Machine Learning for Economists: Neural Networks and Deep Learning

Gentle Introduction

International Monetary Fund

Washington, D.C.,
October, 2018

Disclaimer #1:

The views expressed herein are those of the authors and should not be attributed to the International Monetary Fund, its Executive Board, or its management.

Outline

1. What are 'Neural Networks' and a bit of history...
2. Feed-forward and Recurrent neural networks and example/hands-on
3. Deep Learning & Convolutional Neural Networks
4. LSTM, Attention, ...

This presentation is really just a tip of the iceberg... from far away.

Feed-Forward Networks

Neural Networks are **parametric** models of the form

$$\mathbf{y} = F\left(\sum_{k=1}^{K} \phi_k(\mathbf{x})\right), \qquad (1)$$

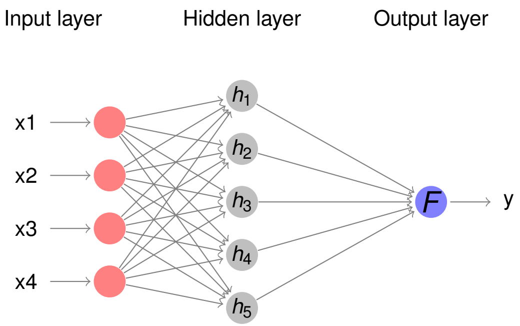
with a very flexible parametric specification of the basis functions, $\phi(.)$.

Neural networks can be complex compositions (networks) of small and simple elements: $f(x) = f^{(3)}[f^{(2)}\{f^{(1)}(x)\}]\ldots$

Neural networks are used for both regression and classification, with uses in non-supervised learning as well (auto-encoders)

Simple Feed-Forward Network

Simple Feed-Forward Network

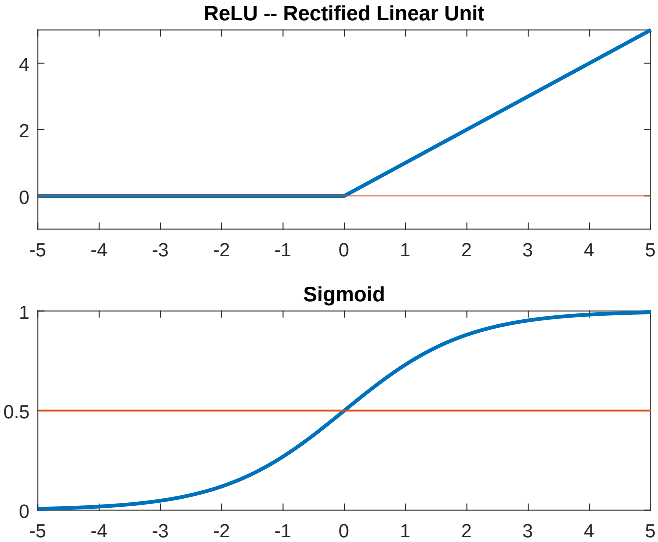


Functions F and g are **nonlinear activation** functions.

$$y = F(b_2 + w_{h,1}h_1 + w_{h,2}h_2 + \ldots + w_{h,5}h_5) \quad (2)$$

$$h_i = g(b_{i,1} + w_{i,1}x_1 + w_{i,2}x_2 + w_{i,3}x_3 + w_{i,4}x_4) \quad (3)$$

Activation Functions



Activation Functions

Activation functions must be non-linear.

Activation functions:

- **ReLU – Rectified Linear Unit**
  $g(z) = \max\{0, z\} \ \in [0, \infty)$
- **Logistic Sigmoid**
  $\sigma(z) = \frac{1}{1+e^{-z}} \in [0, 1]$
- **Hyperbolic Tangent**
  $g(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \in [-1, 1]$

ReLU is a great default cases for most uses. Unlike sigmoids it does not saturate and has large and consistent gradients...

For classifiers' output layer, use some sigmoid...

Universal Approximation Theorem and Learning

Universal approximation theorem says that a feed-forward network with at least one hidden layer equipped with a squashing/activation function can approximate arbitrarily well any* function, if it has enough hidden units...

In **practical** terms, nothing guarantees the training algorithm can learn the function...

