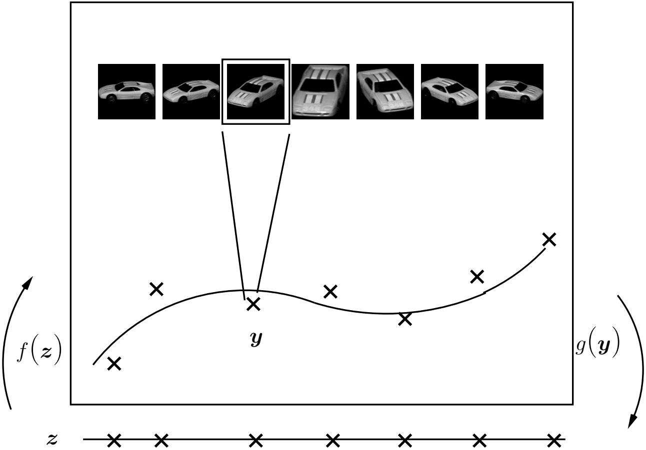
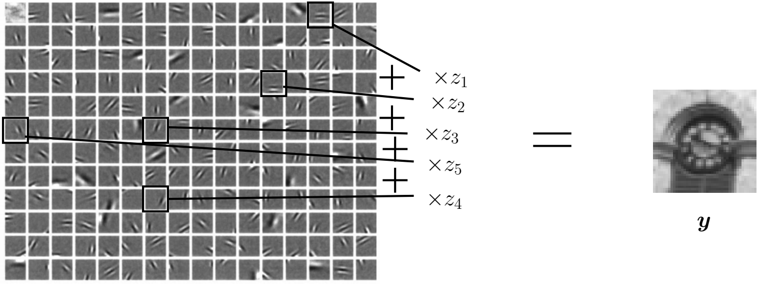
# Outline

# Outline

# Modeling data with latent variables



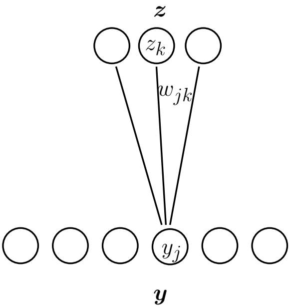
# Sparse Coding Review




- Model an image-patch as the superposition of *basis functions*, or "*filters*":

$$\boldsymbol{y} = \sum_k W_{\cdot k} z_k, \quad y_j = \sum_k w_{jk} z_k$$

# Feature Learning Graphical Model

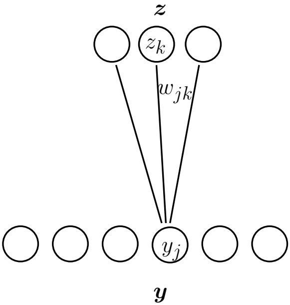



$$y_j^\alpha = \sum_k w_{jk} z_k^\alpha$$

## Synthesis model

- Parameters $w_{jk}$ connect pixels $y_j$ with code components $z_k$
- Dimensionality of $\boldsymbol{z}$ can be smaller, larger, or same as $\boldsymbol{y}$
- When the dimensionality is the same or larger, then $\boldsymbol{z}$ must be *constrained*, eg. by forcing it to be *sparse*.

# Feature Learning Graphical Model

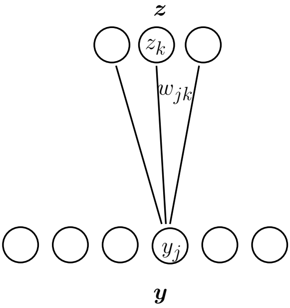



$$y_j^\alpha = \sum_k w_{jk} z_k^\alpha$$

## Learning

- Given data-set $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^N$, adapt parameters $W$, inferring $\boldsymbol{z}^1, \ldots, \boldsymbol{z}^N$ along the way.
- Unsupervised learning.

# Feature Learning Graphical Model




$$y_j^\alpha = \sum_k w_{jk} z_k^\alpha$$

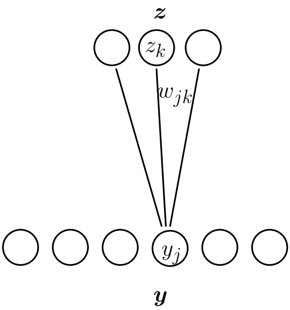
## Learning

- For example

$$\min_{W, \boldsymbol{z}^1, \ldots, \boldsymbol{z}^N} \frac{1}{N} \sum_\alpha \left( \|\boldsymbol{y}^\alpha - \sum_k z_k^\alpha W_{.k}\|^2 + \lambda \sum_k |z_k^\alpha| \right)$$

- Alternating between $W$ and all $\boldsymbol{z}^\alpha$.

