

Commonly-used Tricks

F. Noé¹

Deep Learning Classes, FU Berlin 2018

Normalization versus Autoencoder error

Purpose: Move data close to $\mathbf{x} = 0$ and where network nonlinearities are most effective.

- Given Autoencoder E, D with mean square error:

$$L(\mathbf{X}; \theta) = \frac{1}{N} \sum_t \|\mathbf{x}_t - \hat{\mathbf{x}}_t\|_2^2 = \frac{1}{N} \sum_t \|\mathbf{x}_t - D(E(\mathbf{x}_t, \theta_E), \theta_D)\|_2^2$$

- Find the empirical mean $\bar{\mathbf{x}}$ and standard deviations $\mathbf{S} = \text{diag}(\sigma_1, \dots, \sigma_n)$ and normalize:

$$\mathbf{x}^s = \mathbf{S}^{-1}(\mathbf{x} - \bar{\mathbf{x}}).$$

- We now solve the scaled autoencoding problem:

$$L^s(\mathbf{X}; \theta) = \frac{1}{N} \sum_t \|\mathbf{x}_t^s - \hat{\mathbf{x}}_t^s\|_2^2 = \frac{1}{N} \sum_t \|\mathbf{x}_t^s - D(E(\mathbf{x}_t^s, \theta_E^s), \theta_D^s)\|_2^2$$

Both the loss value and the network parameters are different.

- In order to compute the original loss, we need to rescale it:

$$\begin{aligned} L(\mathbf{X}; \theta) &= \frac{1}{N} \sum_t \|\mathbf{S}(\mathbf{x}_t^s + \bar{\mathbf{x}}) - \mathbf{S}(\hat{\mathbf{x}}_t^s + \bar{\mathbf{x}})\|_2^2 \\ &= \frac{1}{N} \sum_t \|\mathbf{S}(\mathbf{x}_t^s - \hat{\mathbf{x}}_t^s)\|_2^2 = \frac{1}{N} \sum_t \|\mathbf{x}_t^s - \hat{\mathbf{x}}_t^s\|_{\sigma}^2 \end{aligned}$$

This can be seen as a new norm with weighting factors σ_i .

Normalization versus Autoencoder error

Purpose: Move data close to $\mathbf{x} = 0$ and where network nonlinearities are most effective.

- Given Autoencoder E, D with mean square error:

$$L(\mathbf{X}; \theta) = \frac{1}{N} \sum_t \|\mathbf{x}_t - \hat{\mathbf{x}}_t\|_2^2 = \frac{1}{N} \sum_t \|\mathbf{x}_t - D(E(\mathbf{x}_t, \theta_E), \theta_D)\|_2^2$$

- Find the empirical mean $\bar{\mathbf{x}}$ and standard deviations $\mathbf{S} = \text{diag}(\sigma_1, \dots, \sigma_n)$ and normalize:

$$\mathbf{x}^s = \mathbf{S}^{-1}(\mathbf{x} - \bar{\mathbf{x}}).$$

- We now solve the scaled autoencoding problem:

$$L^s(\mathbf{X}; \theta) = \frac{1}{N} \sum_t \|\mathbf{x}_t^s - \hat{\mathbf{x}}_t^s\|_2^2 = \frac{1}{N} \sum_t \|\mathbf{x}_t^s - D(E(\mathbf{x}_t^s, \theta_E^s), \theta_D^s)\|_2^2$$

Both the loss value and the network parameters are different.

- In order to compute the original loss, we need to rescale it:

$$\begin{aligned} L(\mathbf{X}; \theta) &= \frac{1}{N} \sum_t \|\mathbf{S}(\mathbf{x}_t^s + \bar{\mathbf{x}}) - \mathbf{S}(\hat{\mathbf{x}}_t^s + \bar{\mathbf{x}})\|_2^2 \\ &= \frac{1}{N} \sum_t \|\mathbf{S}(\mathbf{x}_t^s - \hat{\mathbf{x}}_t^s)\|_2^2 = \frac{1}{N} \sum_t \|\mathbf{x}_t^s - \hat{\mathbf{x}}_t^s\|_{\sigma}^2 \end{aligned}$$

This can be seen as a new norm with weighting factors σ_i .

Normalization versus Autoencoder error

Purpose: Move data close to $\mathbf{x} = 0$ and where network nonlinearities are most effective.

- Given Autoencoder E, D with mean square error:

$$L(\mathbf{X}; \theta) = \frac{1}{N} \sum_t \|\mathbf{x}_t - \hat{\mathbf{x}}_t\|_2^2 = \frac{1}{N} \sum_t \|\mathbf{x}_t - D(E(\mathbf{x}_t, \theta_E), \theta_D)\|_2^2$$

- Find the empirical mean $\bar{\mathbf{x}}$ and standard deviations $\mathbf{S} = \text{diag}(\sigma_1, \dots, \sigma_n)$ and normalize:

$$\mathbf{x}^s = \mathbf{S}^{-1}(\mathbf{x} - \bar{\mathbf{x}}).$$

- We now solve the scaled autoencoding problem:

$$L^s(\mathbf{X}; \theta) = \frac{1}{N} \sum_t \|\mathbf{x}_t^s - \hat{\mathbf{x}}_t^s\|_2^2 = \frac{1}{N} \sum_t \|\mathbf{x}_t^s - D(E(\mathbf{x}_t^s, \theta_E^s), \theta_D^s)\|_2^2$$

Both the loss value and the network parameters are different.

