

Learning Deep Structured Models

Raquel Urtasun

University of Toronto

August 21, 2015

Current Status of your Field?



Roadmap

- ① Part I: Deep learning
- ② Part II: Deep Structured Models

Part I: Deep Learning

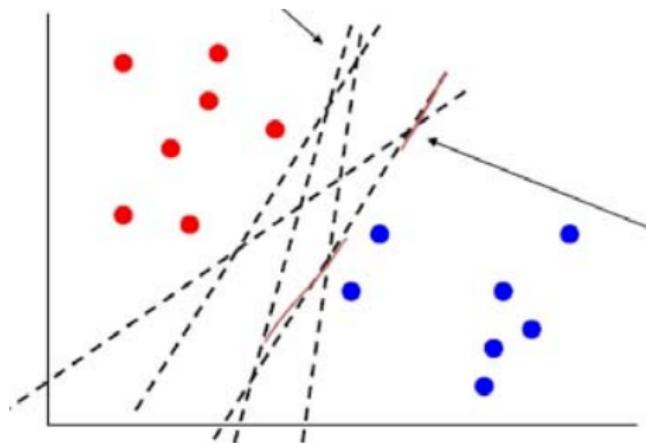
Deep Learning

- Supervised models
- Unsupervised learning (will not talk about this today)
- Generative models (will not talk about this today)

Binary Classification

- Given inputs \mathbf{x} , and outputs $t \in \{-1, 1\}$
- We want to fit a hyperplane that divides the space into half

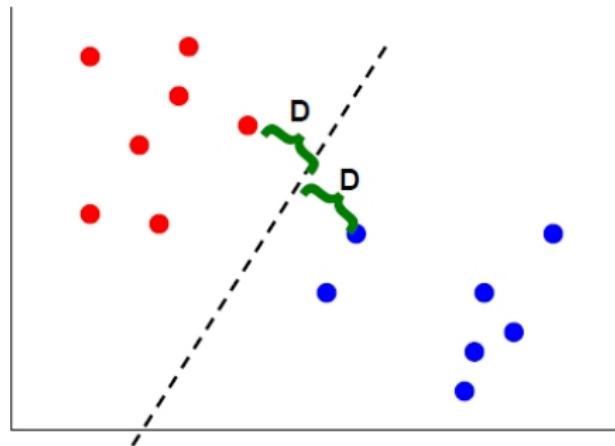
$$y_* = \text{sign}(\mathbf{w}^T \mathbf{x}_* + \mathbf{w}_0)$$



Binary Classification

- Given inputs \mathbf{x} , and outputs $t \in \{-1, 1\}$
- We want to fit a hyperplane that divides the space into half

$$y_* = \text{sign}(\mathbf{w}^T \mathbf{x}_* + \mathbf{w}_0)$$



SVMs try to maximize the margin

Non-linear Predictors

How can we make our classifier more powerful?

Non-linear Predictors

How can we make our classifier more powerful?

- Compute non-linear functions of the input

$$y_* = F(\mathbf{x}_*, \mathbf{w})$$

Non-linear Predictors

How can we make our classifier more powerful?

- Compute non-linear functions of the input

$$y_* = F(\mathbf{x}_*, \mathbf{w})$$

Two types of approaches:

Non-linear Predictors

How can we make our classifier more powerful?

- Compute non-linear functions of the input

$$y_* = F(\mathbf{x}_*, \mathbf{w})$$

Two types of approaches:

- **Kernel Trick:** Fixed functions and optimize linear parameters on non-linear mapping

$$y_* = \text{sign}(\mathbf{w}^T \phi(\mathbf{x}_*) + \mathbf{w}_0)$$

Non-linear Predictors

How can we make our classifier more powerful?

- Compute non-linear functions of the input

$$y_* = F(\mathbf{x}_*, \mathbf{w})$$

Two types of approaches:

- **Kernel Trick:** Fixed functions and optimize linear parameters on non-linear mapping

$$y_* = \text{sign}(\mathbf{w}^T \phi(\mathbf{x}_*) + \mathbf{w}_0)$$

- **Deep Learning:** Learn parametric non-linear functions

$$y_* = F(\mathbf{x}_*, \mathbf{w})$$

Why "Deep"?

Supervised Learning: Examples

Classification



classification

Denoising



regression

OCR



structured
prediction

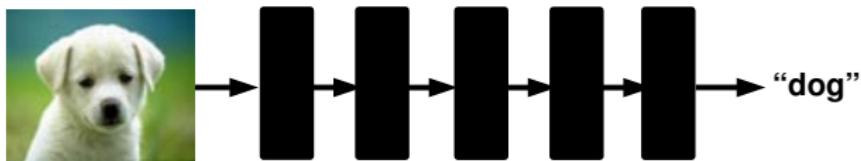
3

Ranzato

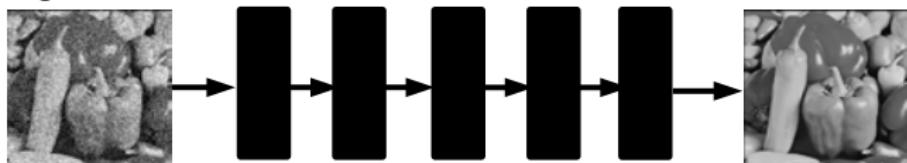
Why "Deep"?

Supervised Deep Learning

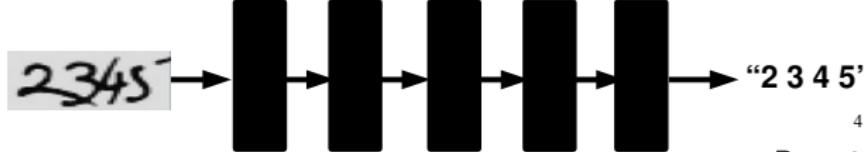
Classification



Denoising



OCR



4

Ranzato

Neural Networks

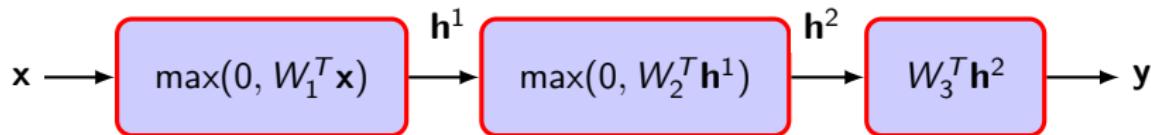
- Deep learning uses composite of simpler functions, e.g., ReLU, sigmoid, tanh, max

Neural Networks

- Deep learning uses composite of simpler functions, e.g., ReLU, sigmoid, tanh, max
- Note: a composite of linear functions is linear!

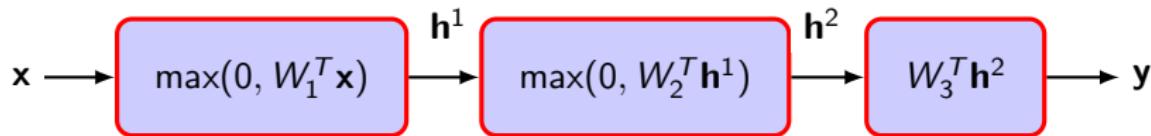
Neural Networks

- Deep learning uses composite of simpler functions, e.g., ReLU, sigmoid, tanh, max
- Note: a composite of linear functions is linear!
- Example: 2 layer NNet



Neural Networks

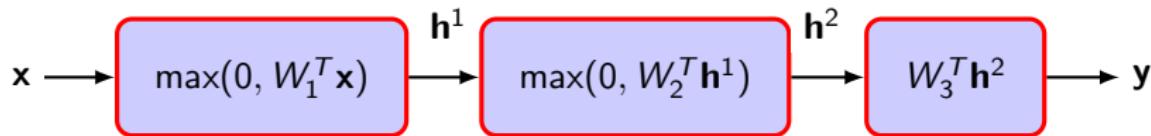
- Deep learning uses composite of simpler functions, e.g., ReLU, sigmoid, tanh, max
- Note: a composite of linear functions is linear!
- Example: 2 layer NNet



- x is the input

Neural Networks

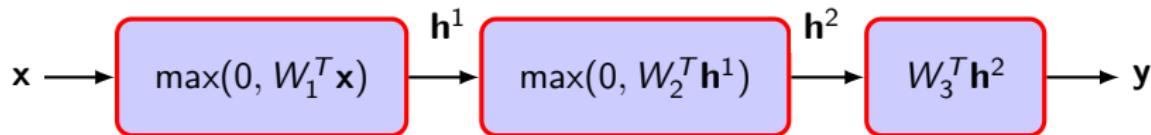
- Deep learning uses composite of simpler functions, e.g., ReLU, sigmoid, tanh, max
- Note: a composite of linear functions is linear!
- Example: 2 layer NNet



- x is the input
- y is the output (what we want to predict)

Neural Networks

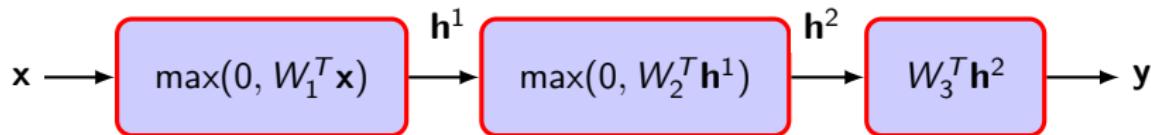
- Deep learning uses composite of simpler functions, e.g., ReLU, sigmoid, tanh, max
- Note: a composite of linear functions is linear!
- Example: 2 layer NNet



- \mathbf{x} is the input
- \mathbf{y} is the output (what we want to predict)
- \mathbf{h}^i is the i -th hidden layer

Neural Networks

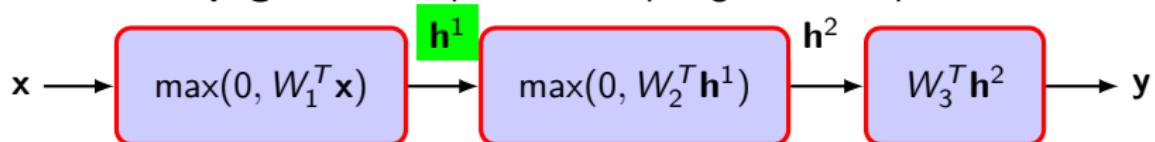
- Deep learning uses composite of simpler functions, e.g., ReLU, sigmoid, tanh, max
- Note: a composite of linear functions is linear!
- Example: 2 layer NNet



- \mathbf{x} is the input
- \mathbf{y} is the output (what we want to predict)
- \mathbf{h}^i is the i -th hidden layer
- \mathbf{W}^i are the parameters of the i -th layer

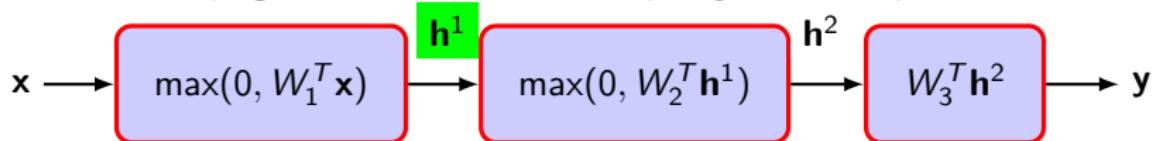
Evaluating the Function

- **Forward Propagation:** compute the output given the input



Evaluating the Function

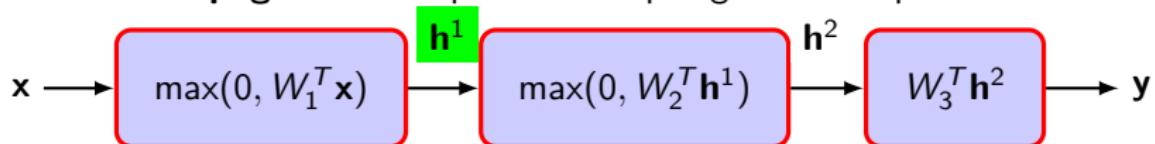
- **Forward Propagation:** compute the output given the input



- **Fully connected layer:** Each hidden unit takes as input all the units from the previous layer

Evaluating the Function

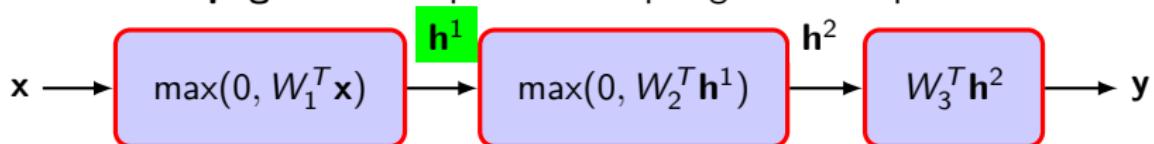
- **Forward Propagation:** compute the output given the input



- **Fully connected layer:** Each hidden unit takes as input all the units from the previous layer
- The non-linearity is called a ReLU (rectified linear unit), with $\mathbf{x} \in \Re^D$, $b^i \in \Re^{N_i}$ the biases and $\mathbf{W}^i \in \Re^{N_i \times N_{i-1}}$ the weights

Evaluating the Function

- **Forward Propagation:** compute the output given the input

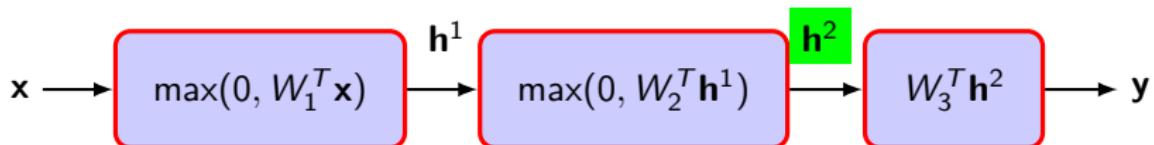


- **Fully connected layer:** Each hidden unit takes as input all the units from the previous layer
- The non-linearity is called a ReLU (rectified linear unit), with $\mathbf{x} \in \Re^D$, $b^i \in \Re^{N_i}$ the biases and $\mathbf{W}^i \in \Re^{N_i \times N_{i-1}}$ the weights
- Do it in a compositional way,

$$\mathbf{h}^1 = \max(0, \mathbf{W}^1 \mathbf{x} + b^1)$$

Evaluating the Function

- **Forward Propagation:** compute the output given the input

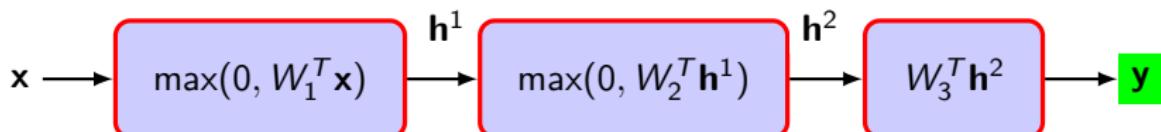


- **Fully connected layer:** Each hidden unit takes as input all the units from the previous layer
- The non-linearity is called a ReLU (rectified linear unit), with $\mathbf{x} \in \mathbb{R}^D$, $b^i \in \mathbb{R}^{N_i}$ the biases and $\mathbf{W}^i \in \mathbb{R}^{N_i \times N_{i-1}}$ the weights
- Do it in a compositional way

$$\begin{aligned}\mathbf{h}^1 &= \max(0, \mathbf{W}^1 \mathbf{x} + b^1) \\ \mathbf{h}^2 &= \max(0, \mathbf{W}^2 \mathbf{h}^1 + b^2)\end{aligned}$$

Evaluating the Function

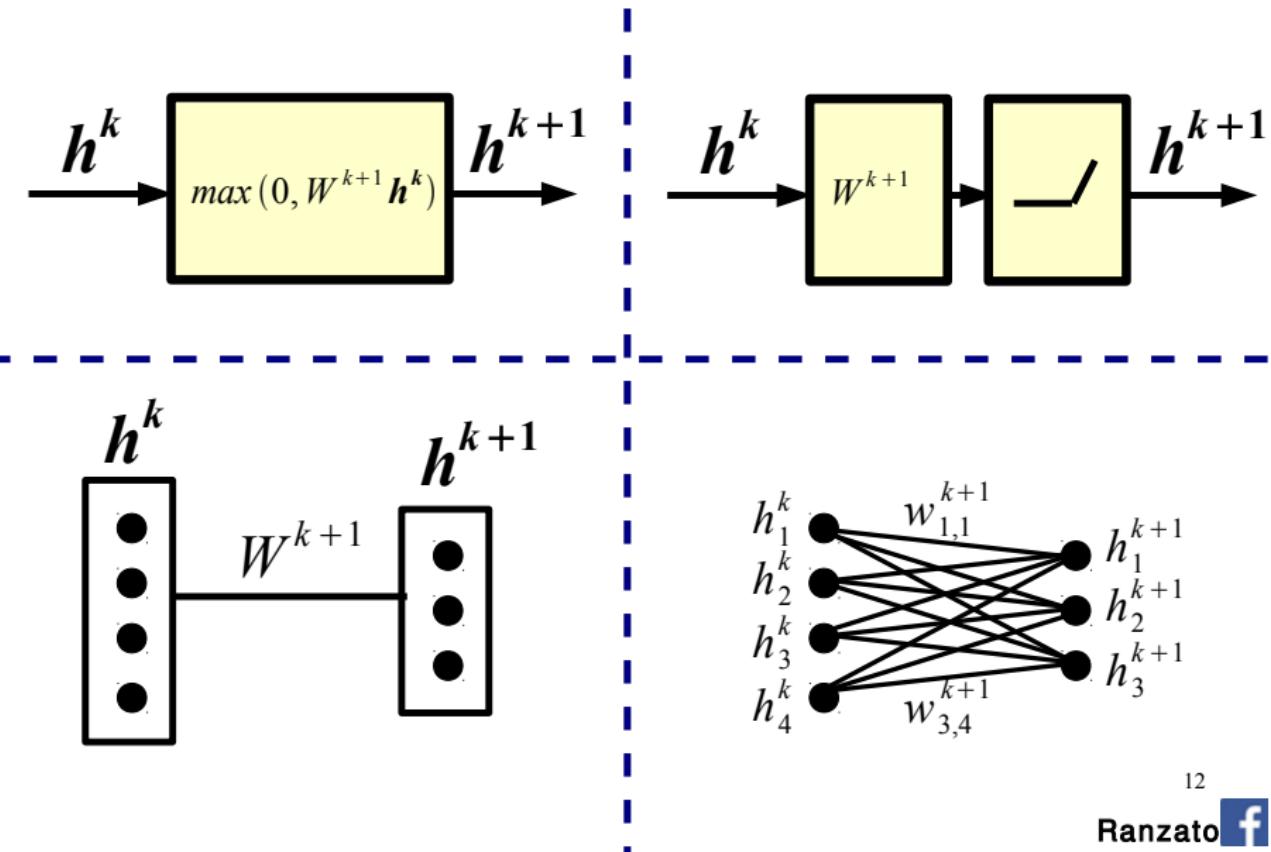
- **Forward Propagation:** compute the output given the input



- **Fully connected layer:** Each hidden unit takes as input all the units from the previous layer
- The non-linearity is called a ReLU (rectified linear unit), with $\mathbf{x} \in \Re^D$, $b^i \in \Re^{N_i}$ the biases and $\mathbf{W}^i \in \Re^{N_i \times N_{i-1}}$ the weights
- Do it in a compositional way

$$\begin{aligned}\mathbf{h}^1 &= \max(0, \mathbf{W}^1 \mathbf{x} + b^1) \\ \mathbf{h}^2 &= \max(0, \mathbf{W}^2 \mathbf{h}^1 + b^2) \\ \mathbf{y} &= \max(0, \mathbf{W}^3 \mathbf{h}^2 + b^3)\end{aligned}$$

Alternative Graphical Representation



Relu Interpretation

- Piece-wise linear tiling: mapping is locally linear.

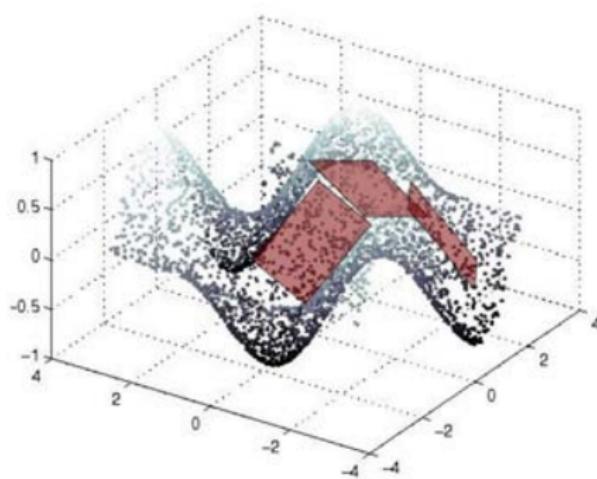


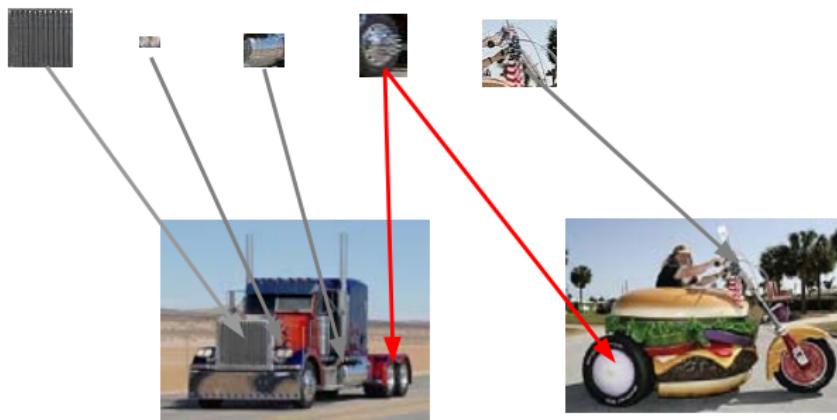
Figure : by M. Ranzato

Why Hierarchical?

Interpretation

[1 1 0 0 0 1 0 1 0 0 0 0 1 1 0 1 ...] motorbike

[0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 0 ...] truck

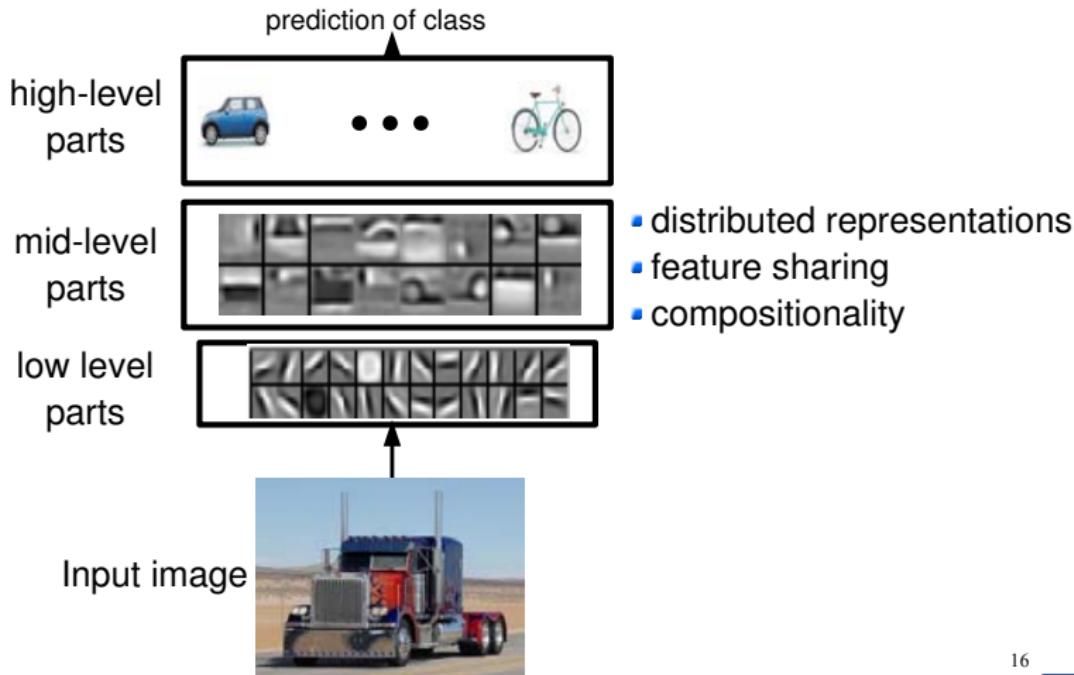


15

Ranzato

Why Hierarchical?

Interpretation

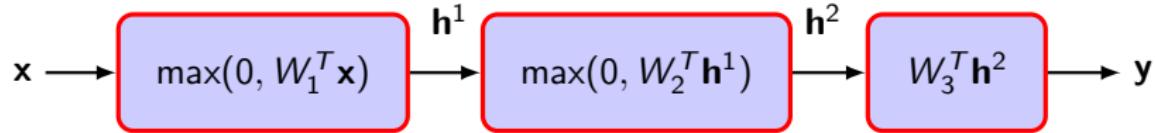


Lee et al. "Convolutional DBN's ..." ICML 2009

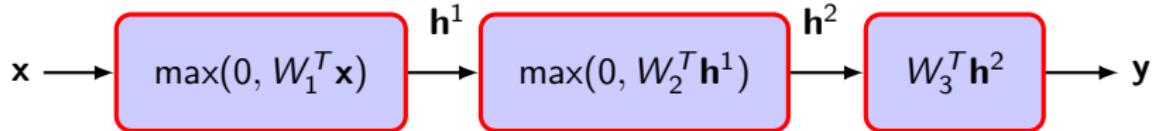
16

Ranzato

Learning

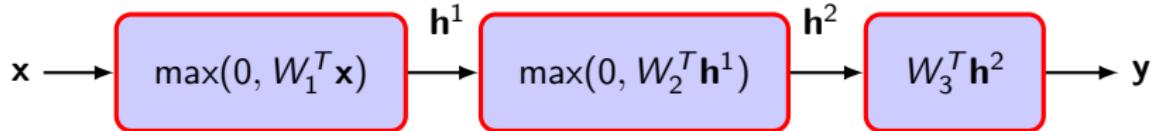


Learning



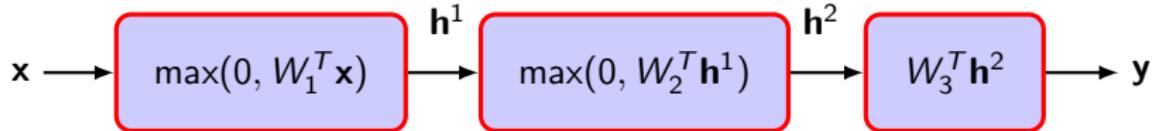
- We want to estimate the parameters, biases and hyper-parameters (e.g., number of layers, number of units) such that we do good predictions

Learning



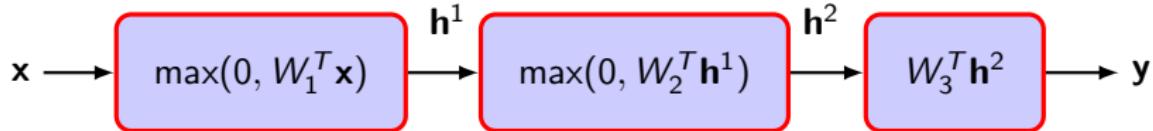
- We want to estimate the parameters, biases and hyper-parameters (e.g., number of layers, number of units) such that we do good predictions
- Collect a training set of input-output pairs $\{\mathbf{x}, t\}$

Learning



- We want to estimate the parameters, biases and hyper-parameters (e.g., number of layers, number of units) such that we do good predictions
- Collect a training set of input-output pairs $\{\mathbf{x}, t\}$
- Encode the output with 1-K encoding $t = [0, \dots, 1, \dots, 0]$

Learning

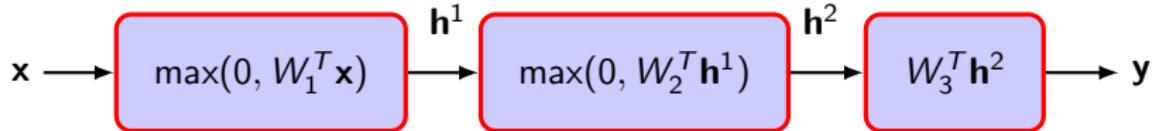


- We want to estimate the parameters, biases and hyper-parameters (e.g., number of layers, number of units) such that we do good predictions
- Collect a training set of input-output pairs $\{\mathbf{x}, t\}$
- Encode the output with 1-K encoding $t = [0, \dots, 1, \dots, 0]$
- Define a loss per training example and minimize the empirical risk

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \mathcal{R}(\mathbf{w})$$

with N number of examples, \mathcal{R} a regularizer, and \mathbf{w} contains all parameters

Learning



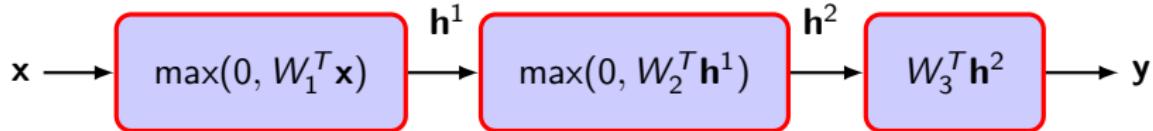
- We want to estimate the parameters, biases and hyper-parameters (e.g., number of layers, number of units) such that we do good predictions
- Collect a training set of input-output pairs $\{\mathbf{x}, t\}$
- Encode the output with 1-K encoding $t = [0, \dots, 1, \dots, 0]$
- Define a loss per training example and minimize the empirical risk

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \mathcal{R}(\mathbf{w})$$

with N number of examples, \mathcal{R} a regularizer, and \mathbf{w} contains all parameters

- What do we want to use as ℓ ?

