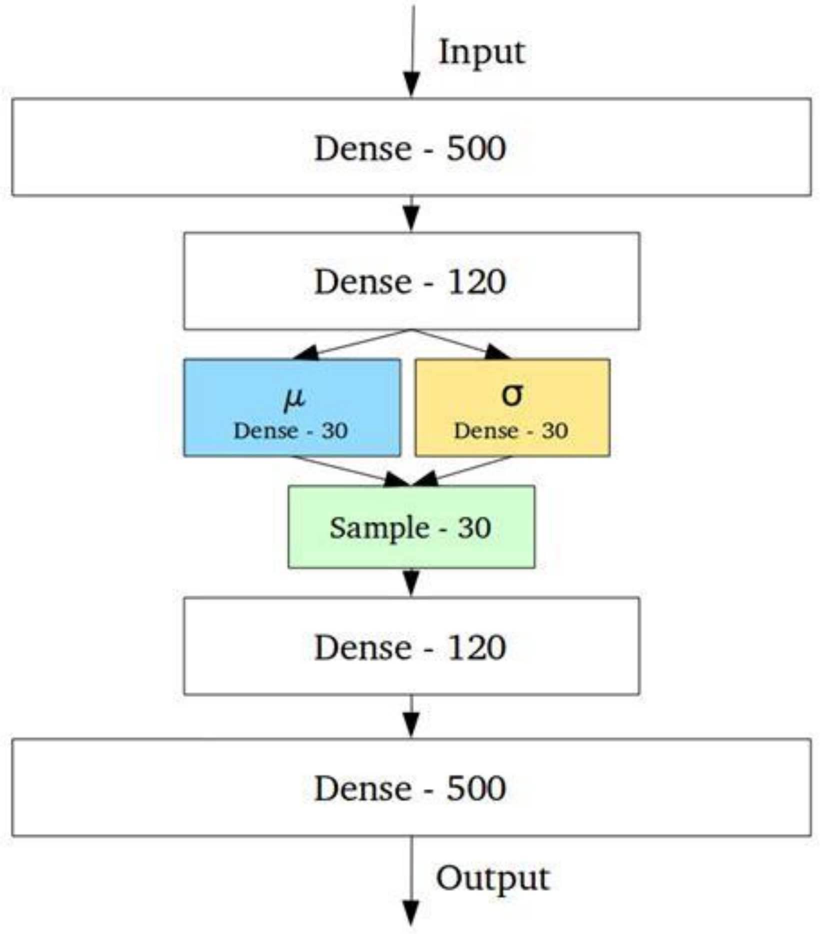
4 Discussions

VAE

- an end-to-end autoencoder mapping inputs to reconstructions
- an encoder mapping inputs to the latent space
- a generator that can take points on the latent space and will output the corresponding reconstructed samples.

