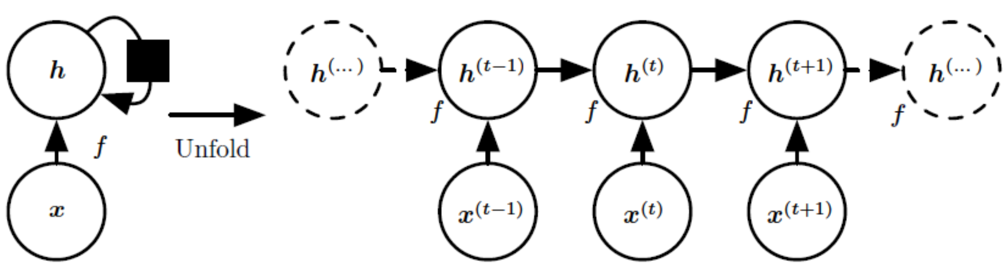
Lecture 10

Deep learning

1

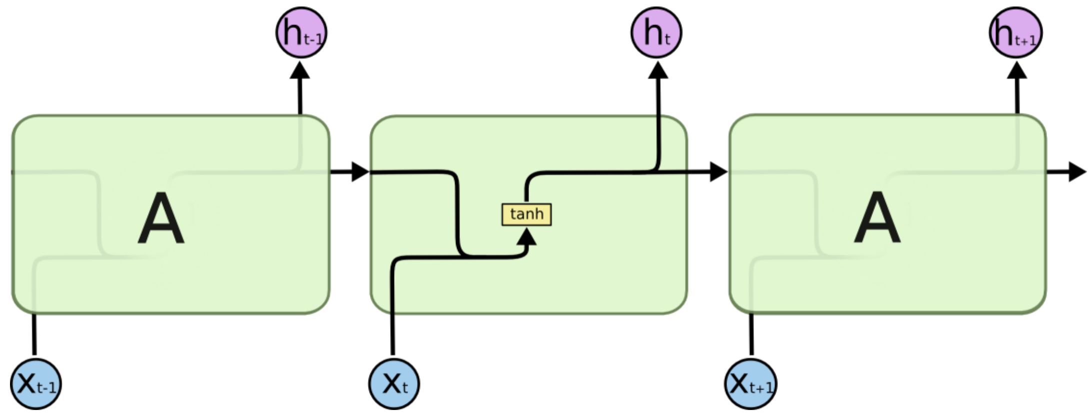
RNN



Goodfellow, Bengio, Courville 2016

2

RNN



Christopher Olah
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

3

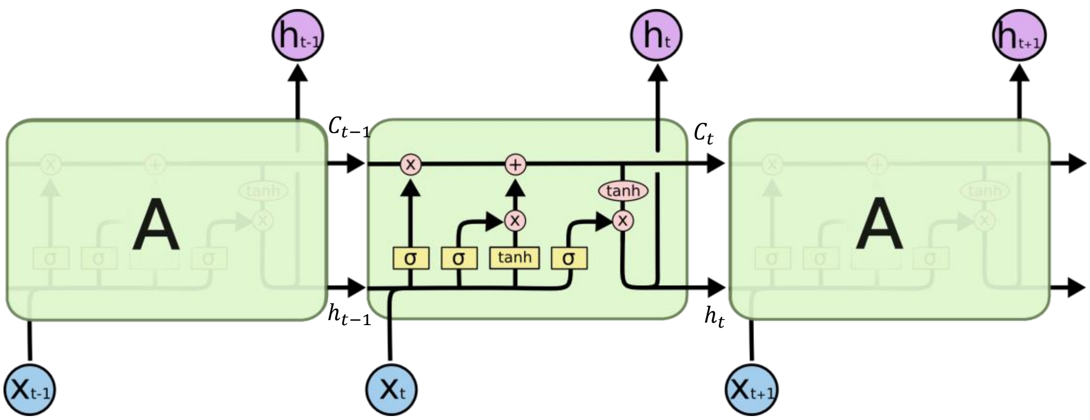
Long Short-Term Memory and Gated Recurrent Units

- Gated RNNs (including LSTM and GRUs) are based on the idea of creating paths through time that have derivatives that neither vanish nor explode.
- Also, Gated RNNs learns what to *forget* and what to keep in an automated manner rather than doing it manually.
- LSTM recurrent networks have "LSTM cells" that have an internal recurrence (a self-loop), in addition to the outer recurrence of the RNN.
- Each cell has the same inputs and outputs as an ordinary recurrent network, but has more parameters and a system of gating units that controls the flow of information.

