Deep Learning: A Statistical Perspective

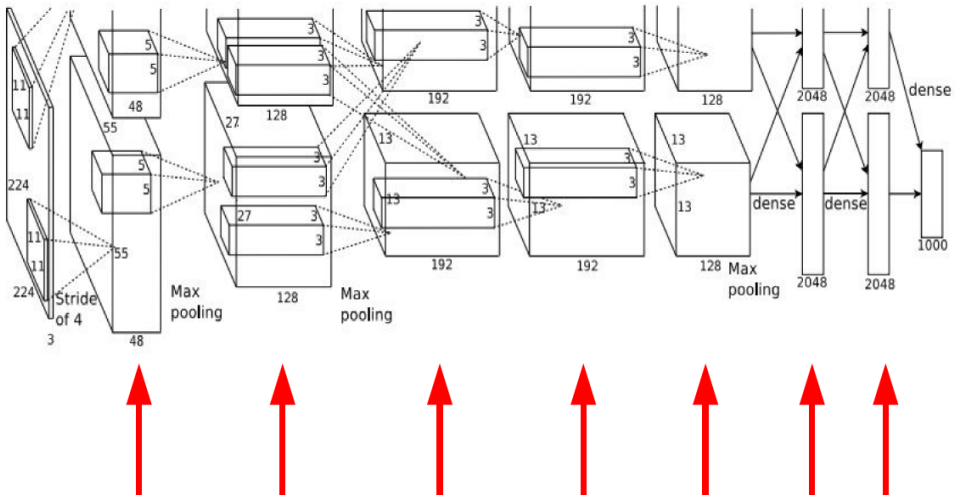
Myunghee Cho Paik
Guest lectures by Gisoo Kim, Yongchan Kwon, Young-geun Kim, Wonyoung Kim and Youngwon Choi

Seoul National University

March-June, 2018

Visualization

Overview of visualization

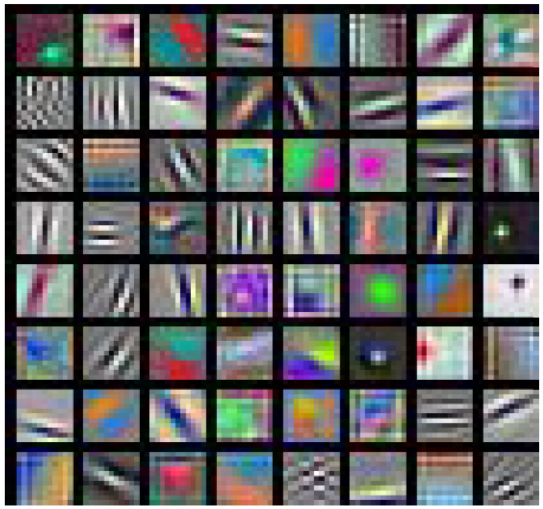


- Visualize filters in each layer
- Visualize randomly chosen feature maps or activation maps
- Visualize patches that activate particular neurons

Overview of visualization-continued

- Deconvolution: Map activation to input space using trained CNN.
  - Compute $\frac{\partial h}{\partial x_n}$ for a node (Zeiler and Fergus, 2013)
  - Compute $\frac{\partial \mathbf{h}}{\partial x_n}$ for a channel (Yosinski et al. 2014)
  - Compute $\frac{\partial h}{\partial x_n}$ for a node using modified approximation called Guided Backprop (Springenberg et al. ICLR 2015)
- Saliency maps (Simonyan Vedaldi and Zisserman, ICLR 2014): Compute gradient of class score with respect to image pixels. Saliency maps can be used for segmentation without supervision
- Occlusion (Zeiler and Fergus, ECCV 2014)
- Generate an image that maximally activate a neuron: Gradient ascent (Yosinski et al. ICML 2014, Nguyen et al. ICML 2016)

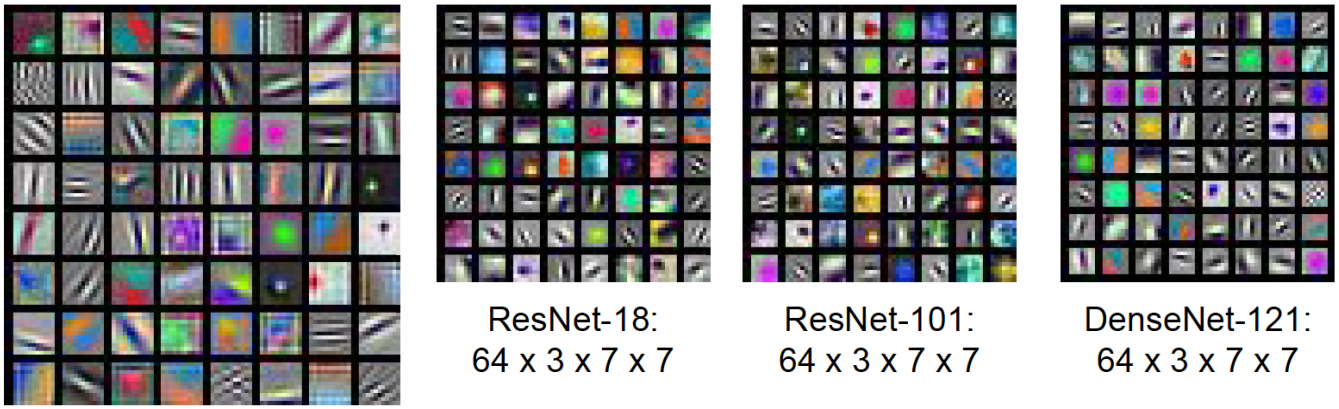
Visualizing filters: First layer



AlexNet:
64 x 3 x 11 x 11

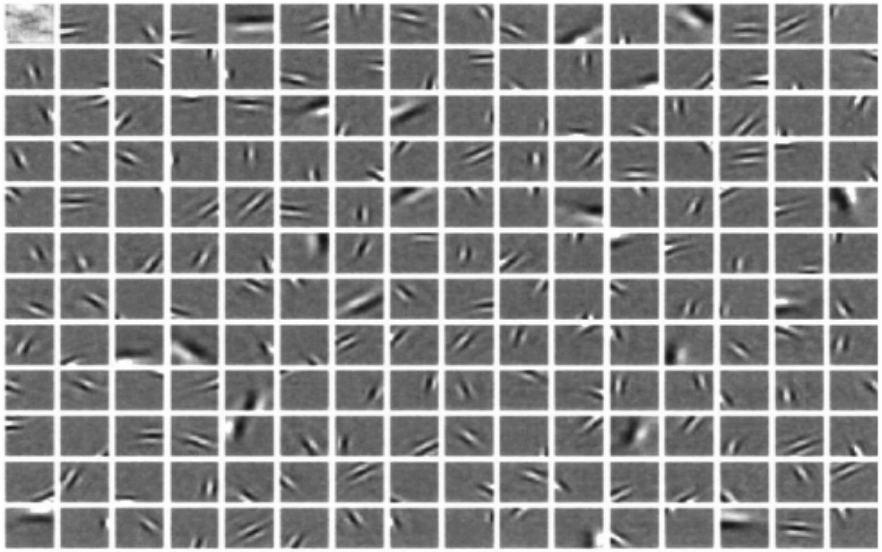
First layer filters show edges and color blobs
Fist layer filters from Alexnet are close to image descriptors such as SIFT and HOG or estimated filter through sparse codeing by Olshausen and Field (1996).

Visualizing filters: First layers from different CNN models



First layer filters from different models show edges and color blobs

First layers from non-neural networks



Filters obtained from sparse coding from Olshausen and Field (1999)

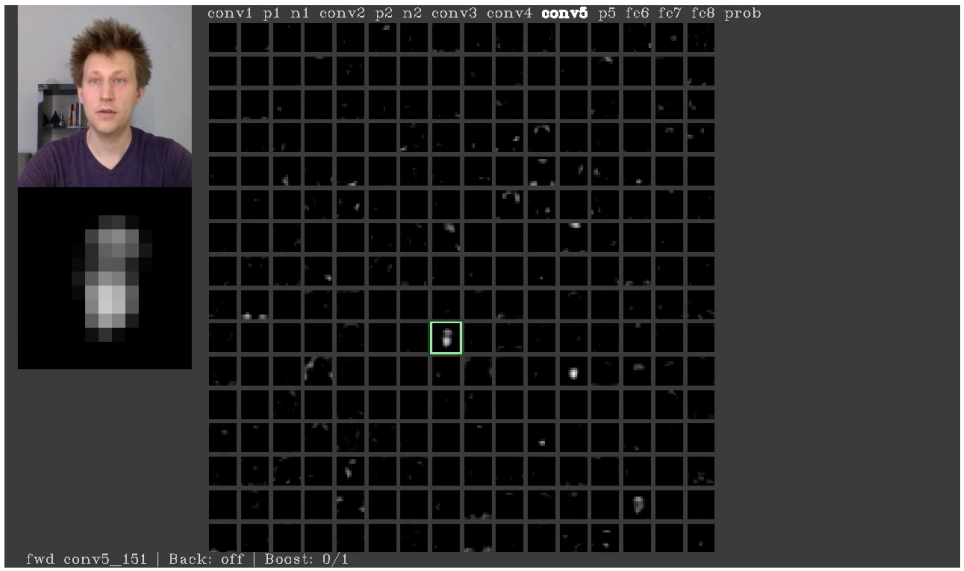
Visualizing filters: Other layers

Karparthy, ConvNetJS CIFAR-10 demo

First layer: 16x(5x5x3); Second layer: 20x(5x5x16); Third layer: 20x(5x5x20).

Hard to interpret the second and the third layers.

Visualizing activation maps



source: YouTube capture from Deep visualization Toolbox by Yosinski

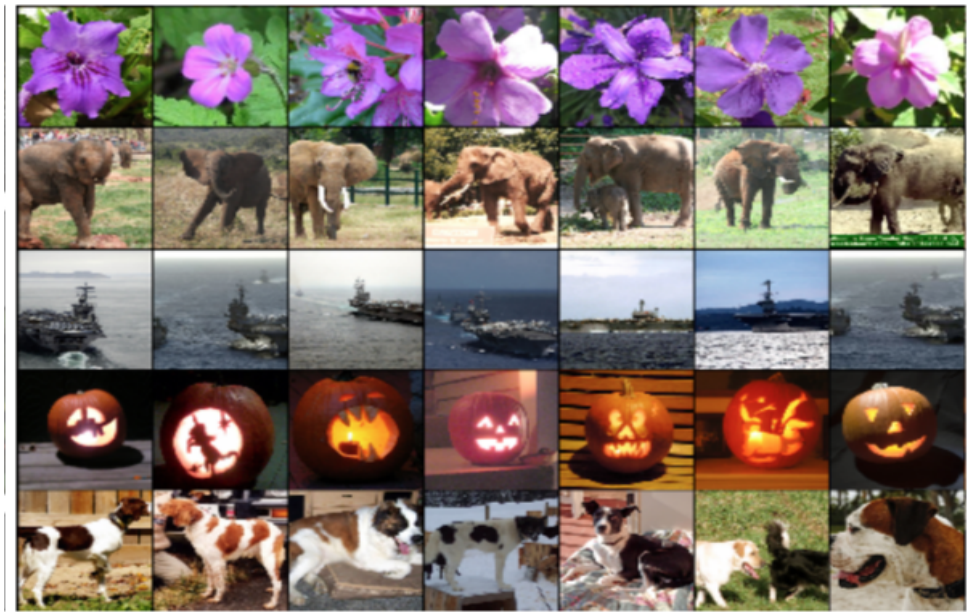
256x13x13 activation map from the 5th convolutional layer from AlexNet on the right and the original input on the left (Yoniski et al. ICML 2014)

Visualizing active maps

Figure 2. A view of the 13×13 activations of the 151st channel on the conv5 layer of a deep neural network trained on ImageNet, a dataset that does not contain a face class, but does contain many images with faces. The channel responds to human and animal faces and is robust to changes in scale, pose, lighting, and context, which can be discerned by a user by actively changing the scene in front of a webcam or by loading static images (e.g. of the lions) and seeing the corresponding response of the unit. Photo of lions via Flickr user arnolouise, licensed under CC BY-NC-SA 2.0.