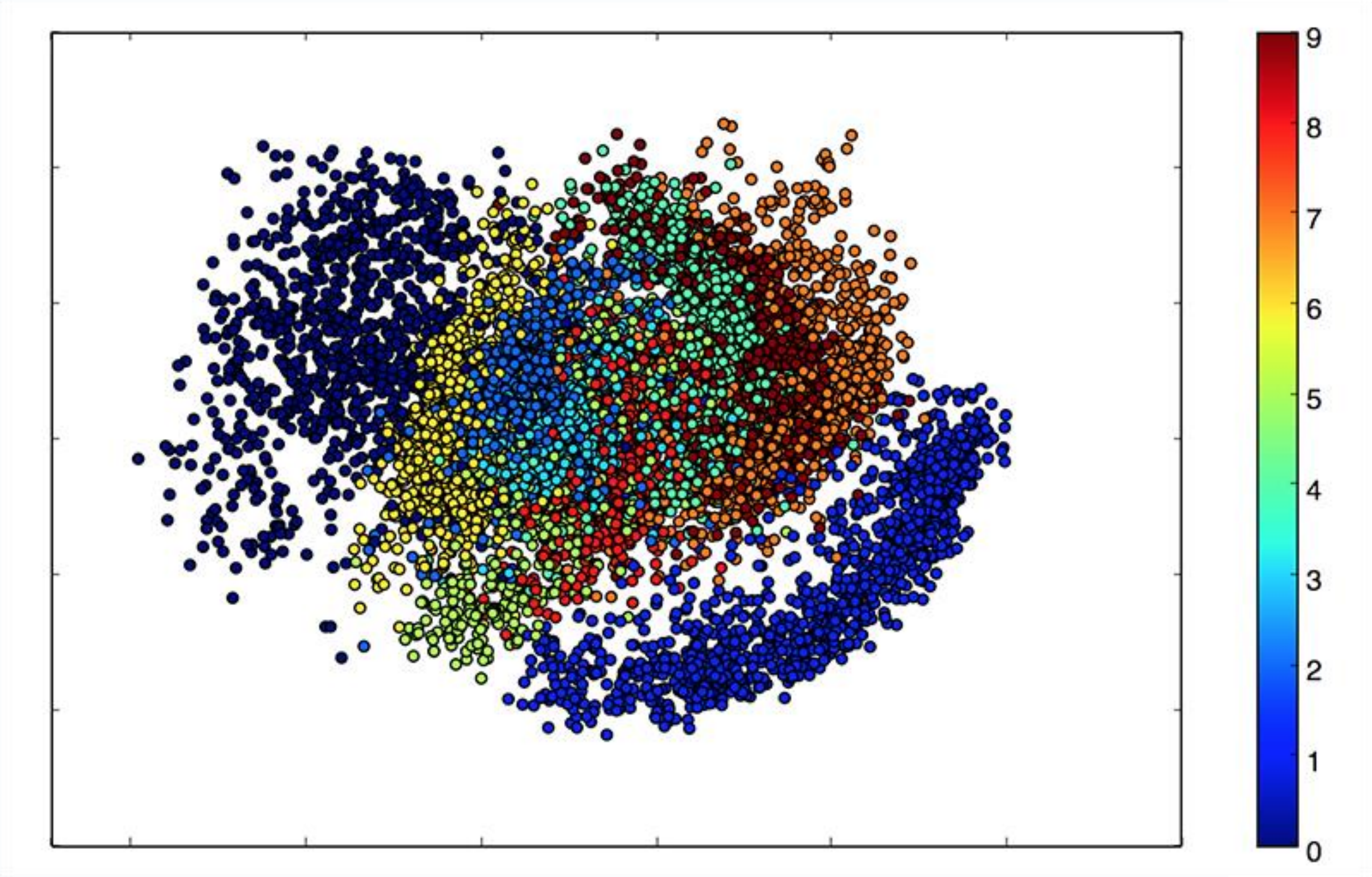
First, here's our encoder network, mapping inputs to our latent distribution parameters:

```
x = Input(batch_shape=(batch_size, original_dim))
h = Dense(intermediate_dim, activation='relu')(x)
z_mean = Dense(latent_dim)(h)
z_log_sigma = Dense(latent_dim)(h)
```