# Feature Learning Graphical Model

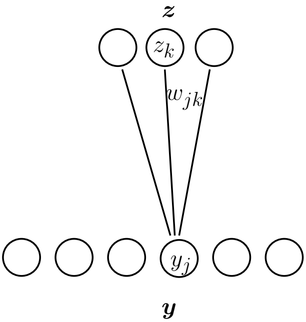



$$y_j^\alpha = \sum_k w_{jk} z_k^\alpha$$

## Inference ("Analysis")

- Given new image $\boldsymbol{y}$, compute $\boldsymbol{z}$.
- This is how we do recognition.

# Feature learning models

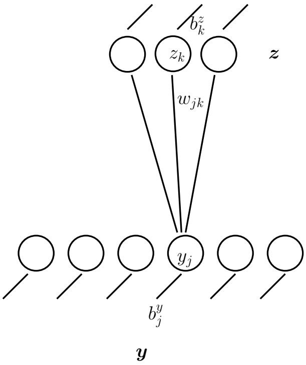



$$y_j = \sum_k w_{jk} z_k$$

## Many Variants

- Probabilistic *vs.* Non-probabilistic;
- Directed *vs.* undirected;
- Mixture *vs.*
- factorial *vs.* non-symmetric

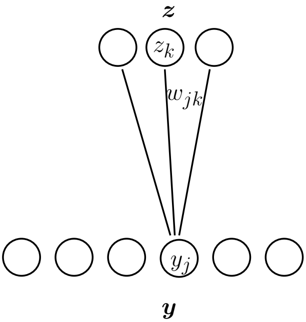
# Feature learning models




$$y_j = \sum_k w_{jk} z_k + b_j^y$$

- In practice: add bias terms.
- But we drop these for now to avoid clutter.

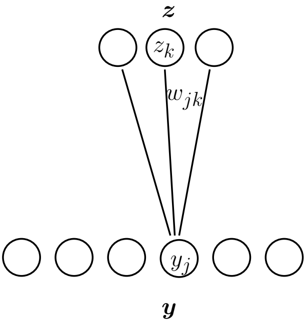
# Feature learning models




$$y_j = \sum_k w_{jk} z_k$$

- Some sparse coding models make inference easy:

# Feature learning models



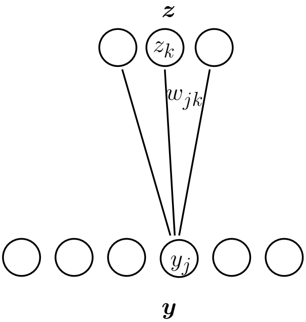

$$p(y_j|\boldsymbol{z}) = \text{sigmoid}\Big(\sum_k w_{jk} z_k\Big)$$

$$p(z_k|\boldsymbol{y}) = \text{sigmoid}\Big(\sum_j w_{jk} y_j\Big)$$

## Restricted Boltzmann machine (RBM)

- $p(\boldsymbol{y}, \boldsymbol{z}) = \frac{1}{Z} \exp\left(\sum_{jk} w_{jk} y_j z_k\right)$
- Contrastive Divergence learning (Hebbian-style learning)
- Inference: $p(z_k|\boldsymbol{y}) = \text{sigmoid}\left(\sum_j w_{jk} y_j\right)$
- Further advantage: Allows for stacking (deep learning).

# Feature learning models

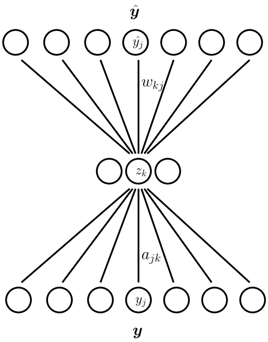



$$p(y_j|\boldsymbol{z}) = \text{sigmoid}\Big(\sum_k w_{jk} z_k\Big)$$

$$p(z_k|\boldsymbol{y}) = \text{sigmoid}\Big(\sum_j w_{jk} y_j\Big)$$

## Restricted Boltzmann machine (RBM)

- $p(\boldsymbol{y}, \boldsymbol{z}) = \frac{1}{Z} \exp\left(\sum_{jk} w_{jk} y_j z_k\right)$
- Contrastive Divergence learning (Hebbian-style learning)
- Inference: $p(z_k|\boldsymbol{y}) = \text{sigmoid}\left(\sum_j w_{jk} y_j\right)$
- Further advantage: Allows for stacking (deep learning).



