Architectural Elements of Neural Networks

Neural Networks in Context of Classification

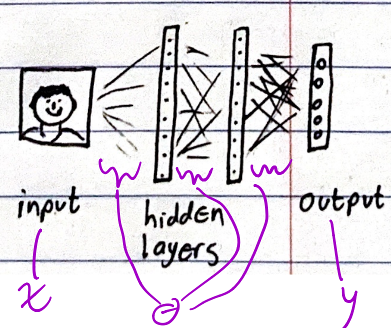
Problem: Given an image of an object, determine what category that object is (eg cat, dog)

A neural network is a function that maps an image to a probability distribution over a known set of classes/categories. The prediction for a given image is category w/ highest probability.

A neural network is built from components:

Linear	Nonlinear
Fully connected	Activation functions
Convolution	Batch Normalization
Linear upsampling/downsampling	Max pooling
	Softmax

Visually:



Parameters are learned from data.

A neural network is a parameterized function from input to output.

$y = f_\theta(x)$

$y = f(x;\theta)$

θ – parameters
x – input
y – output

$x \in \mathbb{R}^k$
$y \in \mathbb{R}^n$

θ – params you optimize to fit data
eg weights, biases

It consist of a series of linear and nonlinear layers

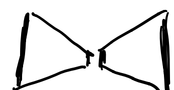
$$y = \sigma\left[W_2\left[\sigma(W_1 x + b_1)\right] + b_2\right]$$

activation function (σ), weights (W), bias (b)

$\sigma(x) = \mathrm{relu}(x) = \max(x, 0)$

Output could be of lower or higher dimension than input.

In a VAE, the decoder



E D

Examples:

MNIST classifier
$\mathbb{R}^{32\times32} \to \mathbb{R}^{10}$

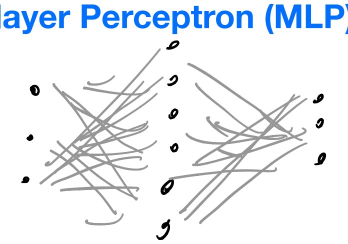
regressor for house price
$\mathbb{R}^k \to \mathbb{R}^1$

NLP generative model
- generate follow up text to input

GANs $\mathbb{R}^{100} \to \mathbb{R}^{128\times128}$
img

Linear layers and bias term

Multi-layer Perceptron (MLP) – Fully-connected net



x $\sigma(W_1x+b_1)$ $W_2(\sigma(W_1x+b_1))+b_2$

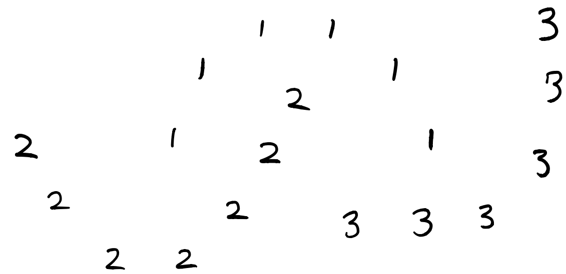
Exercise: The default resolution on some cell phone cameras is 4032 x 3024. Suppose you had a MLP that maps such a photograph to a hidden layer with 1000 neurons and then maps that to an output layer with 10 neurons. How many parameters (weights and biases) would there be? Assume there are no parameters in the activation functions.

n_0 – input dim
n_1 – width of hidden layer
n_2 – output dim

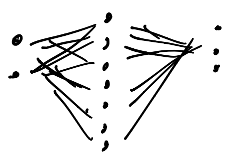
input pixels=12.2Million $n_0 = 36.6M$

$n_1[n_0+1] + n_2[n_1+1] = 36B +$

Probability model corresponding to a neural network used for classification



$f: \mathbb{R}^2 \to \mathbb{R}^3$



— output will be logits

Taking softmax, you get a probab. dist over $\{1,2,3\}$

Logistic Function and Softmax

$$\text{logistic}(z) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}} \quad \text{for } z \in \mathbb{R}$$

Softmax: $\mathbb{R}^N \to \mathbb{R}^N$

$$\{z_i\}_{i=1 \dots N} \mapsto \left\{ \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \right\}_{j=1 \dots N}$$