Neural Network Architectures

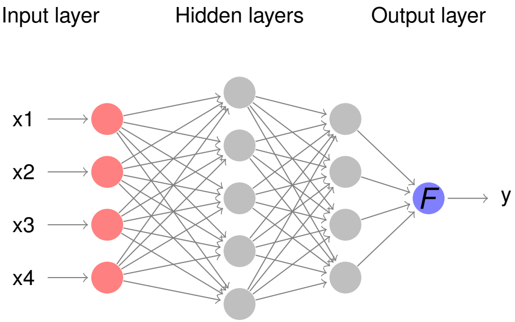
Depth of Networks:

- **'Shallow' networks**
  One or a few hidden layers
- **'Deep learning' networks**
  Cascade of many hidden layers, often deep convolutional networks...

Type of Feedback:

- **Feed-forward networks**
  $y = f(h), h = g(x)$
- **Recurrent networks**
  $y_t = f(h_t), h_t = g(h_{t-1}, x_t)$

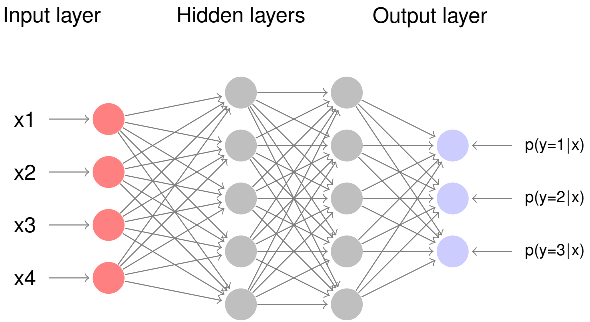
Simple Feed-Forward Network



Functions F and g are **nonlinear activation** functions.

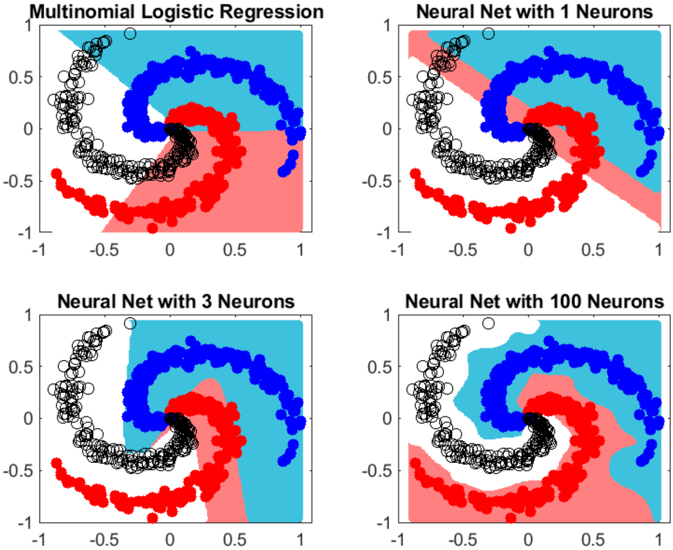
Feed-Forward Network with Multiple Outputs

Classification problem with three categories. . .

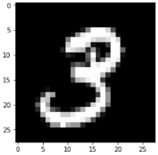


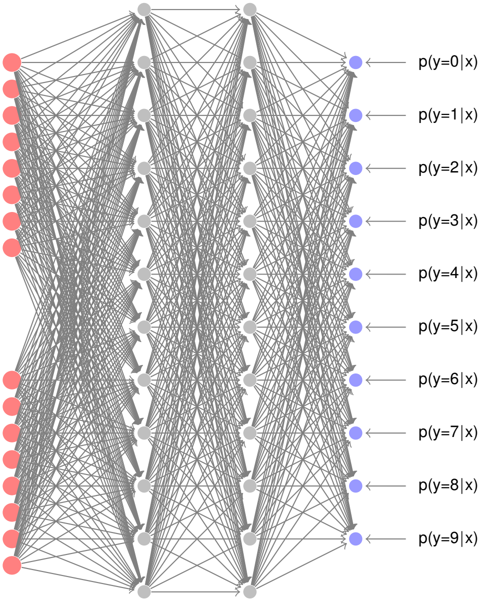
Functions F and g are **nonlinear activation** functions.

Classification Example



“Deep” Fully-Connected Classification Network





. . . things escalate pretty quickly, even for this **tiny toy** model

Training and Testing Neural Networks

Neural networks are a parametric function $\mathbf{y} = f(\mathbf{x}|\mathbf{w})$, sort of a 'non-linear regression'

Given available data $\{x_i, y_i\}$, numerically **loss function**, L, by searching over parameters, $\mathbf{w}$.