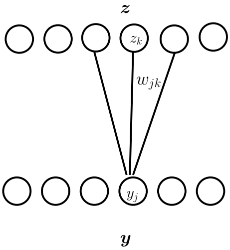

$$z_k = \text{sigmoid}\Big(\sum_j a_{jk} y_j\Big)$$

$$y_j = \sum_k w_{jk} z_k$$

## Autoencoder

- Add **inference parameters** $A$, and set $\boldsymbol{z} = \text{sigmoid}\,(A\boldsymbol{y})$
- Learning: $\min_{W,A} \sum_\alpha \|\boldsymbol{y}^\alpha - W\text{sigmoid}\,(A\boldsymbol{y}^\alpha)\,\|^2$
- Add a sparsity penalty for $\boldsymbol{z}$, or *corrupt inputs during training* (Vincent et al., 2008)

# Feature learning models




$$y_j = \sum_k w_{jk} z_k$$

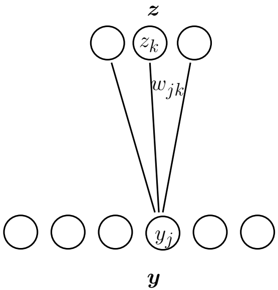
## Independent Components Analysis (ICA)

- Learning: Make responses sparse, while keeping filters sensible

$$\min_W \|W^{\mathrm{T}} \boldsymbol{y}\|_1$$
$$\text{s.t.} \quad W^{\mathrm{T}} W = I$$

# Feature learning summary

$$\boldsymbol{z}(\boldsymbol{y}) = \boldsymbol{W}^{\mathrm{T}}\boldsymbol{y}$$
$$\boldsymbol{y}(\boldsymbol{z}) = \boldsymbol{W}\boldsymbol{z}$$

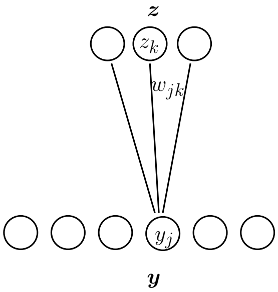



## Linear inference summary

- A lot of methods define inference through **linear dependencies** between $y$ and $z$.
- PCA, ICA, Restricted Boltzmann Machine, Autoencoder, Mixture of Gaussians, KMeans, ...

# Feature learning summary

$$\boldsymbol{z}(\boldsymbol{y}) = \boldsymbol{W}^{\mathrm{T}}\boldsymbol{y}$$
$$\boldsymbol{y}(\boldsymbol{z}) = \boldsymbol{W}\boldsymbol{z}$$

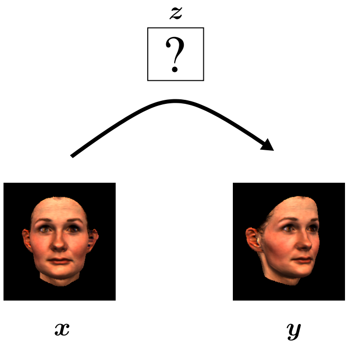



## Feature learning summary

- Almost all methods yield Gabor filters when trained on natural images.
- Almost all based on the same rationale:
- Tease apart the hidden causes of variability in the data.

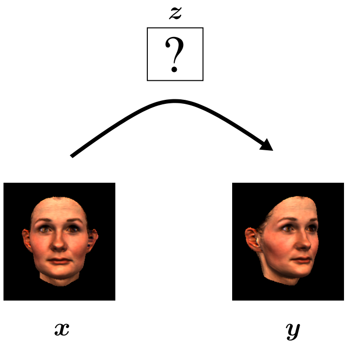
# Outline

# Sparse coding of images pairs?



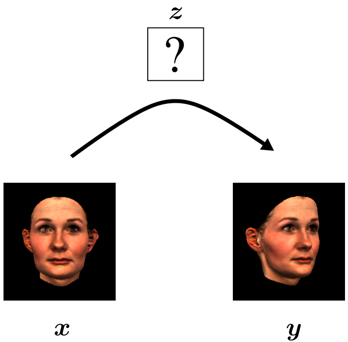

- How to extend sparse coding to model relations?
- Sparse coding on the *concatenation*?

# Sparse coding of images pairs?




- How to extend sparse coding to model relations?
- Sparse coding on the *concatenation*?

# Sparse coding of images pairs?



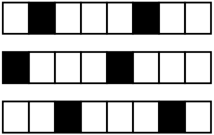

- How to extend sparse coding to model relations?
- Sparse coding on the *concatenation*?