A 13x13 activation map from the 5th convolutional layer on the right and the original input on the left (Yoniski et al. ICML 2014)

Visualizing the last layers using nearest neighbors

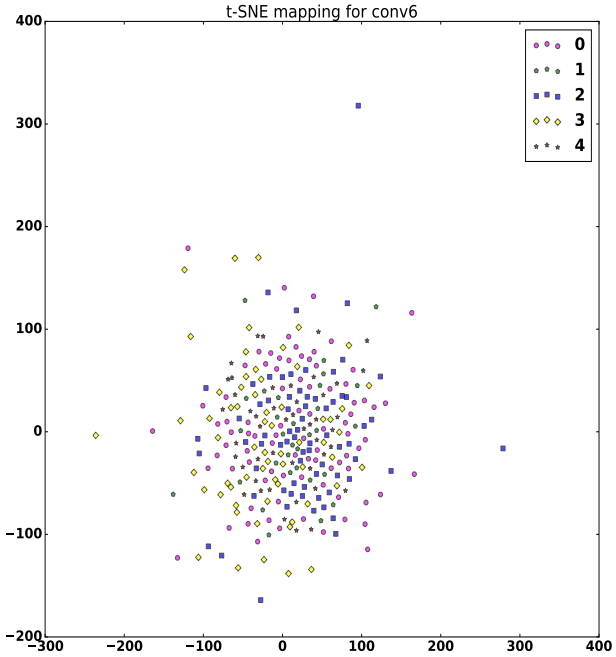


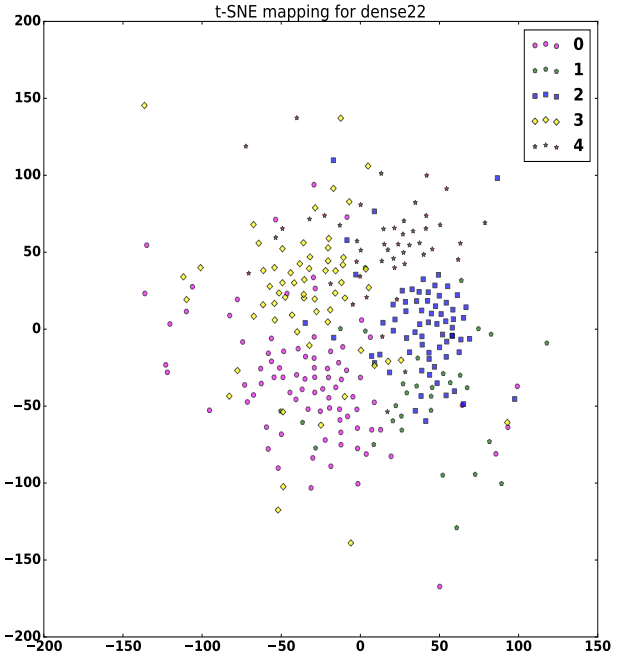
- AlexNet: Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image. (Krizhevsky et al. 2012)
- Images with the smallest Euclidean distance in raw input are not similar.

Visualizing through dimension reduction: t-Stochastic Neighborghood Embedding (t-SNE)

- A nonlinear dimensionality reduction technique
- Models each high-dimensional object by a two or three dimensional point in such a way that similar objects are closer.
- x: high dimensional original vector; y: low dimensional counterpart.
- Distance measure: $p_{j|i} = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-||x_i - x_k||^2/2\sigma_i^2)}$. σ_i is chosen to satisfy $H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}$ to have a user-specified value.
- $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$, $q_{ij} = \frac{(1+||y_i - y_j||^2)^{-1})}{\sum_{k \neq l}(1+||y_l - y_k||^2)^{-1}}$
- tSNE finds y that minimizes $\sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$
- tSNE can be applied to each layer

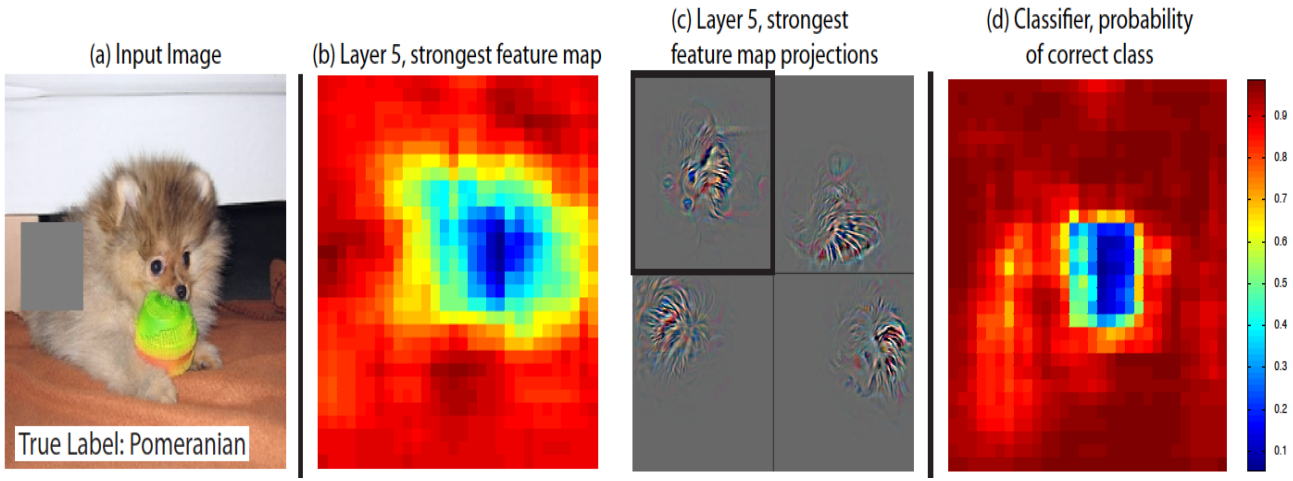
Visualizing filters using tSNE: intermediate layer vs. last layer





Figures provided by Yongchan Kwon using Kaggle Retinopathy data

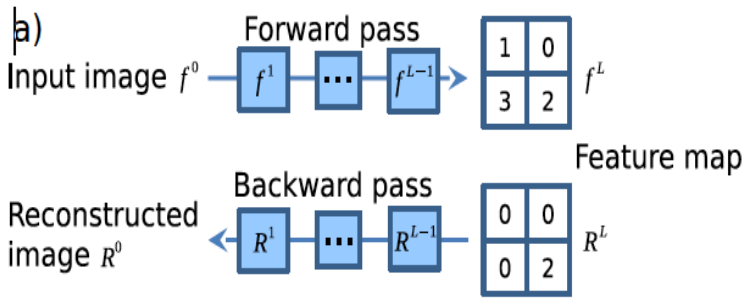
Occlusion



Systematically cover up different portions of the scene with a grey square (1st column) and see how the top (layer 5) feature maps, and classifier output changes. (Zeiler and Fergus, 2014)

Deconvolution (Zeiler and Feugus, 2013)

Goal: Identify input pattern causing activation of neurons
Maps from activation map to input pixel space. DeconvNet approximately inverts the Convnet operations, i.e., convolution, pooling, and rectification



source: Springenberg, Dosovitskiy, Brox and Riedmiller, 2015

Approximately compute $\frac{\partial f_k}{\partial x_i}$

Deconvolution: Convolution layer

- Let X be vectorized input voxel intensities, h be a neuron from an activation map, and β be rearrnaged filter. Convolutional layer gives

$$h = \beta X.$$

- DeConvNet of convolutional layer involves

$$\frac{\partial h}{\partial X} = \beta^T,$$

that is to use the transposed version of learned filters.

Deconvolution: Rectifying layer

- Goal: Project f_i^k, the i^{th} neuron in layer k, to input space is of our interest. For the l^{th} layer, we have

$$f_i^{l+1} = \mathrm{Relu}(f_i^l) = \max(f_i^l, 0).$$

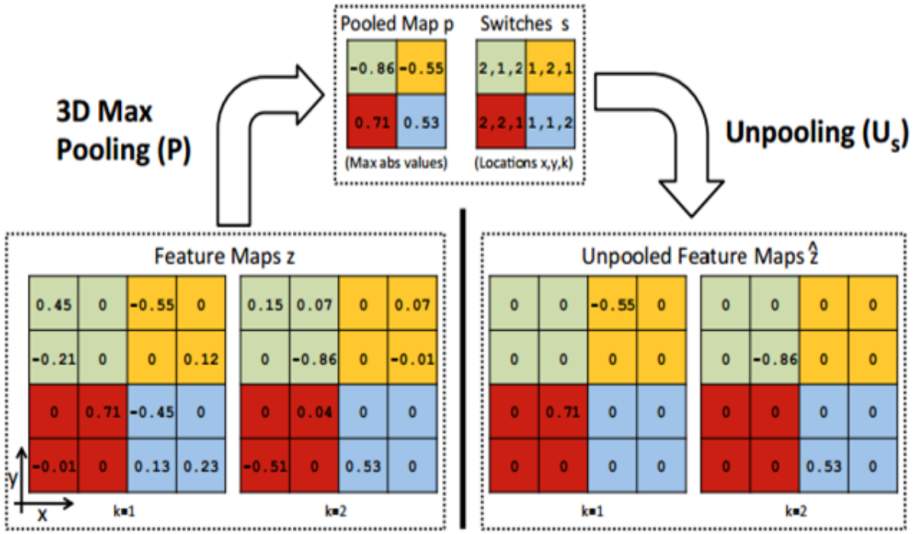
- Deconvolution involves the following operation:

$$R_i^l = I(f_i^l > 0) R_i^{l+1}$$

where $R_i^{l+1} = \frac{\partial f_i^k}{\partial f_i^{l+1}}$

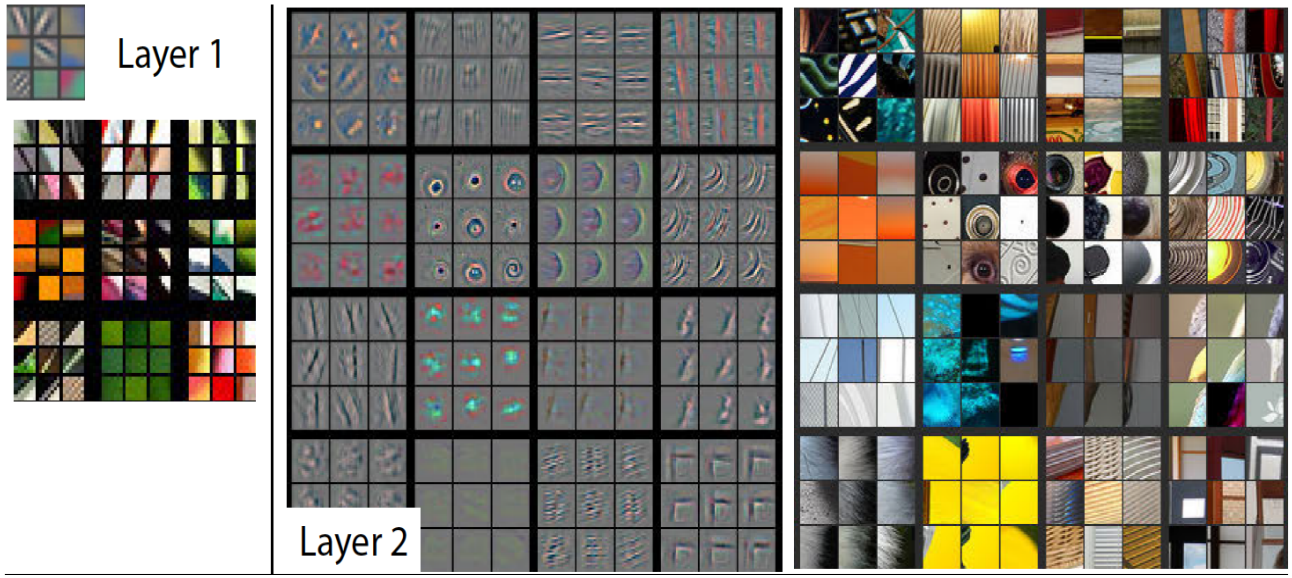
- Rectify feature maps so that feature maps are always positive.

Deconvolution Unpooling



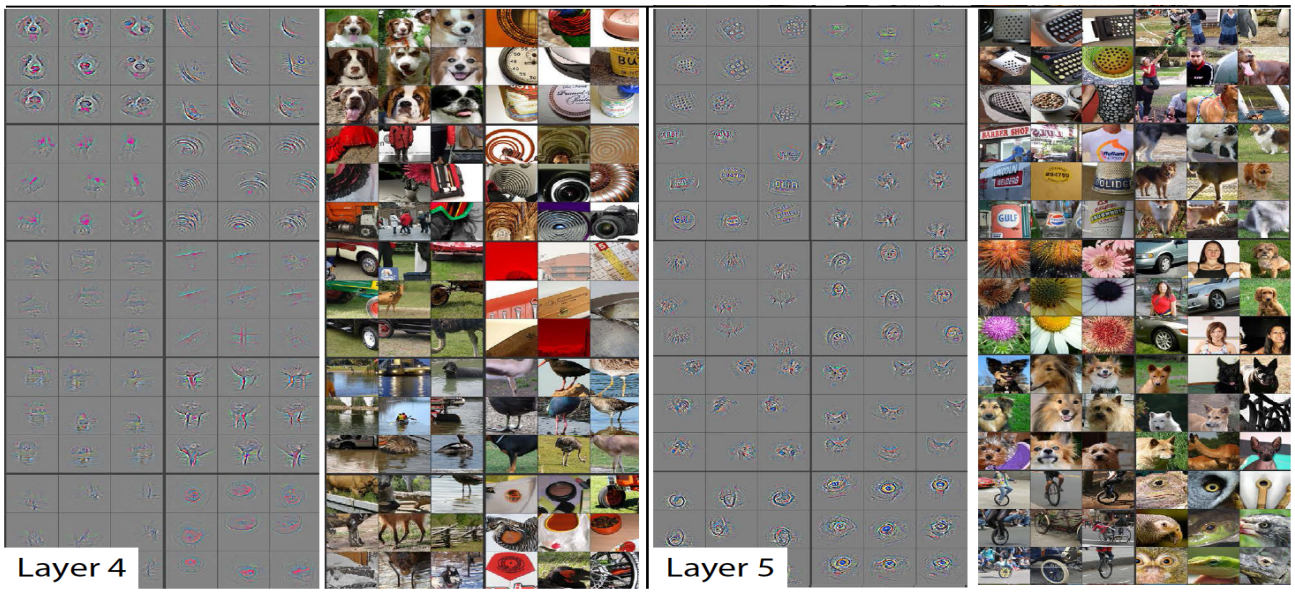
Pooling: Record switch that is the position where a neuron is activated

