Lecture 12

Neural Networks
12. Neural Networks

- Brief history
- Basic formulation (*hierarchical processing*)
- Optimization via gradient descent
- Layer types (*Linear, Pointwise non-linearity*)
- Linear classification with a perceptron
- “Tensor flow”
- Regularizers
- Normalization
http://www.deeplearningbook.org/
By Ian Goodfellow, Yoshua Bengio and Aaron Courville
November 2016
Deep learning

Modeling the visual world is incredibly complicated. We need high capacity models.

In the past, we didn’t have enough data to fit these models. But now we do!

We want a class of high capacity models that are easy to optimize.

Deep neural networks!
A brief history of Neural Networks
Perceptrons, 1958

Rosenblatt


Perceptrons, 1958
Perceptrons, 1958
Perceptrons, expanded edition

An Introduction to Computational Geometry

By Marvin Minsky and Seymour A. Papert

Overview

Perceptrons - the first systematic study of parallelism in computation - has remained a classical work on threshold automata networks for nearly two decades. It marked a historical turn in artificial intelligence, and it is required reading for anyone who wants to understand the connectionist counterrevolution that is going on today.

Artificial-intelligence research, which for a time concentrated on the programming of von Neumann computers, is swinging back to the idea that intelligence might emerge from the activity of networks of neuron-like entities. Minsky and Papert's book was the first example of a mathematical analysis carried far enough to show the exact limitations of a class of computing machines that could seriously be considered as models of the brain. Now the new developments in mathematical tools, the recent interest of physicists in the theory of disordered matter, the new insights into and psychological models of how the brain works, and the evolution of fast computers that can simulate networks of automata have given Perceptrons new importance.

Witnessing the swing of the intellectual pendulum, Minsky and Papert have added a new chapter in which they discuss the current state of parallel computers, review developments since the appearance of the 1972 edition, and identify new research directions related to connectionism. They note a central theoretical challenge facing connectionism: the challenge to reach a deeper understanding of how "objects" or "agents" with individuality can emerge in a network. Progress in this area would link connectionism with what the authors have called "society theories of mind."
Minsky and Papert, 1972
Perceptrons, 1958
PDP authors pointed to the backpropagation algorithm as a breakthrough, allowing multi-layer neural networks to be trained. Among the functions that a multi-layer network can represent but a single-layer network cannot: the XOR function.
Perceptrons, 1958

Minsky and Papert, 1972

PDP book, 1986
LeCun conv nets, 1998

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Demos:
http://yann.lecun.com/exdb/lenet/index.html
Fig. 13. Examples of unusual, distorted, and noisy characters correctly recognized by LeNet-5. The grey-level of the output label represents the penalty (lighter for higher penalties).
Neural networks to recognize handwritten digits? yes

Neural networks for tougher problems? not really
Neural Information Processing Systems 2000

• Neural Information Processing Systems, is the premier conference on machine learning. Evolved from an interdisciplinary conference to a machine learning conference.

• For the 2000 conference:
  – title words predictive of paper acceptance: “Belief Propagation” and “Gaussian”.
  – title words predictive of paper rejection: “Neural” and “Network”.
Enthusiasm over time:

- **Perceptrons**, 1958
- **PDP book**, 1986
- **Minsky and Papert**, 1972
- **AI winter**, 2000
Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

“Alexnet”
ImageNet Classification 2012

- Krizhevsky et al. -- 16.4% error (top-5)
- Next best (non-convnet) – 26.2% error
Krizhevsky, Sutskever, and Hinton, NeurIPS 2012
Enthusiasm over time:

- Perceptrons, 1958
- PDP book, 1986
- Krizhevsky, Sutskever, Hinton, 2012

What comes next?

- Perceptrons, 1958
- PDP book, 1986
- Minsky and Papert, 1972
- Al winter, 2000
- Krizhevsky, Sutskever, Hinton, 2012

28 years 28 years 2028?
What comes next?