VAE

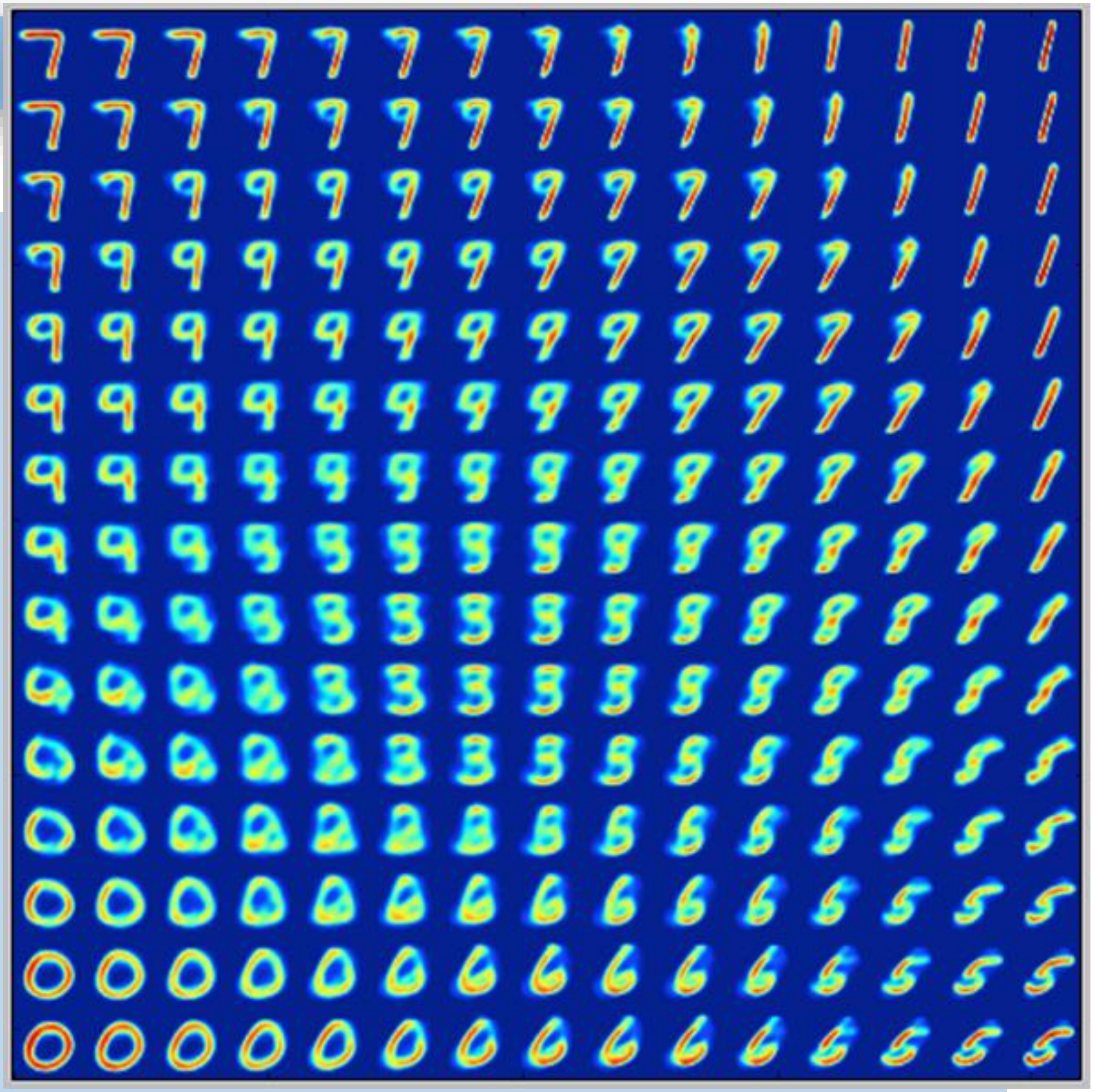
Different classes on the latent 2D plane



https://blog.keras.io/building-autoencoders-in-keras.html

VAE

Visualization of the latent manifold that "generates" the MNIST digits.



https://blog.keras.io/building-autoencoders-in-keras.html

OUTLINES

1 Autoencoder

2 Variational autoencoder (VAE)

3 Paper Reading

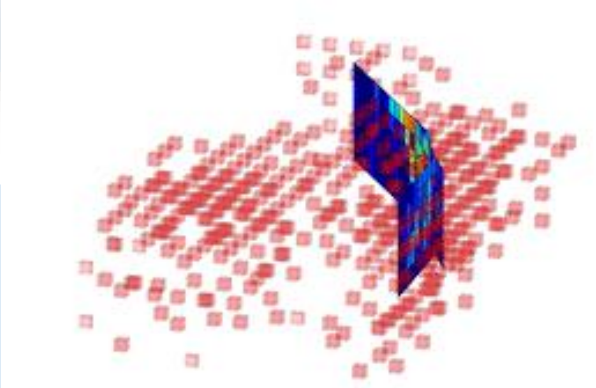
4 Discussions

Aftershock Basics

DeVries, P.M.R., Viégas, F., Wattenberg, M., Meade, B.J., 2018. Deep learning of aftershock patterns following large earthquakes. Nature 560, 632–634. https://doi.org/10.1038/s41586-018-0438-y

Large earthquakes can be followed by thousands of smaller ones, called ***aftershocks***. In February 2011, an aftershock struck the city of Christchurch, New Zealand, and was more destructive than the earthquake it followed, killing more than 100 people.

