COMP [56]630– Machine Learning

Lecture 10 – Backprop (revisited), Demo, Regularization, Intro to DL

Backpropagation

- Before training the neural network, select an initial value for parameters.
  - Common practice: randomly initialize the parameters to small values (e.g., normally distributed around zero; N(0; 0:1))
- Next step: update the parameters.
- After a single forward pass through the neural network, the output will be a predicted value
- We can then compute the loss L, in our case the log loss

$$\mathcal{L}(\hat{y}, y) = -\Big[(1-y)\log(1-\hat{y}) + y\log\hat{y}\Big]$$

Defining Variables

Let X be the input features . [n x f]
y be the output (target) labels [n x 1]

Define the weights [parameters] of the neural network

First layer → W_1 , bias → b_1
Second layer → W_2 , bias → b_2
Third layer → W_3 , bias → b_3

The forward pass

The output of layer 1 is

$$z_1 = w_1 \cdot x + b_1$$
$$a_1 = g(z_1)$$ where "g" is the activation function

The output of layer 2 is

$$z_2 = w_2 \cdot a_1 + b_2$$
$$a_2 = g(z_2)$$

The output of layer 3 is

$$z_3 = w_3 \cdot a_2 + b_3$$
$$a_3 = g(z_3)$$

The o/p of the neural network is

$$\hat{y} = a_3$$

Defining the loss

Task : Binary classification

$\Rightarrow$ loss is log loss or binary cross entropy

$$L = -\left[(1-y)\cdot\log(1-\hat{y}) + y\cdot\log(\hat{y})\right]$$

Learning with gradient descent

To learn the parameters

(1) Provide random values for weights w_1, w_2, w_3 & biases b_1, b_2, b_3

(2) Update the weights using gradient descent by

$$w_i := w_i - \alpha \cdot \frac{dL}{dw_i}$$

$$b_i := b_i - \alpha \cdot \frac{dL}{db_i}$$

where i refers to the ith layer.

(3) repeat until convergence

Finding dL/dW_3

To find $\frac{\partial L}{\partial w_3}$ to update the third layer.

$$\frac{\partial L}{\partial w_3} = \frac{\partial}{\partial w_3}\left[-\left[(1-y)\cdot\log(1-\hat{y}) + y\cdot\log(\hat{y})\right]\right]$$

$$= -\left[(1-y)\cdot\frac{\partial}{\partial w_3}\left(\log(1-\hat{y})\right) + y\cdot\frac{\partial}{\partial w_3}\left(\log(\hat{y})\right)\right]$$

Finding dL/dW_3

Remember: $\frac{d}{dz}(\log(z)) = \frac{1}{z}$

Finding dL/dW_3

$$\therefore \frac{dL}{dw_3} = -\left[\frac{(1-y)}{(1-\hat{y})} \cdot \frac{d}{dw_3}(1-\hat{y}) + \frac{y}{\hat{y}} \cdot \frac{d}{dw_3}(\hat{y})\right]$$

$$= -\left[\frac{-(1-y)}{(1-\hat{y})} \cdot \frac{d\hat{y}}{dw_3} + \frac{y}{\hat{y}} \cdot \frac{d\hat{y}}{dw_3}\right]$$

$$= \frac{(1-y)}{(1-\hat{y})} \cdot \frac{d\hat{y}}{dw_3} - \frac{y}{\hat{y}} \cdot \frac{d\hat{y}}{dw_3}$$

$$= \left[\frac{(1-y)}{(1-\hat{y})} - \frac{y}{\hat{y}}\right] \cdot \frac{d\hat{y}}{dw_3}$$

