Lecture 11:

- MLP is a universal approximator
- Training MLPs: Layer 0 inputs; Layer L outputs
- $\hat{z}_i(l) = i^{th}$ output of l^{th} *layer or state at layer l*

  $= g(\hat{y}_i(l)) == g\left(\sum_{j=0}^{M(l-1)} w_{ij}(l)\hat{z}_j(l-1)\right)$; $\underline{\hat{z}}(l) = \underline{g}(W(l)\underline{z}^{(l-1)})$

  If $g(z_i) = \frac{1}{1+e^{-\alpha z_i}} \Rightarrow g' = \alpha g(1-g)$

  If $g(z_i) = \tanh(\alpha z_i) \Rightarrow g' = \alpha(1-g^2)$

- Cost function: cross entropy, MSE,….
- Want to use incremental gradient descent or stochastic gradient descent algorithm

$$w_{ij}^{(n+1)}(l) = w_{ij}^{(n)}(l) - \eta \frac{\partial J}{\partial w_{ij}^{(n)}(l)} = w_{ij}^{(n)}(l) + \Delta w_{ij}^{(n)}(l)$$

$$= w_{ij}^{(n)}(l) - \eta\lambda_i(l)g'(\hat{y}_i(l))\hat{z}_j(l-1) = w_{ij}^{(n)}(l) - \eta\delta_i(l)\hat{z}_j(l-1)$$

$$\text{cos}\,tate: \lambda_i(l) = \frac{\partial J}{\partial \hat{z}_i(l)} = \sum_{j=0}^{M(l+1)} \lambda_j(l+1)g'(\hat{y}_j(l+1))w_{ji}(l+1); \lambda_i(L) = \frac{\partial J}{\partial \hat{z}_i(L)}$$

Slight variation in implementation:

$$\delta_i(l) = \frac{\partial E}{\partial \hat{y}_i(l)} = \frac{\partial E}{\partial \hat{z}_i(l)}\frac{\partial \hat{z}_i(l)}{\partial \hat{y}_i(l)} = \lambda_i(l)g'(\hat{y}_i(l)) = \left[\sum_{j=0}^{M(l+1)} \delta_j(l+1)w_{ji}(l+1)\right]g'(\hat{y}_i(l)); \delta_i(L) = \frac{\partial J}{\partial \hat{z}_i(L)}g'(\hat{y}_i(L))$$

Implementation issues: need for shuffling data; multiple stopping criteria (cost, gradient, weights,…), acceleration schemes, Nonlinearities, Weight initialization (Xavier initialization), Batch normalization, Minibatch averaged gradient, Training with Noise, Training with hints: e.g., in fault diagnosis, add another output for the fault severity during training only and remove it for testing, Perform cross validation or bootstrap, Dropout, cross entropy

Semi-supervised training:

$S_{nm} = 1\, if\ n\ and\ m\ are\ similar; 0\ otherwise$

Let $\hat{z}(\underline{x}^n, W)$ *be output estimate for an input* $\underline{x}^n$

Define

$$L(\hat{z}(\underline{x}^n, W), \hat{z}(\underline{x}^m, W), S_{nm}) = \begin{cases} \| \hat{z}(\underline{x}^n, W) - \hat{z}(\underline{x}^m, W) \|^2 \ if\ S_{nm} = 1 \\ \max(0, M - \| \hat{z}(\underline{x}^n, W) - \hat{z}(\underline{x}^m, W) \|^2) if\ S_{nm} = 0 \end{cases}$$

$M =$ minimal margin between dissimilar data

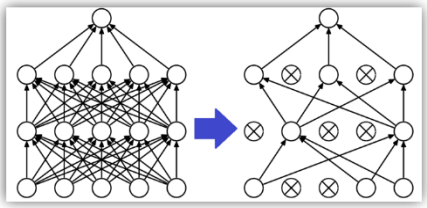
$$J(W) = \frac{1}{2}\left(\sum_{n\in L}\left(z^n - \hat{z}^n(L)\right)^2 + \lambda \sum_{n,m\in U} L(\hat{z}(\underline{x}^n, W), \hat{z}(\underline{x}^m, W), S_{nm})\right)$$

Optimize W *via stochastic gradient by successively sampling from labeled data, unlabeled data with* $S_{nm} = 1$ *and unlabeled data with* $S_{nm} = 0$.