Goodfellow, Bengio, Courville 2016

4

LSTM (Hochreiter & Schmidhuber (1997))

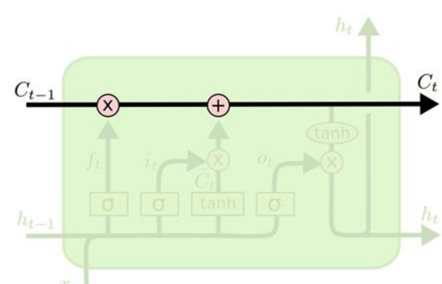


Christopher Olah
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

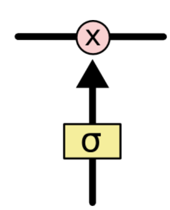
5

LSTM

- The key to LSTMs is the cell state, which the line that appears at the top of the diagram.
- The LSTM does have the ability to remove or add information to the cell states using a gated structure.
- Gates are a way to optionally let information through.
- They are composed out of a sigmoid neural net layer (outputs a number between zero and one that describes how much of each component should be let through) and a pointwise multiplication operation.



Cell state

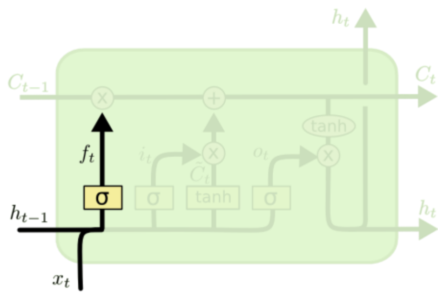


Christopher Olah
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

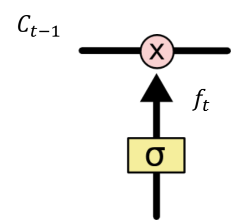
6

LSTM – Forget Gate (remove information from the cell state)

- Forget gate (f_t): a sigmoid gate that decides what information that we will throw away from the cell state.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



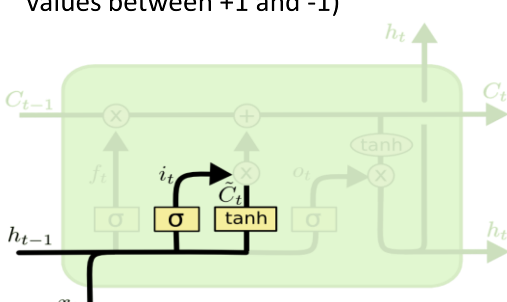
The sigmoid function produces f_t (a vector that has values between 0,1).

Christopher Olah
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

7

LSTM – Input Gate (Add information to the cell state)

- Input gate (i_t): a sigmoid gate that decides what new information we're going to store in the cell state.
- Two parts:
  - A sigmoid layer called the "input gate layer" decides which values we'll update (a vector that has values between 0,1 that will be multiplied by the tanh layer).
  - A tanh layer that creates a vector of new candidate values that can be added to the state (takes values between +1 and -1)



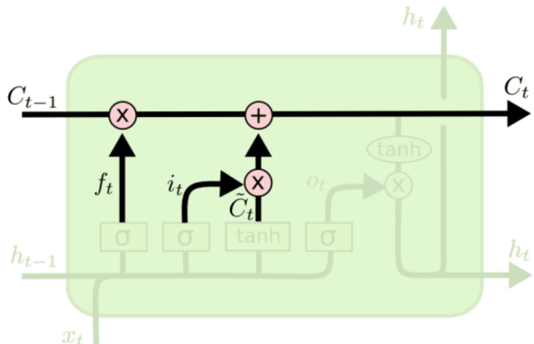
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Christopher Olah
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