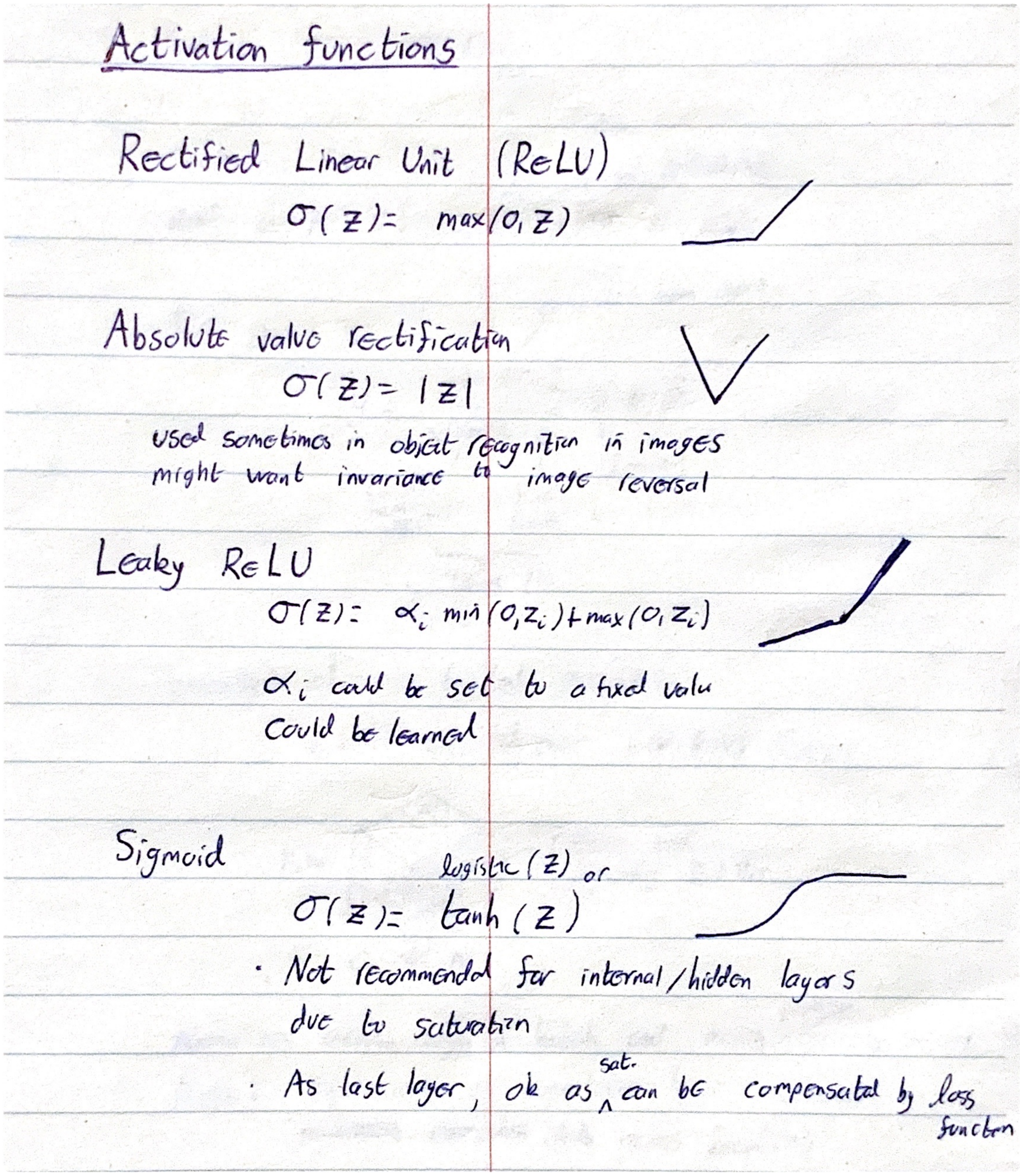
Output of Softmax is a prob. dist.
(could call it softargmax)

Note: If a net computes logits, optimization is not expected to converge.
eg linearly separable binary classification w/ logistic regression

Application: Even after 100% training accuracy is achieved, optimization continues (and test error does too)

Should I build a network that outputs logits or probabilities?