- In order to compute the original loss, we need to rescale it:

$$\begin{aligned} L(\mathbf{X}; \theta) &= \frac{1}{N} \sum_t \|\mathbf{S}(\mathbf{x}_t^s + \bar{\mathbf{x}}) - \mathbf{S}(\hat{\mathbf{x}}_t^s + \bar{\mathbf{x}})\|_2^2 \\ &= \frac{1}{N} \sum_t \|\mathbf{S}(\mathbf{x}_t^s - \hat{\mathbf{x}}_t^s)\|_2^2 = \frac{1}{N} \sum_t \|\mathbf{x}_t^s - \hat{\mathbf{x}}_t^s\|_{\sigma}^2 \end{aligned}$$

This can be seen as a new norm with weighting factors σ_i .

Normalization versus Autoencoder error

Purpose: Move data close to $\mathbf{x} = 0$ and where network nonlinearities are most effective.

- Given Autoencoder E, D with mean square error:

$$L(\mathbf{X}; \theta) = \frac{1}{N} \sum_t \|\mathbf{x}_t - \hat{\mathbf{x}}_t\|_2^2 = \frac{1}{N} \sum_t \|\mathbf{x}_t - D(E(\mathbf{x}_t, \theta_E), \theta_D)\|_2^2$$

- Find the empirical mean $\bar{\mathbf{x}}$ and standard deviations $\mathbf{S} = \text{diag}(\sigma_1, \dots, \sigma_n)$ and normalize:

$$\mathbf{x}^s = \mathbf{S}^{-1}(\mathbf{x} - \bar{\mathbf{x}}).$$

- We now solve the scaled autoencoding problem:

$$L^s(\mathbf{X}; \theta) = \frac{1}{N} \sum_t \|\mathbf{x}_t^s - \hat{\mathbf{x}}_t^s\|_2^2 = \frac{1}{N} \sum_t \|\mathbf{x}_t^s - D(E(\mathbf{x}_t^s, \theta_E^s), \theta_D^s)\|_2^2$$

Both the loss value and the network parameters are different.

- In order to compute the original loss, we need to rescale it:

$$\begin{aligned} L(\mathbf{X}; \theta) &= \frac{1}{N} \sum_t \|\mathbf{S}(\mathbf{x}_t^s + \bar{\mathbf{x}}) - \mathbf{S}(\hat{\mathbf{x}}_t^s + \bar{\mathbf{x}})\|_2^2 \\ &= \frac{1}{N} \sum_t \|\mathbf{S}(\mathbf{x}_t^s - \hat{\mathbf{x}}_t^s)\|_2^2 = \frac{1}{N} \sum_t \|\mathbf{x}_t^s - \hat{\mathbf{x}}_t^s\|_{\sigma}^2 \end{aligned}$$

This can be seen as a new norm with weighting factors σ_i .

Regularization

Purpose: To perform not only well on training data, but also on test data

- **Regularization algorithms** modification learning algorithm in order to reduce its generalization error but not its training error.
- **Most common idea:** regularizing estimators by reducing its variance on the price of increasing its bias.
- **Examples:**
 - **Penalizing** certain ranges of parameter values (e.g., L0, L1, L2 constraints).
 - **Data augmentaion:** Increasing the training set size by exploiting known invariances.
 - **Ensemble methods:** combine multiple hypotheses that explain the training data.

Regularization

Purpose: To perform not only well on training data, but also on test data

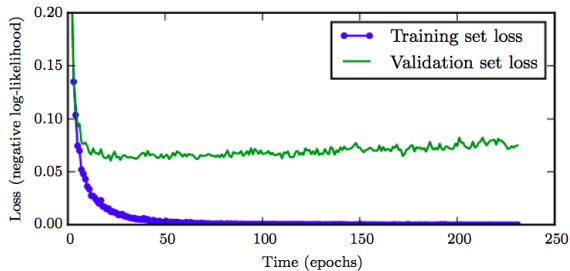
- **Regularization algorithms** modification learning algorithm in order to reduce its generalization error but not its training error.
- **Most common idea:** regularizing estimators by reducing its variance on the price of increasing its bias.
- **Examples:**
 - **Penalizing** certain ranges of parameter values (e.g., L0, L1, L2 constraints).
 - **Data augmentaion:** Increasing the training set size by exploiting known invariances.
 - **Ensemble methods:** combine multiple hypotheses that explain the training data.

Regularization

Purpose: To perform not only well on training data, but also on test data

- **Regularization algorithms** modification learning algorithm in order to reduce its generalization error but not its training error.
- **Most common idea:** regularizing estimators by reducing its variance on the price of increasing its bias.
- **Examples:**
 - **Penalizing** certain ranges of parameter values (e.g., L0, L1, L2 constraints).
 - **Data augmentaion:** Increasing the training set size by exploiting known invariances.
 - **Ensemble methods:** combine multiple hypotheses that explain the training data.

Overfitting during training



Early Stopping

Purpose: To avoid overfitting as a function of training time

- Returning the model with the **minimal validation error** during training procedure, instead of minimal training error.
- **Hyperparameter selection** algorithm where the number of training steps is a hyperparameter.
- Effectively **restricts the optimization procedure** to a small volume of parameter space in the neighborhood of the initial parameter value (Bishop 1995, Sjöberg and Ljung 1995).
- Can be interpreted as a **form of weight decay** (Goodfellow, Courville and Bengio, 2016).
- Can be used to **reduce training time** by exiting when validation error stops decreasing.
- Frequently used, as it is very easy, inexpensive, and compatible with any learning problem.

Early Stopping

Purpose: To avoid overfitting as a function of training time

- Returning the model with the **minimal validation error** during training procedure, instead of minimal training error.
- **Hyperparameter selection** algorithm where the number of training steps is a hyperparameter.
- Effectively **restricts the optimization procedure** to a small volume of parameter space in the neighborhood of the initial parameter value (Bishop 1995, Sjöberg and Ljung 1995).
- Can be interpreted as a **form of weight decay** (Goodfellow, Courville and Bengio, 2016).
- Can be used to **reduce training time** by exiting when validation error stops decreasing.
- Frequently used, as it is very easy, inexpensive, and compatible with any learning problem.