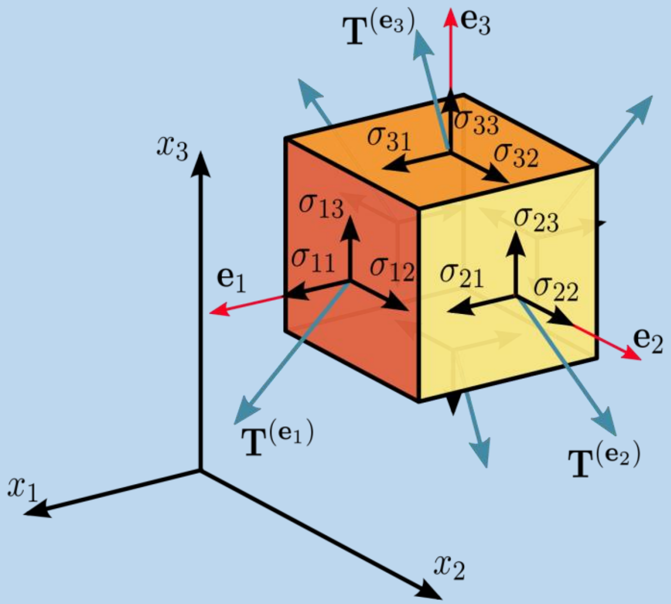
(Source: https://theprovince.com/feature/b-c-earthquake-threatens-vancouver-buildings/chapter-2)



(https://interestingengineering.com/harvard-and-google-develop-ai-that-can-forecast-earthquake-aftershocks-for-up-to-a-year)

Problem Formulation

Aftershock location forecasting as a binary classification problem.



https://en.wikipedia.org/wiki/Stress_(mechanics)#/media/File:Components_stress_tensor_cartesian.svg

Predict each grid cell whether it contains an aftershock with six independent components stress-change tensor calculated at the centroid.

Input and Output

NN Input: Magnitudes of the six independent components of the co-seismically generated static elastic stress-change tensor calculated at the centroid of a grid cell and their negative values

Label: 0.0 for negative and 1.0 for positive.

NN Output: A value between 0.0 and 1.0. ‘Can be interpreted as’ probability.

Inspect Training Data

```
[1]: import h5py
     import numpy as np
```

Open Training File

```
[2]: training_set = h5py.File('Training.h5','r')
```

List contents

```
[3]: for key in training_set.keys():
         print('key: {} shape: {}'.format(key,np.shape(training_set[key])))
```

```
key: aftershocksyn shape: (4743090,)
key: stresses_full_xx shape: (4743090,)
key: stresses_full_xy shape: (4743090,)
key: stresses_full_xz shape: (4743090,)
key: stresses_full_yy shape: (4743090,)
key: stresses_full_yz shape: (4743090,)
key: stresses_full_zz shape: (4743090,)
```

Positive Instance Example

```
In [4]: pos_id = np.argmax(training_set['aftershocksyn'])
```

```
In [5]: print('Instance ID: {}'.format(pos_id))
        for key in training_set.keys():
            print('key: {} value: {}'.format(key,training_set[key][pos_id]))
```

```
Instance ID: 3097
key: aftershocksyn value: 1.0
key: stresses_full_xx value: 2095977.7215682976
key: stresses_full_xy value: 547444.7915342181
key: stresses_full_xz value: -267392.6365251403
key: stresses_full_yy value: 210667.83512167187
key: stresses_full_yz value: 125209.89888368621
key: stresses_full_zz value: 333231.25020027714
```