8

LSTM

- Combine the two forget and input gates to produce C_t



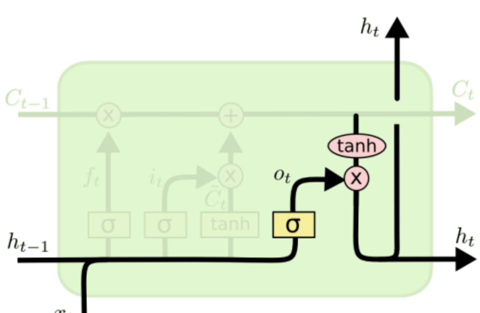
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Christopher Olah
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

9

LSTM - Output Gate

- Output Gate (o_t): A sigmoid gate that decides what parts of the cell state we're going to output.
- Put the cell state through tanh and multiply it by the sigmoid gate



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Christopher Olah
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

10

Variants on Long Short Term Memory

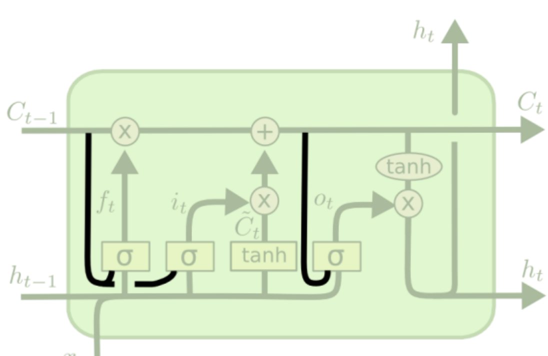
- Which pieces of the LSTM architecture are actually necessary?
- What other successful architectures could be designed that allow the network to dynamically control the time scale and forgetting behavior of different units?

Goodfellow, Bengio, Courville 2016

11

Variants on Long Short Term Memory

- Introduced by Gers & Schmidhuber (2000).
- Adding “peephole connections.” that allows the gate layers look at the cell state.



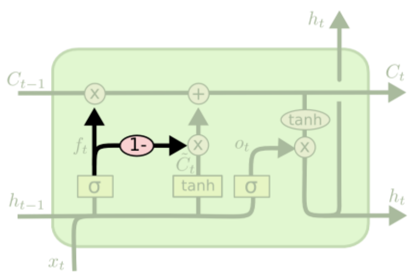
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \;+\; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \;+\; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \;+\; b_o\right)$$

Christopher Olah
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

12

Variants on Long Short Term Memory

- Another variation is to use coupled forget and input gates.
- Instead of separately deciding what to forget and what we keep, we make those decisions together.
- We only forget when we're going to input something in its place. We only input new values to the state when we forget something older.



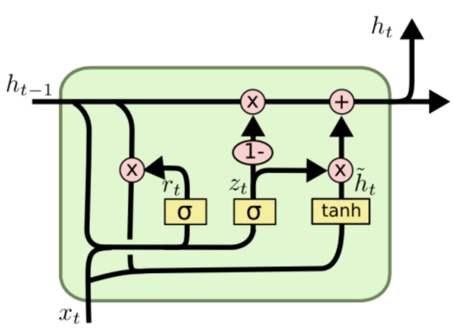
$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_t$$

Christopher Olah
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

13

Gated Recurrent Units: introduced by Cho et al. (2014)

- Simpler than standard LSTM models
- It merges the cell state and hidden state and combines the forget and input gates into a single "update gate."



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Christopher Olah
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

14

Unsupervised Learning Algorithms

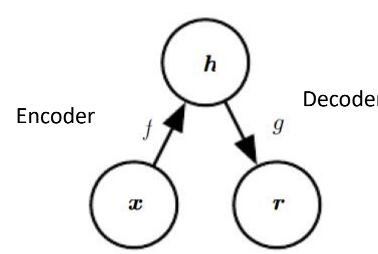
- Useful to learn useful properties about the structure of this dataset (e.g. learn the probability distribution that generated a dataset).
- Can be used in density function estimation or other tasks like denoising.
- Can perform other tasks such as clustering.

Goodfellow, Bengio, Courville 2016

15

Autoencoders (Chapter 14)

- An **autoencoder** is a neural network that is trained to attempt to copy its input to its output.
- Internally, it has a hidden layer *h* that describes a **code** used to represent the input.
- The network may be viewed as consisting of two parts: an encoder function $h = f(x)$ and a decoder that produces a reconstruction $r = g(h)$.