Early Stopping

Purpose: To avoid overfitting as a function of training time

- Returning the model with the **minimal validation error** during training procedure, instead of minimal training error.
- **Hyperparameter selection** algorithm where the number of training steps is a hyperparameter.
- Effectively **restricts the optimization procedure** to a small volume of parameter space in the neighborhood of the initial parameter value (Bishop 1995, Sjöberg and Ljung 1995).
- Can be interpreted as a **form of weight decay** (Goodfellow, Courville and Bengio, 2016).
- Can be used to **reduce training time** by exiting when validation error stops decreasing.
- Frequently used, as it is very easy, inexpensive, and compatible with any learning problem.

Early Stopping

Purpose: To avoid overfitting as a function of training time

- Returning the model with the **minimal validation error** during training procedure, instead of minimal training error.
- **Hyperparameter selection** algorithm where the number of training steps is a hyperparameter.
- Effectively **restricts the optimization procedure** to a small volume of parameter space in the neighborhood of the initial parameter value (Bishop 1995, Sjöberg and Ljung 1995).
- Can be interpreted as a **form of weight decay** (Goodfellow, Courville and Bengio, 2016).
- Can be used to **reduce training time** by exiting when validation error stops decreasing.
- Frequently used, as it is very easy, inexpensive, and compatible with any learning problem.

Early Stopping

Purpose: To avoid overfitting as a function of training time

- Returning the model with the **minimal validation error** during training procedure, instead of minimal training error.
- **Hyperparameter selection** algorithm where the number of training steps is a hyperparameter.
- Effectively **restricts the optimization procedure** to a small volume of parameter space in the neighborhood of the initial parameter value (Bishop 1995, Sjöberg and Ljung 1995).
- Can be interpreted as a **form of weight decay** (Goodfellow, Courville and Bengio, 2016).
- Can be used to **reduce training time** by exiting when validation error stops decreasing.
- Frequently used, as it is very easy, inexpensive, and compatible with any learning problem.

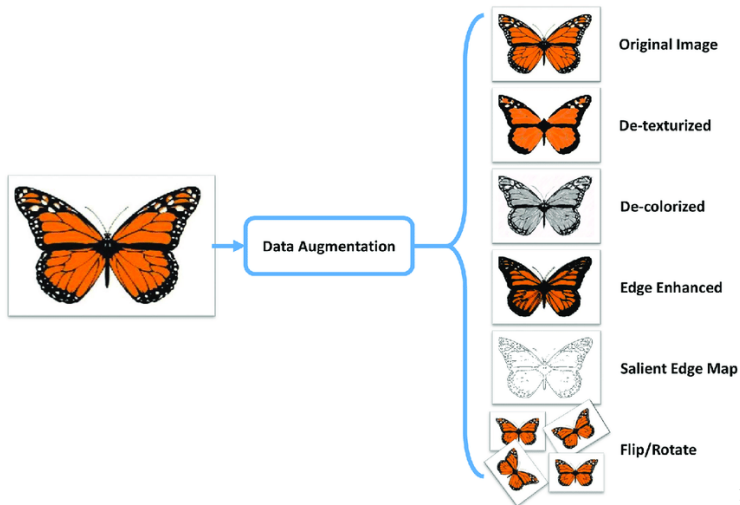
Early Stopping

Purpose: To avoid overfitting as a function of training time

- Returning the model with the **minimal validation error** during training procedure, instead of minimal training error.
- **Hyperparameter selection** algorithm where the number of training steps is a hyperparameter.
- Effectively **restricts the optimization procedure** to a small volume of parameter space in the neighborhood of the initial parameter value (Bishop 1995, Sjöberg and Ljung 1995).
- Can be interpreted as a **form of weight decay** (Goodfellow, Courville and Bengio, 2016).
- Can be used to **reduce training time** by exiting when validation error stops decreasing.
- Frequently used, as it is very easy, inexpensive, and compatible with any learning problem.

Data Augmentation

Purpose: Improve generalization by increasing training data



1

Data Augmentation

Purpose: Improve generalization by increasing training data

- **Artificially increase training data** by adding “fake” samples which account for known invariances in the learning problem.
- **Example:** regularize image classification problem by adding translated and rotated copies of available training images to the training data.
- **Careful: Invariances** are defined with respect to the **learning problem**. Example: a “9” rotated by 180° is no longer class 9 but class 6.
- **Noise:** Adding small amounts of noise should not affect classification, but Neural networks are not to be very robust to noise (Tang and Eliasmith, 2010).

Data Augmentation

Purpose: Improve generalization by increasing training data

- **Artificially increase training data** by adding “fake” samples which account for known invariances in the learning problem.
- **Example:** regularize image classification problem by adding translated and rotated copies of available training images to the training data.
- **Careful: Invariances** are defined with respect to the **learning problem**. Example: a “9” rotated by 180° is no longer class 9 but class 6.
- **Noise:** Adding small amounts of noise should not affect classification, but Neural networks are not to be very robust to noise (Tang and Eliasmith, 2010).

Data Augmentation

Purpose: Improve generalization by increasing training data

- **Artificially increase training data** by adding “fake” samples which account for known invariances in the learning problem.
- **Example:** regularize image classification problem by adding translated and rotated copies of available training images to the training data.
- **Careful: Invariances** are defined with respect to the **learning problem**. Example: a “9” rotated by 180° is no longer class 9 but class 6.
- **Noise:** Adding small amounts of noise should not affect classification, but Neural networks are not to be very robust to noise (Tang and Eliasmith, 2010).