- Perceptrons, 1958
- PDP book, 1986
- Minsky and Papert, 1972
- AI winter, 2000
- Krizhevsky, Sutskever, Hinton, 2012
- 2028 ?

enthusiasm

28 years 28 years
[“Mask RCNN”, He et al. 2017]
<table>
<thead>
<tr>
<th>Question</th>
<th>Action 1</th>
<th>Action 2</th>
<th>Action 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>what color is the vase?</td>
<td>classify<a href="attend%5Bvase%5D">color</a></td>
<td>measure[is](combine[and](attend[bus], attend[full]))</td>
<td>measure[is](combine[and](attend[red], re-attend<a href="attend%5Bcircle%5D">above</a>))</td>
</tr>
<tr>
<td>is the bus full of passengers?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>is there a red shape above a circle?</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Object recognition

Feature extractors

Edges
Texture
Colors

Segments
Parts

“clown fish”

Classifier
Object recognition

Feature extractors

\[ \phi_k(x) \]

Edges

Texture

Colors

Segments

Parts

Classifier

\[ f_\theta(x) = \sum_{k=1}^{K} \theta_k \phi_k(x) \]

“clown fish”
Object recognition

![Clown fish image]

Learned

“clown fish”
Object recognition

Neural net

Learned

“clown fish”
Object recognition

Deep neural net

“clown fish”
Deep learning

\[ \mathbf{x}_i \]

“clown fish”

\[ \mathbf{y}_i \]

\( \theta_1 \quad \theta_2 \quad \theta_3 \quad \theta_4 \quad \theta_5 \quad \theta_6 \)

Loss

\[ \mathcal{L}(f_{\theta}(\mathbf{x}_i), \mathbf{y}_i) \]

\[ \theta^* = \arg \min_{\theta} \sum_{i=1}^{N} \mathcal{L}(f_{\theta}(\mathbf{x}_i), \mathbf{y}_i) \]
Gradient descent

\[
\theta^* = \arg \min_{\theta} \sum_{i=1}^{N} \mathcal{L}(f_{\theta}(x_i), y_i)
\]

\[J(\theta)\]
Gradient descent

\[ J(\theta) \]

\[ \theta^* = \arg \min_{\theta} J(\theta) \]
Gradient descent

\[ \theta^* = \arg \min_{\theta} \sum_{i=1}^{N} \mathcal{L}(f_{\theta}(x_i), y_i) \]

One iteration of gradient descent:

\[ \theta^{t+1} = \theta^t - \eta_t \left. \frac{\partial J(\theta)}{\partial \theta} \right|_{\theta=\theta^t} \]

learning rate
Parameters

\[ \theta \rightarrow J(\theta) \rightarrow \nabla_{\theta} J(\theta) \rightarrow \mathcal{H}_\theta(J(\theta)) \]

\[ \theta^* = \arg \min_{\theta} J(\theta) \]

- What’s the knowledge we have about J?
  - We can evaluate \( J(\theta) \)
  - We can evaluate \( J(\theta) \) and \( \nabla_{\theta} J(\theta) \)
  - We can evaluate \( J(\theta) \), \( \nabla_{\theta} J(\theta) \), and \( \mathcal{H}_\theta(J(\theta)) \)

\[ \text{Black box optimization} \]
\[ \text{First order optimization} \]
\[ \text{Second order optimization} \]
Comparison of gradient descent variants

[http://ruder.io/optimizing-gradient-descent/]
Computation in a neural net
Computation in a neural net

**Linear layer**

\[ y_j = \sum_i w_{ij} x_i \]
Computation in a neural net

\[ y_j = \sum_i w_{ij} x_i + b_j \]
Computation in a neural net

Linear layer

Input representation

\[ x \]

Output representation

\[ w_j \]

\[ b_j \]

\[ y_j = x^T w_j + b_j \]

\[ \theta = \{W, b\} \]

weights

bias

parameters of the model
Example: linear regression with a neural net

**Linear layer**

Input representation

Output representation

\[
f_{w,b}(x) = \mathbf{x}^T \mathbf{w} + b
\]
Computation in a neural net

"Perceptron"