Learning



- We want to estimate the parameters, biases and hyper-parameters (e.g., number of layers, number of units) such that we do good predictions
- Collect a training set of input-output pairs $\{\mathbf{x}, t\}$
- Encode the output with 1-K encoding $t = [0, \dots, 1, \dots, 0]$
- Define a loss per training example and minimize the empirical risk

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \mathcal{R}(\mathbf{w})$$

with N number of examples, \mathcal{R} a regularizer, and \mathbf{w} contains all parameters

- What do we want to use as ℓ ?
- The **task loss**: how we are going to evaluate at test time

Loss Functions

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} \sum_i \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \mathcal{R}(\mathbf{w})$$

Loss Functions

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} \sum_i \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \mathcal{R}(\mathbf{w})$$

- The task loss is too difficult to compute, so one uses a surrogate that is typically convex

Loss Functions

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} \sum_i \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \mathcal{R}(\mathbf{w})$$

- The task loss is too difficult to compute, so one uses a surrogate that is typically convex
- Probability of class k given input (softmax):

$$p(c_k = 1 | \mathbf{x}) = \frac{\exp(y_k)}{\sum_{j=1}^C \exp(y_j)}$$

Loss Functions

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} \sum_i \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \mathcal{R}(\mathbf{w})$$

- The task loss is too difficult to compute, so one uses a surrogate that is typically convex
- Probability of class k given input (softmax):

$$p(c_k = 1 | \mathbf{x}) = \frac{\exp(y_k)}{\sum_{j=1}^C \exp(y_j)}$$

- Cross entropy is the most used loss function for classification

$$\ell(\mathbf{x}, t, \mathbf{w}) = - \sum_i t^{(i)} \log p(c_i | \mathbf{x})$$

Loss Functions

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} \sum_i \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \mathcal{R}(\mathbf{w})$$

- The task loss is too difficult to compute, so one uses a surrogate that is typically convex
- Probability of class k given input (softmax):

$$p(c_k = 1 | \mathbf{x}) = \frac{\exp(y_k)}{\sum_{j=1}^C \exp(y_j)}$$

- Cross entropy is the most used loss function for classification

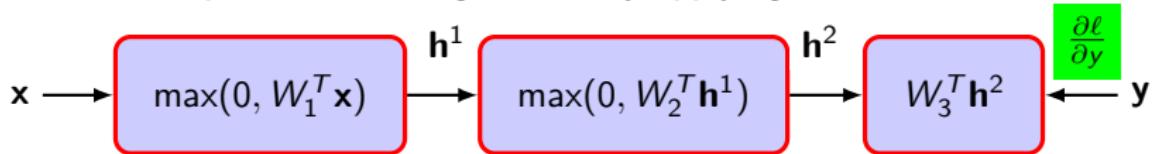
$$\ell(\mathbf{x}, t, \mathbf{w}) = - \sum_i t^{(i)} \log p(c_i | \mathbf{x})$$

- Use gradient descent to train the network

$$\min_{\mathbf{w}} \frac{1}{N} \sum_i \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \mathcal{R}(\mathbf{w})$$

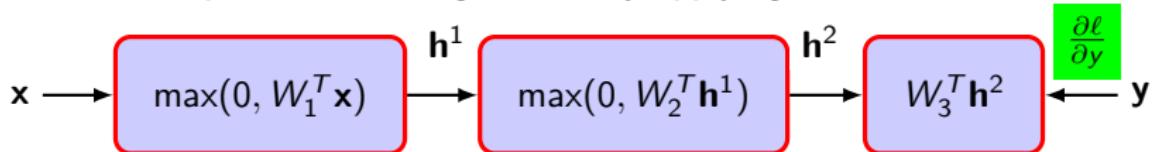
Backpropagation

- Efficient computation of the gradients by applying the chain rule



Backpropagation

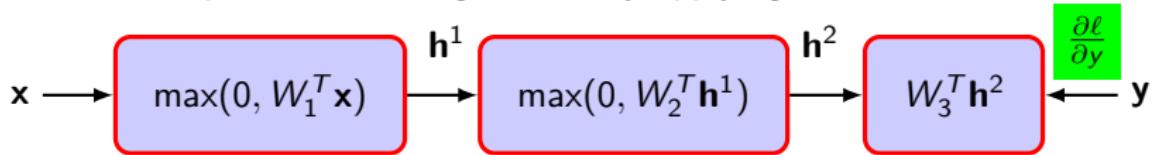
- Efficient computation of the gradients by applying the chain rule



$$p(c_k = 1 | \mathbf{x}) = \frac{\exp(y_k)}{\sum_{j=1}^C \exp(y_j)}$$

Backpropagation

- Efficient computation of the gradients by applying the chain rule

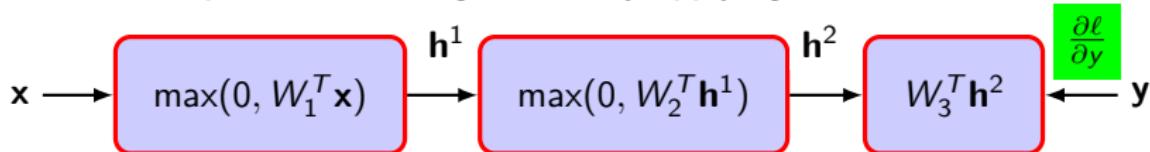


$$p(c_k = 1 | \mathbf{x}) = \frac{\exp(y_k)}{\sum_{j=1}^C \exp(y_j)}$$

$$\ell(\mathbf{x}, t, \mathbf{w}) = - \sum_i t^{(i)} \log p(c_i | \mathbf{x})$$

Backpropagation

- Efficient computation of the gradients by applying the chain rule



$$p(c_k = 1 | \mathbf{x}) = \frac{\exp(y_k)}{\sum_{j=1}^C \exp(y_j)}$$

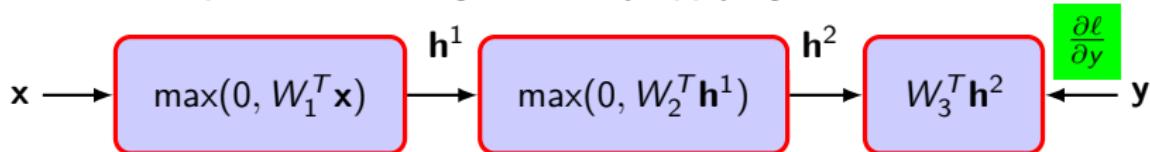
$$\ell(\mathbf{x}, t, \mathbf{w}) = - \sum_i t^{(i)} \log p(c_i | \mathbf{x})$$

- Compute the derivative of loss w.r.t. the output

$$\frac{\partial \ell}{\partial y} = p(c| \mathbf{x}) - t$$

Backpropagation

- Efficient computation of the gradients by applying the chain rule



$$p(c_k = 1 | \mathbf{x}) = \frac{\exp(y_k)}{\sum_{j=1}^C \exp(y_j)}$$

$$\ell(\mathbf{x}, t, \mathbf{w}) = - \sum_i t^{(i)} \log p(c_i | \mathbf{x})$$

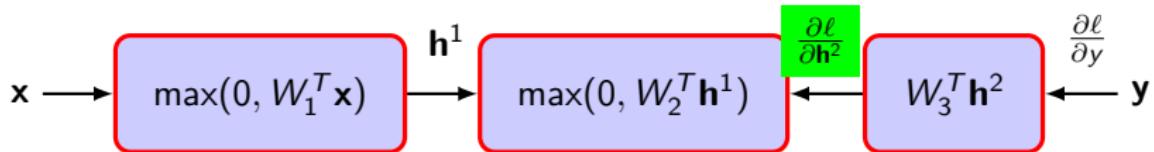
- Compute the derivative of loss w.r.t. the output

$$\frac{\partial \ell}{\partial y} = p(c| \mathbf{x}) - t$$

- Note that the **forward pass** is necessary to compute $\frac{\partial \ell}{\partial y}$

Backpropagation

- Efficient computation of the gradients by applying the chain rule

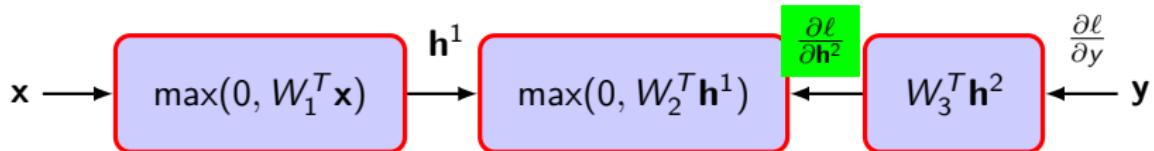


- Compute the derivative of loss w.r.t the output

$$\frac{\partial \ell}{\partial y} = p(c|\mathbf{x}) - t$$

Backpropagation

- Efficient computation of the gradients by applying the chain rule



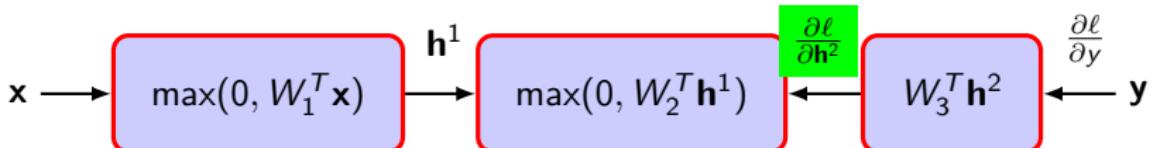
- Compute the derivative of loss w.r.t the output

$$\frac{\partial \ell}{\partial y} = p(c|\mathbf{x}) - t$$

- Given $\frac{\partial \ell}{\partial y}$ if we can compute the Jacobian of each module

Backpropagation

- Efficient computation of the gradients by applying the chain rule



- Compute the derivative of loss w.r.t the output

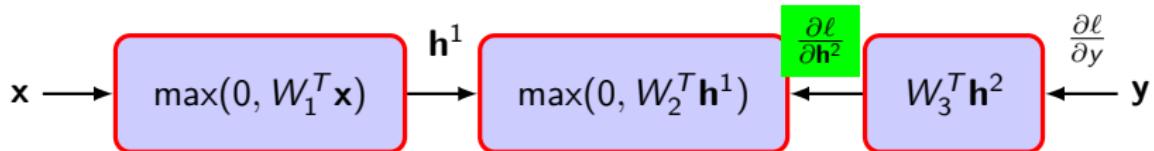
$$\frac{\partial \ell}{\partial y} = p(c|\mathbf{x}) - t$$

- Given $\frac{\partial \ell}{\partial y}$ if we can compute the Jacobian of each module

$$\frac{\partial \ell}{\partial \mathbf{W}^3} =$$

Backpropagation

- Efficient computation of the gradients by applying the chain rule



- Compute the derivative of loss w.r.t the output

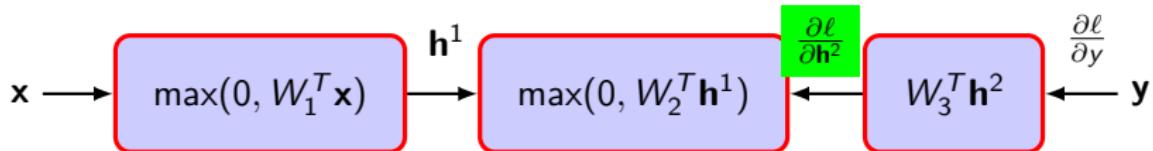
$$\frac{\partial \ell}{\partial y} = p(c|\mathbf{x}) - t$$

- Given $\frac{\partial \ell}{\partial y}$ if we can compute the Jacobian of each module

$$\frac{\partial \ell}{\partial \mathbf{W}^3} = \frac{\partial \ell}{\partial y} \frac{\partial y}{\partial \mathbf{W}^3} =$$

Backpropagation

- Efficient computation of the gradients by applying the chain rule



- Compute the derivative of loss w.r.t the output

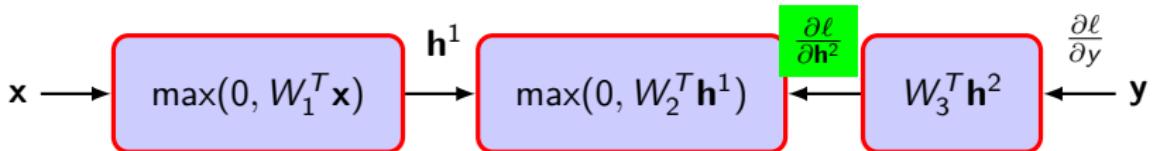
$$\frac{\partial \ell}{\partial y} = p(c|\mathbf{x}) - t$$

- Given $\frac{\partial \ell}{\partial y}$ if we can compute the Jacobian of each module

$$\frac{\partial \ell}{\partial \mathbf{W}^3} = \frac{\partial \ell}{\partial y} \frac{\partial y}{\partial \mathbf{W}^3} = (p(c|\mathbf{x}) - t)(\mathbf{h}^2)^T$$

Backpropagation

- Efficient computation of the gradients by applying the chain rule



- Compute the derivative of loss w.r.t the output

$$\frac{\partial \ell}{\partial y} = p(c|\mathbf{x}) - t$$

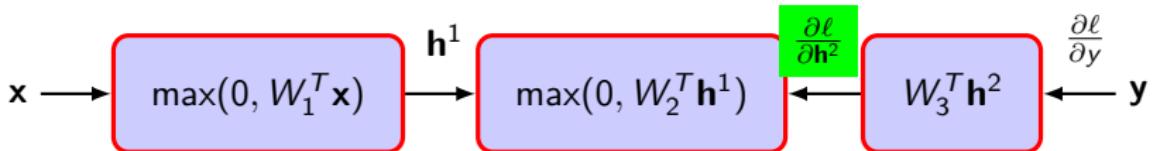
- Given $\frac{\partial \ell}{\partial y}$ if we can compute the Jacobian of each module

$$\frac{\partial \ell}{\partial \mathbf{W}^3} = \frac{\partial \ell}{\partial y} \frac{\partial y}{\partial \mathbf{W}^3} = (p(c|\mathbf{x}) - t)(\mathbf{h}^2)^T$$

$$\frac{\partial \ell}{\partial \mathbf{h}^2} =$$

Backpropagation

- Efficient computation of the gradients by applying the chain rule



- Compute the derivative of loss w.r.t the output

$$\frac{\partial \ell}{\partial y} = p(c|\mathbf{x}) - t$$

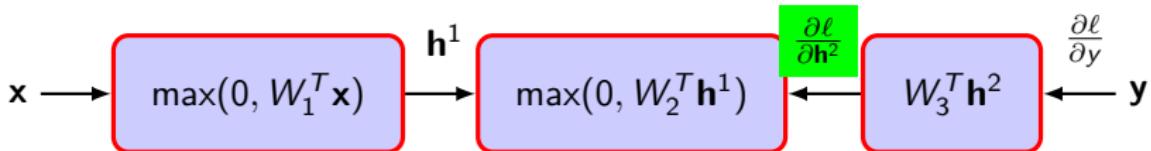
- Given $\frac{\partial \ell}{\partial y}$ if we can compute the Jacobian of each module

$$\frac{\partial \ell}{\partial \mathbf{W}^3} = \frac{\partial \ell}{\partial y} \frac{\partial y}{\partial \mathbf{W}^3} = (p(c|\mathbf{x}) - t)(\mathbf{h}^2)^T$$

$$\frac{\partial \ell}{\partial \mathbf{h}^2} = \frac{\partial \ell}{\partial y} \frac{\partial y}{\partial \mathbf{h}^2} =$$

Backpropagation

- Efficient computation of the gradients by applying the chain rule



- Compute the derivative of loss w.r.t the output

$$\frac{\partial \ell}{\partial y} = p(c|\mathbf{x}) - t$$

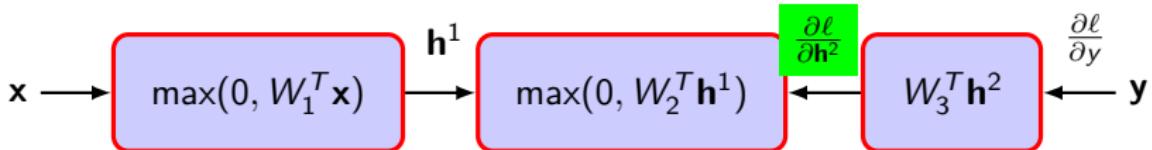
- Given $\frac{\partial \ell}{\partial y}$ if we can compute the Jacobian of each module

$$\frac{\partial \ell}{\partial \mathbf{W}^3} = \frac{\partial \ell}{\partial y} \frac{\partial y}{\partial \mathbf{W}^3} = (p(c|\mathbf{x}) - t)(\mathbf{h}^2)^T$$

$$\frac{\partial \ell}{\partial \mathbf{h}^2} = \frac{\partial \ell}{\partial y} \frac{\partial y}{\partial \mathbf{h}^2} = (\mathbf{W}^3)^T (p(c|\mathbf{x}) - t)$$

Backpropagation

- Efficient computation of the gradients by applying the chain rule



- Compute the derivative of loss w.r.t the output

$$\frac{\partial \ell}{\partial y} = p(c|\mathbf{x}) - t$$

- Given $\frac{\partial \ell}{\partial y}$ if we can compute the Jacobian of each module

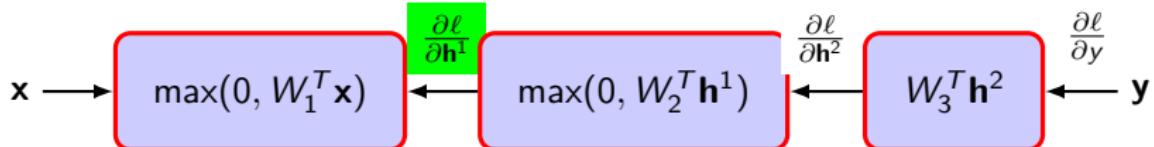
$$\frac{\partial \ell}{\partial \mathbf{W}^3} = \frac{\partial \ell}{\partial y} \frac{\partial y}{\partial \mathbf{W}^3} = (p(c|\mathbf{x}) - t)(\mathbf{h}^2)^T$$

$$\frac{\partial \ell}{\partial \mathbf{h}^2} = \frac{\partial \ell}{\partial y} \frac{\partial y}{\partial \mathbf{h}^2} = (\mathbf{W}^3)^T (p(c|\mathbf{x}) - t)$$

- Need to compute gradient w.r.t. inputs and parameters in each layer

Backpropagation

- Efficient computation of the gradients by applying the chain rule

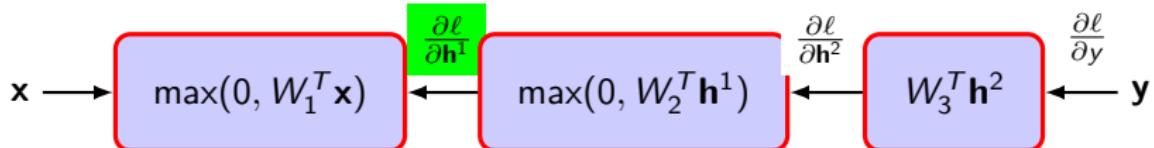


$$\frac{\partial \ell}{\partial \mathbf{h}^2} = \frac{\partial \ell}{\partial y} \frac{\partial y}{\partial \mathbf{h}^2} = (W^3)^T (p(c|\mathbf{x}) - t)$$

- Given $\frac{\partial \ell}{\partial \mathbf{h}^2}$ if we can compute the Jacobian of each module

Backpropagation

- Efficient computation of the gradients by applying the chain rule



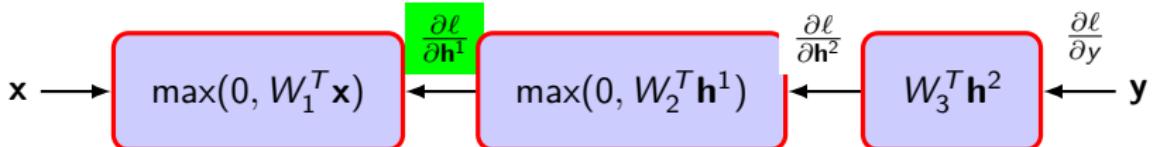
$$\frac{\partial \ell}{\partial \mathbf{h}^2} = \frac{\partial \ell}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{h}^2} = (W^3)^T (p(c|\mathbf{x}) - t)$$

- Given $\frac{\partial \ell}{\partial \mathbf{h}^2}$ if we can compute the Jacobian of each module

$$\frac{\partial \ell}{\partial W^2} =$$

Backpropagation

- Efficient computation of the gradients by applying the chain rule



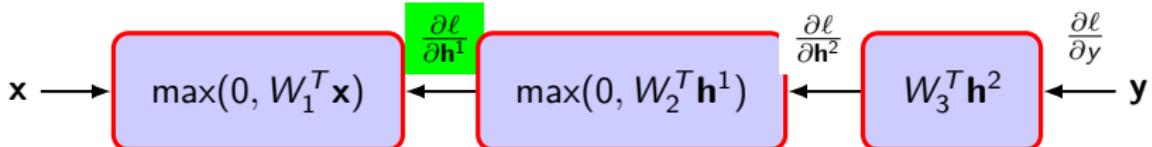
$$\frac{\partial \ell}{\partial \mathbf{h}^2} = \frac{\partial \ell}{\partial y} \frac{\partial y}{\partial \mathbf{h}^2} = (W^3)^T (p(c|\mathbf{x}) - t)$$

- Given $\frac{\partial \ell}{\partial \mathbf{h}^2}$ if we can compute the Jacobian of each module

$$\frac{\partial \ell}{\partial W^2} = \frac{\partial \ell}{\partial \mathbf{h}^2} \frac{\partial \mathbf{h}^2}{\partial W^2}$$

Backpropagation

- Efficient computation of the gradients by applying the chain rule



$$\frac{\partial \ell}{\partial \mathbf{h}^2} = \frac{\partial \ell}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{h}^2} = (\mathbf{W}^3)^T (p(c|\mathbf{x}) - t)$$

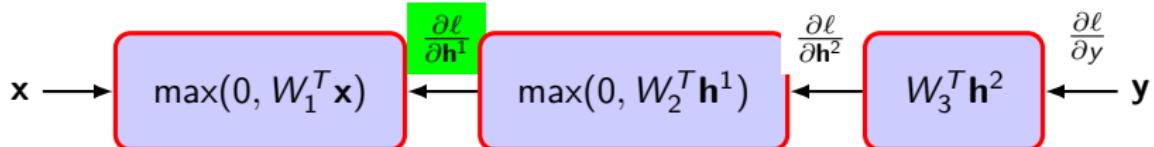
- Given $\frac{\partial \ell}{\partial \mathbf{h}^2}$ if we can compute the Jacobian of each module

$$\frac{\partial \ell}{\partial \mathbf{W}^2} = \frac{\partial \ell}{\partial \mathbf{h}^2} \frac{\partial \mathbf{h}^2}{\partial \mathbf{W}^2}$$

$$\frac{\partial \ell}{\partial \mathbf{h}^1} =$$

Backpropagation

- Efficient computation of the gradients by applying the chain rule



$$\frac{\partial \ell}{\partial \mathbf{h}^2} = \frac{\partial \ell}{\partial y} \frac{\partial y}{\partial \mathbf{h}^2} = (\mathbf{W}^3)^T (p(c|\mathbf{x}) - t)$$

- Given $\frac{\partial \ell}{\partial \mathbf{h}^2}$ if we can compute the Jacobian of each module

$$\frac{\partial \ell}{\partial \mathbf{W}^2} = \frac{\partial \ell}{\partial \mathbf{h}^2} \frac{\partial \mathbf{h}^2}{\partial \mathbf{W}^2}$$

$$\frac{\partial \ell}{\partial \mathbf{h}^1} = \frac{\partial \ell}{\partial \mathbf{h}^2} \frac{\partial \mathbf{h}^2}{\partial \mathbf{h}^1}$$

Gradient Descent

- Gradient descent is a first order method, where one takes steps proportional to the negative of the gradient of the function at the current point

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n)$$

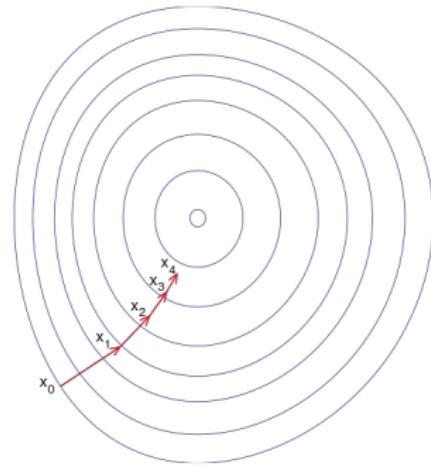
- Example: $f(x) = x^4 - 3x^3 + 2$

```
x_old = 0
x_new = 6 # The algorithm starts at x=6
gamma = 0.01 # step size
precision = 0.00001

def f_derivative(x):
    return 4 * x**3 - 9 * x**2

while abs(x_new - x_old) > precision:
    x_old = x_new
    x_new = x_old - gamma * f_derivative(x_old)

print("Local minimum occurs at", x_new)
```



Learning via Gradient Descent

- Use gradient descent to train the network

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^N \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \mathcal{R}(\mathbf{w})$$

Learning via Gradient Descent

- Use gradient descent to train the network

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^N \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \mathcal{R}(\mathbf{w})$$

- We need to compute at each iteration

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma_n \nabla L(\mathbf{w}_n)$$

Learning via Gradient Descent

- Use gradient descent to train the network

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^N \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \mathcal{R}(\mathbf{w})$$

- We need to compute at each iteration

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma_n \nabla L(\mathbf{w}_n)$$

- Use the backward pass to compute $\nabla L(\mathbf{w}_n)$ efficiently

Learning via Gradient Descent

- Use gradient descent to train the network

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^N \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \mathcal{R}(\mathbf{w})$$

- We need to compute at each iteration

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma_n \nabla L(\mathbf{w}_n)$$

- Use the backward pass to compute $\nabla L(\mathbf{w}_n)$ efficiently
- Recall that the backward pass requires the forward pass first

Toy Code (Matlab): Neural Net Trainer

```
% F-PROP
for i = 1 : nr_layers - 1
    [h{i} jac{i}] = nonlinearity(W{i} * h{i-1} + b{i});
end
h{nr_layers-1} = W{nr_layers-1} * h{nr_layers-2} + b{nr_layers-1};
prediction = softmax(h{l-1});

% CROSS ENTROPY LOSS
loss = - sum(sum(log(prediction) .* target)) / batch_size;

% B-PROP
dh{l-1} = prediction - target;
for i = nr_layers - 1 : -1 : 1
    Wgrad{i} = dh{i} * h{i-1}';
    bgrad{i} = sum(dh{i}, 2);
    dh{i-1} = (W{i}' * dh{i}) .* jac{i-1};
end

% UPDATE
for i = 1 : nr_layers - 1
    W{i} = W{i} - (lr / batch_size) * Wgrad{i};
    b{i} = b{i} - (lr / batch_size) * bgrad{i};
end
```

Dealing with Big Data

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^N \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \mathcal{R}(\mathbf{w})$$

- We need to compute at each iteration

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma_n \nabla L(\mathbf{w}_n)$$

Dealing with Big Data

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^N \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \mathcal{R}(\mathbf{w})$$

- We need to compute at each iteration

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma_n \nabla L(\mathbf{w}_n)$$

with

$$\nabla L(\mathbf{w}_n) = \frac{1}{N} \sum_{i=1}^N \nabla \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \nabla \mathcal{R}(\mathbf{w})$$

Dealing with Big Data

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^N \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \mathcal{R}(\mathbf{w})$$

- We need to compute at each iteration

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma_n \nabla L(\mathbf{w}_n)$$

with

$$\nabla L(\mathbf{w}_n) = \frac{1}{N} \sum_{i=1}^N \nabla \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \nabla \mathcal{R}(\mathbf{w})$$

- Too expensive when having millions of examples

Dealing with Big Data

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^N \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \mathcal{R}(\mathbf{w})$$

- We need to compute at each iteration

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma_n \nabla L(\mathbf{w}_n)$$

with

$$\nabla L(\mathbf{w}_n) = \frac{1}{N} \sum_{i=1}^N \nabla \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \nabla \mathcal{R}(\mathbf{w})$$