Data Augmentation

Purpose: Improve generalization by increasing training data

- **Artificially increase training data** by adding “fake” samples which account for known invariances in the learning problem.
- **Example:** regularize image classification problem by adding translated and rotated copies of available training images to the training data.
- **Careful: Invariances** are defined with respect to the **learning problem**. Example: a “9” rotated by 180° is no longer class 9 but class 6.
- **Noise:** Adding small amounts of noise should not affect classification, but Neural networks are not to be very robust to noise (Tang and Eliasmith, 2010).

Parameter Norm Penalties

Purpose: To limiting model capacity by penalizing certain parameter values

- Add penalty term with hyperparameter α :

$$\tilde{L}(\mathbf{X}, \mathbf{Y}; \theta) = L(\mathbf{X}, \mathbf{Y}; \theta) + \alpha \Omega(\theta)$$

- Typically constraint only weights w_{ij} , not biases b_i , as those contribute less estimator variance and constraining them can introduce much estimator bias.
- L^2 regularization: Ridge or Tikhonov regularization in linear methods, weight decay in neural networks.
- L^1 regularization: Sparsity constraint, generates zero weights.

Parameter Norm Penalties

Purpose: To limiting model capacity by penalizing certain parameter values

- Add penalty term with hyperparameter α :

$$\tilde{L}(\mathbf{X}, \mathbf{Y}; \theta) = L(\mathbf{X}, \mathbf{Y}; \theta) + \alpha \Omega(\theta)$$

- Typically constraint only weights w_{ij} , not biases b_i , as those contribute less estimator variance and constraining them can introduce much estimator bias.
- L^2 regularization: Ridge or Tikhonov regularization in linear methods, weight decay in neural networks.
- L^1 regularization: Sparsity constraint, generates zero weights.

Parameter Norm Penalties

Purpose: To limiting model capacity by penalizing certain parameter values

- Add penalty term with hyperparameter α :

$$\tilde{L}(\mathbf{X}, \mathbf{Y}; \theta) = L(\mathbf{X}, \mathbf{Y}; \theta) + \alpha \Omega(\theta)$$

- Typically constraint only weights w_{ij} , not biases b_i , as those contribute less estimator variance and constraining them can introduce much estimator bias.
- L^2 regularization: Ridge or Tikhonov regularization in linear methods, weight decay in neural networks.
- L^1 regularization: Sparsity constraint, generates zero weights.

Parameter Norm Penalties

Purpose: To limiting model capacity by penalizing certain parameter values

- Add penalty term with hyperparameter α :

$$\tilde{L}(\mathbf{X}, \mathbf{Y}; \theta) = L(\mathbf{X}, \mathbf{Y}; \theta) + \alpha \Omega(\theta)$$

- Typically constraint only weights w_{ij} , not biases b_i , as those contribute less estimator variance and constraining them can introduce much estimator bias.
- L^2 regularization: Ridge or Tikhonov regularization in linear methods, weight decay in neural networks.
- L^1 regularization: Sparsity constraint, generates zero weights.

Parameter Tying and Sharing

Purpose: To limiting model capacity by restricting parameter range or reducing number of parameters

- **Parameter tying:** Two sets of parameters are assumed to be similar, we add a constraint of the form:

$$\left\| \mathbf{w}^{(A)} - \mathbf{w}^{(B)} \right\|_2^2$$

- **Parameter sharing:** Two sets of parameters are equal,
 - Example: convolutional kernels in ConvNets are applied translationally invariant at all image positions.
 - Parameter sharing can dramatically reduce the number of parameters.
 - Natural way to encode previous knowledge about invariances.

Parameter Tying and Sharing

Purpose: To limiting model capacity by restricting parameter range or reducing number of parameters

- **Parameter tying:** Two sets of parameters are assumed to be similar, we add a constraint of the form:

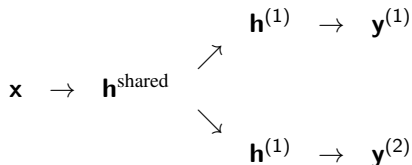
$$\left\| \mathbf{w}^{(A)} - \mathbf{w}^{(B)} \right\|_2^2$$

- **Parameter sharing:** Two sets of parameters are equal,
 - Example: convolutional kernels in ConvNets are applied translationally invariant at all image positions.
 - Parameter sharing can dramatically reduce the number of parameters.
 - Natural way to encode previous knowledge about invariances.

Multi-Task Learning

Purpose: Improve generalization by increasing training data

- Improve generalization by **sharing representation** for different learning tasks with the same input.

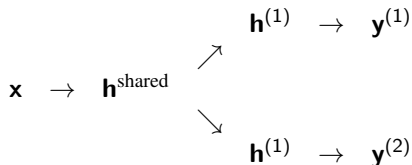


- When sharing part of the representation is useful to predict both tasks, this effectively increases the available training data and puts more pressure on the model to generalize well.

Multi-Task Learning

Purpose: Improve generalization by increasing training data

- Improve generalization by **sharing representation** for different learning tasks with the same input.



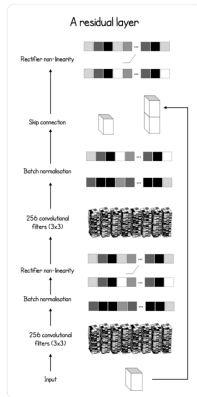
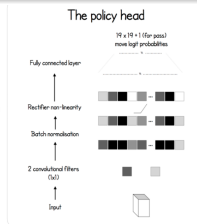
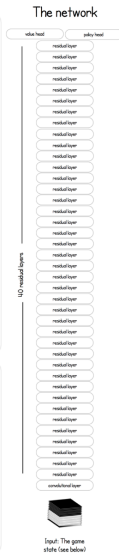
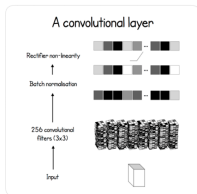
- When sharing part of the representation is useful to predict both tasks, this effectively increases the available training data and puts more pressure on the model to generalize well.