- Why deep networks?
  - Training MLP is difficult if more than two hidden layers are used. The more layers one uses, the more difficult the training becomes (unstable gradients) and probability of getting stuck at local minima increases. Deep networks with sigmoid and tanh nonlinearities experience unstable gradient problem
    - Vanishing Gradient – recall saturation of sigmoids and tanh $\Rightarrow$slow training

$$\delta_i(l) = \left[\sum_{j=0}^{M(l+1)} \delta_j(l+1) w_{ji}(l+1) \right] g'(\hat{y}_i(l))$$

    - Exploding gradient - products of many terms over layers $\Rightarrow$ unstable training
    - Solution: ReLU and cross entropy
  - Is there any need for networks with more than two or three layers?
    - The cortex of human brain can be seen as a multilayer architecture with 5-10 layers dedicated only to our visual system (Hubel and Wiesel, 1962)
    - Using networks with more layers can lead to more compact representations of the input-output mapping.
    - Hastad switching lemma (1987) – Boolean circuits representable with a polynomial number of nodes with k layers may require an exponential number of nodes with (k-1) layers (e.g., parity) … more depth is better
    - Deep networks may require less number of weights than a shallow representation
  - Feature hierarchies (e.g., pixel $\rightarrow$ edge $\rightarrow$texture $\rightarrow$ motif $\rightarrow$ part $\rightarrow$ object in an image recognition task) identified in deep networks can potentially be shared among multiple tasks
    - Transfer/Multi-task learning
- What made deep learning feasible?
  - Deluge of data …. "Big Data"
  - GPUs
  - New nonlinearities (e.g., rectified linear units), batch normalization and cost functions
    - At each layer, make node variables $\underline{z}^{(l)}$ zero mean and unit variance.
  - Dropout to realize an exponentially large ensemble of networks from a single network
    - At each iteration during training, each node is retained with a probability p
    - During testing, the whole network is used, but the weights are scaled-down by p (Google's Tensorflow scales weights by 1/p and does not scale them during testing!)
    - Essentially, dropout may be viewed as an ensemble of 2^n networks (n = number of nodes in the network) …. Better than weight decay regularization



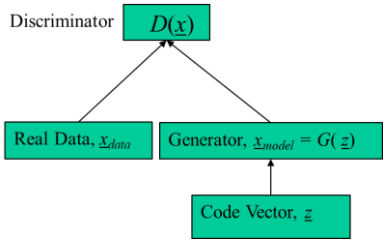
p = 0.5 works!

Approximation:

R = node masking matrix

$$E_R(g(R(l+1)W(l+1)\underline{z}^{(l)})) \approx g(pW(l+1)\underline{z}^{(l)})$$

- Advances in stochastic optimization and adversarial training for robust network training
  - Talked about accelerated SGD methods and mini-batch methods already
  - Adversarial training (game-theoretic training with noise)



- GANs were introduced in 2014
- GANs can fail
- Mode Collapse
  - Vanishing/Exploding Gradients from discriminator to generator
  - Generator produces garbage that can fool the discrimina tor

Minimax (zero-sum) Game : $\max_G \min_D J_D$

$$J_D = -\frac{1}{2}\frac{1}{N}\{\sum_{n=1}^{N}[\ln D(\underline{x}_{data}^n) + \ln(1 - D(G(\underline{z}_{model}^n))]\}$$

$$J_G = -J_D = const + \frac{1}{2}\frac{1}{N}\sum_{n=1}^{N}[\ln(1 - D(G(\underline{z}_{model}^n))]$$

Saturation effect when $D(G(\underline{z}_{model}^n)) \approx 0$

Alternate: "CS hack"

$$J_G = -\frac{1}{2}\frac{1}{N}\sum_{n=1}^{N}[\ln D(G(\underline{z}_{model}^n))]$$

This fixes the saturation problem.