- Too expensive when having millions of examples
- Instead approximate the gradient with a mini-batch (subset of examples)

$$\frac{1}{N} \sum_{i=1}^N \nabla \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) \approx \sum_{i \in \mathcal{S}} \frac{1}{|\mathcal{S}|} \nabla \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)})$$

Dealing with Big Data

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^N \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \mathcal{R}(\mathbf{w})$$

- We need to compute at each iteration

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma_n \nabla L(\mathbf{w}_n)$$

with

$$\nabla L(\mathbf{w}_n) = \frac{1}{N} \sum_{i=1}^N \nabla \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \nabla \mathcal{R}(\mathbf{w})$$

- Too expensive when having millions of examples
- Instead approximate the gradient with a mini-batch (subset of examples)

$$\frac{1}{N} \sum_{i=1}^N \nabla \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) \approx \sum_{i \in \mathcal{S}} \frac{1}{|\mathcal{S}|} \nabla \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)})$$

- This is called **stochastic gradient descent**

Stochastic Gradient Descent with Momentum

- Stochastic Gradient Descent update

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma_n \nabla L(\mathbf{w}_n)$$

with

$$\nabla L(\mathbf{w}_n) = \sum_{i \in \mathcal{S}} \frac{1}{|\mathcal{S}|} \nabla \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \nabla \mathcal{R}(\mathbf{w})$$

Stochastic Gradient Descent with Momentum

- Stochastic Gradient Descent update

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma_n \nabla L(\mathbf{w}_n)$$

with

$$\nabla L(\mathbf{w}_n) = \sum_{i \in \mathcal{S}} \frac{1}{|\mathcal{S}|} \nabla \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \nabla \mathcal{R}(\mathbf{w})$$

- We can also use momentum

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} - \gamma \Delta \\ \Delta &\leftarrow \kappa \Delta + \nabla L\end{aligned}$$

Stochastic Gradient Descent with Momentum

- Stochastic Gradient Descent update

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma_n \nabla L(\mathbf{w}_n)$$

with

$$\nabla L(\mathbf{w}_n) = \sum_{i \in \mathcal{S}} \frac{1}{|\mathcal{S}|} \nabla \ell(\mathbf{w}, \mathbf{x}^{(i)}, t^{(i)}) + \nabla \mathcal{R}(\mathbf{w})$$

- We can also use momentum

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} - \gamma \Delta \\ \Delta &\leftarrow \kappa \Delta + \nabla L\end{aligned}$$

- Many other variants exist

How to deal with large Input Spaces

- Images can have millions of pixels, i.e., \mathbf{x} is very high dimensional

How to deal with large Input Spaces

- Images can have millions of pixels, i.e., \mathbf{x} is very high dimensional
- Prohibitive to have fully-connected layer

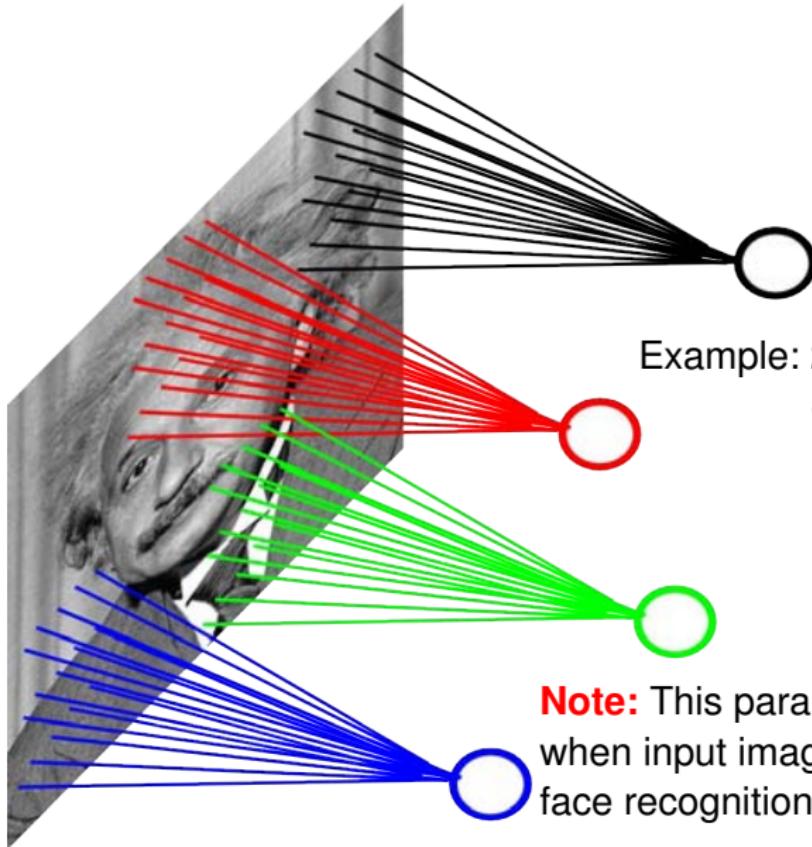
How to deal with large Input Spaces

- Images can have millions of pixels, i.e., \mathbf{x} is very high dimensional
- Prohibitive to have fully-connected layer
- We can use a **locally connected layer**

How to deal with large Input Spaces

- Images can have millions of pixels, i.e., \mathbf{x} is very high dimensional
- Prohibitive to have fully-connected layer
- We can use a **locally connected layer**
- This is good when the **input is registered**

Locally Connected Layer

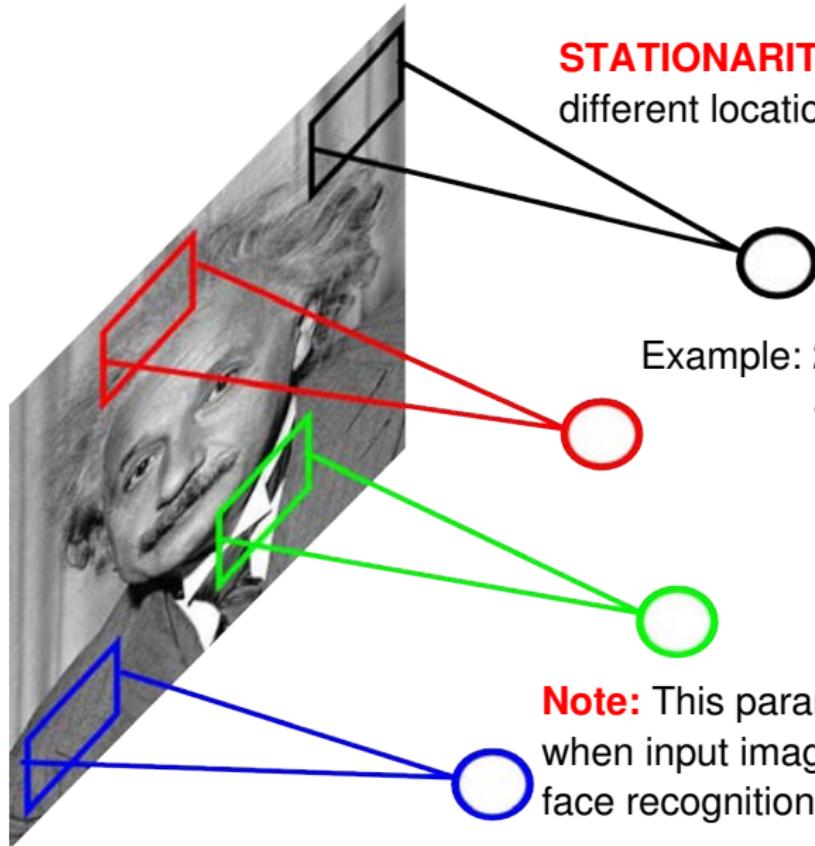


Example:
200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g.,
face recognition).

34

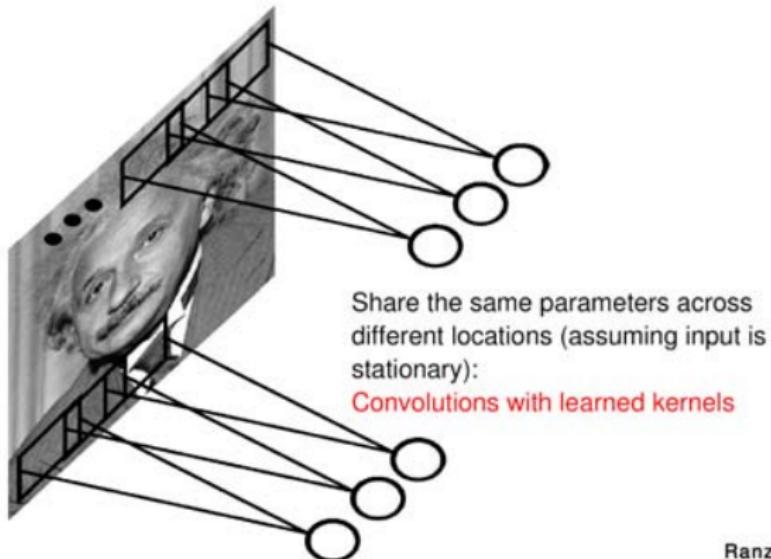
Locally Connected Layer



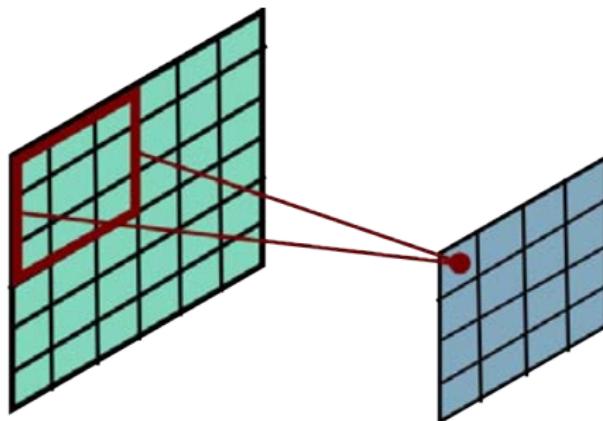
35

Convolutional Neural Net

- Idea: statistics are similar at different locations (Lecun 1998)
- Connect each hidden unit to a small input patch and share the weight across space
- This is called a **convolution layer** and the network is a **convolutional network**



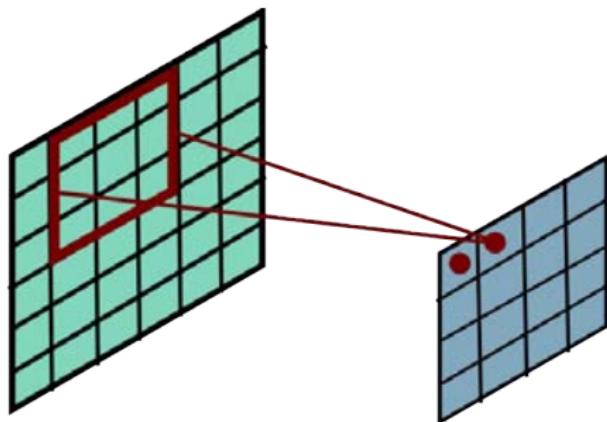
Convolutional Layer



Ranzato

$$h_j^n = \max(0, \sum_{k=1}^K h_k^{n-1} * w_{jk}^n)$$

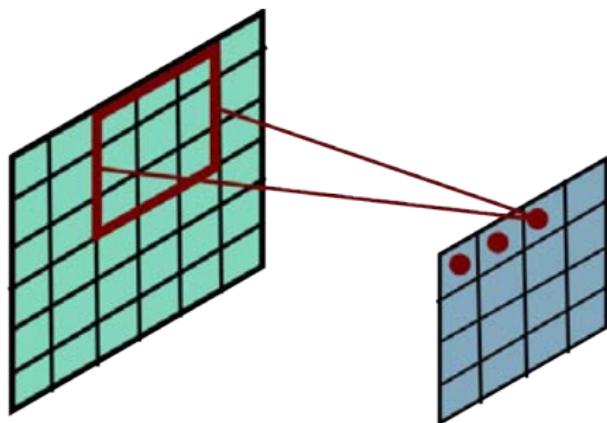
Convolutional Layer



Ranzato

$$h_j^n = \max(0, \sum_{k=1}^K h_k^{n-1} * w_{jk}^n)$$

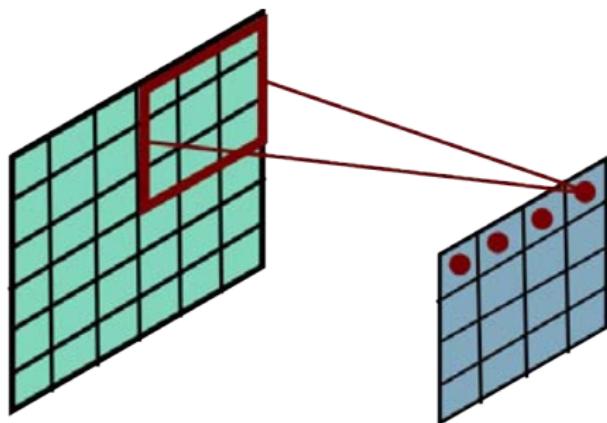
Convolutional Layer



Ranzato

$$h_j^n = \max(0, \sum_{k=1}^K h_k^{n-1} * w_{jk}^n)$$

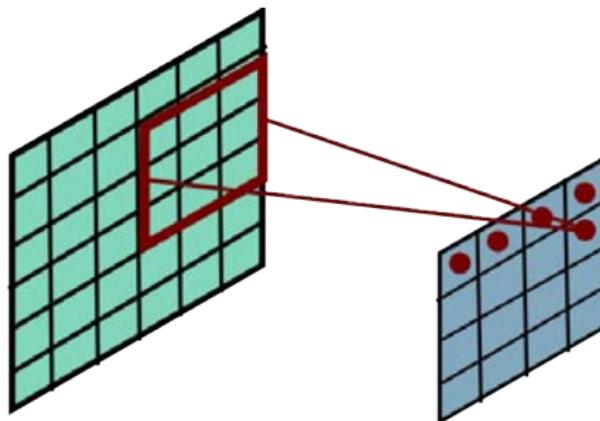
Convolutional Layer



Ranzato

$$h_j^n = \max(0, \sum_{k=1}^K h_k^{n-1} * w_{jk}^n)$$

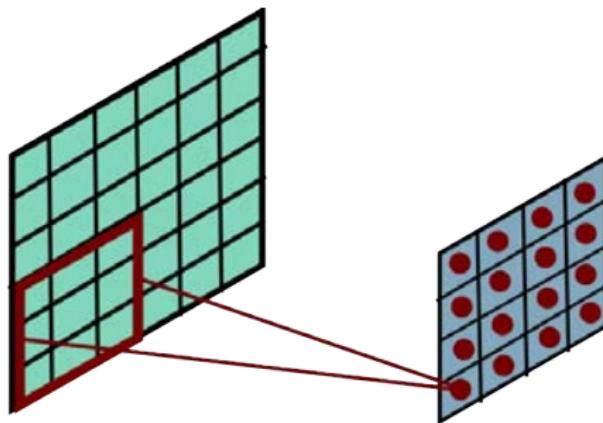
Convolutional Layer



Ranzato

$$h_j^n = \max(0, \sum_{k=1}^K h_k^{n-1} * w_{jk}^n)$$

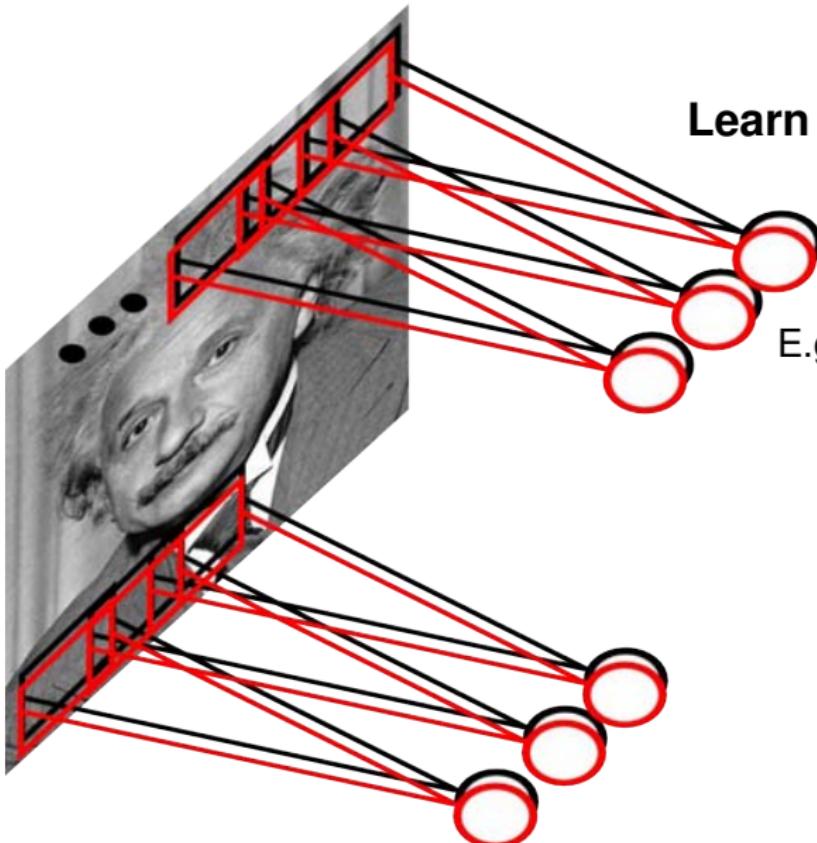
Convolutional Layer



Ranzato

$$h_j^n = \max(0, \sum_{k=1}^K h_k^{n-1} * w_{jk}^n)$$

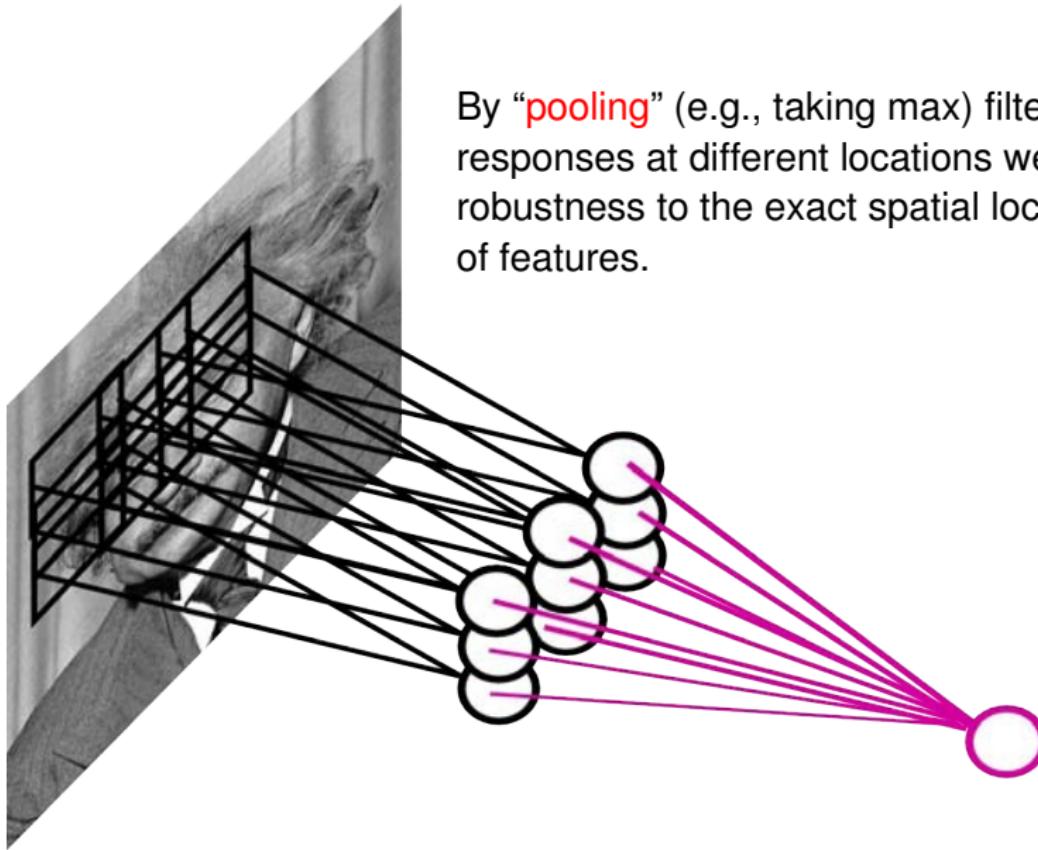
Convolutional Layer



Learn multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

Pooling Layer

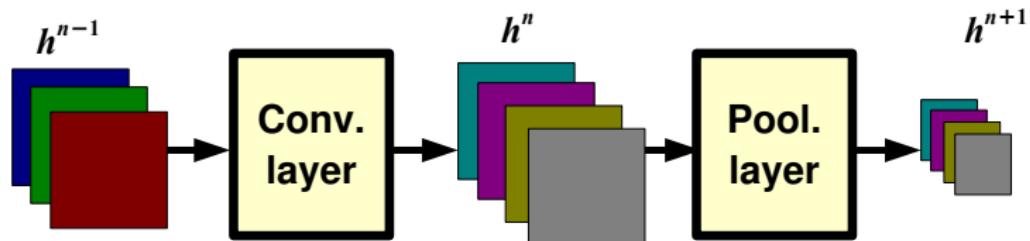


61

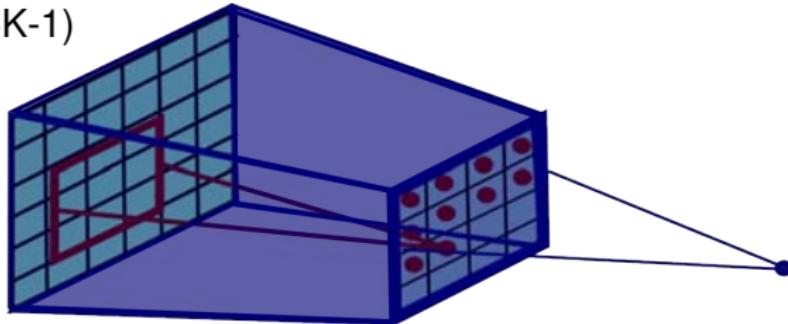
Pooling Options

- **Max Pooling:** return the maximal argument
- **Average Pooling:** return the average of the arguments
- Other types of pooling exist: L_2 pooling

Pooling Layer: Receptive Field Size



If convolutional filters have size $K \times K$ and stride 1, and pooling layer has pools of size $P \times P$, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size:
 $(P+K-1) \times (P+K-1)$



Now let's make this very deep

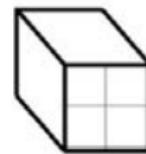
Convolutional Neural Networks (CNN)

- Remember from your image processing / computer vision course about filtering?

Input "image"



Filter



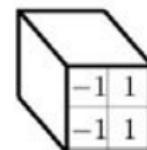
Convolutional Neural Networks (CNN)

- If our filter was $[-1, 1]$, we got a vertical edge detector

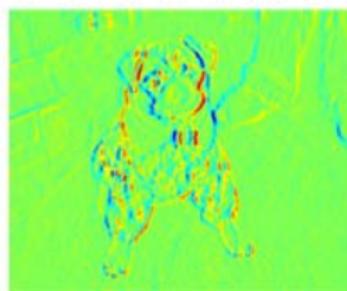
Input "image"



Filter

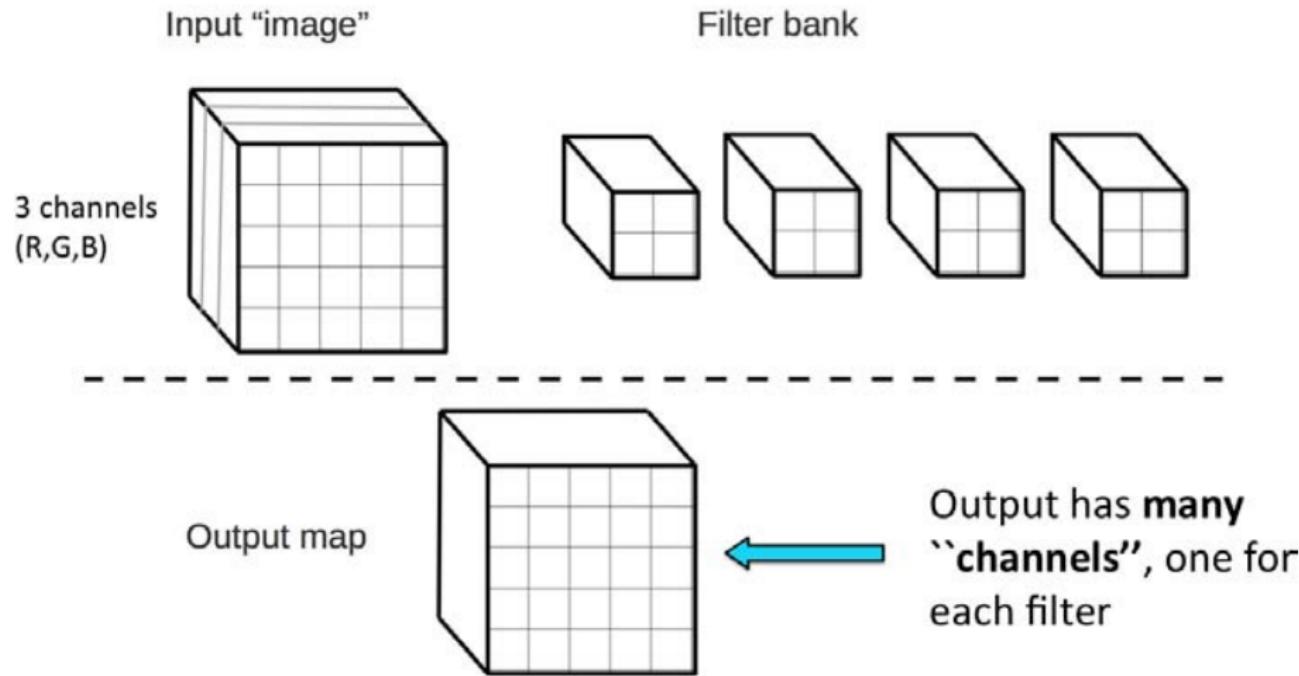


Output map



Convolutional Neural Networks (CNN)

- Now imagine we want to have many filters (e.g., vertical, horizontal, corners, one for dots). We will use a **filterbank**.

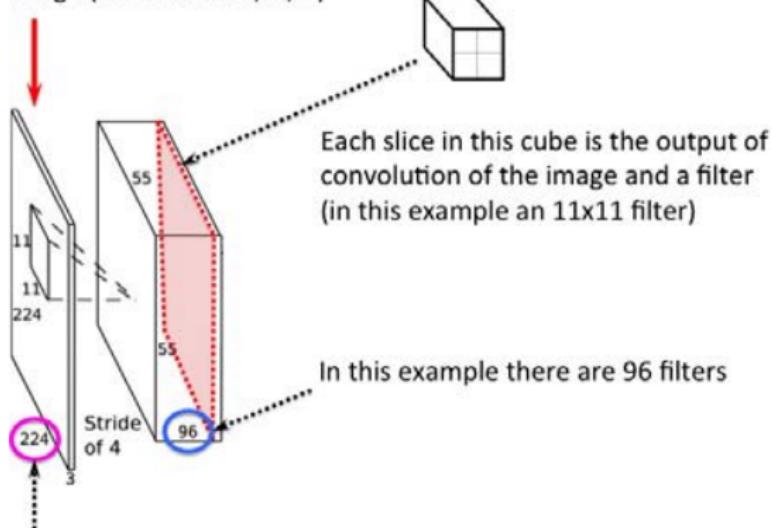


[Slide Credit: Sanja Fidler, Pic adopted from: A. Krizhevsky]

Convolutional Neural Networks (CNN)

- So applying a filterbank to an image yields a cube-like output, a 3D matrix in which each slice is an output of convolution with one filter.

image (3 channels: R, G, B)



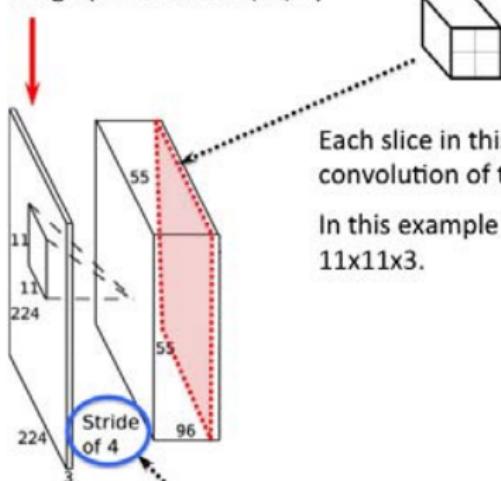
In this example our network will always expect a 224x224x3 image.