# Sparse coding on the concatenation ?

- A case study: Translations of binary, one-d images.
- Suppose images are random and can change in **one of three ways:**

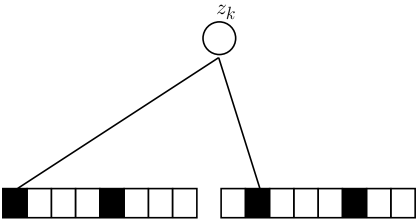
Example Image $\boldsymbol{x}$:

Possible Image $\boldsymbol{y}$:



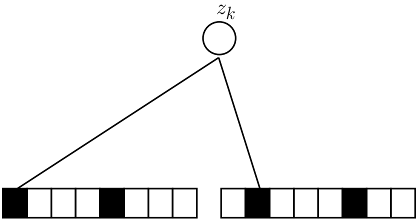
# Sparse coding on the concatenation ?

- A hidden variable that collects evidence for a shift to the right.
- What if the images are random or noisy?
- Can we pool over more than one pixel?



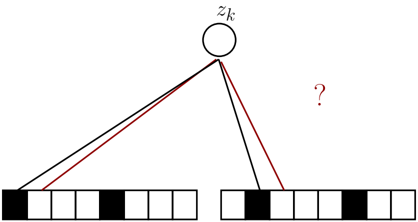
# Sparse coding on the concatenation ?

- A hidden variable that collects evidence for a shift to the right.
- What if the images are random or noisy?
- Can we pool over more than one pixel?



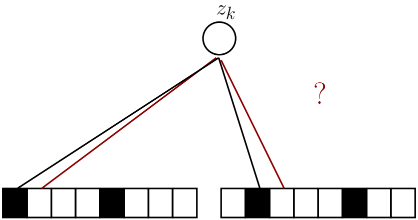
# Sparse coding on the concatenation ?

- A hidden variable that collects evidence for a shift to the right.
- What if the images are random or noisy?
- Can we pool over more than one pixel?



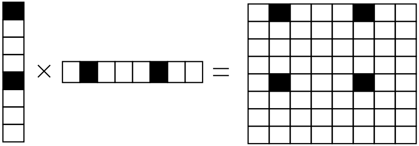
# Sparse coding on the concatenation ?

- Obviously not, because now the hidden unit would get equally happy if it would see the non-shift (second pixel from the left).
- The problem: Hidden variables act like OR-gates, that accumulate evidence.

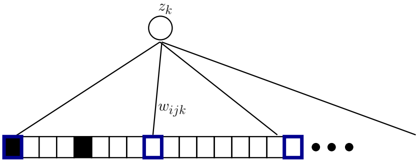


# Cross-products

- Intuitively, it seems, what we need instead are logical ANDs, which can represent *coincidences* (eg. Zetzsche et al., 2003, 2005).
- This amounts to using the outer product $L := \text{outer}(\boldsymbol{x}, \boldsymbol{y})$:

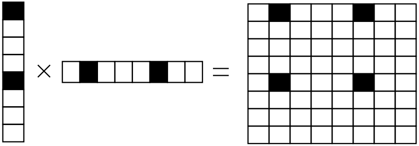


- We can unroll this matrix, and let this be the data:



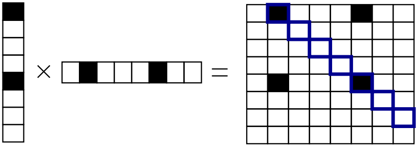
# Cross-products

- In the shift-example, every component $L_{ij}$ of the outer-product matrix will constitute evidence for exactly *one* type of shift.
- Hiddens pool over products of pixels.



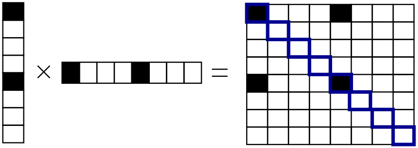
# Cross-products

- In the shift-example, every component $L_{ij}$ of the outer-product matrix will constitute evidence for exactly *one* type of shift.
- Hiddens pool over products of pixels.

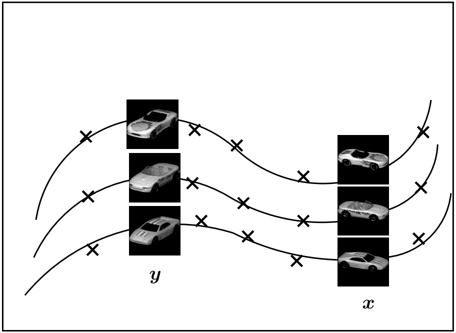


# Cross-products

- In the shift-example, every component $L_{ij}$ of the outer-product matrix will constitute evidence for exactly *one* type of shift.
- Hiddens pool over products of pixels.