Finding dL/dW_3

$$\Rightarrow \frac{\partial L}{\partial w_3} = \frac{\hat{y}(1-\hat{y}) - y(1-\hat{y})}{\hat{y}(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

$$= \frac{\hat{y} - y\hat{y} - y + y\hat{y}}{\hat{y}(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

$$= \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

Finding dL/dW_3

we know that $\hat{y} = a_3 = g(z_3)$

Finding dL/dW_3

if $g(z)$ is a sigmoid function,

$$g'(z) = g(z) \cdot (1 - g(z))$$

$$\therefore \frac{\partial \hat{y}}{\partial w_3} = \frac{\partial a_3}{\partial w_3} = \frac{\partial \cdot g(z_3)}{\partial w_3}$$

$$= g(z_3) \cdot (1 - g(z_3)) \cdot \frac{\partial z_3}{\partial w_3}$$

$$= a_3 (1 - a_3) \cdot \frac{\partial}{\partial w_3} [w_3 a_2 + b_3]$$

$$= a_3 (1 - a_3) \cdot a_2$$

$$= \hat{y} (1 - \hat{y}) \cdot a_2$$

Finding dL/dW_3

$$\therefore \frac{\partial L}{\partial w_3} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot [\hat{y} \cdot (1-\hat{y}) \cdot a_2]$$

$$\therefore \boxed{\frac{\partial L}{\partial w_3} = (\hat{y} - y) \cdot a_2}$$

$$\boxed{\frac{\partial L}{\partial w_3} = (a_3 - y) \cdot a_2^T}$$

when providing a vectorized solution.

Finding dL/dW_3

$$\therefore \frac{\partial L}{\partial w_3} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot [\hat{y} \cdot (1-\hat{y}) \cdot a_2]$$

$$\therefore \boxed{\frac{\partial L}{\partial w_3} = (\hat{y} - y) \cdot a_2}$$

$$\boxed{\frac{\partial L}{\partial w_3} = (a_3 - y) \cdot a_2^T}$$

when providing a vectorized solution.

Similarly,

$$\boxed{\frac{\partial L}{\partial b_3} = a_3 - y}$$

Finding dL/dW_2

- We need to use the chain rule from calculus.
- Why?
- There is no direct relationship between the weights W_2 and the loss L.
- You cannot differentiate a variable by another if there is no direct relationship
  - Else it would be 0
  - Not true if the variable/function is composite

Finding dL/dW_2

We know that Loss is dependent on $\hat{y} = a_3$

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{?} \cdot \frac{?}{\partial w_2}$$

we know that $a_3 = g(z_3)$

$\Rightarrow a_3$ is dependent on z_3

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{?} \cdot \frac{?}{\partial w_2}$$

Finding dL/dW_2

We know that $z_3 = w_3 a_2 + b_3$

$\Rightarrow z_3$ is dependent on a_2

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{?} \cdot \frac{?}{\partial w_2}$$

Finding dL/dW_2

But $a_2 = g(z_2)$

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{?} \cdot \frac{?}{\partial w_2}$$

But $z_2 = w_2 \cdot a_1 + b_2$

$$\therefore \boxed{\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2}}$$

Finding dL/dW_2

If we rewrite $\frac{dL}{dw_3}$ using chain rule,

$$\frac{dL}{dw_3} = \frac{dL}{da_3} \cdot \frac{da_3}{dz_3} \cdot \frac{dz_3}{dw_3}$$

$\frac{dL}{da_3} \cdot \frac{da_3}{dz_3} = a_3 - y$

$\frac{dz_3}{dw_3} = a_2$

Since $z_3 = w_3 a_2 + b_3$

Finding dL/dW_2

$$\Rightarrow \frac{dL}{dW_2} = (a_3 - y) \cdot \frac{dz_3}{da_2} \cdot \frac{da_2}{dz_2} \cdot \frac{dz_2}{dW_2}$$

If $z_3 = w_3 \cdot a_2 + b_3$

Then $\frac{dz_3}{da_2} = w_3$

if $a_2 = g(z_2)$

then $\frac{da_2}{d(z_2)} = g'(z_2)$

If $z_2 = w_2 \cdot a_1 + b_2$

then $\frac{dz_2}{dw_2} = a_1$

Finding dL/dW_2

$$\Rightarrow \boxed{\frac{\partial L}{\partial w_2} = (a_3 - y) \cdot w_3 \cdot g'(z_2) \cdot a_1}$$

Re-arranging for vectorization,

$$\boxed{\frac{\partial L}{\partial w_2} = w_3^T \cdot g'(z_2) \cdot (a_3 - y) \cdot a_1^T}$$

$$\frac{\partial L}{\partial b_2} = w_3^T \cdot g'(z_2) \cdot (a_3 - y)$$

Finding dL/dW_1

Use the chain rule!

