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

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

$$E = \frac{1}{2} \iint d\tilde{x} dt \ (y(\tilde{x}) - t)^2 p(t|\tilde{x}) p(\tilde{x})$$

1D output (single output node) for simplicity

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

Then

$$E = \frac{1}{2} \iiint d\tilde{x} d\xi dt \ i \i \ (y(\tilde{s}(\tilde{x}, \xi)) - t)^2 p(t|\tilde{x}) p(\tilde{x}) p(\xi)$$

over expanded dataset

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

$$E(\xi^2) = \int d\xi \ \xi^2 p(\xi) = \lambda \leq \text{small, variance}$$

s.t. we only consider "small" transformations of $\tilde{x}$.
Then
\[ \mathcal{S}(\bar{x}, \xi) = \mathcal{S}(\bar{x}, 0) + \frac{\xi}{\zeta} \frac{\partial}{\partial \zeta} \mathcal{S}(\bar{x}, \xi) \bigg|_{\zeta=0} + \theta(\xi^3) + \frac{\kappa^2}{2} \frac{\partial^2}{\partial \xi^2} \mathcal{S}(\bar{x}, \xi) \bigg|_{\xi=0} \]

Next,
\[ \mathcal{S}(\bar{x}, \xi) = \mathcal{S}(\bar{x} + \frac{\xi}{\zeta} + \frac{\kappa^2}{2} \xi^2) = \]
\[ = \mathcal{S}(\bar{x}) + \frac{\kappa^2}{2} \xi \frac{\partial \mathcal{S}(\bar{x})}{\partial x_i} + \frac{\kappa^2}{2} \xi^2 \frac{\partial^2 \mathcal{S}(\bar{x})}{\partial x_i \partial x_j} \]
\[ + \frac{\kappa^2}{2} \xi \xi \frac{\partial^2 \mathcal{S}(\bar{x})}{\partial x_i \partial x_j} \text{ sums over } i, j \text{ implied} \]

Then
\[ \bar{E} = \frac{1}{2} \iint \int d\bar{x} dt d\xi p(\bar{t} \bar{x}) p(\bar{x}) p(\xi) \times \]
\[ \times \left[ \mathcal{S}(\bar{x}) + \frac{\kappa^2}{2} \xi \frac{\partial \mathcal{S}(\bar{x})}{\partial x_i} + \frac{\kappa^2}{2} \xi^2 \frac{\partial^2 \mathcal{S}(\bar{x})}{\partial x_i \partial x_j} - t \right]^2 \]
\[ = \frac{1}{2} \iint \int d\bar{x} dt \left[ \mathcal{S}(\bar{x}) - t \right]^2 p(\bar{t} \bar{x}) p(\bar{x}) + \]
\[ \bar{E} \]
\[ \begin{align*}
&+ \frac{1}{2} \int \int d\dot{\mathbf{x}} \, dt \quad \ldots \quad + \frac{1}{2} \int \int d\dot{\mathbf{x}} \, dt \, p(t|\mathbf{x}) \, p(\mathbf{x}) \times \\
&\quad \int_0^\infty \left[ (y(\mathbf{x}) - t) \left[ \tau_i \frac{\partial y}{\partial x_i} + \tau_j \frac{\partial y}{\partial x_j} \right] \right] + \\
&\quad + \tau_i \frac{\partial y}{\partial x_i} \tau_j \frac{\partial y}{\partial x_j} \\
\end{align*} \]

So, \( \tilde{E} = E + 2N \), where

\[ N = \frac{1}{2} \int d\mathbf{x} \, p(\mathbf{x}) \left[ (y(\mathbf{x}) - \bar{E}[t|\mathbf{x}]) \left[ \ldots \right] + \right. \]

\[ \int dt \, p(t|\mathbf{x}) \, t \\
\] \[ \int dt \, p(t|\mathbf{x}) = 1 \]

Now, note that \( y(\mathbf{x}) = \bar{E}[t|\mathbf{x}] \)

minimizes \( E \) and "almost" minimizes \( \tilde{E} \):

\( E: \ y(\mathbf{x}) = \bar{E}[t|\mathbf{x}] + O(\%) \) minimizes \( \tilde{E} \)