Multi-Task Learning

Example: AlphaGo Zero

The network learns 'tabula rasa' (from a blank slate)

At no point is the network trained using human knowledge or expert moves



Ensemble methods

Purpose: Train and average ensemble of learners to avoid susceptibility to individual test set errors

- **Model averaging:** Train several different models separately, then have all of the models vote on the output for test examples.
- **Bagging** (“bootstrap aggregating”, Breiman 1994): construct k different datasets, sampled from the original dataset *with* replacement.
- **Example:** k regression models, each model makes an error $\epsilon_i \sim \mathcal{N}(0, \Sigma)$ with variances $\sigma_{ii}^2 = v$ and covariances $\sigma_{ij}^2 = c$. Expected squared error of the ensemble predictor:

$$\mathbb{E} \left[\left(\frac{1}{k} \sum_i \epsilon_i \right)^2 \right] = \frac{1}{k^2} \mathbb{E} \left[\sum_i \left(\epsilon_i^2 + \sum_{j \neq i} \epsilon_i \epsilon_j \right) \right] = \frac{1}{k} v + \frac{k-1}{k} c.$$

- Worst case: errors are perfectly correlated, $c = v$, MSE stays v .
 - Best case: errors perfectly uncorrelated, $c = 0$, MSE is v/k and reduces with ensemble size.
 - Ensemble is at least as good as any of its members, but often significantly better.
- **Disadvantage:** training multiple models is very expensive.

Ensemble methods

Purpose: Train and average ensemble of learners to avoid susceptibility to individual test set errors

- **Model averaging:** Train several different models separately, then have all of the models vote on the output for test examples.
- **Bagging** (“bootstrap aggregating”, Breiman 1994): construct k different datasets, sampled from the original dataset *with* replacement.
- **Example:** k regression models, each model makes an error $\epsilon_i \sim \mathcal{N}(0, \Sigma)$ with variances $\sigma_{ii}^2 = v$ and covariances $\sigma_{ij}^2 = c$.
Expected squared error of the ensemble predictor:

$$\mathbb{E} \left[\left(\frac{1}{k} \sum_i \epsilon_i \right)^2 \right] = \frac{1}{k^2} \mathbb{E} \left[\sum_i \left(\epsilon_i^2 + \sum_{j \neq i} \epsilon_i \epsilon_j \right) \right] = \frac{1}{k} v + \frac{k-1}{k} c.$$

- Worst case: errors are perfectly correlated, $c = v$, MSE stays v .
 - Best case: errors perfectly uncorrelated, $c = 0$, MSE is v/k and reduces with ensemble size.
 - Ensemble is at least as good as any of its members, but often significantly better.
- **Disadvantage:** training multiple models is very expensive.

Ensemble methods

Purpose: Train and average ensemble of learners to avoid susceptibility to individual test set errors

- **Model averaging:** Train several different models separately, then have all of the models vote on the output for test examples.
- **Bagging** (“bootstrap aggregating”, Breiman 1994): construct k different datasets, sampled from the original dataset *with* replacement.
- **Example:** k regression models, each model makes an error $\varepsilon_i \sim \mathcal{N}(0, \Sigma)$ with variances $\sigma_{ii}^2 = v$ and covariances $\sigma_{ij}^2 = c$. Expected squared error of the ensemble predictor:

$$\mathbb{E} \left[\left(\frac{1}{k} \sum_i \varepsilon_i \right)^2 \right] = \frac{1}{k^2} \mathbb{E} \left[\sum_i \left(\varepsilon_i^2 + \sum_{j \neq i} \varepsilon_i \varepsilon_j \right) \right] = \frac{1}{k} v + \frac{k-1}{k} c.$$

- Worst case: errors are perfectly correlated, $c = v$, MSE stays v .
 - Best case: errors perfectly uncorrelated, $c = 0$, MSE is v/k and reduces with ensemble size.
 - Ensemble is at least as good as any of its members, but often significantly better.
- **Disadvantage:** training multiple models is very expensive.

Ensemble methods

Purpose: Train and average ensemble of learners to avoid susceptibility to individual test set errors

- **Model averaging:** Train several different models separately, then have all of the models vote on the output for test examples.
- **Bagging** (“bootstrap aggregating”, Breiman 1994): construct k different datasets, sampled from the original dataset *with* replacement.
- **Example:** k regression models, each model makes an error $\varepsilon_i \sim \mathcal{N}(0, \Sigma)$ with variances $\sigma_{ii}^2 = v$ and covariances $\sigma_{ij}^2 = c$. Expected squared error of the ensemble predictor:

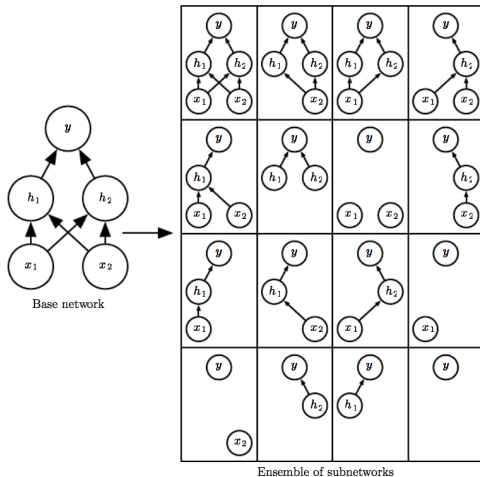
$$\mathbb{E} \left[\left(\frac{1}{k} \sum_i \varepsilon_i \right)^2 \right] = \frac{1}{k^2} \mathbb{E} \left[\sum_i \left(\varepsilon_i^2 + \sum_{j \neq i} \varepsilon_i \varepsilon_j \right) \right] = \frac{1}{k} v + \frac{k-1}{k} c.$$

- Worst case: errors are perfectly correlated, $c = v$, MSE stays v .
 - Best case: errors perfectly uncorrelated, $c = 0$, MSE is v/k and reduces with ensemble size.
 - Ensemble is at least as good as any of its members, but often significantly better.
- **Disadvantage:** training multiple models is very expensive.