- Modern GANs use Deep CNNs for the generator and discriminator

- Convolution and max-pooling operations that exploit local connectivity to reduce the dimension of the weight space, control overfitting and make the network robust
  - Convolutional Neural Networks (image processing/character recognition; used in deep learning)

    Local receptive fields and weight sharing: Inputs from a small region (say 3x3 pixel patch) are mapped into next layer via sigmoid/ReLU (same weights and bias) for all units

    Pooling: Subsampling takes small regions of convolution layers and pools them (e.g., average, max value), scales it, adds a bias and transforms via a nonlinearity, such as ReLU

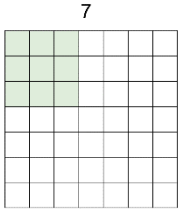
    Advantages of CNN: sparse connectivity, weight sharing and equivalent representation

    Actual convolution:

$$s(i,j) = x(i,j) * w(i,j) = \sum_m \sum_n x(m,n)w(i-m, j-n) = \sum_m \sum_n x(i-m, j-n)w(m,n)$$

    What is used is cross-correlation between *x* and *w*.

$$s(i,j) = x(i,j) * w(i,j) = \sum_m \sum_n x(i+m, j+n)w(m,n)$$



Convolve $\Rightarrow$ slide over spatial locations. 3x3 filter means

You get 5 x5 image at the output…. (7-3+1)

If have three dimensions w(m,n,o) … e.g., o can represent R,G,B

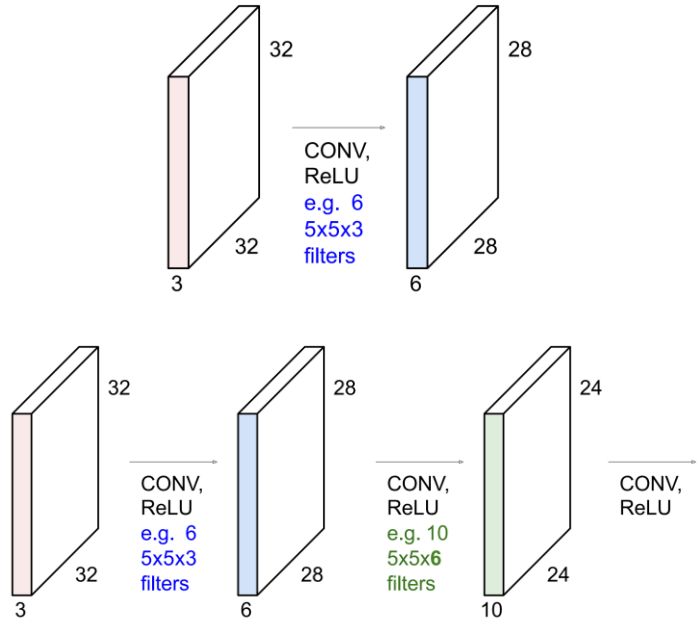
W (receptive field) will have much smaller dimension than image dimension X and W is shared $\Rightarrow$ smaller number of weights to learn. You can have multiple W's … called convolution kernels/filters/maps

Example: X is 32x32x3 and each W is 5 x 5x3 and 6 weight maps $\Rightarrow$ need to learn only

(5x5x3+1)x6=456 filter weight parameters to be learned

You get 6 maps of 28x28x1 as outputs or 28x28x6 stacked maps….. (28=32-5+1)

Connections: 76x6x28x28 =456x28x28=357,504 edges… Note the effect of weight sharing.



What is happening?

- What are the output volume sizes? F: filter size; N: input size; L: # Filters
- 32x32x3 image $\rightarrow$ 6 5x5x3 convolution filters + ReLU $\rightarrow$ 10 5x5x6 convolution filters + ReLU
- Output of first set of convolution filters: (N-F+1)x(N-F+1) x L=28x28x6
- Output of second set of convolution filters: 24x24x10
- Number of weight parameters: 456+1510=1966
- Typically, we move the filter one pixel at a time. What if we skip S pixels called stride, S
- 7x7 with stride 2 and a 3x3 filter: 3x3 output
- 32x32x3 image $\rightarrow$ 6 5x5x3 convolution filters, stride 3+ ReLU $\rightarrow$ 10 5x5x6 convolution filters, stride 1 + ReLU