Negative Instance Example

```
In [6]: neg_id = np.argmin(training_set['aftershocksyn'])
```

```
In [7]: print('Instance ID: {}'.format(neg_id))
        for key in training_set.keys():
            print('key: {} value: {}'.format(key,training_set[key][neg_id]))
```

```
Instance ID: 0
key: aftershocksyn value: 0.0
key: stresses_full_xx value: -9007.11694959848
key: stresses_full_xy value: 273.9997973169O865
key: stresses_full_xz value: -257.37695368028983
key: stresses_full_yy value: -1579.2613506035939
key: stresses_full_yz value: -287.9601254820999
key: stresses_full_zz value: -12.931813245567795
```

Predicting Aftershocks with Deep Learning

```
#model setup
def create_model():
    model = Sequential()
    model.add(Dense(50, input_dim=12, kernel_initializer='lecun_uniform', activation = 'tanh'))
    model.add(Dropout(0.50))
    model.add(Dense(50, kernel_initializer='lecun_uniform', activation= 'tanh'))
    model.add(Dropout(0.50))
    model.add(Dense(50, kernel_initializer='lecun_uniform', activation= 'tanh'))
    model.add(Dropout(0.50))
    model.add(Dense(50, kernel_initializer='lecun_uniform', activation= 'tanh'))
    model.add(Dropout(0.50))
    model.add(Dense(50, kernel_initializer='lecun_uniform', activation= 'tanh'))
    model.add(Dropout(0.50))
    model.add(Dense(50, kernel_initializer='lecun_uniform', activation= 'tanh'))
    model.add(Dropout(0.50))
    model.add(Dense(1, kernel_initializer='lecun_uniform', activation='sigmoid'))
    model.compile(optimizer='adadelta', loss='binary_crossentropy', metrics=[metrics.binary_accuracy])
    return model
```

