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

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

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

1D output (single output node) for simplicity

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

Then

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

over expanded dataset

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

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

s.t. we only consider "small" transformations of \( x \).
Then
\[ \psi(x, \xi) = \psi(x, 0) + \frac{\hbar}{2} \frac{\partial^2}{\partial \xi^2} \psi(x, \xi) \bigg|_{\xi = 0} + \Theta(\hbar^3) \]
\[ + \frac{\hbar^2}{2} \frac{\partial^2}{\partial x^2} \psi(x, \xi) \bigg|_{\xi = 0} \]

Next,
\[ \tilde{y}(\tilde{\psi}(x, \xi)) = \tilde{y}(x + \hbar \tilde{\tau} + \frac{\hbar^2 \tilde{\tau}^2}{2}) = \]
\[ \tilde{y}(\tilde{\tau}) + \hbar \tilde{\tau} \frac{\partial \tilde{y}(\tilde{\tau})}{\partial \tilde{\tau}} + \frac{\hbar^2}{2} \tilde{\tau} \frac{\partial^2 \tilde{y}(\tilde{\tau})}{\partial \tilde{\tau}^2} + \]
\[ + \frac{\hbar^2}{2} \tilde{\tau} \frac{\partial^2 \tilde{y}(\tilde{\tau})}{\partial \tilde{\tau} \partial \tilde{\tau}} \sim \text{sums over } i, j \text{ implied} \]

Then
\[ \bar{E} = \frac{1}{2} \int \int \int dx dt d\xi \quad p(t|x) p(x) p(\xi) \times \]
\[ \times \left[ \tilde{y}(\tilde{\tau}) + \hbar \tilde{\tau} \frac{\partial \tilde{y}(\tilde{\tau})}{\partial \tilde{\tau}} + \frac{\hbar^2}{2} \tilde{\tau} \frac{\partial^2 \tilde{y}(\tilde{\tau})}{\partial \tilde{\tau}^2} + \frac{\hbar^2}{2} \tilde{\tau} \frac{\partial^2 \tilde{y}(\tilde{\tau})}{\partial \tilde{\tau} \partial \tilde{\tau}} \right]^2 \]
\[ = \frac{1}{2} \int \int \int d\xi p(\xi) = 1 \]
\[ = \frac{1}{2} \int \int \int d\xi \left[ \tilde{y}(\tilde{\tau}) - t \right]^2 p(t|x) p(x) \]
\[ \bar{E} \]
\[ E(x) \frac{d}{dx} \int_{-\infty}^{\infty} \cdots + E(x^2) \frac{1}{2} \int_{-\infty}^{\infty} dx^2 \int_{0}^{\infty} \cdots \]

\[ \cdot \left[ (y(x) - t) \left( \sum_{i} \frac{\partial y}{\partial x_i} + \sum_{i,j} \frac{\partial y}{\partial x_i} \frac{\partial y}{\partial x_j} \right) \right] + \]

\[ + \sum_{i} \frac{\partial y}{\partial x_i} \left( \sum_{j} \frac{\partial y}{\partial x_j} \right) \]

So, \[ \tilde{E} = E + \lambda \mathcal{N} \], where

\[ \mathcal{N} = \frac{1}{2} \int_{-\infty}^{\infty} dx \int_{0}^{\infty} dt \left[ (y(x) - E(t|x)) \left( \sum_{i} \frac{\partial y}{\partial x_i} \right) + \right. \]

\[ \left. \int_{0}^{\infty} dt \int_{0}^{\infty} dt \left[ \sum_{i} \frac{\partial y}{\partial x_i} \right] \right] \]

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

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

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

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

\[ \mathcal{N} \approx \frac{1}{2} \int_{-\infty}^{\infty} dx p(x) \left( \sum_{i} \frac{\partial y}{\partial x_i} \right)^2 \]

\[ \approx \text{same as before!} \]

10 Jacobian
Finally, in a special case \( x \rightarrow x + \xi \), we obtain:

\[
\mathcal{L}(x, \xi) = x + \xi \quad \Rightarrow \quad \frac{\partial S}{\partial \xi_i} = \delta_{ik}, \quad \text{not really needed}
\]

Then

\[
y(x) = y(x) + \xi_i \nabla_i y(x) + \frac{1}{2} \xi_i \xi_j \nabla_i \nabla_j y(x) \quad \text{not needed}
\]

Then

\[
E = \frac{1}{2} \int \int \int dx^i dt d\xi_i \quad \left( p(t\tilde{x}) p(\tilde{x}) p(\tilde{\xi}) \right) \times
\]

\[
\left[ y(x) + \xi_i \nabla_i y(x) + \frac{1}{2} \xi_i \xi_j \nabla_i \nabla_j y(x) \right]^2 =
\]

\[
= E + E(\xi_i) \frac{1}{2} \int \int dx^i dt \quad \text{... +}
\]

\[
\int d\xi_i \xi_i \cdot p(\xi_i)
\]

\[
\frac{1}{2} E(\xi_i, \xi_j) \int \int dx^i dt \quad p(t\tilde{x}) p(\tilde{x}) \times
\]

\[
\left[ (y(x) - t) \nabla_i y(x) \quad \text{not needed} + \nabla_i y(x) \nabla_j y(x) \right] =
\]

\[
= E + \chi \mathcal{L}
\]
\[ E(x_i, x_j) = \begin{cases} \int d^dx_i d^dx_j \mathcal{H}_{ij} p(x_i) p(x_j) = 0, & i \neq j \\ \int d^dx_i \mathcal{H}_{ii}^2 p(x_i) = \lambda, & i = j \end{cases} \]

Here, \( \lambda = \frac{1}{2} \int d^dx p(x) \left[ (y(x) - E[t|x]) \nabla^2 y(x) + \frac{\eta(x)}{\Theta(|\eta|)} \right], \) discard \( \sum_i \nabla_i^2 \)

\[ + \nabla_i y(x) \nabla_i y(x) \right] = \frac{1}{2} \int d^dx p(x) \| \nabla y(x) \|^2 \]

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 \( \Rightarrow \) 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 & distortions.

Implementation:
1. $\bar{c}(\bar{z} + \bar{w}_0)$ [Coldder]
   - average of all inputs
2. $\max \{ z \}$ max pooling

---

Finally, 2-3 stages of feature extraction & 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 softmax activation functions for $K \geq 2$ 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 $\sigma(z)$ 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 afterwards, using Boltzmann machines (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:

\[ \text{ReLU} \]
\[ z \rightarrow \max(0, z) \]

\[ \text{Leaky ReLU} \]
\[ \begin{cases} z, & z \geq 0 \\ 0.1z, & z < 0 \end{cases} \]

\[ \text{ELU} \]
\[ \begin{cases} z, & z \geq 0 \\ e^z - 1, & z < 0 \end{cases} \]