- Output of first set of convolution filters: ((N-F)/S+1)x((N-F)/S+1) x L=10x10x6
- Output of second set of convolution filters: 6x6x10
- Stride 2 will not work in the first set of convolution filters because (32-5)/2+1=14.5
- In practice, common to zero pad the border with P= (F-1)/2 zeros meaning N'=N-F+F-1=N-1. So, output is [(N-1)/S +1]x[(N-1)/S+1]xL with zero padding.
- Example: 7x7 with stride S=3 and F=3. Zero padding: P=1$\Rightarrow$ output: 3x3.
- 32x32x3 image $\Rightarrow$ 6 5x5x3 convolution filters, stride 1, zero pad 2+ ReLU $\rightarrow$ 10 5x5x6 convolution filters, stride 1, zero pad 2 + ReLU
- Output: 32x32x3 $\rightarrow$ 32x32x6 $\rightarrow$ 32x32x10
- S=1$\Rightarrow$ No change in size if zero padded! Mostly, use stride, S=1; L is typically a power of 2
- Formal way:
  Convolution layer takes a volume I1 x J1 x K1 as input
  Convolution layer is parametrized by
  F: filter size
  L: # Filters
  S: Stride
  P: Amount of zero padding
  Produces a volume of I2 x J2 x K2 as output

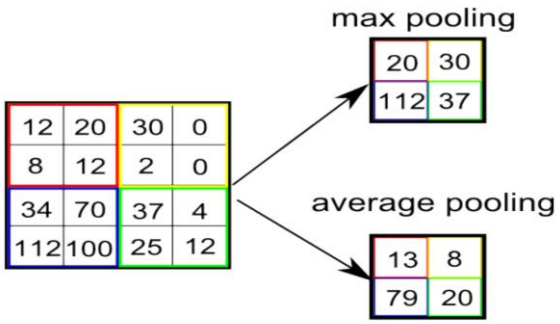
$$I_2 = \frac{(I_1 - F + 2P)}{S} + 1$$
$$J_2 = \frac{(J_1 - F + 2P)}{S} + 1$$
$$K_2 = L$$

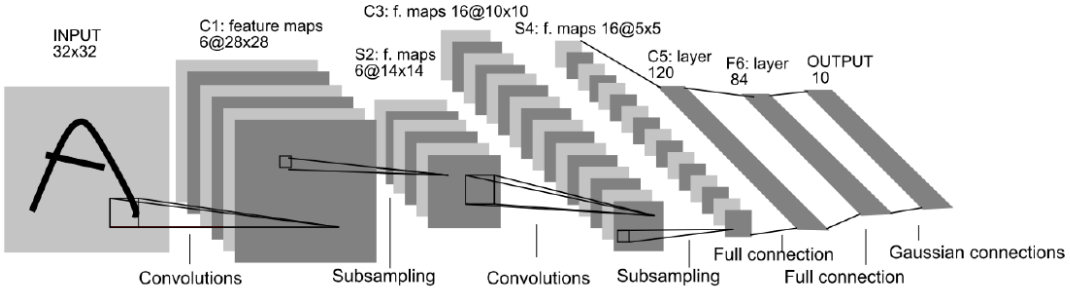
$Note: \frac{(I_1 - F + 2P)}{S}$ and $\frac{(J_1 - F + 2P)}{S}$ must be divisible

  Number of weights per filter: F x F x K1 + 1 (for bias)
  Total Number of weights: (F x F x K1 +1)x L
  1x1 convolution is perfectly OK to use

- Features are learned rather than hand-crafted
- More layers capture more invariances

Pooling: Max pooling, average pooling … down sampling





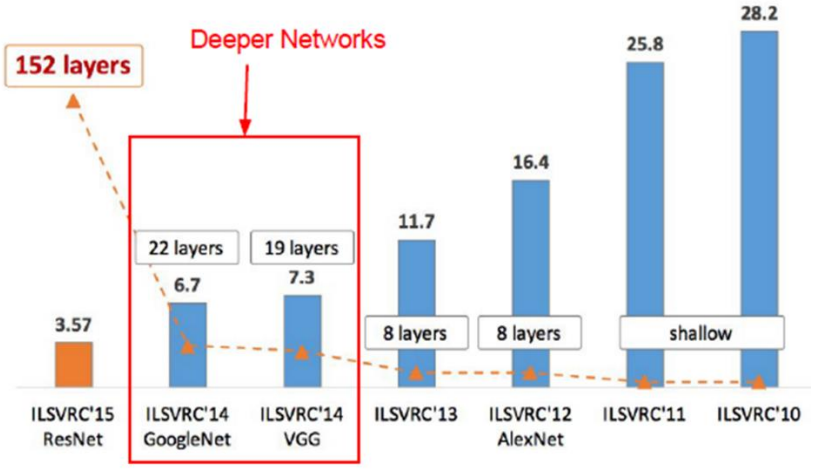
Layer	Trainable Weights	Connections (Edges)
C1	(25+1)x6 = 156	(25+1)x6x28x28 = 122,304
S2	(1+1)x6 = 12 … if averaging; none for max	(4+1)x6x14x14 = 5880 (2x2 links and bias = 5)
C3	6x(25x3+1) + 9x(25x4+1) + 1x(25x6+1) = 1516	1516x10x10 = 151,600
S4	16x2 = 32 … if averaging; none for max	16x5x5x5 = 2000 (2x2 links and bias = 5)
C5	120x(5x5x16+1) = 48,120	Same since fully connected MLP at this point
F6	84x(120+1) = 10,164	Same
Output	10x(84+1) = 850 (RBF)	Same