Predicting Aftershocks with Deep Learning

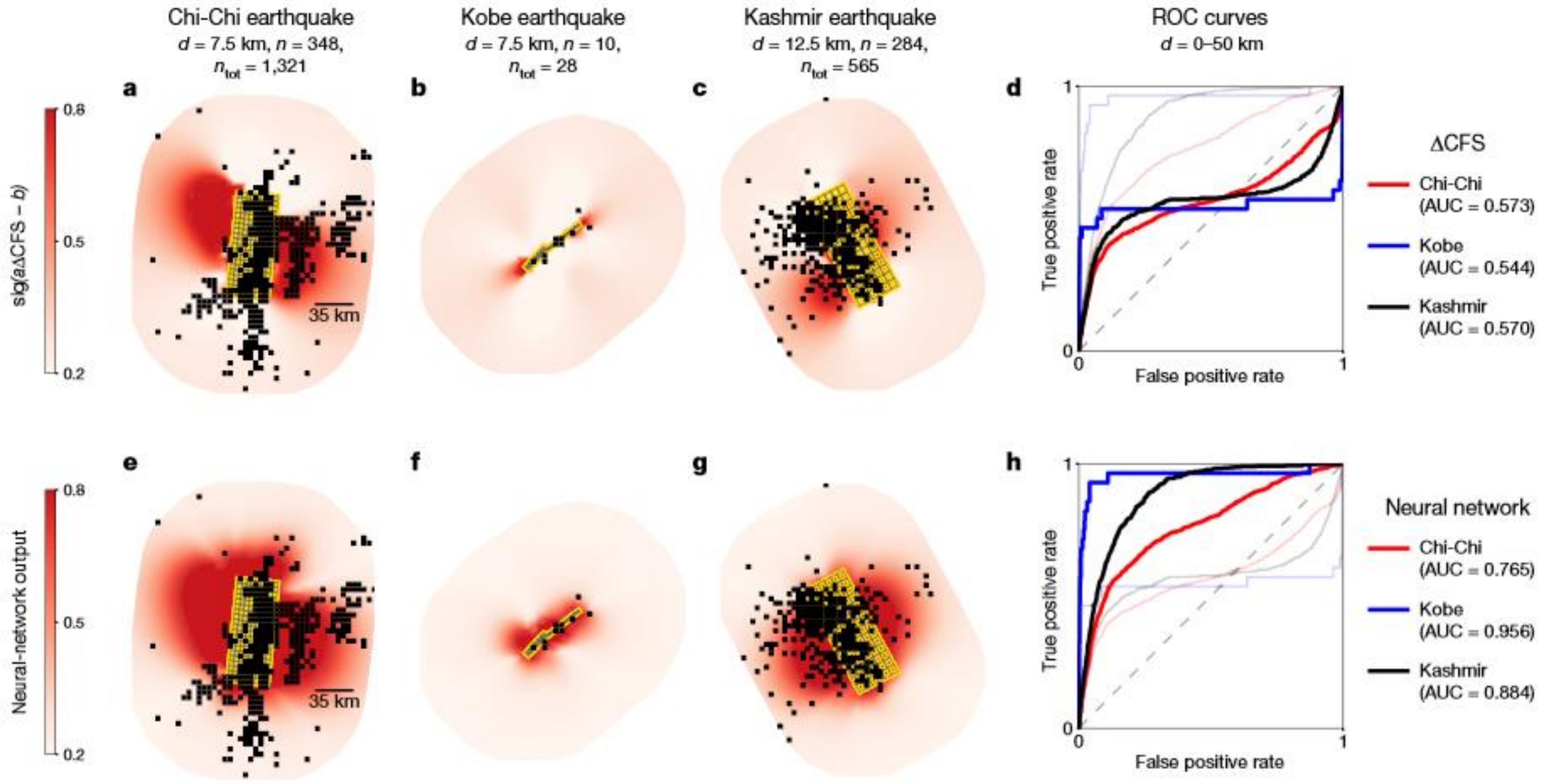


Fig. 1 | Mainshock–aftershock examples. a–d, Spatial patterns of ΔCFS($\mu = 0.4$) for the 1999 $M_w = 7.7$ Chi-Chi earthquake[17] at a depth of 7.5 km (**a**), the 1995 $M_w = 6.9$ Kobe earthquake[18] at a depth of 7.5 km (**b**) and the 2005 $M_w = 7.6$ Kashmir earthquake[19] at a depth of 12.5 km (**c**), along with ROC curves for all three earthquakes across all depths (**d**). In **a**–**c**, n refers to the number of positive grid cells at the depth shown and n_{tot} is the number of positive grid cells across all depths. A 1:1 grey dashed line is included in **d** for reference. Because of possible sign ambiguities, we calculate four versions of ΔCFS($\mu = 0.4$) and use the best-performing sign convention for each slip distribution. In **a**–**c**, ΔCFS($\mu = 0.4$) values (in megapascals) are fed through a sigmoid filter sig(x) = $1/(1 + e^{-x})$ (sig($a\Delta$CFS($\mu = 0.4$) − b), with $a = 10$, $b = 1$; colour scale) to facilitate comparison to the neural network; faults are shown in yellow and grid cells that contain aftershocks are shown in black. **e**–**h**, Analogous to **a**–**d** but for the neural network. To facilitate easy comparison, the ROC curves in **d** are plotted as pale lines in **h** and the ROC curves in **h** are plotted as pale lines in **d**.