Generally speaking, prefer logits, as probability output may yield numerical issues



Activation functions

Rectified Linear Unit (ReLU)

$\sigma(z) = \max(0, z)$

Absolute value rectification

$\sigma(z) = |z|$

used sometimes in object recognition in images
might want invariance to image reversal

Leaky ReLU

$\sigma(z) = \alpha_i \min(0, z_i) + \max(0, z_i)$

α_i could be set to a fixed value
could be learned

Sigmoid

$\sigma(z) =$ logistic(z) or $\tanh(z)$

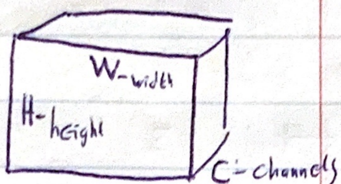
- Not recommended for internal/hidden layers due to saturation
- As last layer, ok as sat. can be compensated by loss function

How do you select an activation function?

Reference: He et al., 2015 “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”

Channels and Batching for images

An image is a 3-dim matrix "tensor"



Grayscale - C = 1

RGB - C = 3

At internal layers of a NN C could take other values

A minibatch of images is a collection of N images

a 4-dim tensor

Pytorch [N, C, H, W]

TensorFlow [N, H, W, C]

Convolutional Layers

Mathematically: $S = X * W$

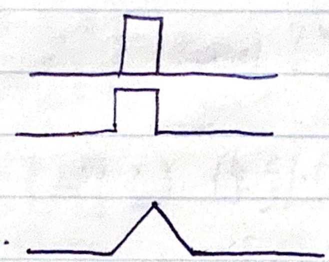
$$S(t) = \int X(a)W(t-a)\,da$$ "convolution"

inner product of X with shifted and flipped W.

eg W =

X =

X*W =



ML terminology:

X – input

W – kernel, filter

S – feature map, activation map

Discrete 1-d convolution:

$$S(i) = X * W(i) = \sum_m X(m)W(i-m)$$

Discrete 2-d convolution

$$S(i,j) = X * W(i,j) = \sum_m \sum_n X(m,n)W(i-m, j-n)$$

$$= \sum_m \sum_n X(i-m, j-n)W(m,n)$$

Instead of convolution (strictly speaking) ML libraries implement cross-correlation (dont flip the kernel)

$$S(i,j) = X * W(i,j) = \sum_m \sum_n X(i+m, j+n) W(m,n)$$

Visually: Input $\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix}$ Kernel $\begin{pmatrix} w & x \\ y & z \end{pmatrix}$

Convolution
$$\begin{pmatrix} \begin{pmatrix} a & b \\ e & f \end{pmatrix} \cdot \begin{pmatrix} w & x \\ y & z \end{pmatrix} & \begin{pmatrix} b & c \\ f & g \end{pmatrix} \cdot \begin{pmatrix} w & x \\ y & z \end{pmatrix} & \begin{pmatrix} c & d \\ g & h \end{pmatrix} \cdot \begin{pmatrix} w & x \\ y & z \end{pmatrix} \\ \begin{pmatrix} e & f \\ i & j \end{pmatrix} \cdot \begin{pmatrix} w & x \\ y & z \end{pmatrix} & \begin{pmatrix} f & g \\ j & k \end{pmatrix} \cdot \begin{pmatrix} w & x \\ y & z \end{pmatrix} & \begin{pmatrix} g & h \\ k & l \end{pmatrix} \cdot \begin{pmatrix} w & x \\ y & z \end{pmatrix} \end{pmatrix}$$

w/ $\begin{pmatrix} a & b \\ e & f \end{pmatrix} \cdot \begin{pmatrix} w & x \\ y & z \end{pmatrix} = aw + bx + ey + fz$

In 1-d, convolution corresponds to multiplication by a Toeplitz matrix

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} * \begin{pmatrix} w \\ x \end{pmatrix} = \begin{pmatrix} aw + bx \\ bw + cx \\ cw + dx \end{pmatrix}$$

$$= \begin{pmatrix} w & x & & \\ & w & x & \\ & & w & x \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$$

Toeplitz matrix $\begin{pmatrix} a & b & c & d \\ e & a & b & c \\ f & e & a & b \\ g & f & e & a \end{pmatrix}$

What about boundaries?