[Slide Credit: Sanja Fidler, Pic adopted from: A. Krizhevsky]

Convolutional Neural Networks (CNN)

- So applying a filterbank to an image yields a cube-like output, a 3D matrix in which each slice is an output of convolution with one filter.

image (3 channels: R, G, B)



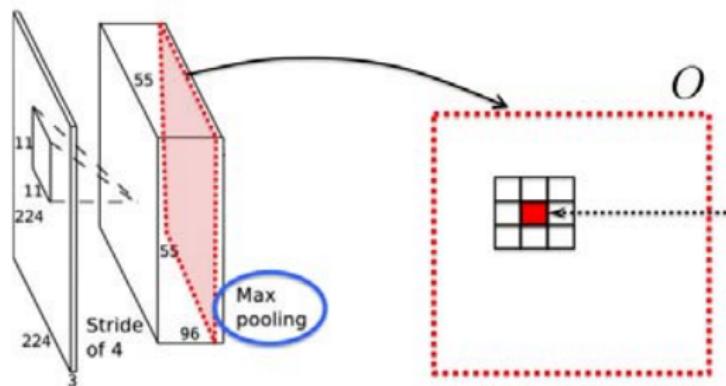
Each slice in this cube is the output of convolution of the image and a filter.

In this example the filter size is 11x11x3.

We don't do convolution in every pixel, but in every 4th pixel (in x and y direction)

Convolutional Neural Networks (CNN)

- Do some additional tricks. A popular one is called **max pooling**. Any idea why you would do this?

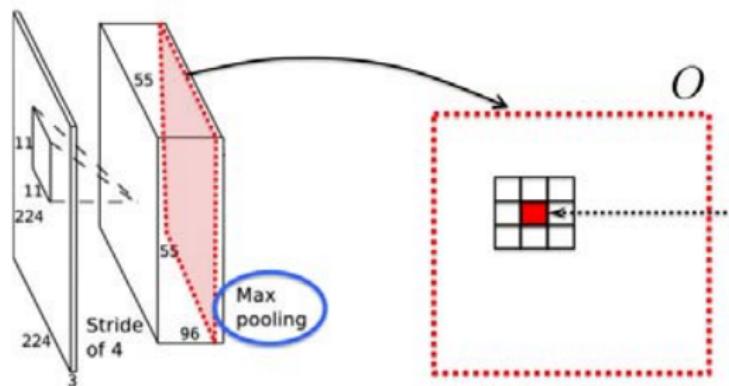


$$O(i, j) = \max_{\substack{k \in \{i-1, i, i+1\} \\ l \in \{j-1, j, j+1\}}} O(k, l)$$

Take each slice in the output cube, and in each pixel compute a max over a small patch around it. This is called **max pooling**.

Convolutional Neural Networks (CNN)

- Do some additional tricks. A popular one is called **max pooling**. Any idea why you would do this? To get **invariance to small shifts in position**.

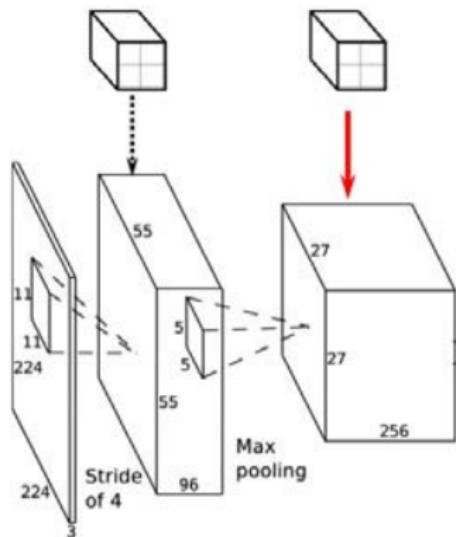


$$O(i, j) = \max_{\substack{k \in \{i-1, i, i+1\} \\ l \in \{j-1, j, j+1\}}} O(k, l)$$

Take each slice in the output cube, and in each pixel compute a max over a small patch around it. This is called **max pooling**.

Convolutional Neural Networks (CNN)

- Now add another “layer” of filters. For each filter again do convolution, but this time with the output cube of the previous layer.



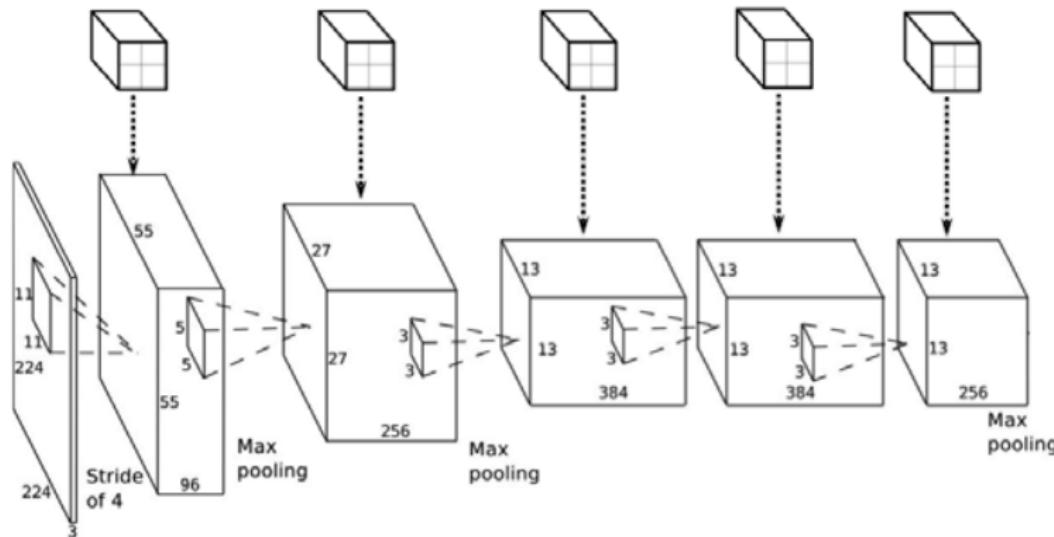
Add one more layer of filters

These filters are convolved with the output of the previous layer. The results of each convolution is again a slice in the cube on the right.

What is the dimension of each of these filters?

Convolutional Neural Networks (CNN)

- Keep adding a few layers. Any idea what's the purpose of more layers? Why can't we just have a full bunch of filters in one layer?



Do it recursively
Have multiple “layers”

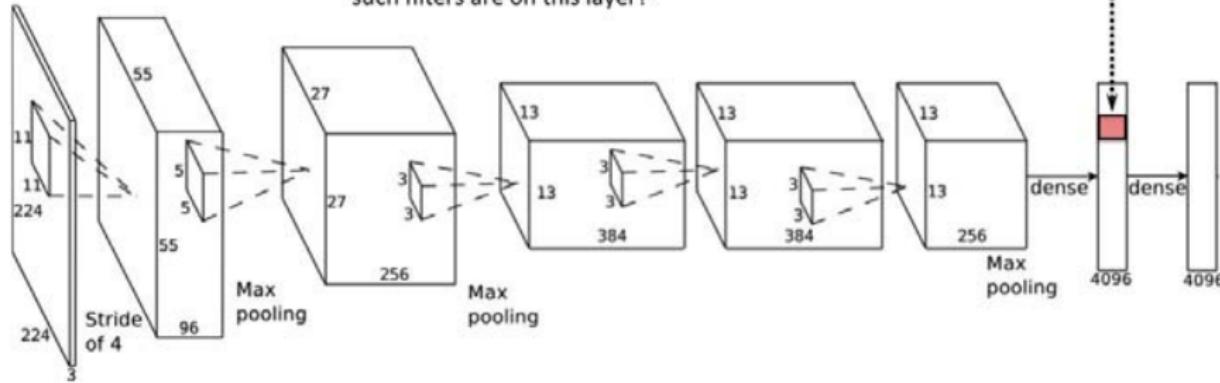
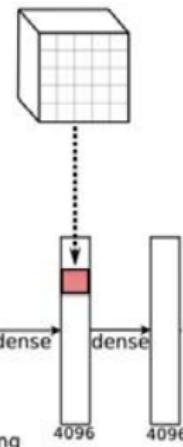
[Slide Credit: Sanja Fidler, Pic adopted from: A. Krizhevsky]

Convolutional Neural Networks (CNN)

- In the end add one or two **fully** (or **densely**) connected layers. In this layer, we don't do convolution we just do a dot-product between the "filter" and the output of the previous layer.

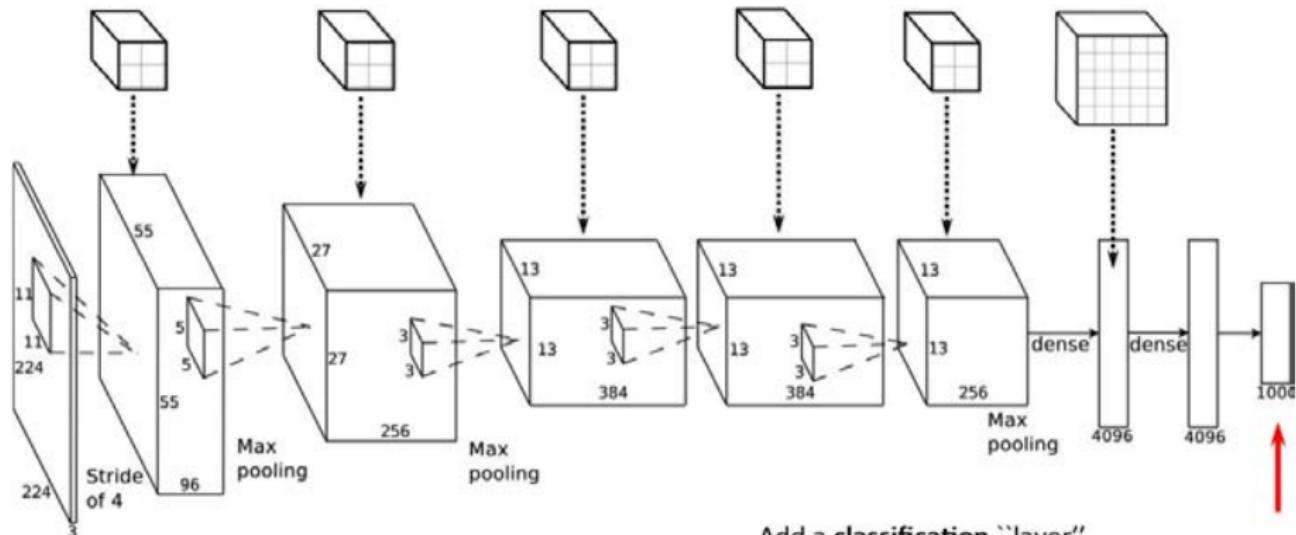
In the top, most networks add a "densely" connected layer. You can think of this as a filter, and the output value is a dot product between the filter and the output cube of the previous layer.

What are the dimensions of this filter in this example? How many such filters are on this layer?



Convolutional Neural Networks (CNN)

- Add one final layer: a **classification** layer. Each dimension of this vector tells us the probability of the input image being of a certain class.

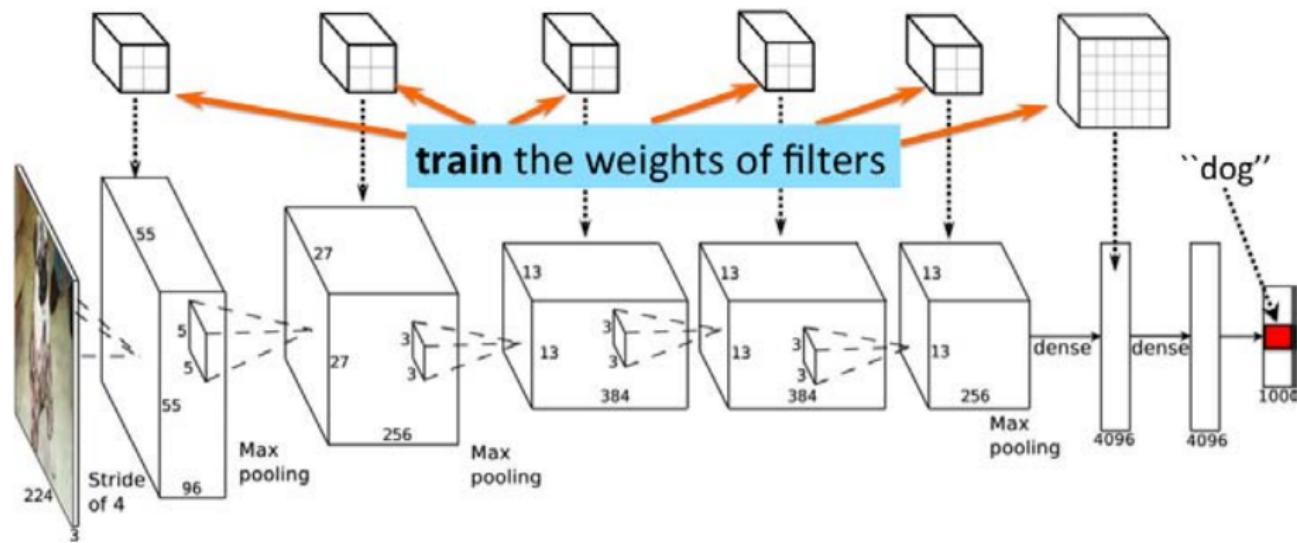


Add a **classification** "layer".

For an input image, the value in a particular dimension of this vector tells you the probability of the corresponding object class.

Convolutional Neural Networks (CNN)

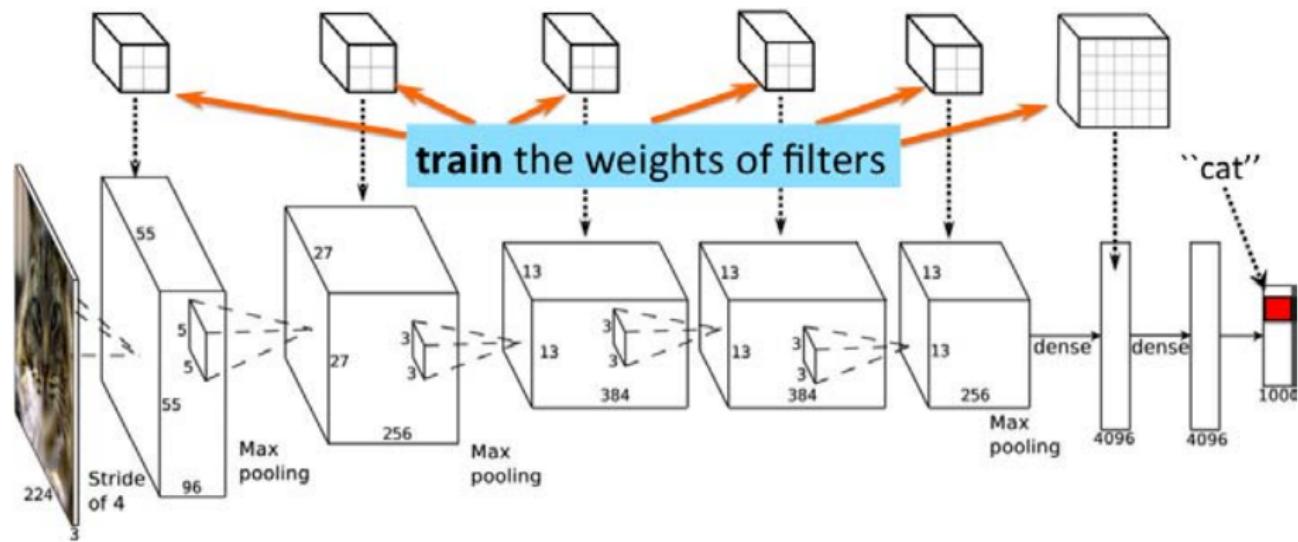
- The trick is to not hand-fix the weights, but to **train** them. Train them such that when the network sees a picture of a dog, the last layer will say “dog”.



[Slide Credit: Sanja Fidler, Pic adopted from: A. Krizhevsky]

Convolutional Neural Networks (CNN)

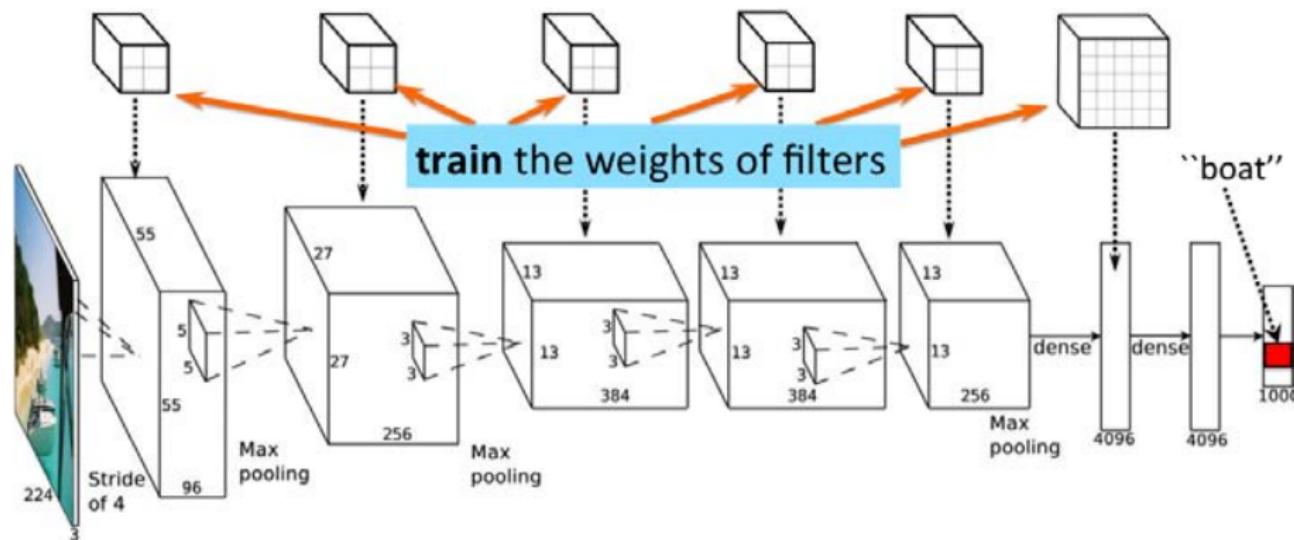
- Or when the network sees a picture of a cat, the last layer will say "cat".



[Slide Credit: Sanja Fidler, Pic adopted from: A. Krizhevsky]

Convolutional Neural Networks (CNN)

- Or when the network sees a picture of a boat, the last layer will say “boat”... The more pictures the network sees, the better.



Train on **lots** of examples. Millions. Tens of millions. Wait a week for training to finish.

Share your network (the weights) with others who are not fortunate enough with GPU power.

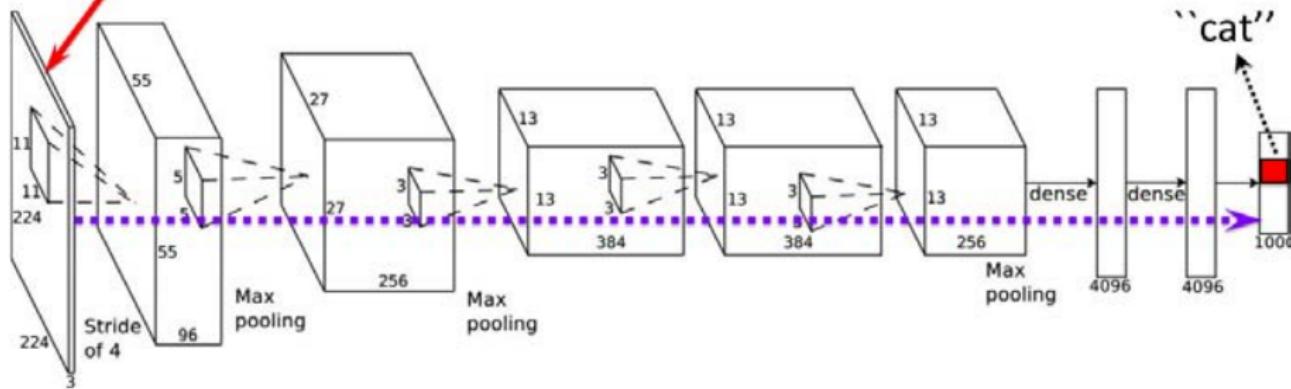
[Slide Credit: Sanja Fidler, Pic adopted from: A. Krizhevsky]

Classification

- Once trained we feed in an image or a crop, run through the network, and read out the class with the highest probability in the last (classif) layer.



What's the class of this object?



[Slide Credit: Sanja Fidler]

Classification Performance

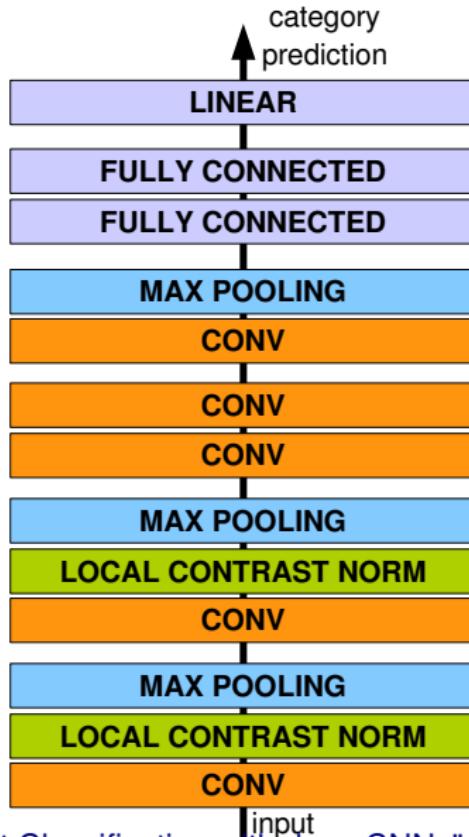
- Imagenet, main challenge for object classification: <http://image-net.org/>
- 1000 classes, 1.2M training images, 150K for test



poster created by Fengjun Lv using VIPBase

Image courtesy of ImageNet (<http://www.image-net.org/challenges/LSVRC/2010/index>)

Architecture for Classification

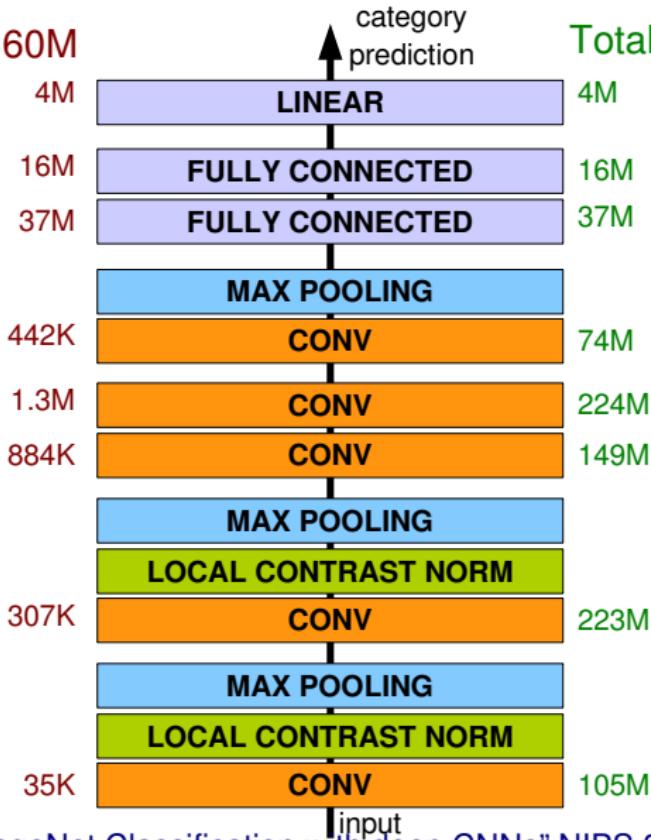


Krizhevsky et al. "ImageNet Classification with deep CNNs" NIPS 2012

Architecture for Classification

Total nr. params: 60M

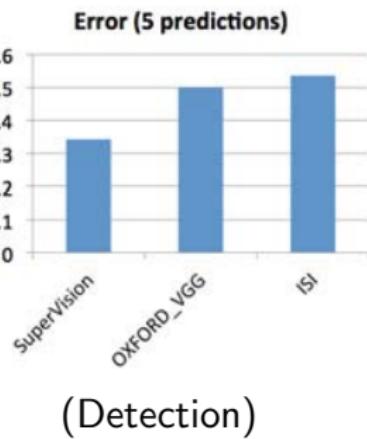
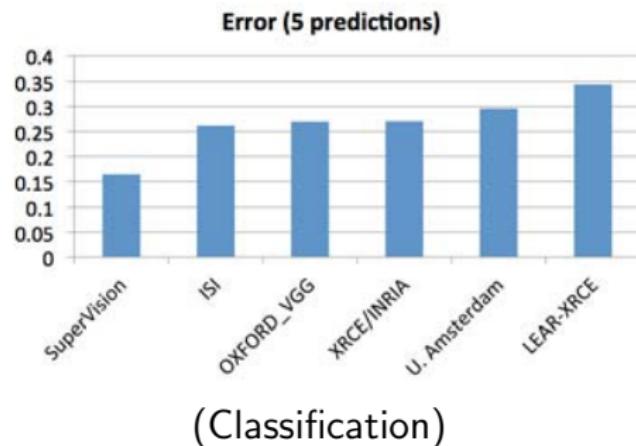
Total nr. flops: 832M



Krizhevsky et al. "ImageNet Classification with deep CNNs" NIPS 2012

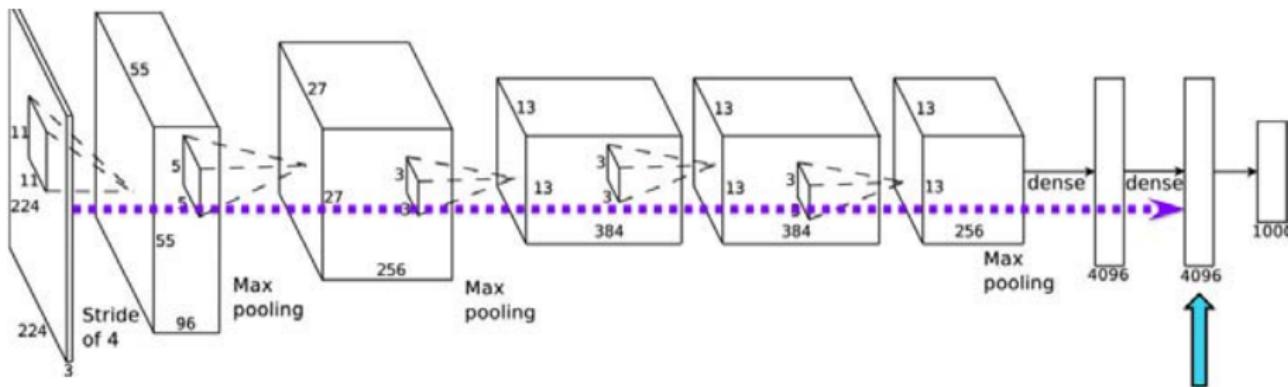
96

The 2012 Computer Vision Crisis



Neural Networks as Descriptors

- What vision people like to do is take the already trained network (avoid one week of training), and remove the last classification layer. Then take the top remaining layer (the 4096 dimensional vector here) and use it as a descriptor (feature vector).



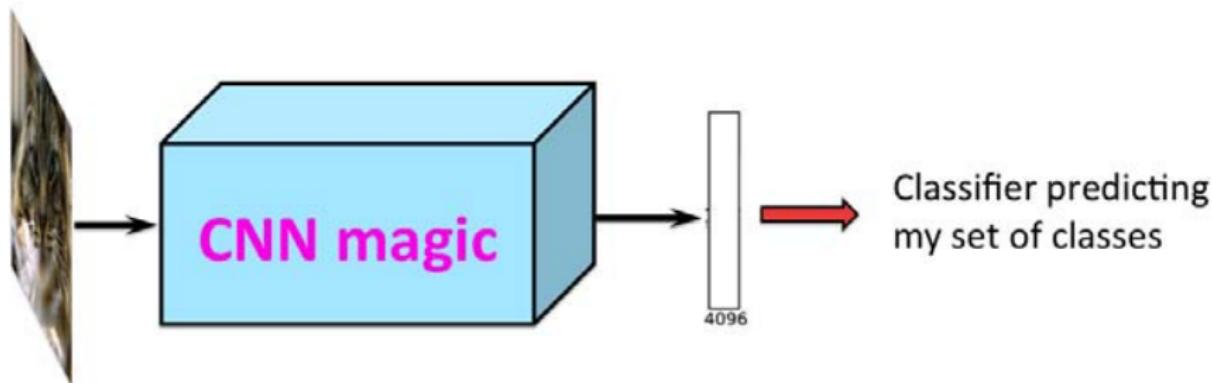
Vision people are mainly interested in this vector. You can use it as a descriptor. A much better descriptor than SIFT, etc.