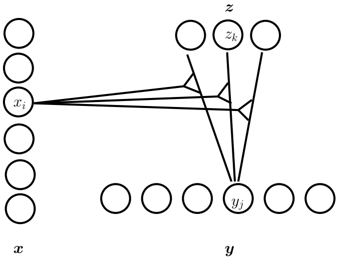


# A family of manifolds




- An different perspective:
- Feature learning reveals the (local) manifold structure in data.
- When $y$ is a transformed version of $x$, we can still think of $y$ as being confined to a manifold, but it will be a *conditional manifold*.
- *Idea:* Learn a model for $y$, but let parameters be *a function of* $x$.

# Three-way interactions

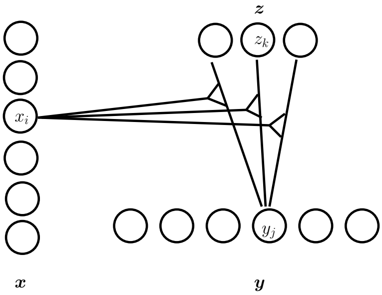



- If we use a linear function, we have $w_{jk}(\boldsymbol{x}) = \sum_i w_{ijk} x_i$.
- Inference turns into:

$$z_k = \sum_j w_{jk} y_j = \sum_j \Big( \sum_i w_{ijk} x_i \Big) y_j = \sum_{ij} w_{ijk} x_i y_j$$

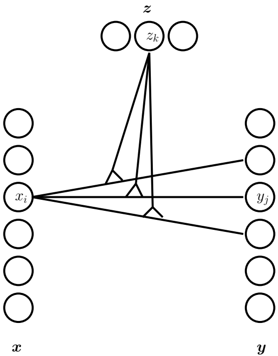
- Hidden units are a **bilinear** function of the two input images.

# Conditional sparse coding




- This is a feature learning model, whose parameters are modulated by inputs.
- So this is a **conditional feature learning** model.
- (Tenenbaum, Freeman; 2000), (Grimes, Rao; 2005), (Ohlshausen; 2007), (Memisevic, Hinton; 2007)

# An alternative visualization

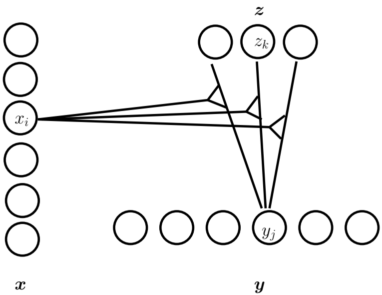



- Each hidden variable can blend in one *slice* $W_{\cdot\cdot k}$ of the parameter tensor.
- Each slice does linear regression in “pixel space”.
- So for binary hiddens, this is a **mixture of** $2^K$ **image warps.**

# Outline

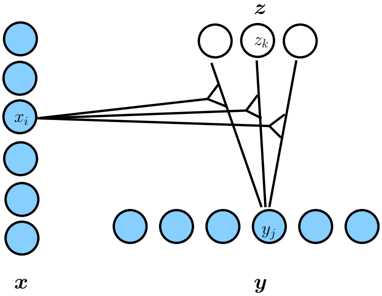
1. **Introduction**
   - Feature Learning
   - Correspondence in Computer Vision
   - Relational feature learning
2. **Learning relational features**
   - Sparse Coding Review
   - Encoding relations
   - Inference
   - Learning
3. **Factorization, eigen-spaces and complex cells**
   - Factorization
   - Eigen-spaces, energy models, complex cells
4. **Applications**
   - Applications
   - Conclusions

# Inference




- Given any two sets of variables, it is easy to infer the third.
- As a result, inference is basically the same as in any standard sparse coding model.
- (Graph is tri-partite, sparse coding bi-partite.)

# Inference

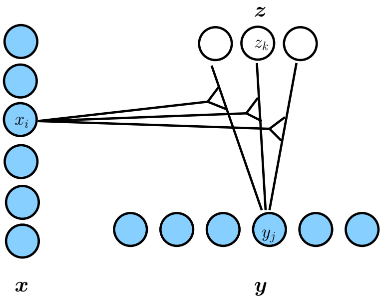



## Inferring $z$ (given $x$ and $y$)

- Infer the transformation $z$ by modulating parameters linearly:

$$z_k = \sum_j w_{jk} y_j = \sum_k \Big(\sum_i w_{ijk} x_i\Big) y_j = \sum_{ij} w_{ijk} x_i y_j$$

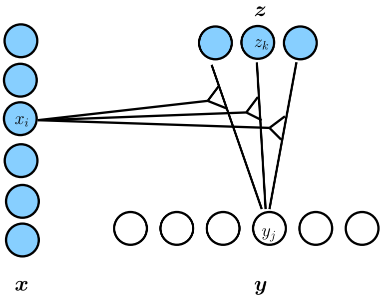
# Inference




**Inferring $z$ (given $x$ and $y$)**

- The meaning of $z$: The *transformation* that takes $x$ to $y$ (or vice versa).