Now, given the flexibility of the net and huge amount of parameters, minimizing the loss function, $L(\mathbf{y}, \mathbf{y}^{obs})$ can be difficult...

It is very easy to **over-fit** with neural networks, so it is important to keep an eye on **generalization properties** (validation, testing) and focus on **regularization**

Training and Testing Neural Networks

Most large-scale neural nets (**deep learning**) are complex and trained on big data datasets. . .

It is feasible to split data into test, validation, and training samples

For smaller models and datasets, cross-validation could be used of course but it is numerically expensive. . .

Training and Testing Neural Networks

Common regularization strategies:

- **Penalty function** – ridge and sparsity penalty, ...
  $\min L(\mathbf{y}, \mathbf{y}^{obs}) + \lambda_r \sum (w_i - 0)^2 + \lambda_{L1} || \sum w_i ||$
- **Early stopping**
  Stop learning when the hold-out set performance using various criteria is good
- **Dropout**
  Similar to bagging, versions of network are trained with some pathways between neurons randomly eliminated.
- **Data augmentation, Parameter Sharing**
  both super-important in deep learning
- ...

Training Neural Networks

To minimize the loss function and estimate the parameters, an **optimization algorithm** is needed...

Neural network training has a few peculiarities to know about:

- **Batch learning**

  Given the cost of evaluating current parameters for all observations, only random portions (batches) are used each time

- **Gradient descent learning:** $\theta^{(n+1)} = \theta^{(n)} + \Delta_\theta Loss$

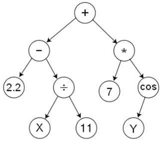
  In practice, optimization relies mostly on gradient descent methods, not using information in Hessians

- **Backpropagation**

  To obtain the gradient $\Delta_\theta Loss = \Delta_\theta \sum_i L(f(x^{(i)}; \theta, y^{obs})$, the derivatives are computed using the back-propagation algorithm, essentially a *chain rule*.

Wonkish: Backpropagation

- At times, backpropagation is sold as magic...
- Backpropagation is a method to compute efficiently the derivatives of the loss function w.r.t coefficients, $\partial L/\partial w_{i,j}$
- All the ingredients for the gradient (derivatives) are computed while the neural network is being evaluated...
- Modern toolkits for Neural Nets let users to specify symbolic *computational graphs* of operations to effectively optimize network structure and distributed computations



$$\left(2.2 - \left(\frac{X}{11}\right)\right) + \left(7 * \cos(Y)\right)$$

Wonkish: Backpropagation Intuition

Simple regression model with one hidden layer of two neurons and two inputs:

$$
\begin{aligned}
Loss &= \sum_{i=1}^{N} L_i, \qquad L_i = \frac{1}{2}(y_i - y_i^{obs})^2 \equiv \frac{1}{2}\delta_i^2 && (4)\\
y_i &= \alpha_1 h_{1,i} + \alpha_2 h_{2,i} + b && (5)\\
h_{1,i} &= \sigma(z_{1,i}), \qquad z_{1,i} = w_{11} x_{1,i} + w_{12} x_{2,i} && (6)\\
h_{2,i} &= \sigma(z_{2,i}), \qquad z_{2,i} = w_{21} x_{2,i} + w_{22} x_{2,i} && (7)\\
\sigma(z) &= \exp(z)/(1+\exp(z)) \qquad \partial\sigma(z)/\partial z = \sigma(z)(1-\sigma(z)) && (8)
\end{aligned}
$$

Now, using the economist' best friend, the chain-rule of differentiation, we can write:

$$
\begin{aligned}
\frac{\partial L_i}{\partial w_{11}} &= \delta_i \frac{\partial y_i}{\partial w_{11}} \rightarrow \frac{\partial y_i}{\partial w_{11}} = \alpha_1 \frac{\partial h_{1,i}}{\partial w_{11}} \rightarrow && (9)\\
\frac{\partial h_{1,i}}{\partial w_{11}} &= \frac{\partial \sigma(z_{1,i})}{\partial z_{1,i}} \frac{\partial z_{1,i}}{\partial w_{11}} = \sigma(z_{1,i})(1-\sigma(z_{1,i})) \times \frac{\partial z_{1,i}}{\partial w_{11}} && (10)\\
\frac{\partial z_{1,i}}{\partial w_{11}} &= x_{1,i} && (11)
\end{aligned}
$$

$$
\boldsymbol{\frac{\partial \mathbf{Loss}}{\partial \mathbf{w_{11}}} = \sum_{\mathbf{i}=1}^{\mathbf{N}} \delta_{\mathbf{i}} \times \alpha_1 \times \sigma(\mathbf{z_{1,i}})(1-\sigma(\mathbf{z_{1,i}})) \times \mathbf{x_{1,i}}} \qquad (12)
$$