Train your own classifier on top for your choice of classes.

[Slide Credit: Sanja Fidler]

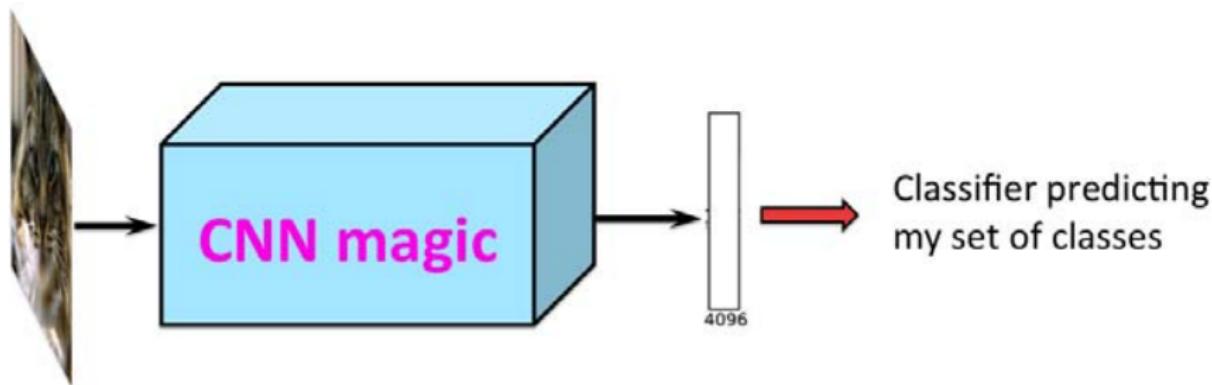
Neural Networks as Descriptors

- What vision people like to do is take the already trained network, and remove the last classification layer. Then take the top remaining layer (the 4096 dimensional vector here) and use it as a descriptor (feature vector).
- Now train your own classifier on top of these features for arbitrary classes.



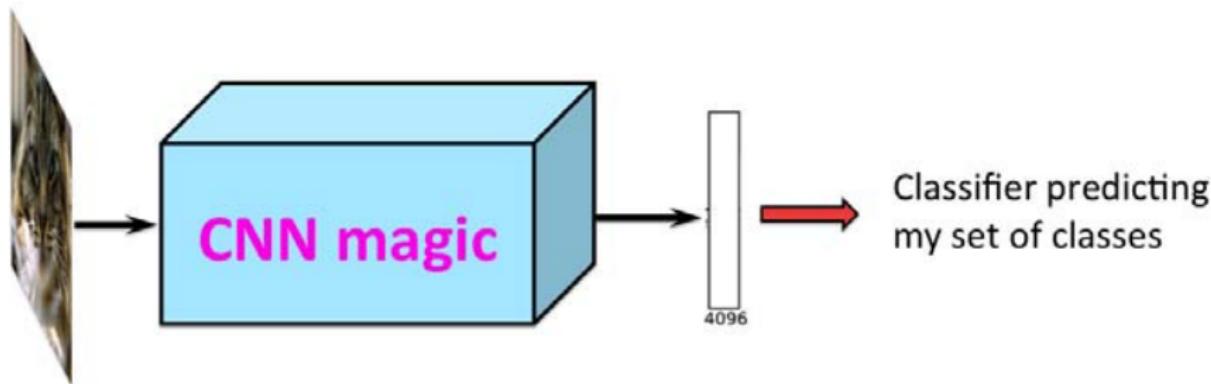
Neural Networks as Descriptors

- What vision people like to do is take the already trained network, and remove the last classification layer. Then take the top remaining layer (the 4096 dimensional vector here) and use it as a descriptor (feature vector).
- Now train your own classifier on top of these features for arbitrary classes.
- This is quite hacky, but works miraculously well.



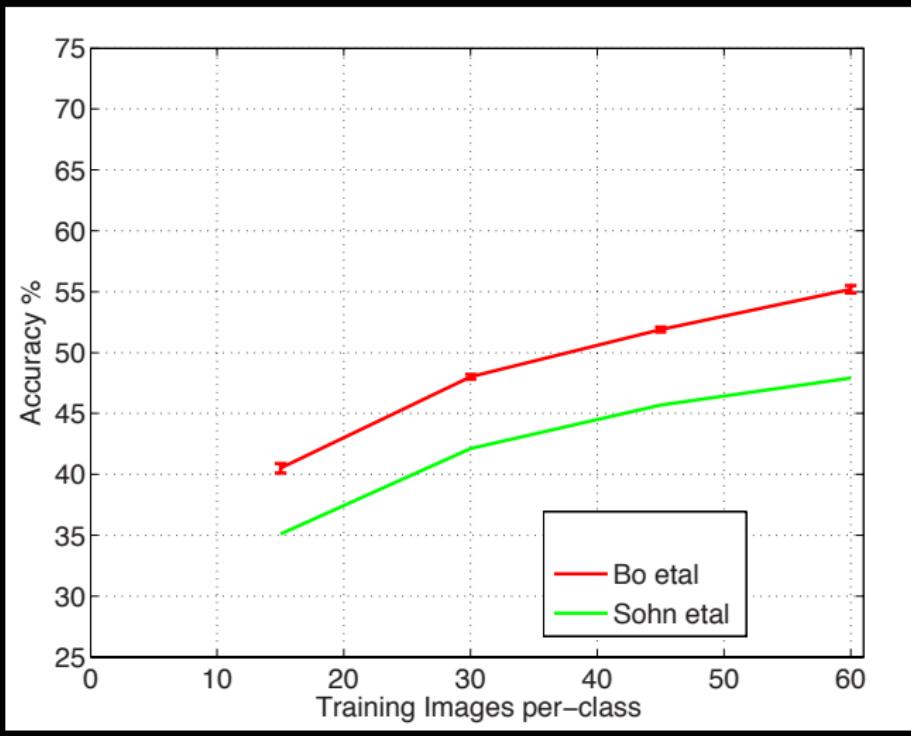
Neural Networks as Descriptors

- What vision people like to do is take the already trained network, and remove the last classification layer. Then take the top remaining layer (the 4096 dimensional vector here) and use it as a descriptor (feature vector).
- Now train your own classifier on top of these features for arbitrary classes.
- This is quite hacky, but works miraculously well.
- Everywhere where we were using SIFT (or anything else), you can use NNs.



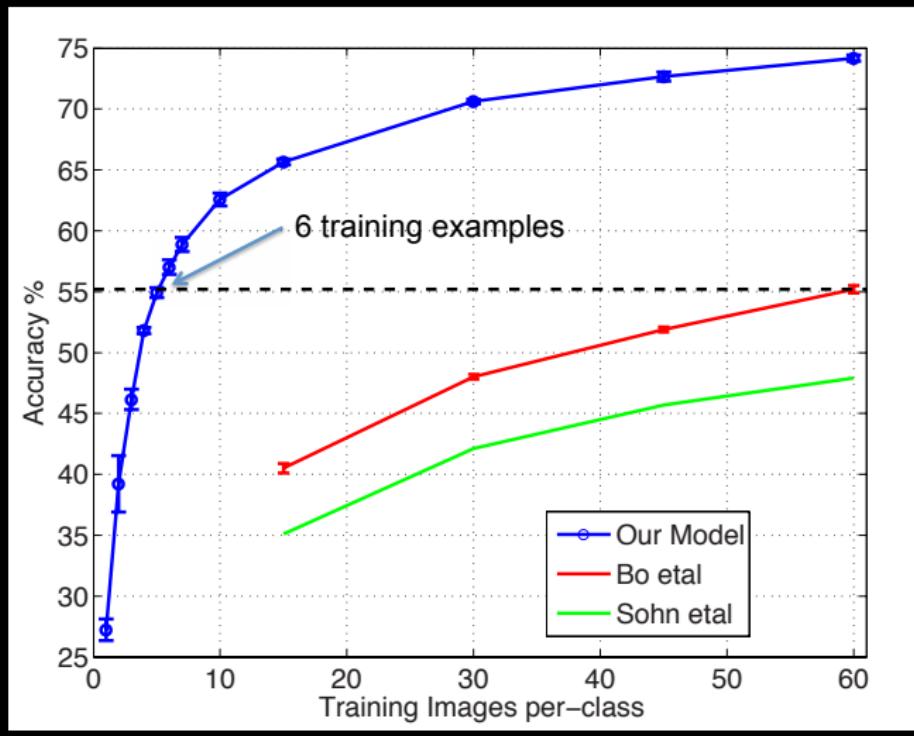
Caltech Results

Zeiler & Fergus, *Visualizing and Understanding Convolutional Networks*, arXiv 1311.2901, 2013



Caltech Results

Zeiler & Fergus, *Visualizing and Understanding Convolutional Networks*, arXiv 1311.2901, 2013



And Detection?

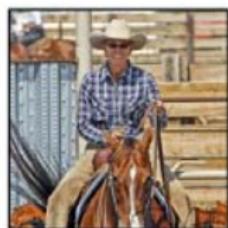
- For classification we feed in the full image to the network. But how can we perform detection?



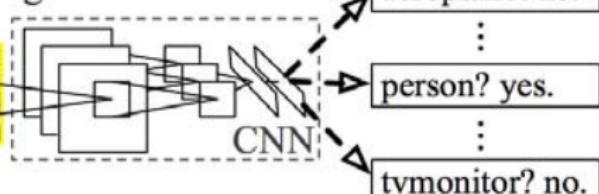
Find all objects of interest in this image!

The Era Post-Alex Net: PASCAL VOC detection

R-CNN: *Regions with CNN features*



warped region



1. Input image

2. Extract region proposals (~2k)

3. Compute CNN features

4. Classify regions

- Extract object proposals with bottom up grouping
- and then classify them using your big net

Detection Performance

- **PASCAL VOC challenge:** <http://pascallin.ecs.soton.ac.uk/challenges/VOC/>.



Figure : PASCAL has 20 object classes, 10K images for training, 10K for test

Detection Performance a Year Ago: 40.4%

A year ago, no networks:

- Results on the main recognition benchmark, the **PASCAL VOC challenge**.

	mean	aero plane	bicycle	bird	boat	bottle	bus	car	cat	chair	cow	dining table	dog	horse	motor bike	person	potted plant	sheep	sofa	train	tv/ monitor	submission date
segDPM [7]	40.4	61.4	53.4	25.6	25.2	35.5	51.7	50.6	50.8	19.3	33.8	26.8	40.4	48.3	54.4	47.1	14.8	38.7	35.0	52.8	43.1	24-Feb-2014
Boosted HOG-LBP and multi-context (LC, EGC, HLC) [7]	36.8	53.3	55.3	19.2	21.0	30.0	54.5	46.7	41.2	20.0	31.5	20.8	30.3	48.6	55.3	46.5	10.2	34.4	26.6	50.3	40.3	29-Aug-2010
MITUCLA_Hierarchy [7]	56.0	54.3	48.5	15.7	19.2	29.2	55.6	43.5	41.7	16.9	28.5	26.7	30.9	48.3	55.0	41.7	9.7	35.8	30.8	47.2	40.8	30-Aug-2010
HOGLBP_context_classification_rescore_v2 [7]	34.2	49.1	52.4	17.8	12.0	30.6	53.5	32.8	37.3	17.7	30.6	27.7	29.5	51.9	58.3	44.2	9.6	14.8	27.9	49.5	38.4	30-Aug-2010
LSVM-MDPM [7]	33.7	52.4	54.3	13.0	15.6	35.1	54.2	49.1	31.8	15.5	26.2	13.5	21.5	45.4	51.6	47.5	9.1	35.1	19.4	46.6	38.0	26-Aug-2010
UOCTTI_LSVMSMDPM [7]	33.4	49.2	53.8	13.1	15.3	35.5	53.4	49.7	27.0	17.2	28.8	14.7	17.8	46.4	51.2	47.7	10.8	34.2	20.7	43.8	38.3	21-May-2012
Detection Monkey [7]	32.9	56.7	39.8	16.8	12.2	13.8	44.9	36.9	47.7	12.1	26.9	26.5	37.2	42.1	51.9	25.7	12.1	37.8	33.0	41.5	41.7	30-Aug-2010
RM2C [7]	32.8	49.8	50.6	15.1	15.5	28.5	51.1	42.2	30.5	17.3	28.8	12.4	26.0	45.6	51.8	41.4	12.6	30.4	26.1	44.0	37.8	29-Oct-2013
UOCTTI_LSVMSMDPM [7]	32.2	48.2	52.2	14.8	13.8	28.7	53.2	44.9	26.0	18.4	24.4	13.7	23.1	45.8	50.5	43.7	9.8	31.1	21.5	44.4	35.7	11-May-2012
GroupLoc [7]	31.9	58.4	39.6	18.0	13.3	11.1	46.4	37.8	43.9	10.3	27.5	20.8	36.0	39.4	48.5	22.9	13.0	36.9	30.5	41.2	41.9	30-Aug-2010
UOCTTI_LSVMSMDPM [7]	29.6	45.6	49.0	11.0	11.6	27.2	50.5	41.3	23.6	17.2	23.2	10.7	20.5	42.5	44.5	41.3	8.7	29.0	18.7	40.0	34.5	21-May-2012
Bonn_FGT_Segm [7]	26.1	52.7	33.7	13.2	11.0	14.2	43.2	31.9	35.6	9.8	25.4	14.4	20.8	38.1	41.7	25.0	5.8	26.3	18.1	37.6	28.1	30-Aug-2010
HOG-LBP + DHOG bag of words, SVM [7]	23.5	40.4	34.7	2.7	8.4	26.0	43.1	33.8	17.2	11.2	14.3	14.5	14.9	31.8	37.3	30.0	6.4	25.2	11.6	30.0	35.7	30-Aug-2010
Svr-Segm [7]	23.4	50.5	24.5	17.1	13.3	10.9	39.5	32.9	36.5	5.6	16.0	6.6	22.3	24.9	29.0	29.8	6.7	28.4	13.3	32.1	27.2	30-Aug-2010
HOG-LBP Linear SVM [7]	22.1	37.9	33.7	2.7	6.5	25.3	37.5	33.1	15.5	10.9	12.3	12.5	13.7	29.7	34.5	33.8	7.2	22.9	9.9	28.9	34.1	29-Aug-2010
HOG+LBP+LTP+PLS2ROOTS [7]	17.5	32.7	29.7	0.8	1.1	19.9	39.4	27.5	8.6	4.5	8.1	6.3	11.0	22.9	34.1	24.6	3.1	24.0	2.0	23.5	27.0	31-Aug-2010
RandomParts [7]	14.2	23.8	31.7	1.2	3.4	11.1	29.7	19.5	14.2	0.8	11.1	7.0	4.7	16.4	31.5	16.0	1.1	15.6	10.2	14.7	21.0	25-Aug-2010
SIFT-GMM-MKL2 [7]	8.3	20.0	14.5	3.8	1.2	0.5	17.6	8.1	28.5	0.1	2.9	3.1	17.5	7.2	18.8	3.3	0.8	2.9	6.3	7.6	1.1	30-Aug-2010
UC3M_Generative_Discriminative [7]	6.3	15.8	5.5	5.6	2.3	0.3	10.2	5.4	12.6	0.5	5.6	4.5	7.7	11.3	12.6	5.3	1.5	2.0	5.9	9.1	3.2	30-Aug-2010
SIFT-GMM-MKL [7]	2.3	10.6	1.6	1.2	0.9	0.1	2.8	1.6	6.7	0.1	2.0	0.4	3.0	2.0	4.4	2.0	0.3	1.1	1.2	2.1	1.9	30-Aug-2010

Figure : Leading method segDPM (ours). Those were the good times...

S. Fidler, R. Mottaghi, A. Yuille, R. Urtasun, Bottom-up Segmentation for Top-down Detection, CVPR'13

The Era Post-Alex Net: PASCAL VOC detection

►	R-CNN [?]	50.2
►	BERKELEY POSELETS [?]	-
►	poselets [?]	-
►	** UCI_LSVM-MDPM-10X ** [?]	-
►	Head-Detect-Segment [?]	-

So Neural Networks are Great

- So networks turn out to be great.
- Everything is deep, even if it's shallow!
- Companies leading the competitions: ImageNet, KITTI, but not yet PASCAL
- At this point Google, Facebook, Microsoft, Baidu “steal” most neural network professors from academia.
- ... and a lot of our good students :(

So Neural Networks are Great

- But to train the networks you need quite a bit of computational power. So what do you do?



So Neural Networks are Great

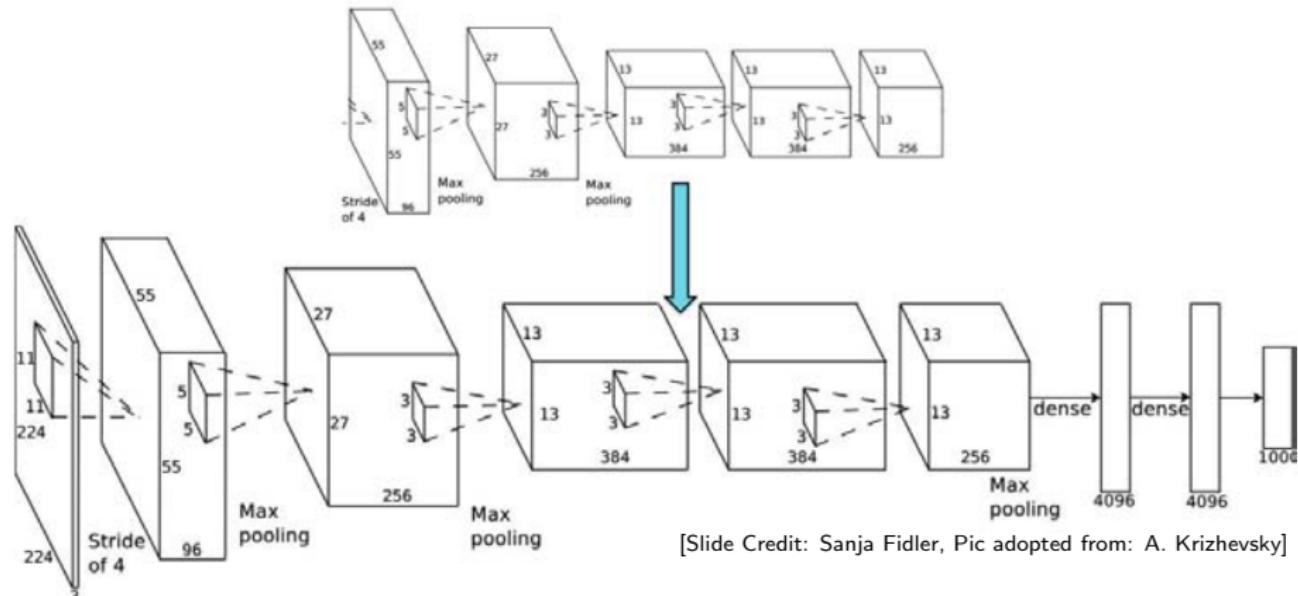
- Buy even more.



So Neural Networks are Great

- And train **more layers**. 16 instead of 7 before. 144 million parameters.

add more layers



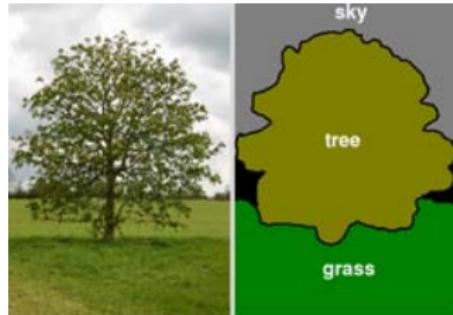
[Slide Credit: Sanja Fidler, Pic adopted from: A. Krizhevsky]

Figure : K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 2014

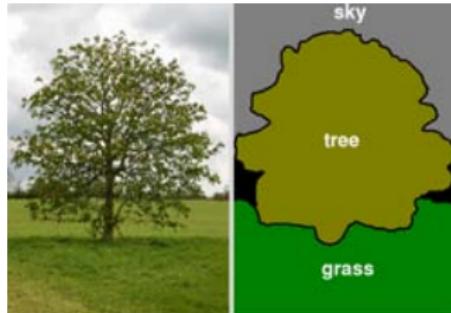
The Era Post-Alex Net: PASCAL VOC detection

▶	Fast R-CNN + YOLO [?]	70.8
▶	Fast R-CNN VGG16 extra data [?]	68.8
▶	segDeepM [?]	67.2
▶	BabyLearning [?]	63.8
▶	R-CNN (bbox reg) [?]	62.9
▶	R-CNN [?]	59.8
▶	Feature Edit [?]	56.4
▶	YOLO [?]	55.3
▶	R-CNN (bbox reg) [?]	53.7
▶	R-CNN [?]	50.2
▶	poselets [?]	-
▶	Head-Detect-Segment [?]	-
▶	BERKELEY POSELETS [?]	-
▶	** UCI_LSVM-MDPM-10X ** [?]	-

What if we Want Semantic Segmentation?

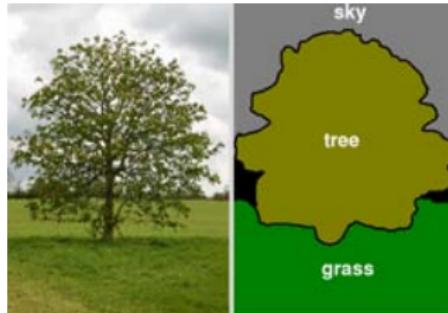


What if we Want Semantic Segmentation?



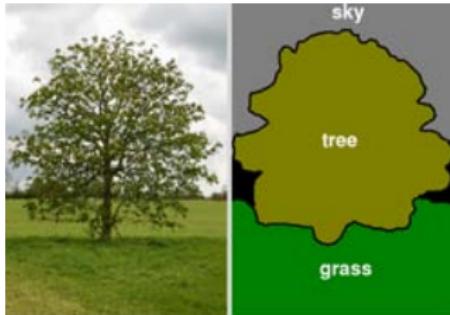
- Every layer, even fully connected can be treated as a convolutional layer, and then we can deal with arbitrary dimensions of the input

What if we Want Semantic Segmentation?



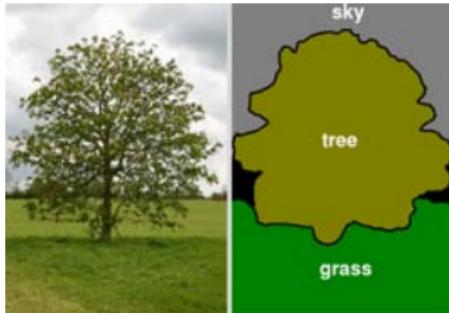
- Every layer, even fully connected can be treated as a convolutional layer, and then we can deal with arbitrary dimensions of the input
- The network can work on super pixels, or can directly operate in pixels

What if we Want Semantic Segmentation?



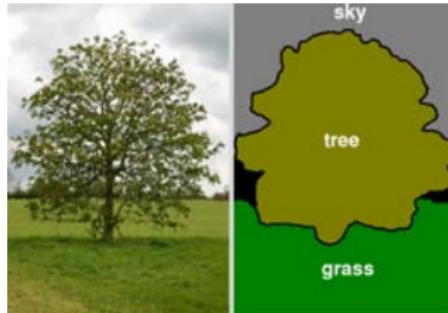
- Every layer, even fully connected can be treated as a convolutional layer, and then we can deal with arbitrary dimensions of the input
- The network can work on super pixels, or can directly operate in pixels
- Due to pooling, the output is typically lower dimensional than the input, use interpolation.

What if we Want Semantic Segmentation?



- Every layer, even fully connected can be treated as a convolutional layer, and then we can deal with arbitrary dimensions of the input
- The network can work on super pixels, or can directly operate in pixels
- Due to pooling, the output is typically lower dimensional than the input, use interpolation.
- PASCAL VOC, 65% IOU

What if we Want Semantic Segmentation?



- Every layer, even fully connected can be treated as a convolutional layer, and then we can deal with arbitrary dimensions of the input
- The network can work on super pixels, or can directly operate in pixels
- Due to pooling, the output is typically lower dimensional than the input, use interpolation.
- PASCAL VOC, 65% IOU
- More to come in Part II

Practical Tips

How to choose Hyperparameters?

- **Hyperparameters:** architecture, learning rate, num layers, num features, etc

How to choose Hyperparameters?

- **Hyperparameters:** architecture, learning rate, num layers, num features, etc
- How to choose them?

How to choose Hyperparameters?

- **Hyperparameters:** architecture, learning rate, num layers, num features, etc
- How to choose them?
 - ① Cross-validation

How to choose Hyperparameters?

- **Hyperparameters:** architecture, learning rate, num layers, num features, etc
- How to choose them?
 - ① Cross-validation
 - ② Grid search (need lots of GPUs)

How to choose Hyperparameters?

- **Hyperparameters:** architecture, learning rate, num layers, num features, etc
- How to choose them?
 - ① Cross-validation
 - ② Grid search (need lots of GPUs)
 - ③ Random [Bergstra & Bengio JMLR 2012]

How to choose Hyperparameters?

- **Hyperparameters:** architecture, learning rate, num layers, num features, etc
- How to choose them?
 - ① Cross-validation
 - ② Grid search (need lots of GPUs)
 - ③ Random [Bergstra & Bengio JMLR 2012]
 - ④ Bayesian optimization [Whetlab Toronto]

Good to Know

- ALWAYS check gradients numerically by finite differences!

Good to Know

- ALWAYS check gradients numerically by finite differences!
- Measure error on both training and validation set, NEVER TEST

Good to Know

- ALWAYS check gradients numerically by finite differences!
- Measure error on both training and validation set, NEVER TEST
- Test on a small subset of the data and check that you can over fit (i.e., error $\rightarrow 0$)

Good to Know

- ALWAYS check gradients numerically by finite differences!
- Measure error on both training and validation set, NEVER TEST
- Test on a small subset of the data and check that you can over fit (i.e., error $\rightarrow 0$)
- Visualize features (feature maps need to be uncorrelated) and have high variance.



(good)



(bad)

Figure : from M. Ranzato

Good to Know

- ALWAYS check gradients numerically by finite differences!
- Measure error on both training and validation set, NEVER TEST
- Test on a small subset of the data and check that you can over fit (i.e., error $\rightarrow 0$)
- Visualize features (feature maps need to be uncorrelated) and have high variance.
- Visualize parameters

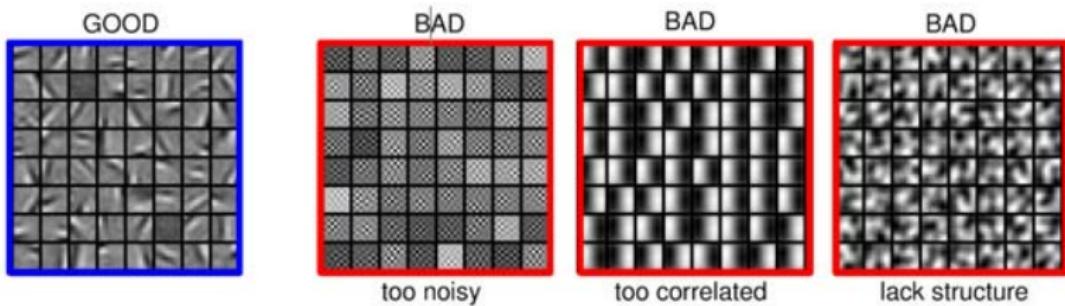


Figure : from M. Ranzato

What if it doesn't work?

- Training diverges

What if it doesn't work?

- Training diverges
 - Decrease learning rate

What if it doesn't work?

- Training diverges
 - Decrease learning rate
 - Check gradients

What if it doesn't work?

- Training diverges
 - Decrease learning rate
 - Check gradients
- Parameters collapse / loss is minimized but accuracy is low

What if it doesn't work?

- Training diverges
 - Decrease learning rate
 - Check gradients
- Parameters collapse / loss is minimized but accuracy is low
 - Appropriate loss function?

What if it doesn't work?

- Training diverges
 - Decrease learning rate
 - Check gradients
- Parameters collapse / loss is minimized but accuracy is low
 - Appropriate loss function?
 - Does loss-function have degenerate solutions?

What if it doesn't work?

- Training diverges
 - Decrease learning rate
 - Check gradients
- Parameters collapse / loss is minimized but accuracy is low
 - Appropriate loss function?
 - Does loss-function have degenerate solutions?
- Network is underperforming

What if it doesn't work?

- Training diverges
 - Decrease learning rate
 - Check gradients
- Parameters collapse / loss is minimized but accuracy is low
 - Appropriate loss function?
 - Does loss-function have degenerate solutions?
- Network is underperforming
 - Make it bigger

What if it doesn't work?

- Training diverges
 - Decrease learning rate
 - Check gradients
- Parameters collapse / loss is minimized but accuracy is low
 - Appropriate loss function?
 - Does loss-function have degenerate solutions?
- Network is underperforming
 - Make it bigger
 - Visualize hidden units/params and fix optimization

What if it doesn't work?