# Inference

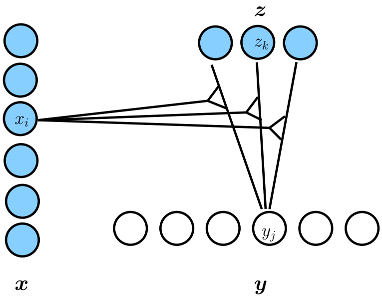



## Inferring $y$ (given $x$ and $z$)

- For $y$, we have

$$y_j = \sum_k w_{jk} z_k = \sum_k \Big(\sum_i w_{ijk} x_i\Big) z_k = \sum_{ik} w_{ijk} x_i z_k$$

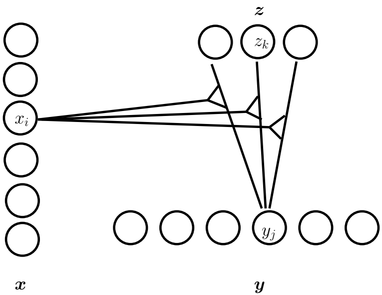
# Inference




**Inferring $y$ (given $x$ and $z$)**

- The meaning of $y$: "$x$ transformed according to $z$".

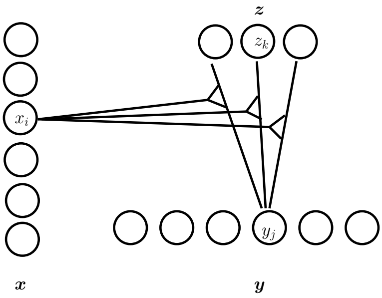
# Inference




- Inference can mean various other things in addition.
- For example, given $x$ and $y$, **how likely are these to come together?**
- More on this type of inference later.

# Outline

# Learning

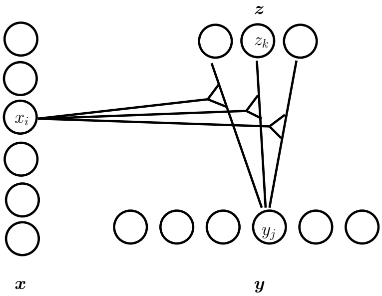



- Training data are now **pairs** $(\boldsymbol{x}^{\alpha}, \boldsymbol{y}^{\alpha})$ – the points we want to relate.
- The parameter-gating relation shows that one way to train this model is:

**Conditional sparse coding**

Predict $\boldsymbol{y}$ from $\boldsymbol{x}$, inferring $\boldsymbol{z}$ along the way as usual.

# Conditional sparse coding




- The cost that data-case $(\boldsymbol{x}^\alpha, \boldsymbol{y}^\alpha)$ contributes is:

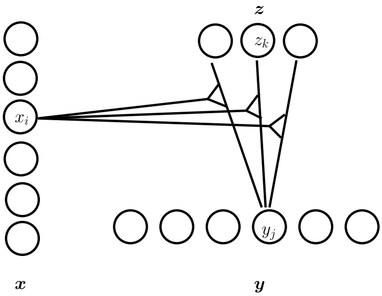
$$\sum_j \left(y_j^\alpha - \sum_{ik} w_{ijk} x_i^\alpha z_k^\alpha\right)^2$$

- Differentiating with respect to $w_{ijk}$ just like before.
- Inference is still *linear* wrt. parameters.

# Conditional sparse coding

- Conditional sparse coding is *predictive coding*:
- We model the next time frame, given the previous one.
- Inference then provides an encoding of the transformation.
- This is often a sensible strategy, but not always as we shall see.

# Example: Gated Boltzmann machine

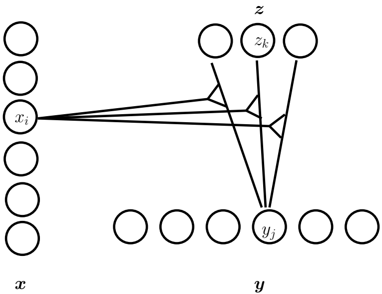



- For a restricted Boltzmann machine, this amounts to changing the energy function into a *three-way energy* (Memisevic, Hinton; 2007):

$$E(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \sum_{ijk} w_{ijk} x_i y_j z_k$$

- Then $p(\boldsymbol{y}, \boldsymbol{z}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \exp\big(E(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})\big)$, $Z(\boldsymbol{x}) = \sum_{\boldsymbol{y},\boldsymbol{z}} \exp\big(E(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})\big)$