Choices include:

- only include windows entirely contained in input (dim decreases)
- pad w/ zeros to keep image same size as feature map
- circular
- etc

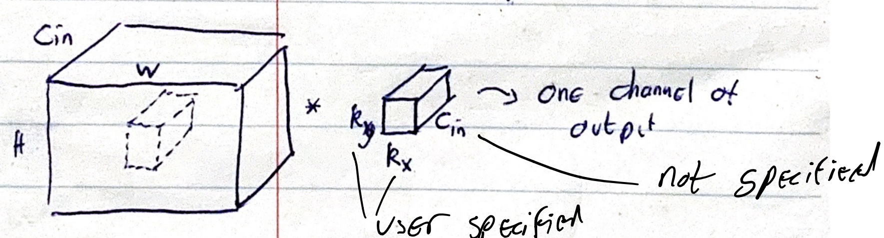
Conv2d operation in Pytorch

$$[N, C_{in}, H, W] \rightarrow [N, C_{out}, H_{out}, W_{out}]$$

Equal (N): images treated in parallel

H_{out}, W_{out}: these can differ from H, W due to padding, stride, etc

Each ~~kernel~~ filter is a 3-tensor (has



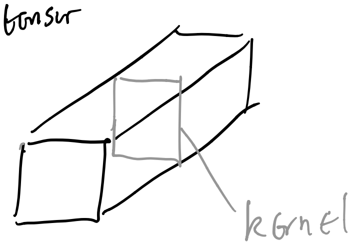
If filter has size $k_x \times k_y$, and no padding, how many parameters are in this convolution? (ignore bias terms)
What is output size

Output $N \times C_{out} \times (W - k_x + 1) \times (H - k_y + 1)$ — assuming stride 1 padding 0

params $C_{out} \cdot k_x \cdot k_y \cdot C_{in}$

2d tensor 3d tensor

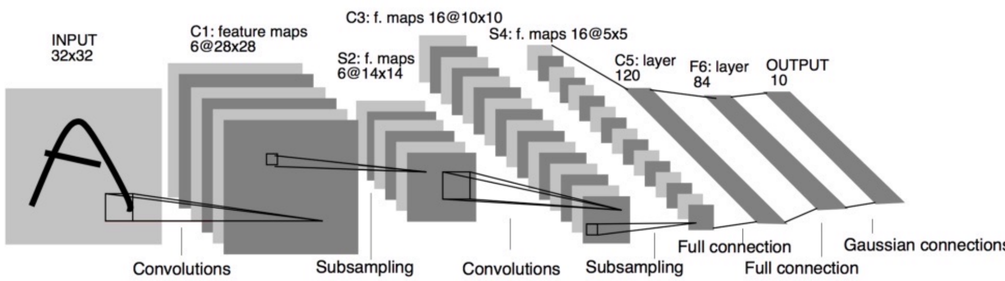
Terms: Kernel, Filter



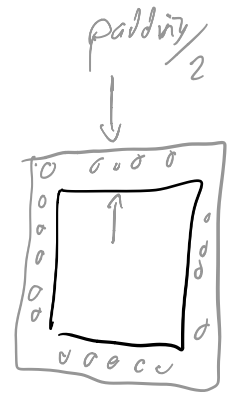
How to choose kernel size?