- Training diverges
 - Decrease learning rate
 - Check gradients
- Parameters collapse / loss is minimized but accuracy is low
 - Appropriate loss function?
 - Does loss-function have degenerate solutions?
- Network is underperforming
 - Make it bigger
 - Visualize hidden units/params and fix optimization
- Network is too slow

What if it doesn't work?

- Training diverges
 - Decrease learning rate
 - Check gradients
- Parameters collapse / loss is minimized but accuracy is low
 - Appropriate loss function?
 - Does loss-function have degenerate solutions?
- Network is underperforming
 - Make it bigger
 - Visualize hidden units/params and fix optimization
- Network is too slow
 - GPU,distrib. framework, make net smaller

[Slide credit: M. Ranzato]

Improving Generalization

- Weight sharing (Reduce the number of parameters)

Improving Generalization

- Weight sharing (Reduce the number of parameters)
- Data augmentation (e.g., jittering, noise injection, transformations)

Improving Generalization

- Weight sharing (Reduce the number of parameters)
- Data augmentation (e.g., jittering, noise injection, transformations)
- Dropout [Hinton et al.]: randomly drop units (along with their connections) from the neural network during training. Use for the fully connected layers only

Improving Generalization

- Weight sharing (Reduce the number of parameters)
- Data augmentation (e.g., jittering, noise injection, transformations)
- Dropout [Hinton et al.]: randomly drop units (along with their connections) from the neural network during training. Use for the fully connected layers only
- Regularization: Weight decay (L2, L1)

Improving Generalization

- Weight sharing (Reduce the number of parameters)
- Data augmentation (e.g., jittering, noise injection, transformations)
- Dropout [Hinton et al.]: randomly drop units (along with their connections) from the neural network during training. Use for the fully connected layers only
- Regularization: Weight decay (L2, L1)
- Sparsity in the hidden units

Improving Generalization

- Weight sharing (Reduce the number of parameters)
- Data augmentation (e.g., jittering, noise injection, transformations)
- Dropout [Hinton et al.]: randomly drop units (along with their connections) from the neural network during training. Use for the fully connected layers only
- Regularization: Weight decay (L2, L1)
- Sparsity in the hidden units
- Multi-task learning

[Slide credit: M. Ranzato]

Software

- Torch7: learning library that supports neural net training
<http://www.torch.ch>
<http://code.cogbits.com/wiki/doku.php> (tutorial with demos by C. Farabet)
<https://github.com/sermanet/OverFeat>
- Python-based learning library (U. Montreal)
<http://deeplearning.net/software/theano/> (does automatic differentiation)
- Efficient CUDA kernels for ConvNets (Krizhevsky)
code.google.com/p/cuda-convnet
- Caffe (Yangqing Jia)
<http://caffe.berkeleyvision.org>
- Deep Structured Models
<http://www.alexander-schwing.de/> (soon available)

[Slide Credit: M. Ranzato]

Part II: Deep Structured Learning

Your current Status?



What's next?



What's next?

- ① Theoretical Understanding
- ② Unsupervised Learning
- ③ Structured models

Structure!

- Many Vision Problems are complex and involve predicting many random variables that are statistically related

Scene understanding



\mathbf{x} = image

\mathbf{y} : room layout

Tag prediction



\mathbf{x} = image

\mathbf{y} : tag "combo"

Segmentation



\mathbf{x} = image

\mathbf{y} : segmentation

Deep Learning

- Complex mapping $F(\mathbf{x}, \mathbf{y}, \mathbf{w})$ to predict output y given input \mathbf{x} through a series of matrix multiplications, non-linearities and pooling operations

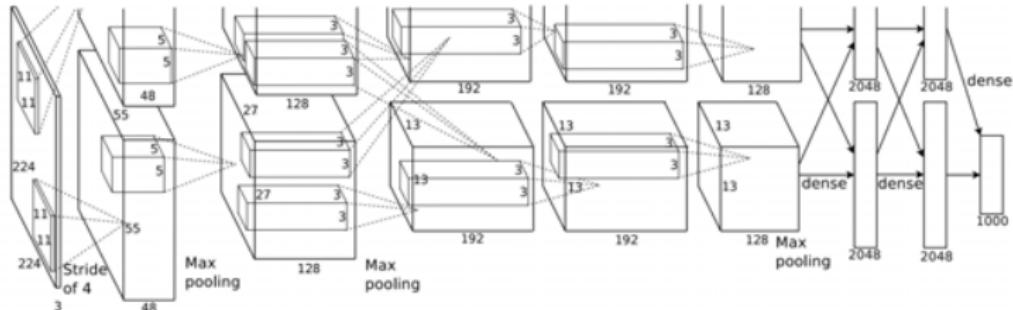


Figure : Imagenet CNN [Krizhevsky et al. 13]

Deep Learning

- Complex mapping $F(\mathbf{x}, \mathbf{y}, \mathbf{w})$ to predict output y given input \mathbf{x} through a series of matrix multiplications, non-linearities and pooling operations

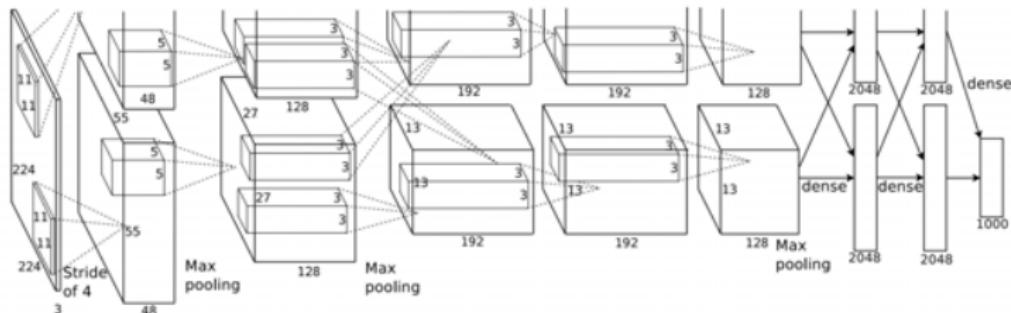


Figure : Imagenet CNN [Krizhevsky et al. 13]

- We typically train the network to predict one random variable (e.g., ImageNet) by minimizing cross-entropy

Deep Learning

- Complex mapping $F(\mathbf{x}, \mathbf{y}, \mathbf{w})$ to predict output y given input \mathbf{x} through a series of matrix multiplications, non-linearities and pooling operations

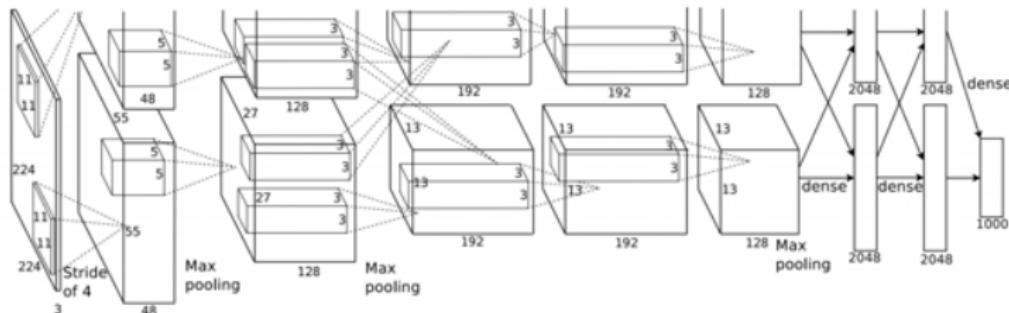


Figure : Imagenet CNN [Krizhevsky et al. 13]

- We typically train the network to predict one random variable (e.g., ImageNet) by minimizing cross-entropy
- Multi-task extensions: sum the loss of each task, and share part of the features (e.g., segmentation)

Deep Learning

- Complex mapping $F(\mathbf{x}, \mathbf{y}, \mathbf{w})$ to predict output y given input \mathbf{x} through a series of matrix multiplications, non-linearities and pooling operations

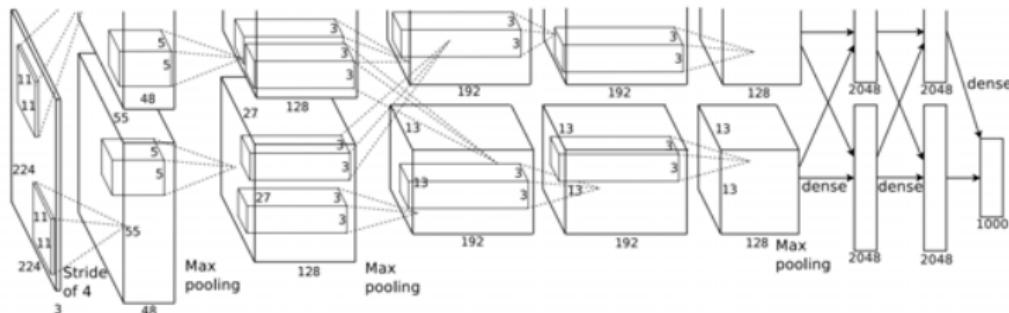


Figure : Imagenet CNN [Krizhevsky et al. 13]

- We typically train the network to predict one random variable (e.g., ImageNet) by minimizing cross-entropy
- Multi-task extensions: sum the loss of each task, and share part of the features (e.g., segmentation)
- Use an MRF as a post processing step

PROBLEM: *How can we take into account complex dependencies when predicting multiple variables?*

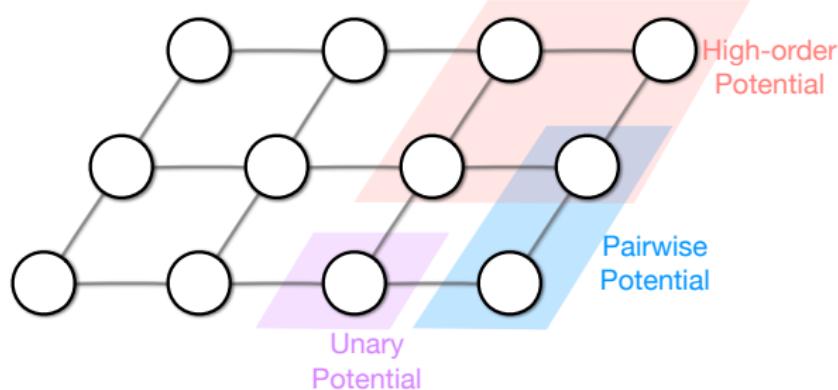
PROBLEM: How can we take into account complex dependencies when predicting multiple variables?

SOLUTION: Graphical models

Graphical Models

- Convenient tool to illustrate dependencies among random variables

$$E(\mathbf{y}) = - \underbrace{\sum_i f_i(y_i)}_{\text{unaries}} - \underbrace{\sum_{i,j \in \mathcal{E}} f(y_i, y_j)}_{\text{pairwise}} - \underbrace{\sum_{\alpha} f_{\alpha}(\mathbf{y}_{\alpha})}_{\text{high-order}}$$



- Widespread usage among different fields: vision, NLP, comp. bio, ...

Compact Notation

- In Computer Vision we usually express

$$E(\mathbf{y}) = - \underbrace{\sum_i f_i(y_i)}_{unaries} - \underbrace{\sum_{i,j \in \mathcal{E}} f(y_i, y_j)}_{pairwise} - \underbrace{\sum_{\alpha} f_{\alpha}(\mathbf{y}_{\alpha})}_{high-order}$$

Compact Notation

- In Computer Vision we usually express

$$E(\mathbf{y}) = - \underbrace{\sum_i f_i(y_i)}_{unaries} - \underbrace{\sum_{i,j \in \mathcal{E}} f(y_i, y_j)}_{pairwise} - \underbrace{\sum_{\alpha} f_{\alpha}(\mathbf{y}_{\alpha})}_{high-order}$$

- For the purpose of this talk we are going to use a more compact notation

$$E(\mathbf{y}, \mathbf{w}) = - \sum_{r \in \mathcal{R}} f_r(\mathbf{y}_r, \mathbf{w})$$

Compact Notation

- In Computer Vision we usually express

$$E(\mathbf{y}) = - \underbrace{\sum_i f_i(y_i)}_{unaries} - \underbrace{\sum_{i,j \in \mathcal{E}} f(y_i, y_j)}_{pairwise} - \underbrace{\sum_{\alpha} f_{\alpha}(\mathbf{y}_{\alpha})}_{high-order}$$

- For the purpose of this talk we are going to use a more compact notation

$$E(\mathbf{y}, \mathbf{w}) = - \sum_{r \in \mathcal{R}} f_r(\mathbf{y}_r, \mathbf{w})$$

- r is a region and \mathcal{R} is the set of all regions

Compact Notation

- In Computer Vision we usually express

$$E(\mathbf{y}) = - \underbrace{\sum_i f_i(y_i)}_{unaries} - \underbrace{\sum_{i,j \in \mathcal{E}} f(y_i, y_j)}_{pairwise} - \underbrace{\sum_{\alpha} f_{\alpha}(\mathbf{y}_{\alpha})}_{high-order}$$

- For the purpose of this talk we are going to use a more compact notation

$$E(\mathbf{y}, \mathbf{w}) = - \sum_{r \in \mathcal{R}} f_r(\mathbf{y}_r, \mathbf{w})$$

- r is a region and \mathcal{R} is the set of all regions
- \mathbf{y}_r is of any order

Compact Notation

- In Computer Vision we usually express

$$E(\mathbf{y}) = - \underbrace{\sum_i f_i(y_i)}_{unaries} - \underbrace{\sum_{i,j \in \mathcal{E}} f(y_i, y_j)}_{pairwise} - \underbrace{\sum_{\alpha} f_{\alpha}(\mathbf{y}_{\alpha})}_{high-order}$$

- For the purpose of this talk we are going to use a more compact notation

$$E(\mathbf{y}, \mathbf{w}) = - \sum_{r \in \mathcal{R}} f_r(\mathbf{y}_r, \mathbf{w})$$

- r is a region and \mathcal{R} is the set of all regions
- \mathbf{y}_r is of any order
- The functions f_r are a function of parameters \mathbf{w}

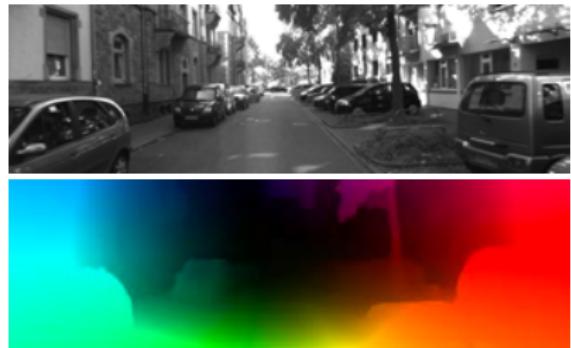
Continuous vs Discrete MRFs

$$E(\mathbf{y}, \mathbf{w}) = - \sum_{r \in \mathcal{R}} f_r(\mathbf{y}_r, \mathbf{w})$$

- Discrete MRFs:
 $y_i \in \{1, \dots, C_i\}$



- Continuous MRFs:
 $y_i \in \mathcal{Y} \subseteq \mathbb{R}$



- Hybrid MRFs with continuous and discrete variables

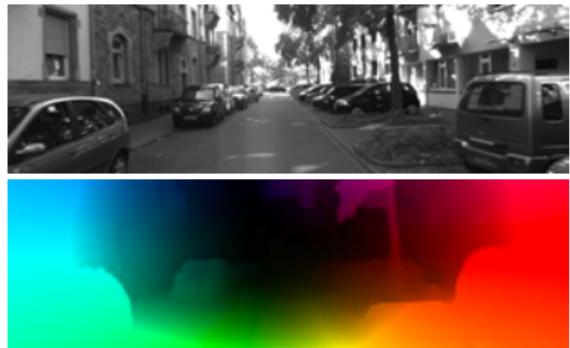
Continuous vs Discrete MRFs

$$E(\mathbf{y}, \mathbf{w}) = - \sum_{r \in \mathcal{R}} f_r(\mathbf{y}_r, \mathbf{w})$$

- Discrete MRFs:
 $y_i \in \{1, \dots, C_i\}$



- Continuous MRFs:
 $y_i \in \mathcal{Y} \subseteq \mathbb{R}$



- Hybrid MRFs with continuous and discrete variables
- Today's talk: only discrete MRFs

Probabilistic Interpretation

- The energy is defined as

$$E(\mathbf{y}, \mathbf{w}) = -F(\mathbf{y}, \mathbf{w}) = -\sum_{r \in \mathcal{R}} f_r(\mathbf{y}_r, \mathbf{w})$$

Probabilistic Interpretation

- The energy is defined as

$$E(\mathbf{y}, \mathbf{w}) = -F(\mathbf{y}, \mathbf{w}) = -\sum_{r \in \mathcal{R}} f_r(\mathbf{y}_r, \mathbf{w})$$

- We can construct a probability distribution over the outputs

$$p(\mathbf{y}; \mathbf{w}) = \frac{1}{Z} \exp \left(\sum_{r \in \mathcal{R}} f_r(\mathbf{y}_r, \mathbf{w}) \right)$$

with $Z(\mathbf{w}) = \sum_{\mathbf{y}} \exp \left(\sum_{r \in \mathcal{R}} f_r(\mathbf{y}_r, \mathbf{w}) \right)$ the partition function

Probabilistic Interpretation

- The energy is defined as

$$E(\mathbf{y}, \mathbf{w}) = -F(\mathbf{y}, \mathbf{w}) = -\sum_{r \in \mathcal{R}} f_r(\mathbf{y}_r, \mathbf{w})$$

- We can construct a probability distribution over the outputs

$$p(\mathbf{y}; \mathbf{w}) = \frac{1}{Z} \exp \left(\sum_{r \in \mathcal{R}} f_r(\mathbf{y}_r, \mathbf{w}) \right)$$

with $Z(\mathbf{w}) = \sum_{\mathbf{y}} \exp \left(\sum_{r \in \mathcal{R}} f_r(\mathbf{y}_r, \mathbf{w}) \right)$ the partition function

- CRFs vs MRFs

$$p(\mathbf{y}|\mathbf{x}; \mathbf{w}) = \frac{1}{Z(\mathbf{x})} \exp \left(\sum_{r \in \mathcal{R}} f_r(\mathbf{x}, \mathbf{y}_r, \mathbf{w}) \right)$$

with $Z(\mathbf{x}, \mathbf{w}) = \sum_{\mathbf{y}} \exp \left(\sum_{r \in \mathcal{R}} f_r(\mathbf{x}, \mathbf{y}_r, \mathbf{w}) \right)$ the partition function

Inference Tasks

- MAP: maximum a posteriori estimate, or minimum energy configuration

$$\mathbf{y}^* = \arg \max_{\mathbf{y}} \sum_{r \in \mathcal{R}} f_r(\mathbf{y}_r, \mathbf{w})$$

Inference Tasks

- MAP: maximum a posteriori estimate, or minimum energy configuration

$$\mathbf{y}^* = \arg \max_{\mathbf{y}} \sum_{r \in \mathcal{R}} f_r(\mathbf{y}_r, \mathbf{w})$$

- Probabilistic Inference: We might want to compute $p(\mathbf{y}_r)$ for any possible subset of variables r , or $p(\mathbf{y}_r | \mathbf{y}_p)$ for any subset r and p

Inference Tasks

- MAP: maximum a posteriori estimate, or minimum energy configuration

$$\mathbf{y}^* = \arg \max_{\mathbf{y}} \sum_{r \in \mathcal{R}} f_r(\mathbf{y}_r, \mathbf{w})$$

- Probabilistic Inference: We might want to compute $p(\mathbf{y}_r)$ for any possible subset of variables r , or $p(\mathbf{y}_r | \mathbf{y}_p)$ for any subset r and p
- M-best configurations (e.g., top-k)

Inference Tasks

- MAP: maximum a posteriori estimate, or minimum energy configuration

$$\mathbf{y}^* = \arg \max_{\mathbf{y}} \sum_{r \in \mathcal{R}} f_r(\mathbf{y}_r, \mathbf{w})$$

- Probabilistic Inference: We might want to compute $p(\mathbf{y}_r)$ for any possible subset of variables r , or $p(\mathbf{y}_r | \mathbf{y}_p)$ for any subset r and p
- M-best configurations (e.g., top-k)

Very difficult tasks in general (i.e., NP-hard). Some exceptions, e.g., low-tree width models and binary MRFs with sub-modular energies

Learning in CRFs

- Given a training set of N pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$, we want to estimate the functions $f_r(\mathbf{x}, \mathbf{y}_r, \mathbf{w})$
- As these functions are parametric, this is equivalent to estimating \mathbf{w}

Learning in CRFs

- Given a training set of N pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$, we want to estimate the functions $f_r(\mathbf{x}, \mathbf{y}_r, \mathbf{w})$
- As these functions are parametric, this is equivalent to estimating \mathbf{w}
- We would like to do this by minimizing the empirical loss

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \ell_{task}(\mathbf{x}, \mathbf{y}, \mathbf{w})$$

where ℓ_{task} is the loss that we'll be evaluated on

Learning in CRFs

- Given a training set of N pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$, we want to estimate the functions $f_r(\mathbf{x}, \mathbf{y}_r, \mathbf{w})$
- As these functions are parametric, this is equivalent to estimating \mathbf{w}
- We would like to do this by minimizing the empirical loss

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \ell_{task}(\mathbf{x}, \mathbf{y}, \mathbf{w})$$

where ℓ_{task} is the loss that we'll be evaluated on

- Very difficult, instead we minimize the sum of a surrogate (typically convex) loss and a regularizer

$$\min_{\mathbf{w}} R(\mathbf{w}) + \frac{C}{N} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \bar{\ell}(\mathbf{x}, \mathbf{y}, \mathbf{w})$$

More on Learning in CRFs

- Given a training set of N pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$, we want to estimate the functions $f_r(\mathbf{y}, \mathbf{x}, \mathbf{w})$
- Minimize a surrogate (typically convex) loss and a regularizer

$$\min_{\mathbf{w}} R(\mathbf{w}) + \frac{C}{N} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \bar{\ell}(\mathbf{x}, \mathbf{y}, \mathbf{w})$$

More on Learning in CRFs

- Given a training set of N pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$, we want to estimate the functions $f_r(\mathbf{y}, \mathbf{x}, \mathbf{w})$
- Minimize a surrogate (typically convex) loss and a regularizer

$$\min_{\mathbf{w}} R(\mathbf{w}) + \frac{C}{N} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \bar{\ell}(\mathbf{x}, \mathbf{y}, \mathbf{w})$$

- The surrogate loss $\bar{\ell}$: hinge-loss, log-loss

$$\bar{\ell}_{\text{log}}(\mathbf{x}, \mathbf{y}, \mathbf{w}) = -\ln p_{x,y}(\mathbf{y}; \mathbf{w}).$$

$$\bar{\ell}_{\text{hinge}}(\mathbf{x}, \mathbf{y}, \mathbf{w}) = \max_{\hat{\mathbf{y}} \in \mathcal{Y}} \{ \ell(\mathbf{y}, \hat{\mathbf{y}}) - \mathbf{w}^\top \Phi(\mathbf{x}, \hat{\mathbf{y}}) + \mathbf{w}^\top \Phi(\mathbf{x}, \mathbf{y}) \}$$

More on Learning in CRFs

- Given a training set of N pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$, we want to estimate the functions $f_r(\mathbf{y}, \mathbf{x}, \mathbf{w})$
- Minimize a surrogate (typically convex) loss and a regularizer

$$\min_{\mathbf{w}} R(\mathbf{w}) + \frac{C}{N} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \bar{\ell}(\mathbf{x}, \mathbf{y}, \mathbf{w})$$

- The surrogate loss $\bar{\ell}$: hinge-loss, log-loss

$$\bar{\ell}_{\text{log}}(\mathbf{x}, \mathbf{y}, \mathbf{w}) = -\ln p_{\mathbf{x}, \mathbf{y}}(\mathbf{y}; \mathbf{w}).$$

$$\bar{\ell}_{\text{hinge}}(\mathbf{x}, \mathbf{y}, \mathbf{w}) = \max_{\hat{\mathbf{y}} \in \mathcal{Y}} \{ \ell(\mathbf{y}, \hat{\mathbf{y}}) - \mathbf{w}^\top \Phi(\mathbf{x}, \hat{\mathbf{y}}) + \mathbf{w}^\top \Phi(\mathbf{x}, \mathbf{y}) \}$$

- The assumption is that the model is **log-linear**

$$E(\mathbf{x}, \mathbf{y}, \mathbf{w}) = -F(\mathbf{x}, \mathbf{y}, \mathbf{w}) = -\mathbf{w}^\top \phi(\mathbf{x}, \mathbf{y})$$

and the features decompose in a graph

$$\mathbf{w}^\top \phi(\mathbf{x}, \mathbf{y}) = \sum_{r \in \mathcal{R}} \mathbf{w}_r^\top \phi_r(\mathbf{x}, \mathbf{y})$$

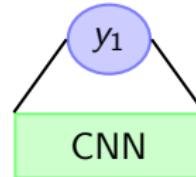
PROBLEM: *How can we remove the log-linear restriction?*

PROBLEM: *How can we remove the log-linear restriction?*

SOLUTION: *Deep Structured Models*

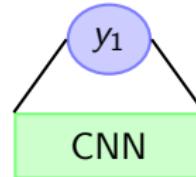
With Pictures ;)

- Standard CNN

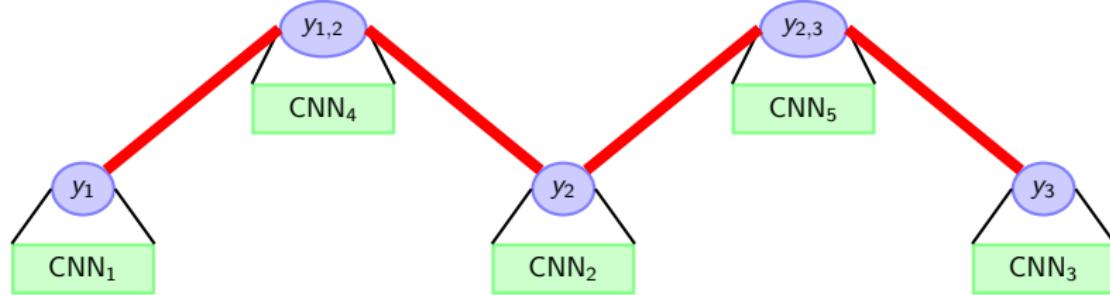


With Pictures ;)

- Standard CNN



- Deep Structured Models



Learning

Probability of a configuration \mathbf{y} :

$$p(\mathbf{y} \mid \mathbf{x}; \mathbf{w}) = \frac{1}{Z(\mathbf{x}, \mathbf{w})} \exp F(\mathbf{x}, \mathbf{y}, \mathbf{w})$$

$$Z(\mathbf{x}, \mathbf{w}) = \sum_{\hat{\mathbf{y}} \in \mathcal{Y}} \exp F(\mathbf{x}, \hat{\mathbf{y}}, \mathbf{w})$$

Learning

Probability of a configuration \mathbf{y} :

$$p(\mathbf{y} \mid \mathbf{x}; \mathbf{w}) = \frac{1}{Z(\mathbf{x}, \mathbf{w})} \exp F(\mathbf{x}, \mathbf{y}, \mathbf{w})$$

$$Z(\mathbf{x}, \mathbf{w}) = \sum_{\hat{\mathbf{y}} \in \mathcal{Y}} \exp F(\mathbf{x}, \hat{\mathbf{y}}, \mathbf{w})$$

Maximize the likelihood of training data via

$$\begin{aligned}\mathbf{w}^* &= \arg \max_{\mathbf{w}} \prod_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} p(\mathbf{y} | \mathbf{x}; \mathbf{w}) \\ &= \arg \max_{\mathbf{w}} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \left(F(\mathbf{x}, \mathbf{y}, \mathbf{w}) - \ln \sum_{\hat{\mathbf{y}} \in \mathcal{Y}} \exp F(\mathbf{x}, \hat{\mathbf{y}}, \mathbf{w}) \right)\end{aligned}$$

Learning

Probability of a configuration \mathbf{y} :

$$p(\mathbf{y} \mid \mathbf{x}; \mathbf{w}) = \frac{1}{Z(\mathbf{x}, \mathbf{w})} \exp F(\mathbf{x}, \mathbf{y}, \mathbf{w})$$

$$Z(\mathbf{x}, \mathbf{w}) = \sum_{\hat{\mathbf{y}} \in \mathcal{Y}} \exp F(\mathbf{x}, \hat{\mathbf{y}}, \mathbf{w})$$

Maximize the likelihood of training data via

$$\begin{aligned}\mathbf{w}^* &= \arg \max_{\mathbf{w}} \prod_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} p(\mathbf{y} | \mathbf{x}; \mathbf{w}) \\ &= \arg \max_{\mathbf{w}} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \left(F(\mathbf{x}, \mathbf{y}, \mathbf{w}) - \ln \sum_{\hat{\mathbf{y}} \in \mathcal{Y}} \exp F(\mathbf{x}, \hat{\mathbf{y}}, \mathbf{w}) \right)\end{aligned}$$

Maximum likelihood is equivalent to maximizing cross-entropy when the target distribution $p_{(\mathbf{x}, \mathbf{y}), \text{tg}}(\hat{\mathbf{y}}) = \delta(\hat{\mathbf{y}} = \mathbf{y})$

Gradient Ascent on Cross Entropy

Program of interest:

$$\max_{\mathbf{w}} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}, \hat{\mathbf{y}}} p_{(\mathbf{x}, \mathbf{y}), \text{tg}}(\hat{\mathbf{y}}) \ln p(\hat{\mathbf{y}} \mid \mathbf{x}; \mathbf{w})$$

Optimize via gradient ascent

$$\begin{aligned} & \frac{\partial}{\partial \mathbf{w}} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}, \hat{\mathbf{y}}} p_{(\mathbf{x}, \mathbf{y}), \text{tg}}(\hat{\mathbf{y}}) \ln p(\hat{\mathbf{y}} \mid \mathbf{x}; \mathbf{w}) \\ &= \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}, \hat{\mathbf{y}}} (p_{(\mathbf{x}, \mathbf{y}), \text{tg}}(\hat{\mathbf{y}}) - p(\hat{\mathbf{y}} \mid \mathbf{x}; \mathbf{w})) \frac{\partial}{\partial \mathbf{w}} F(\hat{\mathbf{y}}, \mathbf{x}, \mathbf{w}) \\ &= \underbrace{\sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \left(\mathbb{E}_{p_{(\mathbf{x}, \mathbf{y}), \text{tg}}} \left[\frac{\partial}{\partial \mathbf{w}} F(\hat{\mathbf{y}}, \mathbf{x}, \mathbf{w}) \right] - \mathbb{E}_{p_{(\mathbf{x}, \mathbf{y})}} \left[\frac{\partial}{\partial \mathbf{w}} F(\hat{\mathbf{y}}, \mathbf{x}, \mathbf{w}) \right] \right)}_{\text{moment matching}} \end{aligned}$$

- Compute predicted distribution $p(\hat{\mathbf{y}} \mid \mathbf{x}; \mathbf{w})$
- Use chain rule to pass back difference between prediction and observation

Deep Structured Learning (algo 1)

[Peng et al. NIPS'09]

Repeat until stopping criteria

- ① Forward pass to compute $F(\mathbf{y}, \mathbf{x}, \mathbf{w})$
- ② Compute $p(\mathbf{y} \mid \mathbf{x}, \mathbf{w})$
- ③ Backward pass via chain rule to obtain gradient
- ④ Update parameters \mathbf{w}

Deep Structured Learning (algo 1)

[Peng et al. NIPS'09]

Repeat until stopping criteria

- ① Forward pass to compute $F(\mathbf{y}, \mathbf{x}, \mathbf{w})$
- ② Compute $p(\mathbf{y} \mid \mathbf{x}, \mathbf{w})$
- ③ Backward pass via chain rule to obtain gradient
- ④ Update parameters \mathbf{w}

What is the PROBLEM?