Goodfellow, Bengio, Courville 2016

16

Autoencoders

- Autoencoders are restricted in ways that allow them to copy only approximately, and to copy only input that resembles the training data.
- Typically, we would like to prioritize learning some useful aspects of the data (e.g. if your input is a noisy data, you would like your autoencoder to learn how to recover the original data)
- If an autoencoder succeeds in simply learning to set $g(f(x)) = x$ everywhere, then it is not especially useful.
- Traditionally, autoencoders were used for dimensionality reduction or feature learning. Recently, theoretical connections between autoencoders and latent variable models have brought autoencoders to the forefront of generative modeling,

Goodfellow, Bengio, Courville 2016

17

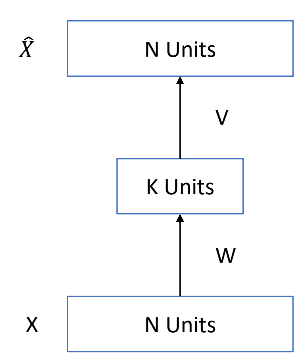
Autoencoders and PCA

- The simplest kind of autoencoder has one hidden layer, linear activations, and squared error loss

$$L(x,\hat{x}) = ||x - \hat{x}||^2$$

$$\hat{x} = \text{WVx (a linear function)}$$

- If K>=N, then we can choose WV that is the identity function
- If K<N, W maps x to a K-dimensional space, so it's doing dimensionality reduction
- The autoencoder should learn to choose the subspace which minimizes the squared distance from the data to the projections.
- Thus, it is equivalent to PCA which maximizes the variance of the projections.



Goodfellow, Bengio, Courville 2016

18

Difference between autoencoders and PCA

- In PCA, transformations are linear.
- When the decoder is linear and *L* is the mean squared error, an undercomplete autoencoder learns *to span the same subspace as PCA*.
- Autoencoders with nonlinear encoder functions *f* and nonlinear decoder functions *g* can learn a more powerful nonlinear generalization of PCA.
- If the encoder and decoder are allowed too much capacity, the autoencoder can learn to perform the copying task without extracting useful information about the distribution of the data.
- If the capacity of the autoencoder is allowed to become too great, an autoencoder can fail to learn anything useful about the dataset.
- Thus, *f* or *g,* in undercomplete autoencoders, typically has low capacity

Goodfellow, Bengio, Courville 2016

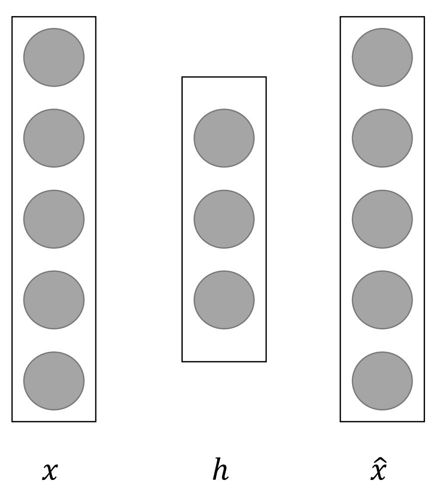
19

Undercomplete Autoencoders

- One way to obtain useful features from the autoencoder is to constrain *h* to have smaller dimension than *x*.
- An autoencoder whose code dimension is less than the input dimension is called **undercomplete**.
- Learning an undercomplete representation forces the autoencoder to capture the most salient (important) features of the training data.
- The learning process is described simply as minimizing a loss function

$$L(x, g(f(x)))$$

- where *L* is a loss function penalizing *g*(*f*(*x*)) is the decoder, f(x) is the encoder function.

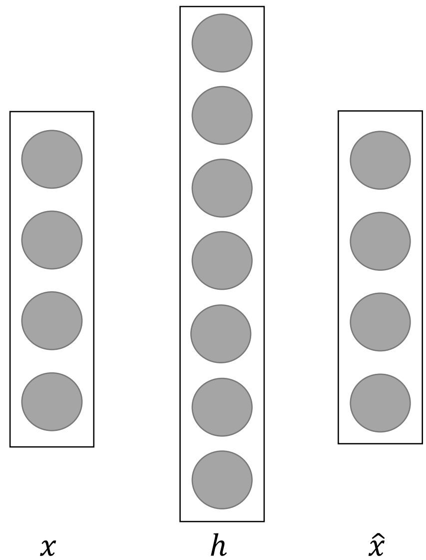


Goodfellow, Bengio, Courville 2016

20

Overcomplete Autoencoders

- An autoencoder whose code dimension is greater than the input dimension is called **overcomplete.**
- In case of overcomplete autoencoder, even a linear encoder and linear decoder may learn to copy the input to the output without learning anything useful about the data distribution.
- Must be regularized