Input representation

Output representation

Pointwise Non-linearity

\[ g(y) = \begin{cases} 
1, & \text{if } y > 0 \\
0, & \text{otherwise} 
\end{cases} \]
Example: linear classification with a perceptron

\[ y = \mathbf{x}^T \mathbf{w} + b \]
Example: linear classification with a perceptron

\[ y = \mathbf{x}^T \mathbf{w} + b \]

\[ g(y) = \begin{cases} 
1, & \text{if } y > 0 \\
0, & \text{otherwise}
\end{cases} \]
Example: linear classification with a perceptron

\[ y = \mathbf{x}^T \mathbf{w} + b \]

\[ g(y) = \begin{cases} 
1, & \text{if } y > 0 \\
0, & \text{otherwise}
\end{cases} \]
Example: linear classification with a perceptron

\[ g(y) \]

\[ y = \mathbf{x}^T \mathbf{w} + b \]

\[ g(y) = \begin{cases} 
1, & \text{if } y > 0 \\
0, & \text{otherwise}
\end{cases} \]
Example: linear classification with a perceptron

\[ \hat{y} = \mathbf{x}^T \mathbf{w} + b \]

\[ g(\hat{y}) = \begin{cases} 
1, & \text{if } \hat{y} > 0 \\
0, & \text{otherwise}
\end{cases} \]

\[ \mathbf{w}^*, b^* = \arg \min_{\mathbf{w}, b} \mathcal{L}(g(\hat{y}), y_i) \]
Example: linear classification with a perceptron

\[ \hat{y} = \mathbf{x}^T \mathbf{w} + b \]

\[ g(\hat{y}) = \begin{cases} 
1, & \text{if } \hat{y} > 0 \\
0, & \text{otherwise}
\end{cases} \]

\[ \mathbf{w}^*, b^* = \arg \min_{\mathbf{w}, b} \mathcal{L}(g(\hat{y}), y_i) \]
Computation in a neural net

\[ g(y) = \begin{cases} 
1, & \text{if } y > 0 \\
0, & \text{otherwise} 
\end{cases} \]
Computation in a neural net — nonlinearity

\[ g(y) = \frac{1}{1 + e^{-y}} \]
Computation in a neural net — nonlinearity

- Interpretation as firing rate of neuron
- Bounded between [0,1]
- Saturation for large +/- inputs
- Gradients go to zero
- Outputs centered at 0.5 (poor conditioning)
- Not used in practice

Sigmoid

\[ g(y) = \frac{1}{1 + e^{-y}} \]
Computation in a neural net — nonlinearity

- Bounded between [-1,+1]
- Saturation for large +/- inputs
- Gradients go to zero
- Outputs centered at 0
- Preferable to sigmoid

\[ \text{tanh}(x) = 2 \text{ sigmoid}(2x) - 1 \]
Computation in a neural net — nonlinearity

- Unbounded output (on positive side)
- Efficient to implement: \( \frac{\partial g}{\partial y} = \begin{cases} 0, & \text{if } y < 0 \\ 1, & \text{if } y \geq 0 \end{cases} \)
- Also seems to help convergence (see 6x speedup vs tanh in [Krizhevsky et al.])
- Drawback: if strongly in negative region, unit is dead forever (no gradient).
- Default choice: widely used in current models.

Rectified linear unit (ReLU)

\[
g(y) = \max(0, y)
\]
Computation in a neural net — nonlinearity

• where $\alpha$ is small (e.g. 0.02)

• Efficient to implement: 

$$ \frac{\partial g}{\partial y} = \begin{cases} 
-a, & \text{if } y < 0 \\
1, & \text{if } y \geq 0 
\end{cases} $$

• Also known as probabilistic ReLU (PReLU)

• Has non-zero gradients everywhere (unlike ReLU)

• $\alpha$ can also be learned (see Kaiming He et al. 2015).

Leaky ReLU

$$ g(y) = \begin{cases} 
\max(0, y), & \text{if } y \geq 0 \\
\alpha \min(0, y), & \text{if } y < 0 
\end{cases} $$
Stacking layers

Input representation

Intermediate representation

Output representation

$h = \text{“hidden units”}$
Stacking layers

\[
\begin{align*}
\text{Input representation} & \quad \text{Intermediate representation} & \quad \text{Output representation} \\
\begin{array}{c}
\mathbf{x} \\
1
\end{array} & \quad \begin{array}{c}
\mathbf{W}^{(1)} \\
\mathbf{h} \\
1 \\
\mathbf{b}^{(1)} \\
\end{array} & \quad \begin{array}{c}
\mathbf{W}^{(2)} \\
\mathbf{y} \\
\mathbf{b}^{(2)} \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
\mathbf{h} &= g(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)}) & \mathbf{y} &= \mathbf{W}^{(2)} \mathbf{h} + \mathbf{b}^{(2)} \\
\theta &= \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}
\end{align*}
\]
Stacking layers