Deep Structured Learning (algo 1)

[Peng et al. NIPS'09]

Repeat until stopping criteria

- ① Forward pass to compute $F(\mathbf{y}, \mathbf{x}, \mathbf{w})$
- ② Compute $p(\mathbf{y} | \mathbf{x}, \mathbf{w})$
- ③ Backward pass via chain rule to obtain gradient
- ④ Update parameters \mathbf{w}

What is the PROBLEM?

- How do we even represent $F(\mathbf{y}, \mathbf{x}, \mathbf{w})$ if \mathcal{Y} is large?
- How do we compute $p(\mathbf{y} | \mathbf{x}, \mathbf{w})$?

Use the Graphical Model Structure

- ① Use the graphical model $F(\mathbf{y}, \mathbf{x}, \mathbf{w}) = \sum_r f_r(\mathbf{y}_r, \mathbf{x}, \mathbf{w})$

$$\begin{aligned} & \frac{\partial}{\partial \mathbf{w}} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}, \hat{\mathbf{y}}} p_{(\mathbf{x}, \mathbf{y}), \text{tg}}(\hat{\mathbf{y}}) \ln p(\hat{\mathbf{y}} \mid \mathbf{x}; \mathbf{w}) \\ &= \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}, \textcolor{red}{r}} \left(\mathbb{E}_{p_{(\mathbf{x}, \mathbf{y}), \textcolor{red}{r}, \text{tg}}} \left[\frac{\partial}{\partial \mathbf{w}} f_{\textcolor{red}{r}}(\hat{\mathbf{y}}_{\textcolor{red}{r}}, \mathbf{x}, \mathbf{w}) \right] - \mathbb{E}_{p_{(\mathbf{x}, \mathbf{y}), \textcolor{red}{r}}} \left[\frac{\partial}{\partial \mathbf{w}} f_{\textcolor{red}{r}}(\hat{\mathbf{y}}_{\textcolor{red}{r}}, \mathbf{x}, \mathbf{w}) \right] \right) \end{aligned}$$

Use the Graphical Model Structure

- ① Use the graphical model $F(\mathbf{y}, \mathbf{x}, \mathbf{w}) = \sum_r f_r(\mathbf{y}_r, \mathbf{x}, \mathbf{w})$

$$\begin{aligned} & \frac{\partial}{\partial \mathbf{w}} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}, \hat{\mathbf{y}}} p_{(\mathbf{x}, \mathbf{y}), \text{tg}}(\hat{\mathbf{y}}) \ln p(\hat{\mathbf{y}} \mid \mathbf{x}; \mathbf{w}) \\ &= \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}, \textcolor{red}{r}} \left(\mathbb{E}_{p_{(\mathbf{x}, \mathbf{y}), \textcolor{red}{r}, \text{tg}}} \left[\frac{\partial}{\partial \mathbf{w}} f_{\textcolor{red}{r}}(\hat{\mathbf{y}}_{\textcolor{red}{r}}, \mathbf{x}, \mathbf{w}) \right] - \mathbb{E}_{p_{(\mathbf{x}, \mathbf{y}), \textcolor{red}{r}}} \left[\frac{\partial}{\partial \mathbf{w}} f_{\textcolor{red}{r}}(\hat{\mathbf{y}}_{\textcolor{red}{r}}, \mathbf{x}, \mathbf{w}) \right] \right) \end{aligned}$$

- ② Approximate marginals $p_{\textcolor{red}{r}}(\hat{\mathbf{y}}_{\textcolor{red}{r}} \mid \mathbf{x}, \mathbf{w})$ via beliefs $b_r(\hat{\mathbf{y}}_r \mid \mathbf{x}, \mathbf{w})$ computed by:
- Sampling methods
 - Variational methods

Deep Structured Learning (algo 2)

[Schwing & Urtasun Arxiv'15, Zheng et al. Arxiv'15]

Repeat until stopping criteria

- ① Forward pass to compute the $f_r(\mathbf{y}_r, \mathbf{x}, \mathbf{w})$
- ② Compute the $b_r(\mathbf{y}_r | \mathbf{x}, \mathbf{w})$ by running approximated inference
- ③ Backward pass via chain rule to obtain gradient
- ④ Update parameters \mathbf{w}

Deep Structured Learning (algo 2)

[Schwing & Urtasun Arxiv'15, Zheng et al. Arxiv'15]

Repeat until stopping criteria

- ① Forward pass to compute the $f_r(\mathbf{y}_r, \mathbf{x}, \mathbf{w})$
- ② Compute the $b_r(\mathbf{y}_r | \mathbf{x}, \mathbf{w})$ by running approximated inference
- ③ Backward pass via chain rule to obtain gradient
- ④ Update parameters \mathbf{w}

PROBLEM: We have to run inference in the graphical model every time we want to update the weights

How to deal with Big Data

Dealing with large number $|\mathcal{D}|$ of training examples:

- Parallelized across samples (any number of machines and GPUs)
- Usage of mini batches

How to deal with Big Data

Dealing with large number $|\mathcal{D}|$ of training examples:

- Parallelized across samples (any number of machines and GPUs)
- Usage of mini batches

Dealing with large output spaces \mathcal{Y} :

- Variational approximations
- Blending of learning and inference

Approximated Deep Structured Learning

[Schwing & Urtasun Arxiv'15]

Sample parallel implementation:

Partition data \mathcal{D} onto compute nodes

Repeat until stopping criteria

- ① Each compute node uses GPU for CNN Forward pass to compute $f_r(\mathbf{y}_r, \mathbf{x}, \mathbf{w})$
- ② Each compute node estimates beliefs $b_r(\mathbf{y}_r | \mathbf{x}, \mathbf{w})$ for assigned samples
- ③ Backpropagation of difference using GPU to obtain **machine local** gradient
- ④ Synchronize gradient across all machines using MPI
- ⑤ Update parameters \mathbf{w}

Better Option: Interleaving Learning and Inference

- Use LP relaxation instead

$$\min_{\mathbf{w}} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \left(\max_{b_{(x,y)} \in \mathcal{C}_{(x,y)}} \left\{ \sum_{r, \hat{\mathbf{y}}_r} b_{(x,y),r} (\hat{\mathbf{y}}_r) f_r(\mathbf{x}, \hat{\mathbf{y}}_r; \mathbf{w}) + \sum_r \epsilon c_r H(b_{(x,y),r}) \right\} - F(\mathbf{x}, \mathbf{y}; \mathbf{w}) \right)$$

Better Option: Interleaving Learning and Inference

- Use LP relaxation instead

$$\min_{\mathbf{w}} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \left(\max_{b_{(\mathbf{x}, \mathbf{y})} \in \mathcal{C}_{(\mathbf{x}, \mathbf{y})}} \left\{ \sum_{r, \hat{\mathbf{y}}_r} b_{(\mathbf{x}, \mathbf{y}), r} (\hat{\mathbf{y}}_r) f_r(\mathbf{x}, \hat{\mathbf{y}}_r; \mathbf{w}) + \sum_r \epsilon c_r H(b_{(\mathbf{x}, \mathbf{y}), r}) \right\} - F(\mathbf{x}, \mathbf{y}; \mathbf{w}) \right)$$

- More efficient algorithm by blending min. w.r.t. \mathbf{w} and max. of the beliefs b

Better Option: Interleaving Learning and Inference

- Use LP relaxation instead

$$\min_{\mathbf{w}} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \left(\max_{b_{(\mathbf{x}, \mathbf{y})} \in \mathcal{C}_{(\mathbf{x}, \mathbf{y})}} \left\{ \sum_{r, \hat{\mathbf{y}}_r} b_{(\mathbf{x}, \mathbf{y}), r} (\hat{\mathbf{y}}_r) f_r(\mathbf{x}, \hat{\mathbf{y}}_r; \mathbf{w}) + \sum_r \epsilon c_r H(b_{(\mathbf{x}, \mathbf{y}), r}) \right\} - F(\mathbf{x}, \mathbf{y}; \mathbf{w}) \right)$$

- More efficient algorithm by blending min. w.r.t. \mathbf{w} and max. of the beliefs b
- After introducing Lagrange multipliers λ , the dual becomes

$$\min_{\mathbf{w}, \lambda} \sum_{(\mathbf{x}, \mathbf{y}), r} \epsilon c_r \ln \sum_{\hat{\mathbf{y}}_r} \exp \frac{f_r(\mathbf{x}, \hat{\mathbf{y}}_r; \mathbf{w}) + \sum_{c \in C(r)} \lambda_{(\mathbf{x}, \mathbf{y}), c \rightarrow r} (\hat{\mathbf{y}}_c) - \sum_{p \in P(r)} \lambda_{(\mathbf{x}, \mathbf{y}), r \rightarrow p} (\hat{\mathbf{y}}_r)}{\epsilon c_r} - \bar{F}(\mathbf{w}).$$

with $\bar{F}(\mathbf{w}) = \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} F(\mathbf{x}, \mathbf{y}; \mathbf{w})$ the sum of empirical function observations

Better Option: Interleaving Learning and Inference

- Use LP relaxation instead

$$\min_{\mathbf{w}} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \left(\max_{b_{(\mathbf{x}, \mathbf{y})} \in \mathcal{C}_{(\mathbf{x}, \mathbf{y})}} \left\{ \sum_{r, \hat{\mathbf{y}}_r} b_{(\mathbf{x}, \mathbf{y}), r} (\hat{\mathbf{y}}_r) f_r(\mathbf{x}, \hat{\mathbf{y}}_r; \mathbf{w}) + \sum_r \epsilon c_r H(b_{(\mathbf{x}, \mathbf{y}), r}) \right\} - F(\mathbf{x}, \mathbf{y}; \mathbf{w}) \right)$$

- More efficient algorithm by blending min. w.r.t. \mathbf{w} and max. of the beliefs b
- After introducing Lagrange multipliers λ , the dual becomes

$$\min_{\mathbf{w}, \lambda} \sum_{(\mathbf{x}, \mathbf{y}), r} \epsilon c_r \ln \sum_{\hat{\mathbf{y}}_r} \exp \frac{f_r(\mathbf{x}, \hat{\mathbf{y}}_r; \mathbf{w}) + \sum_{c \in C(r)} \lambda_{(\mathbf{x}, \mathbf{y}), c \rightarrow r} (\hat{\mathbf{y}}_c) - \sum_{p \in P(r)} \lambda_{(\mathbf{x}, \mathbf{y}), r \rightarrow p} (\hat{\mathbf{y}}_r)}{\epsilon c_r} - \bar{F}(\mathbf{w}).$$

with $\bar{F}(\mathbf{w}) = \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} F(\mathbf{x}, \mathbf{y}; \mathbf{w})$ the sum of empirical function observations

- We can then do block coordinate descent to solve the minimization problem, and we get the following algorithm ...

Deep Structured Learning (algo 3)

[Chen & Schwing & Yuille & Urtasun ICML'15]

Repeat until stopping criteria

- ① Forward pass to compute the $f_r(\mathbf{y}_r, \mathbf{x}, \mathbf{w})$
- ② Update (some) messages λ
- ③ Backward pass via chain rule to obtain gradient
- ④ Update parameters \mathbf{w}

Deep Structured Learning (algo 4)

[Chen & Schwing & Yuille & Urtasun ICML'15]

Sample parallel implementation:

Partition data \mathcal{D} onto compute nodes

Repeat until stopping criteria

- ① Each compute node uses GPU for CNN Forward pass to compute $f_r(\mathbf{y}_r, \mathbf{x}, \mathbf{w})$
- ② Each compute node updates (some) messages λ
- ③ Backpropagation of difference using GPU to obtain **machine local** gradient
- ④ Synchronize gradient across all machines using MPI
- ⑤ Update parameters \mathbf{w}



Application 1: Character Recognition

- **Task:** Word Recognition from a fixed vocabulary of 50 words, 28×28 sized image patches
- Characters have complex backgrounds and suffer many different distortions
- Training, validation and test set sizes are $10k$, $2k$ and $2k$ variations of words



banal



julep



resty



drein



yojan



mothy



snack



feize



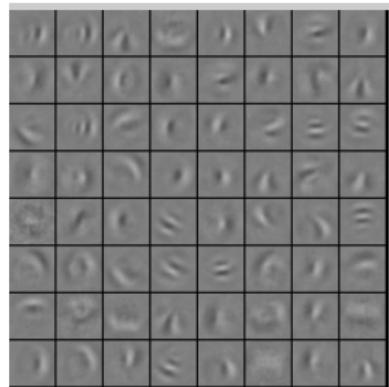
porer

Results

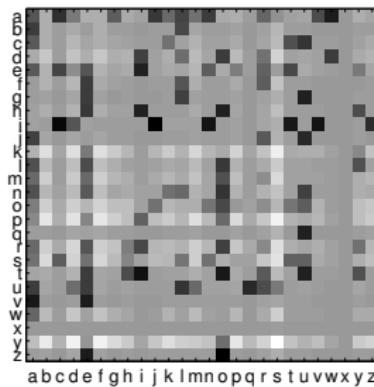
- Graphical model has 5 nodes, MLP for each unary and non-parametric pairwise potentials
- Joint training, structured, deep and more capacity helps**

Grap	MLP	Method	$H_1 = 128$	$H_1 = 256$	$H_1 = 512$	$H_1 = 768$	$H_1 = 1024$
1st	1lay	Unary only	8.60 / 61.32	10.80 / 64.41	12.50 / 65.69	12.95 / 66.66	13.40 / 67.02
		JointTrain	16.80 / 65.28	25.20 / 70.75	31.80 / 74.90	33.05 / 76.42	34.30 / 77.02
		PwTrain	12.70 / 64.35	18.00 / 68.27	22.80 / 71.29	23.25 / 72.62	26.30 / 73.96
		PreTrainJoint	20.65 / 67.42	25.70 / 71.65	31.70 / 75.56	34.50 / 77.14	35.85 / 78.05
2nd	1lay	JointTrain	25.50 / 67.13	34.60 / 73.19	45.55 / 79.60	51.55 / 82.37	54.05 / 83.57
		PwTrain	10.05 / 58.90	14.10 / 63.44	18.10 / 67.31	20.40 / 70.14	22.20 / 71.25
		PreTrainJoint	28.15 / 69.07	36.85 / 75.21	45.75 / 80.09	50.10 / 82.30	52.25 / 83.39
		$H_1 = 512$	$H_2 = 32$	$H_2 = 64$	$H_2 = 128$	$H_2 = 256$	$H_2 = 512$
1st	2lay	Unary only	15.25 / 69.04	18.15 / 70.66	19.00 / 71.43	19.20 / 72.06	20.40 / 72.51
		JointTrain	35.95 / 76.92	43.80 / 81.64	44.75 / 82.22	46.00 / 82.96	47.70 / 83.64
		PwTrain	34.85 / 79.11	38.95 / 80.93	42.75 / 82.38	45.10 / 83.67	45.75 / 83.88
		PreTrainJoint	42.25 / 81.10	44.85 / 82.96	46.85 / 83.50	47.95 / 84.21	47.05 / 84.08
2nd	2lay	JointTrain	54.65 / 83.98	61.80 / 87.30	66.15 / 89.09	64.85 / 88.93	68.00 / 89.96
		PwTrain	39.95 / 81.14	48.25 / 84.45	52.65 / 86.24	57.10 / 87.61	62.90 / 89.49
		PreTrainJoint	62.60 / 88.03	65.80 / 89.32	68.75 / 90.47	68.60 / 90.42	69.35 / 90.75

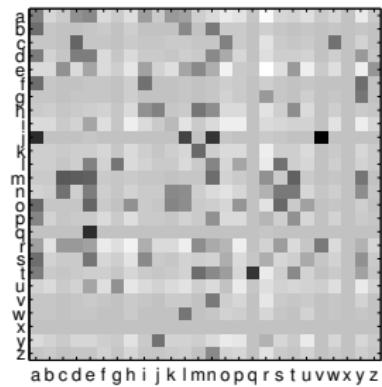
Learned Weights



Unary weights



distance-1 edges



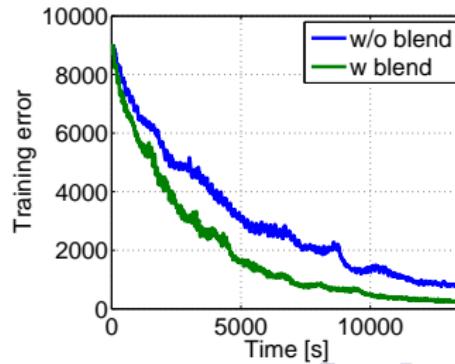
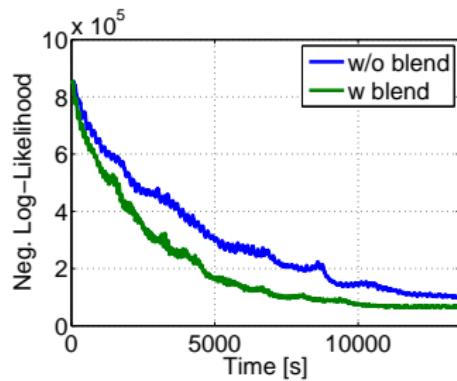
distance-2 edges

Example 2: Image Tagging

[Chen & Schwing & Yuille & Urtasun ICML'15]

- Flickr dataset: 38 possible tags, $|\mathcal{Y}| = 2^{38}$
- 10k training, 10k test examples

Training method	Prediction error [%]
Unary only	9.36
Piecewise	7.70
Joint (with pre-training)	7.25



Visual results



female/indoor/portrait
female/indoor/portrait



sky/plant life/tree
sky/plant life/tree



water/animals/sea
water/animals/sky



animals/dog/indoor
animals/dog



indoor/flower/plant life
∅

Learned class correlations

female	0.00	0.68	0.04	0.24	-0.01	-0.05	0.07	-0.01	0.01
people	0.68	0.00	0.06	0.36	-0.05	-0.12	0.74	-0.04	-0.03
indoor	0.04	0.06	0.00	0.07	-0.35	-0.34	0.02	-0.15	-0.21
portrait	0.24	0.36	0.07	0.00	-0.02	-0.01	0.12	0.02	0.05
sky	-0.01	-0.05	-0.35	-0.02	0.00	0.24	-0.00	0.44	0.30
plant life	-0.05	-0.12	-0.34	-0.01	0.24	0.00	-0.07	0.09	0.68
male	0.07	0.74	0.02	0.12	-0.00	-0.07	0.00	0.00	-0.02
clouds	-0.01	-0.04	-0.15	0.02	0.44	0.09	0.00	0.00	0.11
tree	0.01	-0.03	-0.21	0.05	0.30	0.68	-0.02	0.11	0.00

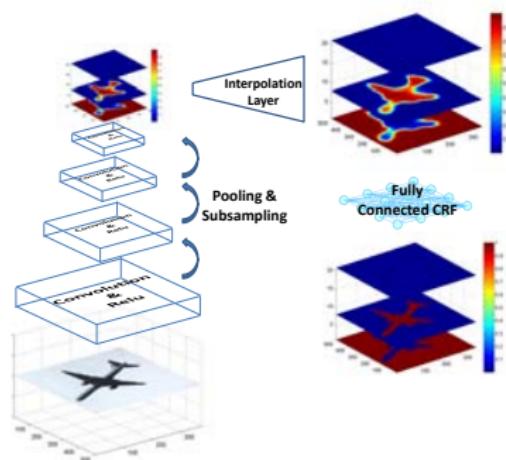
female people indoor portrait sky plant life male clouds tree

Only part of the correlations are shown for clarity

Example 3: Semantic Segmentation

[Chen et al. ICLR'15; Krähenbühl & Koltun NIPS'11, ICML'13; Zhen et al. Arxiv'15;
Schwing & Urtasun Arxiv'15]

- $|\mathcal{Y}| = 21^{350 \cdot 500}$, $\approx 10k$ training, ≈ 1500 test examples
- Oxford-net pre trained on PASCAL, predicts 40×40 + upsampling
- The graphical model is a fully connected CRF with Gaussian potentials
- Inference using (algo2), with mean-field as approx. inference



Pascal VOC 2012 dataset

[Chen et al. ICLR'15; Krähenbühl & Koltun NIPS'11,ICML'13; Zhen et al. Arxiv'15;
Schwing & Urtasun Arxiv'15]

- $|\mathcal{Y}| = 21^{350 \cdot 500}$, $\approx 10k$ training, ≈ 1500 test examples
- Oxford-net pre trained on PASCAL, predicts 40×40 + upsampling
- The graphical model is a fully connected CRF with Gaussian potentials
- Inference using (algo2), with mean-field as approx. inference

Training method	Mean IoU [%]
Unary only	61.476
Joint	64.060

Pascal VOC 2012 dataset

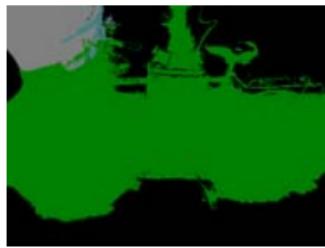
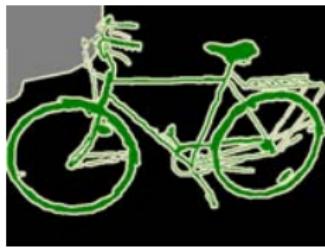
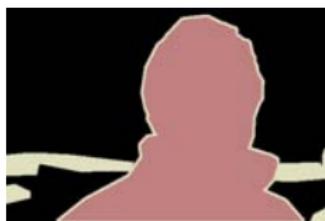
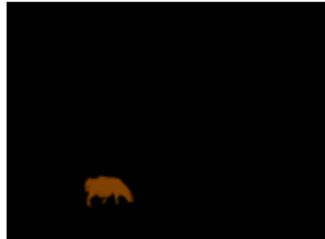
[Chen et al. ICLR'15; Krähenbühl & Koltun NIPS'11,ICML'13; Zhen et al. Arxiv'15;
Schwing & Urtasun Arxiv'15]

- $|\mathcal{Y}| = 21^{350 \cdot 500}$, $\approx 10k$ training, ≈ 1500 test examples
- Oxford-net pre trained on PASCAL, predicts 40×40 + upsampling
- The graphical model is a fully connected CRF with Gaussian potentials
- Inference using (algo2), with mean-field as approx. inference

Training method	Mean IoU [%]
Unary only	61.476
Joint	64.060

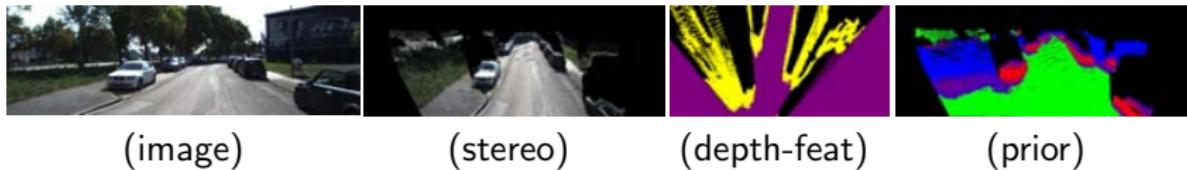
- **Disclaimer:** Much better results now with a few tricks. Zheng et al. 15 is now at 74.7%!

Visual results



Example 4: 3D Object Proposals for Detection

- Use structured prediction to learn to propose object candidates (i.e., grouping)



- Use deep learning to do final detection: OxfordNet



- Only 1.2s to generate proposals

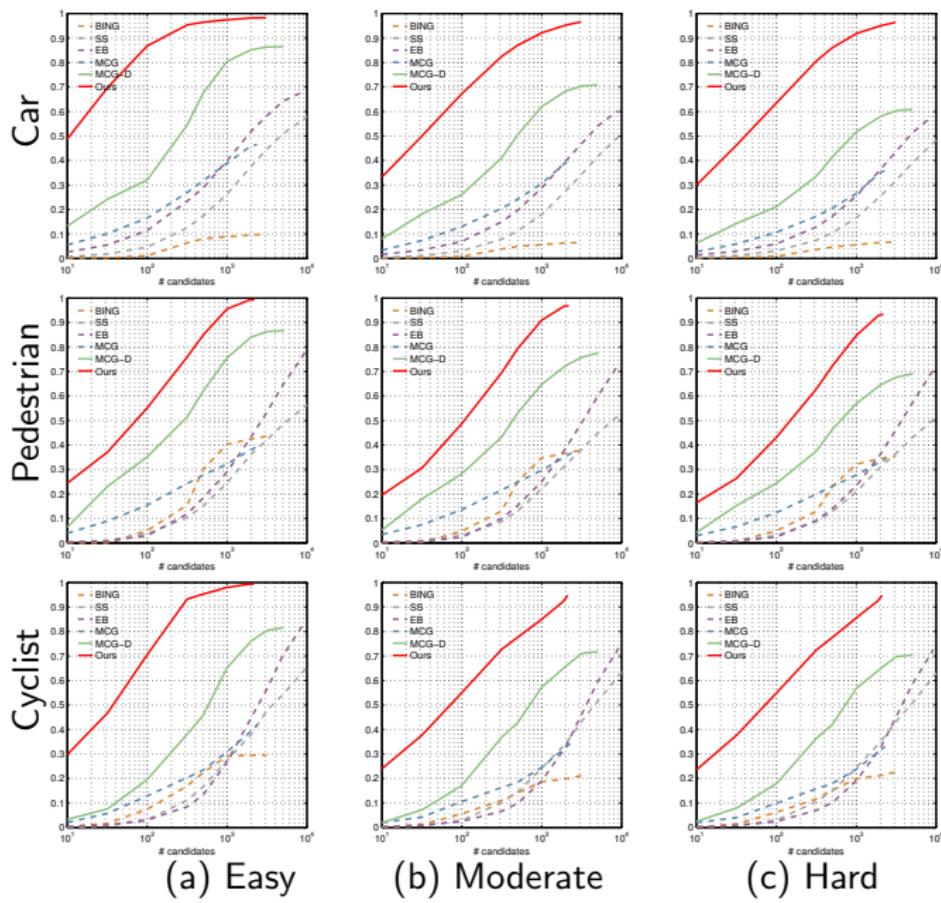
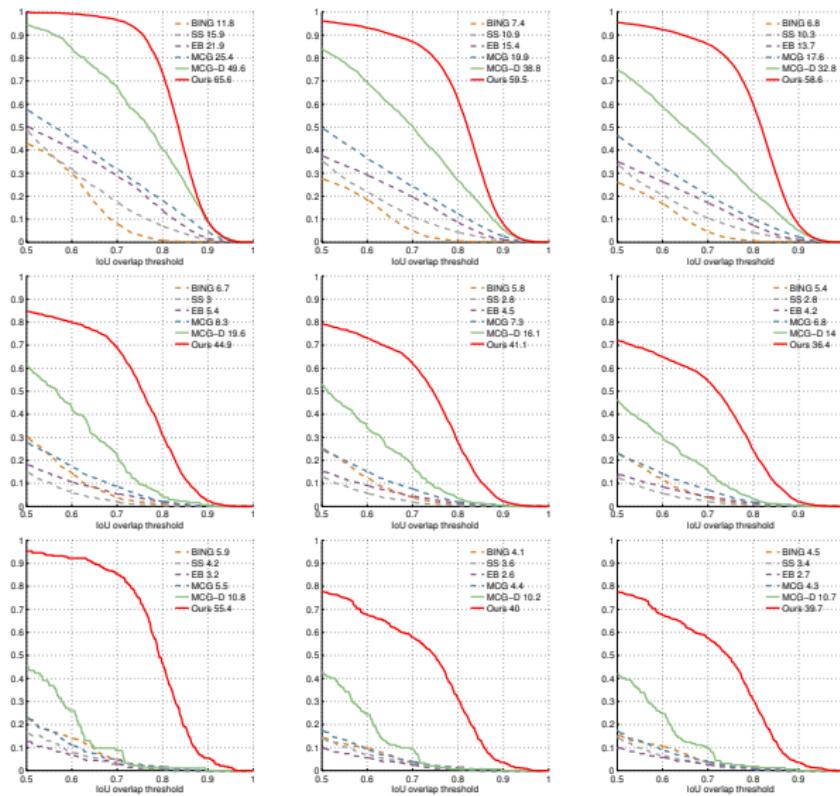


Figure : **Proposal recall:** 0.7 overlap threshold for *Car*, and 0.5 for rest.