Goodfellow, Bengio, Courville 2016

21

Regularized Autoencoders

- Regularized autoencoders provide the ability to choose the decoder based on the complexity of distribution to be modeled.
- Rather than limiting the model capacity by keeping the encoder and decoder shallow and the code size small, regularized autoencoders use a loss function that encourages the model to have other properties besides the ability to copy its input to its output.
- These other properties include sparsity of the representation (sparse autoencoders), robustness to noise or to missing inputs (denoising autoencoders), and smallness of the derivative of the representation (Contractive autoencoders).
- A regularized autoencoder can be nonlinear and overcomplete but still learn something useful about the data distribution even if the model capacity is great enough to learn a trivial identity function.

Goodfellow, Bengio, Courville 2016

22

Sparse Autoencoders

- Typically used to learn features as a pre-processing for another task such as classification.
- A sparse autoencoder is simply an autoencoder whose training criterion involves a sparsity penalty $\Omega(h)$ on the code layer h, in addition to the reconstruction error:

$$L\left(x, g\left(f(x)\right)\right) + \Omega(h)$$

$$\Omega(h) = \lambda \sum_i |h_i|$$

- We can think of the penalty $\Omega(h)$ simply as a regularizer term added to a feedforward network whose primary task is to copy the input to the output (unsupervised learning objective) and possibly also perform some supervised task (with a supervised learning objective) that depends on these sparse features.

https://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf

Goodfellow, Bengio, Courville 2016

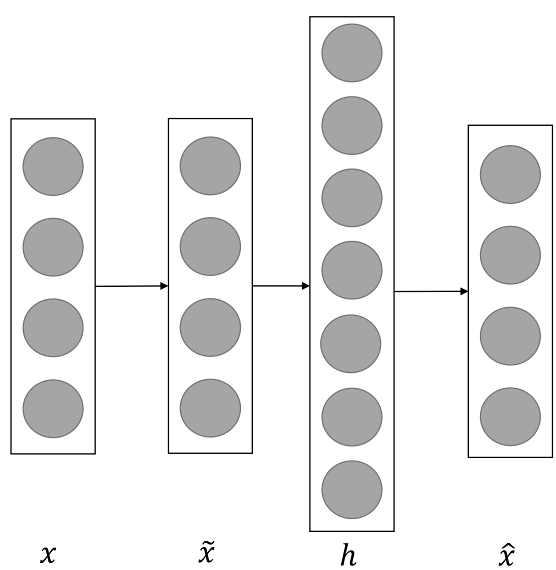
23

Denoising Autoencoders (DAE)

- Instead of feeding the original input x we feed a noise version of it $\tilde{x}$.
- A **denoising autoencoder** or DAE minimizes

$$L\left(x, g\left(f(\tilde{x})\right)\right)$$

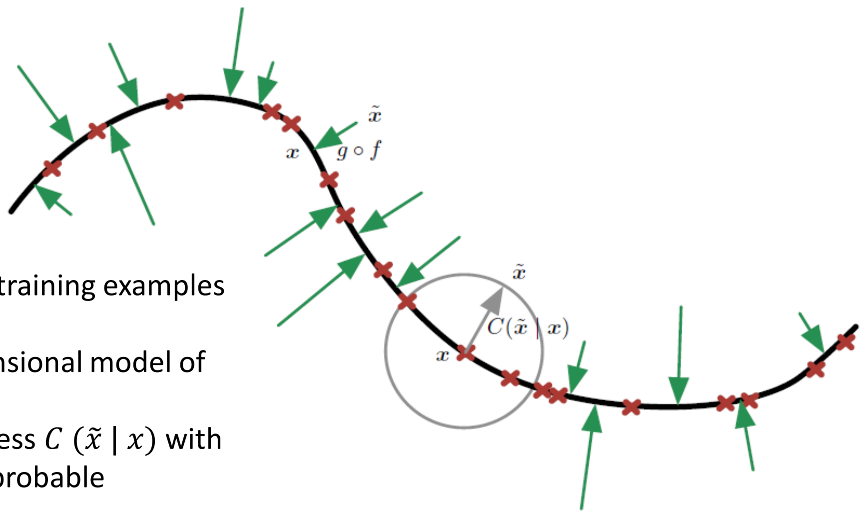
- Reconstruction $\hat{x}$ is computed from the corrupted input $\tilde{x}$
- Loss function compares $\hat{x}$ reconstruction with the noiseless input x