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$$

$$\frac{\partial L}{\partial w_1} = (a_3 - y) \cdot w_3 \cdot g'(z_2) \cdot w_2 \cdot g'(z_1) \cdot x$$

$$\frac{\partial L}{\partial b_1} = (a_3 - y) \cdot w_3 \cdot g'(z_2) \cdot w_2 \cdot g'(z_1)$$

NumPy Demo

Regularization

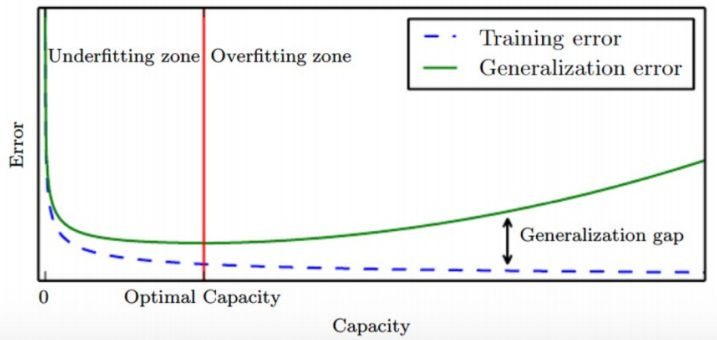
What is regularization?

Generalization error

- Performance on inputs not previously seen
  - Also called as *Test error*

Regularization is:

- "any modification to a learning algorithm to reduce its *generalization error* but not its *training error*"
- Reduce generalization error even at the expense of increasing training error
  - E.g., Limiting model capacity is a regularization method



Goals of Regularization

Some goals of regularization

1. Encode prior knowledge
2. Express preference for simpler model
3. Need to make underdetermined problem determined

Why regularization?

Generalization

- Prevent over-fitting

Occam's razor

Bayesian point of view

- Regularization corresponds to prior distributions on model parameters

Limiting Number of Neurons

No. of input/output units determined by dimensions

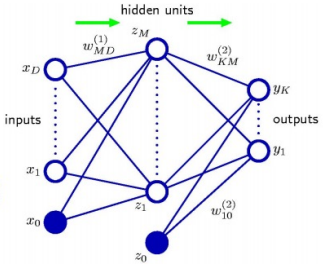
Number of hidden units M is a free parameter

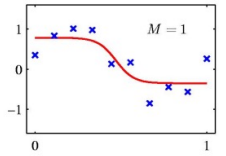
- Adjusted to get best predictive performance

Possible approach is to get maximum likelihood estimate of M for balance between under-fitting and over-fitting

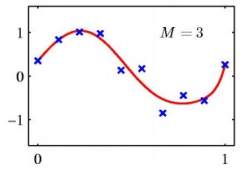
Limiting Number of Neurons

Sinusoidal Regression Prob

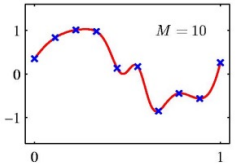




Two layer network trained on 10 data points



$M = 1, 3$ and 10 hidden units



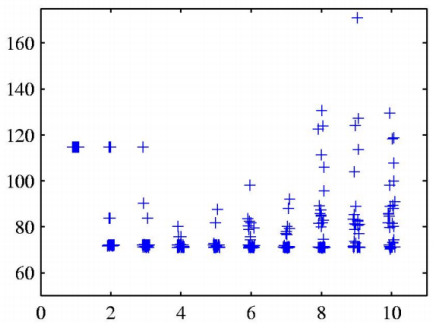
Minimizing sum-of-squared error function
Using conjugate gradient descent

Generalization error is not a simple function of M due to presence of local minima in error function

Using Validation Set to fix no. of hidden units

Plot a graph choosing random starts and different numbers of hidden units M and choose the specific solution having smallest generalization error

Sum of squares *Test* error



No. of hidden units, M

- 30 random starts for each M
  30 points in each column of graph
- Overall best *validation* set performance happened at $M=8$

There are other ways to control the complexity of a neural network in order to avoid over-fitting

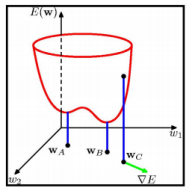
Alternative approach is to choose a relatively large value of M and then control complexity by adding a regularization term