(a) Easy

(b) Moderate

(c) Hard

Figure : Recall vs IoU for 500 proposals. (Top) Cars, (Middle) Pedestrians, (Bottom) Cyclists.

KITTI Detection Results

[X. Chen, K. Kundu and S. Fidler and R. Urtasun, On Arxiv soon]

	Cars			Pedestrians			Cyclists		
	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
LSVM-MDPM-sv	68.02	56.48	44.18	47.74	39.36	35.95	35.04	27.50	26.21
SquaresICF	-	-	-	57.33	44.42	40.08	-	-	-
DPM-C8B1	74.33	60.99	47.16	38.96	29.03	25.61	43.49	29.04	26.20
MDPM-un-BB	71.19	62.16	48.43	-	-	-	-	-	-
DPM-VOC+VP	74.95	64.71	48.76	59.48	44.86	40.37	42.43	31.08	28.23
OC-DPM	74.94	65.95	53.86	-	-	-	-	-	-
AOG	84.36	71.88	59.27	-	-	-	-	-	-
SubCat	84.14	75.46	59.71	54.67	42.34	37.95	-	-	-
DA-DPM	-	-	-	56.36	45.51	41.08	-	-	-
Fusion-DPM	-	-	-	59.51	46.67	42.05	-	-	-
R-CNN	-	-	-	61.61	50.13	44.79	-	-	-
FilteredICF	-	-	-	61.14	53.98	49.29	-	-	-
pAUCEnST	-	-	-	65.26	54.49	48.60	51.62	38.03	33.38
MV-RGBD-RF	-	-	-	70.21	54.56	51.25	54.02	39.72	34.82
3DVP	87.46	75.77	65.38	-	-	-	-	-	-
Regionlets	84.75	76.45	59.70	73.14	61.15	55.21	70.41	58.72	51.83
Ours	88.33	87.14	76.11	70.16	59.35	52.76	77.94	67.35	59.49

Table : Average Precision (AP) (in %) on the test set of the KITTI Object Detection Benchmark.

KITTI Detection Results

[X. Chen, K. Kundu and S. Fidler and R. Urtasun, On Arxiv soon]

	Cars			Pedestrians			Cyclists		
	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
AOG	43.81	38.21	31.53	-	-	-	-	-	-
DPM-C8B1	59.51	50.32	39.22	31.08	23.37	20.72	27.25	19.25	17.95
LSVM-MDPM-sv	67.27	55.77	43.59	43.58	35.49	32.42	27.54	22.07	21.45
DPM-VOC+VP	72.28	61.84	46.54	53.55	39.83	35.73 /	30.52	23.17	21.58
OC-DPM	73.50	64.42	52.40	-	-	-	-	-	-
SubCat	83.41	74.42	58.83	44.32	34.18	30.76	-	-	-
3DVP	86.92	74.59	64.11	-	-	-	-	-	-
Ours	83.03	80.21	69.60	48.58	40.56	36.08	57.72	48.21	42.72

Table : AOS scores on the KITTI Object Detection and Orientation Benchmark (test set).

Car Results

[X. Chen, K. Kundu, Y. Zhu, S. Fidler and R. Urtasun, On Arxiv soon]



Pedestrian Results

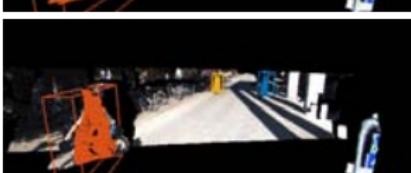
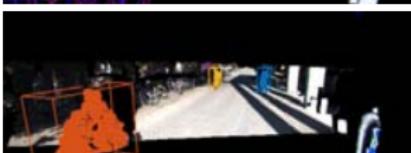
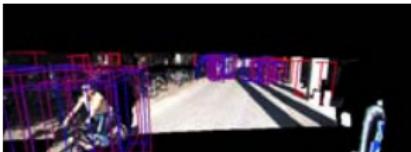
[X. Chen, K. Kundu, Y. Zhu, S. Fidler and R. Urtasun, On Arxiv soon]



Cyclist Results

[X. Chen, K. Kundu, Y. Zhu, S. Fidler and R. Urtasun, On Arxiv soon]

2D images
Top 100 Prop.
Ground truth
Best proposals



Example 5: More Precise Grouping

- Given a single image, we want to infer **Instance-level Segmentation** and **Depth Ordering**



- Use deep convolutional nets to do both tasks simultaneously
- Trick: Encode both tasks with a single parameterization
- Run the conv. net at multiple resolutions
- Use MRF to form a single coherent explanation across all the image combining the conv nets at multiple resolutions
- Important:** we do not use a single pixel-wise training example!

Results on KITTI

[Z. Zhang, A. Schwing, S. Fidler and R. Urtasun, ICCV '15]



More Results (including failures/difficulties)

[Z. Zhang, A. Schwing, S. Fidler and R. Urtasun, ICCV '15]

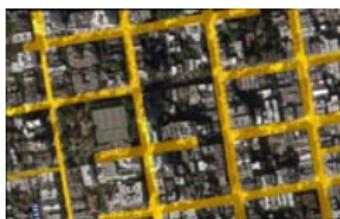


Example 6: Enhancing freely-available maps

[G. Matthyus, S. Wang, S. Fidler and R. Urtasun, ICCV '15]



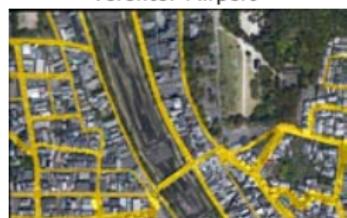
Toronto: Airport



San Francisco: Russian Hill



NYC: Times square



Kyoto: Kinkakuji



Sydney: At Harbour bridge



Monte Carlo: Casino

- Enhancing OpenStreetMaps
- Can be trained on a single image and test on the whole world
- Trick: Not to reason at the pixel level
- Very efficient: 0.1s/km of road
- Preserves topology and is state-of-the-art

Example 7: Fashion

[E. Simo-Serra, S. Fidler, F. Moreno, R. Urtasun, CVPR15]



LOS ANGELES, CA

466 FANS

288 VOTES

62 FAVOURITES

TAGS

CHIC

EVERDAY

FALL

COLOURS

WHITE-BOOTS

NOVEMBER 10, 2014

GARMENTS

White Cheap Monday Boots

Chilli Beans Sunglasses

Missguided Romper

Daniel Wellington Watch

COMMENTS

Nice!!

Love the top!

cute

...

Figure : An example of a post on <http://www.chictopia.com>. We crawled the site for 180K posts.

How Fashionable Are You?

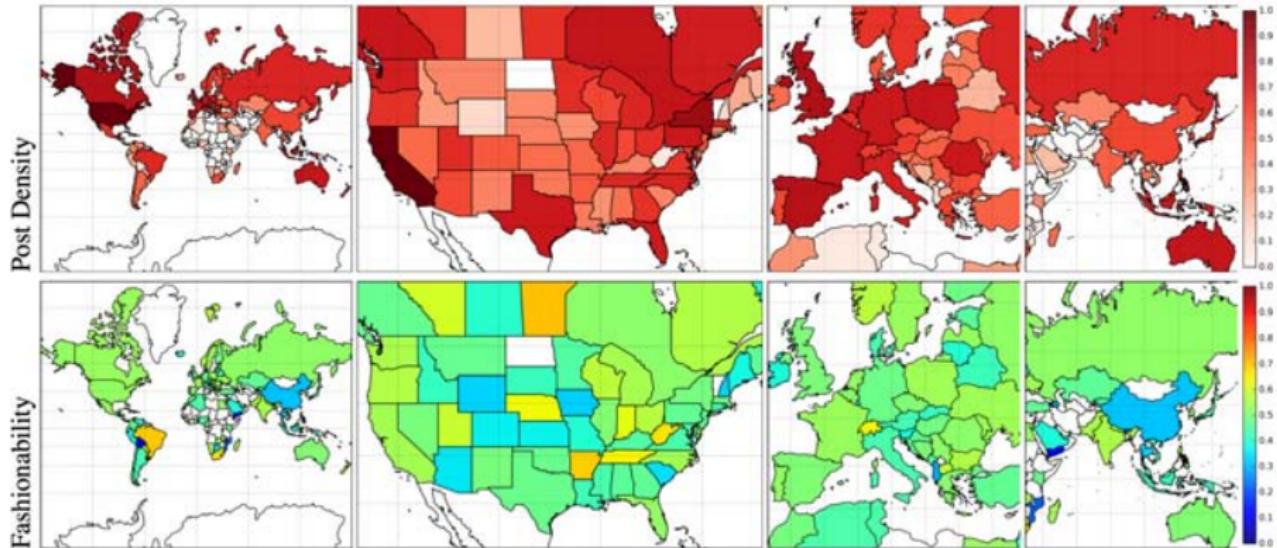


Figure 3: Visualization of the density of posts and fashionability by country.

How Fashionable Are You?

City Name	Posts	Fashionability
Manila	4269	6.627
Los Angeles	8275	6.265
Melbourne	1092	6.176
Montreal	1129	6.144
Paris	2118	6.070
Amsterdam	1111	6.059
Barcelona	1292	5.845
Toronto	1471	5.765
Bucharest	1385	5.667
New York	4984	5.514
London	3655	5.444
San Francisco	2880	5.392
Madrid	1747	5.371
Vancouver	1468	5.266
Jakarta	1156	4.398

Table 2: Fashionability of cities with at least 1000 posts.

How Fashionable Are You?

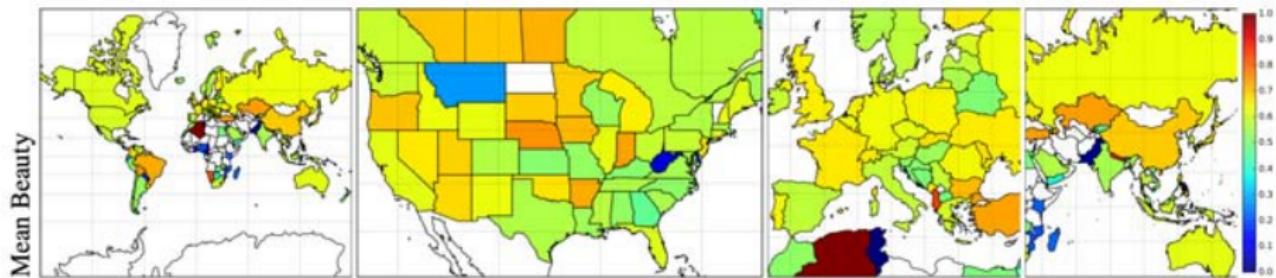
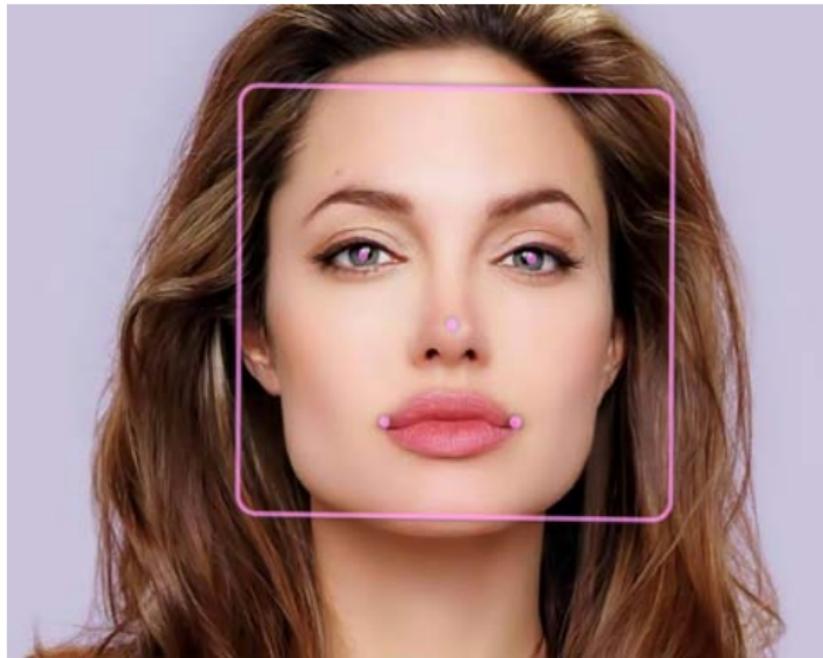


Figure : We ran a face detector that predicts also beauty of the face, age, ethnicity, mood.

How Fashionable Are You?

- Face detector + attributes



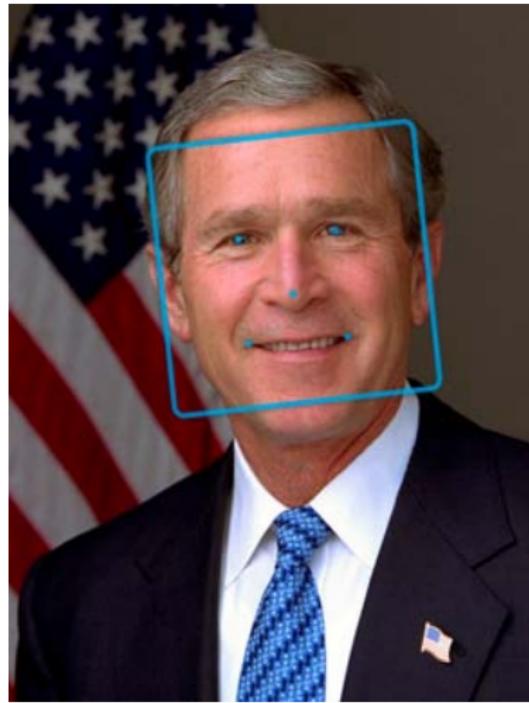
SHARE ON TWITTER

```
confidence : true ( value : 1 )
pose :roll(0.9) ,yaw(3.59) ,pitch(8.63)
race : white(0.28)
emotion : calm:68%,happy:28%
age : 29.52 ( value : 29.52 )
smile : true ( value : 0.65 )
glasses : no glass ( value : 0 )
sunglasses : false ( value : 0 )
eye_closed : open ( value : 0 )
mouth_open_wide : 3% ( value : 0.03 )
beauty : 99.42 ( value : 0.99422 )
gender : female ( value : 0 )
```

<http://www.rekognition.com>

How Fashionable Are You?

- Face detector + attributes



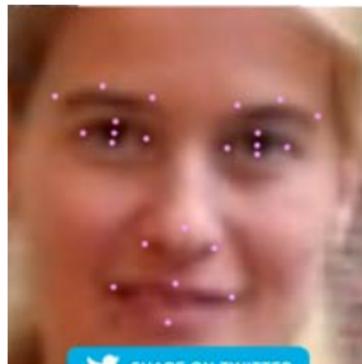
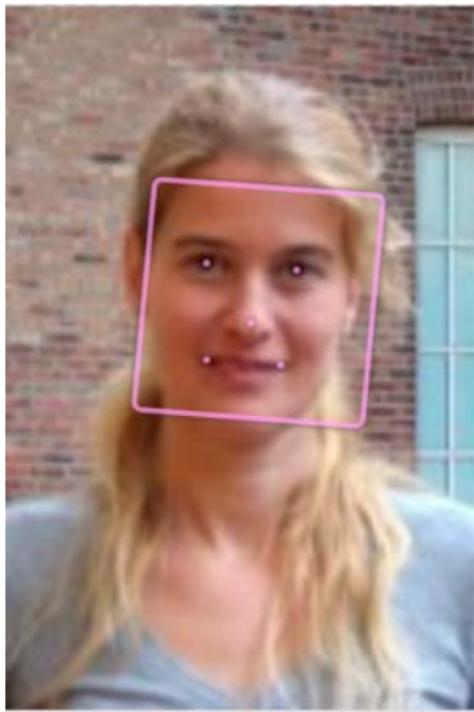
SHARE ON TWITTER

confidence : true (value : 1)
pose : roll(-6.26) , yaw(-6.81) , pitch(1.66)
race : white(0.99)
emotion : happy:92%,confused:1%
age : 60.9 (value : 60.9)
smile : true (value : 0.87)
glasses : no glass (value : 0.01)
sunglasses : false (value : 0)
eye_closed : open (value : 0)
mouth_open_wide : 3% (value : 0.03)
beauty : 78.62 (value : 0.78628)
gender : male (value : 1)

<http://www.rekognition.com>

How Fashionable Are You?

- Face detector + attributes



confidence : true (value : 1)
pose :roll(4.3) ,yaw(10.36) ,pitch(-5.4)
race : white(0.73)
emotion : happy:99%,calm:3%
age : 29.12 (value : 29.12)
smile : true (value : 0.86)
glasses : no glass (value : 0)
sunglasses : false (value : 0)
eye_closed : open (value : 0)
mouth_open_wide : 0% (value : 0)
beauty : 53.67 (value : 0.53674)
gender : female (value : 0.03)

<http://www.rekognition.com>

How Fashionable Are You?

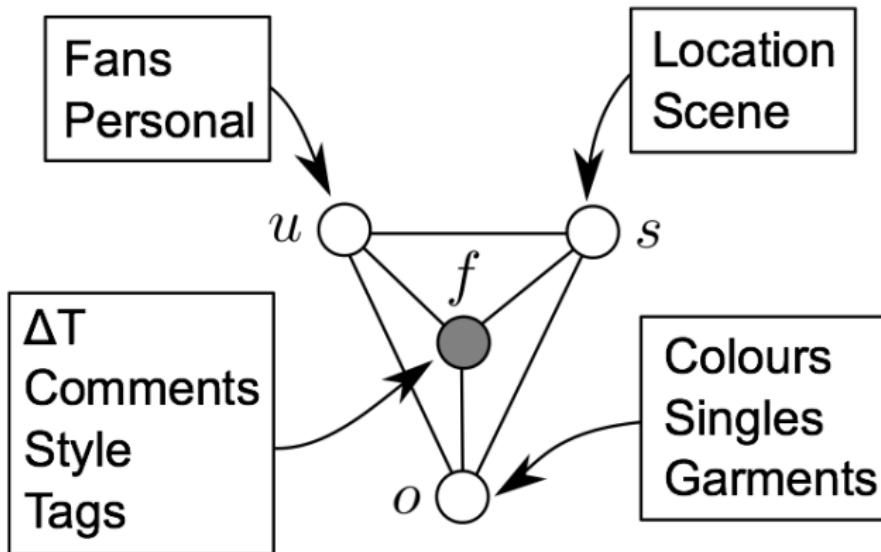


Figure : Our model is a Conditional Random Field that uses many visual and textual features, as well as meta-data features such as where the user is from.

How Fashionable Are You?

True Positives



False Positives



Figure : We predict fashionability of users.

Outfits



Figure : We predict what kind of outfit the person wears.

How Fashionable Can You Become?



Current Outfit:
Pink Outfit (3)

Recommendations:
Heels (8)
Pastel Shirts/Skirts (8)
Black/Gray Tights/Sweater (5)



Current Outfit:
Pink/Blue Shoes/Dress Shorts (3)

Recommendations:
Black/Gray Tights/Sweater (5)
Black Casual (5)
Black Boots/Tights (5)



Current Outfit:
Pink/Black Misc. (5)

Recommendations:
Pastel Dress (8)
Black/Blue Going out (8)
Black Casual (8)



Current Outfit:
Blue with Scarf (3)

Recommendations:
Heels (8)
Pastel Shirts/Skirts (8)
Black Casual (8)



Current Outfit:
Pink/Blue Shoes/Dress Shorts (3)

Recommendations:
Black Casual (7)
Black Heavy (3)
Navy and Bags (3)



Current Outfit:
Formal Blue/Brown (5)

Recommendations:
Pastel Shirts/Skirts (9)
Black/Blue Going out (8)
Black Boots/Tights (8)

Figure : Examples of recommendations provided by our model. The parenthesis we show the fashionability scores.

Not a big deal... but

- Appear all over the Tech and News

News and Tech websites

NewScientist

New Scientist

QUARTZ

Quartz

TECH TIMES

Tech Times

WIRED.co.uk

Wired, UK

Mashable

Mashable



AOL News (video)

THE HUFFINGTON POST
UNITED KINGDOM

Huffington Post, UK (video)

HUFFPOST STYLE
CANADA

Huffington Post, Canada

msn

MSN, Canada

Protein

Protein

YAHOO! NEWS
CANADA

Yahoo, Canada

ScienceDaily

Science Daily

mailOnline

Daily Mail, UK

psfk

PSFK

the star.com

Toronto Star

gizmag

Gizmag

The Record.com

TheRecord.com

iDigitalTimes

iDigitalTimes

Not a big deal... but

- Appear all over the Tech and News
- All over the Fashion press

Fashion Magazines (Online)

BAZAAR
Harper's Bazaar

marieclaire
Marie Claire

ELLE
Elle

Red
Red Magazine
(UK)

GLAMOUR
Glamour

YAHOO!
STYLE
Yahoo Style

COSMOPOLITAN
Cosmopolitan

FASHION
Fashion
Magazine

The Pool
The Pool (UK)

FashionNotes
FASHION + TECHNOLOGY
FashionNotes

Not a big deal... but

- Appear all over the Tech and News
- All over the Fashion press
- International News and TV (Fox, BBC, SkypeNews, RTVE, etc)

International News



Vogue (ES)



Wired (DE)



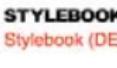
Jetzt (DE)



SinEmbargo (MX)



Nauka (PL)



Stylebook (DE)



Marie Claire (FR)



Fashion Police (NG)



Amsterdam Fashion (NL)



Ansa (IT)



Pluska (SK)



Elle (NL)



IT News (SK)



PopSugar (AU)



CenárioMT (BR)



Pressetext (AT)



La Gazzetta
dello Sport (IT)



Woman (ES)



CSIC (ES)



EFE (ES)

Cosmopolitan (UK): *The technology scores your facial attributes (this just keeps getting better, doesn't it) from your looks, to your age, and the emotion you're showing, before combining all the information using an equation SO complex we won't begin to go into it.*

But the Most Important Impact



Previous Work

- Use the hinge loss to optimize the unaries only which are neural nets (Li and Zemel 14). Correlations between variables are not used for learning

Previous Work

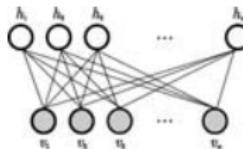
- Use the hinge loss to optimize the unaries only which are neural nets (Li and Zemel 14). Correlations between variables are not used for learning
- If inference is tractable, Conditional Neural Fields (Peng et al. 09) use back-propagation on the log-loss

Previous Work

- Use the hinge loss to optimize the unaries only which are neural nets (Li and Zemel 14). Correlations between variables are not used for learning
- If inference is tractable, Conditional Neural Fields (Peng et al. 09) use back-propagation on the log-loss
- Decision Tree Fields (Nowozin et al. 11), use complex region potentials (decision trees), but given the tree, it is still linear in the parameters. Trained using pseudo likelihood.

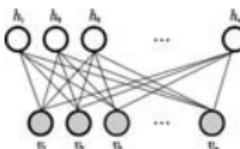
Previous Work

- Use the hinge loss to optimize the unaries only which are neural nets (Li and Zemel 14). Correlations between variables are not used for learning
- If inference is tractable, Conditional Neural Fields (Peng et al. 09) use back-propagation on the log-loss
- Decision Tree Fields (Nowozin et al. 11), use complex region potentials (decision trees), but given the tree, it is still linear in the parameters. Trained using pseudo likelihood.
- Restricted Boltzmann Machines (RBMs): Generative model that has a very particular architecture so that inference is tractable via sampling (Salakhutdinov 07). Problems with partition function.



Previous Work

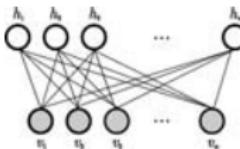
- Use the hinge loss to optimize the unaries only which are neural nets (Li and Zemel 14). Correlations between variables are not used for learning
- If inference is tractable, Conditional Neural Fields (Peng et al. 09) use back-propagation on the log-loss
- Decision Tree Fields (Nowozin et al. 11), use complex region potentials (decision trees), but given the tree, it is still linear in the parameters. Trained using pseudo likelihood.
- Restricted Boltzmann Machines (RBMs): Generative model that has a very particular architecture so that inference is tractable via sampling (Salakhutdinov 07). Problems with partition function.



- (Domke 13) treat the problem as learning a set of logistic regressors

Previous Work

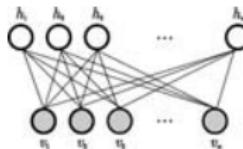
- Use the hinge loss to optimize the unaries only which are neural nets (Li and Zemel 14). Correlations between variables are not used for learning
- If inference is tractable, Conditional Neural Fields (Peng et al. 09) use back-propagation on the log-loss
- Decision Tree Fields (Nowozin et al. 11), use complex region potentials (decision trees), but given the tree, it is still linear in the parameters. Trained using pseudo likelihood.
- Restricted Boltzmann Machines (RBMs): Generative model that has a very particular architecture so that inference is tractable via sampling (Salakhutdinov 07). Problems with partition function.



- (Domke 13) treat the problem as learning a set of logistic regressors
- Fields of experts (Roth et al. 05), not deep, use CD training

Previous Work

- Use the hinge loss to optimize the unaries only which are neural nets (Li and Zemel 14). Correlations between variables are not used for learning
- If inference is tractable, Conditional Neural Fields (Peng et al. 09) use back-propagation on the log-loss
- Decision Tree Fields (Nowozin et al. 11), use complex region potentials (decision trees), but given the tree, it is still linear in the parameters. Trained using pseudo likelihood.
- Restricted Boltzmann Machines (RBMs): Generative model that has a very particular architecture so that inference is tractable via sampling (Salakhutdinov 07). Problems with partition function.



- (Domke 13) treat the problem as learning a set of logistic regressors
- Fields of experts (Roth et al. 05), not deep, use CD training
- Many ideas go back to (Boutou 91)

Conclusions and Future Work

Conclusions:

- Modeling of correlations between variables
- Non-linear dependence on parameters
- Joint training of many convolutional neural networks
- Parallel implementation
- Wide range of applications: Word recognition, Tagging, Segmentation

Future work:

- Latent Variables
- More applications

Acknowledgments

- Liang Chieh Chen (student)
- Xiaozhi Chen (student)
- Sanja Fidler
- Gellert Matthyus (student)
- Francesc Moreno
- Alexander Schwing (postdoc)
- Edgar Simo-Serra (student)
- Shenlong Wang (student)
- Allan Yuille
- Ziyu Zhang (student)
- Yukun Zhu (student)

The introductory slides on deep learning have been inspired by M. Ranzato tutorial on deep learning and S. Fidler lecture notes for CSC420