Goodfellow, Bengio, Courville 2016

24

Denoising autoencoder



- Red crosses are the training examples **x.**
- Black line low-dimensional model of the autoencoder
- The corruption process $C\ (\tilde{x} \mid x)$ with a gray circle of equiprobable corruptions.

Goodfellow, Bengio, Courville 2016

25

Contractive Autoencoders (CAE)

- Contractive Autoencoders forces the model to learn a function that does not change much when *x* changes slightly

$$L\left(x, g\big(f(x)\big)\right) + \Omega(h, x)$$

$$\Omega(h, x) = \lambda \sum_i \|\nabla_x h_i\|^2$$

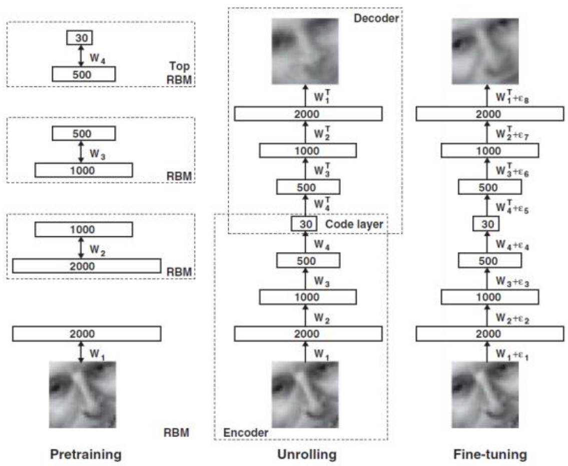
- Features that are sensitive to small changes in the inputs are penalized.
- The name **contractive** arises from the way that the CAE warps space.
- Specifically, because the CAE is trained to resist perturbations of its input, it is encouraged to map a neighborhood of input points to a smaller neighborhood of output points.

Goodfellow, Bengio, Courville 2016

26

Deep Autoencoders

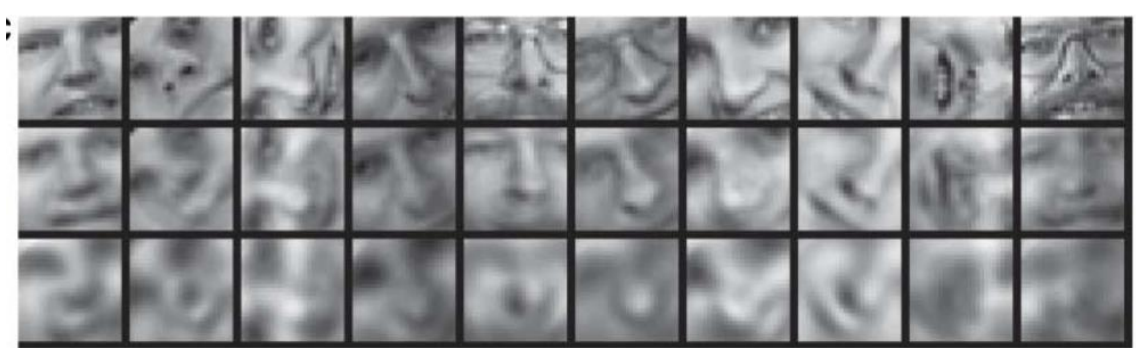
- “Reducing the Dimensionality of Data with Neural Networks” by G. E. Hinton* and R. R. Salakhutdinov



27

Deep Autoencoders

- “Reducing the Dimensionality of Data with Neural Networks” by G. E. Hinton* and R. R. Salakhutdinov



Top to bottom: 1) Random samples from the test data set; 2) reconstructions by the 30-dimensional autoencoder; 3) reconstructions by 30-dimensional PCA. The average squared errors are 126 and 135.

28