The structure of the CNN is usually hand-crafted using trial and error

- Number of layers, number of receptive fields, sizes of receptive fields, sizes of sub-sampling (pooling) fields, which fields of the previous layer to connect to, etc.
- Typically, decrease the size of feature maps and increase the number of feature maps for later layers … typically by a factor of 2 every few layers

Checklist for designing deep networks

- Understand the problem domain first
- Visualize the data. Look for any biases in data that favor one decision over another.
- Use appropriate nonlinearities for the hidden units
- Select the right loss function for the application
- Select the right network size
- Too big □ overfitting (variance); Too small □ bias
- See if you need more data or different types of data
- Training, validation and testing. Use 10-fold cross-validation or bootstrap
- Tune the hyper-parameters (adaptive SGD, mini-batch size) via metrics
- Weight initialization and batch normalization
- GPU-based computational units
- Follow the advances in the field
- Never use libraries as “black-boxes” without understanding



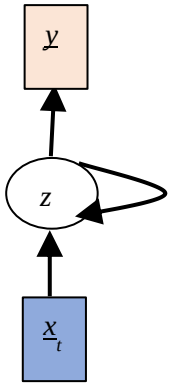
- Alexnet, VGG, GoogleNet, ResNet
- Applications
  - Face Identification versus Face Verification
  - Predicting People's Age based on Images
  - Privacy preserving Identification by suppressing Soft Biometric Information (e.g., gender, age, ethnicity, health, etc.)
  - Dermatologist-level of Skin Cancer
  - Real-time Object Detection
  - Object Segmentation
  - Image Classification under different lighting, contrast and viewpoints
- Dealing with sequences
  - Autoregressive (AR) models: Predict the next term (word, phrase, expression, label) in a sequence from a fixed number of previous ones using delay taps.
  - Nonlinear Autoregressive Moving Average Models (NARMA) via MLPs: MLPs with delayed inputs and outputs generalize autoregressive models with nonlinear hidden units
  - Nonlinear State space models
  - Hidden Markov Models (Lecture 12).. need large # of states to model language
  - Recurrent NNs

Hidden state evolution: $\underline{z}_t = f_W(\underline{z}_{t-1}, \underline{x}_t); \underline{x}_t \sim$ input vector at time t

$f_W \sim$ parameterized function..... same function at each time step t!

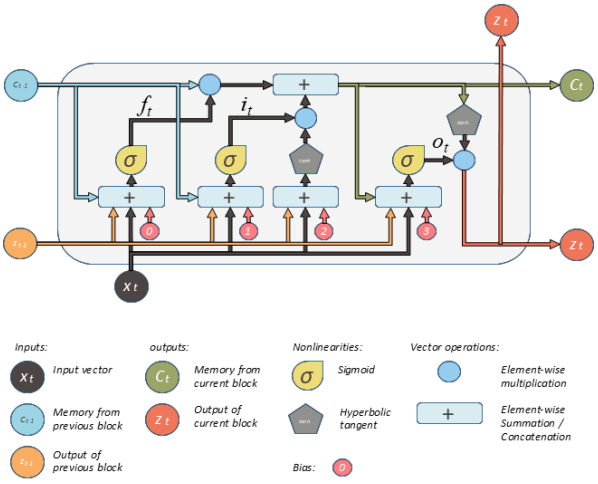
$Example: \underline{z}_t = \tanh(W_{zz}\underline{z}_{t-1} + W_{zx}\underline{x}_t)$

