Consider a transform with a single parameter: \( \tilde{s}(\bar{x}, \xi) \) \( \left[ \tilde{s}(x, 0) = x \right] \)

In the infinite data limit, the sum-of-squares error function is given by:

\[
E = \frac{1}{2} \int \int \int d\bar{x} dt (y(\bar{x}) - t)^2 p(t|\bar{x}) p(\bar{x})
\]

1D output (single output node) for simplicity.

Imagine that each \( \bar{x} \) is perturbed many times: \( \bar{x} \rightarrow \tilde{s}(\bar{x}, \xi) \), where \( \xi \) is drawn from \( p(\xi) \).

Then

\[
\tilde{E} = \frac{1}{2} \int \int \int d\bar{x} dt d\xi (y(\tilde{s}(\bar{x}, \xi)) - t)^2 p(t|\bar{x}) p(\bar{x}) p(\xi)
\]

over expanded dataset.

Now, assume that  \( \int d\xi \xi^2 p(\xi) = E(\xi^2) = 0 \),

\( E(\xi^2) = \int d\xi \xi^2 p(\xi) = \lambda \) \( \xi \) small variance.

s.t. we only consider "small" transformations of \( \bar{x} \).

-1-
Then
\[ z(x, t) = z(x, 0) + \frac{\partial}{\partial t} \frac{\partial z}{\partial t} \bigg|_{t=0} + \frac{\beta^2}{2} \frac{\partial^2}{\partial x^2} z(x, t) \bigg|_{t=0} + O(\beta^3). \]

Next,
\[ y(\tilde{z}(x, t)) = y\left( x + \frac{\beta}{2} t + \frac{\beta^2}{2} t^2 \right) = \]
\[ y(\tilde{x}) + \frac{\beta}{2} t \frac{\partial y(\tilde{x})}{\partial \tilde{x}} + \frac{\beta^2}{2} \frac{t^2}{2} \frac{\partial y(\tilde{x})}{\partial \tilde{x}} + \]
\[ + \frac{\beta^2}{2} t^2 \sum_{ij} \frac{\partial^2 y(\tilde{x})}{\partial \tilde{x}_i \partial \tilde{x}_j} \text{ sums over } i, j \text{ implied.} \]

Then
\[ E = \frac{1}{2} \int \int \int d\tilde{x} dt d\tilde{\theta} \ p(t \tilde{x}) p(\tilde{x}) p(\tilde{\theta}) \times \]
\[ \times \left[ y(\tilde{x}) + \frac{\beta}{2} t \frac{\partial y(\tilde{x})}{\partial \tilde{x}} + \frac{\beta^2}{2} \frac{t^2}{2} \frac{\partial y(\tilde{x})}{\partial \tilde{x}} + \frac{\beta^2}{2} t^2 \sum_{ij} \frac{\partial^2 y(\tilde{x})}{\partial \tilde{x}_i \partial \tilde{x}_j} \right]^2 \]
\[ - \left[ y(\tilde{x}) - t \right]^2 = \]
\[ \int d\tilde{x} p(\tilde{x}) = 1 \]
\[ \implies \frac{1}{2} \int \int \int d\tilde{x} dt \left[ y(\tilde{x}) - t \right]^2 p(t \tilde{x}) p(\tilde{x}) \]
\[ \implies E \]
\( E(\xi) \int_0^T d\xi dt \ldots + E(\xi^2) \int_0^T d\xi dt \ p(t|\xi) \ p(\xi) \times \)
\[
\left( \frac{\partial y_i}{\partial x_i} \right) \left( \frac{\partial y_j}{\partial x_j} \right) + \left( \frac{\partial y_i}{\partial x_i} \right) \left( \frac{\partial y_j}{\partial x_j} \right)
\]
\[
\left( y_i(\xi) - t \right) \left[ \frac{\partial y_i}{\partial x_i} + \frac{\partial y_j}{\partial x_j} \right] + \frac{\partial y_i}{\partial x_i} \frac{\partial y_j}{\partial x_j}
\]

So, \( \tilde{E} = E + 2\xi \), where
\[
\xi = \frac{1}{2} \int d\xi \ p(\xi) \left[ \left( y(\xi) - E[t|\xi] \right) \right] + \int dt \ p(t|\xi) \ t
\]
\[
+ \left( \frac{\partial y_i}{\partial x_i} \right) \left( \frac{\partial y_j}{\partial x_j} \right)
\]
\[
\int dt \ p(t|\xi) = 1
\]

Now, note that \( y(\xi) = E[t|\xi] \)

minimizes \( E \) and "almost" minimizes \( \tilde{E} : y(\xi) = E[t|\xi] + \Theta(\xi) \)
in fact