Architecture and Model Design

Most of the progress in neural networks in past decades is due to changing the network design (convolutions, ReLU, . . .) and regularization (drop-out, mini-batch normalization, . . .) rather than advances in optimization algorithms.

Recurrent Neural Networks

RNNs are suited for modeling sequential data (since 1986)

RNNs are very similar to state-space models frequently used in economics

$$\mathbf{y}_t = F(b_2 + V\mathbf{h}_t) \quad (13)$$

$$\mathbf{h}_t = g(b_1 + W\mathbf{h}_{t-1} + U\mathbf{x}_t) \quad (14)$$

with $\mathbf{h}_t$ being the hidden layer, $\mathbf{x}_t$ the input data, and $\mathbf{y}_t$ the observed output.

Most basic RNNs' gradients vanish on a short interval, the issue is to keep longer memory or attention span…

Gated RNNs and LSTM

LSTM – Long Short-Term Memory

Hochreiter and Schmidhuber, 1997

The dynamics of the hidden layer, $\mathbf{h}_t$ is richer, augmented by another internal memory unit, $\mathbf{s}_t$, driven by forgeting, input, and output layers...

LSTM can retain information about inputs longer, in a context-sensitive way, and are used in natural language processing and time-series analysis...

To some extent, it's like a dynamic model with time-varying parameters

Gated RNNs and LSTM

$$
\begin{aligned}
\mathbf{y} &= \sigma(\mathbf{W}_y \mathbf{h}_t) && (15)\\
h_{i,t} &= o_{i,t} \times \tanh(s_{i,t}) && (16)\\
s_{i,t} &= f_{i,t} s_{i,t-1} + in_{i,t} \bar{s}_{it} && (17)\\
& && (18)
\end{aligned}
$$

$$
\begin{aligned}
f_{i,t} &= \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_h \mathbf{h}_{t-1}) && (19)\\
in_{i,t} &= \sigma(\mathbf{W}_{in} \mathbf{x}_t + \mathbf{U}_{in} \mathbf{h}_{t-1}) && (20)\\
\bar{s}_{i,t} &= \sigma(\mathbf{W}_{\bar{s}} \mathbf{x}_t + \mathbf{U}_{\bar{s}} \mathbf{h}_{t-1}) && (21)\\
o_{i,t} &= \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1}) && (22)
\end{aligned}
$$

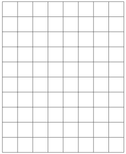
The internal state, $\mathbf{s}_t$ is driven by the past internal state, affected by *forgetting*, $\mathbf{f}_t$, and the 'news channel', $\bar{\mathbf{s}}_t$ with input importance, $\mathbf{in}_t$. How much of the hidden state is used for output, is affected by $\mathbf{o}_t$.

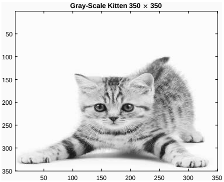
Convolutional Neural Networks

Convolutional neural networks are the dominant approach to image classification, style transfer, etc.

Specialized network design for data with a **grid-like topology**, where feature location is relevant...

- Digital images (rows $\times$ columns $\times$ color layer)
- Sound (time $\times$ frequency)
- ...





Earliest designs: Yann LeCun (1989+) and LeCun et al. (1998)

Convolutional Neural Networks

Convolutional Neural Networks

Convolution is an operation that intertwines two functions

$$s(t) = (x * w)(t) = \sum_{a=-K}^{K} x(a) \times w(t-a) \tag{23}$$

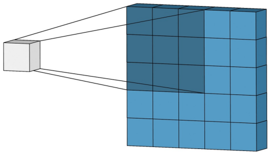
Convolution can have more than an axis, can 'combine' inputs across multiple dimensions...

Convolutions are used in frequency-domain analysis, filters, DFTs, wavelets, VARsetc.

Allows us to extract important location **features** from the data (e.g. edges, structures, ...)