# Example: Gated Boltzmann machine

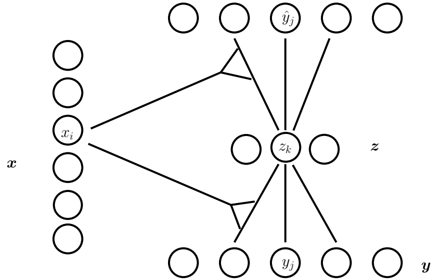



- For a restricted Boltzmann machine, this amounts to changing the energy function into a *three-way energy* (Memisevic, Hinton; 2007):

$$E(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \sum_{ijk} w_{ijk} x_i y_j z_k$$

- Then $p(\boldsymbol{y}, \boldsymbol{z} | \boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \exp\big(E(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})\big)$,
  $Z(\boldsymbol{x}) = \sum_{\boldsymbol{y},\boldsymbol{z}} \exp\big(E(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})\big)$

# Example: Gated auto-encoder

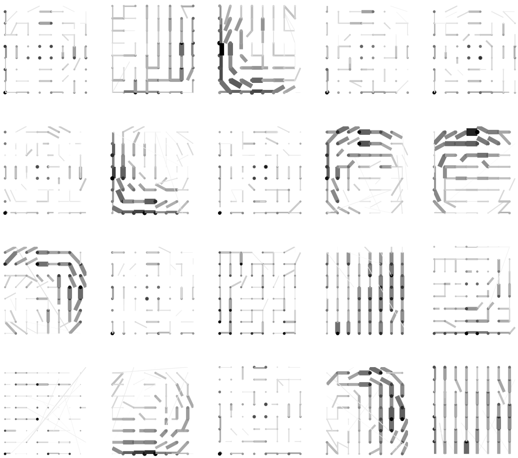



- Similar for autoencoders.
- Both, encoder and decoder weights turn into functions of $x$.
- Learning the same as in a standard auto-encoder modeling $y$.
- The model is still a DAG, so back-prop works *exactly* like in a standard autoencoder.

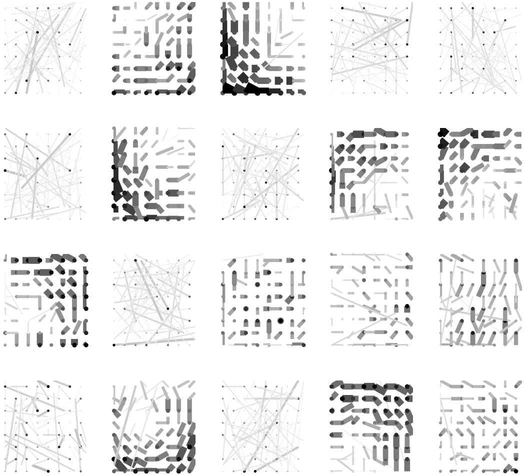
# Other examples

- (Grimes, Rao; 2005): Bi-linear sparse coding
- (Ohlshausen et al.; 2007): Conditional, bi-linear sparse coding
- (Luecke, et al.; 2007): Neurally inspired control unit networks

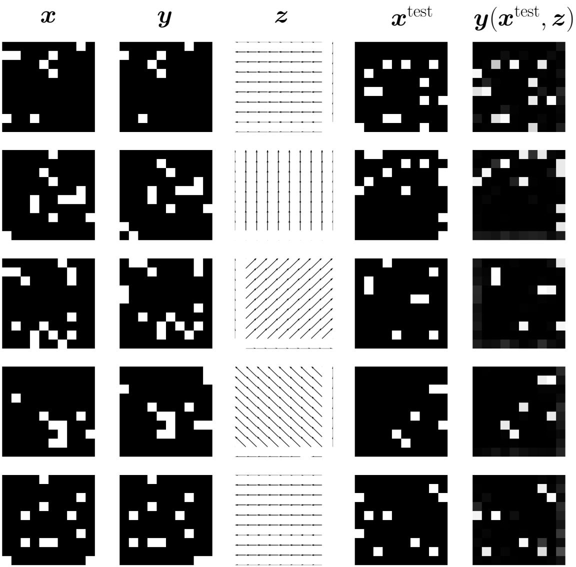
# Toy example: Conditionally trained “Hidden flow-fields”