Input representation | Intermediate representation | Output representation

\[ h = g(W^{(1)}x + b^{(1)}) \quad y = W^{(2)}h + b^{(2)} \]

\[ \theta = \{ W^{(1)}, \ldots, W^{(L)}, b^{(1)}, \ldots, b^{(L)} \} \]
Stacking layers

\[ h = g(W^{(1)} x + b^{(1)}) \quad y = W^{(2)} h + b^{(2)} \]

\[ \theta = \{W^{(1)}, \ldots, W^{(L)}, b^{(1)}, \ldots, b^{(L)}\} \]
Stacking layers

\[ h = g(W^{(1)}x + b^{(1)}) \quad y = W^{(2)}h + b^{(2)} \]

\[ \theta = \{ W^{(1)}, \ldots, W^{(L)}, b^{(1)}, \ldots, b^{(L)} \} \]
Stacking layers

\[
h = g(W^{(1)}x + b^{(1)}) \quad y = W^{(2)}h + b^{(2)}
\]

\[
\theta = \{W^{(1)}, \ldots, W^{(L)}, b^{(1)}, \ldots, b^{(L)}\}
\]
Connectivity patterns

Fully connected layer

Locally connected layer (Sparse $W$)
2-dimensional input representation

\[ \mathbb{R}^{H \times W \times C^{(l)}} \rightarrow \mathbb{R}^{H \times W \times C^{(l+1)}} \]

2-dimensional output representation

[Figure from Andrea Vedaldi]
Deep nets

Linear
Non-linearity

Classify

\[ f(x) = f_L(\ldots f_2(f_1(x))) \]
Classifier layer

Last layer

... -> argmax “clown fish”

- dolphin
- cat
- grizzly bear
- angel fish
- chameleon
- **clown fish**
- iguana
- elephant
...
Loss function

Network output

- dolphin
- cat
- grizzly bear
- angel fish
- chameleon
- clown fish
- iguana
- elephant

Ground truth label

- “clown fish”

Loss → error
Loss function

Network output

- dolphin
- cat
- grizzly bear
- angel fish
- chameleon
- clown fish
- iguana
- elephant

Ground truth label

- “clown fish”

Loss function with output:

- Small
Loss function

Network output

- dolphin
- cat
- grizzly bear
- angel fish
- chameleon
- clown fish
- iguana
- elephant

Ground truth label

- "grizzly bear"

Loss → large
Prediction $\hat{y}$

$$f_\theta : X \rightarrow \mathbb{R}^K$$

Ground truth label $y$

- dolphin
- cat
- grizzly bear
- angel fish
- chameleon
- clown fish
- iguana
- elephant

$\mathbf{x}$ $\rightarrow$ $\mathbf{f}$

- dolphin
- cat
- grizzly bear
- angel fish
- chameleon
- clown fish
- iguana
- elephant
Probability of the observed data under the model

\[ H(y, \hat{y}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k \]
Representational power

- 1 layer? Linear decision surface.
- 2+ layers? In theory, can represent any function. Assuming non-trivial non-linearity.

- But issue is efficiency: very wide two layers vs narrow deep model? In practice, more layers helps.
Example: linear classification with a perceptron

\[ y = \mathbf{x}^T \mathbf{w} + b \]
Example: nonlinear classification with a deep net

\[ h = g(W^{(1)}x + b^{(1)}) \]

\[ y = W^{(2)}h + b^{(2)} \]
Example: nonlinear classification with a deep net

What class is blue point?