Parameter Norm Penalty

Generalization error not a simple function of M

- Due to presence of local minima
- Need to control capacity to avoid over-fitting
  - Alternatively choose large M and control complexity by addition of regularization term

Simplest regularizer is *weight decay*



$$\tilde{E}(\boldsymbol{w}) = E(\boldsymbol{w}) + \frac{\lambda}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}$$

- Effective model complexity determined by choice of λ
- Regularizer is equivalent to a Gaussian prior over $\boldsymbol{w}$

In a 2-layer neural network

$$J_{regularized} = \underbrace{-\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1-a^{[L](i)}\right)\right)}_{\text{cross-entropy cost}} + \underbrace{\frac{1}{m}\frac{\lambda}{2}\sum_{l}\sum_{k}\sum_{j}W_{k,j}^{[l]2}}_{\text{L2 regularization cost}}$$

Weight Decay

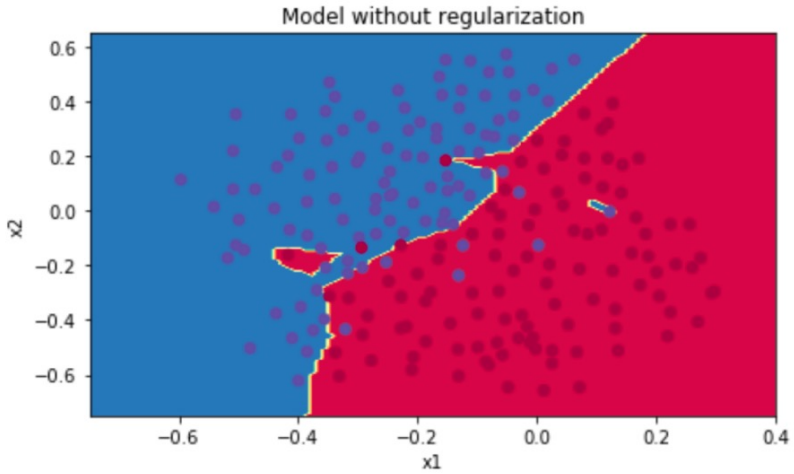
The name weight decay is due to the following

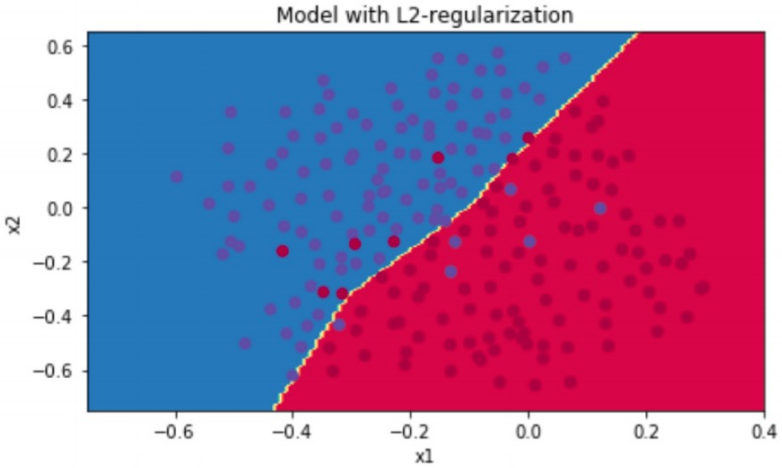
$$w_{t+1} = w_t - \alpha \nabla_w J - \lambda w_t$$

To prevent overfitting, every time we update a weight $\boldsymbol{w}$ with the gradient ∇J in respect to $\boldsymbol{w}$, we also subtract from it $\boldsymbol{\lambda w}$.

This gives the weights a tendency to decay towards zero, hence the name.

Effect of Regularization





$$J_{regularized} = \underbrace{-\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1-a^{[L](i)}\right)\right)}_{\text{cross-entropy cost}} + \underbrace{\frac{1}{m}\frac{\lambda}{2}\sum_{l}\sum_{k}\sum_{j}W_{k,j}^{[l]2}}_{\text{L2 regularization cost}}$$

https://medium.com/@shay.palachy/understanding-the-scaling-of-l%C2%B2-regularization-in-the-context-of-neural-networks-e3d25f8b50db?fbclid=IwAR17rKfnM2JnkWBOZFvQg6ftl0k52diyEra_66UCwI18Gtf7nFd6l3l4VMY

Deep Neural Networks

What is Deep Learning?