Visualizing deconvoluted images



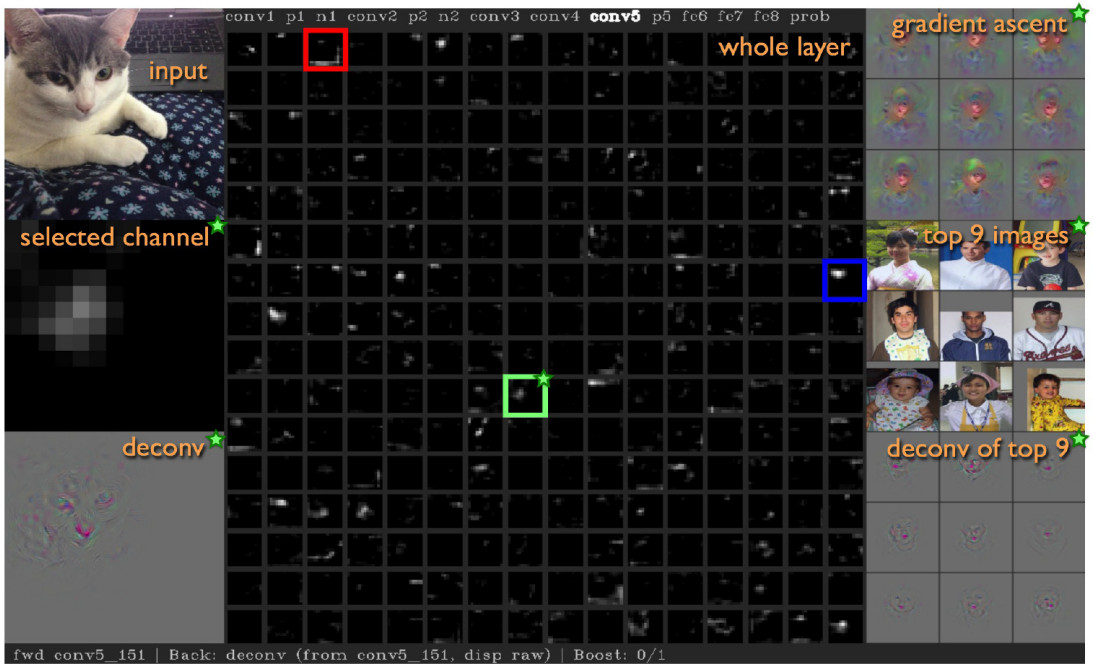
From a random subset of activation maps, pick 9 top activated neurons from validation imgages.
Projection onto the input space is shown.
The image patches corresponding to the top activated neurons are shown.

Visualizing deconvoluted images



Neurons from higher layers show abstract images and patches cover a wider region.

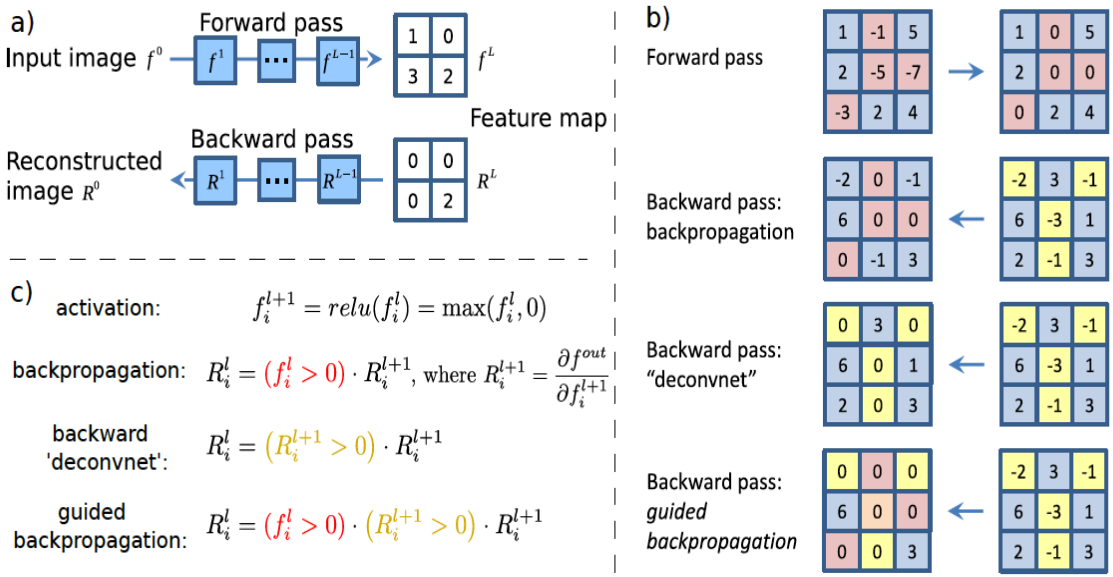
Deconvolution of activation map



- Compute $\frac{\partial \mathbf{h}}{\partial x_i}$ for each pixel for each node of a particular channel (feature map) and average over nodes in the channel (Yoniski et al., 2014).

Guided backprop

Modified inversion for rectified layer



source: Springenberg, Dosovitskiy et al. 2015

- Block negative derivatives flow backward.

Results of guided backprop

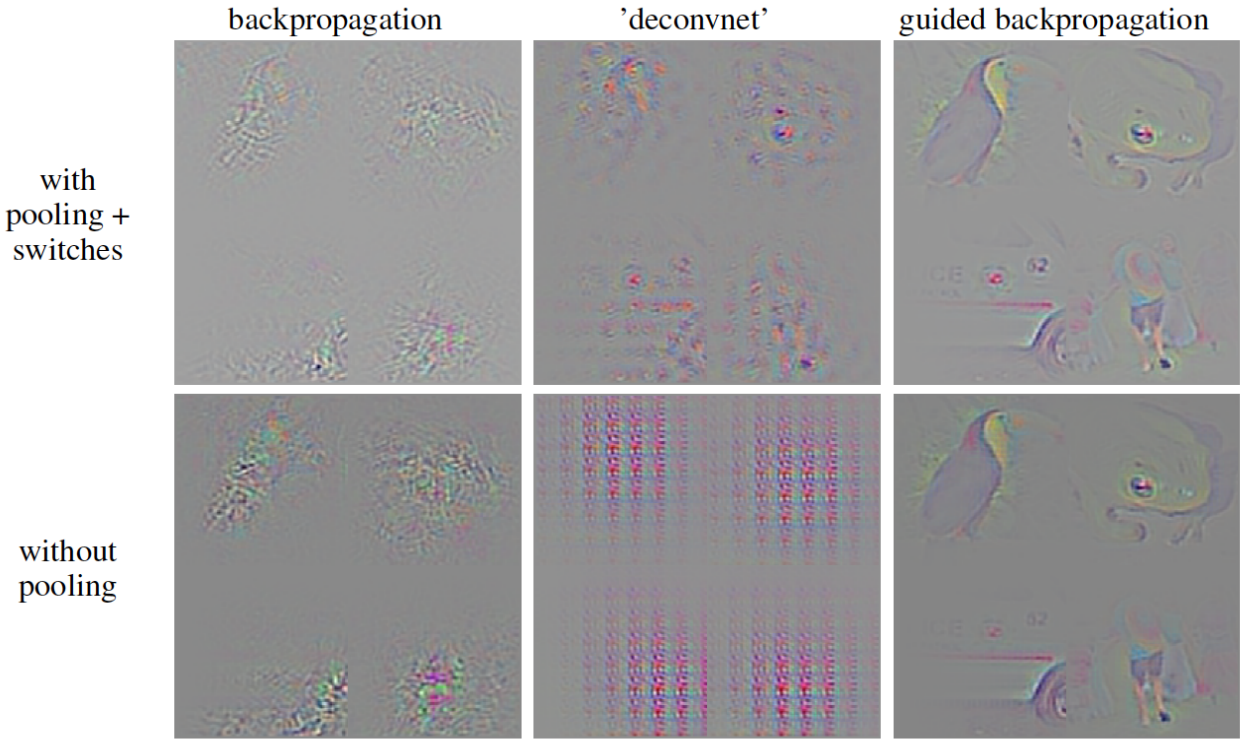


Figure 4: Visualization of descriptive image regions with different methods from the single largest activation in the last convolutional layer conv12 of the network trained on ImageNet. Reconstructions for 4 different images are shown.

source: Springenberg, Dosovitskiy et al. 2015

Guided backprop produces clearer images than the original backpropagation.

Saliency map

- Let $S_c(X)$ be the score before the softmax for the class c given the vectorized image pixel input, X. Compute $W = \frac{\partial S_c(X)}{\partial X}$ using back-propagation
- The saliency map is obtained by rearranging the elements of $\frac{\partial S_c(X)}{\partial X}$. Let this rearranged element be w.
- If input is multichannel (e.g. RGB), w has triple index, row, column and color, say (i, j, c). Then obtain $M_{ij} = max_c |w_{h(i,j,c)}|$.

source: Simonyan, Vedaldi and Zisserman, 2014

Generating Image: Gradient Ascent

- Goal is to find an image that maximizes some class score, the final layer values before softmax. Let $S_c(X)$ be the score given X.
- $$\arg\max_X (S_c(X) - \lambda \|X\|_2^2)$$

1. Initialize X. Forward pass X.
2. Set the gradient of loss with respect to S equal to one hot vector of y.
3. Backprop to the gradient to the image X node.
4. Update image. Go to 2.

Generating Image: Gradient Ascent

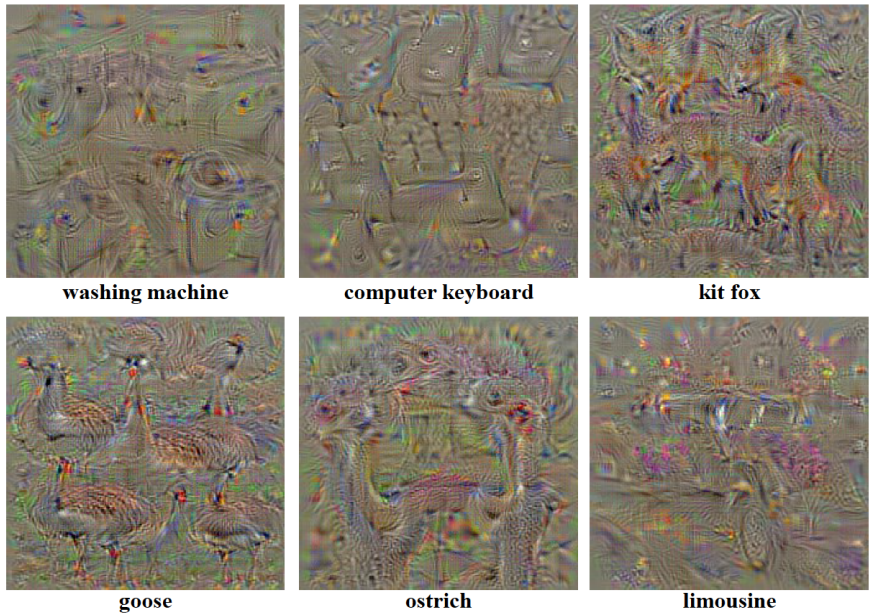
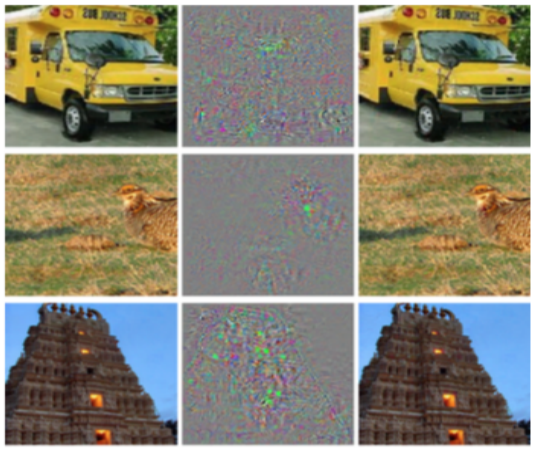


Figure 1: **Numerically computed images, illustrating the class appearance models, learnt by a ConvNet, trained on ILSVRC-2013.** Note how different aspects of class appearance are captured in a single image. Better viewed in colour.

Simonyan, Vedaldi and Zisserman, 2014

Yosinski et al. (2015) used each neuron instead of $S_c(X)$ with slight modification of loss function.

Adversarial examples



source: Szegedy, Zaremba, Sutskever et al., 2014

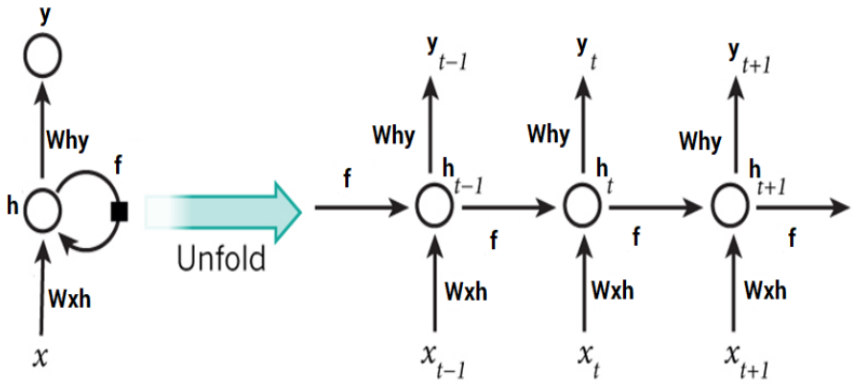
- Applying an imperceptible non-random perturbation to a test image, it is possible to arbitrarily change the network's prediction. These perturbations are found by optimizing the input to maximize the prediction error. (Szegedy, Zaremba, Sutskever et al., 2014)
- Minimize $c|r| + loss(x + r, y^*)$ subject to $x + r \in [0, 1]^m$
- Adversarial examples can be easily constructed for classical statistical model such as logistic regression.