Terms: Channels, Activation Maps, Feature Map

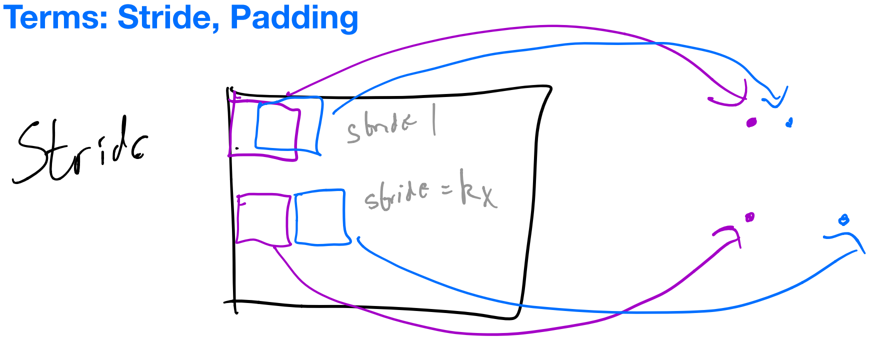
Architecture of LeNet

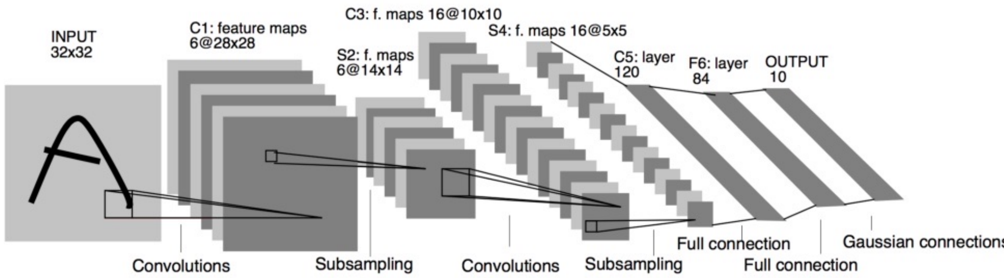


Each output channel has its own learned filter

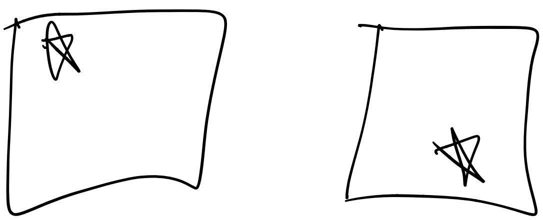


Terms: Stride, Padding





Principles of Convolution layers: Locality, translation invariance



Terms: Receptive Field

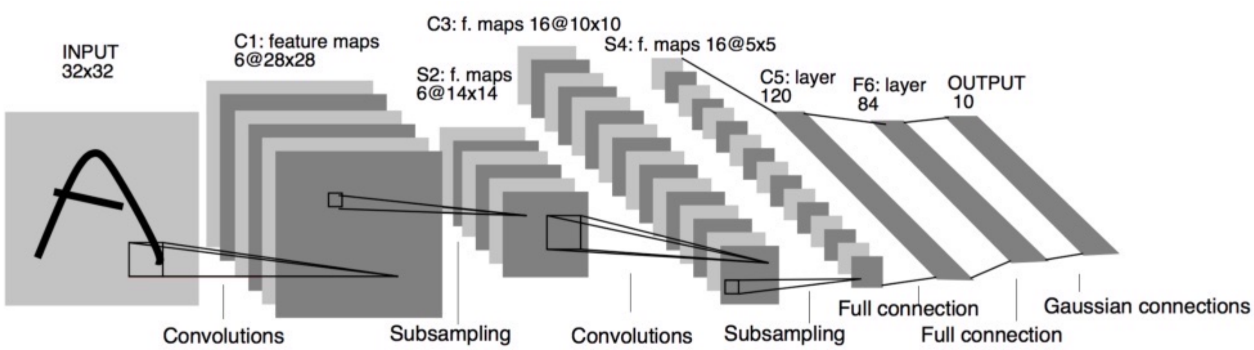
For a given output neuron, the receptive field is the set of input neurons that affect it

How many parameters in a 2d convolution layer with bias have?