1. Computational models composed of multiple processing layers
   - To learn representations of data with multiple levels of abstraction
2. Dramatically improved state-of-the-art in:
   - Speech recognition, Visual object recognition, Object detection
   - Other domains: Drug discovery, Genomics
3. Discovers intricate structure in large data sets
   - Using backpropagation to change parameters
   - Compute representation in each layer from previous layer
4. Deep convolutional nets: image proc, video, speech
5. Recurrent nets: sequential data, e,g., text, speech

Limitations of Conventional ML

- Limited in ability to process natural data in raw form
- Pattern Recognition and Machine Learning systems require careful engineering and domain expertise to transform raw data, e.g., pixel values, into a feature vector for a classifier

Automatic Representation Learning

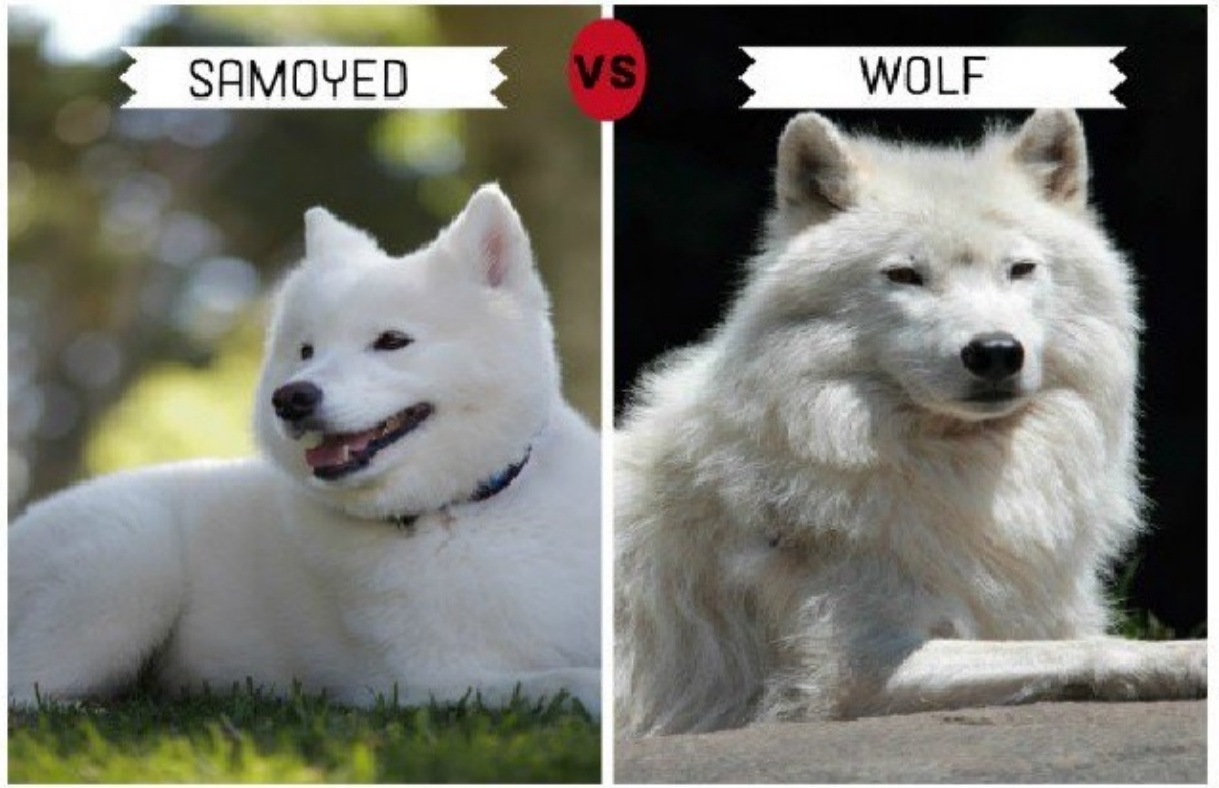
- Methods that allow a machine to be fed with raw data to automatically discover representations needed for detection or classification
- Deep Learning methods are Representation Learning Methods
- Use multiple levels of representation
  - Composing simple but non-linear modules that transform representation at one level (starting with raw input) into a representation at a higher slightly more abstract level
  - Complex functions can be learned
  - Higher layers of representation amplify aspects of input important for discrimination and suppress irrelevant variations