Dropout (Srivastava et al., 2014)

Purpose: Computationally cheap approximation to model averaging over different network architectures

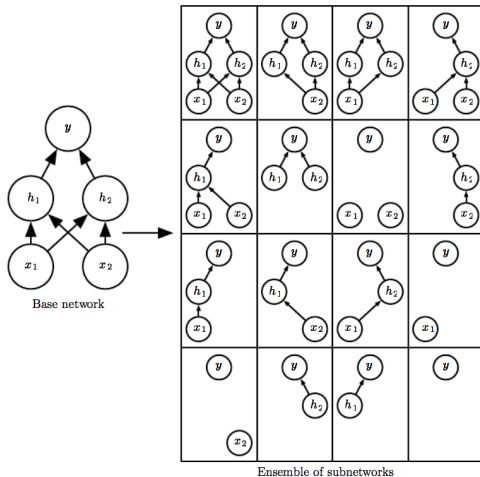
- Approximates Bagging with exponentially many neural networks.
- Trains ensemble of all sub-networks formed by removing input or hidden units.



Dropout (Srivastava et al., 2014)

Purpose: Computationally cheap approximation to model averaging over different network architectures

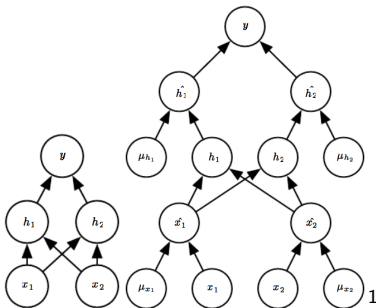
- Approximates Bagging with exponentially many neural networks.
- Trains ensemble of all sub-networks formed by removing input or hidden units.



Dropout (Srivastava et al., 2014)

Purpose: Computationally cheap approximation to model averaging over different network architectures

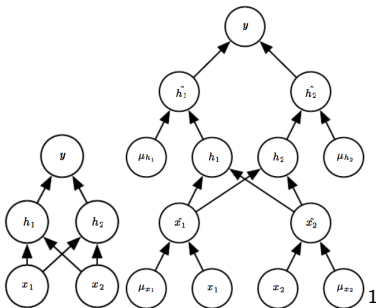
- **Mask vector** μ , $\mu_i \in \{0,1\}$. Defines which neurons are active.
- **Training:** For each minibatch, sample μ with probability p_i^{keep} to keep neuron i . Usual forward and backward propagation.
- **Typically values:** input: $p_i^{\text{keep}} = 0.8$, hidden: $p_i^{\text{keep}} = 0.5$.
- **Advantages of Dropout:**
 - **Cheap:** training and inference little more expensive than without dropout (generation and application of μ vector, scaling weights).
 - **General:** works with nearly any NN model and SGD.



Dropout (Srivastava et al., 2014)

Purpose: Computationally cheap approximation to model averaging over different network architectures

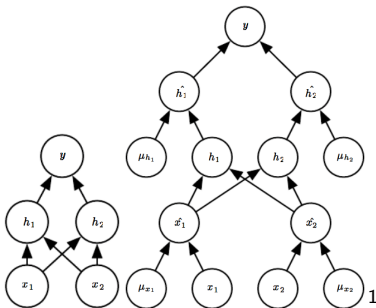
- **Mask vector** μ , $\mu_i \in \{0,1\}$. Defines which neurons are active.
- **Training:** For each minibatch, sample μ with probability p_i^{keep} to keep neuron i . Usual forward and backward propagation.
- **Typically values:** input: $p_i^{\text{keep}} = 0.8$, hidden: $p_i^{\text{keep}} = 0.5$.
- **Advantages of Dropout:**
 - **Cheap:** training and inference little more expensive than without dropout (generation and application of μ vector, scaling weights).
 - **General:** works with nearly any NN model and SGD.



Dropout (Srivastava et al., 2014)

Purpose: Computationally cheap approximation to model averaging over different network architectures

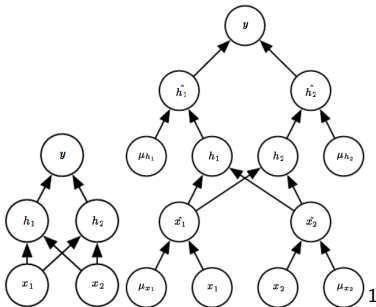
- **Mask vector** μ , $\mu_i \in \{0,1\}$. Defines which neurons are active.
- **Training:** For each minibatch, sample μ with probability p_i^{keep} to keep neuron i . Usual forward and backward propagation.
- **Typically values:** input: $p_i^{\text{keep}} = 0.8$, hidden: $p_i^{\text{keep}} = 0.5$.
- **Advantages of Dropout:**
 - **Cheap:** training and inference little more expensive than without dropout (generation and application of μ vector, scaling weights).
 - **General:** works with nearly any NN model and SGD.



Dropout (Srivastava et al., 2014)

Purpose: Computationally cheap approximation to model averaging over different network architectures

- **Mask vector** μ , $\mu_i \in \{0, 1\}$. Defines which neurons are active.
- **Training:** For each minibatch, sample μ with probability p_i^{keep} to keep neuron i . Usual forward and backward propagation.
- **Typically values:** input: $p_i^{\text{keep}} = 0.8$, hidden: $p_i^{\text{keep}} = 0.5$.
- **Advantages of Dropout:**
 - **Cheap:** training and inference little more expensive than without dropout (generation and application of μ vector, scaling weights).
 - **General:** works with nearly any NN model and SGD.