Recurrent Neural Network

Recurrent Neural Network (RNN)

- Sequence Modeling: Recurrent neural networks, or RNNs (Rumelhart et al., 1986a), are a family of neural networks for processing sequential data.
- $p_t = softmax(c + Vh_t)$
- $h_t = tanh(b + wh_{t-1} + Ux_t)$
- Hidden unit at time t is a function of hidden unit at $t-1$ and the input at time t.
- Unknown parameters do not depend on time.
- Loss function: $L(\{x_1, \cdots, x_t\}, \{y_1, \cdots, y_t\})$

Building units of RNN

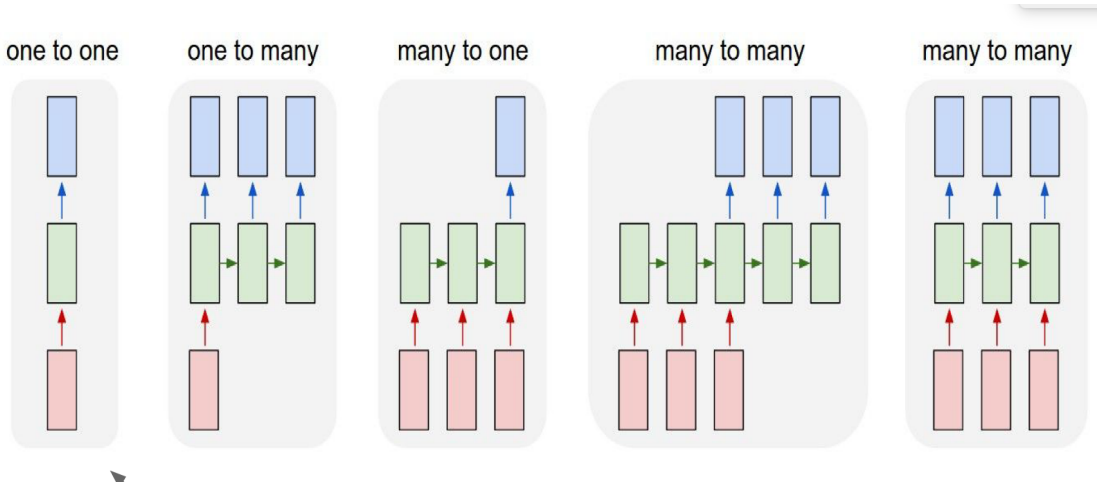


- Input: x_t
- Hidden unit: $h_t = tanh(b + wh_{t-1} + Ux_t)$
- Output unit: $o_t = c + Vh_t$
- Predicted probability: $p_t = softmax(o_t)$

Building units of RNN

- Input: x_t
- Hidden unit: $h_t = tanh(b + wh_{t-1} + Ux_t)$
- Output unit: $o_t = c + Vh_t$
- Predicted probability: $p_t = softmax(o_t)$
- Unknown parameters: (w, U, b, c, V)

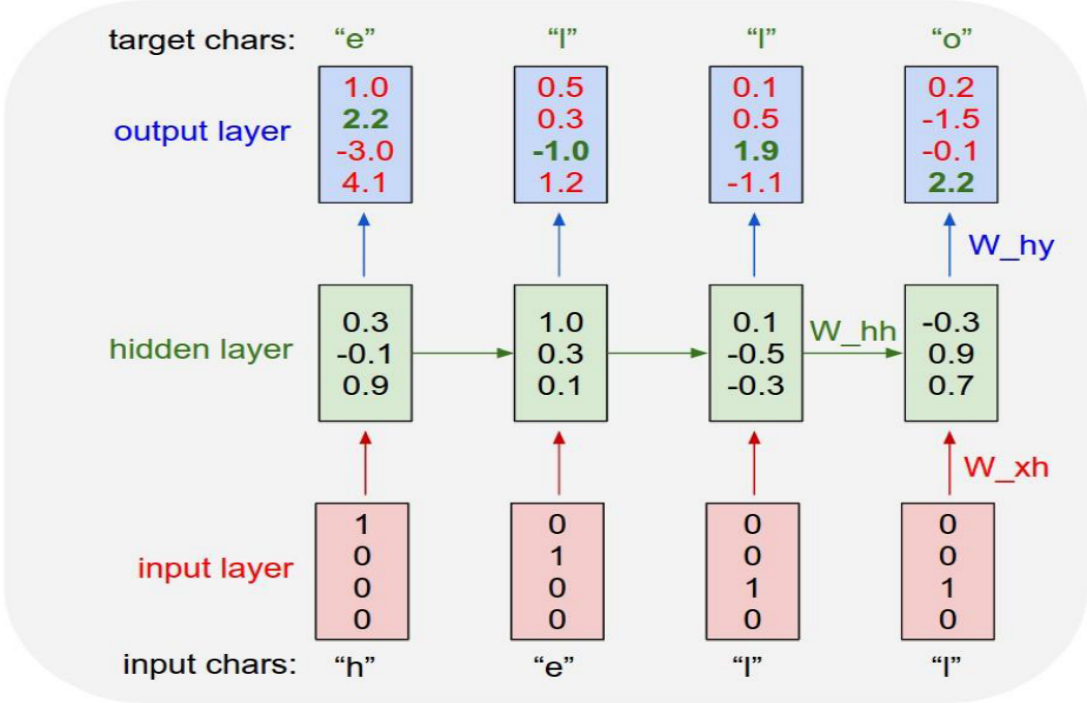
Applications of RNN



source: Andrej Karpathy blog

- one to one: typical
- one to many: image captioning (image to sequence of words)
- many to one: sentiment analysis (sequence of words to sentiment)
- many to many: with lag, machine translation (sequence of words to sequence of words); without lag, video classification on frame level

Applications of RNN: Character-level language models



source: Andrej Karpathy blog

$x_1 = (1, 0, 0, 0)$, $y_1 = (0, 1, 0, 0)$, $\hat{h}_1 = (0.3, -0.1, 0.9)$,
$\hat{o}_1 = (1.0, 2.2, -3.0, 4.2)$.

Image captioning

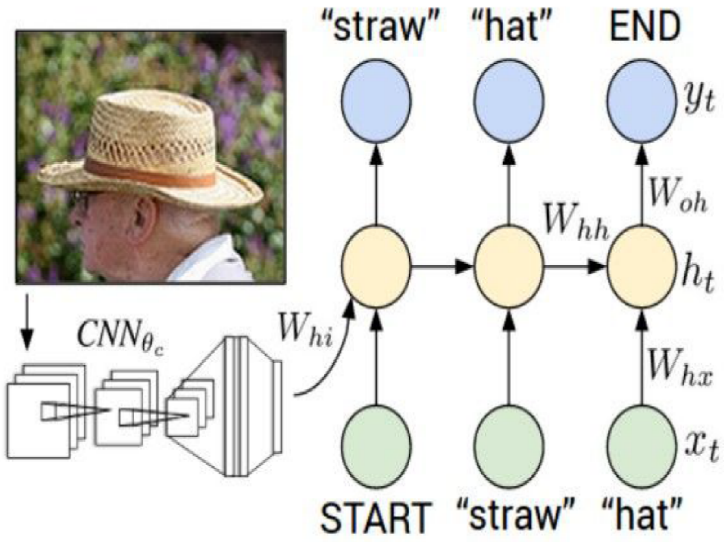
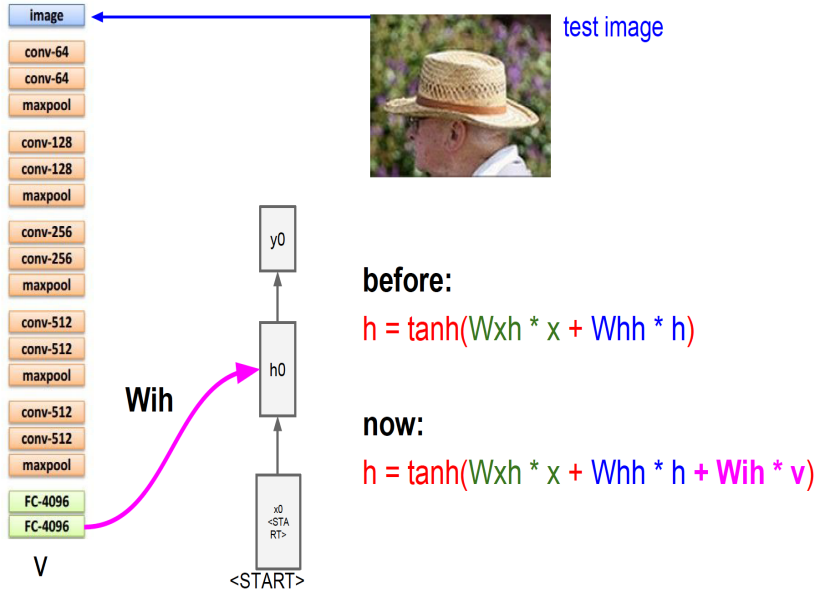


Figure by A. Karpathy



Back propagation in simple RNN

- $a_t = b + wh_{t-1} + Ux_t$
- $h_t = tanh(a_t)$
- $o_t = c + Vh_t$
- $p_t = softmax(o_t)$
- Loss function: $L = -\sum_{t=1}^{T} y_t^T \log(p_t) = \sum_{t=1}^{T} L_t$.
- $\frac{\partial L_t}{\partial p_t} = -\frac{y_t}{y_t^T p_t}$
- $\frac{\partial L_t}{\partial o_t} = \frac{\partial L}{\partial p_t}\frac{\partial p_t}{\partial o_t} = -\frac{y_t}{y_t^T p_t}(diag(p_t) - p_t p_t^T)$
-

$$\frac{\partial L}{\partial V} = \sum_{t=1}^{T} \frac{\partial L}{\partial o_t}\frac{\partial o_t}{\partial V} = \sum_{t=1}^{T} \frac{\partial L}{\partial o_t} h_t$$

Back propagation in simple RNN

- $a_t = b + wh_{t-1} + Ux_t$
- $h_t = tanh(a_t)$
- $o_t = c + Vh_t$
- $p_t = softmax(o_t)$
-

$$\frac{\partial L}{\partial w} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial w}$$

- Contribution from each time point is cumulative in time:

$$\frac{\partial L_t}{\partial w} = \sum_{k=0}^{t} \frac{\partial L_t}{\partial o_t} \frac{\partial o_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial w}$$

Back propagation in simple RNN

- $a_t = b + wh_{t-1} + Ux_t$
- $h_t = tanh(a_t)$
- $o_t = c + Vh_t$
- $p_t = softmax(o_t)$
- $$\frac{\partial L_t}{\partial w} = \sum_{k=0}^{t} \frac{\partial L_t}{\partial o_t} \frac{\partial o_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial w}$$
- $$\frac{\partial L_t}{\partial o_t} = \frac{\partial L}{\partial p_t} \frac{\partial p_t}{\partial o_t} = -\frac{y_t}{y_t^T p_t}(diag(p_t) - p_t p_t^T)$$
- $$\frac{\partial o_t}{\partial h_t} = V$$
- $$\frac{\partial h_k}{\partial w} = \frac{\partial h_k}{\partial a_k} \frac{\partial a_k}{\partial w} = diag(1 - tanh^2(a_k))h_{k-1}$$

Back propagation in simple RNN

- $a_t = b + wh_{t-1} + Ux_t$
- $h_t = tanh(a_t)$
- $o_t = c + Vh_t$
- $p_t = softmax(o_t)$
-

$$\frac{\partial L_t}{\partial w} = \sum_{k=0}^{t} \frac{\partial L_t}{\partial o_t} \frac{\partial o_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial w}$$

- Problematic part:

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k}^{t-1} \frac{\partial h_{j+1}}{\partial h_j}$$

-

$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial h_t}{\partial a_t} \frac{\partial a_t}{\partial h_{t-1}}$$

Back propagation in simple RNN

- $a_t = b + wh_{t-1} + Ux_t$; $h_t = tanh(a_t)$; $o_t = c + Vh_t$; $p_t = softmax(o_t)$
- Denoting $diag(1 - tanh^2(a_t))$ by D_t

$$\frac{\partial h_t}{\partial a_t} = diag(1 - tanh^2(a_t)) = D_t$$

- $$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial h_t}{\partial a_t}\frac{\partial a_t}{\partial h_{t-1}} = D_t w$$
- $$\frac{\partial h_t}{\partial h_k} = \prod_{j=k}^{t-1} \frac{\partial h_{j+1}}{\partial h_j} = (\prod_{j=k}^{t-1} D_{j+1}) w^{t-k}$$

Back propagation in simple RNN

- Contribution of each time point is cumulative up to time t but $\frac{\partial h_t}{\partial h_k}$ is very small if k is far from t. That is, long-range dependence cannot be incorporated in updating weights.

$$\frac{\partial L_t}{\partial w} = \sum_{k=0}^{t} \frac{\partial L_t}{\partial o_t} \frac{\partial o_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial w}$$

$$= \sum_{k=0}^{t} \frac{\partial L_t}{\partial o_t} \frac{\partial o_t}{\partial h_t} [(\prod_{j=k}^{t-1} D_{j+1}) w^{t-k}] \frac{\partial h_k}{\partial w}$$

Two sources of exploding or vanishing gradients in RNN

- Two sources of problems: Nonlinearity and explosion

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k}^{t-1} \frac{\partial h_{j+1}}{\partial h_j} = (\prod_{j=k}^{t-1} D_{j+1}) w^{t-k}$$

- Nonlinearity: $(\prod_{j=k}^{t-1} D_{j+1})$ appears due to $tanh(.)$ relationship.
- Explosion: w^{t-k} could vanish if $|w| < 1$ or explode if $|w| > 1$.
- Attempts have been made to replace $tanh(.)$ with ReLu or gradient clipping to alleviate explosion.
- Long Short-term Memory (LSTM) and Gated Recurrent Unit (GRU) are designed to solve the nonlinearity and explosion problems.