Example: Images

- Input is an array of pixel values
  - First stage is presence or absence of edges at particular locations and orientations of image
  - Second layer detects motifs by spotting particular arrangements of edges, regardless of small variations in edge positions
  - Third layer assembles motifs into larger combinations that corresponds to parts of familiar objects
  - Subsequent layers would detect objects as combinations of these parts
- Key aspect of deep learning:
  - These layers of features are not designed by human engineers
  - They are learned from data using a general purpose learning procedure

Deep versus Shallow Classifiers

- Linear classifiers can only carve the input space into very simple regions
- Image and speech recognition require input-output function to be insensitive to irrelevant variations of the input,
  - e.g., position, orientation and illumination of an object
  - Variations in pitch or accent of speech
  - While being sensitive to minute variations, e.g., white wolf and breed of wolf-like white dog called Samoyed
  - At pixel level two Samoyeds in different positions may be very different, whereas a Samoyed and a wolf in the same position and background may be very similar

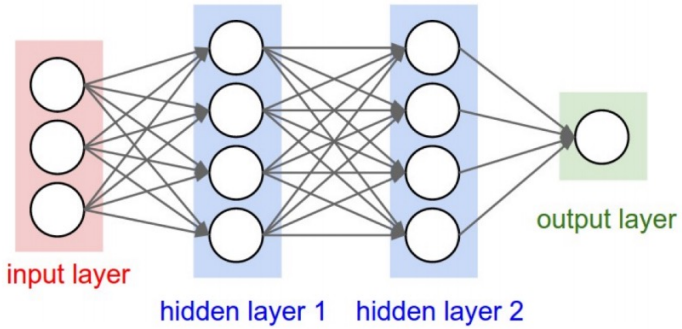


Selectivity-Invariance dilemma

- Shallow classifiers need a good feature extractor
- One that produces representations that are:
  - selective to aspects of image important for discrimination
  - but invariant to irrelevant aspects such as pose of the animal
- Generic features (e.g., Gaussian kernel) do not generalize well far from training examples
- Hand-designing good feature extractors requires engineering skill and domain expertise
- Deep learning learns features automatically

DL Architectures

- Multilayer stack of simple modules
- All modules (or most) subject to:
  - Learning
  - Non-linear input-output mappings
- Each module transforms input to improve both selectivity and invariance of the representation
- With depth of 5 to 20 layers can implement extremely intricate functions of input
  - Sensitive to minute details
    - Distinguish Samoyeds from white wolves
- Insensitive to irrelevant variations
  - Background, pose, lighting, surrounding objects



Convolutional Neural Networks

Key Ideas

- Take advantage of properties of natural signals
  - Local connections
  - Shared weights
  - Pooling
  - Use of many layers

Comparison with Regular NNs

- Regular, Feed-forward NNs:
  - Need substantial number of training samples
  - Slow learning (convergence times)
  - Inadequate parameter selection techniques that lead to poor minima
- **Solution?**

Comparison with Regular NNs

- Regular, Feed-forward NNs:
  - Need substantial number of training samples
  - Slow learning (convergence times)
  - Inadequate parameter selection techniques that lead to poor minima
- **Solution?**
- **Exploitation of Local Properties!**
- Network should exhibit invariance to translation, scaling and elastic deformations
  - A large training set can take care of this
- It ignores a key property of images
  - Nearby pixels are more strongly correlated than distant ones
  - Modern computer vision approaches exploit this property
- Information can be merged at later stages to get higher order features and about whole image

Basic Mechanisms in CNNs

- Three Mechanisms of Convolutional Neural Networks:
  - Convolution Operation
  - Local Receptive Fields
  - Subsampling
  - Weight (Parameter) Sharing