Predicting Aftershocks with Deep Learning

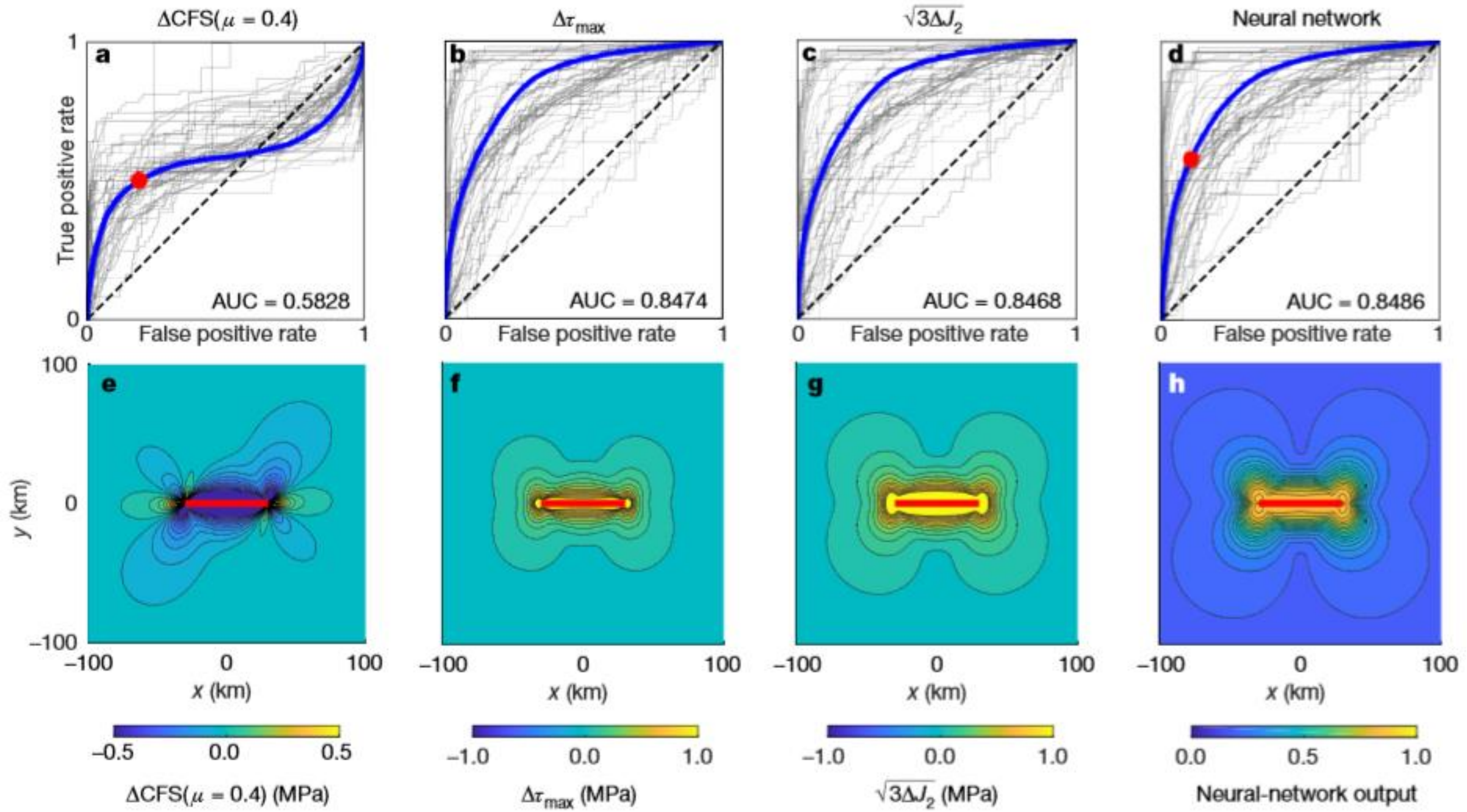


Fig. 2 | Comparison of performance. a–d, ROC curves for every slip distribution in the test dataset (grey curves) for $\Delta CFS(\mu = 0.4)$ (**a**), $\Delta\tau_{max}$ (**b**), $\sqrt{3\Delta J_2}$ (**c**) and the neural network (**d**). Merged ROC curves are shown in blue and the associated AUC values are listed. The red circles in **a** and **d** highlight the thresholds of 0.01 MPa and 0.5, respectively. **e–h**, For a synthetic case of a 60-km-long, right-lateral strike-slip fault (red lines) at a depth of 10 km, we show a comparison of the spatial patterns of $\Delta CFS(\mu = 0.4)$ (**e**), $\Delta\tau_{max}$ (**f**), $\sqrt{3\Delta J_2}$ (**g**) and the neural network (**h**), averaged over the fault strike.