Long Short-term Memory (LSTM)

- Hochreiter and Schmidhuber, 1997
- LSTM allows long-term dependence in time.
- LSTM is designed to avoid the problem due to nonlinear relationship and the problem of gradient explosion.
- Successful in unconstrained handwriting recognition (Graveset al., 2009), speech recognition (Graves et al., 2013; Graves and Jaitly, 2014), handwriting generation (Graves, 2013), machine translation (Sutskever et al., 2014), image captioning (Kiros et al., 2014b; Vinyals et al., 2014b; Xu et al., 2015), and parsing (Vinyals et al., 2014a).

Long Short-term Memory (LSTM)

- input gate: $i_t = \sigma\left(b + Ux_t + wh_{t-1}\right)$
- forget gate: $f_t = \sigma\left(b^f + U^f x_t + w^f h_{t-1}\right)$
- new memory cell: $\tilde{c}_t = tanh\left(b^g + U^g x_t + w^g h_{t-1}\right)$
- updated memory: $c_t = f_t c_{t-1} + i_t \tilde{c}_t$
- output gate: $o_t = \sigma\left(b^o + U^o x_t + w^o h_{t-1}\right)$.
- $h_t = tanh(c_t) o_t$
- $p_t = softmax(c + Vh_t)$
- Existing and new key elements

Long Short-term Memory (LSTM)

- LSTM introduces a memory cell state c_t.
- There is no direct relationship between h_{t-1} and h_t, which caused the problem of vanishing or exploding and lack of long-term dependence.
- Update of c_t controls the time dependence and the information flow.

RNN vs. LSTM

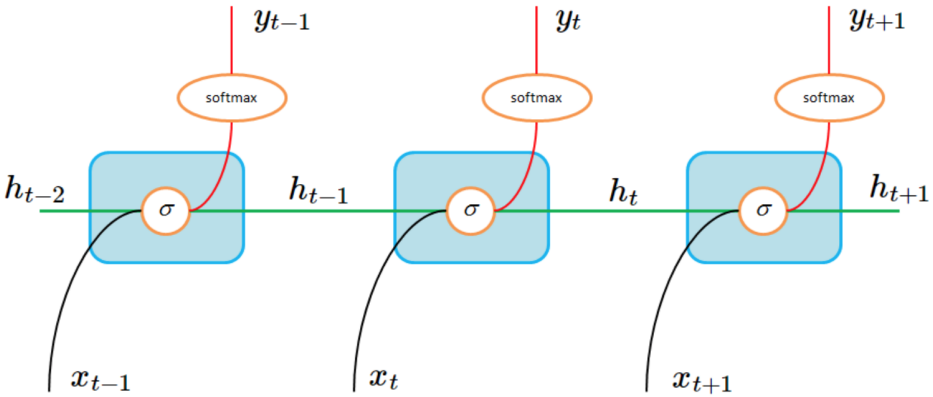


Figure: Simple RNN

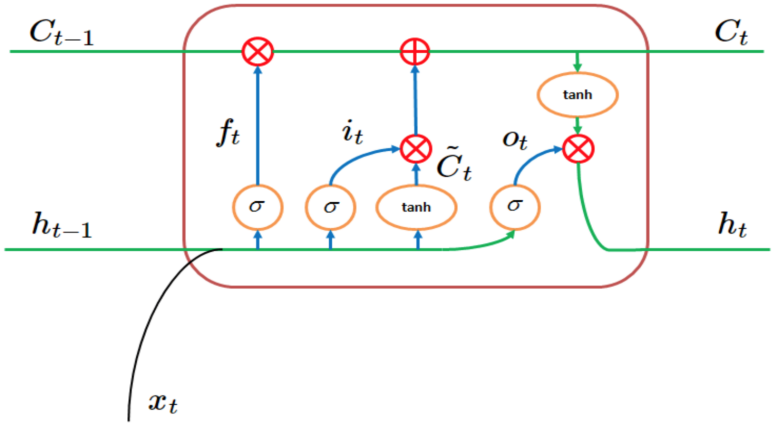


Figure: LSTM

Long Short-term Memory (LSTM)

- Consider (c, V), (b^o, U^o, w^o), $(b^g, U^g, w^g, b^f, U^f, w^f, b, U, w)$
- For (c, V), $\frac{\partial L}{\partial w^s} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial p_t} \frac{\partial p_t}{\partial V}$
- For (b^o, U^o, w^o), $\frac{\partial L}{\partial w^o} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial o_t} \frac{\partial o_t}{\partial w^o}$
- For $(b^g, U^g, w^g, b^f, U^f, w^f, b, U, w)$, derivatives, $\frac{\partial L}{\partial w}$, $\frac{\partial L}{\partial w^g}$, $\frac{\partial L}{\partial w^f}$, involve $\frac{\partial L}{\partial c_t}$ since time dependence is mediated by c_t.
- For w,

$$\frac{\partial L}{\partial w} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial w}$$

- Contribution from time t,

$$\frac{\partial L_t}{\partial w} = \sum_{k=0}^{t} \frac{\partial L_t}{\partial c_t} \frac{\partial c_t}{\partial c_k} \frac{\partial c_k}{\partial w}$$

Long Short-term Memory (LSTM)

- Contribution from time t,

$$\frac{\partial L_t}{\partial w} = \sum_{k=0}^{t} \frac{\partial L_t}{\partial c_t} \frac{\partial c_t}{\partial c_k} \frac{\partial c_k}{\partial w}$$

- Problem part:

$$\frac{\partial c_t}{\partial c_k} = \prod_{j=k}^{t-1} \frac{\partial c_{j+1}}{\partial c_j}$$

- But

$$\frac{\partial c_t}{\partial c_{t-1}} = f_t.$$

-

$$\frac{\partial L_t}{\partial w} = \sum_{k=0}^{t} \frac{\partial L_t}{\partial c_t} (\prod_{j=k}^{t-1} f_j) \frac{\partial c_k}{\partial w}$$

- Additive relationship and forget gate help alleviating vanishing or exploding problem.

Multi-layered RNN and LSTM

- RNN or LSTM

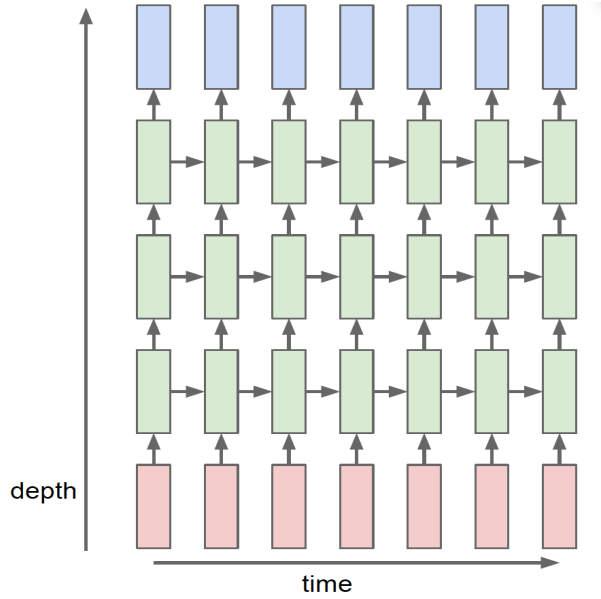
output h_t becomes input for time $t+1$.

- In multi-layered RNN

and LSTM, h_t becomes input for time $t+1$ and h_t for the l^{th} becomes input (x_t) for the $(l+1)^{th}$ layer. For example, LSTM input cells for the 1^{st} and m^{th} are

$$i_t^{(1)} = \sigma\left(b + Ux_t + wh_{t-1}^{(1)}\right)$$

$$i_t^{(m)} = \sigma\left(b + Uh_t^{(m-1)} + wh_{t-1}^{(m)}\right)$$



Gated Recurrent Unit (GRU)

- Proposed by Cho et al. (2014). Further studied by Chung et al. (2014, 2015a); Jozefowicz et al. (2015); Chrupala et al. (2015).
- GRU addresses the two sources of problems of RNN with a simpler structure than LSTM.
- LSTM and GRU are the two popular architectures of RNN.
- GRU are reportedly easier to train.
- LSTM in principle can carry a longer memory.

Gated Recurrent Unit (GRU)

- update gate: $u_t = \sigma\left(b^u + U^u x_t + w^u h_t\right)$
- reset gate: $r_t = \sigma\left(b^r + U^r x_t + w^r h_t\right)$
- $\tilde{h}_t = tanh(U^h x_t + r_t \odot (w^h h_{t-1}) + b^h)$
- $h_t = u_{t-1} h_{t-1} + (1 - u_{t-1})\tilde{h}_t$
- $p_t = softmax(c + V h_t)$
- When $u_{t-1} = 0$ and $r_t = 1$, it reduces to the simple RNN.
- Handles nonlinearity and explosion.

Summary

- RNNs can model sequence data.
- Simple version of RNN has problems of vanishing or exploding gradient.
- LSTM or GRU are designed to alleviate the problem.
- When to use LSTM or GRU is not well understood.

Unsupervised learning

Unsupervised learning

- In unsupervised learning, we try to learn about data without labels.
- Unsupervised learning tasks involve estimation of the density of input x, say $p(x)$ or dimension reduction.
- Dimension reduction tasks are sometimes called feature learning and include PCA, self organizing map, Multidimensional Scaling and autoencoder.
- Density estimation tasks include cluster analysis, anomaly detection, variational autoencoder and generative adversarial network.

Dimensionality and density estimation

- To get the scope of effects of dimensionality on estimating densities, consider kernel density estimators.
- Assume that $X_i \in C \subset \mathbb{R}^d$ where C is compact.
  $\hat{p}(x) = n^{-1} \sum_{i=1}^{n} \frac{1}{h^d} K\left(\frac{\|x - X_i\|}{h}\right)$
- Let β and L be positive numbers. Given a vector $s = (s_1, \cdots, s_d)$, $|s| = s_1 + \cdots + s_d$, and $D^s = \frac{\partial^{s_1 + \cdots + s_d}}{\partial x_1^{d_1} \cdots \partial x_s^{d_s}}$. Define Hölder class
  $\Sigma(\beta, L) = \{|D^s g(x) - D^s g(y)| \leq L\|x - y\|^{\beta - |s|}$, for all s such that $|s| = \lfloor \beta \rfloor$ and all x and $y\}$ where $\lfloor \beta \rfloor$ is the greatest integer strictly less than β.
- Then the MSE is
$$MSE \preceq h^{2\beta} + \frac{1}{nh^d}.$$
- If $h \asymp n^{-1/(2\beta+d)}$,
$$\sup_{p \in \Sigma(\beta, L)} E \int (\hat{p}_h(x) - p(x))^2 dx \leq c n^{-2\beta/(2\beta+d)}$$

High dimension and density estimation

- The rate of convergence $n^{-2\beta/(2\beta+d)}$ is slow when the dimension d is large.
- Instead of estimating p precisely we have to settle for finding an adequate approximation that identifies the regions where p puts large amounts of mass.
- Consider

$$p_h(x) = E(\hat{p}(x)) = \int \frac{1}{h^d} K\left(\frac{\|x - X\|}{h}\right) p(u) du$$

.

- Then applying $P(\|\hat{p}_h - p_h\|_\infty > \epsilon) \leq Ce^{-nc\epsilon^2}$, and

$$\|\hat{p}_h - p_h\|_\infty = O_p\left(\sqrt{\frac{\log n}{n}}\right)$$

- The rate of convergence does not depend on d but how to choose h is not clear.

Unsupervised learning

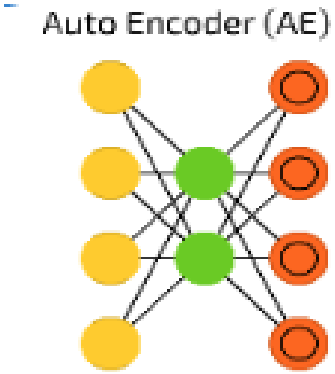
- In deep learning we may face cases where a distribution is concentrated near a lower-dimensional set. This causes problems for density estimation or even lack of definition. Current approaches attempt to find a smoothed density well defined on lower dimensional representation.
- Variational autoencoder and Generative adversarial network attempt to estimate the density through a lower-dimensional representation obtained via deep models.