Thus, the 1st term in \( \mathcal{N} \) is \( \Theta(\xi) \)

while the 2nd is \( \Theta(\xi) \), so that
\[
\mathcal{N} \approx \frac{1}{2} \int d\xi \ p(\xi) \ \left( \frac{\partial y_i}{\partial x_i} \right)^2 \leq \text{same as before!}
\]

1D Jacobian

-3-
Finally, in a special case
\[ \mathbf{x} \rightarrow \mathbf{x} + \xi \]
we obtain:

\[ \sum_{i} = \mathbf{x} + \xi \Rightarrow \frac{\partial s_k}{\partial \xi_i} = \delta_{ik} \]

Then
\[ y'(\mathbf{x}) = y(\mathbf{x}) + \xi_i \nabla_i y(\mathbf{x}) + \frac{1}{2} \xi_i \xi_j \nabla_i \nabla_j y(\mathbf{x}) \]

\[ \frac{\partial}{\partial \mathbf{x}_i} \]

Then
\[ E = \frac{1}{2} \iint \int d\mathbf{x}' dt d\mathbf{\xi}' p(t \mathbf{x}) p(\mathbf{x}) p(\mathbf{\xi}) \times \]
\[ \left[ y(\mathbf{x}) + \xi_i \nabla_i y(\mathbf{x}) + \frac{1}{2} \xi_i \xi_j \nabla_i \nabla_j y(\mathbf{x}) - t \right]^2 = \]
\[ = E + E(\xi_i) \frac{1}{2} \int d\mathbf{x}' dt \ldots + \sigma \]
\[ \int d\xi_i \xi_i p(\xi_i) \]

\[ \sigma \frac{1}{2} E(\xi_i, \xi_j) \iint \int d\mathbf{x}' dt d\mathbf{\xi}' p(t \mathbf{x}) p(\mathbf{x}) \times \]
\[ \left[ (y(\mathbf{x}) - t) \nabla_i y(\mathbf{x}) + \nabla_i y(\mathbf{x}) \nabla_j y(\mathbf{x}) \right] = \]
\[ = E + \kappa \mathcal{N} \]

\[ -y - \]
\[ E(\xi, \eta) = \begin{cases} \int d^2 \xi^i d^2 \eta^j \delta^2_{ij} \delta p(\xi) \rho(\eta) = 0, & i \neq j \\ \int d^2 \xi^i \delta^2_{ij} \delta p(\xi) = \mathcal{N}, & i = j \end{cases} \]

Here, \( \mathcal{N} = \frac{1}{2} \int d\bar{x} p(\bar{x}) \left[ (y(\bar{x}) - \mathbb{E}[y(\bar{x})]) \nabla^2 y(\bar{x}) + \Theta(\bar{x}) \right] \), discard \( \sum_i \delta_{i}^2 \)

\[
\begin{align*}
\left( \nabla y(\bar{x}) \right) \nabla y(\bar{x})
\end{align*}
\]

\[ = \frac{1}{2} \int d\bar{x} p(\bar{x}) \left\| \nabla y(\bar{x}) \right\|^2 .
\]

\text{\textbf{Tikhonov regularization}}
**Convolutional networks**

**Idea:** Avoid manual feature extraction by extracting higher-and-higher level invariant features from raw pixel data. Use the fact that nearby pixels are more strongly correlated than distant ones.

- Array of feature maps; each node in a given feature map has the same weights + bias $\Rightarrow$ acts as a kernel transform: $h(\sum W_{kj}x_j)$ (or convolution)

- These weights are called a filter bank

- Idea: a given feature map detects the presence of a given feature anywhere within a map $\Rightarrow$ weights must be shared since the feature to be detected is the same.
Each pooling node combines data from several nodes in a given feature map (or several feature maps), to reduce dimension of the representation and decrease sensitivity to small shifts 

Implementation: 1. $\tilde{c}(\tilde{g}_2 + \tilde{g}_1)$ [colder]
   - average of all inputs
2. max $\{2, 3\}$ max-pooling

Finally, 2-3 stages of feature extraction and pooling are stacked, with the number of feature maps going up in a given layer as features become higher-level (but lower-dimensional).

The final two layers are a convolutional layer which is fully connected to an output layer, typically with soft-max activation functions for $K > 2$ classification.
Modern implementation:

1. Use ReLU(·) as h(·) in convolutional layers, where

\[ \text{ReLU}(z) = \max(z, 0), \text{ rather than } \text{rectified linear unit} \]

Typically learns much faster with ReLU(·).

2. Use stochastic gradient descent for training \( \Rightarrow \) steepest descent informed by a few datapoints at a time rather than all of them.

3. Use backpropagation to compute the gradients.

4. In deep NNS, pretrain intermediate convolutional layers, afterwards, using Boltzmann machines (refine pre-trained weights by backpropagation).