Thus, the 1st term in \( N \) is \( O(\%) \)

while the 2nd is \( O(\%) \), so that

\[ N \approx \frac{1}{2} \int d\mathbf{x} \, p(\mathbf{x}) \left( \tau_i \frac{\partial y}{\partial x_i} \right)^2 \text{ same as before!} \]

\[ \text{ID Jacobian} \]
Finally, in a special case \( x \rightarrow x + \delta \), we obtain:

\[
5(x, \delta) = x + \delta \quad \Rightarrow \quad \frac{\partial S_k}{\partial \delta_i} = \delta_{ik} \quad \text{not really needed}
\]

Then \( y_1(\delta) = y(x) + \delta_i \nabla_i y(x) + \frac{1}{2} \delta_i \delta_j \nabla_i \nabla_j y(x) \)

Then \( \bar{E} = \frac{1}{2} \iint d\bar{x}d\bar{t}d\bar{\delta} \quad p(t|x)p(x)p(\bar{x}) \times \)

\[
\times \left[ y_1(x) + \delta_i \nabla_i y(x) + \frac{1}{2} \delta_i \delta_j \nabla_i \nabla_j y(x) \right]^2 = \]

\[
= E + E(\delta_i) \frac{1}{2} \iiint d\bar{x}d\bar{t} \quad \text{to}
\]

\[
\int d\delta_i \delta_i \quad p(\delta_i)
\]

\[+ \frac{1}{2} E(\delta_i \delta_j) \iiint d\bar{x}d\bar{t} \quad p(t|x)p(x) \times \]

\[
\times \left[ (y(x)-t) \nabla_i y(x) + \nabla_i y(x) \nabla_j y(x) \right] =
\]

\[
= E + \sum_\mathcal{M}
\]

\[
\text{---}
\]
$$E(x_i; x_j) = \begin{cases} \int \delta x_i \delta x_j p(x_i) p(x_j) = 0, \quad i \neq j \\ \int \delta x_i \delta x_i^2 p(x_i) = \lambda, \quad i = j \end{cases}$$

Here, \( \lambda = \frac{1}{2} \int dx p(x) \left[ \left( \nabla^2 y(x) - E[ti(x)] \right) \nabla^2 y(x) + \theta(x) \right], \text{ discard } \sum_i \Delta_i^2 \]

$$+ \nabla_i y(x) \nabla_i y(x) \right) = \| \nabla y \|^2 \]

$$= \frac{1}{2} \int dx p(x) \| \nabla y(x) \|^2 .$$

Tikhonov regularization
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 => acts as a kernel
transform: h(Σ W_ij 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 => 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 contiguously (or several feature maps), to reduce dimension of the representation and decrease sensitivity to small shifts of distortions.

Implementation:

1. $\bar{c}(\bar{c}(2) + \bar{c}_0)$ (coldest)
   
   \[
   \text{average of all inputs}
   \]

2. $\max \{ z_j \}$ 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 $\text{ReLU}(\cdot)$ as $h(\cdot)$ in convolutional layers, where $\text{ReLU}(z) = \max(z, 0)$, rather than rectified linear unit $\ell(z)$ or $\tanh(z)$. Typically learns much faster with $\text{ReLU}(\cdot)$.

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 using Boltzmann machines, afterwards, refine pre-trained weights by backpropagation.
Non-linear activation functions

"Classical" activation functions are prone to saturation on the tails, where the gradients become small or vanish completely.

Perceptron

\[ \theta(z) \]

Sigmoid

\[ \sigma(z) = \frac{1}{1 + e^{-z}} \]

Tanh

\[ \tanh(z) \]
Gradient saturation is widely believed to reduce convergence and/or predictive power. Therefore, other activation functions have been proposed: (adapted)

**ReLU**

\[
\text{ReLU}(x) = \max(0, x)
\]

**Leaky ReLU**

\[
\text{Leaky ReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0.1x, & x < 0 \end{cases}
\]

**ELU**

\[
\text{ELU}(x) = \begin{cases} x, & x \geq 0 \\ e^x - 1, & x < 0 \end{cases}
\]