Autoencoders

Autoencoders

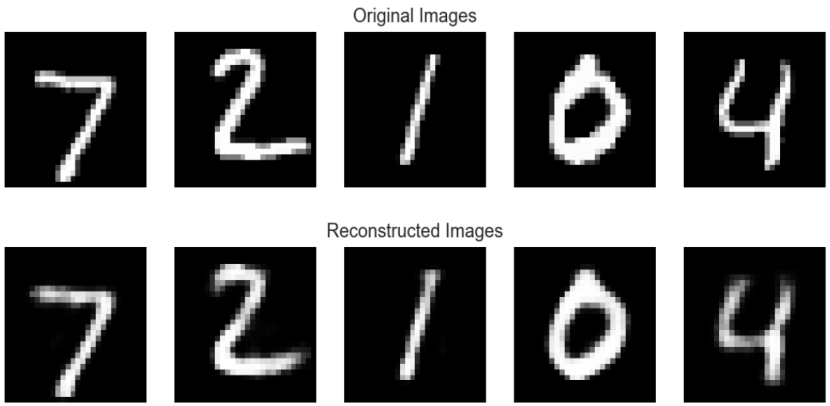
- Goal: Given data $x \in \mathbb{R}^d$, learn a representation $z \in \mathbb{R}^k$, $k < d$, where $z = f(x)$ approximates x well with minimal capacity.

  - If the dimension of z is the same as x, the modelling represents itself. Wants to maximally compress x without losing too much information.
  - The network may be viewed as consisting of two parts: an encoder function $z = f(x)$ and a decoder that produces a reconstruction $x = g(z)$, $g(f(x)) = x$.



Shallow autoencoders

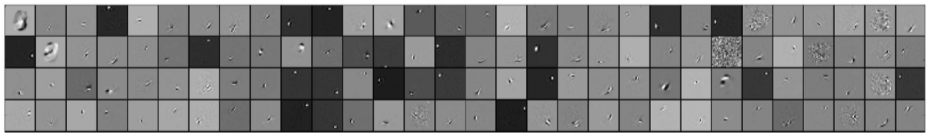
- When $f(.)$ is linear, the problem boils down to principal component analysis (PCA).
- $z = w_1^T x$ and $\hat{x} = w_2 z$. When $w_2 = w_1$, $\hat{x} = w_2 z = w_1 w_1^T x$. Minimizer of $E\|x - w_1^T w_1 x\|_2^2$ over w_1 is $w_1 = \Gamma$, where columns of Γ are the eigenvectors corresponding to the k largest eigenvectors.



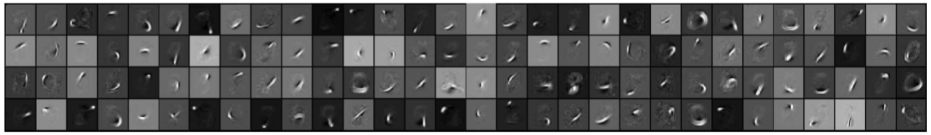
- Loss function $L(x, g(f(x))) = \|x - f(g(x))\|^2$. The higher the dimension of z, loss can be small.

Autoencoders

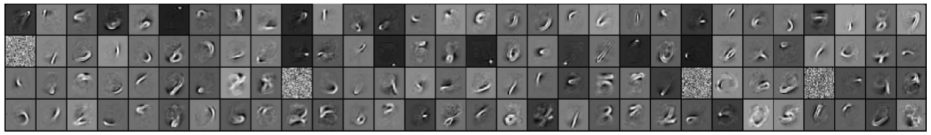
- Sparce Autoencoder: Loss function $L(x, g(f(\mathbf{x}))) + \Omega(z)$ (Ranzato et al., 2007a, 2008)
- $k-$sparse Autoencoder: Autoencoder with linear activation function, where in hidden layers only the k highest activities are kept.



(a) $k = 70$



(b) $k = 40$

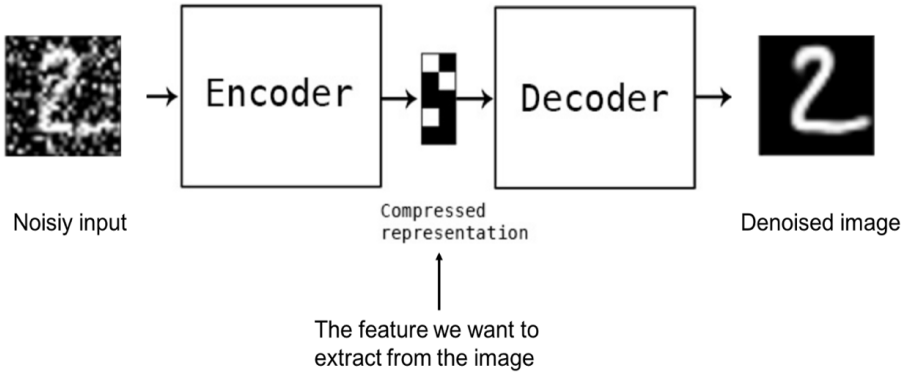


(c) $k = 25$

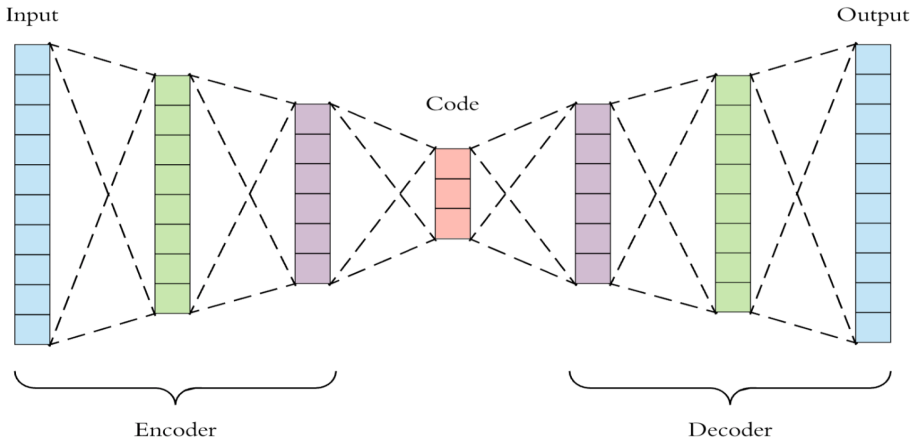
(d) $k = 10$

Denoised autoencoder

- A denoising autoencoder (DAE) instead minimizes $L(x, g(f(\tilde{x})))$, where $\tilde{x}$ is a copy of x that has been corrupted by some form of noise. Denoising autoencoders must therefore undo this corruption rather than simply copying their input.



Deep autoencoders



- $f(.)$ can be compositional including linear, nonlinear, ReLu, convolutional ..etc.
- Encoder/decoder sometimes share weights.

Variational Autoencoders (VAE)

- VAEs have already shown promise in generating many kinds of complicated data, including handwritten digits, faces, house numbers, CIFAR images, physical models of scenes, segmentation, and predicting the future from static images.
- Approximate bayesian inference called variational inference is used.
- We sidetrack for a moment to study variational inference.

Approximate Inference: Variational Inference

Approximate inference on posterior approximation

- In Bayesian inference, posterior distribution

$$p(z|x) = \frac{p(x|z)p(z)}{\int p(x|v)p(v)dv}$$

  is of interest.
- For most models, the posterior is analytically intractable.
- Variation inference attempts to approximate the posterior via $q(z|x)$.
- Minimize Kullback-Leibler divergence $KL(q(z|x)||p(z|x))$ or $KL(p(z|x)||q(z|x))$, where

$$KL(q(z|x)||p(z|x)) = \int q(z|x) \log \frac{q(z|x)}{p(z|x)} dz.$$

Two types of KL divergence

- For $KL(q(z|x)||p(z|x)) = \int q(z|x) \log \frac{q(z|x)}{p(z|x)} dz$, if $p(z|x) = 0$, we must have $q(z|x) = 0$. Minimizing $KL(q||p)$ forces q to choose single mode.
- For $KL(p(z|x)||q(z|x)) = \int p(z|x) \log \frac{p(z|x)}{q(z|x)} dz$ if $p(z|x) > 0$ we must have $q(z|x) > 0$. Minimizing $KL(p||q)$ forces q to cover both modes.

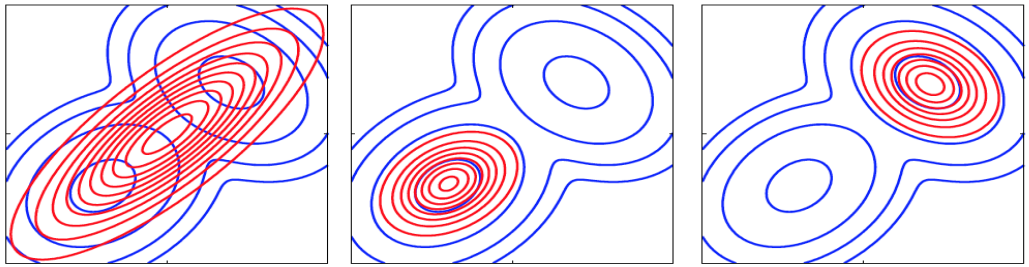


Figure 10.3 Another comparison of the two alternative forms for the Kullback-Leibler divergence. (a) The blue contours show a bimodal distribution $p(\mathbf{Z})$ given by a mixture of two Gaussians, and the red contours correspond to the single Gaussian distribution $q(\mathbf{Z})$ that best approximates $p(\mathbf{Z})$ in the sense of minimizing the Kullback-Leibler divergence $\mathrm{KL}(p||q)$. (b) As in (a) but now the red contours correspond to a Gaussian distribution $q(\mathbf{Z})$ found by numerical minimization of the Kullback-Leibler divergence $\mathrm{KL}(q||p)$. (c) As in (b) but showing a different local minimum of the Kullback-Leibler divergence.

source: Bishop (2006) Pattern recognition and machine learning. Springer

Two types of approximation inference: Variational inference vs. Expectation propagation

- Approximate inference minimizing $KL(p(z|x)||q(z|x))$ includes Expectation propagation (EP).
- $KL(p(z|x)||q(z|x))$ is harder to evaluate.
- Approximate inference minimizing $KL(q(z|x)||p(z|x))$ is called Variational inference (VI).
- VI is more popular than EP due to computational feasibility.

Specifying variational distributions

- Variational inference is originally in terms of an optimization problem in which the quantity being optimized is a functional. The solution is obtained by exploring all possible input functions to find the one that maximizes, or minimizes, the functional. We can consider minimizing KL divergence over q. This approach will be discussed in the mean field method.
- We first consider the case in which q is indexed by a finite dimensional parameter.

Variational Inference: Marginal likelihood, KL divergence and ELBO

- Variational inference minimizes $KL(q(z|x)||p(z|x))$.
- For any distribution $q(z|x;\phi)$ indexed by ϕ, we have

$$\begin{aligned}
\log p(x;\theta) &= \log \frac{p(x,z;\theta)}{p(z|x;\theta)} = \int q(z|x;\phi) \log \frac{p(x,z;\theta)}{p(z|x;\theta)} dz \\
&= \int q(z|x;\phi) \log \Big[\frac{p(x,z;\theta)}{p(z|x;\theta)} \frac{q(z|x;\phi)}{q(z|x;\phi)} \Big] dz \\
&= \int q(z|x;\phi) \Big[\log \frac{q(z|x;\phi)}{p(z|x;\theta)} + \log \frac{p(x,z;\theta)}{q(z|x;\phi)} \Big] dz \\
&= \int q(z|x;\phi) \log \frac{q(z|x;\phi)}{p(z|x;\theta)} dz + \int q(z|x;\phi) \log \frac{p(x,z;\theta)}{q(z|x;\phi)} dz
\end{aligned}$$

Variational Inference: Marginal likelihood, KL divergence and ELBO

- Let

$$Ł(\theta, \phi) = \int q(z|x;\phi) \log \frac{p(x,z;\theta)}{q(z|x;\phi)} dz.$$

Note that

$$\int q(z|x;\phi) \log \frac{q(z|x;\phi)}{p(z|x;\theta)} dz \equiv KL(q(z|x;\phi)||p(z|x;\theta)).$$

Then

$$\log p(x;\theta) = KL(q(z|x;\phi)||p(z|x;\theta)) + L(\theta,\phi)$$

- By maximizing $L(\theta,\phi)$, we minimize $KL(q(z|x;\phi)||p(z;\theta))$.

Variational Inference: ELBO

- For any distribution $q(z|x;\phi)$ indexed by ϕ, we have

$$\log p(x;\theta) = \log\Big(\int p(x,z;\theta)dz\Big) = \log\Big(\int \frac{p(x,z;\theta)}{q(z|x;\phi)} q(z|x;\phi)dz\Big)$$
$$\geq \int \log\Big(\frac{p(x,z;\theta)}{q(z|x;\phi)}\Big) q(z|x;\phi)dz \equiv L(\theta,\phi)$$

  Recall Jensen's inequality $E(f(x)) \geq f(E(x))$ if f is convex.
- We call $L(\theta,\phi)$ the evidence lower bound (ELBO).