Observations (output): $\underline{y}_t = W_{yz}\underline{z}_t$



- RNNs can be unrolled in time as a DAG and apply Back Propagation
- How far to unroll? Typically less than 25 steps… initialize hidden variables from last unroll
- Often layers are stacked vertically (spatially) to obtain deep RNNs with different W parameters at each layer. Back propagation still works.
- Recurrent networks are flexible: image captioning (CNNs feeding into RNNs), sentiment classification, machine translation, video classification frame by frame, code generation

- Trick for vanishing and exploding gradient problem: Long Short-term Memory Networks (Hochreiter and Schmidhuber, 1997).
- Long Short-term Memory Networks (LSTMs): use purpose-built memory cells to store information and are better at finding and exploiting long range dependencies in data. Each LSTM block contains one or more self-connected memory cells (c) and three multiplicative units (input gate (i), output gate (o), forget/remember gate (f)). The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.. It is like ResNet in skipping units. Back propagation still works.



$$\underline{f}_t = \sigma\left(W_{fx}\underline{x}_t + W_{fz}\underline{z}_{t-1} + W_{fc}\underline{c}_{t-1} + \underline{b}_f\right)$$
$$\underline{i}_t = \sigma\left(W_{ix}\underline{x}_t + W_{iz}\underline{z}_{t-1} + W_{ic}\underline{c}_{t-1} + \underline{b}_i\right)$$
$$\underline{c}_t = \underline{f}_t \odot \underline{c}_{t-1} + \underline{i}_t \odot \tanh(W_{cx}\underline{x}_t + W_{cz}\underline{z}_{t-1})$$
$$\underline{o}_t = \sigma\left(W_{ox}\underline{x}_t + W_{oz}\underline{z}_{t-1} + W_{0c}\underline{c}_t + \underline{b}_o\right)$$
$$\underline{z}_t = \underline{o}_t \odot \tanh(\underline{c}_t); \odot = \text{Hadamard product}$$

1. Forget gate decides whether previous cell state should be kept or thrown out
2. Input gate decides what new information we will store in the cell state
3. Cell state equation implements 1 and 2.
4. Output gate decides what should be in hidden state
5. Strength of cell state is encoded in tanh

- LSTM without peepholes (i.e., where (f,i,o) gates are driven only by (xt, zt-1)) perform reasonably well
- Forget and output gates are salient
- Start small, calibrate and build complex network models
- Gated Recurrent Unit (GRU)
- Merges forget and input gates into a reset gate

$$\underline{r}_t = \sigma\left(W_{rx}\underline{x}_t + W_{rz}\underline{z}_{t-1} + \underline{b}_f\right)$$

$$\underline{o}_t = \sigma\left(W_{ox}\underline{x}_t + W_{oz}\underline{z}_{t-1} + \underline{b}_o\right)$$

$$\underline{z}_t = (\underline{e} - \underline{o}_t) \odot \underline{z}_{t-1} + \underline{o}_t \odot \tanh(W_{zx}\underline{x}_t + W_{zz}(\underline{r}_t \odot \underline{z}_{t-1}))$$

- Merges cell state and hidden state
- Fewer parameters and similar or better performance on some tasks
- LSTMs and GRUs can remember sequences of 100's. For longer sequences, recent variants are attention-based encoder-decoder (RNN, LSTM) models and Causal 2D convolution networks (simultaneous source and target modeling)
- Attention-based models: e.g., Transformer : https://arxiv.org/pdf/1706.03762.pdf
- An attention model enables representation of context (e.g., focus on relevant text or image)
- 2D Convolutional Networks: Jointly encodes the source and target sequence in a 2D CNN It models context because each layer re-encodes the input in the context of the target sequence
- Applications of LSTM
    - Image classification (one-to-one): AlexNet, VGG19, VGG16, GoogleNet, ResNet,….
    - Image captioning (one-to-many): https://arxiv.org/pdf/1411.4555v...
    - Sentiment analysis (many-to-one): Sentence (a sequence) classified as expressing positive or negative sentiment
    - Machine Translation also known as sequence-to-sequence learning (e.g., English to French Translation or many to many): https://arxiv.org/pdf/1409.3215.pdf
    - Video to text (synced and sequenced input and output; many-to-many): https://arxiv.org/pdf/1505.00487...
    - Hand writing generation: http://arxiv.org/pdf/1308.0850v5...
    - Image generation using attention models: https://arxiv.org/pdf/1502.04623...
    - Question-answering: http://www.aclweb.org/anthology/...