Answer: Underfitting
Answer: Appropriate model
Answer: Overfitting
Deep learning

$\mathbf{x}_1$

“clown fish”

$\mathbf{y}_1$

$\theta_1 \quad \theta_2 \quad \theta_3 \quad \theta_4 \quad \theta_5 \quad \theta_6$

Loss

$\mathcal{L}(f_\theta(\mathbf{x}_1), \mathbf{y}_1)$

$\theta^* = \arg \min_\theta \sum_{i=1}^N \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)$
Deep learning

\[ \theta^* = \arg \min_{\theta} \sum_{i=1}^{N} \mathcal{L}(f_{\theta}(x_i), y_i) \]
Deep learning

\[ \theta^* = \arg \min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i) \]
Batch (parallel) processing

Images

Features

...
Tensors
(multi-dimensional arrays)

Each layer is a representation of the data
h^{(1)} \in \mathbb{R}^{N_{\text{batch}} \times C^{(1)}}

# neurons
# features
# units
# “channels”
Tensors
(multi-dimensional arrays)

\[ h^{(1)} \in \mathbb{R}^{N_{\text{batch}} \times C^{(1)}} \]

- # neurons
- # features
- # units
- # “channels”
"Tensor flow"

\[ h^{(1)} \in \mathbb{R}^{N_{\text{batch}} \times C^{(1)}} \]

\[ h^{(2)} \in \mathbb{R}^{N_{\text{batch}} \times C^{(2)}} \]
Layer 1 representation

Layer 6 representation

structure, construction
covering
commodity, trade good, good
conveyance, transport
invertebrate
bird
hunting dog

[DeCAF, Donahue, Jia, et al. 2013]

[Visualization technique: t-sne, van der Maaten & Hinton, 2008]
“Tensor flow”

\[ h^{(1)} \in \mathbb{R}^{N_{\text{batch}} \times H^{(1)} \times W^{(1)} \times C^{(1)}} \]

\[ h^{(2)} \in \mathbb{R}^{N_{\text{batch}} \times H^{(2)} \times W^{(2)} \times C^{(2)}} \]
Regularizing deep nets

Deep nets have millions of parameters!

On many datasets, it is easy to overfit — we may have more free parameters than data points to constrain them.

How can we regularize to prevent the network from overfitting?
1. Fewer neurons, fewer layers
2. Weight decay
3. Dropout
4. Normalization layers
5. ...
Recall: regularized least squares

\[ f_\theta(x) = \sum_{k=0}^{K} \theta_k x^k \]

\[ R(\theta) = \lambda \|\theta\|^2 \]  

Only use polynomial terms if you really need them! Most terms should be zero

ridge regression, a.k.a., Tikhonov regularization

Probabilistic interpretation: R is a Gaussian prior over values of the parameters.
Regularizing the weights in a neural net

\[
\theta^* = \arg \min_{\theta} \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i) + R(\theta)
\]

\[
R(W) = \lambda \|W\|_2^2 \quad \text{weight decay}
\]

“We prefer to keep weights small.”
Dropout

Input representation  Intermediate representation  Output representation

$\theta = \{W^{(1)}, \ldots, W^{(L)}, b^{(1)}, \ldots, b^{(L)}\}$
Dropout

Input representation → Intermediate representation → Output representation

\[ \theta = \{W^{(1)}, \ldots, W^{(L)}, b^{(1)}, \ldots, b^{(L)}\} \]
Dropout

\[ \theta = \{ \mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)} \} \]
Dropout

\[ \theta = \{ W^{(1)}, \ldots, W^{(L)}, b^{(1)}, \ldots, b^{(L)} \} \]
Dropout

Randomly zero out hidden units.

Prevents network from relying too much on spurious correlations between different hidden units.

Can be understood as averaging over an exponential ensemble of subnetworks. This averaging smooths the function, thereby reducing the effective capacity of the network.
Normalization layers

\[ h \xrightarrow{\text{Norm}} \hat{h} \xrightarrow{\text{ReLU}} h^+ \]

\[ \hat{h}_k = \frac{h_k - \mathbb{E}[h_k]}{\sqrt{\text{Var}[h_k]}} \]
Normalization layers

\[
\hat{h}_k = \frac{h_k - \mathbb{E}[h_k]}{\sqrt{\text{Var}[h_k]}}
\]
Normalization layers

\[
\hat{h}_k = \frac{h_k - \mathbb{E}[h_k]}{\sqrt{\text{Var}[h_k]}}
\]
Normalization layers

\[ \hat{h}_k = \frac{h_k - \mathbb{E}[h_k]}{\sqrt{\text{Var}[h_k]}} \]
Normalization layers

Keep track of mean and variance of a unit (or a population of units) over time.