Variational Inference: ELBO as a penalized expected likelihood

- We can express

$$\begin{aligned} ELBO = L(\theta, \phi) &= \int q(z|x;\phi) \log \frac{p(x,z;\theta)}{q(z|x;\phi)} dz \\ &= E_{q(z|x;\phi)}[\log p(x,z;\theta)] - E_{q(z|x;\phi)}[\log q(z|x;\phi)] \\ &= E[\log p(x|z;\theta)] + E[\log p(z)] - E[\log q(z|x;\phi)] \\ &= E[\log p(x|z;\theta)] - KL(q(z|x;\phi)||p(z)) \end{aligned}$$

- Maximizing ELBO is equivalent to maximizing $E[\log p(x|z;\theta)]$ with penalty $KL(q(z|x;\phi)||p(z))$.

Optimizing ELBO with respect to parameters

- Optimizing ELBO requires evaluating $\frac{\partial}{\partial\theta}L(\theta,\phi)$ and $\frac{\partial}{\partial\phi}L(\theta,\phi)$.
- First,

$$\begin{aligned}\frac{\partial}{\partial\theta}L(\theta,\phi) &= \frac{\partial}{\partial\theta}E_{q(z|x;\phi)}[\log p(x,z;\theta)] \\ &= \int \frac{p'(x|z;\theta)}{p(x|z;\theta)}q(z|x;\phi)dz\end{aligned}$$

- This term is usually computed stochastically.

Optimizing ELBO

- Optimizing ELBO requires evaluating $\frac{\partial}{\partial\theta}L(\theta,\phi)$ and $\frac{\partial}{\partial\phi}L(\theta,\phi)$.

$$\begin{aligned}\frac{\partial}{\partial\phi}L(\theta,\phi) &= \frac{\partial}{\partial\phi}E_{q(z|x;\phi)}[\log p(x,z;\theta)] - \frac{\partial}{\partial\phi}E_{q(z|x;\phi)}[\log q(z|x;\phi)] \\ &= \int [\log p(x,z;\theta)]\frac{q'(z|x;\phi)}{q(z|x;\phi)}q(z|x;\phi)dz \\ &- \int \frac{q'(z|x;\phi)}{q(z|x;\phi)}q(z|x;\phi)dz \\ &- \int [\log q(z|x;\phi)]\frac{q'(z|x;\phi)}{q(z|x;\phi)}q(z|x;\phi)dz\end{aligned}$$

- Computation is complicated due to parameter appearing in the expectation. An EM-type iterative stochastic evaluation of each term is not guaranteed to be converged.
- One may attempt to reparameterize ELBO so that the expectation is over a distribution free of parameters.

Reparameterization trick

- In evaluating $\frac{\partial}{\partial \phi} \int f(x) p(x; \phi) dx$, let $x = g(\phi, \epsilon)$, where g is differentiable and $p(\epsilon)$ is free of ϕ. That is

$$p(g(\epsilon, \phi)) \left| \frac{\partial g(\phi, \epsilon)}{\partial \epsilon} \right| = p(\epsilon).$$

Then

$$\begin{aligned} \int f(x) p(x; \phi) dz &= \int f(g(\epsilon, \phi)) \left| \frac{\partial g(\phi, \epsilon)}{\partial \epsilon} \right| p(g(\epsilon, \phi)) d\epsilon \\ &= \int f(g(\epsilon, \phi)) p(\epsilon) d\epsilon \end{aligned}$$

Reparameterization trick

- For example, when $p(x;\phi)$ is $N(\mu,\sigma)$,

$$\int f(x)p(x;\phi)dx = \int f(\mu+\epsilon\sigma)p(\epsilon)d\epsilon,$$

  where $\epsilon = (x-\mu)/\sigma$ and $p(\epsilon)$ is the standard normal density.
- Then

$$\frac{\partial}{\partial\phi}\int f(x)p(x;\phi)dz = \int f'(g(\epsilon,\phi))\left|\frac{\partial g(\phi,\epsilon)}{\partial\phi}\right|p(\epsilon)d\epsilon$$

Reparameterization trick

- Back to evaluating ELBO, let

$$h(x, z, \theta, \phi) = \log p(x, z; \theta) - \log q(z|x; \phi).$$

- We need to evaluate

$$L(\theta, \phi) = \int h(x, z, \theta, \phi) q(z|x; \phi) dz = \int h(x, g(\phi, \epsilon), \theta, \phi) p(\epsilon) d\epsilon$$

and

$$\frac{\partial}{\partial \phi} L(\theta, \phi) = \int \frac{\partial}{\partial \phi} h(x, g(\phi, \epsilon), \theta, \phi) p(\epsilon) d\epsilon$$

Evaluation is much simpler since the expectation part is free of parameter ϕ. This can be evaluated stochastically.

Variational Autoencoders

Variational Autoencoder: Example

- Goal: Given data x, learn a representation $z = f(x)$ that approximates x well with minimal capacity.
- ELBO: $L(\theta, \phi) = E_{q(z|x;\phi)}[\log p(x|z;\theta) + \log p(z) - \log q(z|x;\phi)]$
- Encoder: Assume $p(x|z;\theta) \sim N(\mu_{x|z}, \Sigma_{x|z})$, where

$$\mu_{x|z} = a_K(a_{K-1}(..a_2(a_1(z;\theta_1);\theta_2)...\theta_{K-1});\theta_K),$$

$$diag(\Sigma_{z|x}) = b(a_{K-1}(...a_2(a_1(z;\theta_1);\theta_2)...\theta_{K-1});\theta_b).$$

  Parameters for $\mu_{x|z}$ and $diag(\Sigma_{x|z})$ are shared upto the $(K-1)^{th}$ layer.
- Assume $p(z) \sim N(0, I)$

Variational Autoencoder: Example

- Goal: Given data x, learn a representation $z = f(x)$ that approximates x well with minimal capacity.
- Decoder: Assume $q(z|x) \sim N(\mu_{z|x}, \Sigma_{z|x})$ where

$$\mu_{z|x} = f_K(f_{K-1}(...f_2(f_1(z;\phi_1);\phi_2)...\phi_{K-1});\phi_K),$$
$$diag(\Sigma_{z|x}) = g(f_{K-1}(...f_2(f_1(z;\phi_1);\phi_2)...\phi_{K-1});\phi_g)$$

Parameters for $\mu_{z|x}$ and $diag(\Sigma_{z|x})$ are shared upto the $(K-1)^{th}$ layer.

Variational Autoencoder: Example

- ELBO:

$$L(\theta, \phi) = E_{q(z|x;\phi)}[\log p(x|z)] + E_{q(z|x;\phi)}[\log p(z) - \log q(z|x;\phi)].$$

- For the second term, a closed form can be obtained due to the normality assumption of $q(z|x;\phi)$.

$$\begin{aligned}
&E_{q(z|x;\phi)}[\log p(z) - \log q(z|x;\phi)] \\
&\propto -E(\sum_{i=1}^{n} z_i^T z_i | x;\phi) + \sum_{i=1}^{n} \log |\Sigma_{z_i|x_i}| \\
&+ E\sum_{i=1}^{n}(z_i - \mu_{z_i|x_i})\Sigma_{z_i|x_i}^{-1}(z_i - \mu_{z_i|x_i}) \\
&= -\sum_{i=1}^{n} tr(\Sigma_{z_i|x_i}) - \sum_{i=1}^{n} tr(\mu_{z_i|x_i}\mu_{z_i|x_i}^T) + \sum_{i=1}^{n} \log |\Sigma_{z_i|x_i}| + ntr(I)
\end{aligned}$$

Variational Autoencoder: Example

- For the first term, use the reparameterization trick

$$E_{q(z|x;\phi)}[\log p(x|z)] = E_{p(\epsilon)}[\log p(x|g(\epsilon, x; \phi))]$$

and estimate stochastically using

$$\hat{E}_{q(z|x;\phi)}[\log p(x|z)] = \frac{1}{M} \sum_{m=1}^{M} \log p(x|g(\epsilon^{(m)}, x; \phi))$$

Variational Autoencoder: Example

- Putting together, maximize a stochastic version of ELBO,

$$\tilde{L}(\theta, \phi) = \sum_{i=1}^{n} \frac{1}{M} \sum_{m=1}^{M} \log p_\theta(x_i | g(\epsilon_i^m, x_i; \phi)) - \sum_{i=1}^{n} tr(\Sigma_{z_i|x_i}) - \sum_{i=1}^{n} tr(\mu_{z_i|x_i} \mu_{z_i|x_i}^T) + \sum_{i=1}^{n} \log |\Sigma_{z_i|x_i}|$$

  where $g(\epsilon_i^m, x_i; \phi) = \mu_{z_i|x_i} + \Sigma_{z_i|x_i}^{\frac{1}{2}} \epsilon_i^m$ and $\epsilon_i^m \sim N(0, I)$.

- To generate data, sample z from prior and generate x from $p(x|z; \hat{\theta})$.

Mean Field Variational Inference

Mean Field VI

- *Variational* inference is named after variational calculus in which we optimize a functional over all possible input functions. Without assuming parametric models for q and optimize over the parameter, we can consider minimizing KL divergence over function q. Mean field method takes this approach.
- Assume the variational distribution over the latent variables factorizes as $q(z_1, z_2, \cdots, z_m|x) = \prod_{j=1}^{m} q(z_j)$. Denote $q(z_j)$ by q_j.
- This approximation may not contain the true posterior because the latent variables are usually dependent.

Mean Field VI

- ELBO:

$$L(\theta, q_j) = \int \prod_{i=1}^{m} q_i \{ \log p(x,z) - \sum_{i=1}^{m} \log q_i \} dz$$
$$= \int q_j \{ \int \log p(x,z) \prod_{i \neq j} q_i dz_i \} dz_j - \int q_j \log q_j dz_j + c$$

Let

$$\int \log p(x,z) \prod_{i \neq j} q_i dz_i \equiv E_{i \neq j}(\log p(x,z)).$$

Mean Field VI

- ELBO:

$$
\begin{aligned}
L(\theta, q_j) &= \int \prod_{i=1}^{m} q_i \Big\{ \log p(x,z) - \sum_{i=1}^{m} \log q_i \Big\} dz \\
&= \int q_j \Big\{ \int \log p(x,z) \prod_{i \neq j} q_i dz_i \Big\} dz_j - \int q_j \log q_j dz_j + c \\
&= \int q_j E_{i \neq j}(\log p(x,z)) dz_j - \int q_j \log q_j dz_j + c \\
&= \int q_j \log \tilde{p}(x, z_j) dz_j - \int q_j \log q_j dz_j + c
\end{aligned}
$$

where

$$
\log \tilde{p}(x, z_j) = E_{i \neq j}(\log p(x,z)) + c
$$

Mean Field VI

- ELBO

$$L(\theta, \phi) = \int q_j \log \tilde{p}(x, z_j) dz_j - \int q_j \log q_j dz_j + c$$
$$= -KL(q_j || \tilde{p}(x, z_j))$$

- If we fix q_j for $i \neq j$, *ELBO* is negative KL divergence between q_j and $\tilde{p}(x, z_j)$.
- ELBO can be maximized when $q_j^* = \tilde{p}(x, z_j)$,

$$q_j^* = \frac{\exp[E_{i \neq j}\{\log p(x, z_j)\}]}{\int exp[E_{i \neq j}\{\log p(x, z_j)\}] q_j dz_j}$$

Expectation propagation

Expectation propagation

- Another approximated inference. Minimize $KL(p||q)$ (Minka, 2001a; Minka, 2001b).
- Consider the problem of minimizing $KL(p||q)$ with respect to $q(z)$ when $p(z)$ is a fixed distribution and $q(z)$ is a member of the exponential family,

$$q(z) = h(z)g(\eta)\exp(\eta^T u(z))$$

- KL divergence has a form:

$$KL(p||q) = -\log g(\eta) - \eta^T E_{p(z)}(u(z)) + const$$

Expectation propagation

- $$KL(p||q) = -\log g(\eta) - \eta^T E_{p(z)}(u(z)) + const$$
- Setting the derivative with respect to η equals to zero

$$-\frac{\partial}{\partial \eta} \log g(\eta) = E_{p(z)}(u(z)).$$

- Note that

$$-\frac{\partial}{\partial \eta} \log g(\eta) = E_{q(z)}(u(z)).$$

  Then,

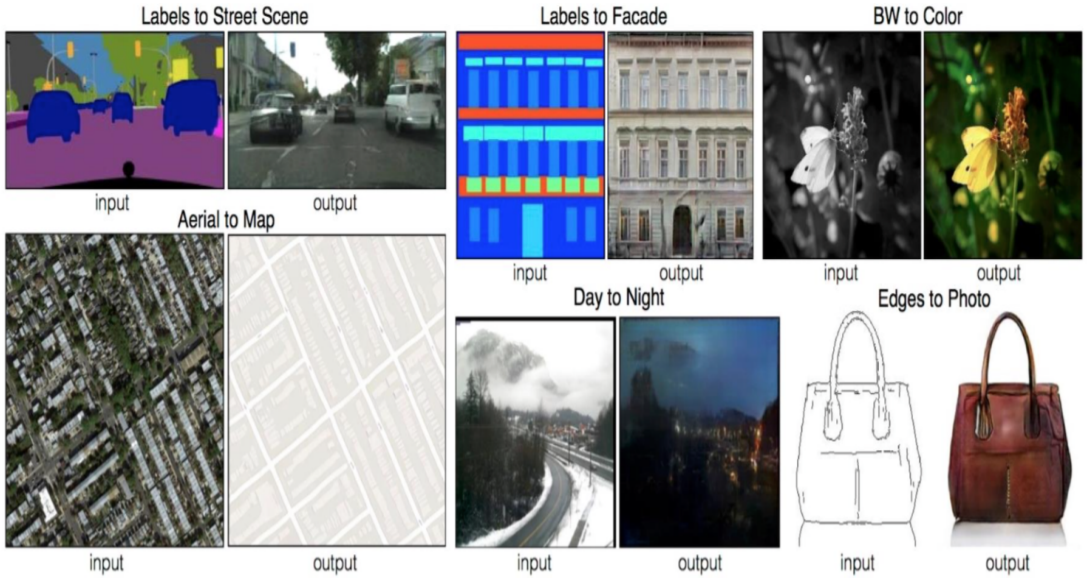
$$E_{q(z)}(u(z)) = E_{p(z)}(u(z))$$

- The optimum solution corresponds to matching the expected sufficient statistics. This is called moment matching.