Dropout as a Bagging method

- Suppose model outputs probability distribution $p(y | \mathbf{x})$.
 - **Bagging**: arithmetic mean over bootstrap samples:

$$\frac{1}{k} \sum_{i=1}^k p^{(i)}(y | \mathbf{x})$$

- **Dropout**: arithmetic mean over all masks

$$\sum_{\mu} p(\mu) p(y | \mathbf{x}, \mu)$$

Cannot be directly evaluated (exponentially many masks).

- **Weight scaling inference rule** (Hinton et al., 2012): approximate the geometric mean of $p(y | \mathbf{x}, \mu)$ by evaluating $p(y | \mathbf{x})$ in the model with all units, but with the weights going out of unit i multiplied by the probability of including unit i .
 - Motivation: capture the expected value of the output from that unit.
 - Accuracy of this algorithm is not theoretically understood, but works well in practice.
 - With typical choice $p_i^{\text{keep}} = 0.5$, weight scaling results in dividing weights by 2 after training.

Dropout as a Bagging method

- Suppose model outputs probability distribution $p(y | \mathbf{x})$.
 - **Bagging**: arithmetic mean over bootstrap samples:

$$\frac{1}{k} \sum_{i=1}^k p^{(i)}(y | \mathbf{x})$$

- **Dropout**: arithmetic mean over all masks

$$\sum_{\mu} p(\mu) p(y | \mathbf{x}, \mu)$$

Cannot be directly evaluated (exponentially many masks).

- **Weight scaling inference rule** (Hinton et al., 2012): approximate the geometric mean of $p(y | \mathbf{x}, \mu)$ by evaluating $p(y | \mathbf{x})$ in the model with all units, but with the weights going out of unit i multiplied by the probability of including unit i .
 - Motivation: capture the expected value of the output from that unit.
 - Accuracy of this algorithm is not theoretically understood, but works well in practice.
 - With typical choice $p_i^{\text{keep}} = 0.5$, weight scaling results in dividing weights by 2 after training.

Adversarial Attacks - Adversarial Training

Purpose: Improve robustness against random or intended perturbations

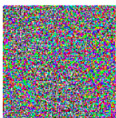
- Neural networks have reached human performance in several benchmark tasks. What about human-level understanding?
- **Adversarial attacks** (trying to fake the network):
 - **Intended perturbation:** Accuracy reduced to random with test examples with small $\|x_i^{\text{test}} - x_j^{\text{train}}\|$ but large $\|y_i^{\text{test}} - y_j^{\text{train}}\|$ for given i, j .
 - **Random perturbation:** Add Gaussian noise with a very small amplitude. Picture indistinguishable for humans, but breaks neural network performance (Szegedy et al. 2014).



x

$y = \text{"panda"}$
w/ 57.7%
confidence

+ .007 ×



=



$x +$

$\text{sign}(\nabla_x J(\theta, x, y))$

$\epsilon \text{sign}(\nabla_x J(\theta, x, y))$

"nematode"
w/ 8.2%
confidence

"gibbon"
w/ 99.3 %
confidence

1

- **Adversarial training:** include adversarial attacks in the training data to force predictions to be locally constant near training data.

Adversarial Attacks - Adversarial Training

Purpose: Improve robustness against random or intended perturbations

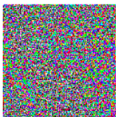
- Neural networks have reached human performance in several benchmark tasks. What about human-level understanding?
- **Adversarial attacks** (trying to fake the network):
 - **Intended perturbation:** Accuracy reduced to random with test examples with small $\|\mathbf{x}_i^{\text{test}} - \mathbf{x}_j^{\text{train}}\|$ but large $\|y_i^{\text{test}} - y_j^{\text{train}}\|$ for given i, j .
 - **Random perturbation:** Add Gaussian noise with a very small amplitude. Picture indistinguishable for humans, but breaks neural network performance (Szegedy et al. 2014).



\mathbf{x}

$y = \text{"panda"}$
w/ 57.7%
confidence

+ .007 ×



=



$\mathbf{x} +$
 $\epsilon \text{sign}(\nabla_{\mathbf{x}} J(\theta, \mathbf{x}, y))$
"gibbon"
w/ 99.3 %
confidence

1

- **Adversarial training:** include adversarial attacks in the training data to force predictions to be locally constant near training data.

Adversarial Attacks - Adversarial Training

Purpose: Improve robustness against random or intended perturbations

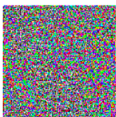
- Neural networks have reached human performance in several benchmark tasks. What about human-level understanding?
- **Adversarial attacks** (trying to fake the network):
 - **Intended perturbation**: Accuracy reduced to random with test examples with small $\|\mathbf{x}_i^{\text{test}} - \mathbf{x}_j^{\text{train}}\|$ but large $\|y_i^{\text{test}} - y_j^{\text{train}}\|$ for given i, j .
 - **Random perturbation**: Add Gaussian noise with a very small amplitude. Picture indistinguishable for humans, but breaks neural network performance (Szegedy et al. 2014).



\mathbf{x}

$y = \text{"panda"}$
w/ 57.7%
confidence

+ .007 ×



$\text{sign}(\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y))$

"nematode"
w/ 8.2%
confidence

=



$\mathbf{x} + \epsilon \text{sign}(\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y))$

"gibbon"
w/ 99.3 %
confidence

1

- **Adversarial training**: include adversarial attacks in the training data to force predictions to be locally constant near training data.