Standardize unit activations by subtracting mean and dividing by variance.

Squashes units into a **standard range**, avoiding overflow.

Also achieves **invariance** to mean and variance of the training signal.

Both these properties reduce the effective capacity of the model, i.e. regularize the model.
Normalization layers

Normalize w.r.t. a single hidden unit’s pattern of activation over training examples (a batch of examples).

[Figure from Wu & He, arXiv 2018]
Normalization layers

Normalize w.r.t. the mean and variance of the activations of all the hidden units (neurons) on this layer (c).

[Figure from Wu & He, arXiv 2018]
Normalization layers

Normalize w.r.t. the mean and variance of the activations of all the hidden units (neurons) on this layer (c) that process this particular location (h,w) in the image.

[Figure from Wu & He, arXiv 2018]
Normalization layers

Batch Norm  Layer Norm  Instance Norm  Group Norm

Might as well…

[Figure from Wu & He, arXiv 2018]
Taxonomy of Learning Problems
Learning from examples

(aka supervised learning)

Training data

\[
\begin{align*}
\{x_1, y_1\} & \quad \rightarrow & \quad \text{Learner} & \quad \rightarrow & \quad f : X \rightarrow Y \\
\{x_2, y_2\} \\
\{x_3, y_3\} \\
\vdots \\
\end{align*}
\]

\[
f^* = \arg \min_{f \in \mathcal{F}} \sum_{i=1}^{N} \mathcal{L}(f(x_i), y_i)
\]
Learning without examples
(includes unsupervised learning and reinforcement learning)

Data

\[
\begin{align*}
\{x_1\} \\
\{x_2\} \\
\{x_3\} \\
\ldots \\
\end{align*}
\]

\[
\begin{array}{ccc}
\text{Learner} & \rightarrow & ?
\end{array}
\]
Density modeling

Data → Learner → Density

$p : \mathcal{X} \rightarrow [0, 1]$

Clustering

Data $\rightarrow$ Learner $\rightarrow$ Clusters

$f : \mathcal{X} \rightarrow \{1, \ldots, k\}$
Representation Learning

\[
\begin{align*}
\text{Data} & \quad \{x_1\} \\
& \quad \{x_2\} \quad \rightarrow \quad \text{Learner} \\
& \quad \{x_3\} \\
& \quad \ldots
\end{align*}
\]

\rightarrow \quad \text{Representations}
Representation Learning

\[ X \rightarrow f \rightarrow \text{Compact mental representation} \]

Image

“Coral”

“Fish”
Reinforcement learning

Score ("Return")

Behavior ("Trajectory")

Data

\{ \tau_1, R_1 \}

\{ \tau_2, R_2 \}

\{ \tau_3, R_3 \}

\ldots

\rightarrow Learner

Policy

\pi : s \rightarrow a

\tau = \{ s_1, a_1, s_2, a_2, \ldots \}
Differentiable programming

Deep nets are popular for a few reasons:
1. High capacity
2. Easy to optimize (differentiable)
3. Compositional “block based programming”

An emerging term for general models with these properties is \textit{differentiable programming}. 

Yann LeCun
January 5

OK, Deep Learning has outlived its usefulness as a buzz-phrase. Deep Learning est mort. Vive Differentiable Programming!

Thomas G. Dietterich
@dietterich

DL is essentially a new style of programming--"differentiable programming"--and the field is trying to work out the reusable constructs in this style. We have some: convolution, pooling, LSTM, GAN, VAE, memory units, routing units, etc. 

8:02 AM - 4 Jan 2018

65 Retweets 134 Likes
Differentiable programming

[Figure from “Neural Module Networks”, Andreas et al. 2017]
Next up:
Optimization of NNs (backprop)
12. Neural Networks

- Brief history
- Basic formulation (*hierarchical processing*)
- Optimization via gradient descent
- Layer types (*Linear, Pointwise non-linearity*)
- Linear classification with a perceptron
- “Tensor flow”
- Regularizers
- Normalization