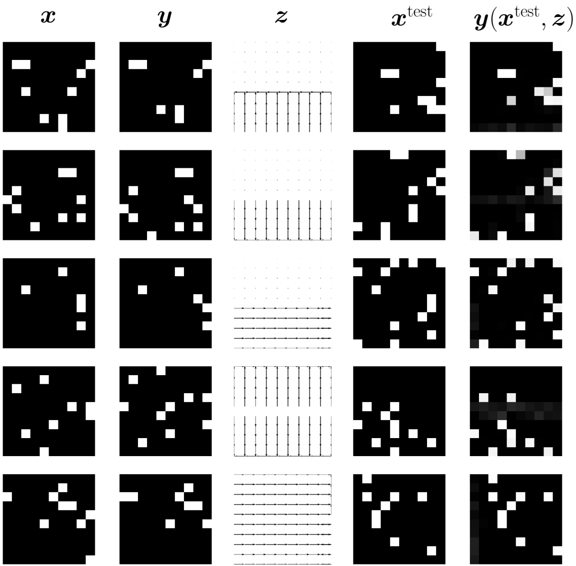
# Toy example: Conditionally trained "Hidden flow-fields", inhibitory connections



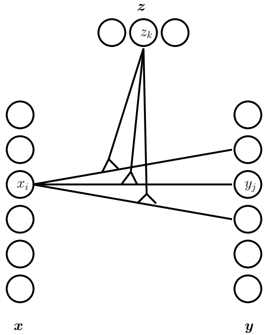
# Toy example: Learning optical flow



# "Combinatorial flowfields"

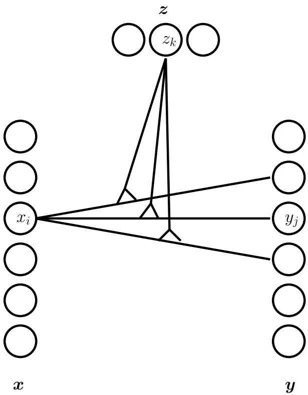


# Joint training




- Conditional training makes it hard to answer questions like:
- "How likely are the given images transforms of one another?"
- To answer questions like these, we require a joint image model, $p(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{z})$, given mapping units.

# Joint training



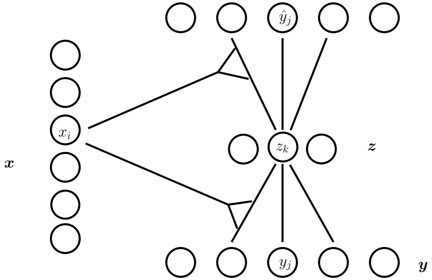

$$E(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \sum_{ijk} w_{ijk} x_i y_j z_k$$

$$p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \frac{1}{Z} \exp\big(E(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})\big)$$

$$Z = \sum_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}} \exp\big(E(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})\big)$$

- (Susskind et al., 2011): Three-way Gibbs sampling in a Gated Boltzmann Machine.
- Can apply this to *matching* tasks (more later).

# Joint training

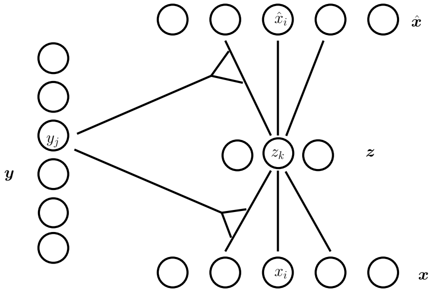



- For the auto-encoder, there is a simple hack:
- Add up two conditional costs:

$$\sum_j \left(y_j^\alpha - \sum_{ik} w_{ijk} x_i^\alpha z_k^\alpha\right)^2 + \sum_i \left(x_i^\alpha - \sum_{jk} w_{ijk} y_j^\alpha z_k^\alpha\right)^2$$

- This forces parameters to be able to transform in both directions.

# Joint training




- For the auto-encoder, there is a simple hack:
- Add up two conditional costs:

$$\sum_j \big(y_j^\alpha - \sum_{ik} w_{ijk} x_i^\alpha z_k^\alpha\big)^2 + \sum_i \big(x_i^\alpha - \sum_{jk} w_{ijk} y_j^\alpha z_k^\alpha\big)^2$$

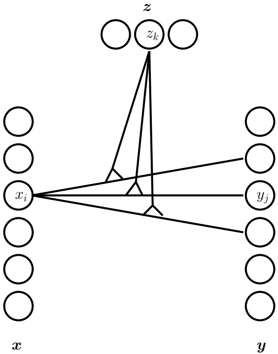
- This forces parameters to be able to transform in both directions.

# Pool over products

## Take-home message

To gather evidence for a transformation,
let each hidden unit pool over products of input-components.

# Some references




- (Hinton; 1981), (v.d. Malsburg; 1981)
- (Grimes, Rao; 2005): Bi-linear sparse coding.
- (Tenenbaum, Freeman; 2000), (Grimes, Rao; 2005), (Ohlshausen; 2007), (Memisevic, Hinton; 2007), (Susskind, et al., 2011)
- (Zetzsche et al.; 2003, 2005)