Adversarial Attacks - Adversarial Training

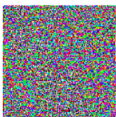
Purpose: Improve robustness against random or intended perturbations

- Neural networks have reached human performance in several benchmark tasks. What about human-level understanding?
- **Adversarial attacks** (trying to fake the network):
 - **Intended perturbation:** Accuracy reduced to random with test examples with small $\|\mathbf{x}_i^{\text{test}} - \mathbf{x}_j^{\text{train}}\|$ but large $\|y_i^{\text{test}} - y_j^{\text{train}}\|$ for given i, j .
 - **Random perturbation:** Add Gaussian noise with a very small amplitude. Picture indistinguishable for humans, but breaks neural network performance (Szegedy et al. 2014).



x
 $y = \text{"panda"}$
w/ 57.7%
confidence

+ .007 ×



$\text{sign}(\nabla_x J(\theta, x, y))$
"nematode"
w/ 8.2%
confidence

=



$x +$
 $\epsilon \text{sign}(\nabla_x J(\theta, x, y))$
"gibbon"
w/ 99.3 %
confidence

1

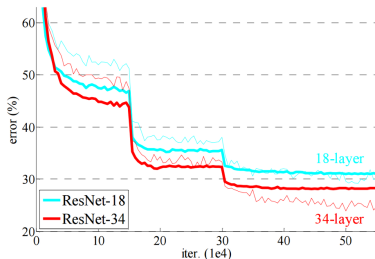
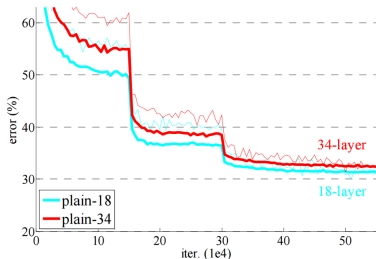
- **Adversarial training:** include adversarial attacks in the training data to force predictions to be locally constant near training data.

- **Transposed Convolutions:**
- **Pooling, Unpooling:**

Residual Networks

Purpose: Make deep networks trainable

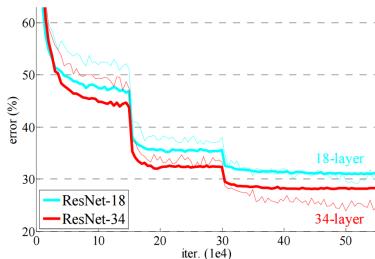
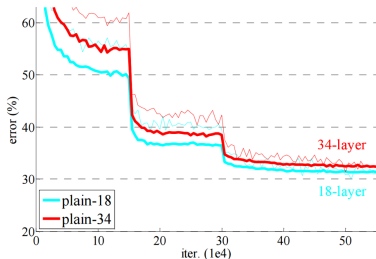
- With most network architectures, when adding layers (increasing depth), the training loss first reduces but then increases.
- Indicates training problem – adding layers make the network more expressive, so training loss should be non-increasing.
→ also affects the test loss.
- Residual Networks (He, Zhang, Ren, Sun, 2015) enable training of very deep networks.



Residual Networks

Purpose: Make deep networks trainable

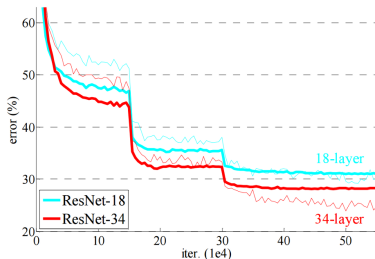
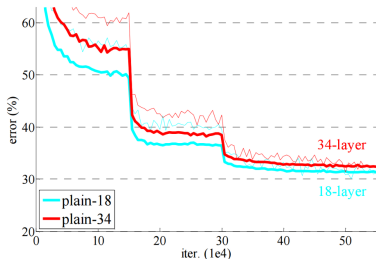
- With most network architectures, when adding layers (increasing depth), the training loss first reduces but then increases.
- Indicates training problem – adding layers make the network more expressive, so training loss should be non-increasing.
→ also affects the test loss.
- Residual Networks (He, Zhang, Ren, Sun, 2015) enable training of very deep networks.



Residual Networks

Purpose: Make deep networks trainable

- With most network architectures, when adding layers (increasing depth), the training loss first reduces but then increases.
- Indicates training problem – adding layers make the network more expressive, so training loss should be non-increasing.
→ also affects the test loss.
- Residual Networks (He, Zhang, Ren, Sun, 2015) enable training of very deep networks.

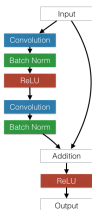


Residual Networks

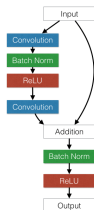
Purpose: Make deep networks trainable

- As in LSTMs, ResNets introduces a short-cut path that can carry gradients deep.
- ResNets are state-of-the-art in many problems (CIFAR, ImageNet, AlphaGo Zero).

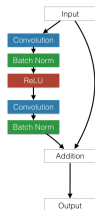
Reference paper



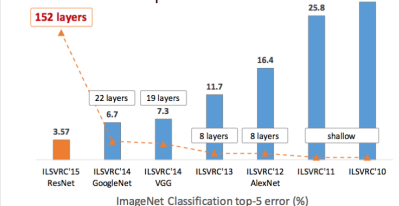
Batch Norm after add



No ReLU



Revolution of Depth

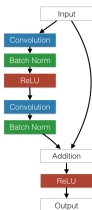


Residual Networks

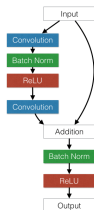
Purpose: Make deep networks trainable

- As in LSTMs, ResNets introduces a short-cut path that can carry gradients deep.
- ResNets are state-of-the-art in many problems (CIFAR, ImageNet, AlphaGo Zero).

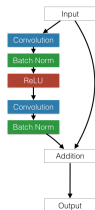
Reference paper



Batch Norm after add



No ReLU



Revolution of Depth