Generative Adversarial Network

Generative adversarial network (Goodfellow, 2014)

Since publication of GAN by Goodfellow (2014), many applications are reported.



Generative adversarial network (Goodfellow, 2014)

Since publication of GAN by Goodfellow (2014), many variants of GAN have been published.

Generative adversarial network

- Similar setup as in VAE. Attempt to generate x given z with a smaller dimension.
- Generative adversarial networks are based on a game theoretic scenario in which the generator network must compete against an adversary.
- The generator network produces samples $x = g(z; \theta_g)$ that attempts to fool the classifier into believing its samples are real. Its adversary, the discriminator network, attempts to distinguish between samples drawn from the training data and samples drawn from the generator through $P(y = 1|x) = D(x)$.

Generative adversarial network: Setup

- Let $y = 1$ if the data is real and $y = 0$ if the data is fake. We assume that there is a lower dimensional representation z of x.
- To generate data, one needs to know $p(x|y = 1)$.
- If P(y=1)=P(y=0)=.5, we have

$$P(y = 1|x) = \frac{p(x|y = 1)}{p(x|y = 1) + p(x|y = 0)}$$

- In GAN, we specify $P(y = 1|x)$ and $p(x|y = 0)$ and estimate $p(x|y = 1)$ by minimizing a distance between $p(x|y = 0)$ and $p(x|y = 1)$.

Generative adversarial network

- The likelihood function based on $y|x \sim Ber(D(x))$ is

$$\sum_{i=1}^{n} \{y_i \log(D(x_i; \theta_d)) + (1 - y_i) \log(1 - D(x_i; \theta_d))\}$$

However, x_i's are not observed for $y_i = 0$ and we replace x_i with $g(z_i; \theta_g)$. z is not observed for $y = 0$, and the marginal likelihood is

$$L(\theta_d, \theta_g) = \prod_{i=1}^{n} \{(D(x_i; \theta_d))^{y_i} \int p(z_i)(1 - D(g(z_i; \theta_g); \theta_d))^{(1-y_i)} dz_i\}$$

Generative adversarial network: Discriminator and Generator

- Consider the following quantity

$$v(\theta_g, \theta_d) = E_{x \sim p_{data}} \log D(x; \theta_d) + E_{x \sim p_{model}} \log[1 - D\{g(z; \theta_g); \theta_d\}],$$

  where $p_{data} \equiv p(x|y = 1)$ and $p_{model} \equiv p(x|y = 0)$. Note that this is not the expected likelihood in usual sense.
- Optimization
  - Discriminator: Maximize $v(\theta_g, \theta_d)$ over θ_d given θ_g.
  - Generator: Minimize $max_{\theta_d} v(\theta_g, \theta_d)$
  - Alternate Discriminator and Generator steps.

Generative adversarial network

- Discriminator: Maximizing $v(\theta_g, \theta_d)$ over θ_d is to estimate the discriminator. When function space of $D(x)$ is not restricted, argmax of v over D is

$$D^*(x) = \frac{p_{data}}{p_{data} + p_{model}}.$$

- Generator: Plugging in $P(y=1|x) = D^*(x) = \frac{p(x|y=1)}{p(x|y=1)+p(x|y=0)}$ to $v(\theta_g, \theta_d)$, we minimize

$$\begin{aligned} v(\theta_g, \theta_d) &= \int p(x|y=1) \log \frac{p(x|y=1)}{p(x|y=1)+p(x|y=0)} dx \\ &+ \int p(x|y=0) \log \frac{p(x|y=0)}{p(x|y=1)+p(x|y=0)} dx \\ &= KL(p_{data}||p^{**}) + KL(p_{model}||p^{**}) + const \end{aligned}$$

where $p^{**} = (p_{data} + p_{model})/2$

AE, VAE and GAN

	AE	VAE	GAN
estimation of	transformation	conditional distribution	distribution through transformation
specifying transformation	$x = g(z; \theta_g)$ $z = f(x; \theta_f)$	none	$x = g(z; \theta_g)$
specifying distributions	none	$p(z)$, $p(x\|z)$ and $q(z\|x)$	$p(z)$ thus indirectly $p(x)$
objective function	$\|x - g(f(x; \theta_f); \theta_g)\|^2$	KL divergence	Jensen-Shannon divergence

GAN algorithm

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Implementation of GAN

- In practice, $D(x)$ is restricted to neural network. We first maximize $v(\theta_g, \theta_d)$ over θ_d to obtain θ_d^*. Then,

$$\theta_g^* = argmin_{\theta_g} E_{x \sim p_{model}} \log[1 - D\{g(z; \theta_g); \theta_d^*\}]$$

- The i^{th} contribution of $E_{x \sim p_{model}} \log[1 - D\{g(z; \theta_g); \theta_d^*\}]$ is stochastically evaluated by

$$\frac{1}{M} \sum_{m=1}^{M} \log[1 - D\{g(z_i^{(m)}; \theta_g); \theta_d^*\}]$$

where $z_i^{(m)}$ is generated from $p(z)$.

Implementation of GAN

- In practice, minimizing $E_{x \sim p_{model}} \log[1 - D\{g(z; \theta_g); \theta_d^*\}]$ does not work well. Instead, one aims to maximize $E_{x \sim p_{model}} \log[D\{g(z; \theta_g); \theta_d^*\}]$ over θ_g.
- The discriminator determines the loss function for the generator, $KL(p_{data}||p^{**}) + KL(p_{model}||p^{**})$, Jensen-Shannon divergence.
- Jensen-Shannon divergence has advantage over KL divergence in that one can avoid the problem of non-overlapping support.
- Many variants of GAN have been proposed. W-GAN (Arjovsky et al., 2017) is the most popular which uses Wasserstein distance metric to optimize the generating distribution.

Wasserstein GAN: Distance

- If the real data distribution $\mathbb{P}_r$ admits a density and $\mathbb{P}_\theta$ is the distribution of the parametrized density P_θ then, asymptotically, the likelihood inference amounts to minimizing the Kullback-Leibler divergence $KL(\mathbb{P}_r \| \mathbb{P}_\theta)$.
- When distributions are supported by low dimensional manifolds the KL distance is not defined.

Distance

- Total variation distance:

$$\delta(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{A \in \Sigma} |\mathbb{P}_r(A) - \mathbb{P}_\theta(A)|$$

- Kullback-Leibler divergence: $KL(\mathbb{P}_r || \mathbb{P}_\theta)$
- Jensen-Shannon divergence:

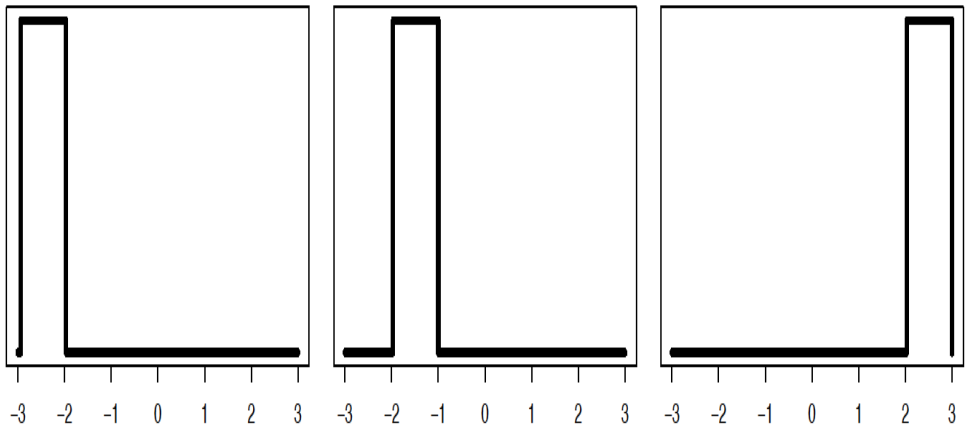
$$JS(\mathbb{P}_r, \mathbb{P}_\theta) = KL(\mathbb{P}_r || \mathbb{P}_m) + KL(\mathbb{P}_\theta || \mathbb{P}_m)$$

- Earth-mover distance or Wasserstein-1 distance:

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} E_{(x,y)\sim\gamma}[\|x - y\|]$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginal distributions are respectively $\mathbb{P}_r$ and $\mathbb{P}_\theta$.

Wasserstein distance



- Distances may ignore the underlying geometry of the space. For three densities p_1, p_2, p_3, we have $\int p_1 - p_2 dx = \int p_1 - p_3 dx = \int p_2 - p_3 dx$ and similarly for the other distances. But our intuition tells us that p_1 and p_2 are close together, which is captured in Wasserstein distance.

Distance: Example

- Let Z be $U[0,1]$ the uniform distribution on the unit interval. Let $\mathbb{P}_0$ be the distribution of $(0, Z) \in \mathbb{R}^2$, 0 on the x-axis and the random variable Z on the y-axis, uniform on a straight vertical line passing through the origin. Let $\mathbb{P}_\theta$ be the distribution of (θ, Z) Then,

$$W(\mathbb{P}_0, \mathbb{P}_\theta) = |\theta|$$

$$JS(\mathbb{P}_0, \mathbb{P}_\theta) = \begin{cases} \log 2 & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases}$$

$$KL(\mathbb{P}_0, \mathbb{P}_\theta) = \begin{cases} +\infty & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases}$$

$$\delta(\mathbb{P}_0, \mathbb{P}_\theta) = \begin{cases} 1 & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases}$$

Distance: Example

- When $\theta \to 0$, the sequence $(\mathbb{P}_{\theta_t})_{t \in N}$ converges to $\mathbb{P}_0$ under the EM distance, but does not converge under the JS, KL, or TV divergences.
- The KL, JS, and TV distances are not sensible loss functions when learning distributions supported by low dimensional manifolds. However the EM distance effectively captures the difference in this setup.

Wasserstein GAN

- Kantorovich-Rubinstein duality:

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{f \in \{|f(x_1) - f(x_2)| \leq |x_1 - x_2|\}} E_{x \sim \mathbb{P}_r}[f(x)] - E_{x \sim \mathbb{P}_\theta}[f(x)]$$

  where the supremum is over all the 1-Lipschitz functions $f : C \to \mathbb{R}$.

- To approximate computation of $W(\mathbb{P}_r, \mathbb{P}_\theta)$, consider parametric family f_w indexed by w and

$$\max_{w \in \Omega} E_{x \sim \mathbb{P}_r}[f_w(x)] - E_{x \sim \mathbb{P}_\theta}[f_w(g_\theta(z))]$$

- W-GAN trains a neural network parameterized with weights w lying in a compact space Ω and then backprop through $E_{z \sim p(z)}[\nabla_\theta f_{w^*}(g_\theta(z))]$, where w^* is the argmax.

Wasserstein GAN

- Note that the fact that Ω is compact implies that all the functions f_w will be $K-$Lipschitz for some K.
- In order to have parameters w lie in a compact space, clamp the weights, for example, $\Omega = [-0.01, 0.01]^l$ after each gradient update.
- Weight clipping is a poor but practical way to enforce a Lipschitz constraint. A large clipping parameter may cause slow convergence, and a small clipping may lead to vanishing gradients.

Wasserstein GAN algorithm

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

WGAN, WAE-GAN and WAE-MMD

	WGAN	WAE-GAN	WAE-MMD
$p(x\|z;\theta)$	$x = g(z;\theta_g)$	$x = g(z;\theta_g)$	$x = g(z;\theta_g)$
$p(z)$	normal	normal	normal
$q(z\|x)$	none	$q(z\|x)$	$q(z\|x)$
$P(y = 1\|x)$	none	$P(y = 1\|x)$	none
w-distance	dual	primal	primal
critic for dual	f_w	-	-
primal contraint	-	$JS(p(z)\|\|q(z))$	$\|\int k(z,)dP(z)$ $-\int k(z,)dQ(z)\|_{\mathcal{H}}$

Family of distance functions

- Given $\mathcal{F}$ and a set of functions from $C \to \mathbb{R}$, define

$$d_{\mathcal{F}}(P, Q) = \sup_{f \in \mathcal{F}} E_{x \sim P}[f(x)] - E_{x \sim Q}[f(x)],$$

  called Integral Probability Metrics (IPM's).
- If $\mathcal{F}$ is the set of Lipschitz functions, $d_{\mathcal{F}}(P, Q)$ is Wasserstein distance.
- If $\mathcal{F}$ is the set of all measurable functions bounded between $[-1, 1]$, $d_{\mathcal{F}}(P, Q)$ is total variation distance.
- If $\mathcal{F} = \{f \in \mathcal{H} : \|f\|_\infty \leq 1\}$ for some Reproducing Kernel Hilbert Space $\mathcal{H}$, $d_{\mathcal{F}}(P, Q)$ is the maximum mean discrepancy (MMD).