Convolutional Neural Networks

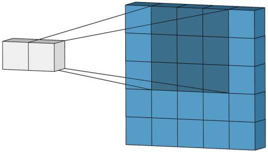
Simple convolution example:
New layer element as a weighted average the input layer



$$out_{1,1} = w_{1,1}x_{1,1} + w_{1,2}x_{1,2} + \cdots + w_{3,3}x_{3,3}$$

Convolutional Neural Networks

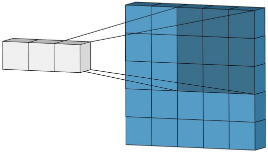
Simple convolution example:
New layer element as a weighted average the input layer



$$out_{1,2} = w_{1,1}x_{1,2} + w_{1,2}x_{1,3} + \cdots + w_{3,3}x_{3,4}$$

Convolutional Neural Networks

Simple convolution example:
New layer element as a weighted average the input layer

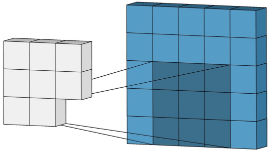


$$out_{1,3} = w_{1,1}x_{1,3} + w_{1,2}x_{1,4} + \cdots + w_{3,3}x_{3,5}$$

Convolutional Neural Networks

Simple convolution example:

New layer element as a weighted average the input layer

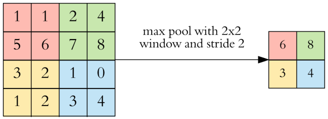


Convolutional Neural Networks

Pooling & Subsampling

After extracting convolution feature maps with *multiple* convolution setups, usually some **sub-sampling** is done

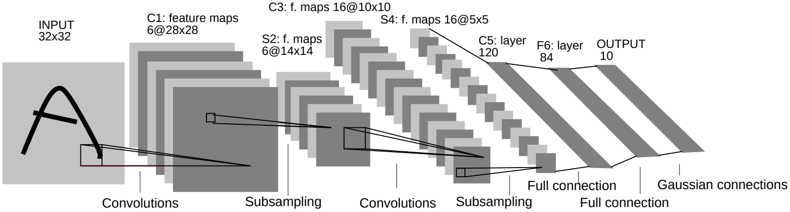
- it lowers the **dimensionality** of the network
- helps with **invariance** to local translation
- allows to handle **varying size** inputs

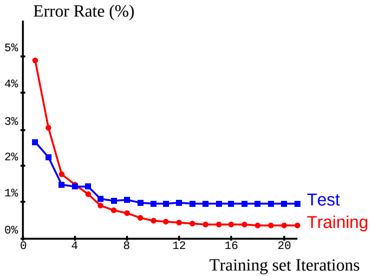


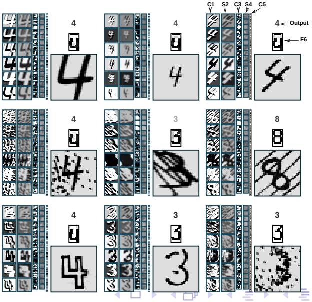
Common operations:
max pooling, mean pooling, L^2 norm, distance from center, ...

Convolutional Neural Networks – LeNet-5

Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner:
Gradient-Based Learning Applied to Document Recognition, Proc. of the IEE, **November 1998**





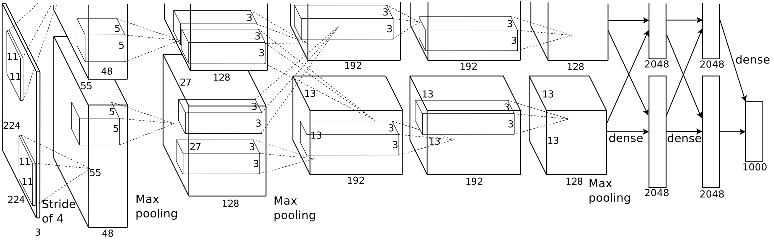


Millions of Coefficients. . .

The success today depends of **large-scale** and **deep** convolutional networks with millions of coefficients. . .

In 2012, with a model now known ‘AlexNet’, Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton used deep neural net to beat other methods in image classification, starting a next wave of the revolution. . .