Predicting Aftershocks with Deep Learning

Quantity	Symbol	Evaluation	%VE
nearest distance	r	$r = \min\left(\sqrt{(x - x_f)^2 + (y - y_f)^2}\right)$	46%
maximum shear	$\Delta\tau_{max}(\chi)$	$\Delta\tau_{max}(\chi) = \lvert\chi_1 - \chi_3\rvert/2$	98%
1st invariant	$\Delta I_1(\chi)$	$\Delta I_1(\chi) = \chi_1 + \chi_2 + \chi_3$	66%
2nd invariant	$\Delta I_2(\chi)$	$\Delta I_2(\chi) = \chi_1\chi_2 + \chi_2\chi_3 + \chi_1\chi_3$	96%
von-Mises criteria	$\sqrt{3\Delta J_2}$	$\sqrt{3\Delta J_2} = \sqrt{\Delta I_1^2(\sigma) - 3\Delta I_2(\sigma)}$	98%
3rd invariant	$\Delta I_3(\chi)$	$\Delta I_3(\chi) = \chi_1\chi_2\chi_3$	84%
Coulomb failure criteria μ = 0.0, 0.2, 0.4, 0.6, 0.8	$\Delta CFS(\chi, \mu)$	$\Delta CFS(\chi, \mu) = (\mathbf{n}_\perp \cdot \chi) \cdot \mathbf{n}_\parallel - \mu(\mathbf{n}_\perp \cdot \chi) \cdot \mathbf{n}_\perp$	89%
Coulomb failure criteria normal only, $\mu = 0.4$	$\Delta CFS_n(\chi)$	$\Delta CFS_n(\chi) = -\mu(\mathbf{n}_\perp \cdot \chi) \cdot \mathbf{n}_\perp$	59%
Coulomb failure criteria total shear	$\Delta CFS_\tau(\chi)$	$\Delta CFS_\tau(\chi) = \lvert(\mathbf{n}_\perp \cdot \chi) \cdot \mathbf{n}_\parallel\rvert + \lvert(\mathbf{n}_\perp \cdot \chi) \cdot (\mathbf{n}_\parallel \times \mathbf{n}_\perp)\rvert$	95%
Coulomb failure criteria total, $\mu = 0.4$	$\Delta CFS_{total}(\chi, \mu)$	$\Delta CFS_{total}(\chi, \mu) = \lvert(\mathbf{n}_\perp \cdot \chi) \cdot \mathbf{n}_\parallel\rvert + \lvert(\mathbf{n}_\perp \cdot \chi) \cdot (\mathbf{n}_\parallel \times \mathbf{n}_\perp)\rvert - \mu(\mathbf{n}_\perp \cdot \chi) \cdot \mathbf{n}_\perp$	93%
Sum of magnitudes of stress components	$m(\Delta\chi)$	$m(\Delta\chi) = \lvert\Delta\chi_{xx}\rvert + \lvert\Delta\chi_{yy}\rvert + \lvert\Delta\chi_{zz}\rvert + \lvert\Delta\chi_{xy}\rvert + \lvert\Delta\chi_{xz}\rvert + \lvert\Delta\chi_{yz}\rvert$	>99%

χ represents either the full (σ) or deviatoric (σ') stress-change tensor, χ_i are the corresponding eigenvalues, x_f and y_f are the x and y locations of the fault plane, respectively, and $\mathbf{n}_\perp$ and $\mathbf{n}_\parallel$ are the unit vectors perpendicular to the average orientation of the mainshock fault plane and parallel to the mean mainshock slip direction, respectively. %VE is the proportion of the variance in the strike-averaged neural-network forecast for the idealized strike-slip case (Fig. 2) that is explained by each strike-averaged physical metric. We include the largest %VE for each metric. For Coulomb failure stress change, the largest %VE corresponds to the magnitude of the Coulomb failure stress change associated with the full stress-change tensor $\lvert\Delta CFS(\sigma, \mu = 0.0)\rvert$. See Methods for details.

OUTLINES

1. Autoencoder
2. Variational autoencoder (VAE)
3. Research Paper Sharing
4. Discussions

Discussions

References

[1] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[2] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition,” *arXiv:1406.4729 [cs]*, vol. 8691, pp. 346–361, 2014.

[3] R. Girshick, “Fast R-CNN,” *arXiv:1504.08083 [cs]*, Apr. 2015.

[4] W. Liu *et al.*, “SSD: Single Shot MultiBox Detector,” *arXiv:1512.02325 [cs]*, vol. 9905, pp. 21–37, 2016.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016, pp. 779–788.

[6] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” p. 6.