What would a corresponding fully connected layer have?

```python
class LeNet5(nn.Module):

    def __init__(self, n_classes):
        super(LeNet5, self).__init__()

        self.feature_extractor = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5, stride=1),
            nn.Tanh(),
            nn.AvgPool2d(kernel_size=2),
            nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5, stride=1),
            nn.Tanh(),
            nn.AvgPool2d(kernel_size=2),
            nn.Conv2d(in_channels=16, out_channels=120, kernel_size=5, stride=1),
            nn.Tanh()
        )

        self.classifier = nn.Sequential(
            nn.Linear(in_features=120, out_features=84),
            nn.Tanh(),
            nn.Linear(in_features=84, out_features=n_classes),
        )


    def forward(self, x):
        x = self.feature_extractor(x)
        x = torch.flatten(x, 1)
        logits = self.classifier(x)
        probs = F.softmax(logits, dim=1)
        return logits, probs
```

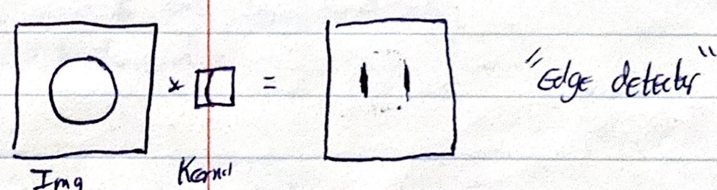

Architecture of LeNet -



Strength's of Convolutional Layers

- Fewer params than FC layers
  - cheaper
  - Easier to optimize
- Equivariance to translation
  - features of image in top left should be treated same as same in bottom right

Visually: Conv layers are "feature detectors"



Batch Normalization

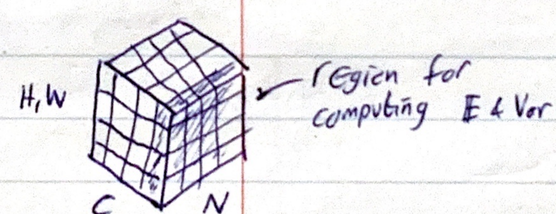
Technique for improving speed, reliability, and perf of optimization of NNs.

For input x

$$y = \frac{x - \mathbb{E}[x]}{\sqrt{\mathrm{Var}(x) + \epsilon}} \cdot \gamma + \beta$$

- $\mathbb{E}[x]$: computed from input
- $\mathrm{Var}(x)$: computed from input
- ϵ: fixed, small
- γ, β: learned
- $\sqrt{\mathrm{Var}(x) + \epsilon}$ (computed from input, and ϵ fixed, small): whitens the input

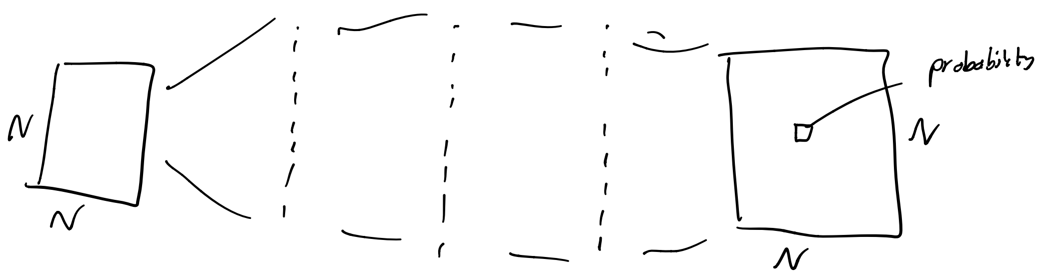
Typically: channels treated separately
$\mathbb{E}$ & Var computed over (N, H, W) slices



Normalizes across images in batch and across ~~pixels~~ positions in img

Note: Equivariance of convolution layer motivates computing stats across locations

Problem: You have an N x N photograph. You are training a network that takes the image and tries to classify each pixel as being the midpoint or not the midpoint of the image. You train the net with a collection of N x N images, and you identify the midpoint of each image as the supervision signal



Will an MLP architecture work?

Will a pure CNN work?

Why does BN work?

Initial explanation

- Internal Covariate shift
  - "the change in distribution of network activations due to the change in net params during training"
- Idea: Modifications in an early layer of net could cause a large change downstream. BN would decouple effects more.
- Explanation is disputed

Other explanation

- Smoothing objective