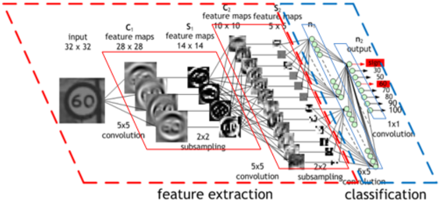
AlexNet has **60 million parameters** and **650,000 neurons**, 5 convolutional layers, three full-connected layers to classify 1000 classes

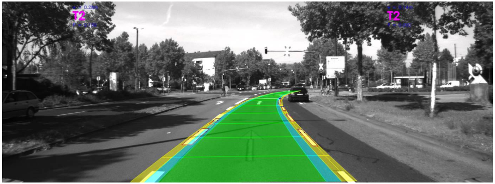


Big Data. . .

To train large-scale CNNs, huge amount of data is needed, often visual, sound, or text. . . **You'd better have lots of corn!**

. . . and hope your car is well-trained!





Style Transfer

Super simplified...since 2015 the literature exploded.

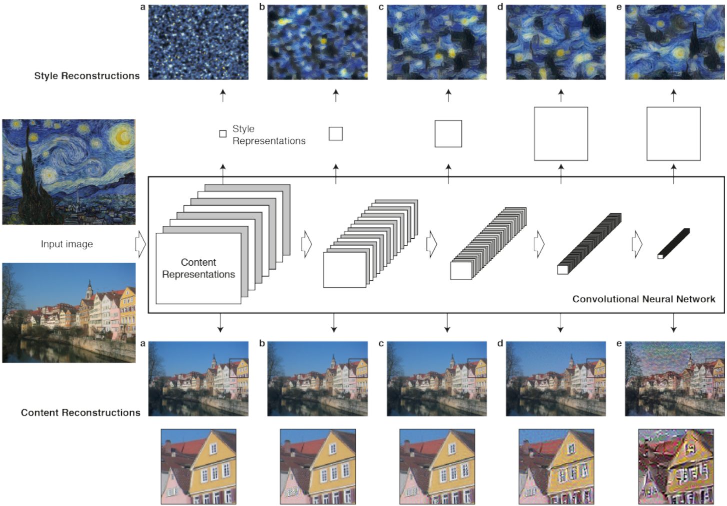
Style transfer research breakthrough by 2015 paper by L. Gatys, A. Ecker, and M. Bethge (GEB) from Tübungen, Germany.

Deep convolutional neural networks represent images in terms of many feature maps, differently filtered version of the image...

GEB show that each layer represents difference concpets, increasingly noting content and that **content** and **style** are separable.

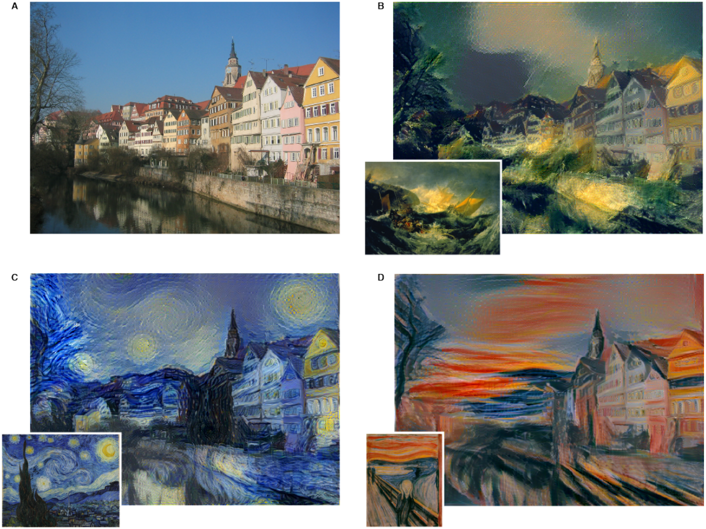
Style transfer then uses content information from one image and style information from another image.

Style Transfer



Gatys et al. (2015)

Style Transfer



Gatys et al. (2015)

AutoEncoders – Unsupervised Learning

Autoencoders are neural nets trained to best copy their inputs to their output...

The catch is, there is a **bottleneck** somewhere – the model must deconstruct the input into lower-dimensional object and re-construct as much as it can

Under-Complete Autoencoders are related to **principal component analysis** (PCA)

Find $f(x)$ and $g(z)$ such that

$$x \to f(x) = z \qquad \text{and} \qquad g(z) \to \widehat{x}$$

by minimizing $L(x, \widehat{x}) = L(x, g(f(x)))$.

AutoEncoders – Unsupervised Learning

Autoencoders are motivated to learn **important** features of x.